Computer-Verified Foundations of Metaphysics and an Ontology of Natural Numbers in Isabelle/HOL

Dissertation
zur Erlangung des Grades eines
Doktors der Naturwissenschaften

am Fachbereich Mathematik und Informatik
der Freien Universität Berlin

vorgelegt von

Daniel Kirchner

Berlin, December 2021

Betreuer:	**Prof. Dr. habil Christoph Benzmüller**
Zweitgutachter:	**Dr. Edward N. Zalta**
Drittgutachter:	**Prof. DDr. Hannes Leitgeb**
Datum der Disputation:	**16. Mai 2022**

Selbstständigkeitserklärung

Ich erkläre gegenüber der Freien Universität Berlin, dass ich die vorliegende Dissertation selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe. Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht. Diese Dissertation wurde in gleicher oder ähnlicher Form noch in keinem früheren Promotionsverfahren eingereicht.

Mit einer Prüfung meiner Arbeit durch ein Plagiatsprüfungsprogramm erkläre ich mich einverstanden.

Abstract

We utilize and extend the method of *shallow semantic embeddings* (SSEs) in classical higher-order logic (HOL) to construct a custom theorem proving environment for *abstract objects theory* (AOT) on the basis of Isabelle/HOL.

SSEs are a means for universal logical reasoning by translating a target logic to HOL using a representation of its semantics. AOT is a foundational metaphysical theory, developed by Edward Zalta, that explains the objects presupposed by the sciences as *abstract objects* that reify property patterns. In particular, AOT aspires to provide a philosphically grounded basis for the construction and analysis of the objects of mathematics.

We can support this claim by verifying Uri Nodelman's and Edward Zalta's reconstruction of Frege's theorem: we can confirm that the Dedekind-Peano postulates for natural numbers are consistently derivable in AOT using Frege's method. Furthermore, we can suggest and discuss generalizations and variants of the construction and can thereby provide theoretical insights into, and contribute to the philosophical justification of, the construction.

In the process, we can demonstrate that our method allows for a nearly transparent exchange of results between traditional pen-and-paper-based reasoning and the computerized implementation, which in turn can retain the automation mechanisms available for Isabelle/HOL.

During our work, we could significantly contribute to the evolution of our target theory itself, while simultaneously solving the technical challenge of using an SSE to implement a theory that is based on logical foundations that significantly differ from the meta-logic HOL.

In general, our results demonstrate the fruitfulness of the practice of Computational Metaphysics, i.e. the application of computational methods to metaphysical questions and theories.

Contents

1. Introduction

1.1. Motivation

The analysis of foundational formal systems using automated theorem provers is as old as automated theorem provers themselves: Already in the middle of the last century, Bertrand Russell was quick to recognize the potential of computational methods, when confronted with the *Logic Theorist*,[1] commonly regarded as the first automated theorem prover, and its ability to prove 38 out of 52 theorems from chapter two of Whitehead and Russell's Principia Mathematica, including a proof more elegant than one of Whitehead and Russell's own (see [23]):

> I am delighted to know that Principia Mathematica can now be done by machinery. I wish Whitehead and I had known of this possibility before we both wasted ten years doing it by hand. I am quite willing to believe that everything in deductive logic can be done by a machine.[2]

However, building up a sound reasoning environment from scratch is a non-trivial task. Consequently, today there is only a limited number of trusted systems that can offer sophisticated interactive and automated reasoning tools like Coq [50], HOL-Light [24] or Isabelle/HOL [39]. Furthermore, most of these systems have at least parts of their logical foundation in common. For example, they are all based on some variation of functional type theory. This may lead to a bias in the computational analysis of foundational theories towards systems that use a similar logical foundation.

The following represents an attempt at overcoming this issue. We utilize the concept of a *shallow semantic embedding* (SSE) with abstraction layers to transfer the merits of the reasoning environment of Isabelle/HOL to a fundamentally different foundational system, namely to Abstract Object Theory (AOT).

While it is not a requirement for our proposed general method, we demonstrate that we can extend Isabelle/HOL by a customized reasoning infrastructure written in Isabelle/ML that allows for an almost entirely transparent transfer of reasoning in our target logic and abstracts the syntactic and inferential differences between Isabelle/HOL and AOT, while still internally using the verified core logic of Isabelle/HOL as semantic backend. This means we effectively construct a dedicated theorem proving environment for AOT that (1) is immediately guaranteed to be sound, (2) can be used to explore the safety of axiomatic extensions of the system and (3) allows for the reuse of the automation infrastructure available for Isabelle/HOL.

[1] A system developed by Allen Newell and Herbert Simon at Carnegie Mellon and programmed by J. C. Shaw using the vacuum tubes of the JOHNNIAC computer at the Institute for Advanced Study.

[2] Letter from Russell to Simon dated 2 November, 1956; preserved in [49], page 208.

While our method can potentially be applied to a multitude of logical systems, AOT is a particularly well-suited target. On the one hand, it aims to be a foundational metaphysical system that can serve as the basis for mathematics and thereby stands in the tradition of Russell and Whitehead's Principia Mathematica, while in fact extending its scope to e.g. linguistics and the sciences in general (see [58]). On the other hand, it is based on logical foundations that significantly differ from classical functional higher-order type-theory and were even argued to be incompatible (see [43]). Initial results of our research (see [29]) demonstrated how our method for formally analyzing models and semantics of such a system can be beneficial and vital for its soundness (see 3.6.2). During our continued work, we could contribute to the evolution of AOT and simultaneously arrived at a model structure and semantics that allows to faithfully reproduce its deductive system in Isabelle/HOL while retaining the existing infrastructure for automated reasoning.[3]

As a prime result, we can show that the construction of Natural Numbers and the derivation of the Dedekind-Peano postulates, including Mathematical Induction, described in Principia Logico-Metaphysica (PLM)[4] are verifiably sound. Furthermore, we can suggest the generalization of an additional axiom required for this construction, that we believe strengthens the argument that the construction does not require any inherently mathematical axioms.

1.2. Prior Work

Since the time of Russell and the Logic Theorist, there has been significant progress both in the development of automated theorem provers in general and in the application of computational methods to metaphysical questions and foundational logical theories in particular. Some of the more recent developments in this area are outlined in the following sections.

1.2.1. Prior Computational Analysis of Abstract Object Theory

The computational analysis of AOT was pioneered by Fitelson and Zalta in [17]. They used the first-order system Prover9 (see [34]) for their work and were able to verify the proofs of the theorems in AOT's analysis of situations and possible worlds in [65]. Furthermore, they discovered an error in a theorem about Platonic Forms in [46] that had been left as an exercise. Other work with Prover9 that does not target AOT includes the simplification of the reconstruction of Anselm's ontological argument (in [41], Oppenheimer and Zalta show that only one of the three premises they used in [42] is sufficient) or the reconstruction of theorems in Spinoza's *Ethics* in [26].

[3] Note, however, that our embedding currently only extends to the second-order fragment of AOT. We briefly discuss the challenges of representing full higher-order object theory in chapter 6.

[4] PLM is a continuously developed online monograph (see [62]) written by Edward Zalta, that contains the most recent canonical presentation of AOT. This thesis is written relative to the version dated October 13, 2021, archived in [63].

However, there are inherent limitations to the approach of analyzing higher-order theories like AOT with the help of first-order provers. While it is possible to reason about the first-order truth conditions of statements by introducing sort predicates and using a number of special techniques to translate the statements into the less-expressive language of multi-sorted first-order logic (a detailed account of such techniques is given in [1]), the complexity of the resulting representation increases for expressive, higher-order philosophical claims. In general, this approach may be sufficient for analyzing concrete isolated arguments, but it becomes infeasible to construct a natural representation of an entire expressive higher-order theory and its full deductive system (see also [30]).

1.2.2. Prior Work involving Shallow Semantic Embeddings

Independently, the emergence of sophisticated higher-order reasoning environments like Isabelle/HOL allows for a different approach, namely the analysis of arguments and theories directly in higher-order logic by constructing Shallow Semantic Embeddings (SSEs) (see [2]). In contrast to a *deep embedding* which defines the syntax of a target system using an inductive data structure and evaluates statements semantically by recursively traversing this data structure, a *shallow* semantic embedding instead provides a syntactic translation from the target logic to the meta-logic. This is done by reusing as much of the infrastructure of the meta-logic as possible, while *defining* the syntactic elements of the target logic that are not part of the meta-logic by means of a representation of their semantics. Since sets have a natural representation in higher-order logic, this approach works well for any logical system that has a semantics defined in terms of sets. The approach of shallow semantic embeddings is discussed in more detail in chapter 2.

For example, Benzmüller et al. provide an extensive analysis of quantified modal logic using SSEs by means of embedding modal operators based on their Kripke semantics [6, 3, 9]. This allowed for an analysis of Gödel's ontological argument in second-order S5 modal logic and weaker logics such as KB (see [8, 4]), followed by a range of studies of similar ontological arguments (see e.g. [20]).

Another more recent example of the application of SSEs is the LogiKEy framework for ethical reasoning, normative theories and deontic logics (see [5] and [10]). The goal of LogiKEy is to develop the means for the control and governance of intelligent autonomous systems. The framework is based on a set of SSEs of different deontic logics, combinations thereof, as well as ethico-legal domain theories in higher-order logic with an implementation in Isabelle/HOL.

The advantage of these studies using SSEs compared to the earlier use of first-order systems is that arguments can be represented in their native syntax and are thereby readable and maintainable, while the theorem proving environment is capable of automatically transforming statements into a suitable first-order representation on the fly to allow first-order theorem provers like E (see [48]) or SPASS (see [51]) to perform proof search much like e.g. Prover9 was able to do on a manually constructed first-order representation.

These studies were still mainly concerned with case studies of concrete arguments or with conservative extensions of higher-order logic like quantified higher-order modal logic.

1.2.3. Analysis of AOT with the SSE Approach

Initial results of our own research were reported in [29], in which we applied an extended version of the technique of SSEs to AOT. For AOT no extensive prior analysis of canonical models was available, in contrast to, for example, the extensive analysis of Kripke models for higher-order modal logic that served as theoretical basis for the previous work using SSEs mentioned above. While the so-called Aczel models of object theory (see [60]) provide an important building block for constructing models of AOT in HOL, no full set-theoretic model of object theory had been constructed. In [29] we extended the existing Aczel models to a richer model structure that was capable of approximating the validity of statements of the at the time most recent formulation of the second-order fragment of AOT in *Principia Logico-Metaphysica*.[5] Furthermore, we introduced the new concept of *abstraction layers*. An abstraction layer consists of a derivation of the axioms and deduction rules of a target system from a given semantics that is then considered as ground truth while "forgetting" the underlying semantic structure, i.e. the reasoning system is prevented from using the semantics for proofs, but is instead configured to solely rely on the derived axioms and deduction rules. Abstraction layers turned out to be a helpful means for reasoning within a target theory without the danger of deriving artifactual theorems (see 4.8), while simultaneously allowing to maintain a flexible semantic backend that can be used to explore axiomatic extensions and variations of the target theory.

A major initial result of this project, reported in [31], was the discovery of an oversight in an early version of PLM that allowed for the reintroduction of a previously known paradox into the system. While multiple quick fixes to restore the consistency of AOT were immediately available, in the aftermath of this result AOT was significantly reworked and improved. The result triggered an extensive debate about the foundations of AOT which culminated in the extension of the free logic of AOT to relations, while previously it was restricted to individual terms only. This evolution of AOT was accompanied by a continuous further development of its embedding in Isabelle/HOL. This mutually beneficial mode of work was described in [30] and resulted in a now stabilized and improved formulation of AOT and a matching embedding of its second-order fragment. The details of this process and its results are the main subject of this thesis.

[5] The respective version of PLM is archived in [64].

1.3. Contributions and Structure of the Thesis

In the following, we first provide a more detailed description of Shallow Semantic Embeddings (chapter 2) and a brief introduction to Abstract Object Theory (chapter 3). Based on that, chapter 4 describes the constructed embedding of the second-order fragment of AOT (as presented in PLM [63]) in Isabelle/HOL.

In the process we highlight the contributions of the embedding to AOT on the one hand and the techniques developed for its implementation on the other hand.

In chapter 5 we present our results on PLM's construction of natural numbers and discuss an extension of AOT with a more general comprehension principle for relations among abstract objects. We also discuss some interesting variations of the construction that may be adopted by PLM in the future.

Finally, in chapter 6 we briefly discuss the issue of applying our method to the full higher-order type-theoretic version of AOT.

Our primary goals are to show that:

- SSEs can not only be used for case studies and the analysis of isolated arguments, but also for implementing the axioms and full deductive system of entire logical theories.
- The above is even feasible for a challenging target like AOT, which itself has the ambition to be a foundational framework and is based on significantly different logical foundations compared to our meta-logic HOL.
- We can reproduce the full deductive system of AOT in readable and usable form while preserving Isabelle's automation mechanisms. Thereby, we can effectively construct a dedicated automated theorem proving environment for AOT.
- Using our method we could significantly contribute to our target theory.
- We can demonstrate the extent of our target theory and the practical feasibility of reproducing complex reasoning in it by reproducing and validating its analysis of natural numbers.
- In the process, we can provide valuable theoretical insights into, and analyze extensions and variations of, the construction of the natural numbers.

1.4. Verified Document Generation and Conventions

This thesis is generated using Isabelle's document preparation system (see [53]). In particular, all formal statements cited in the thesis are renderings of verified theorems in the embedding, unless specifically stated otherwise and marked with vertical bars at the page margins.[6]

The appendix contains a rendering of the raw theory files of the embedding including all proofs.[7] The implementation currently consists of around 25,000 lines of Isabelle proof text.[8] While Isabelle allows producing latex code for raw theories directly,[9] semantic information (e.g. color-coding of free vs. bound variables) is lost in the process, which reduces the readability. For that reason, we devised a custom theory presentation system similar to Isabelle's HTML theory presentation that uses PIDE markup information (see [52]) to provide a color-coded rendering of the theory files equipped with hyperlinks for cross-references.[10]

Whenever a theorem in the appendix refers to a specific item number in PLM, the corresponding item number can be found in parentheses at the right page margin. While we will sometimes refer to item numbers in PLM directly, we will usually refer to the implementation in the appendix by section and line number and rely on the statement in the appendix being annotated with the item number of the corresponding statement in PLM. In particular, the thesis is written relative to the version of PLM dated October 13, 2021 (see [63]).

While a certain degree of familiarity with the reasoning environment of Isabelle/HOL might be helpful, the fact that reasoning in Isabelle/HOL is designed to be natural and intelligible should allow following the constructions without extensive prior knowledge of Isabelle/HOL. An introduction to reasoning in Isabelle/HOL can be found in [38]. The implementation is written relative to the Isabelle2021-1 (December 2021) release of Isabelle.

[6] With the exception of chapter 6 which is not written relative to an embedding in Isabelle and omits the marking at the page margins.

[7] The corresponding theory files can also be found at [27].

[8] Around 20,000 lines are reasoning in the abstraction layer, i.e. reasoning in the logic of the target theory, while the remainder builds up the required model structure and semantics as well as the syntax representation of AOT.

[9] This mechanism is used for raw theory content that is inlined in the main thesis, but not for the appendix.

[10] Therefore, we recommend reading this thesis in digital form.

2. Shallow Semantic Embeddings

2.1. Embeddings of Domain-Specific Languages

In computer science, deep and shallow embeddings have been a traditional means to implement domain-specific languages by embedding them into general-purpose host languages (see for example [22]). A simple example is a language of *expressions* that can be either integer constants, resp. literals, or the addition of two other expressions. If we consider Isabelle/HOL as the host language in this process, the following would constitute a *deep* embedding of this language:

datatype *expression* = *Literal int* | *Addition expression expression*
primrec *eval* :: ‹*expression* ⇒ *int*› **where**
  ‹*eval* (*Literal x*) = *x*›
| ‹*eval* (*Addition x y*) = *eval x* + *eval y*›

The deep embedding consists of a (usually recursive) algebraic datatype that captures the syntactic elements of the language to be embedded. This representation of the syntax is then given a semantics by means of an evaluation function that traverses this algebraic datatype.[1] A shallow embedding on the other hand, represents the syntactic elements of a target language directly in their semantic domain. In our example, the semantic domain of expressions is the integers. On this domain, operations are then *defined* directly by means of their semantics:

type-synonym *expression* = *int*
definition *Literal* :: ‹*int* ⇒ *expression*› **where** ‹*Literal x* ≡ *x*›
definition *Addition* :: ‹*expression* ⇒ *expression* ⇒ *expression*› **where** ‹*Addition x y* ≡ *x* + *y*›

Note that in the shallow embedding, the domain of *expression*s is shared with the meta-language by directly representing expressions in the type to which they evaluate semantically in the deep embedding, namely *int* in the example.

There is a natural correspondence between the deep and shallow representations of this language. In particular it holds that *Deep.eval* (*Deep.Literal x*) = *Shallow.Literal x* and *Deep.eval* (*Deep.Addition x y*) = *Shallow.Addition* (*Deep.eval x*) (*Deep.eval y*). So semantic

[1] In the setting of logical theories this evaluation function would usually depend on interpretations and assignment functions. However, in our simple example this is not necessary, since the simple language of expressions neither involves constants nor variables (respectively since literals have trivial interpretations).

evaluation is implicit in the shallow embedding. On the other hand there are also differences between the two representations. For example, in the deep embedding adding x to y results in an expression that is different from the expression of adding y to x for distinct x and y, even though they are equivalent under evaluation:

$x \neq y \Longrightarrow \mathit{Deep.Addition}\ x\ y \neq \mathit{Deep.Addition}\ y\ x$
$\mathit{Deep.eval}\ (\mathit{Deep.Addition}\ x\ y) = \mathit{Deep.eval}\ (\mathit{Deep.Addition}\ y\ x)$

In contrast, commuted additions are identical in the shallow embedding:

$\mathit{Shallow.Addition}\ x\ y = \mathit{Shallow.Addition}\ y\ x$

In fact, the shallow embedding can be thought of as a *quotient* of the deep embedding under semantic evaluation.

While there are several advantages and disadvantages of using shallow vs. deep embeddings for Domain-Specific languages, we forgo a detailed discussion of them here and focus on shallow embeddings of logical theories in the next sections.

2.2. SSEs as Universal Reasoning Tools

In [2], Benzmüller develops the idea of using *Shallow Semantic Embeddings* (SSEs) in classical higher-order logics (HOL) as a means for universal reasoning.

He notes that while already Leibniz envisioned a *characteristica universalis*, a most universal formal language in which all knowledge (and all arguments) about the world and the sciences can be encoded, in practice, today we rather find a *rich and heterogeneous zoo of different logical systems*.

A solution to this dilemma is the use of a universal *meta*-logic, in which a multitude of logic formalisms can be *embedded*.

While there are multiple such unifying approaches, for example using algebraic logic or category theory as framework, Benzmüller defends the use of SSEs in HOL for pragmatic reasons:

- For HOL there are sophisticated automation tools readily available that have been developed for several decades like e.g. Isabelle/HOL.
- Since HOL itself is very expressive, an embedding into HOL can often be achieved using simple techniques and can result in an elegant and concise representation of the target logic.
- Using a *shallow* embedding approach, the technical overhead of the translation can be kept minimal, which enables the reuse of the automation infrastructure available for the meta-logic.

While we already mentioned a variety of results that were achieved using this general method (see section 1.2.2), in the following we will demonstrate the process of building such an SSE at a simple example.

2.3. SSE of Quantified Higher-Order Modal Logic

An example of a non-classical logic that is used prominently in philosophical arguments is Quantified Higher-Order Modal Logic in various different axiomatizations. While there have been extensive studies of modal logics using SSEs in Isabelle/HOL (see section 1.2.2), we restrict ourselves to the discussion of a simple embedding of S5 modal logic to further illustrate the general concept of SSEs.

A natural semantic basis for SSEs of any modal logic is its Kripke-semantics (see [32]). In general, a Kripke frame consists of a set of possible worlds and a binary relation on these worlds called *accessibility relation*. For S5 there are two versions of semantics, one in which the accessibility relation is an equivalence relation and one in which there is no accessibility relation at all (see [18]). For our purpose the simpler model suffices.[2]

For possible worlds we can introduce a primitive type *w* in Isabelle/HOL.[3]

typedecl *w*

A Kripke model further involves a relation between possible worlds and modal formulas that is usually read as a formula *being satisfied at* a possible world. So the semantic domain of propositions is the boolean-valued functions acting on (or, equivalently, the sets of) possible worlds. In an SSE we use the semantic domains as types for the object-level terms themselves,[4] so we can introduce a type o of propositions as synonym of the type of functions mapping possible worlds (of type *w*) to booleans (type *bool*). This way the proposition can, as a function, be applied to a possible world, yielding *True*, if the proposition is satisfied at that world or *False* otherwise.[5]

type-synonym o = ‹*w* ⇒ *bool*›

A proposition is *valid* in case it is satisfied in all worlds (or, alternatively, in a designated actual world).[6]

definition *valid* :: ‹o ⇒ *bool*› (‹⊨ -› *100*) **where**
‹⊨ *p* ≡ ∀ *w* . *p w*›

Now the classical logical operators can be defined as follows (note the bold print for the defined operators versus the non-bold print of the corresponding operators of the meta-logic):

[2] We will later argue that this is also a natural choice for the particular modal logic of Abstract Object Theory due to its additional actuality operator and rigid definite descriptions, see section 4.7.4.

[3] A set-theoretic model of HOL would represent this type with a non-empty set of objects that may serve as denotation for objects of type *w*.

[4] Note that it is also possible to model restrictions on the evaluation domains explicitly, as recently demonstrated in [7].

[5] Note that this choice of a representation of propositions commits us to a modal logic, in which necessary equivalence implies identity. We will later discuss how we can construct a hyperintensional logic instead.

[6] The specification in parentheses after the type of the defined constant, o ⇒ *bool*, is *mixfix notation* used to introduce the symbol ⊨ as syntax for the introduced constant *valid* with the specified precedence. The means to introduce custom syntax in Isabelle/HOL are discussed in more detail in section 2.7.

definition *not* :: ‹o ⇒ o› (‹¬-› [*140*] *140*) **where**
‹$\neg p \equiv \lambda\ w\ .\ \neg p\ w$›
definition *imp* :: ‹o ⇒ o ⇒ o› (**infixl** ‹→› *125*) **where**
‹$p \rightarrow q \equiv \lambda\ w\ .\ p\ w \longrightarrow q\ w$›
definition *conj* :: ‹o ⇒ o ⇒ o› (**infixl** ‹∧› *135*) **where**
‹$p \wedge q \equiv \lambda\ w\ .\ p\ w \wedge q\ w$›
definition *disj* :: ‹o ⇒ o ⇒ o› (**infixl** ‹∨› *130*) **where**
‹$p \vee q \equiv \lambda\ w\ .\ p\ w \vee q\ w$›

The additional modal operators, i.e. the box operator for *necessity* and the diamond operator for *possibility*, can be further defined as:

definition *box* :: ‹o ⇒ o› (‹□-› [*150*] *150*) **where**
‹$\Box p \equiv \lambda\ w\ .\ \forall\ v\ .\ p\ v$›
definition *dia* :: ‹o ⇒ o› (‹◇-› [*150*] *150*) **where**
‹$\Diamond p \equiv \lambda\ w\ .\ \exists\ v\ .\ p\ v$›

Now Isabelle can show automatically that the S5 axioms are valid:

lemma *K*: ‹$\models \Box(p \rightarrow q) \rightarrow (\Box p \rightarrow \Box q)$›
by (*auto simp*: *box-def imp-def valid-def*)
lemma *T*: ‹$\models \Box p \rightarrow p$›
by (*auto simp*: *box-def imp-def valid-def*)
lemma *5*: ‹$\models \Diamond p \rightarrow \Box\Diamond p$›
by (*auto simp*: *box-def dia-def imp-def valid-def*)

The proofs of the axioms are automatically found by **sledgehammer**, Isabelle/HOL's main tool for automation.[7]

So far we have constructed an embedding of propositional S5 modal logic using what is commonly known as *Standard Translation* of modal logic (see [11]). However it is straightforward to enrich this embedding with quantification.[8]

definition *forall* :: ‹('a ⇒ o) ⇒ o› (**binder** ‹∀ › *110*) **where**
‹$\forall\ x\ .\ \varphi\ x \equiv \lambda w\ .\ \forall\ x\ .\ \varphi\ x\ w$›
definition *exists* :: ‹('a ⇒ o) ⇒ o› (**binder** ‹∃ › *110*) **where**
‹$\exists\ x\ .\ \varphi\ x \equiv \lambda w\ .\ \exists\ x\ .\ \varphi\ x\ w$›

Note that we didn't introduce any particular type for individuals, but stated polymorphic definitions relative to a type variable *'a*. This way the same quantifier can be used for propositions themselves, any desired type for individuals or even properties of any order.[9]

As an example of theorems involving quantifiers and modal logic, we derive the Barcan formulas. **sledgehammer** can again automatically provide proofs.

lemma ‹$\models (\forall x\ .\ \Box\varphi\ x) \rightarrow \Box(\forall x\ .\ \varphi\ x)$›
by (*auto simp*: *box-def forall-def imp-def valid-def*)

[7] **sledgehammer** is discussed in more detail in the following section.
[8] See also the work by Benzmüller et al. cited in section 1.2.2.
[9] Note that this construction implies a shared domains for objects across possible worlds. An additional meta-logical predicate for *logical existence in a possible world* can be added to model varying domains.

lemma $\langle \models \Diamond(\exists\, x\ .\ \varphi\ x) \rightarrow (\exists\, x\ .\ \Diamond\varphi\ x)\rangle$
by (*auto simp*: *dia-def exists-def imp-def valid-def*)
lemma $\langle \models \Box(\forall\, x\ .\ \varphi\ x) \rightarrow\ (\forall\, x\ .\ \Box\varphi\ x)\rangle$
by (*auto simp*: *box-def forall-def imp-def valid-def*)
lemma $\langle \models (\exists\, x\ .\ \Diamond\varphi\ x) \rightarrow \Diamond(\exists\, x\ .\ \varphi\ x)\rangle$
by (*auto simp*: *dia-def exists-def imp-def valid-def*)

However, note that the automatic proofs again unfold the semantic definitions. We have shown that the Barcan formulas are valid in the constructed embedding, but from the proofs we cannot tell which axioms are required for proving them.[10]

Depending on the application, it can be enough to be able to tell if a theorem is semantically valid or if a statement semantically follows from a set of assumptions. However, for the purpose of implementing a full logical theory including its own deductive system, semantic validity is not the primary concern, but rather derivability from the formal system.[11]

Fortunately, it is possible to restrict Isabelle's automated reasoning tools like **sledgehammer**, s.t. they may not unfold semantic definitions. If this is done at larger scale and in a reliable manner for the purpose of analyzing derivability in a given deductive system, we say that we introduce *abstraction layers* to the SSE.

2.4. SSEs with Abstraction Layers

The concept of enriching traditional SSEs with abstraction layers was first introduced in [29]. The goal is to be able to use the automated reasoning tools provided by a system like Isabelle/HOL not merely to analyze semantic validity of statements in the embedded theory, but to reliably determine the derivability of a statement from the deductive system of the theory itself.

An abstraction layer is simply constructed by proving that the axioms and deduction rules of a target logic are semantically valid in the embedding, after which they are considered as ground truths: all subsequent reasoning in the abstraction layer is restricted to only rely on the derived axioms and rules and may no longer refer to the underlying semantics. Consequently, only theorems derivable in the target logic are derivable in the abstraction layer.[12]

So while abstraction layers are conceptually rather simple, an interesting technical question is how the automation capabilities of the meta-logic can be preserved and reliably restricted to respect the imposed restrictions.

[10] As a matter of fact we did not even state any axioms governing implications or quantifiers in the embedded logic.

[11] Even if the target theory is provably complete with respect to the semantics used for constructing the embedding, i.e. semantic validity implies derivability, we still want to know which axioms and rules can be used to construct a concrete derivation.

[12] Note, however, that this relies on the additional assumption that meta-logical inferences based on the derived axioms and rules correspond to derivations in the target logic, as mentioned in the end of this section.

While Isabelle provides its own mechanisms for abstract reasoning like type **class**es, **locale**s and **specification**s, those are not primarily designed for this exact purpose and come with limitations that can make them unsuitable to achieve that purpose on their own, as described in more detail in the following section.

As mentioned in the last section, the main tool for automated reasoning in Isabelle/HOL is **sledgehammer** (see [44]). **sledgehammer** can be invoked during any proof and will try to automatically find a proof for the current proof goal. To that end, simply speaking,[13] it collects all theorems and definitions derived in the current **theory** context together with all local assumptions (collectively referred to as *facts*) and processes the resulting set of facts heuristically to find a subset of relevant facts. It then encodes the problem of deriving the current goal from the chosen facts in a format that can be consumed by external theorem provers like E [48], SPASS [51], verit [15] or Z3 [37]. This may, for example, involve a translation from higher-order problems to first-order problems. If one of the invoked provers can prove the current goal, **sledgehammer** tries to reconstruct a short proof using Isabelle's native proving methods (which operate directly on Isabelle's trusted reasoning core) that can be directly inserted to prove the current goal.[14]

The relevant part of the process to consider for the purpose of constructing an abstraction layer is the initial selection of facts from the theory context. We do not want **sledgehammer** to use the equational theorems that unfold our semantic definitions, but instead derive the goals from only the axioms and specific derivational rules that correspond to the rules of the deductive system of the embedded theory. **sledgehammer** allows us to provide some guidance in its choice. It is possible to (1) indicate that a certain set of facts is likely to be helpful in the proof (using *add*:), (2) prevent it from using certain facts (either using *del*: or by marking facts with the special attribute *no-atp*) or (3) to provide it with a specific set of facts to use directly without taking any other facts into account.

Conceptually, option (3) is the best fit for the purpose of abstraction layers and was used in [29]. However, **sledgehammer** will no longer employ its heuristics and machine learning algorithms to filter the provided facts for relevance, but will directly use the provided set. Consequently, the proving power and therefore the usefulness of **sledgehammer** is significantly diminished, especially for larger theories.

In our current implementation, we therefore use option (2) instead. However, this comes with some challenges. While the equational theorems introduced by simple **definition**s can easily be collected and marked, other more advanced constructions in Isabelle like type definitions or **lift-definition**s (see [25]) introduce several theorems implicitly. While it is still possible to collect these theorems manually, the process is cumbersome and error-prone.

[13] For a precise description of the full details of the process refer to [44].

[14] Furthermore, for provers like veriT and Z3, *proof reconstruction* using the *smt* tactic is available, i.e. they provide proofs that can (sometimes) be directly replayed relative to Isabelle's trusted reasoning core. See [19] and [13].

On the other hand, it is not possible to simply exclude *all* theorems that were derived up to a certain point, since this includes the theorems of Isabelle's *Main* theory, i.e. - among others - the construction of classical higher-order logic on top of Isabelle's more basic *Pure* logic. This includes theorems **sledgehammer** relies on and disbarring them will leave it non-functional (conceptually, such theorems can be thought of as meta-theorems about derivations in our context).

The solution used in the current embedding of AOT is the use of Isabelle's internal ML API to automatically collect theorems to be added to an exclusion list. For convenience, a new command **AOT-sledgehammer** is introduced that internally configures **sledgehammer** to use the desired set of theorems and then passes the current proof state to it.[15] With this method we can achieve significantly better proof automation than [29].

It is important to note that abstraction layers still rely on the implicit assumption that meta-logical reasoning about derivations in the target logic is faithfully represented by the meta-logical inferences in Isabelle enabled by the constructed deduction rules.

In particular, the deductive system of our target theory is implemented as meta-rules in Isabelle's Pure logic, while the used automation mechanisms additionally rely on the logic of Isabelle/HOL. Consequently, we need to convince ourselves that resulting inferences are reproducible in the target system and, conversely, that derivations in our target system are exhaustively captured by the rules of our abstraction layer. For our embedding of AOT we sketch such an argument in section 4.7.5.

2.5. Isabelle's Native Abstraction Mechanisms

While abstraction layers provide a means to insulate reasoning in our embedding from artifactual theorems (i.e. theorems that are merely semantically valid but not derivable in the target theory; see also 4.8), we additionally use Isabelle's native abstraction mechanisms. This serves to establish an additional intermediate abstraction between the concrete model construction and the derivation of the axioms and deductive system of the target theory, which helps in exploring changes to the model structure without having to adjust the full derivation of the abstraction layer.

2.5.1. Specifications

For example, we extensively use **specification**s (see §11.4 in [55]). A **specification** is used to assert statements about previously uninterpreted constants. The **specification** command opens a proof context that requires the user to show that there exists a concrete instantiation for the given constants, for which the desired statements hold. Internally it then uses Isabelle's Hilbert-Epsilon-operator $SOME\ x.\ \varphi\ x$ to augment the given constants with a concrete definition. We will discuss the technical details of this mechanism in section 4.5. As a consequence, a model of the meta-logic may choose any denotation

[15] Alternatively, we allow configuring **sledgehammer** itself to only use the restricted set of theorems.

for the given constants that satisfies the specification, while the existence of such a denotation is guaranteed by the provided witness. However, depending on the use case of this mechanism, care has to be taken to ensure that there actually are non-trivial choices beyond the provided witness.

To illustrate this issue, we showcase the construction of a (hyper-)intensional conjunction in which $p \wedge q$ implies both p and q and vice-versa, but it does not hold that $(p \wedge q) = (q \wedge p)$. We first show a construction that will fail due to the choice of a representation type that implies extensionality:

typedef o_1 = ‹*UNIV*::*bool set*›**..** — Introduce an abstract type of propositions o_1 with the universal set of booleans (i.e. {*True*, *False*}) as representation set.[16]
definition *valid*-o_1 :: ‹o_1 ⇒ *bool*› **where**
‹*valid*-o_1 p ≡ *Rep*-o_1 p› — Validity is simply given by the boolean representing the proposition.[17]

We introduce an uninterpreted constant for conjunctions with infix syntax.

consts o_1-*conj* :: ‹o_1 ⇒ o_1 ⇒ o_1› (**infixl** ‹∧› *100*)

specification (o_1-*conj*) — We specify our conjunction by introduction and elimination rules.
o_1-*conjE1*: ‹*valid*-o_1 $(p \wedge q) \Longrightarrow$ *valid*-o_1 p›
o_1-*conjE2*: ‹*valid*-o_1 $(p \wedge q) \Longrightarrow$ *valid*-o_1 q›
o_1-*conjI*: ‹*valid*-o_1 $p \Longrightarrow$ *valid*-o_1 $q \Longrightarrow$ *valid*-o_1 $(p \wedge q)$›

We need to prove that there is a term satisfying the above specification. The natural choice is the lifted conjunction on the booleans.

by (*rule exI*[**where** x=‹λ p q . *Abs*-o_1 (*Rep*-o_1 p ∧ *Rep*-o_1 q)›])
(*auto simp*: *Abs*-o_1-*inverse valid*-o_1-*def*)

However, even though the identity of commuted conjunctions is not part of the **specification**, it is *still* derivable.[18]

lemma ‹$p \wedge q = q \wedge p$›
by (*metis Rep*-o_1-*inject* o_1-*conjE1* o_1-*conjE2 valid*-o_1-*def*)

The reason is that there is only one choice for a conjunction operator on the booleans that satisfies our specification and this choice is commutative. We can in fact prove that our conjunction has to be identical to the witness we provided:

lemma ‹(∧) = (λp q . *Abs*-o_1 (*Rep*-o_1 p ∧ *Rep*-o_1 q))›
by (*metis Abs*-o_1-*inject Abs*-o_1-*inverse UNIV-I* o_1-*conjE1* o_1-*conjE2* o_1-*conjI*
type-copy-obj-one-point-absE type-definition-o_1 *valid*-o_1-*def*)

[16] For every type definition using an explicit representation set (**typedef**), we need to prove that the set is non-empty. In the case of the universal set of another type, this proof is trivial, as indicated by the two dots.

[17] For any **typedef**, Isabelle introduces constants prefixed with *Abs*- and *Rep*-, mapping the representation type to the defined abstract type and vice-versa.

[18] Note that if we constructed abstraction layers as discussed in the last section, **sledgehammer** would be prevented from considering the implicit theorems introduced by the type definition of o_1 (which relate the type to its representation type) and, therefore, would not be able to prove this theorem.

In order to avoid this issue, we cannot simply rely on the **specification** command, but also have to take care that the *types* of the specified constants can actually deliver the desired degree of intensionality. In our example, we can introduce an abstract *intensional type* for propositions that merely has a boolean *extension*. First we introduce an abstract type:

typedecl o_2 — Introduce an abstract type for propositions.

Thus far, a model of HOL satisfying our theory may choose any non-empty set as representation set for objects of type o_2. To arrive at a meaningful type of propositions, we axiomatically introduce a surjective extension function mapping the abstract propositions to their boolean extension. The surjectivity of the extension function excludes degenerate models in which there is only one proposition.[19]

axiomatization o_2-*ext* :: ‹$o_2 \Rightarrow$ *bool*› **where**
o_2-*ext-surj*: ‹*surj* o_2-*ext*›

definition *valid*-o_2 :: ‹$o_2 \Rightarrow$ *bool*› **where**
‹*valid*-o_2 $p \equiv o_2$-*ext* p› — Validity of a proposition is given by its boolean extension.

consts o_2-*conj* :: ‹$o_2 \Rightarrow o_2 \Rightarrow o_2$› (**infixl** ‹$\wedge$› *100*)

specification (o_2-*conj*) — We again specify our conjunction by introduction and elimination rules.
o_2-*conjE1*: ‹*valid*-o_2 $(p \wedge q) \Longrightarrow$ *valid*-o_2 p›
o_2-*conjE2*: ‹*valid*-o_2 $(p \wedge q) \Longrightarrow$ *valid*-o_2 q›
o_2-*conjI*: ‹*valid*-o_2 $p \Longrightarrow$ *valid*-o_2 $q \Longrightarrow$ *valid*-o_2 $(p \wedge q)$›

We again need to prove the existence of a term satisfying the given specification. Since our extension function is surjective, a natural suitable witness can be constructed using the inverse of the extension function.

by (*rule exI*[**where** x=‹λ p q . (*inv* o_2-*ext*) (o_2-*ext* $p \wedge o_2$-*ext* q)›])
(*simp add*: o_2-*ext-surj f-inv-into-f valid*-o_2-*def*)

Now as a consequence of our specification, our conjunction is still commutative *under validity*:

lemma ‹*valid*-o_2 $(p \wedge q) =$ *valid*-o_2 $(q \wedge p)$›

Note that the proof (found by **sledgehammer**) now solely relies on the properties of ($\wedge$) given in our specification:

[19] Note, that we can also construct an equivalent type without a meta-logical axiom: we can (1) introduce an uninterpreted constant that defines a set of products (or, alternatively, sums) of an additional uninterpreted type of intensions and the type of extensions (*bool* in the example), (2) specify that this set is both non-empty and large enough to allow for a surjective function to the extensions (the universal set of such products will be a witness for this specification) and (3) use this set as representation set for our intensional type. The existence of a surjective extension function will become derivable from the specification. However, we found that the model-finding tool **nitpick** works better with the equivalent axiomatic introduction of an extension function on an abstract type.

using o_2-*conjE1* o_2-*conjE2* o_2-*conjI* **by** *blast*

However, commuted conjunctions are no longer identical. The model-finding tool **nitpick** (see [12]) can provide a counterexample by constructing a model for o_2 that has more than two members.

lemma ‹$(p \wedge q) = (q \wedge p)$›
nitpick[*user-axioms*, *expect* = *genuine*, *show-consts*, *atoms* o_2 = $p\ q\ r$, *format* = *2*]
oops — Abort proof and discard the lemma.

The model given by **nitpick**[20] chooses a three-element set for type o_2. We chose p, q and r as names for these elements. o_2-*ext* is modelled as $(p := True,\ q := False,\ r := False)$ and $(\wedge)$ as $((p,\ p) := p,\ (p,\ q) := q,\ (p,\ r) := r,\ (q,\ p) := r,\ (q,\ q) := q,\ (q,\ r) := r,\ (r,\ p) := r,\ (r,\ q) := r,\ (r,\ r) := r)$.

This is indeed one of the minimal (non-degenerate)[21] models for conjunctions that are classical under validity, but are not identical under commutation. On the other hand, **nitpick** can also *satisfy* the same statement by providing a model with cardinality 2 for type o_2:

lemma ‹$(p \wedge q) = (q \wedge p)$›
nitpick[*satisfy*, *user-axioms*, *expect* = *genuine*, *show-consts*, *atoms* o_2 = $p\ q$, *format* = *2*]
oops

Note that for the above it is sufficient to find a concrete choice for p and q, s.t. the identity holds for those two propositions. However, nitpick can also produce (in this case the same) model satisfying the identity for all propositions, respectively - equivalently - refute the identity failing to hold:

lemma ‹$\forall\, p\ q\ .\ (p \wedge q) = (q \wedge p)$› — Satisfy the identity for all p and q.
nitpick[*satisfy*, *user-axioms*, *expect* = *genuine*, *show-consts*, *atoms* o_2 = $p\ q$, *format* = *2*]
oops

lemma ‹$(p \wedge q) \neq (q \wedge p)$› — Refute non-identity for arbitrary p and q.
nitpick[*user-axioms*, *expect* = *genuine*, *show-consts*, *atoms* o_2 = $p\ q$, *format* = *2*]
oops

While the above describes a general mechanism that (given a careful choice of types) can be used to force Isabelle to rely on a specific set of specified properties for constants while simultaneously retaining assured consistency,[22] the mechanism has limitations.

For instance, **specification**s are limited in their capability to specify polymorphic constants. While it is both possible to provide a shared specification for all types of a

[20] The precise model may vary for different versions of Isabelle.

[21] The specification for conjunctions alone can also be satisfied in degenerate models, in which either all propositions are true or all propositions are false, i.e. in particular for models with only one proposition. However, we excluded such degenerate models by requiring a surjective extension function, which prevents **nitpick** from considering these degenerate cases.

[22] The specification part is guaranteed to be consistent, since we provided an explicit witness in the process; the consistency of the axiom assuring the surjectivity of the extension function is confirmed by **nitpick**.

polymorphic constant, as well as to provide separate specifications for concrete distinct type instantiations of a polymorphic constant, doing both at the same time is in general not possible.

2.5.2. Type Classes and Locales

Isabelle provides further abstraction mechanisms, e.g. type **class**es and **locale**s, but each comes with its own limitations. Technically, a **locale** (see §5.7 in [55]) is a functor that maps parameters and a specification to a list of declarations. In practice, this can be used to reason relative to abstract parameters that validate a set of assumptions and then **interpret** the **locale** by proving the assumptions for a concrete instantiation of its parameters. As a result of this interpretation of the locale, all declarations of, and in particular all theorems proven in, the locale will be instantiated to the given parameters and added to the theory context. A limitation of **locale**s is that they cannot involve polymorphic assumptions, which prevents us from formulating the full system of AOT abstractly as a single locale.

Type **class**es (see §5.8 in [55]) are technically **locale**s that depend on *exactly one* type variable and additionally introduce an axiomatic type class for which, roughly speaking, the parameters of the locale are introduced as global constants that satisfy the locale assumptions. In practice, type classes can be used to define properties on types and reason about any type with those properties. Type classes can then be *instantiated* for a concrete type[23] by proving that the assumptions are satisfied for a concrete definition of the locale parameters at that type.

For example, it is possible to instantiate a type class to products of two generic types (i.e. type variables) of specific sorts. We use this mechanism to inductively define properties of n-ary relations of AOT as relations among arbitrary tuples (see section 4.1).

Ideally, it should be possible to implement the full axiom system and deduction rules of our target system using a system of type classes and locales (which would provide an abstraction layer that is enforced on the logical level) and then merely to validate the consistency of the construction by instantiating, resp. interpreting these type classes and locales using a concrete semantic construction. However, in a complex target system that involves polymorphic axioms and complex interdependencies between its types, this is not always feasible and we have to rely on abstraction layers as described in the last section.

While a full discussion of the subtleties of type **class**es goes beyond the scope of this thesis, the short summary we provided above should be sufficient for understanding our use of type classes in chapter 4. Furthermore, it is important to note that while we use type classes to formulate theorems generically for several types, logically, the type classes can be eliminated for each concrete instantiation of such a theorem with fully specified concrete types.

[23]More precisely, a type constructor that may depend on additional types that can be restricted to certain type classes, resp. *sorts*.

2.6. Implicit Interpretation and Assignment Functions in SSEs

Models of logical theories are usually formulated in terms of set-theory. In the following chapters, when we say that we construct *models* of the target logic AOT using our embedding, we do not construct classical set-theoretic models, but our implementation forms a model of AOT in HOL.

While a deep embedding would make interpretation and assignment functions explicit, they remain implicit in shallow embeddings. The meta-logic Isabelle/HOL itself involves constants and variables that are reused to represent the constants and variables of our target system. Consequently, we do not have to construct explicit interpretation and assignment functions, but can rely on HOL's semantics for constants and variables.

In simple models of HOL,[24] every type has a set as its domain and a statement is valid, if it holds for every interpretation of the constants of each type and every assignment of the variables at each type.

A set-theoretic model of the embedded logic can be constructed by lifting a set-theoretic model of HOL through the semantic definitions of the SSE. The model-finding tool **nitpick** [12] can aid in making these lifted models concrete.

Technically, a shallow embedding defines a substructure in the models of HOL, which yields a model of the embedded logic under the defined validity.

2.7. Reproducing the Syntax of the Target Theory

To achieve the goal of constructing a custom theorem proving environment for a new theory by the means of an embedding, the primary concern is achieving a faithful representation of its axioms and deductive system and, thereby, to be able to faithfully reproduce reasoning in the embedded system.

However, for the embedding to be of practical use, it is equally important that the resulting representation is readable and, ideally, that a person that is familiar with the embedded theory, but has limited expertise in the particularities of the meta-logical system in which the theory is embedded, can still use the embedding to reason in the target system without a steep learning curve.

Isabelle's *Isar* (*Intelligible semi-automated reasoning*) language itself (see [55]) is, as the name suggests, specifically tailored towards being readable. Isar makes up the *outer syntax* of an Isabelle theory file and consists of commands that specify theorems and structured proofs acting on Isabelle's system of terms and types, which are formulated in *inner syntax*. *Inner syntax* is highly customizable. In the examples in the previous sections we already made use of the ability to define new (bold) operators using *mixfix* notation (see §8.2 in [55]). However, we only used the mechanism to provide symbols to be used inside the grammar tree of Isabelle/HOL's own term structure.[25] In general

[24] Ignoring complications due to e.g. polymorphism.

[25] Thereby we cannot use symbols that are already used in HOL for our target logic.

Isabelle's inner syntax is described by a context-free priority grammar. It consists of a set of *terminal symbols*, an extensible set of *non-terminal symbols* and a set of *productions* (see §8.4 in [55]). For the purpose of embedding the syntax of a target theory during the construction of SSEs, it stands to reason to use the defined validity as root for the grammar subtree of the embedded language.

Reusing the example of the simple modal logic in section 2.3, this can be done as follows:

type-synonym o_3 = ‹*w* ⇒ *bool*›

This time we do not use mixfix notation to directly introduce syntax for the validity constant.

definition *valid*-o_3 :: ‹o_3 ⇒ *bool*› **where** ‹*valid*-o_3 *p* ≡ ∀ *w* . *p w*›

Instead, we introduce a **nonterminal** as grammar root for the syntax of the embedded language. The nonterminal then serves as purely syntactic type for propositions in the grammar of our sub-language.

nonterminal *prop*o_3

The nonterminal can be used as syntactic type in **syntax** definitions.

syntax *valid*-o_3 :: ‹*prop*o_3 ⇒ *bool*› (‹⊨ -› *1*)

Furthermore, we need to specify how propositions can be produced from terminals in the grammar. We want to use simple identifiers to refer to proposition variables. To that end we introduce a *copy-production* rule (a rule that is not tied to a constant). The terminal *id-position* is used for identifiers with additional markup information (i.e. it contains an encoding of the source position of the identifier to be used in the context of Isabelle/PIDE; see [52]).

syntax :: ‹*id-position* ⇒ *prop*o_3› (‹-›)

Now we can already construct a simple term in our new syntax:

term ‹⊨ *p*›

Since we introduce an entirely new grammar subtree that is independent of the inner syntax of HOL, we can now reuse the same symbols for logical connectives as used in HOL (instead of having to use bold versions like in the previous section). We first define the connectives without syntax (here the symbols refer to connectives and operators in the language of HOL):

definition *not*-o_3 :: ‹o_3 ⇒ o_3› **where** ‹*not*-o_3 *p* ≡ λ*w* . ¬*p w*›
definition *imp*-o_3 :: ‹o_3 ⇒ o_3 ⇒ o_3› **where** ‹*imp*-o_3 *p q* ≡ λ*w* . *p w* ⟶ *q w*›
definition *conj*-o_3 :: ‹o_3 ⇒ o_3 ⇒ o_3› **where** ‹*conj*-o_3 *p q* ≡ λ*w* . *p w* ∧ *q w*›
definition *disj*-o_3 :: ‹o_3 ⇒ o_3 ⇒ o_3› **where** ‹*disj*-o_3 *p q* ≡ λ*w* . *p w* ∨ *q w*›
definition *box*-o_3 :: ‹o_3 ⇒ o_3› **where** ‹*box*-o_3 *p* ≡ λ*w* . ∀ *v* . *p v*›
definition *dia*-o_3 :: ‹o_3 ⇒ o_3› **where** ‹*dia*-o_3 *p* ≡ λ*w* . ∃ *v* . *p v*›

And then define syntax for them in our grammar subtree. The syntax definitions are only used for parsing terms of the syntactic type *prop*o_3, i.e. terms in the grammar tree spanned by the marker ⊨ introduced above.

syntax *not*-o_3 :: ‹*propo*$_3$ ⇒ *propo*$_3$› (‹¬-› [*40*] *40*)
syntax *imp*-o_3 :: ‹*propo*$_3$ ⇒ *propo*$_3$ ⇒ *propo*$_3$› (**infixl** ‹⟶› *25*)
syntax *conj*-o_3 :: ‹*propo*$_3$ ⇒ *propo*$_3$ ⇒ *propo*$_3$› (**infixl** ‹∧› *35*)
syntax *disj*-o_3 :: ‹*propo*$_3$ ⇒ *propo*$_3$ ⇒ *propo*$_3$› (**infixl** ‹∨› *30*)
syntax *box*-o_3 :: ‹*propo*$_3$ ⇒ *propo*$_3$› (‹□-› [*50*] *50*)
syntax *dia*-o_3 :: ‹*propo*$_3$ ⇒ *propo*$_3$› (‹◊-› [*50*] *50*)

Now we can start to produce complex terms in our new syntax:

term ‹⊨ □*p* ⟶ *q* ∨ ◊*r*›

However, it is noteworthy that since the introduced grammar subtree is independent of the usual HOL grammar, a lot of details need to be considered. For example, without further work it is not possible to specify the types of terms in our grammar sub-tree. For that to work the :: syntax used in HOL would need to be reintroduced,[26] which requires some familiarity with Isabelle's internals like the purely syntactic constant *-constrain* (see §8.5.4 in [55]).

As a simpler example, we also need to introduce parentheses explicitly in our grammar subtree:

syntax :: ‹*propo*$_3$ ⇒ *propo*$_3$› (‹′(-′)›)
term ‹⊨ *p* ∧ (◊*q* ⟶ *p*)› — Without the above this would not parse.

It is still possible to mix the usual HOL syntax with our newly defined subgrammar to argue about validity:

lemma ‹(⊨ ◊*p* ⟶ *q*) ⟶ (¬(⊨ *p*) ∨ (⊨ *q*))›
  using *dia*-o_3-*def* *imp*-o_3-*def* *valid*-o_3-*def* **by** *auto*

In the above the left-most implication and the diamond operator are the implication of the embedded logic and our defined possibility operator. The other logical connectives are the ones of the meta-logic HOL.

While the mechanism described above is sufficient for introducing an accurate representation of the syntax of most target theories that are compatible with the lexical syntax of Isabelle/Pure,[27] *reasoning* in the logic of the target theory entails additional challenges. For example, reasoning using Kripke-semantics usually involves proving statements relative to a fixed, but arbitrary possible world - however semantic possible worlds are not part of the syntax of the target theory and managing them can become a distraction. Therefore, we not only define custom inner syntax for the language of AOT, but also extend Isabelle's outer syntax by custom commands that hide this complexity (see section 4.3).

[26] Or, alternatively, new syntax could be introduced for the same purpose. In our embedding of AOT we will instead reproduce the fact that PLM implicitly imposes type constraints based on the names of its (meta-)variables.

[27] Note that AOT does not fall into this category and requires additional and more complex means to arrive at a good approximation of its syntax as described in section 4.2.

In the following chapter we describe our target theory AOT in terms of our defined syntax, while the technical construction of the syntax representation itself is discussed in section 4.2.

3. Abstract Object Theory

The following sections provide a brief introduction to Abstract Object Theory (AOT or *object theory*). While the first section will explain the general idea and motivation of object theory, the subsequent sections reproduce the language and axiom system of AOT as implemented in our embedding. In the process, we hint at the major differences between the formulation of AOT in PLM and its incarnation in our embedding, referencing the discussion of implementational details in the next chapter where applicable. Recall that, as mentioned in section 1.4, all definitions and theorems are cited directly from our embedding and thereby verified by Isabelle. Exceptions to this rule are explicitly stated and marked by vertical bars at the page margins.

We restrict ourselves to the discussion of the second-order fragment of AOT which is the target of our embedding in Isabelle/HOL.[1] The second-order fragment is expressive enough for the analysis of a wide variety of objects occurring in philosophy and mathematics, including Basic Logical Objects like Truth Values and Extensions of Propositions (see A.8, resp. PLM chapter 10); Platonic Forms (see PLM chapter 11); Situations, Worlds, Times, and Stories (see A.11, resp. PLM chapter 12); Concepts (see PLM chapter 13) and Natural Numbers (see A.12, resp. PLM chapter 13).[2]

The applications of higher-order object theory and the challenges in representing it in Isabelle/HOL are briefly discussed in chapter 6. To get an intuition for the level of expressiveness of full higher-order object theory, note that it can be used to analyze e.g. Zermelo-Fraenkel set-theory itself as one of its abstract objects.

3.1. Overview

AOT is a metaphysical theory inspired by ideas of Ernst Mally and formalized by Edward Zalta. While the theory has been evolving for several decades (see [56, 59]), its most recent canonical presentation is given in *Principia Logico-Metaphysica* (PLM), which is under continuous development and the most recent version of which can be accessed as online monograph (see [62]). This thesis is written relative to the version dated October 13, 2021 (see [63]). A summary similar to the one in this section can also be found in [31].

AOT draws two fundamental distinctions, one between *abstract* and *ordinary* objects, and one between two modes of predication, namely, classical *exemplification* $[F]x$, or more

[1]In the following chapters up until chapter 6, we will refer to the second-order fragment of AOT plainly as AOT or *object theory*.

[2]The chapter numbering of PLM is relative to [63].

generally, $[R]x_1...x_n$ and *encoding* $x[F]$, or more generally, $x_1...x_n[R]$.[3] The variables x_1, x_2, …, x_n, resp. x, y, z, …, range over both ordinary and abstract objects and we can distinguish claims about these two kinds of objects by using the exemplification predications $O!x$ or $A!x$ to assert, respectively, that x exemplifies *being ordinary* or x exemplifies *being abstract*. Whereas ordinary objects are characterized only by the properties they exemplify, abstract objects may be characterized by both the properties they exemplify and the properties they encode. But only the latter play a role in their identity conditions: $A!x\ \&\ A!y \rightarrow (x = y \equiv \Box\forall F\ (x[F] \equiv y[F]))$, i.e, abstract objects are identical if and only if they necessarily encode the same properties. The identity for ordinary objects on the other hand is classical: $O!x\ \&\ O!y \rightarrow (x = y \equiv \Box\forall F\ ([F]x \equiv [F]y))$, i.e., ordinary objects x and y are identical if and only if they necessarily exemplify the same properties. It is axiomatic that ordinary objects fail to encode properties ($O!x \rightarrow \neg\exists F\ x[F]$), and so only abstract objects can be the subject of true encoding predications. For example, whereas Pinkerton (a real American detective) exemplifies being a detective and all his other properties (and doesn't encode any properties), Sherlock Holmes encodes *being a detective* (and all the other properties attributed to him in the novels), but doesn't exemplify *being a detective*. Holmes, on the other hand, intuitively exemplifies being a fictional character (but doesn't encode this property) and exemplifies any property necessarily implied by *being abstract* (e.g., he exemplifies *not having a mass*, *not having a shape*, etc.).[4]

The key axiom of AOT is the comprehension principle for abstract objects. It asserts, for every expressible condition on properties (i.e. for every expressible set of properties), that there exists an abstract object that encodes exactly the properties that satisfy the condition; formally: $\exists x\ (A!x\ \&\ \forall F\ (x[F] \equiv \varphi\{F\}))$

Here $\varphi\{F\}$ is the notation we use in the embedding to signify that φ may contain a free occurrence of the bound variable F (φ may not contain a free occurrence of x, unless we had explicitly added x in curly braces as well).[5] Taken together, the comprehension principle and the identity conditions of abstract objects imply that abstract objects can be modelled as elements of the power set of properties: every abstract object uniquely corresponds to a specific set of properties.

Given this basic theory of abstract objects, AOT can elegantly define a wide variety of objects that have been postulated in philosophy or presupposed in the sciences, including

[3]Note that we use additional square brackets around property terms in exemplification or encoding formulas, except for specific (primitive or defined) constants like $E!$, $O!$ and $A!$. This is a syntactic concession that makes the process of parsing atomic formulas in Isabelle simpler. In AOT's usual notation these square brackets would be omitted, i.e. exemplification would be written as $Fx_1 \dots x_n$ and encoding as xF.

[4]He encodes *having a mass*, *having a shape*, etc., since these are properties attributed to him, at least implicitly, in the story. As an abstract object, however, he does *not* exemplify these properties, and so exemplifies their negations.

[5]PLM, on the other hand, uses the opposite convention: any *meta-variable* like φ may contain free occurrences of arbitrary variables (even those that are bound at the occurrence of φ) unless explicitly excluded, i.e. instead of $\varphi\{F\}$, PLM simply states φ and uses natural language to add the proviso that x may *not* occur free in φ. See 4.7.2 for a more detailed discussion.

Leibnizian concepts, Platonic forms, possible worlds, natural numbers, logically-defined sets, etc.

Another crucial aspect of the theory is its hyperintensionality: Relation identity is defined in terms of encoding rather than in terms of exemplification. Two properties F and G are stipulated to be identical if they are necessarily *encoded* by the same abstract objects ($F = G \equiv \Box\forall x\ (x[F] \equiv x[G])$).[6] The theory does not impose any restrictions on the properties encoded by a particular abstract object. For example, the fact that an abstract object encodes the property $[\lambda x\ [F]x\ \&\ [G]x]$ does not imply that it also encodes either the property F, or G or even $[\lambda x\ [G]x\ \&\ [F]x]$ (which, although extensionally equivalent to $[\lambda x\ [F]x\ \&\ [G]x]$, is a distinct intensional entity).[7]

Therefore, without additional axioms, pairs of materially equivalent properties (in the exemplification sense), and even necessarily equivalent properties, are not forced to be identical. This is a key aspect of the theory that makes it possible to represent the contents of human thought much more accurately than classical exemplification logic would allow. For instance, the properties *being a creature with a heart* and *being a creature with a kidney* may be regarded as distinct properties despite the fact that they are extensionally equivalent. And *being a barber who shaves all and only those persons who don't shave themselves* and *being a set of all those sets that aren't members of themselves* may be regarded as distinct properties, although they are necessarily equivalent (both necessarily fail to be exemplified).

In the following sections, we provide a brief overview of the language, the axiom system and the deductive system of PLM as implemented in our embedding. For the original formulation of the system and a detailed discussion refer to [63], respectively [62].[8]

3.2. The Language

A precise description of AOT's language can be found in PLM chapter 7. In this section we provide a simplified informal description while explaining some of the deviations from AOT's conventions we use in our embedding.

The language distinguishes between constants, variables and terms at each type. The types of the second-order fragment consist of a type of individuals and of a type of n-place relations (for each $n \geq 0$), i.e. relations among n individuals.[9] Formulas are considered as 0-place relation terms. PLM uses the following naming conventions for referring to the primitive language elements of each type:

[6]Traditionally, one might expect properties to be identical, if they are necessarily *exemplified* by the same objects instead.

[7]Note that this hyperintensionality also extends to propositions. We will see that proposition identity is defined in terms of property identity: $p = q \equiv [\lambda x\ p] = [\lambda x\ q]$

[8]At the time of writing both citations refer to the same version of PLM, but in the future [62] will refer to the most recent formulation of PLM, while [63] will contain the archived version of PLM that served as reference for this thesis. Naturally, referenced items and section numbers of PLM will be relative to [63].

[9]We briefly discuss the full higher-order type theory in chapter 6.

- Primitive individual constants: a_1, a_2, ...
- Individual variables: x_1, x_2, ...
- Primitive n-place relation constants: $P_1^n, P_2^n, \ldots$
- n-place relation variables: $F_1^n, F_2^n, \ldots$
- A distinguished 1-place relation constant for *being concrete*: $E!$

For increased readability, it allows to use less formal names, e.g. to use x, y, z, ... in place of x_1, x_2, ...; p, q, r, ... in place of $F_1^0, F_2^0, \ldots$ or F, G, H, ... in place of $F_1^1, F_2^1, \ldots$, etc.[10]

Additionally, PLM uses Greek letters for *meta-variables*, i.e. schematic meta-logical variables that may range over all variable names or all terms at a given type. By convention, it associates specific kinds of meta-variables with Greek letters (with additional numbered subscripts as needed) as follows:

- Meta-variables ranging over individual variables: ν, μ
- Meta-variables ranging over individual terms: κ
- Meta-variables ranging over n-place relation terms: Π^n
- Meta-variables ranging over formulas (i.e. zero-place relation terms): φ, ψ, ...
- Meta-variables ranging over variables of any type: α, β, ...
- Meta-variables ranging over terms of any type: τ, σ, ...

PLM's system of constants, variables and meta-variables does not have to be reproduced in all detail in our embedding for the following reasons:

- The embedding collapses all alphabetic variants. This is discussed in more detail in section 4.7.1.
- The embedding implicitly generalizes free variables in theorems to *schematic variables*. This is discussed in more detail in section 4.7.3.
- The distinction between constants and variables is done at the meta-logical level of Isabelle/HOL, i.e. variables and constants of the same type are distinguished by declaring them as constant, resp. using them as variable in the meta-logic.

Furthermore, AOT introduces the following primitive formula- and term-forming operators:

- $\neg\varphi$, the *negation* operator
- $\Box\varphi$, the *necessity* operator
- $\boldsymbol{\mathcal{A}}\varphi$, the *actuality* operator
- $\varphi \rightarrow \psi$, the *conditional* operator
- $\forall \alpha\ \varphi\{\alpha\}$, the *universal quantifier*[11]

[10] See PLM item (5).

[11] As mentioned in the previous section, PLM, by default, allows any free variables to occur in instances of a meta-variable, unless specified otherwise. For technical reasons, we choose the opposite convention, i.e. while a meta-variable may still contain any occurrence of variables that would remain *free*, any *bound*

- $\iota x\varphi\{x\}$, the *definite description* operator[12]
- $[\lambda x_1...x_n\ \varphi\{x_1...x_n\}]$, the *relation abstraction*- or λ-operator[13]

AOT uses two kinds of atomic formulas, *exemplification* formulas and *encoding* formulas. In PLM exemplification formulas are written as $\Pi\kappa_1\ldots\kappa_n$, whereas encoding formulas are written as $\kappa_1\ldots\kappa_n\Pi$. In our embedding, we use additional square brackets for easier parsing, i.e. we use:

- $[\Pi]\kappa_1...\kappa_n$ for atomic exemplification formulas
- $\kappa_1...\kappa_n[\Pi]$ for atomic encoding formulas

Furthermore, PLM allows for extending the above language using two kinds of definitions: definitions by identity and definitions by equivalence. While the inferential role of these definitions will be discussed in more detail in section 3.4.2, for now we rely on an intuitive understanding of their meaning. PLM *defines* multiple concepts that are commonly taken as primitive, such as logical existence and identity. These basic definitions can be found in section 7.2 of PLM and are implemented in our embedding in section A.5. In particular, PLM defines the following:

Derived connectives and quantifiers (see A.5.7):[14]

$$\begin{array}{l} \varphi \mathbin{\&} \psi \equiv_{df} \neg(\varphi \rightarrow \neg\psi) \\ \varphi \vee \psi \equiv_{df} \neg\varphi \rightarrow \psi \\ \varphi \equiv \psi \equiv_{df} (\varphi \rightarrow \psi) \mathbin{\&} (\psi \rightarrow \varphi) \\ \exists\,\alpha\ \varphi\{\alpha\} \equiv_{df} \neg\forall\,\alpha\ \neg\varphi\{\alpha\} \\ \Diamond\varphi \equiv_{df} \neg\Box\neg\varphi \end{array}$$

Logical existence, i.e. the conditions under which a term *denotes* (see A.5.37):

$$\begin{array}{l} \kappa{\downarrow} \equiv_{df} \exists\,F\ [F]\kappa \\ \Pi{\downarrow} \equiv_{df} \exists\,x_1...\exists\,x_n\ (x_1...x_n[\Pi]) \\ \varphi{\downarrow} \equiv_{df} [\lambda x\ \varphi]{\downarrow} \end{array}$$

Being *ordinary* and being *abstract* (see A.5.67):

$$\begin{array}{l} O! =_{df} [\lambda x\ \Diamond E!x] \\ A! =_{df} [\lambda x\ \neg\Diamond E!x] \end{array}$$

Identity of individuals (see A.5.72):

$$\kappa = \kappa' \equiv_{df} O!\kappa \mathbin{\&} O!\kappa' \mathbin{\&} \Box\forall\,F\ ([F]\kappa \equiv [F]\kappa') \vee (A!\kappa \mathbin{\&} A!\kappa' \mathbin{\&} \Box\forall\,F\ (\kappa[F] \equiv \kappa'[F]))$$

Identity of properties, i.e. 1-place relations (see A.5.81):

variables that may occur in an instance of the meta-variable have to be explicitly listed in curly brackets. See 4.7.2 for a more detailed discussion. Also note that while the meta-logical $\forall$-quantifier in HOL has wide scope, the universal quantifier of AOT has narrow scope and quantifying over complex formulas generally requires parentheses.

[12] Note that similarly to the universal quantifier above, definite descriptions have narrow scope and using complex formulas as matrix requires parentheses.

[13] Note that this includes the zero-place case $[\lambda\ \varphi]$, read as *that* φ. The scope of the λ-expression is explicitly given with the surrounding square brackets.

[14] The diamond operator $\Diamond\varphi$ can be read as *possibly* φ. The precedence of the operators is demonstrated in A.5.25.

$\Pi = \Pi' \equiv_{df} \Pi\downarrow \,\&\, \Pi'\downarrow \,\&\, \Box\forall x\ (x[\Pi] \equiv x[\Pi'])$

Identity of propositions, i.e. 0-place relations (see A.5.114):

$\varphi = \psi \equiv_{df} \varphi\downarrow \,\&\, \psi\downarrow \,\&\, [\lambda x\ \varphi] = [\lambda x\ \psi]$

Identity of n-place relations ($n \geq 2$):[15]

$\Pi = \Pi' \equiv_{df} \Pi\downarrow \,\&\, \Pi'\downarrow \,\&\, \forall y_1 \ldots \forall y_{n-1}\ ([\lambda x\ [\Pi]xy_1 \ldots y_{n-1}] = [\lambda x\ [\Pi']xy_1 \ldots y_{n-1}]$
$\&\ [\lambda x\ [\Pi]y_1 x y_2 \ldots y_{n-1}] = [\lambda x\ [\Pi']y_1 x y_2 \ldots y_{n-1}]\ \&\ \ldots\ \&\ [\lambda x\ [\Pi]y_1 \ldots y_{n-1}x] =$
$[\lambda x\ [\Pi']y_1 \ldots y_{n-1}x]\ )$

Based on the described language and definitions we can state AOT's axiom system.

3.3. The Axiom System

In the following, we reproduce the full axiom system of the latest formulation of AOT, while commenting on several aspects that are specific to AOT. Unless explicitly noted otherwise, we will directly cite the axioms from our implementation while explaining notational and conceptual differences to the original axiom system. The original axiom system is stated in PLM chapter 8 with detailed explanations. The implementation in our embedding can be found in A.6. Throughout the section we will refer to the statements of the axioms in A.6, which will in turn refer to the item numbers of the respective axioms in PLM.

The first set of axioms build up a Hilbert-style deductive system for negation and implications following Mendelson's [35] system (see A.6.9):

$\varphi \rightarrow (\psi \rightarrow \varphi)$
$\varphi \rightarrow (\psi \rightarrow \chi) \rightarrow (\varphi \rightarrow \psi \rightarrow (\varphi \rightarrow \chi))$
$\neg\varphi \rightarrow \neg\psi \rightarrow (\neg\varphi \rightarrow \psi \rightarrow \varphi)$

The next set of axioms constructs a quantifier logic for a free logic with non-denoting terms (see A.6.16, A.6.30). Formulas of the form $\tau\downarrow$ can be read as *the term τ denotes* and refer to the notion of logical existence that was *defined* in the previous section.[16]

$\forall\alpha\ \varphi\{\alpha\} \rightarrow (\tau\downarrow \rightarrow \varphi\{\tau\})$
$\forall\alpha\ (\varphi\{\alpha\} \rightarrow \psi\{\alpha\}) \rightarrow (\forall\alpha\ \varphi\{\alpha\} \rightarrow \forall\alpha\ \psi\{\alpha\})$
$\varphi \rightarrow \forall\alpha\ \varphi$
$[\Pi]\kappa_1 ... \kappa_n \rightarrow \Pi\downarrow \,\&\, \kappa_1 ... \kappa_n\downarrow$
$\kappa_1 ... \kappa_n[\Pi] \rightarrow \Pi\downarrow \,\&\, \kappa_1 ... \kappa_n\downarrow$

The last two axioms in the list above are noteworthy: they establish that if any atomic exemplification or encoding formula is true, then its primary terms are significant.

[15] The idea here is that two n-place relations are identical, if they denote and all their projections to $n - 1$ objects are identical. In the embedding it is tricky to reproduce the ellipse notation used for this definition directly, therefore the statement here is *not* cited from the embedding, as indicated by the vertical bars at the margins. The implementation of this definition in the embedding can be found in A.5.107 and is discussed in more detail in section 4.6.4.

[16] See section 4.7.2 for a discussion of the free variable notation using curly brackets and slight differences in the formulation compared to PLM. $\kappa_1 \ldots \kappa_n\downarrow$ abbreviates $\kappa_1\downarrow\ \&\ \ldots\ \&\ \kappa_n\downarrow$.

Additionally, PLM establishes a base set of denoting terms with the following axiom:

> $\tau\downarrow$, provided τ is a primitive constant, a variable, or a λ-expression in which the initial λ does not bind any variable in any encoding formula subterm.

Reproducing the natural language condition on τ in the embedding is non-trivial (see A.6.19, which uses the auxiliary predicate *INSTANCE-OF-CQT-2* defined in A.4.1283); we discuss the implementation of this axiom in detail in section 4.6.1.

For a simple intuition, note that all λ-expressions, in which every atomic formula in the matrix is an exemplification formula, denote, while special care has to be taken in the presence of encoding formulas.[17] The axiom will be discussed in more detail later in this chapter.

The next axiom states that identical objects or identical relations can be substituted in formulas. Note that the used identity is not primitive, but was *defined* in the last section (see A.6.69).[18]

$$\alpha = \beta \rightarrow (\varphi\{\alpha\} \rightarrow \varphi\{\beta\})$$

The following axiom (see A.6.73) is the single *modally fragile axiom* of the system. This is signified by the turnstile operator $\vdash$. All other axioms are *modally strict* (for simplicity, we assume the corresponding turnstile operator $\vdash_\Box$ by default and refrain from mentioning it explicitly[19]). The distinction is discussed further in section 3.4.7.[20]

$$\vdash \boldsymbol{\mathcal{A}}\varphi \rightarrow \varphi$$

Intuitively, modally-fragile statements extend to *actual* truths, while modally-strict statements refer to *necessary* truths.

Apart from the above modally-fragile principle, the logic of actuality is governed by the following modally-strict axioms (see A.6.77):

$$\boldsymbol{\mathcal{A}}\neg\varphi \equiv \neg\boldsymbol{\mathcal{A}}\varphi$$
$$\boldsymbol{\mathcal{A}}(\varphi \rightarrow \psi) \equiv \boldsymbol{\mathcal{A}}\varphi \rightarrow \boldsymbol{\mathcal{A}}\psi$$
$$\boldsymbol{\mathcal{A}}\forall\,\alpha\ \varphi\{\alpha\} \equiv \forall\,\alpha\ \boldsymbol{\mathcal{A}}\varphi\{\alpha\}$$
$$\boldsymbol{\mathcal{A}}\varphi \equiv \boldsymbol{\mathcal{A}}\boldsymbol{\mathcal{A}}\varphi$$

The logic of necessity and possibility is axiomatized using the classical K, T and 5 axioms of a propositional S5 modal logic (see A.6.91):

$$\Box(\varphi \rightarrow \psi) \rightarrow (\Box\varphi \rightarrow \Box\psi)$$

[17]Note that this includes "implicit" encoding formulas that merely occur in the definiens of a defined constant used in the matrix. Those are also counted as encoding formula subterms of the matrix. See PLM items (8) and (17.3).

[18]PLM formulates the axiom as: $\alpha = \beta \rightarrow (\varphi \rightarrow \varphi')$, whenever β is *substitutable for* α in φ, and φ' is the result of replacing zero or more free occurrences of α in φ with occurrences of β. This is equivalent to the formulation in the embedding modulo the equivalence of alphabetic variants (see 4.7.1). The precise correspondence is discussed in more detail in section 4.7.2 at the example of the first quantifier axiom above.

[19]Respectively, printing of modally-strict statements cited from the embedding is set up in such a way that it does not print the turnstile operator.

[20]Note that PLM uses $\boldsymbol{\mathcal{A}}\varphi \equiv \varphi$ as axiom instead. However, the right-to-left direction is derivable and future versions of PLM will only use the left-to-right implication instead.

$\Box\varphi \rightarrow \varphi$
$\Diamond\varphi \rightarrow \Box\Diamond\varphi$

Additionally, PLM states the following axiom (see A.6.101) that requires that there might be a concrete object that is not *actually* concrete, thereby ensuring that the domain of ordinary (i.e. possibly concrete) objects is non-empty[21] and committing the system against modal collapse.

$\Diamond\exists\, x\ (E!x\ \&\ \neg\boldsymbol{\mathcal{A}}E!x)$

The classical S5 modal logic is connected to the logic of actuality by the following two axioms (see A.6.108):

$\boldsymbol{\mathcal{A}}\varphi \rightarrow \Box\boldsymbol{\mathcal{A}}\varphi$

$\Box\varphi \equiv \boldsymbol{\mathcal{A}}\Box\varphi$

Definite descriptions in AOT are governed by the following axiom (see A.6.115), which will allow the derivation of a version of Russell's analysis of descriptions (see section 3.4.6):

$x = \boldsymbol{\iota} x\varphi\{x\} \equiv \forall\, z\ (\boldsymbol{\mathcal{A}}\varphi\{z\} \equiv z = x)$

A consistent axiomatization of complex relation terms in AOT requires some care. While λ-expressions follow the classical principles of α-, β- and η-conversion, they have to be suitably restricted to denoting terms, since not all λ-expressions are guaranteed to denote. Also note that the embedding generally collapses alphabetic variants (see 4.7.1), so while a version of α-conversion can be stated, it effectively reduces to the statement that denoting λ-expressions are self-identical in our implementation. α-, β- and η-conversion are formulated as follows (see A.6.125):

$[\lambda\nu_1...\nu_n\ \varphi\{\nu_1...\nu_n\}]{\downarrow} \rightarrow [\lambda\nu_1...\nu_n\ \varphi\{\nu_1...\nu_n\}] = [\lambda\mu_1...\mu_n\ \varphi\{\mu_1...\mu_n\}]$
$[\lambda x_1...x_n\ \varphi\{x_1...x_n\}]{\downarrow} \rightarrow ([\lambda x_1...x_n\ \varphi\{x_1...x_n\}]x_1...x_n \equiv \varphi\{x_1...x_n\})$
$[\lambda x_1...x_n\ [F]x_1...x_n] = F$

Note that the last of the above axioms, η-conversion, also has the 0-place case $[\lambda\ p] = p$.[22]

The following axiom of *coexistence* is specific to AOT and, together with generally extending AOT's free logic to relation terms and the refinement of base cases of denoting terms, a main aspect in the evolution of PLM that was originally triggered by its analysis using our embedding. It states that whenever a λ-expression denotes, any λ-expression with a matrix that is necessarily and universally equivalent on all abstracted variables will denote as well (see A.6.143):

$[\lambda\nu_1...\nu_n\ \varphi\{\nu_1...\nu_n\}]{\downarrow}\ \&\ \Box\forall\,\nu_1...\forall\,\nu_n(\varphi\{\nu_1...\nu_n\} \equiv \psi\{\nu_1...\nu_n\}) \rightarrow$
$[\lambda\nu_1...\nu_n\ \psi\{\nu_1...\nu_n\}]{\downarrow}$

[21] Note that this consequence of the axiom relies, among others, on the fact that AOT allows deriving the Barcan formulas, in particular A.7.3470.

[22] While identical by axiom, the syntactically distinct terms p and $[\lambda\ p]$ in AOT are meant to capture the natural-language distinction between the statement p itself and the statement *that p is true*. Also note that in the embedding the *0*-place case is stated separately for η-conversion (see A.6.139) and α-conversion (see A.6.129). β-conversion in PLM is only stated for $n \geq 1$.

This axiom, together with AOT's move to a general free logic for complex terms, is discussed in more detail in section 3.7.

The remaining axioms govern AOT's second mode of predication, *encoding*.

The first of these axioms reduces n-ary encoding to unary encoding of projections as follows:[23]

$$x_1 \ldots x_n[F] \equiv x_1[\lambda y\ [F]yx_2 \ldots x_n]\ \&\ x_2[\lambda y\ [F]x_1yx_3 \ldots x_n]\ \&\ \ldots\ \&\ x_n[\lambda y\ [F]x_1 \ldots x_{n-1}y]$$

The second axiom governing encoding states that encoding is *modally rigid* (see A.6.188):

$$x[F] \rightarrow \Box x[F]$$

Furthermore, as mentioned in the introduction of this chapter, encoding is reserved for *abstract* objects or in other words: ordinary objects do not encode properties (see A.6.192):

$$O!x \rightarrow \neg \exists\, F\ x[F]$$

The last axiom is the core axiom of AOT, the *Comprehension Principle for Abstract Objects*. For any expressible condition on properties, there exists an abstract object that encodes exactly those properties that satisfy the condition (see A.6.200):

$$\exists\, x\ (A!x\ \&\ \forall\, F\ (x[F] \equiv \varphi\{F\}))$$

All above axioms are to be understood as axiom *schemata*, i.e. their universal closures are axioms as well. Furthermore, for all axioms except the modally-fragile axiom of actuality, their modal closures (i.e. actualizations and necessitations) are taken as axioms as well.

3.4. The Deductive System

While an implementation of the complete deductive system of PLM chapter 9 can be found in A.7, a full discussion of the entire system would go beyond the scope of this thesis. However, we will discuss some aspects in detail with a particular focus on concepts that are required for the construction of natural numbers in chapter 5.

3.4.1. Primitive and Derived Meta-Rules

Since the axioms of AOT are to be understood as axiom schemata, i.e. their statement includes the statement of adequate closures, a single primitive rule of inference suffices for the deductive system of PLM, i.e. Modus Ponens (see A.7.16):[24]

If φ *and* $\varphi \rightarrow \psi$ *then* ψ.

[23]Note that similarly to the definition of n-ary relation identity, the formulation using ellipses is non-trivial to reproduce in the embedding. Therefore we again do *not* cite the axiom directly from the embedding, but state it as given in PLM modulo our notational conventions. The precise implementation in the embedding can be found in A.6.169 and is discussed in more detail in section 4.6.4.

[24]Note that we are still citing rules directly from the embedding using a special printing mode for meta-rules.

While PLM can refer to structural induction on the complexity of a formula and the length of derivations to derive further meta-rules, by the nature of a Shallow Semantic Embedding, the precise term structure is not preserved, but instead terms are directly represented as objects in their semantic domain, and theorem-hood is not defined by means of derivations but internally constructed in terms of semantic validity. For that reason, in our embedding we derive the rules in question by referring to the semantic properties of the embedding. In particular, we derive the following rules semantically:

The deduction theorem (see A.7.170):

If $\varphi \vdash \psi$ *then* $\varphi \to \psi$.

I.e. if assuming φ it can be derived that ψ, then φ implies ψ.

The rule of necessitation RN (see A.7.110 and A.7.106):

If $\vdash_\Box \varphi$ *then* $\Box\varphi$.

The rule RN can only be applied to a formula φ, if φ has a *modally-strict proof*, as signified by $\vdash_\Box$. We discuss this in more detail in section 3.4.7.

The rule of generalization GEN (see A.7.99):

If for arbitrary α: $\varphi\{\alpha\}$ *then* $\forall\,\alpha\ \varphi\{\alpha\}$.

for arbitrary is implemented using a meta-logical all quantifier. This means that φ has to hold for an arbitrary choice of α and therefore independently of any local assumptions about any concrete α. This goes along with PLM's restriction to only allow the application of GEN, if α does not occur free in any assumption.

3.4.2. The Inferential Role of Definitions

PLM uses two kinds of definitions: definitions-by-equivalence $\varphi \equiv_{df} \psi$ and definitions-by-identity $\tau =_{df} \sigma$. While, intuitively, definitions by equivalence imply definiens and definiendum to be equivalent ($\equiv$) and definitions by identity imply them to be identical ($=$), further care is required when stating their precise inferential roles.

Definitions by Equivalence Since equivalence ($\equiv$) is itself *defined* using a definition-by-equivalence (as mentioned in section 3.2), equivalence itself cannot be used to specify the inferential role of definitions-by-equivalence. Instead the inferential role has to be formulated in terms of primitives of the language, i.e. in terms of implications.

To that end, PLM formulates a *Rule of Definition by Equivalence* that we reproduce in the embedding as follows (see A.7.118):

If $\varphi \equiv_{df} \psi$ *then* $\varphi \to \psi$.
If $\varphi \equiv_{df} \psi$ *then* $\psi \to \varphi$.

In other words, a definition-by-equivalence of the form $\varphi \equiv_{df} \psi$ introduces the closures of $\varphi \to \psi$ and $\psi \to \varphi$ as necessary axioms.[25]

[25]Therefore, the rule has axiomatic character and also has to be derived from the semantics in the appendix. The same is true for the rule of definition by identity below.

The principle that a definition-by-equivalence in fact implies definiens and definiendum to be equivalent becomes a derived rule (see A.7.601):

If $\varphi \equiv_{df} \psi$ *then* $\varphi \equiv \psi$.

However, note that while this also implies *necessary* equivalence of definiens and definiendum (using the rule of necessitation RN mentioned above), in AOT necessary equivalence of propositions does not imply their identity. Another noteworthy subtlety in PLM's use of definitions by equivalence is its convention that such definitions are stated using object-level variables, even though those variables act as meta-variables. For instance, following PLM's conventions, the definition of identity for properties (see 3.2) can be stated as:

$$F = G \equiv_{df} F{\downarrow} \;\&\; G{\downarrow} \;\&\; \Box\forall x\, (x[F] \equiv x[G])$$

However, replacing F and G by any property term constitutes a valid instance of this definition. In order to avoid confusion for a reader that is not familiar with this convention, we choose to either state the definitions using meta-variables instead,[26] or state a derived equivalence as theorem using object-variables in its place (which allows forgoing clauses about the significance of the involved terms in the definiendum). I.e. in the upcoming chapters, instead of stating the full definition-by-equivalence for e.g. property identity, we may illustrate the definition using a simpler theorem using regular object-level variables while dropping significance constraints:

$$F = G \equiv \Box\forall x\, (x[F] \equiv x[G])$$

This simplification is justified, since most definitions of PLM are formulated in such a way that the definiens implies the significance of all free terms in the definiendum, so unless noted otherwise it can be assumed that the definiendum of a definition-by-equivalence is false for non-denoting terms. A notable example of an exception to this rule is the definition of non-identity: non-identity between two terms holds not only if both terms denote with different denotations, but also if either of them fails to be significant.

Definitions by Identity A subtlety in definitions by identity is the question of when a defined term denotes. This is made explicit in the formulation of the *Rule of Definition by Identity* (see A.7.141):

If $\tau\{\alpha_1...\alpha_n\} =_{df} \sigma\{\alpha_1...\alpha_n\}$ *then*
$(\sigma\{\tau_1...\tau_n\}{\downarrow} \rightarrow \tau\{\tau_1...\tau_n\} = \sigma\{\tau_1...\tau_n\}) \;\&\; (\neg\sigma\{\tau_1...\tau_n\}{\downarrow} \rightarrow \neg\tau\{\tau_1...\tau_n\}{\downarrow})$.

I.e. if the definiens denotes, a definition by identity implies identity, if the definiens fails to denote, a definition by identity implies that the definiendum fails to denote as well. In the simplest case of a definition-by-identity that does not involve any free variables, the definition-by-identity reduces to a plain identity, if the definiens provably denotes.

[26]For example, property identity may be stated as: $\Pi = \Pi' \equiv_{df} \Pi{\downarrow} \;\&\; \Pi'{\downarrow} \;\&\; \Box\forall x\, (x[\Pi] \equiv x[\Pi'])$

3.4.3. Reasoning in PLM

Based on the fundamental meta-rules above, PLM derives further theorems and rules governing Negations and Conditionals (see A.7.165); Quantification (see A.7.629); Logical Existence, Identity and Truth (see A.7.913); Actuality and Descriptions (see A.7.1720); Necessity (see A.7.2507); Relations (see A.7.4220); Objects (see A.7.7508) and Propositional Properties (see A.7.8826).

Apart from the specific items discussed in the following sections, it may generally be helpful to be aware of the following derived properties of the deductive system:

- Propositional reasoning in AOT is classical.[27]
- Modal reasoning can be read semantically as following Kripke-semantics without accessibility relation and with a fixed designated actual world for the actuality operator. In particular, AOT follows an S5 modal logic with actuality operator and Barcan formulas.
- The free logic extends to all types, but all propositions provably denote. Quantifiers range over denoting objects and the defined identity constitutes an *existing identity*, i.e. to be identical two entities need to both denote *and* denote the same thing.[28]

3.4.4. Restricted Variables

A common theme in abstract object theory is the definition and analysis of certain families of objects. For instance, Possible Worlds, Logical Sets or Natural Numbers are specific families of abstract objects. Furthermore, some constructions involve talking about the Ordinary Objects specifically. To be able to more conveniently state theorems involving such families of objects, PLM introduces a generic mechanism for defining *restricted variables* that range over objects satisfying a certain *restriction condition* (see PLM section 10.5).

A formula ψ that involves a single free, unrestricted variable α, i.e. a formula that can be written as $\psi\{\alpha\}$ in the notational convention of the embedding, is called a *restriction condition*, just in case that it is both *non-empty*, i.e. $\exists\alpha\ \psi\{\alpha\}$ is a (modally-strict) theorem, and has *strict existential import*, i.e. $\psi\{\tau\} \rightarrow \tau\downarrow$ is a (modally-strict) theorem. PLM distinguishes *restriction conditions*, in which non-emptiness and strict existential import are modally-strict and *weak restriction conditions*, in which neither are required to be modally-strict. Since the parts of PLM implemented in our embedding do not involve weak restriction conditions, the embedding thus far forgoes an implementation of them. However, the current implementation can easily be extended to also cover weak restriction conditions.

[27] In particular, as stated in PLM item (113), all classical propositional tautologies are theorems of AOT.

[28] Respectively, denote and satisfy the axiom of the substitution of identicals. Our implementation has the property that PLM's defined identity implies meta-logical identity.

Furthermore, a restriction condition is *rigid*, if additionally $\forall\alpha\ (\psi\{\alpha\} \rightarrow \Box\psi\{\alpha\})$ is a modally-strict theorem.[29]

An example of a rigid restriction condition is *being ordinary*, i.e. $O!x$.[30]

Restricted variables are governed by the following conventions (see PLM item (331)): Let γ be a variable that is restricted by the restriction condition $\psi\{\alpha\}$. Then a quantified formula of the form $\forall\gamma\ \varphi\{\gamma\}$ is to be understood as an abbreviation of $\forall\alpha(\psi\{\alpha\} \rightarrow \varphi\{\alpha\})$ for an unrestricted variable α. Furthermore, $\exists\gamma\ \varphi\{\gamma\}$ abbreviates $\exists\alpha(\psi\{\alpha\}\ \&\ \varphi\{\alpha\})$ and similar conventions are introduced for definite descriptions, λ-expressions and definitions.

For non-rigid restriction conditions, PLM bans the use of free restricted variables in theorem statements and merely allows bound occurrences. However, for rigid restriction conditions PLM allows the use of free restricted variables in theorem statements and extends reasoning in the system by allowing to take $\psi\{\gamma\}$ as modally-strict axiom.

This construction allows natural reasoning with rigidly restricted variables, i.e. the fundamental rules *GEN* and *RN* as well as usual quantifier and modal reasoning (e.g. $\forall$-elimination, existential introduction, Barcan formulas, etc.) can be extended to restricted variables governed by a rigid restriction condition.

3.4.5. Identity on the Ordinary Objects

While the general definition of identity for individuals was given in section 3.2, PLM also introduces an identity *relation* on the ordinary objects and matching infix notation (see A.7.7582):[31]

$$(=_E) =_{df} [\lambda xy\ O!x\ \&\ O!y\ \&\ \Box\forall F\ ([F]x \equiv [F]y)]$$

$$x =_E y \equiv O!x\ \&\ O!y\ \&\ \Box\forall F\ ([F]x \equiv [F]y)$$

Notably, while the above definition of $=_E$ constitutes a denoting *relation* (the λ-expression does not involve encoding claims and thereby denotes by axiom), general identity of both ordinary *and* abstract objects *does* involve encoding claims and does not constitute a general relation of identity. In particular, the assumption that general identity among individuals is a relation contradicts the existence of indistinguishable abstract objects discussed in section 3.8.1.

Identity on the ordinary objects will play an important role in PLM's analysis of Natural Numbers, discussed in chapter 5.

[29] While our embedding supports non-rigid restriction condition, the parts of PLM currently implemented involve only rigid restriction conditions.

[30] It is a theorem that there necessarily exists an ordinary object $\Box\exists x\ O!x$ (see A.7.7510), as a consequence of the modal axiom $\Diamond\exists x\ (E!x\ \&\ \neg\boldsymbol{\mathcal{A}}E!x)$, the definition of *being ordinary* as $O! =_{df} [\lambda x\ \Diamond E!x]$, β-conversion and some modal reasoning. Furthermore, strict existential instance follows from one of the quantifier axioms (see A.6.34). Rigidity is a consequence of the definition of being ordinary (see A.7.4158).

[31] Note that the introduced infix notation $x =_E y$ merely abbreviates $[(=_E)]xy$ and the stated equivalence is a theorem derived by β-conversion.

3.4.6. Definite Descriptions

The following axiom, that was already mentioned in section 3.3, governs definite descriptions:

$$x = \iota x\varphi\{x\} \equiv \forall z\, (\boldsymbol{\mathcal{A}}\varphi\{z\} \equiv z = x)$$

In particular, definite descriptions are *modally rigid* and refer to the object that satisfies the matrix φ in the actual world. While an extensive set of theorems and rules for reasoning with definite descriptions is given in section 9.8 of PLM (see A.7.1720), for an intuitive understanding of descriptions in AOT it suffices to note that they follow a variant of Russell's analysis of definite descriptions. In particular, atomic formulas involving definite descriptions can be translated to existence claims as follows:[32]

$$[\Pi]\iota x\varphi\{x\} \equiv \exists x\, (\boldsymbol{\mathcal{A}}\varphi\{x\}\ \&\ \forall z\, (\boldsymbol{\mathcal{A}}\varphi\{z\} \rightarrow z = x)\ \&\ [\Pi]x)$$

$$\iota x\varphi\{x\}[\Pi] \equiv \exists x\, (\boldsymbol{\mathcal{A}}\varphi\{x\}\ \&\ \forall z\, (\boldsymbol{\mathcal{A}}\varphi\{z\} \rightarrow z = x)\ \&\ x[\Pi])$$

I.e. an atomic formula involving a description is equivalent to there being a unique object that actually satisfies the matrix of the description and this object satisfies the given atomic formula. ***The** author of PLM is called "Edward Zalta"* is equivalent to *There is a unique object that is actually the author of PLM and this object is called "Edward Zalta"*, respectively to *There is an object that is actually the author of PLM, every other object that is actually the author of PLM is the same object, and this object is called "Edward Zalta"*.

Similarly, a definite description denotes, just in case that there is a unique object that actually satisfies its matrix:

$$\iota x\varphi\{x\}\downarrow \equiv \exists!x(\boldsymbol{\mathcal{A}}\varphi\{x\})$$

E.g. "**the** author of Principia Mathematica" does not denote, since Principia Mathematica was actually coauthored by Russell and Whitehead, i.e. there is no unique object that actually authored Principia Mathematica (even if possibly either either of them might have authored it alone in a non-actual world).

3.4.7. Modally-Strict and Modally-Fragile Theorems

PLM constructs two derivational systems, the first, written as $\vdash$, is concerned with modally-fragile theorems, while the second, written as $\vdash_\Box$, is concerned with modally-strict theorems.[33] The main difference between the two is that the $\vdash$-system is equipped with the modally-fragile axiom of actuality and its universal (though not its necessary or actual) closures (as mentioned in section 3.3):

[32] For simplicity we restrict ourselves to the case of unary exemplification and encoding. Analog principles hold for n-ary exemplification and encoding formulas.

[33] To state modally-strict and modally-fragile theorems in our embedding, we also use **AOT-theorem** and **AOT-act-theorem** respectively. Cited statements that are undecorated are modally-strict by default.

$\vdash \boldsymbol{\mathcal{A}}\varphi \rightarrow \varphi$

On the other hand, every modally-strict axiom is also part of the $\vdash$-system. As indicated by this axiom, semantically, the modally-fragile theorems are concerned with all actual truths, whereas modally-strict theorems are concerned with truths that are necessary.[34]

Consequently, the fundamental meta-rule RN mentioned in section 3.4.1 is restricted to modally-strict derivations: If φ has a modally-strict proof, then its necessitation $\Box\varphi$ is an (again modally-strict) theorem.

PLM's axiom system has the property that the actualization of any modally-fragile axiom (in particular $\boldsymbol{\mathcal{A}}(\boldsymbol{\mathcal{A}}\varphi \rightarrow \varphi)$) is a modally-strict theorem (see A.7.1730).

Consequently, for any formula that is derivable as modally-fragile theorem, i.e. $\vdash \varphi$, it holds that $\vdash_\Box \boldsymbol{\mathcal{A}}\varphi$. In particular, it follows from $\vdash \Box\varphi$ that $\vdash_\Box \boldsymbol{\mathcal{A}}\Box\varphi$, which implies $\vdash_\Box \varphi$. PLM refers to this principle as the *weak converse of RN*.

However, while true in our semantics and derivable in the unextended axiom system of PLM, PLM rejects the weak converse of RN in general. The goal is to explicitly allow augmenting the theory by asserting contingent truths without having to assert their actualization as modally-strict fact. After doing so, the weak converse of RN would indeed fail, so in order to keep the reasoning system robust against additional assertions of this kind, PLM does not allow reasoning using the weak converse of RN. A detailed discussion of this issue can be found in PLM item (71).

While section 4.7.4 hints at a potential way of reproducing this strict distinction using a more complex semantics, for simplicity we refrain from doing so in our embedding and instead rely on our abstraction layer to prevent reasoning using the weak converse of RN, while it remains valid in our semantics.[35]

3.5. Interesting Theorems of AOT

Before we continue our technical discussion of the specifics of AOT, we give some examples of interesting abstract objects and properties that can be derived about them.

3.5.1. Truth Values as Abstract Objects

An example of AOT's ability to define metaphysical entities as abstract objects and derive interesting properties about them, is truth values.

In AOT, it is possible to provide a *syntactic* definition of what it means to be a truth value of a proposition (see A.8.11):

$$\mathit{TruthValueOf}(x,p) \equiv_{df} A!x \;\&\; \forall F\ (x[F] \equiv \exists q\ ((q \equiv p) \;\&\; F = [\lambda y\ q]))$$

[34] Consequently, in our models modally-fragile axioms and theorems are semantically valid in a designated actual world, while modally-strict axioms and theorems are valid in all semantic possible worlds.

[35] Note that it is still possible to add contingent truths to the modally-fragile system of the embedding and - while it would immediately become derivable semantically - just refrain from adding a modally-strict actualization of the assertion to the abstraction layer.

An abstract object x is the truth value of a proposition p, just in case it encodes a property F, if and only if there is a proposition q that is equivalent to p and F is the propositional property *being a y, such that q*.

We say that an abstract object x encodes a proposition p, if it encodes the property *being a y, s.t. p*, i.e. if $x[\lambda y\ p]$. Using that notion, it is possible to define two particular truth values, i.e. The True and The False, as *the* abstract object that encodes all propositions that are true, resp. all propositions that are false (see A.8.217):

$$\top =_{df} \iota x(A!x \;\&\; \forall F\ (x[F] \equiv \exists p\ (p \;\&\; F = [\lambda y\ p])))$$

$$\bot =_{df} \iota x(A!x \;\&\; \forall F\ (x[F] \equiv \exists p\ (\neg p \;\&\; F = [\lambda y\ p])))$$

Now it becomes possible to *derive* natural properties of truth values analytically. For instance:

- There are exactly two truth values (see A.8.312):[36]
  $\exists x\ \exists y\ (\mathit{TruthValue}(x) \;\&\; \mathit{TruthValue}(y) \;\&\; x \neq y \;\&$
  $\quad \forall z\ (\mathit{TruthValue}(z) \rightarrow z = x \lor z = y))$
- The True is distinct from The False (see A.8.222): $\top \neq \bot$
- The True and The False are truth values (see A.8.290):[37]
  $\vdash \mathit{TruthValue}(\top)$
  $\vdash \mathit{TruthValue}(\bot)$
- A proposition is true, iff its truth value is The True and it is false, iff its truth value is The False (see A.8.400):
  $\vdash \mathit{TruthValueOf}(x,p) \rightarrow (p \equiv x = \top)$
  $\vdash \mathit{TruthValueOf}(x,p) \rightarrow (\neg p \equiv x = \bot)$

The analysis of truth values is an example of AOT's ability to define and analyze abstract objects that faithfully represent entities that are usually only considered semantically. AOT's analysis of possible worlds that is discussed in the next section is another example of this feature.

3.5.2. Fundamental Theorems of Possible Worlds

Similarly to truth values, possible worlds are usually solely considered semantically. However, AOT allows to *define* possible worlds as a class of abstract objects and derive fundamental theorems about them.

As a first step, consider the following definition of a *Situation* (see A.11.10):

$$\mathit{Situation}(x) \equiv_{df} A!x \;\&\; \forall F\ (x[F] \rightarrow \mathit{Propositional}(F))$$

[36] An abstract object x is a truth value, if it is the truth value of some proposition p: $\mathit{TruthValue}(x) \equiv_{df} \exists p\ \mathit{TruthValueOf}(x,p)$

[37] Note that since descriptions are modally rigid and refer to objects in the actual world, these theorems are modally-fragile, i.e. *actual*, but not *necessary* truths: While there are necessarily exactly two truth values, different possible worlds have different truth values (exactly two in each world). The defined $\top$ and $\bot$ are the truth values of the actual world.

A situation is an abstract object that only encodes propositional properties. A property F is propositional, just in case that there is a proposition p, s.t. $F = [\lambda y\ p]$ (see A.7.8829):

$$\mathit{Propositional}(F) \equiv_{df} \exists p\ F = [\lambda y\ p]$$

Being a situation is a *rigid restriction condition*,[38] so as explained in section 3.4.4, we can use s as a restricted variable that ranges over situations. A situation makes a proposition true, written $s \models p$, just in case it encodes $[\lambda y\ p]$:[39]

$$s \models p \equiv s[\lambda y\ p]$$

Now a *possible world* can be defined as a situation that possibly makes exactly those propositions true that are true (see A.11.1399):

$$\mathit{PossibleWorld}(x) \equiv_{df} \mathit{Situation}(x)\ \&\ \Diamond\forall p\ (x \models p \equiv p)$$

Similarly to situations, it can be shown that being a possible world is a rigid restriction condition, allowing the use of w as a restricted variable ranging over possible worlds.

Now it can be derived that possible worlds are *possible* (i.e. *possibly actual*), *consistent* and *maximal* situations and, furthermore, that any abstract object that is a possible and maximal situation is a possible world:

- $\mathit{Actual}(x) \equiv_{df} \mathit{Situation}(x)\ \&\ \forall p\ (x \models p \rightarrow p)$

  A situation is actual, if it only makes true propositions true (see A.11.728).
- $\mathit{Possible}(x) \equiv_{df} \mathit{Situation}(x)\ \&\ \Diamond \mathit{Actual}(x)$

  A situation is possible, if it is possibly actual (see A.11.1127).
- $\mathit{Consistent}(x) \equiv_{df} \mathit{Situation}(x)\ \&\ \neg\exists p\ (x \models p\ \&\ x \models \neg p)$

  A situation is consistent, if it does not make any proposition and its negation true at the same time (see A.11.1039).
- $\mathit{Maximal}(x) \equiv_{df} \mathit{Situation}(x)\ \&\ \forall p\ (x \models p \vee x \models \neg p)$

  A situation is maximal, if any proposition is either true or false in it (see A.11.1559).
- $\mathit{Possible}(w)\ \&\ \mathit{Consistent}(w)\ \&\ \mathit{Maximal}(w)$

  Possible worlds are possible, consistent and maximal (see A.11.1484, A.11.1508 and A.11.1561).
- $\mathit{PossibleWorld}(x) \equiv \mathit{Maximal}(x)\ \&\ \mathit{Possible}(x)$

  An abstract object is a possible world, if and only if it is a maximal and possible situation (see A.11.1610).

The *fundamental theorems of possible worlds* relate the defined possible worlds to possibility and necessity of the modal logic of AOT (see A.11.2236 and A.11.2263):

$$\Diamond p \equiv \exists w\ w \models p$$

[38]Note that by *being a situation* we refer to the *formula Situation(x)* in this case. The term $[\lambda x\ \mathit{Situation}(x)]$ is not guaranteed to denote. Similarly for *being a possible world* below.

[39]Note that the double turnstile symbol $\models$ used here is a defined symbol in AOT and not the semantic symbol used in chapter 2 and again starting from section 4.1.3 to symbolize truth conditions in a semantic possible world relative to the meta-logic HOL. Also note that by convention the defined double turnstile symbol $\models$ of AOT is to be understood to have the narrowest possible scope, i.e. $w \models p\ \&\ q$ is to be read as $(w \models p)\ \&\ q$.

$\Box p \equiv \forall\, w\; w \models p$

A proposition is possible, just in case *some* possible world makes it true, and necessary, just in case *every* possible world makes it true.

Furthermore, it can be shown that the basic connectives and quantifiers are well-behaved with respect to being true in a possible world, i.e. (see A.11.2361 and following):[40]

- $w \models (p\ \&\ q) \equiv w \models p\ \&\ w \models q$
- $w \models (p \rightarrow q) \equiv w \models p \rightarrow w \models q$
- $w \models (p \vee q) \equiv w \models p \vee w \models q$
- $w \models (p \equiv q) \equiv (w \models p \equiv w \models q)$
- $w \models \forall\, \alpha\ \varphi\{\alpha\} \equiv \forall\, \alpha\ w \models \varphi\{\alpha\}$
- $w \models \exists\, \alpha\ \varphi\{\alpha\} \equiv \exists\, \alpha\ w \models \varphi\{\alpha\}$

Taken together this reproduces the semantic analysis of AOT with Kripke semantics syntactically within the derivational system of AOT itself. It is a notable feature of AOT that it can, in this sense, accurately reason about its own semantics.

While PLM provides an analysis of a range of further interesting objects, including Logical Sets, Platonic forms, Stories and Fictional Characters and Leibnizian Concepts, a full discussion of them would go beyond the scope of this thesis.

Instead, we discuss some further technical properties of the system of AOT that will be relevant for our discussion of natural numbers in chapter 5.

3.6. Avoiding Known Paradoxes

3.6.1. The Clark-Boolos Paradox

Naive formulations of AOT, in which all λ-expression are assumed to denote relations, are subject to the Clark-Boolos Paradox.[41]

In particular consider the λ-expression $[\lambda x\ \exists\, F\ (x[F]\ \&\ \neg[F]x)]$, i.e. *being an object, s.t. there is a property it encodes, but does not exemplify.* The assumption that this property denotes leads to paradox (see A.7.4362): Assuming that the λ-expression denotes, call it K, s.t. $K = [\lambda x\ \exists\, F\ (x[F]\ \&\ \neg[F]x)]$. By the comprehension principle of abstract objects, there is an abstract object a that encodes exactly K and no other properties. Now if a were to exemplify K, it would follow by β-conversion that there is a property that a encodes, but does not exemplify. However, the only property encoded by a is K, which *is* exemplified by a by assumption yielding a contradiction.

[40]Notably, the proofs of the last two theorems were contributed to AOT on the basis of proofs in our embedding.

[41]The paradox was discovered by Romane Clark in a formalization of Meinong's theories by William Rapaport who reported it in [47], p. 225. Independently, George Boolos constructs the same paradox in [14], p. 17, under the name *SuperRussell* in an analysis of Frege's foundations of arithmetic.

If, on the other hand, a does not exemplify K, it follows that a encodes K and does not exemplify K, so it serves as witness to the claim $\exists F\ (a[F]\ \&\ \neg[F]a)$. Thus it follows by β-conversion that a *does* exemplify K yielding a contradiction.

Previous formulations of PLM disbarred λ-expressions like K syntactically: a λ-expression was only considered to be *well-formed*, if its matrix was a so-called *propositional formula*. A formula was defined to be propositional, just in case that it did not contain encoding subformulas. However, an oversight in the precise formulation of these provisos made it possible to reintroduce the paradox as described in the next section.

In the current formulation of PLM, the paradoxical λ-expression is well-formed, but does not fall under the axiom that stipulates base cases of denoting terms (see 3.3): The initial λ binds a variable that occurs in an encoding formula subterm.

Given that the assumption that the λ-expression denotes leads to contradiction, it now provably fails to denote (see A.7.4362):

$$\neg[\lambda x\ \exists G\ (x[G]\ \&\ \neg[G]x)]{\downarrow}$$

3.6.2. Reintroduction of the Clark-Boolos Paradox

When attempting to construct an embedding of a previous formulation of PLM (see [64]) that relied on restricting matrices of λ-expressions to *propositional formulas* as defined in the previous section, we found the following oversight (see [29] and [31]):

Encoding formulas embedded in the matrix of definite descriptions within complex formulas were not considered *encoding subformulas* and thereby such complex formulas were still considered propositional.[42]

This allowed constructing λ-expressions that were considered well-formed, but (actually) equivalent to the paradoxical Clark-Boolos property K discussed above, namely:

$$K' =_{df} [\lambda x\ [\lambda y\ \forall p\ (p \rightarrow p)]\iota z(z = x\ \&\ \exists F\ (z[F]\ \&\ \neg[F]z))]$$

Since $[\lambda y\ \forall p\ (p \rightarrow p)]$ is (necessarily) universally exemplified by all objects, it being exemplified be a definite description is equivalent to the matrix of the description being *actually* satisfied by a unique object, i.e.:[43]

AOT-theorem ‹$[\lambda y\ \forall p\ (p \rightarrow p)]\iota z\ (z = x\ \&\ \exists F\ (z[F]\ \&\ \neg[F]z)) \equiv$
$\exists! z\ (\mathcal{A}(z = x\ \&\ \exists F\ (z[F]\ \&\ \neg[F]z)))$›
proof(*safe intro*!: $\equiv I \rightarrow I$)
  AOT-assume ‹$[\lambda y\ \forall p\ (p \rightarrow p)]\iota z\ (z = x\ \&\ \exists F\ (z[F]\ \&\ \neg[F]z))$›
  AOT-hence ‹$\iota z\ (z = x\ \&\ \exists F\ (z[F]\ \&\ \neg[F]z)){\downarrow}$›
    using *cqt*:*5*:*a*[*axiom-inst*, *THEN* $\rightarrow E$, *THEN* $\& E(2)$] **by** *blast*
  AOT-thus ‹$\exists! z\ (\mathcal{A}(z = x\ \&\ \exists F\ (z[F]\ \&\ \neg[F]z)))$›
    using *actual−desc*:*1*[*THEN* $\equiv E(1)$] **by** *blast*

[42] While the matrix of a definite description (or a λ-expression) is a *subterm* of any formula containing the description (or λ-expression), it is not a *subformula*. See PLM items (6), (7) and (8).

[43] We choose this opportunity to demonstrate that reasoning in our embedding is readable and intuitively understandable, by directly proving the equivalence in the syntax of the embedding. The proof was automatically verified during the generation of this document as mentioned in section 1.4.

next
  AOT-assume ‹$\exists! z\ (\mathcal{A}(z = x\ \&\ \exists F\ (z[F]\ \&\ \neg[F]z)))$›
  AOT-hence ‹$\iota z\ (z = x\ \&\ \exists F\ (z[F]\ \&\ \neg[F]z))\downarrow$›
    using *actual−desc:1*[*THEN* ≡*E*(*2*)] **by** *simp*
  AOT-thus ‹$[\lambda y\ \forall p\ (p \to p)]\iota z\ (z = x\ \&\ \exists F\ (z[F]\ \&\ \neg[F]z))$›
    by (*safe intro*!: *β*←*C cqt*:*2 GEN* →*I*)
qed

The left-hand side is equivalent to $[K']x$ by β-conversion (assuming K' is a well-formed, respectively a denoting relation). The right-hand side can be simplified to $\mathcal{A}\exists F\ (x[F]\ \&\ \neg[F]x)$, so it is equivalent to $\mathcal{A}[K]x$. Thereby, assuming K' denotes yields a modally-fragile proof of a contradiction following the argument given in the previous section.[44]

An obvious solution to this issue would have been to further restrict propositional formulas to not only disbar encoding subformulas, but also encoding formula subterms, i.e. to also disallow encoding formulas embedded in matrices of descriptions and thereby disbarring K' as not well-formed.

However, this had resulted in the loss of the ability to formulate interesting λ-expressions involving descriptions that are safe and were deemed worthwhile to preserve. Therefore, this solution was rejected in favour of extending AOT's free logic to relation terms as described in the next section. In the most recent formulation of AOT, it becomes a theorem that the paradoxical relation K' does not denote on pain of contradiction (see A.7.4560, resp. A.7.4620).

3.7. Extending AOT's Free Logic to Relations

In the aftermath of the discovery of the reintroduction of the Clark-Boolos paradox, AOT's free logic was extended to all its types.[45]

In the process, the definitions for logical existence ($\tau\downarrow$) mentioned in section 3.2 were introduced.[46] Notably, it is possible to define the conditions under which relation terms denote using *encoding*, i.e. $\Pi\downarrow \equiv_{df} \exists x_1 ... \exists x_n\ (x_1 ... x_n[\Pi])$, while a similar definition using exemplification would fail in the second-order fragment, since there are denoting, but necessarily unexemplified properties, i.e. $\exists x_1 ... \exists x_n\ ([\Pi]x_1 ... x_n)$ may be false for denoting Π.[47]

[44] The proof can also be strengthened to be modally-strict, see A.7.4620.

[45] AOT previously also involved a free logic. However, it was restricted to individual terms to account for non-denoting definite descriptions. While there were λ-expressions that were not considered well-formed syntactically, all λ-expressions that were well-formed were implicitly assumed to denote.

[46] Previously, the free logic for individuals relied on a notion of logical existence that was based on identity, i.e. κ was considered to denote, just in case $\exists x\ x = \kappa$. While the new definition of logical existence is more primitive, i.e. it is formulated in terms of primitives of the language rather than defined identity, it now becomes a theorem that $\tau\downarrow \equiv \exists \beta\ \beta = \tau$ (see A.7.1448).

[47] In higher-order object theory, however, it is possible to define existence of relations using higher-order exemplification as $\exists \mathcal{F}\ [\mathcal{F}]\Pi$.

The switch to a richer free logic also involved multiple changes to the axiom system ultimately resulting in the version given in section 3.3. The quantifier axioms were reformulated using the defined notion of $\tau\downarrow$ for all types. Furthermore, α- and β-conversion were restricted to denoting λ-expressions, the coexistence axiom was added and the base cases for denoting terms were adjusted. The coexistence axiom was based on a similar principle that was discovered as an artifactual theorem of the embedding of AOT at the time.[48] Recall its statement as:

$$[\lambda\nu_1...\nu_n\ \varphi\{\nu_1...\nu_n\}]\downarrow \;\&\; \Box\forall\nu_1...\forall\nu_n(\varphi\{\nu_1...\nu_n\} \equiv \psi\{\nu_1...\nu_n\}) \rightarrow [\lambda\nu_1...\nu_n\ \psi\{\nu_1...\nu_n\}]\downarrow$$

It is also referred to as *safe extension axiom*, since it merely asserts that a λ-expression with matrix ψ denotes, in case there provably is a denoting λ-expression with a matrix φ, s.t. both matrices are necessarily equivalent on all objects, i.e. in case the extension of the λ-expression is known to be safe. Consequently, the axiom has no impact on the size of models (or on consistency): a model can always choose the same denotation for $[\lambda\nu_1...\nu_n\ \psi\{\nu_1...\nu_n\}]$ as it chose for $[\lambda\nu_1...\nu_n\ \varphi\{\nu_1...\nu_n\}]$.[49]

Initial versions of PLM that were equipped with a free logic on all types still retained the concept of *propositional formulas* (formulas without encoding subformulas), but dropped the implicit assumption that well-formed λ-expressions (i.e. λ-expressions with propositional matrix) generally denote, but instead excluded λ-expressions with matrices that contain definite descriptions from the base cases of axiomatically denoting terms.

The coexistence axiom allowed to safely derive that certain λ-expressions involving definite descriptions may still denote: Whenever it was possible to eliminate a description from the matrix of a λ-expression using a description-free propositional formula that is necessarily equivalent on all objects, it was safe to derive that the λ-expression denotes.

However, due to the fact that no longer all λ-expressions with propositional matrix could be assumed to denote, the distinction between propositional and non-propositional formulas lost most of its relevance. Consequently, the next step was to simplify the system by replacing this syntactic distinction entirely by a restriction of the base cases of denoting terms, i.e. all λ-expressions became well-formed, but only λ-expressions without definite descriptions and without encoding formula subterms were asserted to denote by axiom.

The, at the time of writing, most recent extension of the set of axiomatically denoting λ-expression, which resulted in the formulation given in section 3.3, allowed us to derive necessary and sufficient conditions for λ-expressions to denote, as explained in section 3.8.2 below. A potential further refinement is discussed in section 4.6.1.

[48] In particular $\exists\, G\Box\forall\, x_1...x_n(Gx_1..x_n \equiv \varphi\{x_1...x_n\}) \rightarrow \exists\, F(F = [\lambda x_1...x_n\ \varphi\{x_1...x_n\}])$.

[49] However, note that this is not a requirement: While, by construction, both λ-expressions are necessarily equivalent under β-conversion, AOT does not require them to be identical.

3.8. Further Properties of AOT

3.8.1. Indistinguishable Abstract Objects

The issue raised in form of the Clark-Boolos Paradox and its variants in section 3.6 is a general issue of the comprehension principle for abstract object and their identity conditions, which intuitively imply that abstract objects correspond to sets of properties, together with the fact that abstract objects are simultaneously meant to themselves *exemplify* properties:

Properties have exemplification-extensions which are traditionally conceived of as sets of individuals.[50] However, if abstract objects correspond to sets of properties and exemplification-extensions of properties themselves correspond to sets of objects, one may wonder how this can be achieved consistently: How can abstract objects be sets of properties and simultaneously (in the simplest case of non-modal and extensional properties) elements of properties?

A semantic solution to this dilemma is given by Aczel models which are described in section 4.1.1. But there are also derivable theorems of AOT that serve to clarify how this dangling paradox may be avoided.

In particular, it is derivable that there are distinct, but exemplification-indistinguishable abstract objects (see A.7.8572):

$$\exists\, x\ \exists\, y\ (A!x\ \&\ A!y\ \&\ x \neq y\ \&\ \forall\, F\ ([F]x \equiv [F]y))$$

The comprehension principle for abstract objects requires the existence of sufficient abstract objects, that it has to be the case that some of them collapse under exemplification. In light of avoiding a violation of Cantor's Theorem one may even argue that *most* abstract objects are indistinguishable, since the cardinality of the set of abstract objects is strictly larger than the cardinality of the set of properties.[51]

Interestingly, though, it is impossible to independently construct two concrete abstract objects that provably fail to be distinguishable. This is also discussed in section 5.19 in the context of our proposed extended comprehension principle for relations among abstract objects. While PLM's proof of the theorem above (see A.7.8572) uses a slightly different construction, we can provide a proof that makes it explicit that we can construct an abstract object particularly in such a way that there has to be a distinct abstract object that is indistinguishable from it:

AOT-theorem $\langle \exists\, x \exists\, y(A!x\ \&\ A!y\ \&\ x \neq y\ \&\ \forall\, F([F]x \equiv [F]y))\rangle$
proof –
— Consider the object a that encodes being indistinguishable from any abstract object that does not encode being indistinguishable from itself.

[50] In a modal setting properties are even associated with multiple sets of objects for different semantic possible worlds or, equivalently, extensions of modal properties are conceived of as mapping objects to sets of possible worlds in which the property is exemplified by the objects.

[51] And even the set of properties in turn has a strictly larger cardinality than the set of urelements, which in Aczel models will serve as proxies for abstract objects to define their exemplification behaviour, as described in more detail in section 4.1.1.

AOT-obtain a **where** *a-prop*:
‹$A!a$ & $\forall F\ (a[F] \equiv \exists y(A!y$ & $F = [\lambda z\ \forall G([G]z \equiv [G]y)]$ & $\neg y[\lambda z\ \forall G([G]z \equiv [G]y)]))$›
using *A−objects[axiom-inst]* $\exists E$*[rotated]* **by** *fast*

— We show that a encodes being indistinguishable from itself using a proof by contradiction.
AOT-have *0*: ‹$a[\lambda z\ \forall G([G]z \equiv [G]a)]$›
proof (*rule raa−cor:1*)
AOT-assume *1*: ‹$\neg a[\lambda z\ \forall G([G]z \equiv [G]a)]$›
AOT-hence ‹$\neg\exists y\ (A!y$ & $[\lambda z\ \forall G([G]z \equiv [G]a)] = [\lambda z\ \forall G([G]z \equiv [G]y)]$ & $\neg y[\lambda z\ \forall G([G]z \equiv [G]y)])$›
by (*safe intro*!: *a-prop[THEN &E(2), THEN* $\forall E$*(1), THEN* $\equiv$*E(3)] cqt:2*)
AOT-hence ‹$\forall y\ \neg(A!y$ & $[\lambda z\ \forall G([G]z \equiv [G]a)] = [\lambda z\ \forall G([G]z \equiv [G]y)]$ & $\neg y[\lambda z\ \forall G([G]z \equiv [G]y)])$›
using *cqt−further:4[THEN →E]* **by** *blast*
AOT-hence ‹$\neg(A!a$ & $[\lambda z\ \forall G([G]z \equiv [G]a)] = [\lambda z\ \forall G([G]z \equiv [G]a)]$ & $\neg a[\lambda z\ \forall G([G]z \equiv [G]a)])$›
using $\forall E$*(2)* **by** *blast*
moreover AOT-have ‹$A!a$ & $[\lambda z\ \forall G([G]z \equiv [G]a)] = [\lambda z\ \forall G([G]z \equiv [G]a)]$ & $\neg a[\lambda z\ \forall G([G]z \equiv [G]a)]$›
by (*safe intro*!: *a-prop[THEN &E(1)] &I rule=I:1 cqt:2 1*)
ultimately AOT-show ‹p & $\neg p$› **for** p **using** *reductio−aa:1* **by** *blast*
qed
— Hence, by construction, there is an abstract object, s.t. being indistinguishable from it is identical to being indistinguishable from a, but which does not encode being indistinguishable from itself.
AOT-hence ‹$\exists y\ (A!y$ & $[\lambda z\ \forall G([G]z \equiv [G]a)] = [\lambda z\ \forall G([G]z \equiv [G]y)]$ & $\neg y[\lambda z\ \forall G([G]z \equiv [G]y)])$›
by (*safe intro*!: *a-prop[THEN &E(2), THEN* $\forall E$*(1), THEN* $\equiv$*E(1)] cqt:2*)
— Call this object b.
then AOT-obtain b **where** *b-prop*:
‹$A!b$ & $[\lambda z\ \forall G([G]z \equiv [G]a)] = [\lambda z\ \forall G([G]z \equiv [G]b)]$ & $\neg b[\lambda z\ \forall G([G]z \equiv [G]b)]$›
using $\exists E$*[rotated]* **by** *blast*
— Now a and b are indistinguishable.
AOT-have ‹$\forall G([G]a \equiv [G]b)$›
proof −
AOT-have ‹$[\lambda z\ \forall G([G]z \equiv [G]a)]a$›
by (*safe intro*!: *β←C cqt:2 GEN ≡I →I*)
AOT-hence ‹$[\lambda z\ \forall G([G]z \equiv [G]b)]a$›
using *b-prop[THEN &E(1), THEN &E(2)] rule=E* **by** *fast*
thus *?thesis*
using *β→C* **by** *blast*
qed
— But while a encodes being indistinguishable from b, b does not encode being indistinguishable from itself and therefore a is not identical to b.
moreover {
AOT-have ‹$a[\lambda z\ \forall G([G]z \equiv [G]b)]$›
using *b-prop[THEN &E(1), THEN &E(2)] 0 rule=E* **by** *fast*
AOT-hence ‹$a \neq b$›
by (*safe intro*!: *ab−obey:2[THEN →E]* $\vee$*I(1)* $\exists$*I(1)*[**where** τ=‹«$[\lambda z\ \forall G([G]z \equiv [G]b)]$»›]

&I b-prop[THEN &E(2)] cqt:2)

}

— Therefore, a and b are witnesses to the claim of the theorem.

ultimately AOT-have ‹$A!a$ & $A!b$ & $a \neq b$ & $\forall G([G]a \equiv [G]b)$›

using *&I a-prop[THEN &E(1)] b-prop[THEN &E(1), THEN &E(1)]* **by** *blast*

AOT-hence ‹$\exists y(A!a$ & $A!y$ & $a \neq y$ & $\forall G([G]a \equiv [G]y))$› **by** *(rule ∃I)*

AOT-thus ‹$\exists x \exists y(A!x$ & $A!y$ & $x \neq y$ & $\forall G([G]x \equiv [G]y))$› **by** *(rule ∃I)*

qed

Notably, the existence of indistinguishable abstract objects can be used to prove that there is no general relation of identity in AOT, i.e. $[\lambda xy\ x = y]$ does not denote:

AOT-theorem ‹$\neg[\lambda xy\ x = y]\downarrow$›

proof*(rule raa−cor:2)* — Proof by contradiction.

AOT-assume *0*: ‹$[\lambda xy\ x = y]\downarrow$›

— Let a and b be witnesses to the theorem discussed above.

AOT-obtain *a b* **where** *1*: ‹$A!a$ & $A!b$ & $a \neq b$ & $\forall F([F]a \equiv [F]b)$›

using *aclassical2 ∃E[rotated]* **by** *blast*

— From our assumption and the fact that a is self-identical, it follows that a exemplifies the projection of the identity relation to a.

moreover AOT-have ‹$[\lambda x\ [\lambda xy\ x = y]ax]a$›

by *(safe intro!: 0 β←C cqt:2 tuple-denotes[THEN* $\equiv_{df}$*I] &I =I)*

— Since a and b are indistinguishable, b has to exemplify this property as well.

ultimately AOT-have ‹$[\lambda x\ [\lambda xy\ x = y]ax]b$›

by *(safe intro!: 1[THEN &E(2), THEN ∀E(1), THEN ≡E(1)] 0 cqt:2)*

— Which by beta-conversion yields that a is identical to b.

AOT-hence ‹$a = b$›

by *(safe dest!: β→C)*

— Which contradicts the fact that a and b are distinct by construction.

AOT-thus ‹p & $\neg p$› **for** p

using *1 &E =−infix[THEN* $\equiv_{df}$*E] reductio−aa:1* **by** *blast*

qed

This aspect of AOT will be of notable importance during the construction of natural numbers in chapter 5. In the following section, we will see another prominent example of a theorem of AOT that involves indistinguishable objects and relates to Aczel models.

3.8.2. Necessary and Sufficient Condition for Relations to Denote

The move to a free logic for relation terms and the iterative extension of the base cases of denoting terms mentioned in section 3.7, ultimately allowed us to contribute the following theorem to AOT:

$$[\lambda x\ \varphi\{x\}]\downarrow \equiv \Box\forall x\ \forall y\ (\forall F\ ([F]x \equiv [F]y) \rightarrow (\varphi\{x\} \equiv \varphi\{y\}))$$

A λ-expression denotes, if and only if necessarily its matrix agrees on all indistinguishable objects.

The proof (see A.7.8603) relies on the fact that under the assumption of the right-hand-side, it follows that $\Box\forall y\ (\exists x\ (\forall F\ ([F]x \equiv [F]y)\ \&\ \varphi\{x\}) \equiv \varphi\{y\})$. Now since

$[\lambda y\ \exists x\ (\forall F\ ([F]x \equiv [F]y)\ \&\ \varphi\{x\})]\downarrow$ by axiom (by construction the initial λ does not bind a variable that occurs in an encoding formula subterm - in particular it occurs only in the exemplification formula $[F]y$), $[\lambda x\ \varphi\{x\}]\downarrow$ follows by the coexistence axiom. The left-to-right direction can be shown by instantiating F to $[\lambda x\ \varphi\{x\}]$ and some modal reasoning.

This theorem has several repercussions. It provides the analytic means to judge whether a λ-expression denotes within the system of AOT itself. Notably, this led to a proof of the existence of world-relative relations and thereby of rigidifying relations, as discussed in more detail in the next section.

Furthermore, it can contribute to a potential reformulation of the construction of natural numbers that does not require a modal axiom that generates ordinary objects. This is mentioned in section 5.21, although at the time of writing, the analysis of this potential change is not yet complete, so the current version of PLM at the time of writing does not yet contain this new enhanced construction.

In general, this theorem is a prime example of the benefits of the semantic analysis of AOT using our embedding that has led to significant theoretical improvements of AOT and may yet allow for further improvements in the future.

Semantically, the theorem is closely related to Aczel models of AOT. The condition of being indistinguishable, $\forall F\ ([F]x \equiv [F]y)$, semantically corresponds to x and y sharing the same *urelement*. Consequently, the theorem states that λ-expressions denote, if their matrix agrees on objects with the same urelements or, in other words, if they can be represented as functions acting on urelements. A more detailed semantic discussion and a precise construction involving a mapping from individuals to urelements and relations modelled as proposition-valued functions acting on these urelements can be found in chapter 4.

3.8.3. World-Relative Relations and Rigidifying Relations

A notable consequence of the theory of possible worlds outlined in section 3.5.2 and the necessary and sufficient conditions for relations to denote described in the previous section is the fact that world-relative relations denote.

In particular, it can be derived that any denoting λ-expression can be relativized to a possible world, i.e. (see A.11.2920 and A.11.2953):

$$[\lambda x\ \varphi\{x\}]\downarrow \rightarrow [\lambda x\ w \models \varphi\{x\}]\downarrow$$

$$[\lambda x_1 \ldots x_n\ \varphi\{x_1 \ldots x_n\}]\downarrow \rightarrow [\lambda x_1 \ldots x_n\ w \models \varphi\{x_1 \ldots x_n\}]\downarrow$$

This allows for a definition of world-relative relations as follows (see A.11.2992):

$$F_w =_{df} [\lambda x_1 \ldots x_n\ w \models [F]x_1 \ldots x_n]$$

Notably, it becomes a theorem that there exist *rigidifying relations*.[52]

[52] Zalta refers to [21], in which Daniel Gallin postulates the existence of rigidifying relations as an axiom.

A relation is *rigid*, if exemplifying it is modally collapsed (see A.11.2995):

$$Rigid(F) \equiv \Box\forall x_1...\forall x_n([F]x_1...x_n \to \Box[F]x_1...x_n)$$

And a relation F *rigidifies* a relation G, just in case F is rigid and exemplifying it is equivalent to exemplifying G (see A.11.2999):

$$Rigidifies(F,G) \equiv_{df} Rigid(F) \;\&\; \forall x_1...\forall x_n([F]x_1...x_n \equiv [G]x_1...x_n)$$

World-relative relations can now be used as a witness to show that there exist rigidifying relations (see A.11.3057):

$$\exists F \; Rigidifies(F,G)$$

Rigidifying relations will play an important role in the construction of natural numbers described in chapter 5 and their existence previously had to be ensured by stating this last theorem as axiom.

3.8.4. Sixteen Distinct Properties

Another result that can be traced back directly to the construction of the embedding is the derivation of a more refined theorem about minimal models of AOT. While previously PLM merely derived that there are six distinct properties, it is a natural consequence of our constructed models that there are at least sixteen distinct properties. This is due to the fact that there need to be at least two distinguishable individuals (discerned by being ordinary and being abstract) and two possible worlds (required by the axiom asserting the existence of a contingently non-concrete objects object as mentioned in 3.3). Two possible worlds imply that there are at least $2^2 = 4$ distinct propositions. Mapping two discernible objects to 4 propositions can be done in $2^4 = 16$ distinct ways.

And indeed we could construct a proof in the system of AOT itself that verifies that this is not a mere artifact of the model construction, but a proper theorem in AOT. See A.7.6909 for a detailed (though somewhat tedious) proof.

Notably, this result also implies that there is at least $2^{16} = 65556$ distinct abstract objects in minimal models of AOT. On the other hand, models that validate the theory of natural numbers described in chapter 5 involve at least countably infinitely many ordinary objects[53] and thereby uncountably many properties and abstract objects.

Before we proceed to discuss AOT's analysis of natural numbers in chapter 5, we describe the technical details of the implementation of AOT in our embedding in the next chapter.

[53] At least in the current construction. A potential future version of the construction mentioned in section 5.21 may instead require at least countably infinitely many *special urelements*, but not ordinary objects.

4. SSE of AOT in Isabelle/HOL

4.1. Model Construction

While the precise model construction of the embedding can be found in A.1, this section provides a high-level description of this construction. The general idea is based on Aczel models of AOT, which are extended to accommodate for AOT's hyperintensional modal logic on the one hand and its free logic for individual and relation terms on the other hand. Furthermore, we use a system of type classes to construct relations of arbitrary arity as relations among tuples of individuals.[1]

Recall that, as mentioned in section 2.6, we do not construct set-theoretic models of AOT, but instead construct models of AOT in HOL, while any set-theoretic model of HOL that validates our construction can be lifted to a set-theoretic model of AOT.

4.1.1. Aczel Models

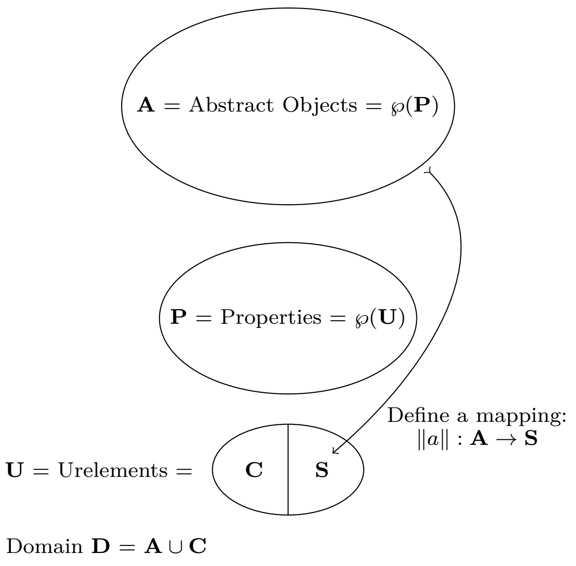

Figure 4.1.: Extensional, non-modal Aczel model of AOT.

[1]However, for each fixed arity of relations the type classes can be logically eliminated.

The general structure of our models is based on Aczel models (see [60]). Aczel models are extensional models that validate both the comprehension principle for abstract objects[2] and classical relation comprehension in the absence of encoding formulas.

Aczel models involve a domain of *urelements* U that is split into *ordinary urelements* C and *special urelements* S. In the extensional, non-modal setting, the power set of the set of urelements suffices for representing properties. Abstract objects in turn are modelled using the power set of properties.

Furthermore, the models involve a (non-injective) mapping from abstract objects to special urelements. The special urelement $||x||$ to which an abstract object x is mapped determines which properties the abstract object x *exemplifies*.

The domain of individuals D is defined as the union of abstract objects and ordinary urelements (resp. ordinary objects).

Any individual $x \in D$ can be associated with an urelement $|x| \in U$:

$$|x| = \begin{cases} x, \text{ if } x \in C \\ ||x||, \text{ if } x \in A \end{cases}$$

Based on this construction the truth conditions for AOT's atomic formulas, i.e. encoding and exemplification, can be defined as follows:

- An object x *exemplifies* a property F, just in case that $|x| \in F$.
- An object x *encodes* a property F, just in case $x \in A$ and $F \in x$.

This construction immediately validates both the identity conditions for abstract objects and the comprehension principle of abstract objects:

- Two abstract objects are identical, if they encode the same properties.
- For every set of properties, there is an abstract object that encodes exactly those properties in the set.

Furthermore, Aczel models validate a restricted version of relation comprehension. Since the truth conditions of any exemplification formula solely depend on the urelement associated with the exemplifying individual, any condition φ on individuals that does not contain encoding claims can equivalently be represented as a condition on urelements. Therefore, for any such condition φ, there exists a relation F that is exemplified by exactly those objects that satisfy φ: $\exists F \forall x([F]x \equiv \varphi\{x\})$, given that φ does not involve encoding claims.

While Aczel models generally demonstrate that abstract objects and encoding can be modelled without being subject to the Clark-Boolos paradox (recall 3.6.1), there are several issues that remain unaddressed, including:

- AOT's relations are not extensional and not even merely intensional, but fully hyperintensional.

[2]The last axiom in section 3.3, resp. A.6.200 in the embedding: $\exists x\ (A!x\ \&\ \forall F\ (x[F] \equiv \varphi\{F\}))$

- Complex individual and relation terms and the free logic of AOT are not modelled explicitly.
- Relation comprehension for formulas in the absence of encoding formulas does not immediately cover all the base cases of axiomatically denoting relation terms as mentioned in section 3.3.
- Aczel models do not cover n-ary relations for $n \geq 2$.[3]

Therefore, while the models used for our embedding inherit the idea of urelements and a mapping from abstract objects to special urelements, we extend the general model structure for our embedding.

4.1.2. Types of the Embedding

The terms of AOT are represented in our embedding using the following types in the meta-logic:[4]

- Type o for formulas, resp. propositions.
- Type κ for individual terms.
- Type $<'a>$ for relation terms. Here $'a$ is a type variable that is restricted to types of class *AOT-κs*, which is instantiated for κ (yielding unary relations, resp. properties, as $<\kappa>$) and arbitrary tuples of type κ (i.e. $<\kappa \times \kappa>$ is used to represent two-place-relations, etc.).[5]

In the following, we will briefly explain how each of these types is constructed.[6] The language elements of AOT (i.e. atomic formulas, logical connectives, quantifiers, complex individual and relation terms) can then be represented by introducing constants that act on objects of these types. We will introduce a custom sub-grammar in the inner syntax of Isabelle/HOL that approximates AOT's syntax and translates to terms involving these constants (as outlined in section 2.7). We will then formulate specifications for the constants that will allow us to derive the axiom system and deduction rules of AOT. The construction of the types will ensure that there are suitable witnesses for these specifications.

The type class *AOT-Term* (see A.1.463) is used as a common type class that is instantiated for each of the types above. It involves a single parameter *AOT-model-denotes* of type $'a \Rightarrow bool$, that determines the meta-logical conditions under which a term of type $'a$ denotes. We will explain how *AOT-model-denotes* is instantiated for each type below.

[3]While in an extensional setting they can be interpreted as sets of tuples of urelements, validating AOT's definition of relation identity in a hyperintensional context requires further care.

[4]Note that types and objects have separate namespaces in Isabelle. Also recall the brief introduction to type classes in section 2.5.2.

[5]Technically, *AOT-κs* is instantiated for products of a type of class *AOT-κ* and a type of *AOT-κs*, while *AOT-κ* abstracts the properties of type κ (and is only instantiated for κ).

[6]This will involve introducing additional types, in particular for urelements, that will not be used for representing terms of AOT directly, but merely to construct the types above.

The additional type $'a$ *AOT-var* is defined for each type $'a$ of class *AOT-Term* using the objects of type $'a$, for which *AOT-model-denotes* is *True*, as representation set (see A.1.1240).[7] This type is used to represent the variables of each of the types above, e.g. κ *AOT-var* will be the type of individual variables. Thereby, variables range exactly over the denoting objects at each type. To be used in place of terms, a variable of type $'a$ *AOT-var* is mapped to its representation type $'a$ using the constant *AOT-term-of-var*.[8]

4.1.3. Hyperintensional Propositions

The hyperintensionality of AOT is modelled at the level of propositions. The construction follows the general method outlined in section 2.5.1.

The type o is introduced as a primitive type (see A.1.12). It is used to represent hyperintensional propositions and is associated with modal extensions following Kripke semantics: a primitive type w for semantic possible worlds is introduced (see A.1.20) and it is axiomatized that there be a surjective mapping *AOT-model-d*o from propositions of type o to Montague intensions, i.e. boolean valued functions on possible worlds (type $w \Rightarrow$ *bool*; see A.1.21).

We define for a proposition φ of type o to be valid in a given semantic possible world v (written $[v \models \varphi]$)[9], just in case *AOT-model-d*o maps p to a Montague intension that evaluates to *True* for v (see A.1.30).

This way, our type of propositions o is assured to contain a proposition for each Montague intension, but does not require the collapse of necessarily equivalent propositions:

For any given Montague intension φ, the inverse of *AOT-model-d*o yields a proposition of type o that is valid in exactly those worlds for which φ evaluates to *True* (see A.1.36).[10]

However, the construction allows for the type o to contain more propositions than there are Montague intensions. I.e there may be two distinct objects p and q of type o that are necessarily equivalent, i.e. they are valid in the same semantic possible worlds. This can be confirmed by **nitpick**:

lemma ‹$\forall\ v\ .\ [v \models p] \longleftrightarrow [v \models q]$› **and** ‹$p \neq q$›
  nitpick[*satisfy*, *user-axioms*, *expect=genuine*]

[7]Since the representation set of a type in Isabelle/HOL cannot be empty, the type class *AOT-Term* involves the assumption that there is an object for which *AOT-model-denotes* is *True*, which has to be proven for each instantiation of the type class and thereby can be assumed for each type of class *AOT-Term*.

[8]Each **typedef** that defines a type using a representation set automatically introduces morphisms, usually prefixed with *Rep-* and *Abs-*, that map objects of the defined type to objects of its representation set and vice-versa. In this case we chose the custom names *AOT-term-of-var* and *AOT-var-of-term* instead of *Rep-AOT-var* and *Abs-AOT-var*.

[9]Note that this use of the double turnstile symbol $\models$ is defined within the meta-logic HOL and distinct from the use in AOT's possible world theory described in section 3.5.2.

[10]This fact relies on the surjectivity of *AOT-model-d*o. The embedding introduces the notation $\varepsilon_{o}\ w\ .\ \varphi\ w$ for the proposition given by *inv AOT-model-d*o φ. We will use such propositions during witness proofs in specifications.

nitpick can find a model in which p and q are represented by two distinct objects, while both of them have the same Montague intension under *AOT-model-do*.

Note, however, that the construction also *allows* for necessary equivalent propositions to be collapsed:

lemma ‹$\forall\ p\ q\ .\ (\forall\ v\ .\ [v \models p] \longleftrightarrow [v \models q]) \longrightarrow p = q$›
  nitpick[*satisfy*, *user-axioms*, *expect=genuine*]

In this case **nitpick** chooses a model in which the type o is isomorphic to the type of Montague intensions $w \Rightarrow bool$, i.e. there are just as many objects of type o as there are Montague intensions.

Just as AOT itself, the model construction does not presuppose the degree of hyperintensionality of propositions.

To instantiate the type class *AOT-Term* for type o, we need to define the conditions under which propositions denote. Since in AOT all formulas denote, *AOT-model-denotes* is *True* for all objects of type o. Consequently, the type o *AOT-var* is isomorphic to the type o.

On top of this hyperintensional type of propositions, the logical connectives will later be defined by **specification** as outlined in section 2.5.1.

Notably, previous versions of our embedding (in particular the construction in [29]), modelled hyperintensionality more explicitly by using an additional primitive type s of *intensional states*. Propositions were modelled explicitly as boolean-valued functions acting on states and possible worlds (type $s \Rightarrow w \Rightarrow bool$). Semantic validity was defined using the evaluation of propositions in a designated *actual* state. The logical connectives were defined to have classical behaviour in the actual state, while their behaviour was left unspecified in non-actual states. While such an explicit construction using intensional states can still serve as a concrete model for our abstract type o, the fact that AOT does not presuppose any additional structure on non-actual states allowed us to replace the explicit construction by the more general abstraction described above.

4.1.4. Extended Aczel Model Structure

Our representation is based on Aczel models, so the construction of the types of individuals and relations relies on urelements.

The embedding introduces a type of urelements υ (see A.1.57) that is comprised of three separate kinds of urelements:

- Ordinary urelements of type ω (see A.1.45),
- Special urelements of type σ (see A.1.53) and
- Null-urelements of type *null* (see A.1.55).

Following the structure of Aczel models, ordinary urelements are used to model ordinary objects and special urelements determine the exemplification behaviour of abstract objects. The additional null-urelements are introduced to be able to distinguish between non-denoting individual terms (see below).

For simple models, the types of ordinary, special and null urelements can all remain purely abstract types.[11]

Hyperintensional relations are modelled as proposition-valued functions. In particular, the embedding introduces the type *urrel* (see A.1.63) that is represented by the set of all functions from urelements to propositions (type $\upsilon \Rightarrow$ o), which map null-urelements to necessarily false propositions.[12] This type of *urrelations* will be in one-to-one correspondence with the denoting property terms, i.e. denoting objects of type $<\kappa>$, respectively objects of type $<\kappa>$ *AOT-var*.

The additional null-urelements serve to avoid two kinds of artifactual theorems:

- Let p be the proposition denoted by the term $[F]\iota x\varphi\{x\}$ and let q be the proposition denoted by the term $[F]\iota x\psi\{x\}$. Furthermore, assume that provably neither of the descriptions denote, i.e. both $\neg\iota x\varphi\{x\}\downarrow$ and $\neg\iota x\psi\{x\}\downarrow$ are theorems. Now while AOT requires p and q to be necessarily equivalent, in particular they are both necessarily false, it does not (in general) presuppose that p is *identical* to q.[13] In the embedding this is achieved by allowing descriptions (with distinct matrices) to be mapped to distinct null-urelements to which the urrelation corresponding to F can assign distinct (albeit necessarily false) propositions.[14] While artifactual theorems of this kind could also be avoided by merely allowing exemplification formulas to choose distinct propositions for distinct non-denoting terms, this would not be sufficient to avoid the second kind of artifactual theorems:
- In AOT there may be distinct properties, s.t. for any object exemplifying either of them necessarily results in the same proposition. I.e. $\forall x\, \Box([F]x = [G]x)$ does *not* imply $F = G$. The $\forall$-quantifier ranges over all denoting individuals. If relations were merely modelled as functions from urelements that correspond to denoting individual terms to propositions, the identity would follow, since two functions are identical, if they agree on all arguments. By introducing null-urelements, however, we allow F and G to vary on additional urelements outside of the range of the quantifier.[15]

[11]I.e. a model of HOL may choose (non-empty) domains of any size for each kind of urelements. In chapter 5 we will discuss a more specific construction that is required to validate the additional axioms needed for the construction of natural numbers.

[12]Note that our construction allows for multiple distinct propositions that are necessarily false.

[13]An example of an exception is the case in which the matrices are alphabetic variants of each other or can be transformed into each other by substituting identical subterms, in which case φ and ψ are also meta-logically identical.

[14]Note that this is not a mere technicality, but it may be desirable to distinguish e.g. between the proposition *The number smaller than 3 is a natural number* (which fails due to there not being a unique such number) and *The number greater than 3 and smaller than 2 is a natural number* (which fails due to there not being any such number). Furthermore, it might make sense to consider the proposition *The present king of France is a natural number* to be an entirely different proposition than the first two. The embedding allows to assign each of the non-denoting definite descriptions distinct null-urelements and thereby allows the propositions to differ. However, it also allows to choose a model with only a single null-urelement which would collapse these propositions.

[15]An alternative approach would be to introduce a primitive type of relations that is merely assigned a proposition-valued function as extension, similarly to how Montague intensions are assigned to the

Note that the additional null-urelements have no impact on minimal models of AOT. In minimal models, propositions are in one-to-one correspondence to Montague intensions: for every boolean valued function on possible worlds there is exactly one proposition. While urrelations have to assign propositions to null-urelements, by construction, urrelations are required to evaluate to necessarily false propositions on null-urelements. Hence, there is only one choice for doing so, namely the single proposition with the constant-false function as Montague intensions. Consequently, the number of relations in minimal models of AOT is unaffected.

As a last ingredient of our Aczel model structure, we require a mapping $\alpha\sigma$ from sets of urrelations (which will be used to represent abstract objects) to special urelements (see A.1.235). As in the basic Aczel model construction, this mapping will determine the exemplification behavior of abstract objects.

For urrelations to become a proper quotient of proposition-valued functions acting on individual *terms*, as described below, we require this mapping to be surjective. However, we can show that any mapping $\alpha\sigma'$ from sets of urrelations to special urelements can be extended to a surjective mapping $\alpha\sigma$ that distinguishes all abstract objects that are distinguished by $\alpha\sigma'$, i.e. if $\alpha\sigma'\ x \neq \alpha\sigma'\ y$, then $\alpha\sigma\ x \neq \alpha\sigma\ y$. This is possible due to the fact that the set of abstract objects is significantly larger than the set of special urelements. In particular, under any arbitrary mapping from abstract objects to special urelements, there has to be at least one abstract object a that shares the same urelement with an amount of other abstract objects that is larger than the total amount of special urelements (proof by a pigeonhole-style argument, see A.1.73). Therefore, any mapping $\alpha\sigma'$ that is not surjective, can be extended to a surjective mapping by further differentiating the abstract objects that share their urelements with a.

To keep the construction as flexible as possible, we first introduce an uninterpreted constant $\alpha\sigma'$ and then generically extend it to a surjective mapping $\alpha\sigma$ (see A.1.234).

To validate extended relation comprehension we can then augment $\alpha\sigma'$ using a suitable **specification**. The precise construction of $\alpha\sigma'$ needed for extended relation comprehension is discussed in more detail in section 5.19.

Additionally, we introduce the constant *AOT-model-concreteω* (see A.1.452) and specify it in such a way, that (1) for every object x (of type ω) there is a possible world w (of type w), s.t. *AOT-model-concreteω* $x\ w$ and (2) there is an object x and a possible world w, s.t. *AOT-model-concreteω* $x\ w \wedge \neg$ *AOT-model-concreteω* $x\ w_0$ (where w_0 is the designated actual world). This constant will be used to construct AOT's relation of being concrete. The specified properties ensure that objects of type ω will be possibly concrete, i.e. ordinary, and that there possibly is an object that is concrete, but not actually concrete, which is asserted by AOT as an axiom. A function that is true for an object x (of type ω) and

primitive type of propositions. However, this would require a polymorphic axiomatization to account for relations of all arities which is incompatible with the model-checking tool **nitpick**. Even only axiomatizing a finite subset of all arities would require **nitpick** to construct significantly larger models and thereby diminish its usefulness. Furthermore, this construction would further complicate validating the definition of n-ary relation identity.

a semantic possible world w (of type w), just in case w is not the actual world w_0, can serve as witness for the specification.[16]

Based on the type of urelements υ and the type of urrelations *urrel* we can construct the type κ of individual terms.

4.1.5. Individual Terms and Properties

The type κ (see A.1.430) consists of ordinary objects of type ω (shared with ordinary urelements), abstract objects modelled as sets of urelements (type *urrel set*) and null-objects of type *null* (shared with null-urelements) that will serve to model non-denoting definite descriptions. We can lift the surjective mapping from abstract objects to special urelements $\alpha\sigma$ to a surjective mapping $\kappa\upsilon$ from individual terms to urelements (i.e. type $\kappa \Rightarrow \upsilon$) (see A.1.434), s.t. for any urelement we can find an object of type κ that is mapped to that urelement (see A.1.439).

To instantiate the type class *AOT-Term* for type κ, we define *AOT-model-denotes* to be *True* for exactly those objects of type κ that are not null-objects.

Relation terms will be defined relative to types of a type class that abstracts individuals and tuples of individuals. We will explain this generic construction below. However, it may be helpful to first consider the case of properties (i.e. type $<\kappa>$) specifically, even though in the embedding this case will only occur as a special case of the generic construction.

Property terms (of type $<\kappa>$) are represented by proposition-valued functions acting on individuals (type $\kappa \Rightarrow$ o). A property term *denotes*, if its representing function φ satisfies the following conditions:

- $\varphi\ \kappa = \varphi\ \kappa'$, whenever $\kappa\upsilon\ \kappa = \kappa\upsilon\ \kappa'$, i.e. φ evaluates to the same propositions for objects that have the same urelements.
- φ evaluates to necessarily false propositions for objects of type κ that do not denote.

Consequently, since $\kappa\upsilon$ is surjective and urrelations have the property to be necessarily unexemplified on null-urelements, denoting property terms are in one-to-one correspondence with urrelations (see A.1.691). This is crucial for constructing encoding and validating the comprehension principle of abstract objects, since abstract objects are modelled as sets of urrelations.

We can now now construct a function that can later serve as witness for our specification of exemplification. For a property term Π and an individual term κ, we can choose a proposition p, such that:

[16] Note that we have to assert the existence of a non-actual world using a meta-logical axiom, see A.1.26. Also note that this construction does not imply that in our embedding no objects will be actually concrete and all ordinary objects will be concrete in all non-actual worlds. While our witness has this additional property, a model of HOL may choose any denotation for *AOT-model-concreteω* that merely satisfies the properties of the specification.

- If Π denotes, then $p = \textit{Rep-rel}\ \Pi\ \kappa$, i.e. the proposition resulting from applying the function representing Π to κ. This proposition will, by construction, be necessarily false, if κ does not denote.
- p is a necessarily false proposition otherwise.

Furthermore, the construction allows us to define the meta-logical truth conditions of encoding as follows: κ encodes Π just in case that (1) Π denotes, (2) κ is represented by an abstract object x and (3) the urrelation corresponding to Π is contained in x.

4.1.6. Type Classes for Individual Terms

The type class *AOT-κs* is a combination of three more specific type classes: *AOT-IndividualTerm* (see A.1.510), *AOT-RelationProjection* (see A.4.407) and *AOT-Enc* (see A.4.714). The latter two formulate conditions on relations among objects of their type variable. Therefore, they can only be formulated after a type of relations is introduced. The type of relations itself will be defined relative to the class *AOT-IndividualTerm*.

The most important parameter of this class is *AOT-model-term-equiv*, an equivalence relation which is satisfied for two objects, if they have common urelements.[17] We furthermore introduce the notion of individual terms to be *regular* and specify a transformation of proposition-valued functions acting on individual terms, s.t. after the transformation the behaviour of the function is solely determined by its values on regular terms. This will be relevant for the definition of n-ary relation identity (see 4.6.4). An unary individual term (i.e. an object of type κ) is always regular, while a tuple will only be regular, if at most one of its elements does not denote.

In the next section, we will introduce relations as proposition-valued functions acting on objects of sort *AOT-IndividualTerm*. The class *AOT-RelationProjection* defines an abstract notion of projections of relations that will be relevant for defining n-ary relation identity. The class *AOT-Enc* defines an abstract notion of encoding. Encoding for type κ is specified as explained in the last section, while for tuples it is constructed in such a way that the axiom of n-ary encoding will become derivable. Together, the three type classes form the class *AOT-κs*.

In the formulation of the axiom system, individuals in ellipses notation will be allowed to have any type of class *AOT-κs*, and relations will be assumed to act on any type of class *AOT-κs*.[18] This way axioms about relations can be stated for all arities at the same time (since the concrete type of individuals κ as well as arbitrary iterated products of it, e.g. $\kappa \times \kappa \times \kappa$, are all of class *AOT-$\kappa$s*).

[17] Note that an *object* of a type of class *AOT-IndividualTerm* may itself e.g. be a pair of two objects of type κ, since the product of κ with itself, i.e. type $\kappa \times \kappa$, is also of class *AOT-IndividualTerm*. *AOT-model-term-equiv* for pairs is defined as the conjunction of *AOT-model-term-equiv* on both projections. Consequently, two tuples $(\kappa_1,\ldots\kappa_n)$ and $(\kappa_1{}',\ldots\kappa_n{}')$ of objects of type κ are *AOT-model-term-equiv*-equivalent if for all $1 \leq i \leq n$, κ_i has the same urelement as $\kappa_i{}'$.

[18] Unless a statement involves explicit exemplification or encoding formulas that imply restrictions on the type, e.g. a particular arity.

4.1.7. Generic Relation Terms

The generic type of relation terms is defined as the type of proposition-valued functions acting on a type of class *AOT-IndividualTerm* (see A.1.582).

To instantiate the type class *AOT-Term* to our generic type of relation terms, we have to define the conditions under which a relation term denotes.

A relation term denotes, if it is represented by a proposition-valued functions φ on individual terms, such that (see A.1.606):

- φ agrees on *AOT-model-term-equiv*-equivalent terms, i.e. it evaluates to the same proposition for individual terms that share the same urelements.
- For non-denoting individual terms, φ evaluates to necessarily false propositions.
- φ is well-behaved on irregular terms (i.e. on irregular terms it evaluates to the proposition given by *AOT-model-irregular* φ, which solely depends on φ's behaviour on regular terms). This will be important to validate the definition of n-ary relation identity and is discussed in section 4.6.4. Note that since unary individual terms, i.e. objects of type κ, are always regular, this restriction does not apply to properties of type $<\kappa>$.

Consequently, exemplification of denoting relation terms, can (as already indicated for the unary case) simply be modelled by the application of the proposition-valued function representing the relation term to the given individual term (which may be a tuple of terms of type κ), while exemplifying non-denoting relation terms yields a necessarily false proposition.[19]

Generic encoding was already described in the last section.

We now have constructed all the required types and prepared the required witnesses for constructing an abstract semantics of AOT using specifications in section 4.4. However, this semantics is formulated using our implementation of AOT's syntax, so in the following two sections we will first briefly discuss how we extend Isabelle's inner syntax by an approximation of the syntax used in PLM and how we extend Isabelle's outer syntax by custom commands used for structured reasoning in the embedding.

4.2. Syntax of the Target Theory

We already discussed the possibility of extending Isabelle's inner syntax in general in section 2.7. Following the method described in that section, we introduce *AOT-prop* as syntactic root type for propositions in AOT (see A.2) and define a custom grammar for AOT on top of it (see A.3). However, Isabelle's high-level mechanisms for defining custom syntax have certain limitations that make an accurate representation of AOT's syntax challenging.

[19]By *can be modelled* here we mean that we can construct a witness for the semantic specification of exemplification.

In particular, Isabelle's lexical analysis is not designed to be configurable. It presupposes that identifiers consist of multiple characters and have to be delimited by whitespace or certain delimiter tokens.

While requiring identifiers to be delimited can be considered as a reasonable syntactic concession, we found that reproducing the compact form of atomic formulas used in PLM results in significantly improved readability.

Therefore we utilize Isabelle's low-level mechanisms to customize syntax by providing transformations on its abstract syntax tree and its term representation written in Standard ML.

In particular, we use **parse-ast-translation**s and **parse-translation**s (see §8.4 in [55]) to split what Isabelle would natively regard as a single identifier. That way we are able to e.g. translate the term $[\Pi]\kappa\kappa'$ to *AOT-exe* Π (κ,κ'). The 2-ary exemplification formula is translated to an application of the constant *AOT-exe* to the relation term and a tuple of individual terms. Similarly, $\kappa\kappa'[\Pi]$ is translated to *AOT-enc* (κ,κ') Π. Involved constants are introduced in A.3 as uninterpreted constants (see A.3.41), which are only later enriched with semantic structure using **specification**s (see A.4 and section 4.4).[20]

Furthermore, PLM associates the symbols used for its terms with their types, as described in section 3.2. While it is possible to rely on Isabelle's type inference in most cases, this will not always result in correctly typed terms without additional type annotations which would negatively affect readability.

For that reason, we construct an extensible system for typing terms based on their names. In particular we introduce the command **AOT-register-type-constraints** that can be used to introduce named categories of types and equip them with type constraints both for unary terms and tuples. We then allow registering symbols as variables and meta-variables of a given category with **AOT-register-variable-names** and **AOT-register-metavariable-names**. The extensible design allows for reproducing AOT's concept of *restricted variables* (see 3.4.4) by further associating a term category with a restriction condition (see A.9).[21]

A danger in the extensive use of complex custom syntax is silent errors in the syntactic translations that could result in an expression to be parsed contrary to their intended meaning. To alleviate this danger we define multiple *printing modes*. The embedding can be configured to print terms in an approximation of AOT's syntax, e.g.:

$$[\Pi]\kappa y \to p \vee \varphi$$

using *meta-syntax*, an enriched version of HOL's syntax without complex transformations, e.g.:

$$(\!|\Pi,(\kappa,\ \langle y\rangle)|\!) \to \langle p\rangle \vee \varphi$$

[20] The type construction discussed in the previous section allows us to construct witnesses for these specifications.

[21] The restriction condition will be added when parsing quantifiers using restricted variables. For rigidly restricted variables a sub-type is introduced that is restricted to all terms that satisfy the restriction condition, allowing to add the restriction condition as axiom for objects of this restricted type.

or as plain HOL terms without any syntactic sugar, e.g.:

$$AOT\text{-}imp\ (AOT\text{-}exe\ \Pi\ (\kappa,\ AOT\text{-}term\text{-}of\text{-}var\ y))\ (AOT\text{-}disj\ (AOT\text{-}term\text{-}of\text{-}var\ p)\ \varphi)$$

Note that while the meta-syntax already involves distracting complexities like the annotation of non-meta-variables using $\langle\text{-}\rangle$, additional explicit syntax for exemplification ⦇-,-⦈ and explicit tuples, plain HOL syntax quickly becomes unreadable for complex terms.

For the purpose of implementing a full theory with an extensive body of theorems, we contend that the improved readability outweighs the potential danger of complex syntax transformations, especially given the ability to confirm the accuracy of the translation using less complex printing modes.

4.3. Extending Isabelle's Outer Syntax

While the syntax transformations described in the last section go a long way in allowing the intuitive statement of terms and formulas of AOT, *reasoning* in the target logic entails additional challenges.

For example, reasoning in the embedding involves keeping track of the semantic possible world in which statements are valid. To avoid this cognitive overhead, we implement a copy of Isabelle's Isar language in Standard ML that automatically handles semantic possible worlds and allows theorem statements and proofs to be transferred directly from and to PLM without the need of explicitly mentioning semantic possible worlds.

While modally-strict theorems of PLM are valid in all semantic possible worlds, conceptually its proofs work relative to an arbitrary but fixed world. For proving a necessary fact during a proof, e.g. $\Box\varphi$, PLM often reasons by providing a *modally-strict* sub-proof of φ and appealing to the rule RN. In our embedding we reproduce this by introducing an outer syntax command **AOT-modally-strict {** that opens a block of reasoning relative to a fresh possible world. For example:

AOT-theorem ‹$\Box(\neg\varphi\ \&\ \neg\psi) \rightarrow \Box(\varphi \equiv \psi)$›
proof(*rule* $\rightarrow I$)
  AOT-assume *0*: ‹$\Box(\neg\varphi\ \&\ \neg\psi)$›
  — Start a modally-strict sub-proof.
  AOT-modally-strict {
    AOT-assume ‹$\neg\varphi\ \&\ \neg\psi$›
    AOT-hence ‹$\varphi \equiv \psi$›
      by (*metis* $\&E$ $\rightarrow I$ $\equiv I$ *reductio−aa:1*)
  }
  — Conclude the necessitation of the result by RN.
  AOT-hence ‹$\Box((\neg\varphi\ \&\ \neg\psi) \rightarrow (\varphi \equiv \psi))$›
    by (*metis* $\rightarrow I$ *RN*)
  AOT-thus ‹$\Box(\varphi \equiv \psi)$› **using** *0 qml:1*[*axiom-inst*] $\rightarrow E$ **by** *blast*
qed

This corresponds to the following proof using Isabelle's native outer syntax:

theorem ‹[$v \models \Box(\neg\varphi \ \& \ \neg\psi) \rightarrow \Box(\varphi \equiv \psi)$]›
proof(*rule* $\rightarrow I$)
  assume *0*: ‹[$v \models \Box(\neg\varphi \ \& \ \neg\psi)$]›
  {
    fix w — We choose a fresh possible world for our sub-proof.
    assume ‹[$w \models \neg\varphi \ \& \ \neg\psi$]›
    hence ‹[$w \models (\varphi \equiv \psi)$]›
      by (*metis* &*E* $\rightarrow I$ $\equiv I$ *reductio−aa*:*1*)
  }
  hence ‹[$v \models \Box((\neg\varphi \ \& \ \neg\psi) \rightarrow (\varphi \equiv \psi))$]›
    by (*metis* $\rightarrow I$ *RN*)
  thus ‹[$v \models \Box(\varphi \equiv \psi)$]› **using** *0 qml*:*1*[*axiom-inst*] $\rightarrow E$ **by** *blast*
qed

Additionally, we introduce the command **AOT-define**, which allows to directly state definitions of PLM (see 3.4.2). Internally, this involves introducing a new constant for the defined entity and setting up the syntax for parsing and printing according to the specified *syntactic* type (while the logical type of the constant is deduced). This new constant is then automatically specified to fulfill the given definition using a mechanism similar to the **specification** command, while the entailed existence proof is constructed automatically.[22]

The convenience of this mechanism becomes apparent by inspecting a definition of *exclusive or*:

AOT-define *xor1* :: ‹$\varphi \Rightarrow \varphi \Rightarrow \varphi$› (**infixl** ‹*XOR1*› *10*)
  xor1-spec: ‹φ *XOR1* $\psi \equiv_{df} (\varphi \vee \psi) \ \& \ \neg(\varphi \ \& \ \psi)$›

This is (roughly)[23] the same as:

consts *xor2* :: ‹o ⇒ o ⇒ o›
syntax *xor2* :: ‹$\varphi \Rightarrow \varphi \Rightarrow \varphi$› (**infixl** ‹*XOR2*› *10*)
specification(*xor2*)
  xor2-spec: ‹*AOT-model-equiv-def* «φ *XOR2* ψ» «$(\varphi \vee \psi) \ \& \ \neg(\varphi \ \& \ \psi)$»›
  by (*auto intro*!: *exI*[**where** x=‹$\lambda \ \varphi \ \psi \ . \ \varepsilon_o \ w \ . \ [w \models (\varphi \vee \psi) \ \& \ \neg(\varphi \ \& \ \psi)]$›]
      simp: *AOT-model-equiv-def AOT-model-proposition-choice-simp*)

We also introduce auxiliary commands like **AOT-find-theorems** and **AOT-sledgehammer** to aid in constructing proofs. **AOT-find-theorems** works similar to the Isar command **find-theorems**, but automatically parses AOT syntax and generalizes concrete variables to schematic variables for pattern matching. **AOT-sledgehammer** is a wrapper that invokes **sledgehammer** while restricting its search for theorems, s.t. the model-specific theorems are ignored and only the theorems and rules of the abstraction layer are allowed for proofs.

The list of commands can be found in A.2, while the actual ML implementation is available at [27].

[22] The existence proofs are generally trivial: the definiens itself can be chosen as witness.

[23] **AOT-define** additionally supports our printing modes and performs internal book-keeping needed for example for the substitution methods to recognize the new definition.

4.4. Representation of an Abstract Semantics of AOT

In A.4, we construct an abstract semantics for the primitive (and some of the basic defined) language elements of AOT. The goal of this layer of abstraction is to specify only the properties of the models that are required to derive the axiom system and rules of AOT later.

The defined semantics heavily relies on Isabelle's **specification** command to abstract specific model choices to more general semantic properties. The model construction merely enables us to construct witnesses for the specifications.

As a simple example, we specify implications by requiring that $\varphi \rightarrow \psi$ is true in a semantic possible world w, just in case φ being true in w implies ψ being true in w (see A.4.21).

More complex examples include the specification of descriptions (see A.4.71) and the joint specification of exemplification and λ-abstraction (see A.4.125).

Notably, we specify necessity (see A.4.32) using validity in all semantic possible worlds and actuality (see A.4.38) using validity in a designated actual world w_0 (see also 4.7.4). Furthermore, we specify AOT's identity as existing identity of meta-logical terms (see A.4.63), while we derive that this corresponds to AOT's definition of identity at each type in A.5.72.[24]

One goal of this intermediate layer of abstraction is to keep the derivation of the abstraction layer that contains the axioms and the deductive system of AOT impervious to minor changes in the model construction.

However, it also eliminates artifactual theorems: instead of simply defining λ-abstraction and exemplification using a concrete model construction, we introduce them using abstracted properties and merely provide a concrete witness that satisfies those properties. This increases the choice of admissible models of HOL validating our construction, since such a model is not restricted to the provided witness, but is merely bound by the abstract properties. This eliminates artifactual theorems that would merely be true for our provided witness, but are not derivable from the required properties.

For example, in the witness proof of the specification of exemplification and λ-abstraction (see A.4.125), we define exemplification, as indicated in the previous sections, as a function *exe* (type $<'a> \Rightarrow 'a \Rightarrow \mathrm{o}$ with $'a$ of sort *AOT-IndividualTerm*) taking a relation term Π and individual terms κs to a proposition p, s.t. if Π denotes, p is given by applying the function representing Π to the individual terms κs, and if Π does not denote, p is a specific, fixed necessarily false proposition. This choice of a witness implies that $[\Pi]\kappa = [\Pi]\kappa'$ for any κ and κ', whenever Π does not denote. However, since our specification does not imply this fact, the construction still allows for models in which $[\Pi]\kappa$ is a proposition that is distinct from $[\Pi]\kappa'$ for distinct κ and κ' (though both propositions have to be necessarily false).

In this sense, the technical details of the constructed witnesses are not particularly relevant in contrast to that we (1) have chosen representation types and basic definitions

[24]Logical existence $\tau\!\downarrow$ is handled similarly.

(e.g. for terms to denote) that *allow* constructing suitable witnesses, (2) our specification is sufficiently strong to validate the axiom system of AOT and (3) our specification is weak enough and our types are general enough to preserve hyperintensionality and avoid most artifactual theorems. The details of our specifications can be found in A.4.

4.5. Specifications and the Hilbert-Epsilon-Operator

As mentioned in section 2.5.1, the **specification** command internally uses Isabelle's native Hilbert-Epsilon-operator $\mathit{SOME}\ x.\ \varphi\ x$. This operator is axiomatized in the meta-logic using the following single principle:

$$\varphi\ x \Longrightarrow \varphi\ (\mathit{SOME}\ x.\ \varphi\ x)$$

In particular, this implies that the operator behaves like the classical Hilbert-Epsilon-operator, i.e. it holds that $(\exists\, x.\ \varphi\ x) = \varphi\ (\mathit{SOME}\ x.\ \varphi\ x)$. Consequently, whenever there is a witness for φ, then whatever is true for *everything* that satisfies φ is true for $\mathit{SOME}\ x.\ \varphi\ x$:

$$[\![\exists\, a.\ \varphi\ a;\ \bigwedge x.\ \varphi\ x \Longrightarrow \psi\ x]\!] \Longrightarrow \psi\ (\mathit{SOME}\ x.\ \varphi\ x)$$

However, it is noteworthy that this operator obeys the following principle of extensionality:

$$(\forall\, x.\ \varphi\ x = \psi\ x) \longrightarrow (\mathit{SOME}\ x.\ \varphi\ x) = (\mathit{SOME}\ x.\ \psi\ x)$$

This is due to the fact, that in the meta-logic, extensional equivalence implies identity, i.e. the antecedent implies $\varphi = \psi$ and the consequent follows by substitution of identicals. Therefore, we *cannot* e.g. define an intensional conjunction as follows (we reuse the type o_2 and its defined validity from section 2.5.1):[25]

definition o_2-*conj'* (**infixl** ‹∧''› *100*) **where**
‹$\varphi \wedge' \psi \equiv \mathit{SOME}\ \chi\ .\ \mathit{valid}\text{-}o_2\ \chi \longleftrightarrow (\mathit{valid}\text{-}o_2\ \varphi \wedge \mathit{valid}\text{-}o_2\ \psi)$›

Since it holds that $(\mathit{valid}\text{-}o_2\ \chi = (\mathit{valid}\text{-}o_2\ \varphi \wedge \mathit{valid}\text{-}o_2\ \psi)) = (\mathit{valid}\text{-}o_2\ \chi = (\mathit{valid}\text{-}o_2\ \psi \wedge \mathit{valid}\text{-}o_2\ \varphi))$, commutativity of $(\wedge')$ is immediately derivable:

lemma ‹$(p \wedge' q) = (q \wedge' p)$›
unfolding o_2-*conj'-def* **by** *metis*

However, we can avoid this issue, if we do not define the *value* of the conjunction function for specific arguments using the Epsilon-operator, but instead the conjunction function itself, i.e.:

definition o_2-*conj''* (**infixl** ‹∧'''› *100*) **where**
‹$(\wedge'') \equiv \mathit{SOME}\ \mathit{conj}\ .\ \forall\ \varphi\ \psi\ .\ \mathit{valid}\text{-}o_2\ (\mathit{conj}\ \varphi\ \psi) = (\mathit{valid}\text{-}o_2\ \varphi \wedge \mathit{valid}\text{-}o_2\ \psi)$›

[25] Note that in *mixfix* notation a single quote ' is used as escape character for distinguishing placeholders - from underscores '-. A syntactic single quote is therefore given as ''.

This way, our conjunction has any property that is true for *all possible* functions that behave as conjunction under validity. In other words, any choice for a concrete conjunction is admissible, including intensional ones, as long as it has our required extensional property under validity.[26]

This is exactly how the **specification** command works: the specification statements are transformed to closed terms by universal generalization and combined via conjunction and the result is used as the matrix of the Hilbert-Epsilon-operator. Given the provided witness, the desired properties of the Hilbert-Epsilon term become derivable.

Note that the extensionality of the Hilbert-Epsilon operator still implies that any other operator defined using a meta-logically equivalent condition is identical, i.e.:

definition o_2-*conj*''' (**infixl** ‹∧'''''› *100*) **where**
‹(∧''') ≡ *SOME conj* . ∀ φ ψ . (*valid*-o_2 ψ ∧ *valid*-o_2 φ) = *valid*-o_2 (*conj* φ ψ)›

lemma ‹(∧'') = (∧''')›
by (*auto intro*!: *Eps-cong simp*: o_2-*conj'''-def* o_2-*conj''-def*)

To avoid this issue completely, we would need to introduce an additional dependency on a meta-logical parameter that is allowed to vary across otherwise meta-logically equivalent definitions.[27]

4.6. Axiom System and Deductive System

The axiom system as derived in the embedding was already described in section 3.3 and the fundamental meta-rules were mentioned in section 3.4. By construction, most of them can be derived from the abstract semantics using simple, automatically generated proofs.

While the full derivation of the axiom system in the embedding can be found in A.6 and the deductive system of PLM chapter 9 is derived in A.7, in the following, we will focus on some particular axioms, rules and proofs that are challenging to represent in the embedding. This mostly happens due to PLM's statement involving either complex preconditions given in natural language or due to the statement extending over multiple types.

[26] Note, however, that we still need to make sure that the involved *types* are sufficiently intensional as discussed in section 2.5.1.

[27] Note that **nitpick** has specific support for the **specification** command: it ignores the underlying definition using the Hilbert-Epsilon operator, and instead solely considers the given specification, see [12]. In that sense, the underlying definition of a **specification** is commonly treated as part of an inaccessible implementational detail of an abstraction layer, even in the meta-logic HOL itself.

4.6.1. Base Cases of Denoting Terms

One of the axioms we mentioned explicitly as difficult to implement in section 3.3 is the second (in PLM's numbering) quantifier axiom which establishes a set of base cases of denoting terms. Recall the formulation of the axiom in PLM (item (39.2)):

> $\tau\downarrow$, provided τ is a primitive constant, a variable, or a λ-expression in which the initial λ does not bind any variable in any encoding formula subterm.

We implement this axiom by splitting it up into cases. The first and obvious way to split the axiom is to split it into the separate cases listed in the natural language formulation: constants, variables and λ-expressions.

The embedding does not have to distinguish explicitly between constants and variables: both constants and variables are modelled as entities of the same type (*'a AOT-var*) and the distinction between constants and variables is done by declaring the entity as a constant or using it as a variable in the meta-logic. So it suffices to state one case for constants *and* variables (see A.6.19):

$\alpha\downarrow$

α ranges over all expressions of type *'a AOT-var* (see 4.1.2) and therefore ranges over the denoting objects of type *'a*, which immediately validates $\alpha\downarrow$ semantically. Note that the axiom only extends to *primitive* constants, i.e. it does *not* extend to *defined* constants. In our embedding defined constants are modelled as *terms* of a given type, i.e. directly in the base type *'a*, not the type *'a AOT-var*, so the axiom cannot be instantiated to them, as intended.

The remaining case concerns λ-expressions and is more complex to represent. Internally, a λ-expression denotes, just in case that its matrix φ is necessarily equivalent on all denoting objects that share an urelement, or formally (see A.4.269):

$$
\begin{aligned}
&\textit{AOT-model-denotes } (\textit{AOT-lambda } \varphi) = \\
&(\forall\, v\ \kappa\ \kappa'. \\
&\quad \textit{AOT-model-denotes } \kappa \wedge \\
&\quad \textit{AOT-model-denotes } \kappa' \wedge \textit{AOT-model-term-equiv } \kappa\ \kappa' \longrightarrow \\
&\quad [v \models \varphi\ \kappa] = [v \models \varphi\ \kappa'])
\end{aligned}
$$

However, this is a semantic criterion and does not directly correspond to the formulation of above axiom. While, for arbitrary complex terms, we cannot directly capture the syntactic restriction stating that the initial λ does not bind any variable in any encoding formula subterm, we can construct a set of introduction rules for a predicate on matrices that will cover all terms that match the natural language description.

To that end, we define the auxiliary constant *AOT-instance-of-cqt-2* (see A.4.1283). This constant acts on matrices of λ-expressions, i.e. on functions that map entities of a type of class *AOT-κs* (recall that this may either be an unary individual or a tuple of individuals, see 4.1.6) to propositions.

AOT-instance-of-cqt-2 is true for any such function that agrees on arguments that denote and are *AOT-model-term-equiv*-equivalent, i.e. that has identical values for arguments that denote and share the same urelements. By construction of λ-expressions the use of any such function as matrix of a λ-expression will result in a denoting relation term.

Now we enrich the abstraction layer with several introduction rules for *AOT-instance-of-cqt-2*:

- Functions that do not depend on their argument correspond to matrices in which the λ-bound variables do not occur. Therefore such functions trivially fall under the formulation of the axiom (see A.4.1304).
- Exemplification formulas of the form $[\Pi]\kappa_1...\kappa_n$ in which the λ-bound variable does not occur in Π fall under the axiom, if all individual terms κ_i do not contain an occurrence of the λ-bound variable in encoding formula subterms. This is captured in another auxiliary constant *AOT-instance-of-cqt-2-exe-arg* (see A.4.1286) described below.
- Let $\nu_1...\nu_n$ be the variables bound by the initial λ. Then an exemplification formula of the form $[\lambda\mu_1...\mu_n\ \varphi\{\nu_1...\nu_n,\mu_1...\mu_n\}]\kappa_1...\kappa_n$ as matrix falls under the axiom, if (1) all individual terms κ_i fall under the axiom as described below and (2) φ falls under the axiom w.r.t $\nu_1...\nu_n$, i.e. φ does not contain any occurrences of $\nu_1...\nu_n$ in encoding formula subterms, respectively for any $\mu_1...\mu_n$ it holds that $\varphi\{\nu_1...\nu_n,\mu_1...\mu_n\}$ as function on $\nu_1...\nu_n$ satisfies *AOT-instance-of-cqt-2* (see A.4.1431).
- Complex formulas fall under the formulation of the axiom, just in case all its operands fall under the formulation of the axiom. E.g. a negation falls under the axiom, just in case the negated formula falls under the axiom (see A.4.1307).
- Encoding formulas only fall under the axiom, if the λ-bound variables do not occur in them at all. This is already covered in the first case above. However, this may be refined in the future anticipating an upcoming change in PLM as discussed at the end of this section.

The above rules cover all cases except the primary individual terms in exemplification formulas. The additional auxiliary constant *AOT-instance-of-cqt-2-exe-arg* (see A.4.1286) acts on functions taking entities of a type *'a* of class *AOT-κs* to entities of a type *'b* of class *AOT-κs*. *AOT-instance-of-cqt-2-exe-arg* holds for any such function that sends denoting and *AOT-model-term-equiv*-equivalent arguments to again *AOT-model-term-equiv*-equivalent values. By construction, if the application of any such function to the variables $\nu_1...\nu_n$ occurs as primary individual term in an exemplification formula, then the exemplification formula satisfies the meta-logical definition of *AOT-instance-of-cqt-2* (since the result of the exemplification is known to agree on objects with the same urelements).

Similarly to *AOT-instance-of-cqt-2* we add introduction rules for *AOT-instance-of-cqt-2-exe-arg* to the abstraction layer:

- The identity function falls under *AOT-instance-of-cqt-2-exe-arg* (this is the case in which the λ-bound variables themselves occur as primary individual terms in an exemplification formula; see A.4.1366).
- Constant functions fall under *AOT-instance-of-cqt-2-exe-arg* (this is the case in which the λ-bound variables do *not* occur in a primary individual term of an exemplification formula; see A.4.1371).

- Definite descriptions fall under *AOT-instance-of-cqt-2-exe-arg* just in case their matrix (as function acting on the λ-bound variables) falls under *AOT-instance-of-cqt-2*, i.e. a description may occur in a primary term of an exemplification formula, if its matrix does not contain the λ-bound variables in an encoding formula subterm (see A.4.1392).
- There are further technical introduction rules due to the implementation of n-ary relations as relations acting on tuples (see A.4.1376), e.g. the *fst* and *snd* projections fall under *AOT-instance-of-cqt-2-exe-arg* (i.e. $[\lambda xy\ [F]x]$ and $[\lambda xy\ [F]y]$) and the application of the *Pair* function to two terms falls under the axiom, if both terms fall under *AOT-instance-of-cqt-2-exe-arg* (i.e. $[\lambda x\ [F]\kappa\kappa']$ falls under the axiom, if neither κ nor κ' contain x in an encoding formula subterm).

While the details of this construction are complex, the result is a set of introduction rules that allow proving *AOT-instance-of-cqt-2* exactly for those matrices that fall under the natural language condition of the axiom. The axiom itself is then implemented conditionally: a λ-expression denotes axiomatically, if its matrix satisfies *AOT-instance-of-cqt-2* (see A.6.21). The introduced introduction rules may be used in the abstraction layer, while it is inadmissible to unfold the definition of *AOT-instance-of-cqt-2* itself (i.e. the only matrices for which *AOT-instance-of-cqt-2* is derivable in the abstraction layer are exactly those that satisfy the natural language restriction of PLM's axiom).

Note that at the time of writing, a generalization of the axiom is under discussion that would extend it to the following:[28]

> $\tau\downarrow$, provided τ is a primitive constant, a variable, or a λ-expression in which the initial λ does not bind any variable that is a primary term in an encoding formula subterm.

In an encoding formula $\kappa_1...\kappa_n[\Pi]$ only Π as well as κ_1 through κ_n are defined to be primary terms, but no nested term counts as primary term, so this entails strictly more cases than the formulation given above.

In anticipation of this change, this is already validated by the embedding, however, the corresponding introduction rules are not yet added to the abstraction layer to disbar their use for the time being (see A.4.1468).

See 4.8.1 for a discussion of some consequences of this upcoming change.

4.6.2. The Rule of Substitution

Similar to the axiom above, there is also derived rules in PLM that are challenging to reproduce in the embedding. A prominent example is the Rule of Substitution. PLM formulates this rule in item (159) as follows:[29]

[28] The precise formulation in the upcoming next version of PLM may vary slightly in its wording, but is likely to extend over the same amount of cases.

[29] PLM formulates the rule relative to modally-fragile derivations $\vdash$, but further argues that it is equally valid for modally-strict derivations $\vdash_\Box$. Furthermore, it also states a variant in which the precondition is weakened to $\vdash_\Box \varphi \equiv \chi$, which allows to derive $\vdash_\Box \Box(\varphi \equiv \chi)$ by RN.

> If $\vdash \Box(\varphi \equiv \chi)$, then where Γ is any set of formulas and φ' is the result of substituting the formula χ for zero or more occurrences of ψ where the latter is a subformula of φ, $\Gamma \vdash \varphi$ if and only if $\Gamma \vdash \varphi'$.

The notable restriction in this formulation is the proviso that ψ is a *subformula* of φ. Subformulas are defined recursively in PLM item (6) and notably do not entail matrices of descriptions or non-nullary λ-expressions: E.g. the formula φ is *not* a subformula of $[F]\iota x\varphi\{x\}$ or of $[\lambda y\ \varphi\{y\}]x$.

While the inductive base cases for proving the rule can easily be reproduced in the embedding (see A.7.2702), combining the rule to a single statement in Isabelle is challenging. Therefore we instead provide custom-written proving **method**s that allow applying the rule as intended by PLM. This works by internally analyzing the structure of (the ML representation of)[30] the involved formulas in order to choose the appropriate rule that allows to reduce the goal to a substitution in a less complex formulas. In that sense, the proving methods reconstruct the general proof of the rule in PLM by induction on the complexity of the involved formulas at every invocation of the proving method on a concrete formula.

4.6.3. Proofs by Type Distinction

PLM involves proofs that involve a case distinction by type. An example is the theorem that two terms being identical implies that both denote (see A.7.930).

In our embedding, we reproduce this kind of reasoning by introducing a new type class, in this case *AOT-Term-id*, that assumes the statement of the theorem, and then by instantiating this type class to all the types the statement is supposed to apply to. We then augment the type constraints for terms of these types to include the newly defined class.

In a future version of the embedding, we intend to use Standard ML to define a simple outer syntax command (similarly to **AOT-define** discussed in section 4.3) that will hide the complexity of this process and will allow for a more intuitive statement of theorems that are to be proven by type distinction.

4.6.4. Definition of n-ary Relation Identity

Recall the definition of n-ary relation identity of PLM given in section 3.2:

> $\Pi = \Pi' \equiv_{df} \Pi{\downarrow}\ \&\ \Pi'{\downarrow}\ \&\ \forall y_1 \ldots \forall y_{n-1}\ ([\lambda x\ [\Pi]xy_1 \ldots y_{n-1}] = [\lambda x\ [\Pi']xy_1 \ldots y_{n-1}]$ $\&\ [\lambda x\ [\Pi]y_1xy_2 \ldots y_{n-1}] = [\lambda x\ [\Pi']y_1xy_2 \ldots y_{n-1}]\ \&\ \ldots\ \&\ [\lambda x\ [\Pi]y_1 \ldots y_{n-1}x] = [\lambda x$ $[\Pi']y_1 \ldots y_{n-1}x]\ )$

[30] A proving method written in Isabelle/ML can traverse the ML representation of terms and determine structural properties. However, properties determined in this way cannot be used as logical preconditions in inner syntax. They are meta-logical properties that, in general, cannot be represented in the logical layer.

While we can easily represent ellipse notation in terms that are uniform over arities, as e.g. in β-conversion, by choosing a single variable of a type class that can be instantiated to tuples in place of the ellipse list of variables, this definition involves additional conjunctive clauses depending on the arity and is thereby harder to implement.

A solution would be to approximate the statement of the definition by stating it explicitly for finitely many arities.[31] However, the construction using type class instantiations on product types described in section 4.1 also allows us to state the definition generically, albeit that we have to rely on an auxiliary construction in the meta-logic.

The generic version of the definition in our embedding is the following (see A.5.107):

$$\begin{array}{l} \Pi = \Pi' \equiv_{df} \\ \Pi\downarrow \;\&\; \Pi'\downarrow \;\&\\ \forall\, x_1 ... \forall\, x_n (\text{«}\mathit{AOT\text{-}sem\text{-}proj\text{-}id}\ \text{«}x_1 ... x_n\text{»}\ (\lambda \kappa_1\kappa_n.\ \text{«}[\Pi]\kappa_1 ... \kappa_n\text{»}) \\ \qquad\qquad (\lambda \kappa_1\kappa_n.\ \text{«}[\Pi^\urcorner]\kappa_1 ... \kappa_n\text{»})\text{»}) \end{array}$$

The quotation marks «-» allow us to inject meta-logical terms into the custom grammar we introduced for AOT syntax and vice-versa. Here ellipses like $x_1 ... x_n$ are, meta-logically, a single variable x_1x_n restricted to an arbitrary type of the type class *AOT-κs*. The auxiliary constant *AOT-sem-proj-id* is defined in the type class *AOT-RelationProjection* (see A.4.407; recall that this is a subclass of *AOT-κs*). It satisfies an additional restriction on types of the class *AOT-UnaryRelationProjection* (resp. on the concrete type κ; see A.4.416) and has a concrete definition on products:

$$\mathit{AOT\text{-}sem\text{-}proj\text{-}id}\ \kappa\ \varphi\ \psi = \text{«}[\lambda x\ \varphi\{x\}] = [\lambda x\ \psi\{x\}]\text{»}$$

$$\begin{array}{l} \mathit{AOT\text{-}sem\text{-}proj\text{-}id}\ (\kappa_1,\ \kappa_2\kappa_n)\ (\lambda(x,\ y_1y_n).\ \text{«}\varphi\{x,y_1 ... y_n\}\text{»}) \\ \ (\lambda(x,\ y_1y_n).\ \text{«}\psi\{x,y_1 ... y_n\}\text{»}) = \\ \text{«}[\lambda x\ \varphi\{x,\kappa_2 ... \kappa_n\}] = [\lambda x\ \psi\{x,\kappa_2 ... \kappa_n\}]\ \& \\ \ \ \text{«}\mathit{AOT\text{-}sem\text{-}proj\text{-}id}\ \kappa_2\kappa_n\ (\lambda y_1y_n.\ \text{«}\varphi\{\kappa_1,y_1 ... y_n\}\text{»})\ (\lambda y_1y_n.\ \text{«}\psi\{\kappa_1,y_1 ... y_n\}\text{»})\text{»»} \end{array}$$

Note that the outermost identities in these statements are meta-logical identities that thereby allow immediate meta-logical substitution. In the unary case, *AOT-sem-proj-id* reduces to the the identity of the one-place relations given by λ-abstracting the given matrices φ and ψ.

In the product case, it is defined for matrices acting on pairs (of type $'a \times {}'b$) as a conjunction. The first conjunct is the identity of the one-place relations resulting from λ-abstracting x in the applications of the matrices to x and $\kappa_2 ... \kappa_n$. The second conjunct recursively refers to *AOT-sem-proj-id* on type $'b$ acting on $\kappa_2\kappa_n$ (corresponding to $\kappa_2 ... \kappa_n$ in our AOT syntax implementation) and partial applications of the matrices to κ_1.

Now restricting the generic definition to type κ, yields the following instance:

$$\Pi = \Pi' \equiv_{df} \Pi\downarrow \;\&\; \Pi'\downarrow \;\&\; \forall\, x\ \text{«}\mathit{AOT\text{-}sem\text{-}proj\text{-}id}\ \text{«}x\text{»}\ (\lambda\kappa.\ \text{«}[\Pi]\kappa\text{»})\ (\lambda\kappa.\ \text{«}[\Pi^\urcorner]\kappa\text{»})\text{»}$$

Unfolding the definition of *AOT-sem-proj-id* in the unary case, this yields

$$\Pi = \Pi' \equiv_{df} \Pi\downarrow \;\&\; \Pi'\downarrow \;\&\; \forall\, x\ ([\lambda x\ [\Pi]x] = [\lambda x\ [\Pi^\urcorner]x])$$

While this is technically not a definition of AOT, the implied equivalence is a theorem as a consequence of η-conversion.

[31] In fact, for convenience we do this for arities up to four (see A.5.87).

Restricting the definition to type $\kappa \times \kappa$, yields this instance:

$$\Pi = \Pi' \equiv_{df} \Pi\downarrow \,\&\, \Pi'\downarrow \,\&\, \forall\, x \text{ «}\mathit{AOT\text{-}sem\text{-}proj\text{-}id} \text{ «}x\text{» } (\lambda\kappa_1\kappa_2.\ \text{«}[\Pi]\kappa_1...\kappa_2\text{»})\ (\lambda\kappa_1\kappa_2.\ \text{«}[\Pi']\kappa_1...\kappa_2\text{»})\text{»}$$

Now unfolding the definition of *AOT-sem-proj-id* in the product case (i.e. for type $\kappa \times \kappa$) followed by unfolding it for the recursive unary case, yields the proper definition of *2*-ary relation identity:[32]

$$\Pi = \Pi' \equiv_{df} \Pi\downarrow \,\&\, \Pi'\downarrow \,\&\, \forall\, y\ ([\lambda z\ [\Pi]zy] = [\lambda z\ [\Pi']zy] \,\&\, [\lambda z\ [\Pi]yz] = [\lambda z\ [\Pi']yz])$$

Similarly, instantiating to type $\kappa \times \kappa \times \kappa$ yields ternary relation identity, etc.

While this construction yields the technical means to state the definition of n-ary relation identity as well as the axiom of n-ary encoding generically, properly unfolding the meta-logical definitions can be cumbersome in practice.

For that reason we additionally explicitly derive the definition of identity and the axiom of n-ary encoding for arities up to four, which is more than sufficient for the instances currently used in PLM. For n-ary encoding we currently do not formulate a generic version, even though the same mechanism as above can be applied to this case as well.

In the future, we intend to define a convenient *theorem attribute* (see below) that can be used to immediately instantiate n-ary statements of generic form directly to an arbitrary arity n given as argument to the attribute.

Another subtlety in the definition of n-ary relation identity is the fact that two n-ary relations already have to be identical, if all their projections to unary relations using $n-1$ *denoting* individual terms are identical. However, in order to avoid artifactual theorems, we defined relations as functions that also act on *null*-urelements, resp. on tuples that may involve *null*-urelements. The identity of their projections merely implies that the functions representing the n-ary relations in question evaluate to the same propositions for all tuples of $n-1$ urelements that correspond to denoting individuals (i.e. that are not *null*-urelements) and one urelement that may be a *null*-urelement. This is the reason why in section 4.1.7 we required the behaviour of an n-ary relation on *irregular* individual terms (i.e. tuples that involve more than one *null*-urelement) to be completely determined by the behaviour of the relation on *regular* individual terms (i.e. tuples that involve at most one *null*-urelement). This way the identity of all projections of two n-ary relations to unary relations indeed implies their identity as required for validating the definition, while we still avoid the artifactual theorem that $\forall\, x_1...\forall\, x_n\ ([\Pi]x_1...x_n = [\Pi']x_1...x_n) \rightarrow \Pi = \Pi'$.

4.6.5. Auxiliary Theorem Attributes

The embedding defines several auxiliary *theorem attributes* that help in reproducing common reasoning patterns of PLM that would otherwise be subject to technical complications.

[32] Technically, this additionally involves expanding the n-ary quantifier to two unary quantifiers, one of which can be eliminated.

PLM often prefers stating theorems using free object level variables rather than meta-variables (that would range over potentially non-denoting terms) in order to avoid having to specifically state the precondition that the respective terms denote.

However, whenever a term is trivially known to denote from context, PLM may simply instantiate such theorems directly to terms. This is valid, since it is always possible to apply GEN followed by $\forall$-elimination for terms to the theorem. To reproduce this transformation within the embedding the theorem attribute *unvarify* is introduced (see A.7.705), which takes the variable to be generalized as argument and automatically performs the required transformation on the theorem. Similarly, the attribute *unconstrain* (see A.9.224) can be used to transform a theorem formulated with restricted variables to a theorem involving unconstrained variables with the added precondition that they satisfy the respective restriction conditions.

4.7. Meta Theorems

4.7.1. The Collapse of Alphabetic Variants

We already informally stated that the embedding collapses alphabetic variants. In this section we will define more precisely what this means and justify this collapse.

Isabelle internally represents bound variables using de-Bruijn indices (see [16]). We will showcase this mechanism in detail below. As a consequence, terms that are alphabetic variants are meta-logically indistinguishable. To justify representing AOT's bound variables directly using bound variables in Isabelle, we need to show that both (1) AOT's notion of alphabetic variants is equivalent to Isabelle's use of de-Bruijn indices and (2) any rule of AOT is still valid if any assumption or the conclusion are replaced by an alphabetic variant (as a generalization of PLM's existing *Rule of Alphabetic Variants*).[33]

AOT's Alphabetic Variants align with Isabelle's use of de-Bruijn Indices

Internally, Isabelle represents binding notation by function application and abstraction. E.g. if we let Isabelle print the internal ML representation of the term $\forall p\ (p \rightarrow p)$, we arrive at the following:[34]

$\forall p\ (p \rightarrow p)$
Const (AOT-syntax.AOT-forall, $(\mathrm{o} \Rightarrow \mathrm{o}) \Rightarrow \mathrm{o}$*)* \$
Abs (p, o,
Const (AOT-syntax.AOT-imp, $\mathrm{o} \Rightarrow \mathrm{o} \Rightarrow \mathrm{o}$*)* \$ *Bound 0* \$ *Bound 0*)

[33]This includes theorems and axioms by thinking of them as rules with an empty set of assumptions.

[34]Note that we are not merely talking about a representation in the meta-logic HOL, but about the internal ML representation of HOL terms. Technically, we have setup an *antiquotation* that allows us to print a term together with its internal representation.

While a complete discussion of the ML representation of terms goes beyond the scope of this thesis, it suffices to have a rough understanding of the involved syntax. The atomic terms are typed constants, *Const* ([*identifier*], [*type*]), bound variables *Bound* [*de−Bruijn index*] and free variables *Free* ([*identifier*], [*type*]). \$ is a binary operator that signifies function application between terms. *Abs* ([*name*], [*type*], [*term*]) is the abstraction of [*term*] over a bound variable of type [*type*]. Note that while the internal representation retains the name of the bound variable *p*, it has no logical meaning and is merely used e.g. for term printing, while, logically, occurrences of the bound variables are referred to by *Bound* with a de-Bruijn index. An index of zero refers to the innermost abstraction the bound variable is contained in. An index of one refers to the next outer abstraction, e.g.

$\forall\, p\ (p \to \forall\, q\ (q \to p))$

```
Const (AOT-syntax.AOT-forall, (o ⇒ o) ⇒ o) $
 Abs (p, o,
   Const (AOT-syntax.AOT-imp, o ⇒ o ⇒ o) $ Bound 0 $
    (Const (AOT-syntax.AOT-forall, (o ⇒ o) ⇒ o) $
      Abs (q, o,
        Const (AOT-syntax.AOT-imp, o ⇒ o ⇒ o) $ Bound 0 $ Bound 1)))
```

Note that in the inner abstraction *Bound 0* refers to *q*, while *Bound 1* refers to *p*.

Our claim is that two terms or formulas of AOT are alphabetic variants, if and only if their representation using de-Bruijn indices is the same.

PLM defines alphabetic variants as follows (see PLM item (16)): It refers to two occurrences of a variable as *linked*, if both are free or they are bound by the same occurrence of a variable-binding operator. PLM further introduces *BV-notation* for formulas and terms:[35] the BV-notation of a formula φ is $\varphi[\alpha_1, \ldots, \alpha_n]$, where $\alpha_1, \ldots\ \alpha_n$ is the list of all variables that occur bound in φ, including repetitions. Further $\varphi[\beta_1/\alpha_1, \ldots, \beta_n/\alpha_n]$ refers to the result of replacing α_i by β_i in $\varphi[\alpha_1, \ldots, \alpha_n]$. Now φ' is defined to be an *alphabetic variant* of φ just in case for some *n*:

- $\varphi' = \varphi[\beta_1/\alpha_1, \ldots, \beta_n/\alpha_n]$,
- φ' has the same number of bound variable occurrences as φ and so can be written as $\varphi'[\beta_1, \ldots, \beta_n]$, and
- for $1 \leq i, j \leq n$, α_i and α_j are linked in $\varphi[\alpha_1, \ldots, \alpha_n]$ if and only if β_i and β_j are linked in $\varphi'[\beta_1, \ldots\ \beta_n]$.

By definition, each group of *linked* variable occurrences in AOT corresponds to exactly one abstraction in Isabelle's internal representation and all de-Bruijn indexed *Bound* terms that refer to this abstraction. Since changing the variable name of a linking group will not affect the de-Bruijn indices, the de-Bruijn representation of two alphabetic variants is therefore the same. Conversely, changing any index in the de-Bruijn representation translates to breaking a linking group as defined in PLM, thereby terms with different de-Bruijn representation are not alphabetic variants.

[35]In the following we will restrict our discussion to formulas, but the argument applies analogously to terms as well.

Since thereby the formulas and terms that are collapsed in Isabelle's internal representation are exactly the alphabetic variants of AOT, it remains to argue that the collapse is inferentially valid, i.e. AOT allows to freely interchange alphabetic variants in any derivation.

Equivalence of Alphabetic Variants in AOT

Conveniently, PLM itself derives the following *Rule of Alphabetic Variants* (see PLM item (114)):[36]

> $\Gamma \vdash \varphi$ if and only if $\Gamma \vdash \varphi'$, where φ' is any alphabetic variant of φ.

It is straightforward to strengthen this further to the following:

> $\Gamma \vdash \varphi$ if and only if $\Gamma' \vdash \varphi'$, where φ' is any alphabetic variant of φ and Γ' is a set of alphabetic variants of Γ, i.e. for every $\psi \in \Gamma$ there is an alphabetic variant ψ' of ψ, s.t. $\psi' \in \Gamma'$, and vice-versa.

To see that this rule is valid, it suffices to realize that for every $\psi \in \Gamma$ and $\psi' \in \Gamma'$ by the above rule it holds that $\psi \dashv\vdash \psi'$ and hence all premises in Γ are derivable from Γ' and vice-versa. More rigorously, the version with assumptions can be reduced to the version without assumptions by arguing with successive applications of the deduction theorem to eliminate the assumptions, applying the version of the rule without assumptions and then reconstructing the result using modus ponens. This mechanism is shown explicitly in section 4.7.3 for a similar case.

Hence, AOT allows one to freely move from any formula to an alphabetic variant in all theorems and assumptions, justifying the fact that the embedding identifies alphabetic variants.

4.7.2. Free Variable Notation, Substitutability and Bound Variables

As mentioned in chapter 3, PLM allows terms and formulas with arbitrary free variables to be used in place of its meta-variables, except for free variables that are explicitly excluded in natural language. The embedding on the other hand requires one to explicitly mention any variables that are bound at the occurrence of a meta-variable, if they should be allowed to occur in an instance of the meta-variable. This is due to the fact that binders are implemented in the embedding as operators that act on functions. Similarly, the substitution of variables in meta-variables is implemented using function application.

For example, PLM formulates the first quantifier axiom as follows (see PLM item (39.1)):

> $\forall\alpha\,\varphi \;\rightarrow\; (\tau\downarrow\rightarrow\; \varphi^\tau_\alpha)$, provided τ is substitutable for α in φ

Here φ^τ_α is defined in PLM item (14) as recursively replacing all occurrences of α in φ that are not bound *within* φ *itself* with τ.

[36]Note that while PLM states meta-rules using $\vdash$, unless otherwise noted by convention they apply to both $\vdash$ and $\vdash_\Box$. See remark (67) in PLM. We adopt this convention in the following sections.

The precise definition of *being substitutable* can be found in PLM item (15). In particular, it states the following summary:

> τ is substitutable at an occurrence of α in φ or σ just in case every occurrence of any variable β free in τ remains an occurrence that is free when τ is substituted for that occurrence of α in φ or σ.

and further:

> τ is substitutable for α in φ or σ just in case τ is substitutable at every free occurrence of α in φ or σ.

In the embedding, the same axiom is stated as follows:

$$\forall\,\alpha\ \varphi\{\alpha\} \to (\tau\downarrow \to \varphi\{\tau\})$$

Internally, φ is a function acting on terms and both $\varphi\{\alpha\}$, resp. $\varphi\{\tau\}$, are the function application of φ to α, resp. τ. The following is the HOL representation of the formula of the axiom:

$$\textit{AOT-imp}\ (\textit{AOT-forall}\ (\lambda\alpha.\ \varphi\ \alpha))\ (\textit{AOT-imp}\ (\textit{AOT-denotes}\ \tau)\ (\varphi\ \tau))$$

The $\forall$-quantifier is represented as the function application of the constant *AOT-forall* to the meta-logical λ-abstraction of φ applied to the bound variable α. The substitution of τ for α in φ is represented as the function application of φ to τ.

As mentioned in section 4.7.1, internally Isabelle represents bound variables using de-Bruijn indices that uniquely associate any bound variable with its binder, independently of the name of the variable. β-reduction of the function application of an abstraction to a term merely replaces the bound variables referring to the outermost abstracted variable. Thereby, substitutability is implicit in the construction: applying a meta-variable that is represented as a function to different arguments does not affect variables bound by nested binders.

Therefore, strictly speaking, the implementation of the axiom in the embedding is stronger than the axiom stated in PLM. Consider the following instance of the axiom:

$$\forall\,\alpha\ \exists\,\beta\ (\beta = \alpha) \to (\tau\downarrow \to \exists\,\beta\ (\beta = \tau))$$

Here φ is $\exists\,\beta\ \beta = \alpha$. Now in PLM's terms, β itself would not be substitutable for α in φ, since substituting β for α directly would result in β being bound by the existence quantifier. However, Isabelle allows this instantiation and resolves this issue by automatically generating an alphabetic variant of the nested binder. The following is the direct result of instantiating φ to $\exists\,\beta\ \beta = \alpha$ and τ to β in above axiom:

$$\forall\,\alpha\ \exists\,\beta\ (\beta = \alpha) \to (\beta\downarrow \to \exists\,\beta'\ (\beta' = \beta))$$

While this is not a direct instance of the axiom in PLM, we have argued in section 4.7.1 that it is a meta-theorem of AOT that all alphabetic variants are interderivable. Furthermore, for any φ an alphabetic variant can be constructed that makes any τ substitutable for an occurrence of α in φ by replacing all variables bound in φ that occur free in τ by fresh variables.

This signifies one of the main advantages and simultaneously disadvantages of the use of SSEs. While the use of the meta-logical mechanisms to deal with alphabetic variants and binders allows the implementation to forgo a custom implementation of concepts like substitutions and substitutability, this in turn requires a careful meta-theoretical analysis to assure that the resulting implementation remains faithful. However, for practical purposes the advantages outweigh the disadvantages. Not only is a custom implementation of substitutions and alphabetic variants error-prone and cumbersome, since it is at the same time seemingly trivial, but nonetheless implementationally complex, but relying on the meta-logical implementation has also significant advantages for automated reasoning: For example, while by construction Isabelle will see alphabetic variants as identical entities and can freely substitute them, manual substitution, as it would be required for deep embeddings, would require rigorous proofs about recursively defined transformations on the deep syntax representation that can quickly go beyond the limits of the available automation capabilities, even without attempting to prove complex theorems.

4.7.3. Generalizing Free Variables to Schematic Variables

After a theorem is proven in Isabelle, it is implicitly exported to the current theory context in *schematic* form. That means each free variable used in the theorem is implicitly generalized to a *schematic variable* that can be instantiated to any variable or term of the same type. Since the embedding uses distinct types for (denoting) variables and (potentially non-denoting) terms that have the same type in AOT (see 4.6.1), this does *not* mean that any theorem involving AOT variables can be directly instantiated to AOT terms, however, it does mean that all theorems of AOT are implicitly stated using meta-variables ranging over all variable names. As an example the theorem $\forall F\ ([F]x \to [F]x)$ not only implicitly asserts its alphabetic variants, e.g. $\forall G\ ([G]x \to [G]x)$, but can also be directly instantiated for a different free individual variable, e.g. $\forall G\ ([G]y \to [G]y)$. In the notation of AOT this means that we actually state the theorem $\forall G\ ([G]\nu \to [G]\nu)$, where ν ranges over all names for individual variables. While PLM does not derive a meta-rule that matches this principle, it is usually a straightforward consequence of a series of applications of the meta-rule of universal generalization GEN followed by applications of the rule of $\forall$ Elimination for variables. However, to formulate this as a general principle, some care has to be taken and we have to additionally rely on the collapse of alphabetic variants.

We start by stating and proving the trivial case as a rule in AOT's system:

If $\vdash \varphi$, then $\vdash \varphi^{\beta}_{\alpha}$ where β is substitutable for α in φ.

Assume $\vdash \varphi$. Since the derivation of φ does not need any premises, it follows by the rule of universal generalization (GEN) (see section 3.4.1) that $\vdash \forall \alpha\ \varphi$.[37] Since by assumption

[37]Note that we are using PLM's syntactic convention here, i.e. α may occur free in φ, which using our conventions we would usually signify by writing $\varphi\{\alpha\}$.

β is substitutable for α in φ we can immediately conclude by $\forall$Elimination (see A.7.643) that $\vdash \varphi^{\beta}_{\alpha}$.

However, we want to generalize this rule further to a version that allows for premises and does not require the proviso that β is substitutable for α in φ.

To that end the next step is to generalize above rule to include premises:

> If $\Gamma \vdash \varphi$, then $\Gamma^{\beta}_{\alpha} \vdash \varphi^{\beta}_{\alpha}$ where (1) β is substitutable for α in φ and (2) β is substitutable for α in all $\psi \in \Gamma$ and (3) Γ^{β}_{α} is the set of all ψ^{β}_{α} for $\psi \in \Gamma$.

One way to show this is by first eliminating all premises in Γ using the deduction theorem (see section 3.4.1) and then referring to the simpler rule above. The resulting theorem will yield φ^{β}_{α} from Γ^{β}_{α} by successive applications of modus ponens.

In particular, let $\psi_1, \ldots, \psi_n$ be the list of premises in Γ, s.t. $\psi_1, \ldots, \psi_n \vdash \varphi$.[38] By the deduction theorem it follows that $\psi_1, \ldots, \psi_{n-1} \vdash \psi_n \to \varphi$. Continuing to apply the deduction theorem, we end up with $\vdash \psi_1 \to (\psi_2 \to (\ldots \to (\psi_n \to \varphi)\ldots))$. By assumption β is substitutable for α in this theorem, hence by the rule above we can conclude that: $\vdash {\psi_1}^{\beta}_{\alpha} \to ({\psi_2}^{\beta}_{\alpha} \to (\cdots \to ({\psi_n}^{\beta}_{\alpha} \to \varphi^{\beta}_{\alpha})\ldots)$

Since all ${\psi_i}^{\beta}_{\alpha}$ are in Γ^{β}_{α}, it follows that $\Gamma^{\beta}_{\alpha} \vdash \varphi^{\beta}_{\alpha}$ by n applications of modus ponens.

What remains is the proviso that β be substitutable for α in φ and in all $\psi \in \Gamma$. However, note that for every φ and Γ we can choose alphabetic variants φ' and Γ' that replace all bound occurrences of β with a fresh variable γ that does not occur in φ or in any $\psi \in \Gamma$.

In the last section we have seen that $\Gamma \vdash \varphi$, if and only if $\Gamma' \vdash \varphi'$. Since β is trivially substitutable for α in φ' and in all $\psi \in \Gamma'$, it follows by the rule above that ${\Gamma'}^{\beta}_{\alpha} \vdash {\varphi'}^{\beta}_{\alpha}$. Since Isabelle collapses alphabetic variants by eliminating concrete variable names with de-Bruijn indices, this suffices as justification for the schematic generalization of free variables in theorems and rules in the embedding.

To clarify the last argument, consider the following theorem as example:

$\forall x\ ([R]xy \to [R]xy)$

Isabelle will let us instantiate this theorem using z in place of y, i.e. $\forall x\ ([R]xz \to [R]xz)$ is an instance of above theorem.

However, Isabelle will not allow one to *directly* instantiate y to x, since in $\forall x\ ([R]xx \to [R]xx)$ (which also happens to be a theorem, but a distinct one) all occurrences of x are *bound* and while Isabelle allows to instantiate *schematic variables* to free variable, it does not allow instantiating them to bound variables.[39]

But since alphabetic variants are collapsed, the following is *identical* to the original theorem: $\forall z\ ([R]zy \to [R]zy)$

In this formulation of the theorem, there is no a naming conflict and we *can* instantiate y to x to get $\forall z\ ([R]zx \to [R]zx)$.

[38] Note the discussion of derivations in PLM item (59).

[39] To be precise Isabelle *will* in fact allow this kind of instantiation, but only in situations in which it can automatically rename the bound variable on its own, as we do manually in the continuation of the example.

Note that this is still an *instance* of the original theorem, but we just cannot state this instance without simultaneously renaming the bound variable. Even though, internally, the variable names are eliminated, concrete variable names, of course, still make a difference when *parsing* inner syntax.

Given this discussion, the meta-rule derived above together and the justification of the collapse of alphabetic variants, we may conclude that the fact that Isabelle implicitly generalizes free variables to schematic variables remains faithful to the derivational system of AOT.[40]

4.7.4. Trivial Accessibility Relation for the Modal Logic

As already hinted at in section 2.3, choosing a trivial accessibility relation (respectively, equivalently, no accessibility relation at all) is a natural choice for modelling the modal logic of AOT. In fact, it can be shown that if AOT's actuality operator is modelled using a fixed actual world, any accessibility relation used for modelling necessity has to be trivial.

To see this, consider the following simple embedding of a general extensional modal logic with actuality operator, in which the actuality operator is modelled as validity in an arbitrary, but fixed actual world w_0.

consts r :: ‹$w \Rightarrow w \Rightarrow$ *bool*›
consts w_0 :: w
type-synonym o = ‹$w \Rightarrow$ *bool*›
definition *valid* :: ‹o $\Rightarrow$ *bool*› (‹$\models$ -›) **where** ‹*valid* $\equiv \lambda\ \varphi\ .\ \forall\ w\ .\ \varphi\ w$›
definition *impl* :: ‹o $\Rightarrow$ o $\Rightarrow$ o› (**infixl** ‹$\rightarrow$› *8*) **where** ‹*impl* $\equiv \lambda\ \varphi\ \psi\ w\ .\ \varphi\ w \longrightarrow \psi\ w$›
definition *box* :: ‹o $\Rightarrow$ o› (‹$\Box$-› [*50*] *50*) **where** ‹*box* $\equiv \lambda\ \varphi\ w\ .\ \forall\ v\ .\ r\ w\ v \longrightarrow \varphi\ v$›
definition *actual* :: ‹o $\Rightarrow$ o› (‹$\boldsymbol{\mathcal{A}}$-› [*50*] *50*) **where** ‹*actual* $\equiv \lambda\ \varphi\ w\ .\ \varphi\ w_0$›
definition *equiv* :: ‹o $\Rightarrow$ o $\Rightarrow$ o› (**infixl** ‹$\equiv$› *10*) **where** ‹*equiv* $\equiv \lambda\ \varphi\ \psi\ w\ .\ \varphi\ w \longleftrightarrow \psi\ w$›

In this simple system, **sledgehammer** can immediately prove that all semantic possible worlds have to be related by the accessibility relation, given the T axiom and one of AOT's axioms of actuality and necessity:

lemma
  assumes ‹$\bigwedge\ \varphi\ .\ \models (\Box\varphi \rightarrow \varphi)$›
    and ‹$\bigwedge\ \varphi\ .\ \models (\Box\varphi \equiv \boldsymbol{\mathcal{A}}\Box\varphi)$›
   shows ‹$\forall\ x\ y\ .\ r\ x\ y$›
  by (*metis* (*mono-tags*, *opaque-lifting*) *assms equiv-def actual-def box-def impl-def valid-def*)

However, note that this does not mean that a trivial accessibility relation is in fact the only choice in modelling AOT's modal logic. While the S5 axioms imply that the accessibility relation has to be an equivalence relation, we conjecture that it is possible to model an actuality operator by choosing a different actual world for each equivalence class of worlds induced by the accessibility relation.

[40]Note that for free meta-variables the generalization to schematic form is in fact a requirement for being able to instantiate the meta-variables to arbitrary terms as intended by AOT.

However, independently of potential philosophical issues one may raise against presuming (even if only for the purpose of modelling) that, in different modal contexts, different worlds may be *actual*, an additional practical problem arises: in order to additionally satisfy AOT's axiom for rigid definite descriptions, the description operator would need to become world-relative: instead of choosing the unique object that satisfies the matrix of the description in the globally-fixed actual world, the description operator would have to choose the unique object that satisfies the matrix in the respective actual world of the equivalence class of possible worlds in which the formula containing the description is evaluated.

Allowing the denotation of an individual term to vary depending on modal context constitutes a significant complication for the models. Therefore, our current work forgoes further analysis of this way to extend our representation of AOT. However, such an extension of the models may constitute an interesting topic for future research. We conjecture that it is possible to construct models with a different actual world for each equivalence class of worlds, and that doing so could lead to a means to reproduce the strict distinction between modally-strict and modally-fragile theorems in AOT as follows (recall section 3.4.7): while modally-strict theorems could be modelled as being valid in all possible worlds, i.e. across all equivalence classes in the accessibility relation, modally-fragile axioms could be modelled as being valid in a globally-fixed actual world. This way, adding a contingent truth to the modally-fragile derivation system would merely assert that it holds in the globally-fixed actual world, whereas a modally-strict proof of some statement being *actually true* would require that statement to be true in *all* actual worlds. This would constitute a model in which $\vdash \varphi$ would no longer imply $\vdash_\Box \boldsymbol{\mathcal{A}}\varphi$ and, consequently, in which the converse of RN fails (as allowed by PLM), i.e. $\vdash \Box\varphi$ would no longer imply $\vdash_\Box \varphi$ (while the former merely requires φ to be valid in all worlds accessible from the globally-fixed actual world, the latter also requires φ to be true even in worlds inaccessible from the global actual world).

4.7.5. Primitive Inferences of Isabelle/Pure and Derivations of AOT

As mentioned in section 2.4, being able to trust the abstraction layer constructed for AOT relies on verifying that inferences in the meta-logic correspond to valid reasoning in the system of PLM, given that the set of available theorems and rules is suitably restricted.

We implement the rules of AOT as rules in Isabelle's Pure logic. The primitive inferences of Pure are described in section 2.3 of [54].[41] In this section we will in particular argue that the rules in figure 4.2, when applied in our abstraction layer, will correspond to valid reasoning in AOT.[42]

[41] In particular figure 4.2 is presented as figure 2.2 in section 2.3.1 of [54].

[42] As noted below, an exhaustive analysis would also need to consider the richer logic of Isabelle/HOL.

$$\frac{A \in \Theta}{\vdash A}\ (axiom) \qquad \frac{}{A \vdash A}\ (assume)$$

$$\frac{\Gamma \vdash B[x] \quad x \notin \Gamma}{\Gamma \vdash \bigwedge x.\ B[x]}\ (\bigwedge\text{-}intro) \qquad \frac{\Gamma \vdash \bigwedge x.\ B[x]}{\Gamma \vdash B[a]}\ (\bigwedge\text{-}elim)$$

$$\frac{\Gamma \vdash B}{\Gamma - A \vdash A \Longrightarrow B}\ (\Longrightarrow\text{-}intro) \qquad \frac{\Gamma_1 \vdash A \Longrightarrow B \quad \Gamma_2 \vdash A}{\Gamma_1 \cup \Gamma_2 \vdash B}\ (\Longrightarrow\text{-}elim)$$

Figure 4.2.: Primitive inferences of Pure

The meta-logical *axiom* rule corresponds to PLM items (63.1) and (63.3) which state that axioms and theorems of AOT can be used in derivations.

assume corresponds to the special case of PLM item (63.2) given as $\varphi \vdash \varphi$.

$\bigwedge$-*intro* and $\bigwedge$-*elim* align with our implementations of PLM's GEN rule and $\forall$-elimination: Using our notational convention, it is an instance of $\bigwedge$-*intro* that $\Gamma \vdash \varphi\{\alpha\}$ and $\alpha \notin \Gamma$ implies $\Gamma \vdash$ *for arbitrary* α: $\varphi\{\alpha\}$. The latter is the precondition of our GEN rule, i.e. we can derive $\forall\alpha\ \varphi\{\alpha\}$ in AOT. Similarly, the $\bigwedge$-*elim* rule corresponds to the rule given in A.7.643, which states that we can derive $\varphi\{\beta\}$ from $\forall\alpha\ \varphi\{\alpha\}$.

Note, however, that the $\bigwedge$-*intro* and $\bigwedge$-*elim* rules are not restricted to our defined types of object-level variables of AOT. In particular, they can also be applied to meta-variables ranging over terms of AOT. However, applications of $\bigwedge$-*intro* and $\bigwedge$-*elim* to meta-variables exactly corresponds to the fact that PLM allows to instantiate arbitrary terms of a given type in place of its meta-variables.

The $\Longrightarrow$-*intro* and $\Longrightarrow$-*elim* rules correspond to the deduction theorem (PLM item (75)), which states that if $\Gamma, \varphi \vdash \psi$, then $\Gamma \vdash (\varphi \rightarrow \psi)$ and the meta-rule stated in PLM item (63.5) stating that if $\Gamma_1 \vdash \varphi$ and $\Gamma_2 \vdash (\varphi \rightarrow \psi)$, then $\Gamma_1,\Gamma_2 \vdash \psi$.

Furthermore, Pure is equipped with a primitive equality that allows for substituting terms that are meta-logically equal. In general, PLM's identity corresponds to meta-logical equality on denoting terms (see A.4.63) and non-denoting terms in PLM are *not* meta-logically identical (e.g. recall the fact that non-denoting definite descriptions can be assigned distinct *null*-urelements). While in certain corner cases, the embedding may involve artifactual identities (see 4.8), those cannot be derived without explicit appeals to the semantics. For implicit meta-logical identities that occur in alphabetic variants, we argued in section 4.7.1 that the meta-logical equality is consistent with reasoning in PLM.

While we do not claim that this analysis is exhaustive,[43] it nevertheless provides strong evidence for the assumption that reasoning in our abstraction layer is in fact reproducible

[43]For example, while the rules of our target theory are implemented in the format of rules of Isabelle/Pure, the automated proving methods we use (e.g. *metis*, *meson* and *blast*) work relative to the richer logic of Isabelle/HOL (see chapter 2 of [45]) and for a full account the relevant axioms and inferences of Isabelle/HOL would need to be considered as well.

as derivations in the sense of PLM. For the purpose of a seamless exchange of results between our embedding and PLM, this level of assurance has proven sufficient. In our work we have not encountered a proof in our abstraction layer that could not be reproduced in the system of PLM.

Conversely, the fact that we can derive PLM's axioms and rules in the embedding shows that derivations of PLM can be reproduced in the embedding.

An interesting project for future research may be to implement AOT directly as an object logic of Isabelle/Pure. However, instead of being able to rely on the soundness of Isabelle/HOL as semantic backend, this would require a direct axiomatization of AOT in Pure, which means that we would loose the ability to easily judge the consistency of our representation of AOT and of extensions of its axiom system, which is one of the prime objectives of our current project.

4.8. Artifactual Theorems

In general, artifactual theorems can be defined as follows:

Let T be the target theory and M be the theory in which we are building a model $\mathcal{M}$ of T (so that $\mathcal{M}$ is expressible in M). Then an artifactual theorem ϕ of T relative to M and $\mathcal{M}$ is a formula expressible in the language of T that is derivable in M from $\mathcal{M}$ but which is not derivable in T itself. For example, if T is second-order logic with identity (2OL=) and M is ZF+U (Zermelo-Fraenkel set theory with Urelements) and the model $\mathcal{M}$ in ZF+U represents the predicates of T as sets of Urelements in ZF+U, then the claim:

$\phi = \forall x(Fx \equiv Gx) \rightarrow F = G$

becomes derivable in ZF+U from $\mathcal{M}$, even though it is not a theorem of 2OL=. (In this case, ϕ is interpreted in $\mathcal{M}$ as an instance of the axiom of Extensionality of ZF+U.) This particular ϕ is therefore an artifactual theorem of 2OL= relative to ZF+U and the model $\mathcal{M}$ of predicates as sets.

The abstraction layer we define in our embedding aims to disallow artifactual theorems by limiting theoremhood to what can be derived from the representation of the axioms and rules of T in $\mathcal{M}$; thus, appeals to the axioms and rules of M (beyond those that correspond to the axioms and rules of T) are not allowed in the derivations of theorems of T.

We have discussed in section 4.7 that for technical reasons the embedding collapses certain classes of statements (e.g. alphabetic variants), but that this merely extends to statements that are interderivable in AOT itself. As a result we can reasonably assume that well-formed statements of AOT that are provable in the abstraction layer of our embedding also have a derivation in AOT, i.e. only theorems that are derivable from the formal system T itself are derivable from $\mathcal{M}$ using the representation of the axioms and rules of T.

Ideally, the construction of $\mathcal{M}$ is general enough, s.t. even using the full system of axioms and rules of M, no theorem is derivable from $\mathcal{M}$ that does not have a derivation in the formal system of T itself. However, in the case of our embedding, there are still counterexamples.

As a matter of fact, comparing derivability in the abstraction layer of the embedding, respectively in the formal system of PLM itself, with validity in our underlying semantic structure has been the driving force in our collaboration with the authors of AOT.

In particular, whenever a potential artifactual theorem was recognized, this resulted in an analysis of the discrepancy which regularly led to either a further abstraction of the semantics used in the embedding to eliminate the theorem or to an extension of AOT's axiom system itself, in case it turned out that (1) the discrepancy could be resolved by a natural extension of AOT's axiom system, (2) this extension had merit in that it allowed for deriving new interesting theorems in AOT or that it simplified existing derivations and (3) the extension was philosophically justifiable.

An example of a statement that is now a theorem of AOT, but originated as an artifactual theorem of the embedding, is the necessary and sufficient conditions for relations to denote discussed in section 3.8.2. An earlier example is the coexistence axiom discussed in 3.7, the formulation of which was based on a similar principle that was discovered in the analysis of the semantic properties of the embedding at the time.

This process is ongoing and in the remainder of this section we will discuss some examples of remaining artifactual theorems and the current state of their discussion.

4.8.1. Identity of Projections to Indistinguishable Objects

A variant of the fact that there are indistinguishable abstract objects discussed in section 3.8.1 is the fact that for every two-place relation of AOT there are distinct abstract objects, s.t. the projections of the relation to these abstract objects are identical (see A.7.8428 and A.7.8473):

$$\exists\, x\ \exists\, y\ (A!x\ \&\ A!y\ \&\ x \neq y\ \&\ [\lambda z\ [R]zx] = [\lambda z\ [R]zy])$$

$$\exists\, x\ \exists\, y\ (A!x\ \&\ A!y\ \&\ x \neq y\ \&\ [\lambda z\ [R]xz] = [\lambda z\ [R]yz])$$

However, the construction used for the embedding makes the following stronger statements true:

$$\forall\, F\ ([F]a \equiv [F]b) \rightarrow [\lambda x\ [R]xa] = [\lambda x\ [R]xb]$$

$$\forall\, F\ ([F]a \equiv [F]b) \rightarrow [\lambda x\ [R]ax] = [\lambda x\ [R]bx]$$

This is an artifact of modelling relations as proposition-valued functions acting on urelements. Since being indistinguishable, $\forall\, F\ ([F]a \equiv [F]b)$, semantically implies that a and b share the same urelement, the projections are forced to collapse.

However, we already mentioned in section 4.6.1 that it is currently being considered to extend the base cases of denoting λ-expressions. This extension has particular merit in deriving theorems in higher-order object theory. In the second-order fragment it would be a consequence of this change that the following λ-expressions denotes by axiom:

$[\lambda x\ y[\lambda z\ [R]zx]]\downarrow$

Under this assumption, however, the currently artifactual theorems above become proper theorems of AOT, respectively theorems of the abstraction layer of the embedding (see A.13.807 for a proof). By an analogous proof (see A.13.831), even the following becomes derivable (since the extended axiom will also assert that $[\lambda x\ y[\lambda z\ [G]x]]\downarrow$):

$\forall F\ ([F]a \equiv [F]b) \rightarrow \forall G\ ([G]a = [G]b)$

Semantically, this theorem states that whenever two objects share an urelement, then exemplifying any property results in the same proposition for both of them, which further consolidates the derivational system of AOT with the representation of relations as proposition-valued functions acting on urelements.

So while the theorems above are currently artifactual, they are likely to become proper theorems of the next upcoming version of PLM.

4.8.2. Proposition Identity and Identity of Propositional Relations

AOT's definition of proposition identity reduces proposition identity to the identity of unary propositional relations (see A.5.114):

$p = q \equiv_{df} p\downarrow \ \&\ q\downarrow \ \&\ [\lambda x\ p] = [\lambda x\ q]$

However, due to the fact that our semantic specification of exemplification and λ-abstraction (see A.4.125) is polymorphic and simultaneously specifies relations of all arities, it involves the following more general assertion:

$[\lambda \nu_1 ... \nu_n\ \varphi] = [\lambda \nu_1 ... \nu_n\ \psi] \Longrightarrow \varphi = \psi$

It is a consequence of this more general semantic principle that, for example, the following becomes an artifactual theorem:

$[\lambda xy\ p] = [\lambda xy\ q] \equiv p = q$

Even though relations are modelled as proposition-valued functions in the embedding, theoretically, it is possible to allow the λ-expressions in question to map to propositions that are merely necessarily equivalent to p, resp. q, but not identical to them. However, since the definition of proposition identity still needs to be validated, this would require splitting the specification of exemplification and λ-expressions into separate cases for relations on unary individual terms and tuples of individual terms (e.g. using an additional system of type classes), which represents a technical challenge. The details of such a modified construction also depend on more general open questions regarding n-ary relation identity and generalized η-conversion, which we will discuss in the next section.

4.8.3. Corner-Cases of Relation Identity

AOT involves two axiomatic, respectively definitional claims about identity between n-ary ($n \geq 2$) relation terms (besides the identity of alphabetic variants), in particular η-conversion:

$$[\lambda x_1...x_n\ [F]x_1...x_n] = F$$

As well as n-ary relation identity, e.g. for $n = 2$:

$$\Pi = \Pi' \equiv_{df} \Pi{\downarrow}\ \&\ \Pi'{\downarrow}\ \&\ \forall y\ ([\lambda z\ [\Pi]zy] = [\lambda z\ [\Pi']zy]\ \&\ [\lambda z\ [\Pi]yz] = [\lambda z\ [\Pi']yz])$$

However, AOT does not presuppose that nested atomic exemplification formulas in λ-expressions can be arbitrarily contracted to identical relations. For example, none of the following are theorems of AOT:

$$[\lambda xy\ [\lambda z\ [F]zy]x] = [\lambda xy\ [F]xy]$$

$$[\lambda xy\ [\lambda z\ [F]xz]y] = [\lambda xy\ [F]xy]$$

$$[\lambda xy\ [\lambda z\ [F]zy]x] = [\lambda xy\ [\lambda z\ [F]xz]y]$$

The embedding constructs λ-abstraction and exemplification using the **specification** command by postulating that λ-abstraction and exemplification have to exhibit certain properties (e.g. β- and η-conversion) and by then providing a concrete witness that satisfies these properties.

However, the postulated properties given in the specification go beyond the axioms of AOT they ultimately validate. Validating the axioms of AOT for arbitrary n-ary relations in the embedding while maintaining the definition of n-ary relation identity requires, at least in the current construction of the embedding, additional properties for λ-abstraction and exemplification.[44]

While the artifactual theorems above are validated by the provided witness for our specification, it is currently unknown whether the properties postulated in the specification are sufficient to derive them as artifactual theorem. Neither can **nitpick** provide a counterexample for the theorem, nor can **sledgehammer** construct a proof from the specification, so further manual inspection of the situation is required.

Interestingly, in general, AOT refrains from presuming the identity of its intensional entities, even under conditions that would usually be assumed to imply equality. η-conversion is a notable exception to this principle. However, there are also arguments for *rejecting* η-conversion in an hyperintensional context that is meant to accurately represent human thought and language, see e.g. [36].

So independently of the potential artifactual theorem discussed above, it is an interesting philosophical question whether η-conversion should be presumed by axiom at all. Similarly, there are open questions about the definition of identity of n-place relations in AOT and a potential alternative definition using n-ary encoding as discussed in PLM [63] item (37). Curiously, while the current definition of n-ary relation identity reduces the identity of ternary relations to the identity of all their projections to unary relations, the identity of all their projections to two-place relations does not imply the identity of direct projections to unary relations (without a more general contraction principle) and therefore does not imply the identity of the ternary relations.

[44]For example, the property named *AOT-sem-exe-denoting* in A.4.125 is solely used for validating n-ary relation identity.

We expect that a future more extensive analysis of this issue will, similarly to previous artifactual theorems, result in further theoretical insights, ultimately followed by either an enhancement of the formulation of AOT or a refined embedding, in which e.g. the above might provably not be theorems, even outside the abstraction layer.

4.9. Discussion

We have described an implementation of the second-order fragment of AOT in classical higher-order logic by means of an SSE that can accurately reproduce AOT's reasoning in an abstraction layer. While our semantic backend is not provably free of artifactual theorems, this can be explained due to the fact that AOT does not itself presuppose a strong and exhaustive *intended semantics*, relative to which a completeness result is intended and could be achieved. On the contrary, the authors of AOT explicitly try to avoid letting the axioms and deductive system of AOT be *driven by semantics*, but rather aspire to devise a philosophically justifiable formal system that stands independently of a set-theoretic semantics and in which notions like truth values and possible worlds can instead be analysed as objects of the system itself:

> It is important to remember that the formal semantics simply provides a set-theoretic framework in which models of the metaphysical theory may be constructed. The models serve the heuristic purpose of helping us to visualize or picture the theory in a rigorous way. It is extremely important not to confuse the models of the theory with the world itself. [...] So the goal of our enterprise is not to build a model, but rather to construct a formal theory that correctly mirrors the structure the world may have and, as a result, correctly reflects the entailments among the data.[45]

Nevertheless, our semantic analysis could significantly contribute especially to the theoretical understanding of the conditions, in AOT, under which relations exist. These existence questions require rigorous philosophical consideration and can have a profound impact on the axiom system (recall e.g. 3.7).

While there are open questions e.g. concerning the identity of n-ary relation terms, we anticipate these questions to be the subject of future debate that will, similar to past examples of similar discussions, result in both theoretical insights and an improved implementation.

Given our discussion of the general system of AOT in the previous chapter and its implementation in our embedding in this chapter, we are now suitably equipped to discuss our implementation of PLM's construction of natural numbers, including the extended model construction required for validating its additional axioms.

[45] Edward Zalta in [59] section 2.4.

5. Natural Numbers in AOT

While AOT can represent mathematical theories themselves as *abstract objects* (see chapter 6), it distinguishes this analysis of *Theoretical Mathematics* from the notion of *Natural Mathematics*. *Natural Mathematics* consists of ordinary, pretheoretic claims about mathematical objects and arises directly as abstraction of exemplification patterns rather than being based on the axioms of some mathematical theory (see item (304) in PLM[1]).

Following this idea, Uri Nodelman's and Edward Zalta's claim in PLM chapter 14 is that natural numbers can be naturally defined within object theory and the laws they abide by up to and including Second-Order Peano Arithmetic can be derived without having to appeal to any intrinsically mathematical axioms or notions.

We have reproduced parts of this construction in our implementation[2] and arrived at the following results:

- The construction of natural numbers is sound and the Dedekind-Peano postulates, including mathematical induction, are consistently derivable.
- We could model the additional axioms required for the construction in our framework.
- We could generalize one of the aforementioned axioms, strengthening the theoretical basis and justification of the construction.
- We can analyze variations of the construction that may be adopted in the future.

In particular, we can derive the Dedekind-Peano postulates about Natural Numbers as follows:

1. Zero is a natural number.
2. No natural number has Zero as its successor.
3. If a natural number k succeeds the numbers n and m, then $n = m$.
4. Every natural number has a Successor.
5. Mathematical Induction: If (1) Zero exemplifies a property F and (2) for any number n, it follows from the fact that n exemplifies F that the successor of n exemplifies F, then F is exemplified by all natural numbers.

[1] As in the previous chapters, we refer to at the time of writing most recent version of PLM, dated October 13 2021, which will continue to be available at [63].

[2] At the time of writing the implementation encompasses the construction of natural numbers and the Dedekind-Peano postulates. We expect a full implementation of the derivation of Second-Order Peano Arithmetic in the foreseeable future.

Furthermore, the contributions to the general evolution of AOT we described in the previous chapters have had repercussions on the details of the construction. We will describe this interaction in more detail in the following sections, while reproducing the construction Nodelman and Zalta present in PLM chapter 14.

5.1. General Idea of the Construction

The strategy for constructing natural numbers in AOT basically follows the idea of Frege's Theorem (see [57]). Frege showed that the Peano axioms can be derived from *Hume's Principle* using Second-Order Logic. Hume's Principle states that the number of Fs is equal to the number of Gs if and only if F and G are *equinumerous*. Two relations are *equinumerous*, if and only if there is a one-to-one correspondence between them or, in other words, if and only if there is a bijection between the objects exemplifying F and the objects exemplifying G.

Frege himself derived Hume's Principle from *Basic Law V*, which together with second-order logic leads to Russel's Paradox. However, deriving Peano arithmetic *from* Hume's Principle itself does not require *Basic Law V*. In PLM's chapter 14, Nodelman and Zalta propose a definition of *equinumerosity* and *the number of Fs* in object theory and are able to derive Hume's Principle. Based on that, they present a definition of Natural Numbers and a consistent derivation of the Dedekind-Peano postulates.

5.2. Equinumerosity of Relations

On the basis of traditional mathematical training based on set theory and functional logic, the seemingly most natural conception of *equinumerosity* involves the notion of a bijection. Two properties are equinumerous (i.e., intuitively, they are exemplified by "the same number" of objects), if and only if there is a bijection between the sets of objects exemplifying them.

However, this conception of equinumerosity relies on objects of theoretical mathematics and their axiomatization (sets, functions, bijections). While object theory can in fact define those notions as well, it takes relations to be the more primitive, fundamental concept and thereby prefers a definition in terms of relations alone.

The concept of there being a bijection between the sets of objects exemplifying two properties can equivalently be captured by the notion of there being a *one-to-one correspondence* between the properties.

5.2.1. One-to-One Correspondences

A *one-to-one correspondence* between the properties F and G is a relation R, s.t. (1) for every object x that exemplifies F, there is a unique object y exemplifying G, s.t. x bears

R to y and conversely (2) for every object y that exemplifies G, there is a unique object x exemplifying F, s.t. x bears R to y. Formally (see A.12.12):[3]

$$R\ |\!:\ F\ {}_{1-1}\!\!\longleftrightarrow\ G \equiv$$
$$\forall x\ ([F]x \to \exists!y(([G]y\ \&\ [R]xy)))\ \&\ \forall y\ ([G]y \to \exists!x(([F]x\ \&\ [R]xy)))$$

The relation to a bijection is readily apparent: for any object exemplified by F, the relation R identifies a unique object exemplified by G and vice-versa.

However, this unrestricted notion of a one-to-one correspondence is not well-suited for a definition of equinumerosity that validates Hume's principle in AOT. The intuitive reason for this is that abstract objects cannot be counted. In particular, recall that there are distinct, but exemplification-indistinguishable abstract objects (see section 3.8.1 and A.7.8572):

$$\exists x\ \exists y\ (A!x\ \&\ A!y\ \&\ x \neq y\ \&\ \forall F\ ([F]x \equiv [F]y))$$

Based on this fact, we can prove that there is no one-to-one correspondence between $A!$ and itself:

AOT-theorem ‹$\neg\exists R\ R\ |\!:\ A!\ {}_{1-1}\!\!\longleftrightarrow\ A!$›
proof(*rule raa−cor:2*) — Proof by contradiction.
  AOT-assume ‹$\exists R\ R\ |\!:\ A!\ {}_{1-1}\!\!\longleftrightarrow\ A!$› — Assume the contrary.
  then AOT-obtain R **where** *0*: ‹$R\ |\!:\ A!\ {}_{1-1}\!\!\longleftrightarrow\ A!$› — Let R be a witness.
    using $\exists E$ **by** *metis*
  — By definition of a one-to-one correspondence it follows that:
  AOT-hence ‹$\forall x\ ([A!]x \to \exists!y\ ([A!]y\ \&\ [R]xy))$›
    using *1−1−cor*[*THEN* $\equiv_{df}E$] *&E* **by** *blast*
  — Now let a and b be witnesses to the theorem cited above.
  moreover AOT-obtain a b **where** *1*: ‹$A!a\ \&\ A!b\ \&\ a \neq b\ \&\ \forall F([F]a \equiv [F]b)$›
    using *aclassical2* $\exists E$ **by** *blast*
  — Taken together, this means there has to be a unique abstract object to which a bears R.
  ultimately AOT-have ‹$\exists!y\ ([A!]y\ \&\ [R]ay)$›
    using $\forall E(2)$ *&E* $\to E$ **by** *blast*
  — Now let c be a witness, s.t. c is abstract and a bears R to c.
  then AOT-obtain c **where** *2*: ‹$A!c\ \&\ [R]ac$›
    using *&E(1)* $\exists E$ *uniqueness:1*[*THEN* $\equiv_{df}E$] **by** *blast*
  — By beta-conversion it follows that a exemplifies *being an* x *that bears* R *to* c.
  AOT-hence ‹$[\lambda x\ [R]xc]a$›
    by (*auto intro*!: $\beta{\leftarrow}C$ *cqt:2 dest*: *&E*)
  — Since by construction a and b exemplify the same properties, the same holds true for b.
  AOT-hence ‹$[\lambda x\ [R]xc]b$›
    by (*safe intro*!: *1*[*THEN &E(2)*, *THEN* $\forall E(1)$, *THEN* $\equiv E(1)$]) *cqt:2*[*lambda*]
  — Again by beta conversion it follows that b bears R to c.
  AOT-hence *5*: ‹$[R]bc$›
    using $\beta{\to}C$ **by** *blast*

[3] Note that as mentioned in section 3.4.2, instead of stating the original definitions-by-equivalence of AOT that involve additional significance clauses, we may instead illustrate the definitions in simpler form using derived equivalences formulated using object-level variables throughout this chapter. In each case the full definition in the appendix is referenced.

— Now the following is a consequence of the assumption that $A!$ is in one-to-one correspondence to itself:

AOT-have ‹$\forall x \, \forall y \, \forall z \, ([A!]x \;\&\; [A!]y \;\&\; [A!]z \rightarrow ([R]xz \;\&\; [R]yz \rightarrow x = y))$›
using *eq−1−1*[*unvarify F G*, *OF oa−exist*:*2*, *OF oa−exist*:*2*, *THEN* $\equiv$*E*(*1*),
THEN fFG:*4*[*THEN* $\equiv_{df}$*E*], *THEN* &*E*(*1*),
THEN fFG:*2*[*THEN* $\equiv_{df}$*E*], *THEN* &*E*(*2*), *OF 0*]**.**

— In particular this is true for a, b and c.

AOT-hence ‹$[A!]a \;\&\; [A!]b \;\&\; [A!]c \rightarrow ([R]ac \;\&\; [R]bc \rightarrow a = b)$›
using $\forall$*E*(*2*) **by** *blast*

— But we already established that a, b and c are abstract, as well as that a bears R to c and b bears R to c, so a and b have to be identical.

AOT-hence ‹$a = b$›
using *1*[*THEN* &*E*(*1*)] *2 5* &*E* $\rightarrow$*E* &*I* **by** *metis*

— Which contradicts a and b being distinct by construction.

AOT-thus ‹$p \;\&\; \neg p$› **for** p
using *1 =−infix* $\equiv_{df}$*E* &*I* &*E raa−cor*:*1* **by** *fast*

qed

So if *equinumerosity* was defined in terms of the existence of a full one-to-one correspondence, $A!$ would not be equinumerous to itself and consequently equinumerosity would not be an equivalence relation. However, Frege's construction assumes that equinumerosity partitions all properties into equivalence classes, i.e. that equinumerosity *is* an equivalence relation. While it is an interesting question for future research, whether a variant of the construction was possible, in which equinumerosity merely was a partial equivalence relation (and consequently not all properties could be counted, resp. *the number of Fs* would not denote for every F), the construction in the current version of PLM at the time of writing chooses to stay closer to Frege's original method. In particular, Nodelman and Zalta restrict their analysis to *ordinary objects* and we will therefore choose this option for the main discussion in this chapter.

In section 5.21 we will discuss alternative options to address the issue that may lead to an enhanced version of the construction in the future.

5.2.2. One-to-One Correspondences on the Ordinary Objects

As mentioned in the introduction of this chapter, natural mathematics arises from abstracting exemplification patterns. In case of natural numbers, those patterns in particular need to be among objects that can be counted. While abstract objects in general cannot, ordinary objects can always naturally be counted. Hence Nodelman and Zalta, following [60], introduce the notion of one-to-one correspondences *among the ordinary objects*. To that end, they introduce the restricted variables u, v, r, s that range over only the ordinary objects.[4] Using these restricted variables, a one-to-one correspondence$_E$ among the ordinary objects can be defined in the same way as a full one-to-one correspondence (see A.12.182):

[4]Recall the discussion of restricted variables in section 3.4.4.

$$R \mid: F \;_{1-1}{\longleftrightarrow}_E\; G \equiv$$
$$\forall\, u\ ([F]u \rightarrow \exists!v(([G]v\ \&\ [R]uv)))\ \&\ \forall\, v\ ([G]v \rightarrow \exists!u(([F]u\ \&\ [R]uv)))$$

5.2.3. Definition of Equinumerosity

Based on one-to-one correspondences$_E$ on the ordinary objects, *equinumerosity* on the ordinary objects can be defined as suggested above: two relations are equinumerous$_E$, if and only if there is a one-to-one correspondence$_E$ on the ordinary objects between them (see A.12.187):[5]

$$F \approx_E G \equiv_{df} \exists\, R\ R \mid: F \;_{1-1}{\longleftrightarrow}_E\; G$$

Equinumerosity on the ordinary objects is indeed an equivalence relation (see A.12.211):

$$F \approx_E F$$
$$F \approx_E G \rightarrow G \approx_E F$$
$$F \approx_E G\ \&\ G \approx_E H \rightarrow F \approx_E H$$

Reflexivity can be shown by using the identity on the ordinary objects ($=_E$) (see 3.4.5) as witness for the existence of a one-to-one-correspondence$_E$ between any property and itself. Note that this is only possible, since, in contrast to general identity, identity on the ordinary objects constitutes a (denoting) relation.

Symmetry is a simple consequence of the symmetry of the definition of one-to-one correspondences$_E$.

Transitivity requires a slightly more verbose proof (see A.12.276), that hinges on the fact that $[\lambda xy\ O!x\ \&\ O!y\ \&\ \exists\, v\ ([G]v\ \&\ [R_1]xv\ \&\ [R_2]vy)]$ can be chosen as a witness for the existence of a one-to-one-correspondence$_E$ between F and H, if R_1 is a one-to-one-correspondence$_E$ between F and G and R_2 is a one-to-one-correspondence$_E$ between G and H.

5.2.4. Properties of Equinumerosity

Nodelman and Zalta continue to derive a variety of properties of equinumerosity that are helpful for the remainder of the construction. While a full account of the progression of theorems can be found in PLM, respectively in our implementation in A.12, the following is a selection of noteworthy auxiliary theorems:

Properties that are unexemplified on the ordinary objects are equinumerous (any relation may serve as one-to-one-correspondence$_E$ between them; see A.12.712):

$$\neg\exists\, u\ [F]u\ \&\ \neg\exists\, v\ [H]v \rightarrow F \approx_E H$$

A property F that is exemplified by some ordinary object is *not* equinumerous to a property H that is not exemplified by any ordinary object (proof by contradiction, since

[5] In the following sections we will drop the explicit mention of the restriction to the ordinary objects and simply talk about *equinumerosity* and being *equinumerous* instead of *equinumerosity on the ordinary objects*, resp. *equinumerous*$_E$.

the existence of a one-to-one correspondence$_E$ would imply that H is exemplified by an ordinary object; see A.12.737):

$$\exists\, u\ [F]u\ \&\ \neg\exists\, v\ [H]v \rightarrow \neg F \approx_E H$$

Respectively, removing (adding) ordinary objects from (to) a pair of equinumerous properties yields equinumerous properties (see A.12.772):[6]

$$F \approx_E G\ \&\ [F]u\ \&\ [G]v \rightarrow F^{-u} \approx_E G^{-v}$$
$$F^{-u} \approx_E G^{-v}\ \&\ [F]u\ \&\ [G]v \rightarrow F \approx_E G$$

Properties that are equivalent on the ordinary objects (written as $\equiv_E$) are equinumerous (see A.10.74, A.12.1755):[7]

$$F \equiv_E G \equiv \forall\, u\ ([F]u \equiv [G]u)$$
$$F \equiv_E G \rightarrow F \approx_E G$$

5.2.5. Modal Properties of Equinumerosity

It is noteworthy that, in general, equinumerosity is *not* modally rigid. In particular, it is provable that there are relations that are possibly equinumerous and possibly not equinumerous (see A.12.1514):

$$\exists\, F\ \exists\, G\ \Diamond(F \approx_E G\ \&\ \Diamond\neg F \approx_E G)$$

As a simple example consider a property that is necessarily unexemplified and another property that is *actually* unexemplified, but *possibly* exemplified by some ordinary object.[8] While such properties are equinumerous in the actual world, there is no one-to-one-correspondence$_E$ between them in the possible world, in which the second property is exemplified by an object.

We will see in the next section that for this reason it makes sense to use the actual world as a reference for the definition of *numbering properties*.

In any modal context, it is possible to express equinumerosity relative to the behaviour of properties in the actual world. In particular the following is a (modally-strict) theorem (see A.12.1896):

$$F \approx_E G \equiv \forall\, H\ ([\lambda z\, \boldsymbol{\mathcal{A}}[H]z] \approx_E F \equiv [\lambda z\, \boldsymbol{\mathcal{A}}[H]z] \approx_E G)$$

I.e. two properties F and G are equinumerous, if and only if for all properties H, F is equinumerous to *actually exemplifying* H just in case that G is. Furthermore, for *rigid* properties,[9] equinumerosity is modally collapsed (see A.12.1951):

$$\mathit{Rigid}(F)\ \&\ \mathit{Rigid}(G) \rightarrow \Box(F \approx_E G \rightarrow \Box F \approx_E G)$$

[6]The statements rely on the following definition of F^{-x}, i.e. *being an F that is not x*: $F^{-x} =_{df} [\lambda z\ [F]z\ \&\ z \neq_E x]$. The proofs rely on constructing suitable one-to-one correspondences$_E$ by case analysis.

[7]The identity on the ordinary objects ($=_E$) can be chosen as witness for the existence of a one-to-one-correspondence$_E$.

[8]The existence of such properties is guaranteed by the fact that by axiom there is an object that is not actually, but possibly concrete as mentioned in section 3.3.

[9]I.e. properties that are modally collapsed: $\mathit{Rigid}(F) \equiv \Box\forall\, x\ ([F]x \rightarrow \Box[F]x)$, see also 3.8.3.

The proofs of the last two theorems hinge on the existence of *rigidifying* relations. Recall the earlier discussion of this topic in section 3.8.3 - notably, in earlier versions of PLM, the existence of rigidifying relations had to be ensured by axiom. In the current formulation of AOT, the necessary and sufficient conditions for relations to denote that we contributed to the theory (see section 3.8.2), can be used to prove the existence of *world-indexed properties* that can serve as witnesses for the existence of rigidifying relations, thereby eliminating the need for the additional axiom.

5.3. The Number of Fs and Hume's Theorem

To state Hume's Theorem, additionally to the definition of *equinumerosity* above, a definition of *The Number of Fs* (written as $\#F$) is required. To that end Nodelman and Zalta first define what it means for an object to number a property as follows (see A.12.2177):

$$\mathit{Numbers}(x,G) \equiv A!x \;\&\; \forall F\ (x[F] \equiv [\lambda z\ \boldsymbol{\mathcal{A}}[F]z] \approx_E G)$$

An abstract object x numbers a property G, if it encodes exactly those properties, such that *actually exemplifying* them is equinumerous to G. An alternative choice would be to forgo the actuality operator and merely require that x encodes exactly those properties that are themselves equinumerous to G.[10] However, we noted in the last section that equinumerosity is (in general) not modally rigid, so such a definition would have the undesirable consequence that numbering properties would depend on modal context and consequently that every possible world would need its own sets of numbers (see 5.5). To avoid this issue the actual world is used as a reference. Later in this chapter, we will show that this does *not* mean that it is impossible to count in non-actual worlds and that this definition is consistent with the pretheoretic intuition of one group of natural numbers that can count objects at any possible world (see 5.5).

Now *The Number of Gs* can simply be defined as *the* object that numbers G (see A.12.2503):

$$\#G =_{df} \iota x \mathit{Numbers}(x,G)$$

Using these definitions Hume's principle becomes derivable as a theorem (see A.12.2589):

$$\vdash \#F = \#G \equiv F \approx_E G$$

Note that, due to the fact that AOT's definite descriptions are modally rigid and refer to objects in the actual world, this theorem is not modally strict.[11] However, the following variants are necessary facts with modally-strict proofs (see A.12.2602):

$$\exists x\ (\mathit{Numbers}(x,F) \;\&\; \mathit{Numbers}(x,G)) \equiv F \approx_E G$$

$$\begin{aligned}&\exists x\ \exists y\ (\mathit{Numbers}(x,F) \;\&\; \forall z\ (\mathit{Numbers}(z,F) \to z = x) \;\&\; \mathit{Numbers}(y,G) \;\&\\ &\qquad \forall z\ (\mathit{Numbers}(z,G) \to z = y) \;\&\\ &\qquad x = y) \equiv\\ &F \approx_E G\end{aligned}$$

[10] In fact, earlier versions of the construction used this definition (see e.g. [60]).

[11] Recall that this is signified by the turnstile symbol $\vdash$ and recall the discussion in section 3.4.7.

Note that the last theorem corresponds to a translation of the descriptions in Hume's theorem according to Russell's analysis of definite descriptions.

5.4. The Number Zero

Given the fact that we defined numbers by means of the properties they number, which in turn is - informally speaking - based on how many objects those properties exemplify, a natural definition of the number Zero arises. The number Zero is the object that numbers the empty property, to be more precise the number of *being a non-self-identical ordinary object* (see A.12.2683).[12]

$$0 =_{df} \#[\lambda x\ O!x\ \&\ x \neq_E x]$$

Note that while the above definition introduces the number Zero as (abstract) object, we have not defined the notion of a *Natural Number* yet, nor shown that the number Zero indeed *is* a natural number. The definition of *Natural Number* will rely on introducing a *predecessor* relation and, intuitively speaking, defining that an abstract object is a natural number if there is a series of objects starting at Zero, ending at the given abstract object, s.t. two consecutive objects in that series bear the predecessor relation to each other. While we will describe this construction in detail in the following sections, we can already define the strictly more general[13] notion of a *Natural Cardinal* and it will immediately follow that Zero is a natural cardinal. An object x is a natural cardinal, just in case that there is a property G, s.t. x is the number of Gs (see A.12.2570):

$$NaturalCardinal(x) \equiv_{df} \exists\, G\ x = \#G$$

By the definition of the number Zero, it becomes immediately apparent that Zero is a natural cardinal (see A.12.2688):[14]

$$NaturalCardinal(0)$$

[12]To be precise being a non-self-identical$_E$ object (see section 3.4.5). This distinction is non-trivial: While $O!x\ \&\ x \neq_E x \equiv O!x\ \&\ x \neq x$ is a theorem, due to the hyperintensionality of object theory, it does not have to be the case that $[\lambda x\ O!x\ \&\ x \neq_E x]$ and $[\lambda x\ O!x\ \&\ x \neq x]$ are the same property (as a matter of fact it is not even asserted *a priori* that the latter even denotes a property at all). So $\#[\lambda x\ O!x\ \&\ x \neq_E x]$ and $\#[\lambda x\ O!x\ \&\ x \neq x]$ are not the same object *a priori*, even though it is a theorem that they are identical. But this theorem has to appeal to the fact that both properties are equinumerous and to Hume's Theorem. Further examples of terms denoting the number Zero are $\#[\lambda x\ x \neq x]$ and $\#[\lambda x\ \exists\, p\ (p\ \&\ \neg p)]$.

[13]It is a theorem that $\#O!$ is a natural cardinal that is infinite and not a natural number (see A.12.5456).

[14]However, note that the proof has to appeal to the fact that $\#G{\downarrow}$ (see A.12.2505) as well as the fact that $[\lambda x\ O!x\ \&\ x \neq_E x]{\downarrow}$ by axiom.

5.5. Counting in Possible Worlds

In section 5.3, we mentioned the use of the actual world as reference for defining numbering properties and hinted at the fact that this is justified and consistent with pretheoretic intuition. We can now discuss this in more detail at the example of the number Zero.

The number of a property is defined as rigid definite description and thereby uses the actual world as frame of reference. In particular, using the number Zero as example, this means the following (see A.12.2918):

$$\neg\exists\, u\ \boldsymbol{\mathcal{A}}[F]u \equiv \#F = \mathit{0}$$

If and only if a property F is not *actually* exemplified by any ordinary object, the number of that property is Zero. This may seem counter-intuitive at first: the stated theorem is modally-strict and thereby a *necessary* fact. So in any possible world, even in worlds in which F could be exemplified, the number of F is still Zero, if F is not *actually* exemplified. However, this is merely due to the fact that definite descriptions are rigid and themselves refer to objects in the actual world.

Moving away from the rigidly defined *number of Fs*, it is a modally-strict theorem (and thereby a *necessary* fact), that Zero *numbers* any property that is not exemplified by any ordinary object (see A.12.2844):

$$\neg\exists\, u\ [F]u \equiv \mathit{Numbers}(\mathit{0},F)$$
$$\Box(\neg\exists\, u\ [F]u \equiv \mathit{Numbers}(\mathit{0},F))$$

I.e. Zero numbers empty properties in all possible worlds. A different take on this is the fact that any object that *possibly* numbers a necessarily empty property is the number Zero (see A.13.5):

$$\Diamond\mathit{Numbers}(x,[\lambda z\ O!z\ \&\ z \neq_E z]) \rightarrow x = \mathit{0}$$

By contrast, if numbering a property had been defined without using the actual world as reference, then "the" number Zero would be a different abstract object in different possible worlds:

If we define $\mathit{Numbers}'$ without the use of the actuality operator, s.t.:

$$\mathit{Numbers}'(x,G) \equiv A!x\ \&\ \forall\, F\ (x[F] \equiv F \approx_E G)$$

Then it is a theorem (see A.13.46) that:

$$\exists\, x\ \exists\, y\ (\Diamond\mathit{Numbers}'(x,[\lambda z\ O!z\ \&\ z \neq_E z])\ \&\ \Diamond\mathit{Numbers}'(y,[\lambda z\ O!z\ \&\ z \neq_E z])\ \&\ x \neq y)$$

I.e. there would be distinct abstract objects that might count necessarily empty properties. This is clearly contrary to the pretheoretic intuition that numbers are universal, i.e. that counting empty properties in any world will yield one and the same number Zero.

On the other hand, we can further consolidate the use of the actual world as reference frame, by realizing that we *can* talk about the numbers of properties in different worlds, despite the rigidity of definite descriptions and the use of the actual world as reference for defining numbers. We again use the number Zero as example and can show that if and only if a property F is not exemplified in a given possible world w, then *the number of exemplifying F at w* is Zero (see A.12.2987):[15]

$$w \models \neg\exists\, u\ [F]u \equiv \#F_w = 0$$

5.6. Ancestral Relations and Transitive Closures

As mentioned above, natural numbers will, informally speaking, be defined by the means of series of objects that bear a (yet to be introduced) predecessor relation to each other. However, traditionally, a series of objects relies on it being possible to index its objects using a continuous sequence of natural numbers. Since our goal is to *define* natural numbers, using this traditional notion of a series is not an option. Instead we construct *ancestral relations*. In particular the *strong ancestral* of a relation will match the concept of the transitive closure of the relation. Natural numbers will be defined as the objects to which the number Zero bears the *weak ancestral* (i.e. the transitive and reflexive[16] closure) of the predecessor relation, i.e. the number Zero itself or any object that is transitively preceded by Zero.

The first step in this process is to define being a *hereditary* property with respect to a relation, which will lead to a definition of the *strong ancestral* of a relation.

5.6.1. Properties that are Hereditary with respect to a Relation

A property F is *hereditary* w.r.t. a relation R, if and only if for every pair of objects x and y, s.t. x bears R to y, if x exemplifies F, then y exemplifies F (see A.12.3117):

$$\mathit{Hereditary}(F,R) \equiv \forall\, x\ \forall\, y\ ([R]xy \to ([F]x \to [F]y))$$

Intuitively, a relation R defines sequences of objects as follows: we call a list of objects x_1, ..., x_n an R-induced sequence, if for every $0 < i < n$, x_i bears R to x_{i+1}. Then F being hereditary w.r.t. R means that any R-induced sequence starting in F (i.e. starting with an object exemplified by F), is completely contained in F (i.e. every object in the sequence exemplifies F as well).

Or in other words, a property F is hereditary w.r.t a relation R, if "F-ness" is inherited from all objects that exemplify F to the objects that are R-related to them.

5.6.2. Strong Ancestral of a Relation and Transitive Closures

Using the above definition, we can introduce the *Strong Ancestral* of a relation R, which is written as R^* (see A.12.3125):[17]

[15] Recall the discussion of AOT's theory of Possible Worlds and world-indexed properties in section 3.8.3.

[16] We will see that reflexivity will have to be restricted to a specific domain.

[17] Note that while PLM uses R^* for the strong ancestral, i.e. the transitive closure, of R and later R^+ for the weak ancestral, i.e. the transitive and reflexive closure, of R, Isabelle's HOL library uses the opposite convention, i.e. it uses r^+ as transitive and r^* as reflexive-transitive closure.

$R^* =_{df} [\lambda xy\ \forall F\ (\forall z\ ([R]xz \to [F]z)\ \&\ \mathit{Hereditary}(F,R) \to [F]y)]$

An object x bears R^* to y, just in case that y exemplifies every property F that is hereditary w.r.t R and that is exemplified by all objects to which x bears R. To convince ourselves that this definition captures the transitive closure of R, we may fix x and consider a property F, s.t. $\forall z\ ([R]xz \to [F]z)\ \&\ \mathit{Hereditary}(F,R)$. Any such property F is exemplified by all objects immediately following x in an R-induced sequence (first conjunct) as well as all objects in any longer R-induced sequence starting at x (second conjunct). Hence (informally thinking of properties as sets) any such F contains all objects that are transitively related to x. Note, however, that F may exemplify additional objects that are not part of any R-induced sequence. However, the intersection of all such properties F yields the smallest set of objects that are transitively related to x, which is *exactly* those objects that *are* transitively related to x.

It is interesting to note that there is a different way to define the transitive closure of a relation R, namely:

The transitive closure of a relation R is the intersection of all transitive relations R' that are contained in R. As a matter of fact, we can state this definition in AOT as well, and prove it to be equivalent to the strong ancestral of R.

First we define for a relation to be transitive as follows:

AOT-define *Transitive* :: $\langle\tau \Rightarrow \varphi\rangle$ ($\langle\mathit{Transitive}'(\text{-}')\rangle$)
$\langle\mathit{Transitive}(R) \equiv_{df} \forall x \forall y \forall z([R]xy\ \&\ [R]yz \to [R]xz)\rangle$

Next we can define for a relation to be contained in another relation, or alternatively, moving away from set-theoretic concepts, for a relation to entail another relation:

AOT-define *Entails* :: $\langle\tau \Rightarrow \tau \Rightarrow \varphi\rangle$ ($\langle\mathit{Entails}'(\text{-},\text{-}')\rangle$)
$\langle\mathit{Entails}(R,R') \equiv_{df} \forall x \forall y([R]xy \to [R']xy)\rangle$

Being in the intersection of all transitive relations entailed by R means exemplifying all of them. Hence we can define the transitive closure of R as follows:

AOT-define *TransitiveClosure* :: $\langle\tau \Rightarrow \Pi\rangle$ ($\langle\mathit{TransitiveClosure}'(\text{-}')\rangle$)
$\langle\mathit{TransitiveClosure}(R) =_{df} [\lambda xy\ \forall R'(\mathit{Transitive}(R')\ \&\ \mathit{Entails}(R,R') \to [R']xy)]\rangle$

Now we can prove that this definition of a transitive closure is equivalent to the definition of a strong ancestral above:

AOT-theorem $\langle[\mathit{TransitiveClosure}(R)]xy \equiv [R^*]xy\rangle$
proof(*safe intro*!: $\equiv I \to I$)
  AOT-assume $\langle[\mathit{TransitiveClosure}(R)]xy\rangle$
  AOT-hence $\langle[\lambda xy\ \forall R'(\mathit{Transitive}(R')\ \&\ \mathit{Entails}(R,R') \to [R']xy)]xy\rangle$
    by (*auto intro*: *rule−id−df*:*2*:*a*[*OF TransitiveClosure*] *intro*!: *cqt*:*2*)
  AOT-hence $\langle\forall R'(\mathit{Transitive}(R')\ \&\ \mathit{Entails}(R,R') \to [R']xy)\rangle$
    using $\beta{\to}C$ **by** *fast*
  AOT-hence $\langle\mathit{Transitive}(R^*)\ \&\ \mathit{Entails}(R,R^*) \to [R^*]xy\rangle$
    using $\forall E(1)$ *rule−id−df*:*2*:*b*[*OF ances−df*] *hered*:*2* **by** *blast*
 — The following is a consequence of PLM's theorems about strong ancestral relations (see A.12.3136 and A.12.3222).

```
  moreover AOT-have ‹Transitive(R*) & Entails(R,R*)›
    by (auto intro!: &I Entails[THEN ≡df I] Transitive[THEN ≡df I] GEN →I
            simp: anc−her:1[THEN →E] anc−her:6[THEN →E])
  ultimately AOT-show ‹[R*]xy›
    using →E &I by blast
next
  AOT-assume 0: ‹[R*]xy›
  AOT-have ‹∀ R′(Transitive(R′) & Entails(R,R′) → [R′]xy)›
  proof(safe intro!: GEN →I; frule &E(1); drule &E(2))
    fix R′
    AOT-assume transitive: ‹Transitive(R′)› and entails: ‹Entails(R,R′)›
 — The following is an instance of another theorem about strong ancestral relations (see A.12.3148).
    AOT-have ‹[R*]xy & ∀ z([R]xz → [λz [R′]xz]z) & Hereditary([λz [R′]xz],R) → [λz [R′]xz]y›
      by (rule anc−her:2[unvarify F]) cqt:2[lambda]
    moreover AOT-have ‹Hereditary([λz [R′]xz],R)›
    proof (safe intro!: hered:1[THEN ≡df I] &I cqt:2 GEN →I)
      fix z y
      AOT-assume ‹[R]zy› and ‹[λz [R′]xz]z›
      AOT-hence ‹[R′]zy› and ‹[R′]xz›
        using entails by (auto dest: Entails[THEN ≡df E] ∀ E(2) →E β→C)
      AOT-hence ‹[R′]xy›
        using transitive by (auto dest!: Transitive[THEN ≡df E] dest: ∀ E(2) →E intro!: &I)
      AOT-thus ‹[λz [R′]xz]y›
        by (auto intro!: β←C cqt:2)
    qed
    moreover AOT-have ‹∀ z([R]xz → [λz [R′]xz]z)›
      using entails[THEN Entails[THEN ≡df E]]
      by (auto intro!: GEN →I β←C cqt:2 dest: ∀ E(2) →E)
    ultimately AOT-have ‹[λz [R′]xz]y›
      using 0 &I →E by auto
    AOT-thus ‹[R′]xy›
      by (rule β→C)
  qed
  AOT-thus ‹[TransitiveClosure(R)]xy›
    by (auto intro: rule−id−df:2:b[OF TransitiveClosure]
             intro!: β←C cqt:2 tuple-denotes[THEN ≡df I, OF &I])
qed
```

5.7. Weak Ancestral Relations

As mentioned above, our goal is to define being a natural number as either being Zero or being an object, s.t. Zero bears the strong ancestral of the to-be-defined predecessor relation to it. This matches the notion of the *weak ancestral* of the predecessor relation. Traditionally, the weak ancestral of a relation R^+ is defined, s.t. an object x bears R^+ to an object y, if and only if either x bears the strong ancestral R^* to y or $x = y$.

However, recall that in AOT there is no general relation of identity, i.e. $[\lambda xy\ x = y]$ does not denote (see 3.8.1). Consequently, the immediate candidate for defining the weak ancestral of a relation $[\lambda xy\ [R^*]xy \vee x = y]$ does not denote for arbitrary choices of R.[18]

For this reason Nodelman and Zalta proceed by introducing *rigid one-to-one relations*. Rigid one-to-one relations induce a notion of identity on their *domain* that is consistent with general identity (on this domain), but constitutes a denoting relation.

5.7.1. Rigid One-to-One Relations

For a relation to be *one-to-one* is related to the notion of a function being injective. A relation R is *one-to-one*, if whenever two objects x and y bear R to the same object z, then x and y are identical (see A.12.3256):

$$\mathit{1{-}1}(R) \equiv \forall x\ \forall y\ \forall z\ ([R]xz\ \&\ [R]yz \rightarrow x = y)$$

Note, however, that one-to-one relations are more general than injective functions, since the criterion to be one-to-one does not imply that the relation is *functional*, i.e. that each object in its domain is related to exactly one object.

An object x is in the domain of a relation R, just in case that there is an object y, s.t. x bears R to y (see A.12.3322):

$$\mathit{InDomainOf}(x,R) \equiv \exists y\ [R]xy$$

While the predecessor relation will in fact be a functional relation, a relation that relates a single object to all other objects, but no other object to any object, is an example of a one-to-one relation that's succinctly non-functional. However, in order to introduce a restricted notion of identity, functionality is not a requirement.

On the other hand, in order to simplify modal reasoning and to be able to introduce well-behaved restricted variables, it is helpful to only consider *rigid* one-to-one relations. A rigid one-to-one relation is a relation that is one-to-one and rigid (see A.11.2995, A.12.3259):[19]

$$\mathit{Rigid}_{1-1}(R) \equiv \mathit{1{-}1}(R)\ \&\ \mathit{Rigid}(R)$$

Since being a rigid one-to-one relation is a rigid restriction condition, we can introduce well-behaved restricted variables that range over them.[20]

In the following we will use $\mathcal{R}$ as a restricted variable ranging over rigid one-to-one relations.[21]

5.7.2. Identity Restricted to the Domain of Rigid One-to-one Relations

For a variable $\mathcal{R}$ that is restricted to rigid one-to-one relations, a restricted notion of identity can now be defined as follows (see A.12.3372):

[18] For example, if R is a necessarily empty relation, the matrix of $[\lambda xy\ [R^*]xy \vee x = y]$ is necessarily equivalent to $[\lambda xy\ x = y]$ for all x and y and $[\lambda xy\ [R^*]xy \vee x = y]$ fails to denote by co-existence.

[19] Recall the discussion about rigid relations in section 3.8.3.

[20] Recall the discussion of restricted variables in section 3.4.4.

[21] Note that PLM uses $\underline{R}$. However, in our framework choosing $\mathcal{R}$ is simpler for technical reasons.

$$(=_{\mathcal{R}}) =_{df} [\lambda xy\ \exists z\ ([\mathcal{R}]xz\ \&\ [\mathcal{R}]yz)]$$

Note that in contrast to general identity, the definiens of $\mathcal{R}$-identity (trivially) denotes a proper relation.

By β-conversion and using infix notation, two objects x and y are $\mathcal{R}$-identical, just in case that there is an object to which they are both $\mathcal{R}$-related (see A.12.3379):

$$x =_{\mathcal{R}} y \equiv \exists z\ ([\mathcal{R}]xz\ \&\ [\mathcal{R}]yz)$$

Since $\mathcal{R}$ is restricted to rigid one-to-one relations, the resulting identity relation is exactly the restriction of general identity to the domain of $\mathcal{R}$ (see A.13.266):

$$x =_{\mathcal{R}} y \equiv \mathit{InDomainOf}(x,\mathcal{R})\ \&\ \mathit{InDomainOf}(y,\mathcal{R})\ \&\ x = y$$

Consequently, the defined identity is a partial equivalence relation that is reflexive on the domain of $\mathcal{R}$ (see A.12.3470):

$$\mathit{InDomainOf}(x,\mathcal{R}) \to x =_{\mathcal{R}} x$$
$$x =_{\mathcal{R}} y \to y =_{\mathcal{R}} x$$
$$x =_{\mathcal{R}} y\ \&\ y =_{\mathcal{R}} z \to x =_{\mathcal{R}} z$$

A simple example of a rigid one-to-one-relation is the identity on the ordinary objects $(=_E)$, the domain of which is the ordinary objects (see A.13.768 and A.13.788).

5.7.3. The Weak Ancestral of a Relation

Based on the concept of $\mathcal{R}$-identity, the *weak ancestral* $\mathcal{R}^+$ of a rigid one-to-one relation $\mathcal{R}$ can be defined as follows (see A.12.3529):

$$\mathcal{R}^+ =_{df} [\lambda xy\ [\mathcal{R}^*]xy \vee x =_{\mathcal{R}} y]$$

Restricting to the domain of $\mathcal{R}$, two objects are now exactly in the weak ancestral relation $\mathcal{R}^+$ if they are either transitively $\mathcal{R}$-related (i.e. in the strong ancestral relation $\mathcal{R}^*$) or identical:

$$\mathit{InDomainOf}(x,\mathcal{R}) \to ([\mathcal{R}^+]xy \equiv [\mathcal{R}^*]xy \vee x = y)$$

In other words, the weak ancestral of a relation is its transitive and reflexive closure, with reflexivity being restricted to the domain of the relation.

5.8. Generalized Induction

In order to understand the formulation of generalized induction, first consider the following theorem that Nodelman and Zalta prove before even introducing weak ancestral relations, but which already has "inductive character" (see A.12.3160):

$$[F]x\ \&\ [R^*]xy\ \&\ \mathit{Hereditary}(F,R) \to [F]y$$

While this may not look like an inductive principle as stated, unfolding the definition of *Hereditary*, this is equivalent (under some trivial transformations) to the following:

AOT-theorem *pre-ind′*: ‹$[F]z$ & $\forall x \forall y([R]xy \rightarrow ([F]x \rightarrow [F]y)) \rightarrow \forall y\ ([R]^*zy \rightarrow [F]y)$›
proof(*safe intro*!: $\rightarrow I$ *GEN*)
  fix y
  AOT-assume ‹$[F]z$ & $\forall x \forall y([R]xy \rightarrow ([F]x \rightarrow [F]y))$›
  AOT-hence ‹$[F]z$ & *Hereditary*(F,R)›
    by (*AOT-subst-def hered*:*1*) (*auto intro*!: &*I cqt*:*2 elim*: &*E*)
  moreover AOT-assume ‹$[R]^*zy$›
  moreover AOT-have ‹$[F]z$ & $[R^*]zy$ & *Hereditary*$(F,R) \rightarrow [F]y$›
    using *anc−her*:*3*. — This is an instance of the theorem cited above.
  ultimately AOT-show ‹$[F]y$›
    using &*I* &*E* $\rightarrow E$ **by** *metis*
qed

I.e. if an object z exemplifies F and F is inherited via R, then any object that is transitively R-related to z exemplifies F.

Pretend for a moment that R was a well-defined predecessor relation and z the number Zero. Then this theorem would imply that if (1) F holds for Zero and (2) for any x and y, s.t. x precedes y, if x exemplifies F, then y exemplifies F, then F holds for all numbers transitively preceded by Zero (and since F also holds for Zero by assumption this would trivially imply that F holds for any natural number).

In principle, this is how mathematical induction will be derived. However, it is inconvenient that the induction step in this formulation ranges over the full domain of all objects. Instead, it should be sufficient to consider all natural numbers.

By instantiating F to $[\lambda x\ [F]x\ \&\ [\mathcal{R}^+]zx]$, we arrive at the following theorem, which PLM refers to as *Generalized Induction* (see A.12.3851):[22]

$$[F]z\ \&\ \forall x\ \forall y\ ([\mathcal{R}^+]zx\ \&\ [\mathcal{R}^+]zy \rightarrow ([\mathcal{R}]xy \rightarrow ([F]x \rightarrow [F]y))) \rightarrow$$
$$\forall x\ ([\mathcal{R}^+]zx \rightarrow [F]x)$$

In this formulation, the induction step merely ranges over objects to which z bears the weak ancestral relation of $\mathcal{R}$. Thinking of $\mathcal{R}$ as the predecessor relation and z as the number Zero, this will be exactly the natural numbers. I.e. instantiating this generalized principle of induction to the predecessor relation and the number Zero, yields classical mathematical induction (relative to the upcoming definition of natural numbers).

5.9. The Predecessor Relation

5.9.1. Definition

While the definition of the predecessor relation is rather straightforward, the interesting question will be whether it actually denotes a relation, which we will discuss in detail

[22]Note that (1) $[\mathcal{R}^+]zy$ for any y implies $[\mathcal{R}^+]zz$, yielding $[\lambda x\ [F]x\ \&\ [\mathcal{R}^+]zx]z$ in all cases in which the consequent of the strengthened theorem does not trivially hold (i.e. if $\neg\exists\, y\ [\mathcal{R}^+]zy$) and (2) that the consequent of the weaker theorem can be strengthened since $[\mathcal{R}^+]zy$ either implies (a) $z = y$, in which case $[F]y$ follows from the assumption $[F]z$, or it implies (b) $[\mathcal{R}^*]zy$, in which case the consequent of the weaker principle yields $[F]y$. The additional fact that $[\mathcal{R}]xy$ and $[\mathcal{R}^+]zx$ imply $[\mathcal{R}^+]zy$ is sufficient to arrive at the strengthened theorem.

in section 5.9.2. For the moment assume that the λ-expression in the definiens of the following definition denotes (see A.12.4288):

$$\mathbb{P} =_{df} [\lambda xy\ \exists F\ \exists u\ ([F]u\ \&\ \mathit{Numbers}(y,F)\ \&\ \mathit{Numbers}(x,F^{-u}))]$$

Given the assumption that this relation denotes, it follows by β-conversion that (see A.12.4294):

$$\mathbb{P}xy \equiv \exists F\ \exists u\ ([F]u\ \&\ \mathit{Numbers}(y,F)\ \&\ \mathit{Numbers}(x,F^{-u}))$$

So an object x precedes an object y just in case there is a property F and an ordinary object u, s.t. u exemplifies F, y numbers F and x numbers being an F other than u (via the definition $F^{-u} =_{df} [\lambda z\ [F]z\ \&\ z \neq_E u]$).

This is a variant of Frege's definition of the successor relation.[23] The idea can be clarified by considering how the first natural numbers are related w.r.t. this relation:

- The number Zero numbers properties that are not exemplified by any ordinary object. Hence there cannot be a property F that is exemplified by an object u, s.t. Zero numbers F, which means that Zero is not preceded by any object.
- The number One numbers properties that are exemplified by a single ordinary object.[24] Hence any property F numbered by One is exemplified by some ordinary object u. Furthermore, F^{-u}, i.e. being an object exemplifying F other than u, is not exemplified by any ordinary object. Hence Zero is the unique predecessor of One.
- The number Two numbers properties that are exemplified by two distinct ordinary objects. Being an object that exemplifies any of these properties other than any particular object the given property exemplifies, is a property exemplified by only one ordinary object, hence numbered by One, i.e. One precedes Two, etc.

5.9.2. Assuring that the Predecessor Relation Denotes

It is important to realize that the λ-expression used in the definition above does not trivially denote a relation in AOT. *Numbering a property* is an encoding claim: an object numbers a property G, just in case it encodes all properties, s.t. actually exemplifying it is equinumerous to G. In general, encoding claims cannot be abstracted to denoting properties and relations.[25]

In fact it is easy to see that the minimal model of AOT does not validate this axiom: the minimal model contains one ordinary urelement (resp. one ordinary object) and one special urelement. Since special urelements determine the exemplification extensions of abstract objects, there being only one special urelement implies that all abstract objects

[23] Nodelman and Zalta argue in favour of a predecessor relation due to the fact that in contrast to a successor relation, the argument order of the predecessor relation matches the numerical order of objects in the relation. Apart from that, the notions are interchangeable, i.e. $\mathit{Succeeds}(y,x)$ is exactly $\mathbb{P}xy$.

[24] While we have not formally introduced any number other than Zero, we consider this intuitive understanding helpful in clarifying the idea of the predecessor relation. The number One will formally be introduced later in this chapter.

[25] Recall the discussion in section 3.6.

exemplify the same properties and relations. This implies in particular that either all objects are preceded by Zero (including Zero itself) or no object is, i.e. $\mathbb{P}\mathit{00}$ or $\neg\exists x\ \mathbb{P}\mathit{0}x$. However, we have already (informally) argued above that Zero is not preceded by any object.[26] Hence in this model it would have to hold that $\neg\exists x\ \mathbb{P}\mathit{0}x$. However, since the minimal model still contains one ordinary object, the number One can be constructed and (again as argued above) is preceded by Zero, i.e. $\mathbb{P}\mathit{01}$, which yields a contradiction.

Nodelman and Zalta assert that the predecessor relation denotes by axiom and emphasize that the relation is not inherently mathematical and no mathematical primitives are needed to assert, as an axiom, that it denotes (see PLM item (782)). In particular, they argue that expressions of the form $\mathit{Numbers}(y,F)$, while seemingly mathematical in nature, can be eliminated, since they are *defined* in terms of primitives of AOT. Furthermore, they argue that the relation merely asserts the existence of an ordering relation on abstract objects and ordering relations can, in general, be expressed in entirely logical terms.

However, even if one concedes that the axiom is not inherently mathematical, it can be objected that it is rather *ad-hoc*: rather than asserting a general principle according to which encoding claims can be abstracted to relations, it singles out a specific relation and this relation is, after all, used to *define* a concept that is very much mathematical in nature. Furthermore, the axiom is not trivially consistent: as we have seen, minimal models of the base system of AOT do not validate it.

Using our embedding we can, however, contribute to this situation in two ways:

- We can show that the axiom is consistent by constructing models that validate it.
- We can generalize the axiom to an independently justifiable comprehension principle for relations among abstract objects, s.t. it becomes a theorem that the predecessor relation in particular denotes.

We defer a more detailed discussion to section 5.19 and in the following continue to reproduce the construction of natural numbers and the derivation of the Dedekind-Peano postulates as given by Nodelman and Zalta in PLM.

5.9.3. The Predecessor Relation as Rigid One-to-One Relation.

It can be derived that the predecessor relation is modally rigid: $\mathit{Rigid}(\mathbb{P})$, respectively $\mathbb{P}xy \rightarrow \Box\mathbb{P}xy$. While the full proofs can be found in A.12.4301, it is noteworthy that it again requires that one argue by appealing to *rigidifying* relations: by the theorem governing the predecessor relation given above, $\mathbb{P}xy$ implies that there exists a property F and an ordinary object u, s.t. $[F]u$ & $\mathit{Numbers}(y,F)$ & $\mathit{Numbers}(x,F^{-u})$. However, none of the conjuncts are guaranteed to be necessary. But we may refer to the fact that for any property F there exists a property G that *rigidifies* F and this property G can serve as witness for the claim that $\Box\mathbb{P}xy$.

[26]Both $\neg\exists x\ \mathbb{P}x\mathit{0}$ and $\mathbb{P}\mathit{01}$ are formally derived in A.12.4498, resp. A.12.5437.

Furthermore, it is a consequence of a modally-strict variant of Hume's principle that the predecessor relation is one-to-one (see A.12.4426): $\mathit{1{-}1}(\mathbb{P})$.

Consequently, the Predecessor Relation is a rigid one-to-one relation and we can instantiate the definition of the *strong* ancestral to $\mathbb{P}$ (see A.12.4468):

$$\mathbb{P}^* = [\lambda xy\ \forall F\ (\forall z\ (\mathbb{P}xz \rightarrow [F]z)\ \&\ \mathit{Hereditary}(F,\mathbb{P}) \rightarrow [F]y)]$$

Furthermore, being $\mathbb{P}$-identical as well as the *weak* ancestral of $\mathbb{P}$ are also well-defined (see A.12.4540):

$$x =_{\mathbb{P}} y \equiv \exists z\ (\mathbb{P}xz\ \&\ \mathbb{P}yz)$$
$$\mathbb{P}^+ = [\lambda xy\ [\mathbb{P}^*]xy \vee x =_{\mathbb{P}} y]$$

Before we continue to define natural numbers, note that it is already derivable that the number Zero neither has a direct nor a transitive predecessor (see A.12.4498): $\neg\exists x\ \mathbb{P}x\mathit{0}$ respectively $\neg\exists x\ [\mathbb{P}^*]x\mathit{0}$

5.10. Natural Numbers

Using the infrastructure introduced in the past sections, we can now follow through with the strategy described in the beginning of the chapter and define *being a natural number* as being an object, s.t. Zero bears the weak ancestral of the predecessor relation to it (see A.12.4577):

$$\mathbb{N} =_{df} [\lambda x\ [\mathbb{P}^+]\mathit{0}x]$$

Since being a natural number trivially denotes, it follows by β-conversion that (see A.12.4582):

$$\mathbb{N}x \equiv [\mathbb{P}^+]\mathit{0}x$$

5.11. Zero is a Natural Number

The first Dedekind-Peano postulate can now be derived (see A.12.4588):

$$\mathbb{N}\mathit{0}$$

Interestingly, both in Frege's original work and in Zalta's initial reconstruction (see [60]) the weak ancestral was defined using general identity and consequently $[\mathbb{P}^+]\mathit{00}$ is a simple consequence of the fact that Zero is self-identical. However, due to the construction via rigid one-to-one relations this theorem requires a non-trivial proof: $[\mathbb{P}^+]\mathit{00}$ by definition is just the case if either $[\mathbb{P}^*]\mathit{00}$ (which was already refuted above) or $\mathit{0} =_{\mathbb{P}} \mathit{0}$.

However, $\mathit{0} =_{\mathbb{P}} \mathit{0}$ is not a simple consequence of the fact that $\mathit{0} = \mathit{0}$, but additionally requires that $\mathit{InDomainOf}(\mathit{0},\mathbb{P})$, respectively that $\exists y\ \mathbb{P}\mathit{0}y$, i.e. the proof effectively requires to construct the number One as witness.[27]

[27] The number One can for example be introduced as the number of any relation exemplified by exactly one ordinary object. Since it is a theorem (see A.7.7510) that there is an ordinary object $\exists x\ O!x$, we can choose a to be a witness to this existential claim and choose $\#[\lambda x\ O!x\ \&\ x =_E a]$ as a witness to $\exists y\ \mathbb{P}\mathit{0}y$..

Preliminary working versions of the chapter of PLM left this non-trivial proof as an exercise referring to it being a trivial consequence of the self-identity of the number Zero. Trying to prove the statement in the embedding showed that additional work is required due to the changes in the construction compared to previous versions and we were able to notify Nodelman and Zalta both that the proof is non-trivial and suggest to them the proof given in A.12.4588 and outlined in the footnote of the last paragraph.[28]

5.12. Being a Natural Number is Rigid

From the generalized principle of induction when instantiating F to $[\lambda x\ \Box\mathbb{N}x]$ and $\mathcal{R}$ to $\mathbb{P}$, it follows that $\mathbb{N}x \rightarrow \Box\mathbb{N}x$ and consequently that $Rigid(\mathbb{N})$ (see A.12.4631, A.12.4689).

Since furthermore Zero is a witness to the existence of natural numbers and it is easy to prove that $\mathbb{N}\kappa \rightarrow \kappa\downarrow$,[29] *being a natural number* is a *rigid restriction condition* and it is possible to introduce well-behaved restricted variables ranging over the natural numbers (recall section 3.4.4).

In the following the variable names m, n, k, i and j are used to range over natural numbers.

5.13. Zero Has No Predecessor

We have already mentioned the fact that $\neg\exists\, x\ \mathbb{P}x\mathit{0}$ above, but we can now restate this theorem *a fortiori* for variables restricted to natural numbers, which constitutes the second Dedekind-Peano postulate (as mentioned earlier this formulation is equivalent to the assertion that Zero is not the successor of any natural number; see A.12.4714):

$$\neg\exists\, n\ \mathbb{P}n\mathit{0}$$

5.14. No Two Natural Numbers have the Same Successor

The third Dedekind-Peano postulate is a general property of any one-to-one relation, but can be stated explicitly using restricted variables for natural numbers (on which $\mathbb{P}$-identity matches general identity) as follows (see A.12.4725):

$$\forall\, n\ \forall\, m\ \forall\, k\ (\mathbb{P}nk\ \&\ \mathbb{P}mk \rightarrow n = m)$$

Whenever two natural numbers n and m precede the same natural number k (or, equivalently, if n and m have the same successor), they have to be identical.

[28]Note that the chapter was under heavy revision at the time and this omission would likely have been independently discovered eventually. However, it is one of the merits of working in a computer-verified setting that such omissions become immediately apparent.

[29]It is a simple consequence of one of the quantifier axioms mentioned in section 3.3.

5.15. Mathematical Induction

Furthermore, we can now derive Mathematical Induction (see A.12.4738):

$$\forall F\ ([F]0\ \&\ \forall n\ \forall m\ (\mathbb{P}nm \rightarrow ([F]n \rightarrow [F]m)) \rightarrow \forall n\ [F]n)$$

If a property (1) is exemplified by the number Zero and (2) it being exemplified by a natural number implies it being exemplified by its successor, then all natural numbers exemplify that property.[30] This is a simple consequence of instantiating generalized induction (recall section 5.8) to the predecessor relation.

Thereby the fifth Dedekind-Peano postulate is derivable. Note, however, that we haven't yet derived the fourth postulate, i.e. that every natural number has a unique successor. The construction so far is validated by the minimal models of AOT that are extended to validate the predecessor axiom (i.e. in which the predecessor relation denotes). Validating the predecessor axiom involves increasing the number of special urelements in the model (see 5.19), but it does not require to increase the number of ordinary urelements/objects, so there are still models with only a single ordinary urelement/object, in which the predecessor relation denotes. However, in such models the only natural numbers are Zero and One and the number One does not have a successor. For that reason, Nodelman and Zalta extend the system by another axiom, which we will discuss below after stating a few more derived properties of the predecessor relation and natural numbers.

5.16. Properties of the Predecessor Relation and Natural Numbers

Successors of natural numbers are (transitively) natural numbers (see A.12.4766):

$$\mathbb{P}nx \rightarrow \mathbb{N}x$$
$$[\mathbb{P}^*]nx \rightarrow \mathbb{N}x$$
$$[\mathbb{P}^+]nx \rightarrow \mathbb{N}x$$

Predecessors of natural numbers are (transitively) natural numbers (see A.12.4824):

$$\mathbb{P}xn \rightarrow \mathbb{N}x$$
$$[\mathbb{P}^*]xn \rightarrow \mathbb{N}x$$
$$[\mathbb{P}^+]xn \rightarrow \mathbb{N}x$$

Natural numbers are natural cardinals (see A.12.4865):

$$\mathbb{N}x \rightarrow \mathit{NaturalCardinal}(x)$$

The predecessor relation is functional (see A.12.4899):

[30] Note that, strictly speaking, our natural language formulation rather corresponds to the derived theorem $\forall F\ ([F]0\ \&\ \forall n\ ([F]n \rightarrow [F]n') \rightarrow \forall n\ [F]n)$ (see A.13.273), where n' is defined as *the* successor of n, resp. *the* natural number that is preceded by n (see A.12.5336). However, this formulation can only be derived after proving that every number *has* a (unique) successor.

$\mathbb{P}xy \ \& \ \mathbb{P}xz \to y = z$
$\mathbb{P}nm \ \& \ \mathbb{P}nk \to m = k$

5.17. Possible Richness of Objects

As mentioned above, the construction so far is valid in models with only a single ordinary object and consequently with only two natural numbers, which is not sufficient to derive that every natural number has a successor.

The following modal axiom, by which Nodelman and Zalta proceed to extend the system, changes this (see A.12.4956):

$\exists x \ (\mathbb{N}x \ \& \ x = \#G) \to \Diamond\exists y \ (E!y \ \& \ \forall u \ (\boldsymbol{\mathcal{A}}[G]u \to u \neq_E y))$

If there is a natural number which numbers G, then there might have been a concrete object y which is distinct from every ordinary object that *actually* exemplifies G. We will explain in detail how we extend our models to be able to validate this axiom in section 5.20. In summary, the axiom requires extending the domain of ordinary urelements/objects to an at least countably infinite set.

This axiom requires some justification, especially given the claim that the construction is *purely logical* and does not require to presuppose any intrinsically mathematical claims. Traditionally, a system is no longer considered to be *purely logical*, if it asserts the existence of more than one object.[31] While Nodelman and Zalta agree with this principle, they argue (see PLM item (799)) that it only extends to *concrete* objects. While above axiom does imply that the domain of *ordinary* objects (recall that *being ordinary* is defined as *being* possibly *concrete*) is at least countably infinite, it does not imply that there is even a single object that is *actually concrete*. Nodelman and Zalta further argue that on the one hand it is in fact common for logical systems to assert the existence of more than one *abstract* object, for example that there are two distinct truth values, the True and the False,[32] and that on the other hand logicians traditionally work under the assumption that *the domain of objects* might *be of any size*, which they take as a modal claim: while logic may not presuppose that the domain of concrete objects has any particular size, it allows for the *possibility* of the domain being of any size, i.e. it is valid for a logic to presuppose that there *may possibly* be more than one object, as long as that does not imply that there is *actually* more than one (concrete) object.

Independently of the question whether this axiom may or may not be considered as *purely logical*, towards which we refrain from presuming to pass judgement either way, we certainly agree that it captures a pretheoretic intuition: it can be considered as a prerequisite of talking about natural numbers to be able to imagine that no matter how many objects we currently consider that there *possibly might have been* yet another

[31] E.g. PLM cites Boolos [14]: "In logic, we ban the empty domain as a concession to technical convenience but draw the line there: We firmly believe that the existence of even two objects, let alone infinitely many, cannot be guaranteed by logic alone."

[32] In particular, they refer to Frege's logic.

object, even though for doing so we do not need to be able to point to this object in the actual world (i.e. it may not be concrete, but merely *possibly concrete*).

While this may serve as justification for the axiom, Frege's original construction does not rely on a similar assumption, but can use the number of the property *being less than or equal to* n, $\#[\lambda x\ x \leq n]$, as witness for a successor of any natural number n. In the presented construction that relies on equinumerosity among the ordinary objects, this is not an option: since natural numbers are abstract, being a natural number smaller or equal to n is only exemplified by abstract objects and therefore unexemplified by ordinary objects. Thus $\#[\lambda x\ x \leq n]$ is Zero and, in particular, cannot serve as the successor of any number.

However, we will discuss two variants of the construction in section 5.21 in which *discernible* abstract objects *can* be counted (and in which natural numbers, in particular, will be discernible). This allows for the construction of a successor of n as $\#[\lambda x\ x \leq n]$, thereby eliminating the need for this axiom.

5.18. Every Number has a Unique Successor

The axiom above is sufficient to derive the last Dedekind-Peano postulate, i.e. that every natural number has a unique successor (see A.12.5249):

$$\forall\, n\ \exists!m(\mathbb{P}nm)$$

Every natural number n is a natural cardinal and, by definition (see A.12.2570), natural cardinals are the number of some property and thus $\mathit{NaturalCardinal}(n) \rightarrow \exists\, G\ n = \#G$.

Let G be a property such that $n = \#G$.

Now the axiom implies that there is an ordinary object v, s.t. G does not actually exemplify v. This requires an appeal to the Barcan formulas (in particular A.7.3470) and relies on the additional fact (see A.12.5215) that:

$$\Diamond\forall\, u\ (\boldsymbol{\mathcal{A}}[G]u \rightarrow u \neq_E v) \rightarrow \forall\, u\ (\boldsymbol{\mathcal{A}}[G]u \rightarrow u \neq_E v)$$

Hence, since $n = \#G$ implies that n numbers $[\lambda x\ \boldsymbol{\mathcal{A}}[G]x]$ (see A.12.2742), the object that numbers $[\lambda x\ \boldsymbol{\mathcal{A}}[G]x \vee x =_E v]$ can be used as witness for a successor of n.

Uniqueness follows from the fact that the predecessor relation is functional.

Hence, it is possible to define *the* successor n' of a natural number n as *the* natural number that is preceded by m:

$$n' =_{df}\ \iota x(\mathbb{N}x\ \&\ \mathbb{P}nx)$$

Numerals can be defined as iterated successors, e.g. $1 =_{df}\ 0'$.

While PLM continues to derive further theorems of Number Theory, defines mathematical functions and operations, including recursively defined functions such as addition, and proceeds to derive Second-Order Dedekind-Peano arithmetic, we will conclude our discussion of the topic here and instead discuss in more detail how we modelled the two required additional axioms.

5.19. The Predecessor Axiom in Detail

Recall that the predecessor axiom of PLM is stated as follows (see A.12.4284):

$$[\lambda xy\ \exists F\ \exists u\ ([F]u\ \&\ Numbers(y,F)\ \&\ Numbers(x,F^{-u}))]{\downarrow}$$

In section 5.9.2 we have already established that the relation in question distinguishes certain abstract objects that number properties and that this relation does *not* denote in the minimal models of the base system of AOT. We also have already discussed that there cannot be a relation in AOT that generally distinguishes between arbitrary abstract objects (in particular $[\lambda xy\ x = y]$ does not denote; see 3.8.1). So we need to determine what is special about the abstract objects that are distinguished by the predecessor relation and allows us to construct models for it.

To that end, we first show that the predecessor relation coexists with *numbering a property*. In particular we can prove the following (see A.12.3905):

$$[\lambda xy\ \exists F\ \exists u\ ([F]u\ \&\ Numbers(y,F)\ \&\ Numbers(x,F^{-u}))]{\downarrow} \equiv \forall F\ [\lambda x\ Numbers(x,F)]{\downarrow}$$

So to validate the predecessor axiom, we can equivalently construct models in which $[\lambda x\ Numbers(x,F)]{\downarrow}$ is a theorem. Recall that *numbering a property* is equivalent to the following (see A.12.2180):

$$Numbers(x,G) \equiv A!x\ \&\ \forall F\ (x[F] \equiv [\lambda z\ \boldsymbol{\mathcal{A}}[F]z] \approx_E G)$$

So while *numbering a property* is a condition on the properties an abstract object encodes, it requires the abstract object to encode an entire class of properties, namely all properties, s.t. *actually* exemplifying them is equinumerous$_E$ to the numbered property. Further recall that being *equinumerous$_E$*, informally speaking, means to be exemplified by the same amount of *ordinary* objects.

This is the crucial fact that allows us to construct suitable models: while we need to distinguish between abstract objects based on the properties they *encode*, the condition under which these abstract objects encode or do not encode properties solely depends on the exemplification patterns of those properties on the *ordinary* objects.

In our models, two abstract objects are exemplification-distinguishable, if they are mapped to distinct *special urelements*. If we wanted to be able to distinguish between abstract objects in general based on the exemplification patterns of the properties they encode, this would mean that there had to be a distinct *special urelements* for any such pattern. Exemplification of a property is a functions from *urelements* (including special urelements) to modal truth conditions (i.e. functions from semantic possible worlds to booleans).

Therefore, if we wanted to assign distinct special urelements based on *general* exemplification patterns, we would need an injective function from exemplification patterns (i.e. sets of functions acting on urelements) to special urelements, which would be in violation of Cantor's theorem.

However, fortunately, we only need to distinguish between exemplification patterns on *ordinary* objects. Since the domains of special urelements and ordinary urelements are independent, it is consistently possible to construct special urelements in such a way that there can be an injective function mapping distinct sets of functions *acting on ordinary urelements alone* to distinct special urelements.

In our general models we choose an *abstract* type σ as type of special urelements.[33] In our extended models that validate the predecessor axiom, we instead *define* the type σ using the set of objects of type $(\omega \Rightarrow w \Rightarrow \mathit{bool})\ \mathit{set} \times (\omega \Rightarrow w \Rightarrow \mathit{bool})\ \mathit{set} \times \sigma'$ as representation set.[34]

Recall that the type ω is the type of ordinary urelements and w is the type of semantic possible worlds. σ' is an additional abstract type of *very special urelements* that will retain the model's ability to distinguish between abstract objects beyond those that differ in exemplification patterns on the ordinary objects. So in these models, special urelements are tuples of two sets of property extensions on ordinary objects and a very special urelement. We refer to the first set of extensions as the *intersection set of ordinary property extensions* and to the second copy as the *union set of ordinary property extensions*.

When we map an abstract object a to this new type of special urelements, we insert a property extension on the ordinary objects into the intersection set, just in case a encodes *all* properties with this extension on the ordinary objects. And we insert an extension into the union set, just in case that there *exists* a property with that extension (on the ordinary objects) that is encoded by a.

We use this construction as witness for a specification of the mapping $\alpha\sigma'$, which will then be extended to a surjective mapping $\alpha\sigma$ as explained in section 4.1.4.

This construction *forces* two abstract objects to be assigned different special urelements, in case either (1) one of them encodes a property with a given exemplification extension on the ordinary object, while the other doesn't encode any such property, or (2) one of them encodes all properties with a given extension on the ordinary object, while the other fails to encode at least one such property.

Furthermore, the construction still *allows* two abstract objects to be assigned different special urelements, in case they differ only in encoding properties with the same extension on the ordinary objects (by assigning them distinct *very special* urelements).

This extended model validates the following two axioms (see A.6.245, A.6.252):

- $\Pi\!\downarrow\ \&\ A!x\ \&\ A!y\ \&\ \forall F\ \Box([F]x \equiv [F]y) \rightarrow (\forall G\ (\forall u\ \Box([G]u \equiv [\Pi]u) \rightarrow x[G]) \equiv \forall G\ (\forall u\ \Box([G]u \equiv [\Pi]u) \rightarrow y[G]))$
- $\Pi\!\downarrow\ \&\ A!x\ \&\ A!y\ \&\ \forall F\ \Box([F]x \equiv [F]y) \rightarrow (\exists G\ (\forall u\ \Box([G]u \equiv [\Pi]u)\ \&\ x[G]) \equiv \exists G\ (\forall u\ \Box([G]u \equiv [\Pi]u)\ \&\ y[G]))$

I.e. if two abstract objects are (exemplification-)indistinguishable, then (1) if either one encodes all properties that are necessarily equivalent on the ordinary objects to any given denoting property term Π, then the other also encodes all these properties, and

[33]I.e. we allow any non-empty domain for σ in models of the meta-logic without restriction.

[34]A smaller subset of the set of such triples $(a,\ b,\ s)$, e.g. for which it always holds that $a \subseteq b$ and for which $a = b$ implies $s = s_0$ for some fixed s_0, would suffice.

(2) if either one encodes any property that is necessarily equivalent to Π on the ordinary objects, there is also such a property that is encoded by the other.

While this formulation of the axioms is rather complex and not particularly intuitive, we can equivalently (given the necessary and sufficient conditions for relation terms to denote described in section 3.8.2) state them as follows (see A.10.445, A.10.458):

$$[\lambda x\ \exists\, G\ (\Box G \equiv_E F\ \&\ x[G])]{\downarrow}$$
$$[\lambda x\ \exists\, G\ (\Box G \equiv_E F\ \&\ \neg x[G])]{\downarrow}$$

I.e. (1) *encoding a property that is necessarily equivalent on the ordinary objects to a given property F* denotes a property and (2) *not encoding a property that is necessarily equivalent on the ordinary objects to a given property F* denotes a property.[35]

The following comprehension principles are derivable from the fact that above properties denote (see A.10.473, A.10.531):

$$\Box\forall\, F\ \forall\, G\ (\Box G \equiv_E F \to (\varphi\{F\} \equiv \varphi\{G\})) \to [\lambda x\ \exists\, F\ (\varphi\{F\}\ \&\ x[F])]{\downarrow}$$
$$\Box\forall\, F\ \forall\, G\ (\Box G \equiv_E F \to (\varphi\{F\} \equiv \varphi\{G\})) \to [\lambda x\ \exists\, F\ (\varphi\{F\}\ \&\ \neg x[F])]{\downarrow}$$

We call φ a *condition on extensions on ordinary objects*, just in case it satisfies the antecedent, i.e. just in case that $\Box\forall\, F\ \forall\, G\ (\Box G \equiv_E F \to (\varphi\{F\} \equiv \varphi\{G\}))$. Then the comprehension principles state that for any condition φ on extensions on ordinary objects, both *encoding a property that satisfies φ* and *not encoding a property that satisfies φ* denote properties.[36]

In combination these two principles yield the following (see A.10.633):[37]

$$\Box\forall\, F\ \forall\, G\ (\Box G \equiv_E F \to (\varphi\{F\} \equiv \varphi\{G\})) \to [\lambda x\ \forall\, F\ (x[F] \equiv \varphi\{F\})]{\downarrow}$$

I.e. for every condition φ on extensions on ordinary objects, *encoding exactly those properties that satisfy φ* denotes a property.

It is easy to show that *being an F, s.t. actually exemplifying F is equinumerous to G*, is a condition on extensions on ordinary objects. Hence it is a consequence of this last comprehension principle that $[\lambda x\ \forall\, F\ (x[F] \equiv [\lambda z\ \boldsymbol{\mathcal{A}}[F]z] \approx_E G)]{\downarrow}$ and thereby *numbering a property* denotes by coexistence (see A.12.4236).

Justification of the Comprehension Principles

While the predecessor axiom singles out a particular relation among abstract objects for the sole purpose of defining a mathematical relation, the comprehension principles we suggest provide a general means to construct relations among abstract objects based on specific encoding patterns in a manner that is provably consistent, but also independently justifiable.

[35]Note that these properties coexist with their negations, i.e. $[\lambda x\ \forall\, G\ (\Box G \equiv_E F \to \neg x[G])]{\downarrow}$ is equivalent to the first, $[\lambda x\ \forall\, G\ (\Box G \equiv_E F \to x[G])]{\downarrow}$ is equivalent to the second.

[36]See also A.10.591 and A.10.613 for derived variants of these principles.

[37]However, note that above principles are stronger, i.e. they are *not* derivable from the combined principle.

In general, the burden of justification rather lies in the fact that some abstract objects *are* exemplification-indistinguishable: let R_t be the relation *thinking about*, s.t. $[R_t]xy$ can be read as x is thinking about y. Then for two distinct abstract objects a and b to be exemplification-indistinguishable implies that it is impossible for anyone to think about one without thinking about the other: $\forall x\, \Box([R_t]xa \equiv [R_t]xb)$, resp. $\neg\Diamond\exists x\, ([R_t]xa \,\&\, \neg[R_t]xb \vee [R_t]xb \,\&\, \neg[R_t]xa)$.

While the existence of such objects is justifiable, it is not necessarily a pretheoretic intuition. Interestingly, it is not possible to *independently* construct two abstract objects that are in fact exemplification-indistinguishable: while it is provable that there *exist* such pairs of objects, the construction always has to rely on constructing one of the objects particularly in such a way that it cannot be distinguished from the other.[38] Whenever two abstract objects are constructed independently, a model can generally choose two distinct special urelements for them, thereby making them distinguishable. Only if the construction of the second abstract object *depends* on the choice of a special urelement for the first and forces both objects to be collapsed under the mapping from abstract objects to special urelements, this becomes infeasible.

This helps in reconciling the fact that there are indistinguishable abstract objects with the following pretheoretic intuition: given two independent abstract objects, we can always find ourselves thinking about one, but not the other. However, we can conceive of concepts that e.g. themselves involve *being indistinguishable from other abstract objects*, for which a clever construction in fact yields distinct concepts that are indistinguishable.[39]

So while we can always consistently distinguish between *particular, independent* abstract objects, given that there still *are* indistinguishable abstract objects, we cannot formulate a completely general principle that allows for distinguishing *arbitrary* abstract objects.

However, our suggested comprehension principles are restricted to abstract objects that have encoding conditions that differ in exemplification patterns on the *ordinary* objects. If for two abstract objects we can point to a pattern among the ordinary objects, s.t. one of the object involves this pattern (i.e. it encodes a property that satisfies this pattern), but the other one doesn't involve this pattern at all (i.e. it encodes no property that satisfies this pattern), we have a concrete criterion for telling the objects apart.[40] The same can be said, if one of the object fails to fully encode such a pattern (i.e. there is a property with this pattern on the ordinary objects that it doesn't encode), while the other encodes all properties with this pattern.

The third, combined principle (which is weaker than the first two principles, but strong enough for *numbering a property* to denote) is seemingly even easier to justify: if an abstract object encodes *exactly* those properties that satisfy a given pattern on the

[38] And even this is only possible for specific choices of a first abstract object: For example, we cannot construct an abstract object that is indistinguishable from the null-object (that encodes no properties) since we can always conceive of a model that maps the null-object to a designated special urelement that no other abstract object maps to.

[39] Recall the discussion in section 3.8.1.

[40] Respectively, equivalently by 3.8.2, for allowing a property that tells them apart to denote.

ordinary objects, then it is fully determined by this pattern, so in this sense we can *identify* such abstract objects with the respective patterns on the ordinary objects they encode.

Assuming that there are distinct patterns among the ordinary objects that are indistinguishable seems hardly justifiable. However, this relies on a particular understanding of what it means to encode patterns among the ordinary objects that may not be completely intuitive, as conceded in the next section.

However, our construction already shows that it is not necessary to justify the predecessor relation directly as a denoting relation: We can generalize the issue to the question of when abstract objects can be assured to be exemplification-distinguishable. In this more general question we no longer see any ties to Mathematics whatsoever, but rather a metaphysical discussion of the nature of abstract objects and relations among them.

Caveats of the Comprehension Principles

While the comprehension principles suggested above have some justification and allow for deriving that useful encoding conditions such as *numbering a property* can be abstracted to properties, they are not the only conceivable way of generically extending AOT with relations among abstract objects.

In particular, it does *not* follow from the suggested principles that any of the following properties denote:

- $[\lambda x\, \forall F\ (x[F] \rightarrow \Box\forall z\ ([F]z \rightarrow O!z))]$, i.e. *encoding only properties that are necessarily restricted to ordinary objects.*
- $[\lambda x\, x[\lambda z\ O!z\ \&\ \varphi\{z\}]]$, i.e. encoding a particular pattern among the ordinary objects.
- $[\lambda x\ \mathit{ExtensionOf}(x,[\lambda z\ O!z\ \&\ [G]z])]$ where $\mathit{ExtensionOf}(x,G)$ is defined by PLM as $\mathit{ExtensionOf}(x,G) \equiv_{df} A!x\ \&\ G{\downarrow}\ \&\ \forall F\ (x[F] \equiv \forall z\ ([F]z \equiv [G]z))$ (see A.13.298).

The notion of an *extension on the ordinary objects* we used above would have to be defined as (see A.13.300):

$$\mathit{OrdinaryExtensionOf}(x,G) \equiv_{df} A!x\ \&\ G{\downarrow}\ \&\ \forall F\ (x[F] \equiv \forall z\ (O!z \rightarrow ([F]z \equiv [G]z)))$$

With this definition, $[\lambda x\ \mathit{OrdinaryExtensionOf}(x,G)]{\downarrow}$ is derivable from the suggested principles (see A.13.303). However, using this conception of extensions on ordinary objects as the basis for our comprehension principles, has some potentially counter-intuitive implications:

If one abstract objects encodes exactly the property *being an ordinary table*, and another abstract object encodes exactly *being an ordinary table or being abstract*, our comprehension principles are not sufficient for telling them apart. Both objects involve the same pattern among the ordinary objects, but neither encodes it fully, since, for instance, neither encodes *being an ordinary table or being a natural number*, which also has the same exemplification pattern among the ordinary objects.

The third, combined principle alone cannot even distinguish between an object that encodes exactly *being an ordinary table* and an object that encodes exactly *being a mathematician* - neither of these objects are *fully determined by a pattern on the ordinary objects* in the sense of our principles, since neither encodes *all* properties with this pattern.[41]

Relation to Leibnizian Concepts and Platonic Forms

Despite the concessions above, our comprehension principles align well with the analysis of other philosophical objects in AOT. PLM defines for an abstract object to be the Leibnizian Concept of a property as follows (see A.13.412):[42]

$$\mathit{ConceptOf}(x,G) \equiv_{df} C!x \;\&\; (G\downarrow \;\&\; \forall F\ (x[F] \equiv G \Rightarrow F))$$

An object x is a concept of G, just in case it encodes exactly those properties that are necessarily implied by G, using the following definition of necessary implications between properties (see A.13.353):

$$F \Rightarrow G \equiv_{df} G\downarrow \;\&\; F\downarrow \;\&\; \Box\forall x\ ([F]x \to [G]x)$$

Now our comprehension principles make it derivable that *being a concept of* H is a property, if H necessarily implies *being ordinary* (see A.13.414):

$$H \Rightarrow O! \to [\lambda x\ \mathit{ConceptOf}(x,H)]\downarrow$$

So concepts of properties that do not involve abstract objects can always be distinguished from other abstract objects.

Reusing the example above, the concept of *being an ordinary table does* encode *being an ordinary table or being abstract*, since the former necessarily implies the latter. In fact it encodes all properties that are necessarily equivalent on the ordinary objects to *being an ordinary table*, since all those properties are necessarily implied by *being an ordinary table*.

Consequently, concepts of properties that necessarily imply being ordinary and possibly differ on some ordinary object become provably distinguishable. In particular, it becomes a *theorem* that the concept of *being an ordinary table* is discernible from the concept of *being a mathematician* (assuming that these properties are not necessarily exemplified by the same objects).

Further examples of theorems that can be derived from our comprehension principles are (see A.13.703 and A.13.720):

$$H \Rightarrow O! \to [\lambda x\ \mathbf{c}_H \preceq x]\downarrow$$

$$H \Rightarrow O! \to [\lambda x\ x \preceq \mathbf{c}_H]\downarrow$$

[41] However, they become distinguishable on the bases of the first principle above, since we can find a pattern among ordinary objects one of the abstract objects encodes, while the other one doesn't (assuming mathematicians aren't tables).

[42] *Being a concept* $C!$ is defined as $C! =_{df} A!$.

I.e. both *including* and *being included by* the concept of a property H denote, given that H necessarily implies being ordinary.[43]

Thick platonic forms are defined similarly to Leibnizian concepts of properties (see A.13.738):

$$FormOf(x,G) \equiv_{df} A!x \;\&\; G\downarrow \;\&\; \forall F\ (x[F] \equiv G \Rightarrow F)$$

So we can also derive that *being the (thick) platonic form of H* denotes a property, if H necessarily implies *being ordinary* (see A.13.740):

$$H \Rightarrow O! \rightarrow [\lambda x\ FormOf(x,H)]\downarrow$$

This shows that our comprehension principles are by no means *ad hoc* and have relevant implications for philosophical objects beyond the natural numbers. A detailed study of the implications of these principles will be an interesting topic for future research.

However, given the prospect of a move from abstracting patterns among *ordinary* objects to abstracting patterns among *discernible* objects instead, an even more interesting question may be whether similar general comprehension principles can be formulated for distinguishing objects that encode different patterns among *discernible* objects. We will discuss this further in section 5.21.

5.20. Modelling Possible Richness of Objects

Recall that the axiom of possible richness of objects was stated as follows (see A.12.4956):

$$\exists x\ (\mathbb{N}x \;\&\; x = \#G) \rightarrow \Diamond\exists y\ (E!y \;\&\; \forall u\ (\boldsymbol{\mathcal{A}}[G]u \rightarrow u \neq_E y))$$

Compared with the predecessor axiom, modelling possible richness of objects is straightforward. The axiom implies that there are countably infinitely many ordinary (even though potentially not *actually*, but merely *possibly* concrete) objects, so in our models we simply require there being a surjection from our type ω of ordinary urelements to Isabelle's type of natural numbers *nat*. While deriving the axiom from this change in the model is still non-trivial, we can prove (notably, our proof relies on the extended relation comprehension principles we introduced for modelling the predecessor relation as well as AOT's defined mathematical induction), that *being a natural number* in the models implies encoding only properties that are actually exemplified by only finitely many ordinary objects. Thereby, whenever a natural number numbers a property, it is only actually exemplified by a finite number of ordinary objects and since we have required infinitely many ordinary objects in our model, we can produce a witness to the claim of the axiom (modulo some further modal reasoning).

Furthermore, there is no way to model the axiom *without* extending the domain of ordinary objects in the model to infinitely many objects.

So for this axiom, the more interesting issue compared to modelling it is whether it can be philosophically justified as a purely logical axiom or not (see 5.17). While we

[43] The definitions of $\mathbf{c}_G$ (see A.13.551) and $\preceq$ (see A.13.408) can be found in appendix A.13, which implements fragments of the theory of concepts given in PLM chapter 13.

do not presume to judge whether the justification provided by Nodelman and Zalta in PLM item (799), resp. in [60], is sufficient to establish this axiom as purely logical, we certainly agree that it captures a natural and intuitive conception of *counting*.

Interestingly, however, it may be possible to eliminate the axiom altogether and more closely reproduce Frege's proof that every natural number has a successor as discussed in the next section.

5.21. Prospect of an Enhanced Version of the Construction

At the time of writing, there is an ongoing debate concerning variations of the analysis of natural numbers. In particular, instead of restricting the analysis to ordinary objects, identity on the ordinary objects and equinumerosity on the ordinary object, Nodelman and Zalta brought up the idea to instead follow the same basic construction relative to *discernible objects*.[44]

Being discernible, $D!$, can be defined as the following relation:

$$D! =_{df} [\lambda x \ \Box\forall\, y \ (y \neq x \rightarrow \exists\, F \ \neg([F]y \equiv [F]x))]$$

Using the necessary and sufficient conditions for relations to denote discussed in section 3.8.2, it can be shown that $D!$ denotes.[45] Furthermore, just as *being ordinary*, *being discernible* is a rigid restriction condition. Similar to $=_E$ on the ordinary objects, a relation of identity on the discernible objects $=_D$ can be defined as $[\lambda xy \ \Box\forall\, F \ ([F]x \equiv [F]y)]$, i.e. for discernible objects *being indistinguishable* implies identity. The construction up until the modal axiom of section 5.17 can be preserved without any major changes. Being equinumerous$_D$ can be defined just as equinumerous$_E$ (see section 5.2.3), but relative to a one-to-one correspondence$_D$ on discernible objects, which in turn can be defined just as a one-to-one correspondence$_E$ (see section 5.2.2), but using restricted variables ranging over discernible instead of ordinary objects.

The fact that *numbering a property* coexists with the predecessor relation described in section 5.19 is invariant under this change. Moreover, natural numbers will themselves become discernible (since by Hume's theorem for two objects numbering the same properties implies their identity). This allows for abandoning the modal axiom for possible richness of ordinary objects and instead to more closely follow Frege's construction, in which the successor of a number n is defined as the number of the property *being smaller-or-equal to* n, i.e. $n' = \#[\lambda m \ m \leq n]$, yielding $\mathbb{P}nn'$.

At the time of writing, we have prototypes for models of this new derivation available. In these models we restrict the domain of ordinary urelements to be at most countably infinite (i.e. either finite or in bijection to the natural numbers), and require the domain

[44] Personal correspondence of 14 October 2021 and 2 November 201. They are now revising the chapter on number theory on the basis of this idea.

[45] Note that due to the matrix involving a non-identity claim and identity on individuals being defined in terms of encoding, the λ-expression does not denote axiomatically.

of special urelements to be countably infinite.[46] From this restriction it can be derived that the class of cardinal numbers that measure the size of sets of discernible objects is itself a countable set.[47] Since abstract objects that number properties will be in one-to-one correspondence with the cardinals of sets of discernible urelements,[48] they can thus injectively be mapped into the special urelements, making them discernible. Hence this validates the theorem that *numbering a property* denotes and consequently yields models for the predecessor axiom.

As mentioned in section 5.19, it is an interesting question whether similar general comprehension principles can be formulated for distinguishing objects that encode different patterns among *discernible* objects, as we could suggest for patterns among *ordinary* objects. However, since discerning abstract object based on patterns among discernible objects yields new discernible objects, there is an increased danger of general comprehension principles for encoding patterns among discernible objects to become self-referential and thereby inconsistent. So while we expect to be able to formulate meta-theorems about the conditions under which it will be safe to assert the existence of relations among abstract objects that encode patterns among discernible objects,[49] it is unclear if we will be able to arrive at general comprehension principles that can be formulated in the theory itself.

In general, the price of being able to eliminate the modal axiom described in section 5.17 using the new construction will be that the predecessor axiom will become stronger and may have to rely on independent means of justification.

Another similar variant of the construction, for which we have already constructed full models (see [28]), does not restrict the domain of objects that can be counted at all, but instead of counting distinct objects rather counts equivalence classes of objects that are indistinguishable.[50] This involves weakening the unique existence used in one-to-one correspondences to uniqueness up to distinguishability, i.e. we define unique existence$_D$ as follows:

$$\exists!_D\alpha\ \varphi\{\alpha\} \equiv_{df} \exists\,\alpha\ (\varphi\{\alpha\}\ \&\ \forall\,\beta\ (\varphi\{\beta\} \rightarrow \beta =_D \alpha))$$

One-to-one correspondences and equinumerosity are then constructed relative to this restricted notion of unique existence:

$$R\ |\!:\ F \ {}_{1-1}\!\!\longleftrightarrow_D\ G \equiv$$
$$\forall\,x\ ([F]x \rightarrow \exists!_D y\ ([G]y\ \&\ [R]xy))\ \&\ \forall\,y\ ([G]y \rightarrow \exists!_D x\ ([F]x\ \&\ [R]xy))$$

$$F \approx_D G \equiv_{df} \exists\,R\ R\ |\!:\ F \ {}_{1-1}\!\!\longleftrightarrow_D\ G$$

[46] In a more general construction, it would be sufficient to require there being countably infinitely many special urelements that serve as proxies for discernible objects, while allowing an arbitrary number of special urelements for indiscernible objects.

[47] For every n, there is one cardinal number for finite sets of n discernibles, and additionally there is one cardinal for countably infinite sets of discernibles.

[48] In another variant mentioned below they will be in one-to-one correspondence with the cardinals of sets of arbitrary urelements.

[49] For example, any axiom that implies that certain abstract objects become discernible can be consistently modelled, as long as it discerns at most countably many abstract objects.

[50] I.e. indistinguishable objects belong to the same equivalence class and objects belonging to different equivalence classes are distinguishable.

While a construction based on discernible objects ignores objects that are indiscernible for the purpose of counting, i.e. a property that is exemplified by two indistinguishable abstract objects and no other objects is counted by Zero, this construction would count such objects in bulk, i.e. the same property would be counted by One. For properties that are only exemplified by discernible objects, both constructions are equivalent (i.e. such properties are equinumerous in the first variant if and only if they are equinumerous in the second).

5.22. Summary

In summary, we can conclude that the construction of natural numbers and the derivation of the Dedekind-Peano postulates given in PLM is provably sound. While the construction relies on additional axioms, we can say that:

- PLM can present reasonable justifications for both axioms.
- The predecessor axiom in the current construction can be generalized to comprehension principles that are independently justifiable, which strengthens the argument that the axiom is not intrinsically mathematical.
- In a future construction, the modal axiom of possible richness of objects may no longer be required, eliminating the need for its justification.
- It will be an interesting question for future research to determine whether the predecessor axiom can be similarly generalized in this future construction.

Methodologically, we can conclude that:

- Our embedding can successfully reproduce even complex constructions and reasoning in our target system AOT.
- We can achieve our goal to provide a natural and readable implementation that accurately reproduces syntax and reasoning in AOT without the need of keeping complex translations in mind.
- The automation infrastructure of Isabelle can be preserved even for complex constructions in the target system.
- Using our method we could provide insights into the construction and efficiently analyze potential extensions.

6. Higher-Order Object Theory

While the second-order fragment of AOT is expressive enough for a variety of applications, including applications in *natural mathematics*, as demonstrated in the last chapter at the example of the analysis of natural numbers, the theory can be generalized to a full type-theoretic higher-order version. A notable application of this generalized version of AOT is the analysis of *theoretical mathematics*.

While natural mathematics involves the construction of mathematical objects directly by abstracting exemplification patterns, and their properties are derived from the principles of AOT itself, theoretical mathematics involves analyzing mathematical theories themselves (as well as their objects, axioms and relations) as abstract objects.

While a full discussion of the type-theoretic version of AOT is beyond the scope of this thesis, this chapter will provide a short, informal overview of its construction and the challenges in constructing an embedding of it in Isabelle/HOL.

Note that while we reuse the notational conventions of our embedding for consistency with the last chapters (e.g. we use square brackets in exemplification and encoding formulas and the free-variable notation discussed in section 4.7.2), this chapter is *not* written relative to an Isabelle representation, so in contrast to the last chapters, none of the statements and terms are cited from an embedding. We forgo marking the statements in this chapter using vertical bars at the page margin.

6.1. Overview of Higher-Order Object Theory

Our description is based on an at the time of writing unpublished draft of a chapter of PLM. However, the full type-theoretic version of AOT is also already discussed in [61] and a simplified version serves as the basis of the upcoming paper *A Defense of Logicism* jointly authored by Hannes Leitgeb, Uri Nodelman and Edward Zalta (see [33]).

We already hinted at AOT's system of types in section 3.2. Formally, it involves the following types:

- i is a type.
- Whenever $t_1,\cdots,t_n$ are types ($n \geq 0$), $\langle t_1,\ldots,t_n\rangle$ is a type.

i is the primitive type of individuals, $\langle t_1,\ldots,t_n\rangle$ is the type of relations among n objects of the respective types $t_1,\ldots,t_n$. Zero-place relations, i.e. relations of type $\langle\rangle$, form the type of propositions. $\langle i\rangle$ is the type of properties among individuals. $\langle\langle i\rangle\rangle$ is the type of properties of properties of individuals. $\langle\langle i\rangle, \langle\rangle\rangle$ is the type of binary relations between properties and propositions, etc.

The distinction between exemplification and encoding is reproduced for higher-order types, i.e. the language involves exemplification formulas of the form $[\tau^{\langle t_1,\ldots,t_n\rangle}]\tau^{\tau_1}\ldots\tau^{\tau_n}$ and encoding formulas of the form $\tau^{\tau_1}\ldots\tau^{\tau_n}[\tau^{\langle t_1,\ldots,t_n\rangle}]$.

Furthermore, the distinction between ordinary and abstract objects is generalized to all types. I.e. for every type t there is a distinguished constant $E!^{\langle t\rangle}$ exemplified by all concrete objects of type t, which yields definitions of *being ordinary* and *being abstract* at every type.

While the definitions and axiom system are similar to the second-order version described in sections 3.2 and 3.3, there are some notable differences. The following is a non-exhaustive list:

- Relation identity for relations of type $\langle t\rangle$ is defined as:[1]
  $F = G \equiv_{df} ([O!]F\ \&\ [O!]G\ \&\ \Box\forall x(x[F] \equiv x[G])) \vee ([A!]F\ \&\ [A!]G\ \&\ \Box\forall \mathcal{H}(F[\mathcal{H}] \equiv G[\mathcal{H}]))$
- It is axiomatic that significant λ-expressions denote ordinary relations.
- η-conversion is restricted to ordinary relations.

Notably, the comprehension principle for abstract objects is retained at all types t. I.e. let α by of type t and F be of type $\langle t\rangle$, then the following is an axiom:

$$\exists\,\alpha([A!]\alpha\ \&\ \forall\,F(\alpha[F] \equiv \varphi\{F\}))$$

6.2. Applications to Theoretical Mathematics

The analysis of Theoretical Mathematics in higher-order object theory was described in [61] and a variant is discussed in [33].

While a full-discussion of the subtleties involved again goes beyond the scope of this thesis, we illustrate the general idea at the example of the representation of Zermelo-Fraenkel set-theory as an abstract object *ZF* in higher-order AOT.

Technically, a mathematical theory in AOT is a *situation*, i.e. an abstract object that encodes only propositional properties.[2] So we can reuse the notation $T \models p$ as the proposition p is true in theory T.

One of the cornerstones of the analysis is the *Importation Principle*, stated in [33] as follows:

> When φ is a closed theorem of T, then $T \models \varphi^*$ shall be an axiom, where φ^* is the result of indexing every occurrence of a term or predicate of T to T.

So taking S as ZF's property of *being a set*, it is a theorem of ZF that:

$$\vdash_{ZF} \neg\exists\,y([S]y\ \&\ y \in \emptyset)$$

This theorem can be imported to AOT using the following instance of the Importation Principle:

[1] n-ary relation identity for $n \geq 2$ and proposition identity are extended in a similar manner to account for abstract n-place relations, resp. propositions.

[2] Recall the discussion in section 3.5.2.

$$ZF \models \neg\exists\, y([S_{ZF}]y \ \&\ y \in_{ZF} \emptyset_{ZF})$$

Furthermore, the involved indexed terms of ZF are in turn abstract objects in AOT, e.g.

$$\emptyset_{ZF} = \iota x([A!]x \ \&\ \forall\, F(x[F] \equiv ZF \models [F]\emptyset_{ZF})$$

$$S_{ZF} = \iota F([A!]F \ \&\ \forall\, \mathcal{F}(F[\mathcal{F}] \equiv ZF \models [\mathcal{F}]S_{ZF})$$

$$\in_{ZF} = \iota R([A!]R \ \&\ \forall\, \mathcal{R}(R[\mathcal{R}] \equiv ZF \models [\mathcal{R}]\in_{ZF})$$

Exemplifying properties in ZF can be translated to encoding claims in AOT. E.g. in ZF, $\emptyset$ exemplifies the property $[\lambda x\ \neg\exists\, y([S]y \ \&\ y \in x)]$. This property can be captured as an *abstract property* in AOT that is *encoded* by $\emptyset_{ZF}$:[3]

$$\emptyset_{ZF}[[\lambda x\ \neg\exists\, y([S_{SF}]y \ \&\ y \in_{ZF} x)]_{ZF}]$$

While a detailed account of the construction and its implications is the topic of the upcoming paper [33], we will discuss the general issue of embedding higher-order AOT in Isabelle/HOL in the next sections.

6.3. Bounded Models

[33] constructs minimal extensional models for the simplified version of higher-order AOT it uses for its argumentation. This construction defines the *height* of a type t, written $h(t)$, and the *width* of a type t, written $w(t)$ as:

- $h(i) = 0$
- $h(\langle\rangle) = 1$
- $h(\langle t_1,\ldots,t_n\rangle) = 1 + max\{h(t_1),\ldots,h(t_n)\}$
- $w(i) = 1$
- $w(\langle\rangle) = 1$
- $w(\langle t_1, \ldots, t_n\rangle) = \sum_1^k w(t_k)$

[33] then presents a concrete model construction for bounded languages $\mathcal{L}_{n,m}$ that are *cut off* at width n and height m, i.e. the well-formed expressions of the language $\mathcal{L}_{n,m}$ are the expressions of the unbounded language $\mathcal{L}$ in which only terms of type t are well-formed, if $w(t) \leq n$ and $h(t) \leq m$. In particular, types of height m only involve ordinary objects, not abstract objects. For example, the second-order fragment described in the last chapters, is cut off at height *1*: while it involves abstract individuals, all relations and propositions are ordinary. Furthermore, while the second-order fragment considers properties of objects (height 1), it does not consider higher-order relations like properties of properties or properties of propositions.[4]

While we expect it to be feasible to construct a representation in Isabelle/HOL that allows for an arbitrary parameter as cut-off in height (and potentially width, though it may be possible to keep width unbounded), we expect the details of such a construction

[3] While λ-expressions in higher-order AOT are ordinary, theory-indexed λ-expressions are abstract.

[4] Note that the cut-off involves subtle changes in the precise formulation of the definitions and the axiom system.

to be non-trivial due to the non-uniform nature of the representation sets of types. We leave the construction of such an embedding to future research.

6.4. Abstract Objects in Unbounded Models

While, arguably, a construction of models for higher-order object theory with a fixed, but arbitrary cutoff may be sufficient for all intents and purposes, the issue of constructing unbounded models (resp. an unrestricted embedding of higher-order AOT in HOL) is nevertheless interesting: theoretically, it may provide insights into the relative strength of higher-order AOT compared to HOL. Technically, unbounded models have the advantage of being uniform in all types, which is beneficial for a generic implementation.

However, if we consider the extent of the generalized comprehension principle of abstract objects and the identity conditions of abstract objects, it becomes clear that the construction of such models is not trivial.

In particular, note that the comprehension principle for abstract individuals has the following instance:

$$\exists\, x\ ([A!]x\ \&\ \forall\, F\ (x[F] \equiv ([O!]F\ \&\ \varphi\{F\} \vee [A!]F\ \&\ \forall\, \mathcal{F}\ (F[\mathcal{F}] \equiv \psi\{\mathcal{F}\}))))$$

Such an abstract object x (at type i) encodes all ordinary properties F (at type $\langle i\rangle$) that satisfy an arbitrary condition φ and all abstract properties F that encode exactly those properties of properties $\mathcal{F}$ (at type $\langle\langle i\rangle\rangle$) that satisfy an arbitrary condition ψ on $\mathcal{F}$.

Now for two such abstract objects (at type i) to be identical, they not only have to encode the same ordinary properties (at type $\langle i\rangle$), but also the same abstract properties (at type $\langle i\rangle$). Those abstract properties in turn are identical, if they encode the same properties of properties (at type $\langle\langle i\rangle\rangle$).

This can be iterated further, since there are also abstract properties of properties among individuals that may encode properties of properties of properties among individuals, etc. pp.

While we leave a more detailed and rigorous analysis to future research, we try to informally illustrate the expected size of the set of abstract objects in unbounded models.

Thinking in terms of Aczel models, let O_t be the set of ordinary objects at type t and S_t the set of special urelements of type t. Now the set of relations among objects of type t, i.e. $O_{\langle t\rangle}$ will be at least as large as the power set $\mathcal{P}(O_t \cup S_t)$. For simplicity, we consider minimal, extensional Aczel models, in which we have $O_{\langle t\rangle} = \mathcal{P}(O_t \cup S_t)$.

If we restrict ourselves to unary relations and write 0 for the type of ordinary individuals i, 1 for the type of relations among individuals $\langle i\rangle$ and so on, i.e. in general we choose $n+1$ for unary relations among the type we identified with n, we get the following:

$O_1 = \mathcal{P}(O_0 \cup S_0)$
$O_2 = \mathcal{P}(O_1 \cup S_1)$
$O_3 = \mathcal{P}(O_2 \cup S_2)$
$\ldots$

Now if we, solely for the purpose of arriving at a crude size estimate, further assume O_0 is empty and $S_i = S_0 = S$, we get:

$O_0 = \emptyset$
$O_1 = \mathcal{P}(O_0 \cup S) = \mathcal{P}(S)$
$O_2 = \mathcal{P}(O_1 \cup S) = \mathcal{P}(\mathcal{P}(S) \cup S) \supseteq \mathcal{P}(\mathcal{P}(S)) \cup \mathcal{P}(S)$
$O_3 = \mathcal{P}(O_2 \cup S) = \mathcal{P}(\mathcal{P}(\mathcal{P}(S) \cup S) \cup S) \supseteq \mathcal{P}(\mathcal{P}(\mathcal{P}(S))) \cup \mathcal{P}(\mathcal{P}(S)) \cup \mathcal{P}(S)$
...

Now if we assume that S has only one element and identify it with $\mathcal{P}(\emptyset)$, and (informally for the purpose of illustrating) consider the limit O_ω of relations at countably infinite height, we arrive at a model of the natural numbers, i.e. $|O_\omega| \geq |\mathbb{N}|$.

The set of abstract objects at type *m − 1* is the power set of ordinary and abstract objects of type *m*, i.e. $A_{m-1} = \mathcal{P}(O_m \cup A_m)$. So we get:

$A_{m-1} = \mathcal{P}(O_m \cup A_m)$
$A_{m-2} = \mathcal{P}(O_{m-1} \cup A_{m-1}) = \mathcal{P}(O_{m-1} \cup \mathcal{P}(O_m \cup A_m))$
$A_{m-3} = \mathcal{P}(O_{m-2} \cup A_{m-2}) = \mathcal{P}(O_{m-2} \cup \mathcal{P}(O_{m-1} \cup \mathcal{P}(O_m \cup A_m)))$
...
$A_0 = \mathcal{P}(O_1 \cup \mathcal{P}(O_2 \cup \mathcal{P}(O_3 \cup \mathcal{P}(\ldots \cup A_m)\ldots)))$

In particular, no finite application of power set operations is enough to construct A_0 from the (illustrative) limit set A_ω, which in turn would be the power set of O_ω, i.e. of a set at least as large as the natural numbers.

While this informal argument may not hold up to scrutiny, it is safe to say that the set of abstract objects in an unbounded model of higher-order object theory will be huge. We wouldn't be surprised if a future more rigorous analysis were to conclude that the set of abstract individuals in non-trivial models of higher-order AOT had to be sufficiently large to form a model of ZF itself (resp. that the cardinality of A_0 is strongly inaccessible).

Consequently, a verifiably sound implementation relative to the unextended background theory of Isabelle/HOL may be challenging, since the expressive power of higher-order AOT may be on par with or even exceed the expressive power of this choice of a meta-logic. However, even if this turns out to be the case, it may be possible to construct a representation based on a stronger extension of Isabelle/HOL, for example HOLZF [40], which axiomatizes the ZF universe itself as a type in HOL. The feasibility of such an embedding as well as the question of the relative strength of higher-order object theory compared to HOL, are interesting questions for future research.

7. Conclusion

We have presented an implementation of a foundational metaphysical theory in an automated reasoning environment by leveraging and extending the concept of *shallow semantic embeddings* (SSEs) in classical higher-order logic.

Methodologically, we could demonstrate that:

- The SSE approach is scalable and can not only be used to analyze isolated arguments, but can also be applied to full metaphysical theories.
- We can construct an accurate implementation of the axioms and deductive system of the target theory using abstraction layers.
- The automation infrastructure of Isabelle/HOL can be preserved and applied to construct proofs that accurately correspond to derivations in the target system.

While some constructions and modes of reasoning in a target system may be challenging to reproduce in an embedding, we developed several techniques to address such cases, including the definition of custom theorem attributes and proving methods and the extension of Isabelle's Isar language by specialized outer syntax commands. Furthermore, we devised a system of syntax translations on a custom sub-grammar of Isabelle's inner syntax to construct an accurate representation of the syntax of our target theory.

Using these techniques, it is not only possible to technically reproduce the logic of a target theory, but also to construct a nearly transparent representation of its syntax and reasoning flow. This allows for an efficient and effortless exchange of results between traditional pen-and-paper based reasoning and the computerized implementation.

This way, we can effectively arrive at a dedicated automated theorem proving environment for our target system, while retaining a verifiably consistent meta-logical backend.

The construction of such a framework is not merely a technical exercise, but can trigger a fruitful exchange that, in our case, led to significant improvements of the analyzed theory itself.

In particular, in the application of our method to the second-order fragment of *Abstract Object Theory* (AOT), we could demonstrate that:

- A semantic implementation can serve as a flexible backend that can be used to explore variations and axiomatic extensions of the target system.
- Our semantic analysis could significantly contribute especially to the theoretical understanding of the conditions, in AOT, under which relations exist. This has led to considerable improvements in the formulation of AOT.

- We can verify complex constructions and reasoning within a given axiomatization of the target system and efficiently analyze the effects of variations and extensions of such constructions.

Concretely, we can confirm that AOT can serve as a sound basis for a variant of Frege's construction of natural numbers. We can verify that the Dedekind-Peano postulates thus become consistently derivable in AOT.

We could contribute to the evolution of this construction and provide insights into the nature of its required additional axioms, and into variants of the construction. This includes a generalization of one of the axioms that may serve to strengthen the philosophical justification of the construction.

Interestingly, our results simultaneously support the use of HOL as universal meta-logic in that we can demonstrate that the SSE approach can be used to accurately represent even challenging logical theories, while our results also strengthen the position of our target theory AOT as foundational system in confirming its ability to provide a philosophically grounded construction of mathematical objects.

In this context, an attempt of an implementation of the full type-theoretic higher-order version of AOT using the SSE approach, as well as the formal analysis of its relative strength compared to HOL and ZF are fascinating opportunities for future research.

A. Isabelle Theory

A.1. Model for the Logic of AOT

```
(*<*)
theory AOT_model
  imports Main "HOL-Cardinals.Cardinals"
begin

declare[[typedef_overloaded]]
(*>*)

section‹Model for the Logic of AOT›

text‹We introduce a primitive type for hyperintensional propositions.›
typedecl o

text‹To be able to model modal operators following Kripke semantics,
     we introduce a primitive type for possible worlds and assert, by axiom,
     that there is a surjective function mapping propositions to the
     boolean-valued functions acting on possible worlds. We call the result
     of applying this function to a proposition the Montague intension
     of the proposition.›
typedecl w ―‹The primtive type of possible worlds.›
axiomatization AOT_model_do :: ‹o⇒(w⇒bool)› where
  do_surj: ‹surj AOT_model_do›

text‹The axioms of PLM require the existence of a non-actual world.›
consts w0 :: w ―‹The designated actual world.›
axiomatization where AOT_model_nonactual_world: ‹∃w . w ≠ w0›

text‹Validity of a proposition in a given world can now be modelled as the result
     of applying that world to the Montague intension of the proposition.›
definition AOT_model_valid_in :: ‹w⇒o⇒bool› where
  ‹AOT_model_valid_in w φ ≡ AOT_model_do φ w›

text‹By construction, we can choose a proposition for any given Montague intension,
     s.t. the proposition is valid in a possible world iff the Montague intension
     evaluates to true at that world.›
definition AOT_model_proposition_choice :: ‹(w⇒bool) ⇒ o› (binder ‹εo › 8)
  where ‹εo w. φ w ≡ (inv AOT_model_do) φ›
lemma AOT_model_proposition_choice_simp: ‹AOT_model_valid_in w (εo w. φ w) = φ w›
  by (simp add: surj_f_inv_f[OF do_surj] AOT_model_valid_in_def
                AOT_model_proposition_choice_def)

text‹Nitpick can trivially show that there are models for the axioms above.›
lemma ‹True› nitpick[satisfy, user_axioms, expect = genuine] ..

typedecl ω ―‹The primtive type of ordinary objects/urelements.›

text‹Validating extended relation comprehension requires a large set of
     special urelements. For simple models that do not validate extended
     relation comprehension (and consequently the predecessor axiom in the
     theory of natural numbers), it suffices to use a primitive type as @{text σ},
     i.e. @{theory_text ‹typedecl σ›}.›
typedecl σ'
typedef σ = ‹UNIV::((ω ⇒ w ⇒ bool) set × (ω ⇒ w ⇒ bool) set × σ') set› ..
```

```

typedecl null — ‹Null-urelements representing non-denoting terms.›

datatype υ = ωυ ω | συ σ | is_nullυ: nullυ null — ‹Type of urelements›

text‹Urrelations are proposition-valued functions on urelements.
     Urrelations are required to evaluate to necessarily false propositions for
     null-urelements (note that there may be several distinct necessarily false
     propositions).›
typedef urrel = ‹{ φ . ∀ x w . ¬AOT_model_valid_in w (φ (nullυ x)) }›
  by (rule exI[where x=‹λ x . (εo w . ¬is_nullυ x)›])
     (auto simp: AOT_model_proposition_choice_simp)

text‹Abstract objects will be modelled as sets of urrelations and will
     have to be mapped surjectively into the set of special urelements.
     We show that any mapping from abstract objects to special urelements
     has to involve at least one large set of collapsed abstract objects.
     We will use this fact to extend arbitrary mappings from abstract objects
     to special urelements to surjective mappings.›
lemma ασ_pigeonhole:
  — ‹For any arbitrary mapping @{term ασ} from sets of urrelations to special
     urelements, there exists an abstract object x, s.t. the cardinal of the set
     of special urelements is strictly smaller than the cardinal of the set of
     abstract objects that are mapped to the same urelement as x under @{term ασ}.›
  ‹∃x . |UNIV::σ set| <o |{y . ασ x = ασ y}|›
  for ασ :: ‹urrel set ⇒ σ›
proof(rule ccontr)
  have card_σ_set_set_bound: ‹|UNIV::σ set set| ≤o |UNIV::urrel set|›
  proof -
    let ?pick = ‹λu s . εo w . case u of (συ s') ⇒ s' ∈ s | _ ⇒ False›
    have ‹∃f :: σ set ⇒ urrel . inj f›
    proof
      show ‹inj (λs . Abs_urrel (λu . ?pick u s))›
      proof(rule injI)
        fix x y
        assume ‹Abs_urrel (λu. ?pick u x) = Abs_urrel (λu. ?pick u y)›
        hence ‹(λu. ?pick u x) = (λu. ?pick u y)›
          by (auto intro!: Abs_urrel_inject[THEN iffD1]
                     simp: AOT_model_proposition_choice_simp)
        hence ‹AOT_model_valid_in w0 (?pick (συ s) x) =
                  AOT_model_valid_in w0 (?pick (συ s) y)›
          for s by metis
        hence ‹(s ∈ x) = (s ∈ y)› for s
          by (auto simp: AOT_model_proposition_choice_simp)
        thus ‹x = y›
          by blast
      qed
    qed
    thus ?thesis
      by (metis card_of_image inj_imp_surj_inv)
  qed

  text‹Assume, for a proof by contradiction, that there is no large collapsed set.›
  assume ‹∄x . |UNIV::σ set| <o |{y . ασ x = ασ y}|›
  hence A: ‹∀x . |{y . ασ x = ασ y}| ≤o |UNIV::σ set|›
    by auto
  have union_univ: ‹(⋃x ∈ range(inv ασ) . {y . ασ x = ασ y}) = UNIV›
    by auto (meson f_inv_into_f range_eqI)

  text‹We refute by case distinction: there is either finitely many or
       infinitely many special urelements and in both cases we can derive
       a contradiction from the assumption above.›
  {
```

```
text‹Finite case.›
assume finite_σ_set: ‹finite (UNIV::σ set)›
hence finite_collapsed: ‹finite {y . ασ x = ασ y}› for x
  using A card_of_ordLeq_infinite by blast
hence 0: ‹∀x . card {y . ασ x = ασ y} ≤ card (UNIV::σ set)›
  by (metis A finite_σ_set card_of_ordLeq inj_on_iff_card_le)
have 1: ‹card (range (inv ασ)) ≤ card (UNIV::σ set)›
  using finite_σ_set card_image_le by blast
hence 2: ‹finite (range (inv ασ))›
  using finite_σ_set by blast

define n where ‹n = card (UNIV::urrel set set)›
define m where ‹m = card (UNIV::σ set)›

have ‹n = card (⋃x ∈ range(inv ασ) . {y . ασ x = ασ y})›
  unfolding n_def using union_univ by argo
also have ‹... ≤ (∑i∈range (inv ασ). card {y. ασ i = ασ y})›
  using card_UN_le 2 by blast
also have ‹... ≤ (∑i∈range (inv ασ). card (UNIV::σ set))›
  by (metis (no_types, lifting) 0 sum_mono)
also have ‹... ≤ card (range (inv ασ)) * card (UNIV::σ set)›
  using sum_bounded_above by auto
also have ‹... ≤ card (UNIV::σ set) * card (UNIV::σ set)›
  using 1 by force
also have ‹... = m*m›
  unfolding m_def by blast
finally have n_upper: ‹n ≤ m*m›.

have ‹finite (⋃x ∈ range(inv ασ) . {y . ασ x = ασ y})›
  using 2 finite_collapsed by blast
hence finite_αset: ‹finite (UNIV::urrel set set)›
  using union_univ by argo

have ‹2^2^m = (2::nat)^(card (UNIV::σ set set))›
  by (metis Pow_UNIV card_Pow finite_σ_set m_def)
moreover have ‹card (UNIV::σ set set) ≤ (card (UNIV::urrel set))›
  using card_σ_set_set_bound
  by (meson Finite_Set.finite_set card_of_ordLeq finite_αset
            finite_σ_set inj_on_iff_card_le)
ultimately have ‹2^2^m ≤ (2::nat)^(card (UNIV:: urrel set))›
  by simp
also have ‹... = n›
  unfolding n_def
  by (metis Finite_Set.finite_set Pow_UNIV card_Pow finite_αset)
finally have ‹2^2^m ≤ n› by blast
hence ‹2^2^m ≤ m*m› using n_upper by linarith
moreover {
  have ‹(2::nat)^(2^m) ≥ (2^(m + 1))›
    by (metis Suc_eq_plus1 Suc_leI less_exp one_le_numeral power_increasing)
  also have ‹(2^(m + 1)) = (2::nat) * 2^m›
    by auto
  have ‹m < 2^m›
    by (simp add: less_exp)
  hence ‹m*m < (2^m)*(2^m)›
    by (simp add: mult_strict_mono)
  moreover have ‹... = 2^(m+m)›
    by (simp add: power_add)
  ultimately have ‹m*m < 2 ^ (m + m)› by presburger
  moreover have ‹m+m ≤ 2^m›
  proof (induct m)
    case 0
    thus ?case by auto
  next
```

```
        case (Suc m)
        thus ?case
          by (metis Suc_leI less_exp mult_2 mult_le_mono2 power_Suc)
      qed
      ultimately have ‹m*m < 2^2^m›
        by (meson less_le_trans one_le_numeral power_increasing)
    }
    ultimately have False by auto
  }
  moreover {
    text‹Infinite case.›
    assume ‹infinite (UNIV::σ set)›
    hence Cinfσ: ‹Cinfinite |UNIV::σ set|›
      by (simp add: cinfinite_def)
    have 1: ‹|range (inv ασ)| ≤o |UNIV::σ set|›
      by auto
    have 2: ‹∀i∈range (inv ασ). |{y . ασ i = ασ y}| ≤o |UNIV::σ set|›
    proof
      fix i :: ‹urrel set›
      assume ‹i ∈ range (inv ασ)›
      show ‹|{y . ασ i = ασ y}| ≤o |UNIV::σ set|›
        using A by blast
    qed
    have ‹|⋃ ((λi. {y. ασ i = ασ y}) ` (range (inv ασ)))| ≤o
                |Sigma (range (inv ασ)) (λi. {y. ασ i = ασ y})|›
      using card_of_UNION_Sigma by blast
    hence ‹|UNIV::urrel set set| ≤o
            |Sigma (range (inv ασ)) (λi. {y. ασ i = ασ y})|›
      using union_univ by argo
    moreover have ‹|Sigma (range (inv ασ)) (λi. {y. ασ i = ασ y})| ≤o |UNIV::σ set|›
      using card_of_Sigma_ordLeq_Cinfinite[OF Cinfσ, OF 1, OF 2] by blast
    ultimately have ‹|UNIV::urrel set set| ≤o |UNIV::σ set|›
      using ordLeq_transitive by blast
    moreover {
      have ‹|UNIV::σ set| <o |UNIV::σ set set|›
        by auto
      moreover have ‹|UNIV::σ set set| ≤o |UNIV::urrel set|›
        using card_σ_set_set_bound by blast
      moreover have ‹|UNIV::urrel set| <o |UNIV::urrel set set|›
        by auto
      ultimately have ‹|UNIV::σ set| <o |UNIV::urrel set set|›
        by (metis ordLess_imp_ordLeq ordLess_ordLeq_trans)
    }
    ultimately have False
      using not_ordLeq_ordLess by blast
  }
  ultimately show False by blast
qed

text‹We introduce a mapping from abstract objects (i.e. sets of urrelations) to
     special urelements @{text ‹ασ›} that is surjective and distinguishes all
     abstract objects that are distinguished by a (not necessarily surjective)
     mapping @{text ‹ασ'›}. @{text ‹ασ'›} will be used to model extended relation
     comprehension.›
consts ασ' :: ‹urrel set ⇒ σ›
consts ασ :: ‹urrel set ⇒ σ›

specification(ασ)
  ασ_surj: ‹surj ασ›
  ασ_ασ': ‹ασ x = ασ y ⟹ ασ' x = ασ' y›
proof -
  obtain x where x_prop: ‹|UNIV::σ set| <o |{y. ασ' x = ασ' y}|›
    using ασ_pigeonhole by blast
```

```
have ‹∃f :: urrel set ⇒ σ . f ‘ {y. ασ' x = ασ' y} = UNIV ∧ f x = ασ' x›
proof -
  have ‹∃f :: urrel set ⇒ σ . f ‘ {y. ασ' x = ασ' y} = UNIV›
    by (simp add: x_prop card_of_ordLeq2 ordLess_imp_ordLeq)
  then obtain f :: ‹urrel set ⇒ σ› where ‹f ‘ {y. ασ' x = ασ' y} = UNIV›
    by presburger
  moreover obtain a where ‹f a = ασ' x› and ‹ασ' a = ασ' x›
    by (smt (verit, best) calculation UNIV_I image_iff mem_Collect_eq)
  ultimately have ‹(f (a := f x, x := f a)) ‘ {y. ασ' x = ασ' y} = UNIV ∧
                  (f (a := f x, x := f a)) x = ασ' x›
    by (auto simp: image_def)
  thus ?thesis by blast
qed
then obtain f where fimage: ‹f ‘ {y. ασ' x = ασ' y} = UNIV›
                and fx: ‹f x = ασ' x›
  by blast

define ασ :: ‹urrel set ⇒ σ› where
  ‹ασ ≡ λ urrels . if ασ' urrels = ασ' x ∧ f urrels ∉ range ασ'
                   then f urrels
                   else ασ' urrels›
have ‹surj ασ›
proof -
  {
  fix s :: σ
  {
    assume ‹s ∈ range ασ'›
    hence 0: ‹ασ' (inv ασ' s) = s›
      by (meson f_inv_into_f)
    {
      assume ‹s = ασ' x›
      hence ‹ασ x = s›
        using ασ_def fx by presburger
      hence ‹∃f . ασ (f s) = s›
        by auto
    }
    moreover {
      assume ‹s ≠ ασ' x›
      hence ‹ασ (inv ασ' s) = s›
        unfolding ασ_def 0 by presburger
      hence ‹∃f . ασ (f s) = s›
        by blast
    }
    ultimately have ‹∃f . ασ (f s) = s›
      by blast
  }
  moreover {
    assume ‹s ∉ range ασ'›
    moreover obtain urrels where ‹f urrels = s› and ‹ασ' x = ασ' urrels›
      by (smt (verit, best) UNIV_I fimage image_iff mem_Collect_eq)
    ultimately have ‹ασ urrels = s›
      using ασ_def by presburger
    hence ‹∃f . ασ (f s) = s›
      by (meson f_inv_into_f range_eqI)
  }
  ultimately have ‹∃f . ασ (f s) = s›
    by blast
  }
  thus ?thesis
    by (metis surj_def)
qed
moreover have ‹∀x y. ασ x = ασ y ⟶ ασ' x = ασ' y›
  by (metis ασ_def rangeI)
```

```
  ultimately show ?thesis
    by blast
qed

text‹For extended models that validate extended relation comprehension
     (and consequently the predecessor axiom), we specify which
     abstract objects are distinguished by @{const ασ'}.›

definition urrel_to_ωrel :: ‹urrel ⇒ (ω ⇒ w ⇒ bool)› where
  ‹urrel_to_ωrel ≡ λ r u w . AOT_model_valid_in w (Rep_urrel r (ωυ u))›
definition ωrel_to_urrel :: ‹(ω ⇒ w ⇒ bool) ⇒ urrel› where
  ‹ωrel_to_urrel ≡ λ φ . Abs_urrel
    (λ u . ε○ w . case u of ωυ x ⇒ φ x w | _ ⇒ False)›

definition AOT_urrel_ωequiv :: ‹urrel ⇒ urrel ⇒ bool› where
  ‹AOT_urrel_ωequiv ≡ λ r s . ∀ u v . AOT_model_valid_in v (Rep_urrel r (ωυ u)) =
                                      AOT_model_valid_in v (Rep_urrel s (ωυ u))›

lemma urrel_ωrel_quot: ‹Quotient3 AOT_urrel_ωequiv urrel_to_ωrel ωrel_to_urrel›
proof(rule Quotient3I)
  show ‹urrel_to_ωrel (ωrel_to_urrel a) = a› for a
    unfolding ωrel_to_urrel_def urrel_to_ωrel_def
    apply (rule ext)
    apply (subst Abs_urrel_inverse)
    by (auto simp: AOT_model_proposition_choice_simp)
next
  show ‹AOT_urrel_ωequiv (ωrel_to_urrel a) (ωrel_to_urrel a)› for a
    unfolding ωrel_to_urrel_def AOT_urrel_ωequiv_def
    apply (subst (1 2) Abs_urrel_inverse)
    by (auto simp: AOT_model_proposition_choice_simp)
next
  show ‹AOT_urrel_ωequiv r s = (AOT_urrel_ωequiv r r ∧ AOT_urrel_ωequiv s s ∧
                                urrel_to_ωrel r = urrel_to_ωrel s)› for r s
  proof
    assume ‹AOT_urrel_ωequiv r s›
    hence ‹AOT_model_valid_in v (Rep_urrel r (ωυ u)) =
           AOT_model_valid_in v (Rep_urrel s (ωυ u))› for u v
      using AOT_urrel_ωequiv_def by metis
    hence ‹urrel_to_ωrel r = urrel_to_ωrel s›
      unfolding urrel_to_ωrel_def
      by simp
    thus ‹AOT_urrel_ωequiv r r ∧ AOT_urrel_ωequiv s s ∧
          urrel_to_ωrel r = urrel_to_ωrel s›
      unfolding AOT_urrel_ωequiv_def
      by auto
  next
    assume ‹AOT_urrel_ωequiv r r ∧ AOT_urrel_ωequiv s s ∧
            urrel_to_ωrel r = urrel_to_ωrel s›
    hence ‹AOT_model_valid_in v (Rep_urrel r (ωυ u)) =
           AOT_model_valid_in v (Rep_urrel s (ωυ u))› for u v
      by (metis urrel_to_ωrel_def)
    thus ‹AOT_urrel_ωequiv r s›
      using AOT_urrel_ωequiv_def by presburger
  qed
qed

specification (ασ')
  ασ_eq_ord_exts_all:
    ‹ασ' a = ασ' b ⟹ (⋀s . urrel_to_ωrel s = urrel_to_ωrel r ⟹ s ∈ a)
      ⟹ (⋀ s . urrel_to_ωrel s = urrel_to_ωrel r ⟹ s ∈ b)›
  ασ_eq_ord_exts_ex:
    ‹ασ' a = ασ' b ⟹ (∃ s . s ∈ a ∧ urrel_to_ωrel s = urrel_to_ωrel r)
      ⟹ (∃s . s ∈ b ∧ urrel_to_ωrel s = urrel_to_ωrel r)›
```

```
proof -
  define ασ_wit_intersection where
      ‹ασ_wit_intersection ≡ λ urrels .
        {ordext . ∀urrel . urrel_to_ωrel urrel = ordext ⟶ urrel ∈ urrels}›
  define ασ_wit_union where
      ‹ασ_wit_union ≡ λ urrels .
        {ordext . ∃urrel∈urrels . urrel_to_ωrel urrel = ordext}›

  let ?ασ_wit = ‹λ urrels .
    let ordexts = ασ_wit_intersection urrels in
    let ordexts' = ασ_wit_union urrels in
    (ordexts, ordexts', undefined)›
  define ασ_wit :: ‹urrel set ⇒ σ› where
    ‹ασ_wit ≡ λ urrels . Abs_σ (?ασ_wit urrels)›
  {
    fix a b :: ‹urrel set› and r s
    assume ‹ασ_wit a = ασ_wit b›
    hence 0: ‹{ordext. ∀urrel. urrel_to_ωrel urrel = ordext ⟶ urrel ∈ a} =
              {ordext. ∀urrel. urrel_to_ωrel urrel = ordext ⟶ urrel ∈ b}›
      unfolding ασ_wit_def Let_def
      apply (subst (asm) Abs_σ_inject)
      by (auto simp: ασ_wit_intersection_def ασ_wit_union_def)
    assume ‹urrel_to_ωrel s = urrel_to_ωrel r ⟹ s ∈ a› for s
    hence ‹urrel_to_ωrel r ∈
             {ordext. ∀urrel. urrel_to_ωrel urrel = ordext ⟶ urrel ∈ a}›
      by auto
    hence ‹urrel_to_ωrel r ∈
             {ordext. ∀urrel. urrel_to_ωrel urrel = ordext ⟶ urrel ∈ b}›
      using 0 by blast
    moreover assume ‹urrel_to_ωrel s = urrel_to_ωrel r›
    ultimately have ‹s ∈ b›
      by blast
  }
  moreover {
    fix a b :: ‹urrel set› and s r
    assume ‹ασ_wit a = ασ_wit b›
    hence 0: ‹{ordext. ∃urrel ∈ a. urrel_to_ωrel urrel = ordext} =
              {ordext. ∃urrel ∈ b. urrel_to_ωrel urrel = ordext}›
      unfolding ασ_wit_def
      apply (subst (asm) Abs_σ_inject)
      by (auto simp: Let_def ασ_wit_intersection_def ασ_wit_union_def)
    assume ‹s ∈ a›
    hence ‹urrel_to_ωrel s ∈ {ordext. ∃urrel ∈ a. urrel_to_ωrel urrel = ordext}›
      by blast
    moreover assume ‹urrel_to_ωrel s = urrel_to_ωrel r›
    ultimately have ‹urrel_to_ωrel r ∈
                       {ordext. ∃urrel ∈ b. urrel_to_ωrel urrel = ordext}›
      using "0" by argo
    hence ‹∃s. s ∈ b ∧ urrel_to_ωrel s = urrel_to_ωrel r›
      by blast
  }
  ultimately show ?thesis
    by (safe intro!: exI[where x=ασ_wit]; metis)
qed

text‹We enable the extended model version.›
abbreviation (input) AOT_ExtendedModel where ‹AOT_ExtendedModel ≡ True›

text‹Individual terms are either ordinary objects, represented by ordinary urelements,
     abstract objects, modelled as sets of urrelations, or null objects, used to
     represent non-denoting definite descriptions.›
datatype κ = ωκ ω | ακ ‹urrel set› | is_nullκ: nullκ null
```

```
text‹The mapping from abstract objects to urelements can be naturally
     lifted to a surjective mapping from individual terms to urelements.›
primrec κυ :: ‹κ⇒υ› where
  ‹κυ (ωκ x) = ωυ x›
| ‹κυ (ακ x) = συ (ασ x)›
| ‹κυ (nullκ x) = nullυ x›

lemma κυ_surj: ‹surj κυ›
  using ασ_surj by (metis κυ.simps(1) κυ.simps(2) κυ.simps(3) υ.exhaust surj_def)

text‹By construction if the urelement of an individual term is exemplified by
     an urrelation, it cannot be a null-object.›
lemma urrel_null_false:
  assumes ‹AOT_model_valid_in w (Rep_urrel f (κυ x))›
  shows ‹¬is_nullκ x›
  by (metis (mono_tags, lifting) assms Rep_urrel κ.collapse(3) κυ.simps(3)
        mem_Collect_eq)

text‹AOT requires any ordinary object to be @{emph ‹possibly concrete›} and that
     there is an object that is not actually, but possibly concrete.›
consts AOT_model_concreteω :: ‹ω ⇒ w ⇒  bool›
specification (AOT_model_concreteω)
  AOT_model_ω_concrete_in_some_world:
  ‹∃ w . AOT_model_concreteω x w›
  AOT_model_contingent_object:
  ‹∃ x w . AOT_model_concreteω x w ∧ ¬AOT_model_concreteω x w₀›
  by (rule exI[where x=‹λ_ w. w ≠ w₀›]) (auto simp: AOT_model_nonactual_world)

text‹We define a type class for AOT's terms specifying the conditions under which
     objects of that type denote and require the set of denoting terms to be
     non-empty.›
class AOT_Term =
  fixes AOT_model_denotes :: ‹'a ⇒ bool›
  assumes AOT_model_denoting_ex: ‹∃ x . AOT_model_denotes x›

text‹All types except the type of propositions involve non-denoting terms. We
     define a refined type class for those.›
class AOT_IncompleteTerm = AOT_Term +
  assumes AOT_model_nondenoting_ex: ‹∃ x . ¬AOT_model_denotes x›

text‹Generic non-denoting term.›
definition AOT_model_nondenoting :: ‹'a::AOT_IncompleteTerm› where
  ‹AOT_model_nondenoting ≡ SOME τ . ¬AOT_model_denotes τ›
lemma AOT_model_nondenoing: ‹¬AOT_model_denotes (AOT_model_nondenoting)›
  using someI_ex[OF AOT_model_nondenoting_ex]
  unfolding AOT_model_nondenoting_def by blast

text‹@{const AOT_model_denotes} can trivially be extended to products of types.›
instantiation prod :: (AOT_Term, AOT_Term) AOT_Term
begin
definition AOT_model_denotes_prod :: ‹'a×'b ⇒ bool› where
  ‹AOT_model_denotes_prod ≡ λ(x,y) . AOT_model_denotes x ∧ AOT_model_denotes y›
instance proof
  show ‹∃x::'a×'b. AOT_model_denotes x›
    by (simp add: AOT_model_denotes_prod_def AOT_model_denoting_ex)
qed
end

text‹We specify a transformation of proposition-valued functions on terms, s.t.
     the result is fully determined by @{emph ‹regular›} terms. This will be required
     for modelling n-ary relations as functions on tuples while preserving AOT's
     definition of n-ary relation identity.›
locale AOT_model_irregular_spec =
```

```
  fixes AOT_model_irregular :: ‹('a ⇒ o) ⇒ 'a ⇒ o›
    and AOT_model_regular :: ‹'a ⇒ bool›
    and AOT_model_term_equiv :: ‹'a ⇒ 'a ⇒ bool›
  assumes AOT_model_irregular_false:
    ‹¬AOT_model_valid_in w (AOT_model_irregular φ x)›
  assumes AOT_model_irregular_equiv:
    ‹AOT_model_term_equiv x y ⟹
     AOT_model_irregular φ x = AOT_model_irregular φ y›
  assumes AOT_model_irregular_eqI:
    ‹(⋀ x . AOT_model_regular x ⟹ φ x = ψ x) ⟹
     AOT_model_irregular φ x = AOT_model_irregular ψ x›

text‹We introduce a type class for individual terms that specifies being regular,
     being equivalent (i.e. conceptually @{emph ‹sharing urelements›}) and the
     transformation on proposition-valued functions as specified above.›
class AOT_IndividualTerm = AOT_IncompleteTerm +
  fixes AOT_model_regular :: ‹'a ⇒ bool›
  fixes AOT_model_term_equiv :: ‹'a ⇒ 'a ⇒ bool›
  fixes AOT_model_irregular :: ‹('a ⇒ o) ⇒ 'a ⇒ o›
  assumes AOT_model_irregular_nondenoting:
    ‹¬AOT_model_regular x ⟹ ¬AOT_model_denotes x›
  assumes AOT_model_term_equiv_part_equivp:
    ‹equivp AOT_model_term_equiv›
  assumes AOT_model_term_equiv_denotes:
    ‹AOT_model_term_equiv x y ⟹ (AOT_model_denotes x = AOT_model_denotes y)›
  assumes AOT_model_term_equiv_regular:
    ‹AOT_model_term_equiv x y ⟹ (AOT_model_regular x = AOT_model_regular y)›
  assumes AOT_model_irregular:
    ‹AOT_model_irregular_spec AOT_model_irregular AOT_model_regular
                              AOT_model_term_equiv›

interpretation AOT_model_irregular_spec AOT_model_irregular AOT_model_regular
                                         AOT_model_term_equiv
  using AOT_model_irregular .

text‹Our concrete type for individual terms satisfies the type class of
     individual terms.
     Note that all unary individuals are regular. In general, an individual term
     may be a tuple and is regular, if at most one tuple element does not denote.›
instantiation κ :: AOT_IndividualTerm
begin
definition AOT_model_term_equiv_κ :: ‹κ ⇒ κ ⇒ bool› where
  ‹AOT_model_term_equiv_κ ≡ λ x y . κυ x = κυ y›
definition AOT_model_denotes_κ :: ‹κ ⇒ bool› where
  ‹AOT_model_denotes_κ ≡ λ x . ¬is_nullκ x›
definition AOT_model_regular_κ :: ‹κ ⇒ bool› where
  ‹AOT_model_regular_κ ≡ λ x . True›
definition AOT_model_irregular_κ :: ‹(κ ⇒ o) ⇒ κ ⇒ o› where
  ‹AOT_model_irregular_κ ≡ SOME φ . AOT_model_irregular_spec φ
                                   AOT_model_regular AOT_model_term_equiv›
instance proof
  show ‹∃x :: κ. AOT_model_denotes x›
    by (rule exI[where x=‹ωκ undefined›])
       (simp add: AOT_model_denotes_κ_def)
next
  show ‹∃x :: κ. ¬AOT_model_denotes x›
    by (rule exI[where x=‹nullκ undefined›])
       (simp add: AOT_model_denotes_κ_def AOT_model_regular_κ_def)
next
  show "¬AOT_model_regular x ⟹ ¬ AOT_model_denotes x" for x :: κ
    by (simp add: AOT_model_regular_κ_def)
next
  show ‹equivp (AOT_model_term_equiv :: κ ⇒ κ ⇒ bool)›
```

```
    by (rule equivpI; rule reflpI exI sympI transpI)
       (simp_all add: AOT_model_term_equiv_κ_def)
next
  fix x y :: κ
  show ‹AOT_model_term_equiv x y ⟹ AOT_model_denotes x = AOT_model_denotes y›
    by (metis AOT_model_denotes_κ_def AOT_model_term_equiv_κ_def κ.exhaust_disc
              κυ.simps υ.disc(1,3,5,6) is_ακ_def is_ωκ_def is_nullκ_def)
next
  fix x y :: κ
  show ‹AOT_model_term_equiv x y ⟹ AOT_model_regular x = AOT_model_regular y›
    by (simp add: AOT_model_regular_κ_def)
next
  have "AOT_model_irregular_spec (λ φ (x::κ) .  ε_o w . False)
          AOT_model_regular AOT_model_term_equiv"
    by standard (auto simp: AOT_model_proposition_choice_simp)
  thus ‹AOT_model_irregular_spec (AOT_model_irregular::(κ⇒o) ⇒ κ ⇒ o)
          AOT_model_regular AOT_model_term_equiv›
    unfolding AOT_model_irregular_κ_def by (metis (no_types, lifting) someI_ex)
qed
end

text‹We define relations among individuals as proposition valued functions.
     @{emph ‹Denoting›} unary relations (among @{typ κ}) will match the
     urrelations introduced above.›
typedef 'a rel (‹<_>›) = ‹UNIV::('a::AOT_IndividualTerm ⇒ o) set› ..
setup_lifting type_definition_rel

text‹We will use the transformation specified above to "fix" the behaviour of
     functions on irregular terms when defining @{text ‹λ›}-expressions.›
definition fix_irregular :: ‹('a::AOT_IndividualTerm ⇒ o) ⇒ ('a ⇒ o)› where
  ‹fix_irregular ≡ λ φ x . if AOT_model_regular x
                           then φ x else AOT_model_irregular φ x›
lemma fix_irregular_denoting:
  ‹AOT_model_denotes x ⟹ fix_irregular φ x = φ x›
  by (meson AOT_model_irregular_nondenoting fix_irregular_def)
lemma fix_irregular_regular:
  ‹AOT_model_regular x ⟹ fix_irregular φ x = φ x›
  by (meson AOT_model_irregular_nondenoting fix_irregular_def)
lemma fix_irregular_irregular:
  ‹¬AOT_model_regular x ⟹ fix_irregular φ x = AOT_model_irregular φ x›
  by (simp add: fix_irregular_def)

text‹Relations among individual terms are (potentially non-denoting) terms.
     A relation denotes, if it agrees on all equivalent terms (i.e. terms sharing
     urelements), is necessarily false on all non-denoting terms and is
     well-behaved on irregular terms.›
instantiation rel :: (AOT_IndividualTerm) AOT_IncompleteTerm
begin
text‹\linelabel{AOT_model_denotes_rel}›
lift_definition AOT_model_denotes_rel :: ‹<'a> ⇒ bool› is
  ‹λ φ . (∀ x y . AOT_model_term_equiv x y ⟶ φ x = φ y) ∧
         (∀ w x . AOT_model_valid_in w (φ x) ⟶ AOT_model_denotes x) ∧
         (∀ x . ¬AOT_model_regular x ⟶ φ x = AOT_model_irregular φ x)› .
instance proof
  have ‹AOT_model_irregular (fix_irregular φ) x = AOT_model_irregular φ x›
    for φ and x :: 'a
    by (rule AOT_model_irregular_eqI) (simp add: fix_irregular_def)
  thus ‹∃ x :: <'a> . AOT_model_denotes x›
    by (safe intro!: exI[where x=‹Abs_rel (fix_irregular (λx. ε_o w . False))›])
       (transfer; auto simp: AOT_model_proposition_choice_simp fix_irregular_def
                             AOT_model_irregular_equiv AOT_model_term_equiv_regular
                             AOT_model_irregular_false)
next
```

```
  show ‹∃f :: <'a> . ¬AOT_model_denotes f›
    by (rule exI[where x=‹Abs_rel (λx. ε∘ w . True)›];
        auto simp: AOT_model_denotes_rel.abs_eq AOT_model_nondenoting_ex
                   AOT_model_proposition_choice_simp)
qed
end

text‹Auxiliary lemmata.›

lemma AOT_model_term_equiv_eps:
  shows ‹AOT_model_term_equiv (Eps (AOT_model_term_equiv κ)) κ›
    and ‹AOT_model_term_equiv κ (Eps (AOT_model_term_equiv κ))›
    and ‹AOT_model_term_equiv κ κ' ⟹
         (Eps (AOT_model_term_equiv κ)) = (Eps (AOT_model_term_equiv κ'))›
  apply (metis AOT_model_term_equiv_part_equivp equivp_def someI_ex)
  apply (metis AOT_model_term_equiv_part_equivp equivp_def someI_ex)
  by (metis AOT_model_term_equiv_part_equivp equivp_def)

lemma AOT_model_denotes_Abs_rel_fix_irregularI:
  assumes ‹⋀ x y . AOT_model_term_equiv x y ⟹ φ x = φ y›
      and ‹⋀ w x . AOT_model_valid_in w (φ x) ⟹ AOT_model_denotes x›
    shows ‹AOT_model_denotes (Abs_rel (fix_irregular φ))›
proof -
  have ‹AOT_model_irregular φ x = AOT_model_irregular
         (λx. if AOT_model_regular x then φ x else AOT_model_irregular φ x) x›
    if ‹¬ AOT_model_regular x›
    for x
    by (rule AOT_model_irregular_eqI) auto
  thus ?thesis
  unfolding AOT_model_denotes_rel.rep_eq
  using assms by (auto simp: AOT_model_irregular_false Abs_rel_inverse
                             AOT_model_irregular_equiv fix_irregular_def
                             AOT_model_term_equiv_regular)
qed

lemma AOT_model_term_equiv_rel_equiv:
  assumes ‹AOT_model_denotes x›
      and ‹AOT_model_denotes y›
    shows ‹AOT_model_term_equiv x y = (∀ Π w . AOT_model_denotes Π ⟶
           AOT_model_valid_in w (Rep_rel Π x) = AOT_model_valid_in w (Rep_rel Π y))›
proof
  assume ‹AOT_model_term_equiv x y›
  thus ‹∀ Π w . AOT_model_denotes Π ⟶ AOT_model_valid_in w (Rep_rel Π x) =
                                      AOT_model_valid_in w (Rep_rel Π y)›
    by (simp add: AOT_model_denotes_rel.rep_eq)
next
  have 0: ‹(AOT_model_denotes x' ∧ AOT_model_term_equiv x' y) =
          (AOT_model_denotes y' ∧ AOT_model_term_equiv y' y)›
    if ‹AOT_model_term_equiv x' y'› for x' y'
    by (metis that AOT_model_term_equiv_denotes AOT_model_term_equiv_part_equivp
              equivp_def)
  assume ‹∀ Π w . AOT_model_denotes Π ⟶ AOT_model_valid_in w (Rep_rel Π x) =
                                        AOT_model_valid_in w (Rep_rel Π y)›
  moreover have ‹AOT_model_denotes (Abs_rel (fix_irregular
    (λ x . ε∘ w . AOT_model_denotes x ∧ AOT_model_term_equiv x y)))›
    (is "AOT_model_denotes ?r")
    by (rule AOT_model_denotes_Abs_rel_fix_irregularI)
       (auto simp: 0 AOT_model_denotes_rel.rep_eq Abs_rel_inverse fix_irregular_def
                   AOT_model_proposition_choice_simp AOT_model_irregular_false)
  ultimately have ‹AOT_model_valid_in w (Rep_rel ?r x) =
                   AOT_model_valid_in w (Rep_rel ?r y)› for w
    by blast
  thus ‹AOT_model_term_equiv x y›
```

```
    by (simp add: Abs_rel_inverse AOT_model_proposition_choice_simp
                  fix_irregular_denoting[OF assms(1)] AOT_model_term_equiv_part_equivp
                  fix_irregular_denoting[OF assms(2)] assms equivp_reflp)
qed

text‹Denoting relations among terms of type @{typ κ} correspond to urrelations.›

definition rel_to_urrel :: ‹<κ> ⇒ urrel› where
  ‹rel_to_urrel ≡ λ Π . Abs_urrel (λ u . Rep_rel Π (SOME x . κυ x = u))›
definition urrel_to_rel :: ‹urrel ⇒ <κ>› where
  ‹urrel_to_rel ≡ λ φ . Abs_rel (λ x . Rep_urrel φ (κυ x))›
definition AOT_rel_equiv :: ‹<'a::AOT_IndividualTerm> ⇒ <'a> ⇒ bool› where
  ‹AOT_rel_equiv ≡ λ f g . AOT_model_denotes f ∧ AOT_model_denotes g ∧ f = g›

lemma urrel_quotient3: ‹Quotient3 AOT_rel_equiv rel_to_urrel urrel_to_rel›
proof (rule Quotient3I)
  have ‹(λu. Rep_urrel a (κυ (SOME x. κυ x = u))) = (λu. Rep_urrel a u)› for a
    by (rule ext) (metis (mono_tags, lifting) κυ_surj surj_f_inv_f verit_sko_ex')
  thus ‹rel_to_urrel (urrel_to_rel a) = a› for a
    by (simp add: Abs_rel_inverse rel_to_urrel_def urrel_to_rel_def
                  Rep_urrel_inverse)
next
  show ‹AOT_rel_equiv (urrel_to_rel a) (urrel_to_rel a)› for a
    unfolding AOT_rel_equiv_def urrel_to_rel_def
    by transfer (simp add: AOT_model_regular_κ_def AOT_model_denotes_κ_def
                           AOT_model_term_equiv_κ_def urrel_null_false)
next
  {
    fix a
    assume ‹∀w x. AOT_model_valid_in w (a x) ⟶ ¬ is_nullκ x›
    hence ‹(λu. a (SOME x. κυ x = u)) ∈
           {φ. ∀x w. ¬ AOT_model_valid_in w (φ (nullυ x))}›
      by (simp; metis (mono_tags, lifting) κ.exhaust_disc κυ.simps υ.disc(1,3,5)
                                          υ.disc(6) is_ακ_def is_ωκ_def someI_ex)
  } note 1 = this
  {
    fix r s :: ‹κ ⇒ o›
    assume A: ‹∀x y. AOT_model_term_equiv x y ⟶ r x = r y›
    assume ‹∀w x. AOT_model_valid_in w (r x) ⟶ AOT_model_denotes x›
    hence 2: ‹(λu. r (SOME x. κυ x = u)) ∈
              {φ. ∀x w. ¬ AOT_model_valid_in w (φ (nullυ x))}›
      using 1 AOT_model_denotes_κ_def by meson
    assume B: ‹∀x y. AOT_model_term_equiv x y ⟶ s x = s y›
    assume ‹∀w x. AOT_model_valid_in w (s x) ⟶ AOT_model_denotes x›
    hence 3: ‹(λu. s (SOME x. κυ x = u)) ∈
              {φ. ∀x w. ¬ AOT_model_valid_in w (φ (nullυ x))}›
      using 1 AOT_model_denotes_κ_def by meson
    assume ‹Abs_urrel (λu. r (SOME x. κυ x = u)) =
            Abs_urrel (λu. s (SOME x. κυ x = u))›
    hence 4: ‹r (SOME x. κυ x = u) = s (SOME x::κ. κυ x = u)› for u
      unfolding Abs_urrel_inject[OF 2 3] by metis
    have ‹r x = s x› for x
      using 4[of ‹κυ x›]
      by (metis (mono_tags, lifting) A B AOT_model_term_equiv_κ_def someI_ex)
    hence ‹r = s› by auto
  }
  thus ‹AOT_rel_equiv r s = (AOT_rel_equiv r r ∧ AOT_rel_equiv s s ∧
                             rel_to_urrel r = rel_to_urrel s)› for r s
    unfolding AOT_rel_equiv_def rel_to_urrel_def
    by transfer auto
qed

lemma urrel_quotient:
```

```
  ‹Quotient AOT_rel_equiv rel_to_urrel urrel_to_rel
            (λx y. AOT_rel_equiv x x ∧ rel_to_urrel x = y)›
  using Quotient3_to_Quotient[OF urrel_quotient3] by auto

text‹Unary individual terms are always regular and equipped with encoding and
     concreteness. The specification of the type class anticipates the required
     properties for deriving the axiom system.›
class AOT_UnaryIndividualTerm =
  fixes AOT_model_enc :: ‹'a ⇒ <'a::AOT_IndividualTerm> ⇒ bool›
    and AOT_model_concrete :: ‹w ⇒ 'a ⇒ bool›
  assumes AOT_model_unary_regular:
      ‹AOT_model_regular x› ― ‹All unary individual terms are regular.›
      and AOT_model_enc_relid:
        ‹AOT_model_denotes F ⟹
         AOT_model_denotes G ⟹
         (⋀ x . AOT_model_enc x F ⟷ AOT_model_enc x G)
         ⟹ F = G›
      and AOT_model_A_objects:
        ‹∃x . AOT_model_denotes x ∧
              (∀w. ¬ AOT_model_concrete w x) ∧
              (∀F. AOT_model_denotes F ⟶ AOT_model_enc x F = φ F)›
      and AOT_model_contingent:
        ‹∃ x w. AOT_model_concrete w x ∧ ¬ AOT_model_concrete w₀ x›
      and AOT_model_nocoder:
        ‹AOT_model_concrete w x ⟹ ¬AOT_model_enc x F›
      and AOT_model_concrete_equiv:
        ‹AOT_model_term_equiv x y ⟹
          AOT_model_concrete w x = AOT_model_concrete w y›
      and AOT_model_concrete_denotes:
        ‹AOT_model_concrete w x ⟹ AOT_model_denotes x›
      ― ‹The following are properties that will only hold in the extended models.›
      and AOT_model_enc_indistinguishable_all:
       ‹AOT_ExtendedModel ⟹
        AOT_model_denotes a ⟹ ¬(∃ w . AOT_model_concrete w a) ⟹
        AOT_model_denotes b ⟹ ¬(∃ w . AOT_model_concrete w b) ⟹
        AOT_model_denotes Π ⟹
        (⋀ Π' . AOT_model_denotes Π' ⟹
          (⋀ v . AOT_model_valid_in v (Rep_rel Π' a) =
                 AOT_model_valid_in v (Rep_rel Π' b))) ⟹
        (⋀ Π' . AOT_model_denotes Π' ⟹
            (⋀ v x . ∃ w . AOT_model_concrete w x ⟹
                AOT_model_valid_in v (Rep_rel Π' x) =
                AOT_model_valid_in v (Rep_rel Π x)) ⟹
            AOT_model_enc a Π') ⟹
        (⋀ Π' . AOT_model_denotes Π' ⟹
            (⋀ v x . ∃ w . AOT_model_concrete w x ⟹
                AOT_model_valid_in v (Rep_rel Π' x) =
                AOT_model_valid_in v (Rep_rel Π x)) ⟹
            AOT_model_enc b Π')›
      and AOT_model_enc_indistinguishable_ex:
       ‹AOT_ExtendedModel ⟹
        AOT_model_denotes a ⟹ ¬(∃ w . AOT_model_concrete w a) ⟹
        AOT_model_denotes b ⟹ ¬(∃ w . AOT_model_concrete w b) ⟹
        AOT_model_denotes Π ⟹
        (⋀ Π' . AOT_model_denotes Π' ⟹
          (⋀ v . AOT_model_valid_in v (Rep_rel Π' a) =
                 AOT_model_valid_in v (Rep_rel Π' b))) ⟹
        (∃ Π' . AOT_model_denotes Π' ∧ AOT_model_enc a Π' ∧
            (∀ v x . (∃ w . AOT_model_concrete w x) ⟶
                AOT_model_valid_in v (Rep_rel Π' x) =
                AOT_model_valid_in v (Rep_rel Π x))) ⟹
        (∃ Π' . AOT_model_denotes Π' ∧ AOT_model_enc b Π' ∧
            (∀ v x . (∃ w . AOT_model_concrete w x) ⟶
```

```
                AOT_model_valid_in v (Rep_rel Π' x) =
                AOT_model_valid_in v (Rep_rel Π x)))›

text‹Instantiate the class of unary individual terms for our concrete type of
     individual terms @{typ κ}.›
instantiation κ :: AOT_UnaryIndividualTerm
begin

definition AOT_model_enc_κ :: ‹κ ⇒ <κ> ⇒ bool› where
  ‹AOT_model_enc_κ ≡ λ x F .
      case x of ακ a ⇒ AOT_model_denotes F ∧ rel_to_urrel F ∈ a
             | _ ⇒ False›
primrec AOT_model_concrete_κ :: ‹w ⇒ κ ⇒ bool› where
  ‹AOT_model_concrete_κ w (ωκ x) = AOT_model_concreteω x w›
| ‹AOT_model_concrete_κ w (ακ x) = False›
| ‹AOT_model_concrete_κ w (nullκ x) = False›

lemma AOT_meta_A_objects_κ:
  ‹∃x :: κ. AOT_model_denotes x ∧
            (∀w. ¬ AOT_model_concrete w x) ∧
            (∀F. AOT_model_denotes F ⟶ AOT_model_enc x F = φ F)› for φ
  apply (rule exI[where x=‹ακ {f . φ (urrel_to_rel f)}›])
  apply (simp add: AOT_model_enc_κ_def AOT_model_denotes_κ_def)
  by (metis (no_types, lifting) AOT_rel_equiv_def urrel_quotient
                                Quotient_rep_abs_fold_unmap)

instance proof
  show ‹AOT_model_regular x› for x :: κ
    by (simp add: AOT_model_regular_κ_def)
next
  fix F G :: ‹<κ>›
  assume ‹AOT_model_denotes F›
  moreover assume ‹AOT_model_denotes G›
  moreover assume ‹⋀x. AOT_model_enc x F = AOT_model_enc x G›
  moreover obtain x where ‹∀G. AOT_model_denotes G ⟶ AOT_model_enc x G = (F = G)›
    using AOT_meta_A_objects_κ by blast
  ultimately show ‹F = G› by blast
next
  show ‹∃x :: κ. AOT_model_denotes x ∧
                 (∀w. ¬ AOT_model_concrete w x) ∧
                 (∀F. AOT_model_denotes F ⟶ AOT_model_enc x F = φ F)› for φ
    using AOT_meta_A_objects_κ .
next
  show ‹∃ (x::κ) w. AOT_model_concrete w x ∧ ¬ AOT_model_concrete w0 x›
    using AOT_model_concrete_κ.simps(1) AOT_model_contingent_object by blast
next
  show ‹AOT_model_concrete w x ⟹ ¬ AOT_model_enc x F› for w and x :: κ and F
    by (metis AOT_model_concrete_κ.simps(2) AOT_model_enc_κ_def κ.case_eq_if
              κ.collapse(2))
next
  show ‹AOT_model_concrete w x = AOT_model_concrete w y›
    if ‹AOT_model_term_equiv x y›
    for x y :: κ and w
    using that by (induct x; induct y; auto simp: AOT_model_term_equiv_κ_def)
next
  show ‹AOT_model_concrete w x ⟹ AOT_model_denotes x› for w and x :: κ
    by (metis AOT_model_concrete_κ.simps(3) AOT_model_denotes_κ_def κ.collapse(3))
(* Extended models only *)
next
  fix κ κ' :: κ and Π Π' :: ‹<κ>› and w :: w
  assume ext: ‹AOT_ExtendedModel›
  assume ‹AOT_model_denotes κ›
  moreover assume ‹∄w. AOT_model_concrete w κ›
```

```
ultimately obtain a where a_def: ‹ακ a = κ›
  by (metis AOT_model_ω_concrete_in_some_world AOT_model_concrete_κ.simps(1)
            AOT_model_denotes_κ_def κ.discI(3) κ.exhaust_sel)
assume ‹AOT_model_denotes κ'›
moreover assume ‹∄w. AOT_model_concrete w κ'›
ultimately obtain b where b_def: ‹ακ b = κ'›
  by (metis AOT_model_ω_concrete_in_some_world AOT_model_concrete_κ.simps(1)
            AOT_model_denotes_κ_def κ.discI(3) κ.exhaust_sel)
assume ‹AOT_model_denotes Π' ⟹ AOT_model_valid_in w (Rep_rel Π' κ) =
                                 AOT_model_valid_in w (Rep_rel Π' κ')› for Π' w
hence ‹AOT_model_valid_in w (Rep_urrel r (κυ κ)) =
       AOT_model_valid_in w (Rep_urrel r (κυ κ'))› for r
  by (metis AOT_rel_equiv_def Abs_rel_inverse Quotient3_rel_rep
            iso_tuple_UNIV_I urrel_quotient3 urrel_to_rel_def)
hence ‹let r = (Abs_urrel (λ u . ε₀ w . u = κυ κ)) in
       AOT_model_valid_in w (Rep_urrel r (κυ κ)) =
       AOT_model_valid_in w (Rep_urrel r (κυ κ'))›
  by presburger
hence ασ_eq: ‹ασ a = ασ b›
  unfolding Let_def
  apply (subst (asm) (1 2) Abs_urrel_inverse)
  using AOT_model_proposition_choice_simp a_def b_def by force+
assume Π_den: ‹AOT_model_denotes Π›
have ‹∃r . ∀ x . Rep_rel Π (ωκ x) = Rep_urrel r (ωυ x)›
  apply (rule exI[where x=‹rel_to_urrel Π›])
  apply auto
  unfolding rel_to_urrel_def
  apply (subst Abs_urrel_inverse)
  apply auto
   apply (metis (mono_tags, lifting) AOT_model_denotes_κ_def
            AOT_model_denotes_rel.rep_eq κ.exhaust_disc κυ.simps(1,2,3)
            ‹AOT_model_denotes Π› υ.disc(8,9) υ.distinct(3)
            is_ακ_def is_ωκ_def verit_sko_ex')
  by (metis (mono_tags, lifting) AOT_model_denotes_rel.rep_eq
        AOT_model_term_equiv_κ_def κυ.simps(1) Π_den verit_sko_ex')
then obtain r where r_prop: ‹Rep_rel Π (ωκ x) = Rep_urrel r (ωυ x)› for x
  by blast
assume ‹AOT_model_denotes Π' ⟹
  (⋀v x. ∃w. AOT_model_concrete w x ⟹
      AOT_model_valid_in v (Rep_rel Π' x) =
      AOT_model_valid_in v (Rep_rel Π x)) ⟹ AOT_model_enc κ Π'› for Π'
hence ‹AOT_model_denotes Π' ⟹
  (⋀v x. AOT_model_valid_in v (Rep_rel Π' (ωκ x)) =
          AOT_model_valid_in v (Rep_rel Π (ωκ x))) ⟹ AOT_model_enc κ Π'› for Π'
  by (metis AOT_model_concrete_κ.simps(2) AOT_model_concrete_κ.simps(3)
           κ.exhaust_disc is_ακ_def is_ωκ_def is_nullκ_def)
hence ‹(⋀v x. AOT_model_valid_in v (Rep_urrel r (ωυ x)) =
             AOT_model_valid_in v (Rep_rel Π (ωκ x))) ⟹ r ∈ a› for r
  unfolding a_def[symmetric] AOT_model_enc_κ_def apply simp
  by (smt (verit, best) AOT_rel_equiv_def Abs_rel_inverse Quotient3_def
          κυ.simps(1) iso_tuple_UNIV_I urrel_quotient3 urrel_to_rel_def)
hence ‹(⋀v x. AOT_model_valid_in v (Rep_urrel r' (ωυ x)) =
             AOT_model_valid_in v (Rep_urrel r (ωυ x))) ⟹ r' ∈ a› for r'
  unfolding r_prop.
hence ‹⋀s. urrel_to_ωrel s = urrel_to_ωrel r ⟹ s ∈ a›
  by (metis urrel_to_ωrel_def)
hence 0: ‹⋀s. urrel_to_ωrel s = urrel_to_ωrel r ⟹ s ∈ b›
  using ασ_eq_ord_exts_all ασ_eq ext ασ_ασ' by blast

assume Π'_den: ‹AOT_model_denotes Π'›
assume ‹∃w. AOT_model_concrete w x ⟹ AOT_model_valid_in v (Rep_rel Π' x) =
                                     AOT_model_valid_in v (Rep_rel Π x)› for v x
hence ‹AOT_model_valid_in v (Rep_rel Π' (ωκ x)) =
```

```
        AOT_model_valid_in v (Rep_rel Π (ωκ x))› for v x
    using AOT_model_ω_concrete_in_some_world AOT_model_concrete_κ.simps(1)
    by presburger
  hence ‹AOT_model_valid_in v (Rep_urrel (rel_to_urrel Π')) (ωυ x)) =
        AOT_model_valid_in v (Rep_urrel r (ωυ x))› for v x
    by (smt (verit, best) AOT_rel_equiv_def Abs_rel_inverse Quotient3_def
        κυ.simps(1) iso_tuple_UNIV_I r_prop urrel_quotient3 urrel_to_rel_def Π'_den)
  hence ‹urrel_to_ωrel (rel_to_urrel Π') = urrel_to_ωrel r›
    by (metis (full_types) AOT_urrel_ωequiv_def Quotient3_def urrel_ωrel_quot)
  hence ‹rel_to_urrel Π' ∈ b› using 0 by blast
  thus ‹AOT_model_enc κ' Π'›
    unfolding b_def[symmetric] AOT_model_enc_κ_def by (auto simp: Π'_den)
next
  fix κ κ' :: κ and Π Π' :: ‹<κ>› and w :: w
  assume ext: ‹AOT_ExtendedModel›
  assume ‹AOT_model_denotes κ›
  moreover assume ‹∄w. AOT_model_concrete w κ›
  ultimately obtain a where a_def: ‹ακ a = κ›
    by (metis AOT_model_ω_concrete_in_some_world AOT_model_concrete_κ.simps(1)
         AOT_model_denotes_κ_def κ.discI(3) κ.exhaust_sel)
  assume ‹AOT_model_denotes κ'›
  moreover assume ‹∄w. AOT_model_concrete w κ'›
  ultimately obtain b where b_def: ‹ακ b = κ'›
    by (metis AOT_model_ω_concrete_in_some_world AOT_model_concrete_κ.simps(1)
             AOT_model_denotes_κ_def κ.discI(3) κ.exhaust_sel)
  assume ‹AOT_model_denotes Π' ⟹ AOT_model_valid_in w (Rep_rel Π' κ) =
                                  AOT_model_valid_in w (Rep_rel Π' κ')› for Π' w
  hence ‹AOT_model_valid_in w (Rep_urrel r (κυ κ)) =
        AOT_model_valid_in w (Rep_urrel r (κυ κ'))› for r
    by (metis AOT_rel_equiv_def Abs_rel_inverse Quotient3_rel_rep
             iso_tuple_UNIV_I urrel_quotient3 urrel_to_rel_def)
  hence ‹let r = (Abs_urrel (λ u . ε∘ w . u = κυ κ)) in
        AOT_model_valid_in w (Rep_urrel r (κυ κ)) =
        AOT_model_valid_in w (Rep_urrel r (κυ κ'))›
    by presburger
  hence ασ_eq: ‹ασ a = ασ b›
    unfolding Let_def
    apply (subst (asm) (1 2) Abs_urrel_inverse)
    using AOT_model_proposition_choice_simp a_def b_def by force+
  assume Π_den: ‹AOT_model_denotes Π›
  have ‹∃r . ∀ x . Rep_rel Π (ωκ x) = Rep_urrel r (ωυ x)›
    apply (rule exI[where x=‹rel_to_urrel Π›])
    apply auto
    unfolding rel_to_urrel_def
    apply (subst Abs_urrel_inverse)
    apply auto
     apply (metis (mono_tags, lifting) AOT_model_denotes_κ_def
          AOT_model_denotes_rel.rep_eq κ.exhaust_disc κυ.simps(1,2,3)
          ‹AOT_model_denotes Π› υ.disc(8) υ.disc(9) υ.distinct(3)
          is_ακ_def is_ωκ_def verit_sko_ex')
    by (metis (mono_tags, lifting) AOT_model_denotes_rel.rep_eq
          AOT_model_term_equiv_κ_def κυ.simps(1) Π_den verit_sko_ex')
  then obtain r where r_prop: ‹Rep_rel Π (ωκ x) = Rep_urrel r (ωυ x)› for x
    by blast

  assume ‹∃Π'. AOT_model_denotes Π' ∧
    AOT_model_enc κ Π' ∧
    (∀v x. (∃w. AOT_model_concrete w x) ⟶ AOT_model_valid_in v (Rep_rel Π' x) =
                                          AOT_model_valid_in v (Rep_rel Π x))›
  then obtain Π' where
      Π'_den: ‹AOT_model_denotes Π'› and
      κ_enc_Π': ‹AOT_model_enc κ Π'› and
      Π'_prop: ‹∃w. AOT_model_concrete w x ⟹
```

```
                    AOT_model_valid_in v (Rep_rel Π' x) =
                    AOT_model_valid_in v (Rep_rel Π x)› for v x
    by blast
  have ‹AOT_model_valid_in v (Rep_rel Π' (ωκ x)) =
        AOT_model_valid_in v (Rep_rel Π (ωκ x))› for x v
    by (simp add: AOT_model_ω_concrete_in_some_world Π'_prop)
  hence 0: ‹AOT_urrel_ωequiv (rel_to_urrel Π') (rel_to_urrel Π)›
    unfolding AOT_urrel_ωequiv_def
    by (smt (verit) AOT_rel_equiv_def Abs_rel_inverse Quotient3_def
                    κυ.simps(1) iso_tuple_UNIV_I urrel_quotient3 urrel_to_rel_def
                    Π_den Π'_den)
  have ‹rel_to_urrel Π' ∈ a›
   and ‹urrel_to_ωrel (rel_to_urrel Π') = urrel_to_ωrel (rel_to_urrel Π)›
    apply (metis AOT_model_enc_κ_def κ.simps(11) κ_enc_Π' a_def)
    by (metis Quotient3_rel 0 urrel_ωrel_quot)
  hence ‹∃s. s ∈ b ∧ urrel_to_ωrel s = urrel_to_ωrel (rel_to_urrel Π)›
    using ασ_eq_ord_exts_ex ασ_eq ext ασ_ασ' by blast
  then obtain s where
    s_prop: ‹s ∈ b ∧ urrel_to_ωrel s = urrel_to_ωrel (rel_to_urrel Π)›
    by blast
  then obtain Π" where
    Π"_prop: ‹rel_to_urrel Π" = s› and Π"_den: ‹AOT_model_denotes Π"›
    by (metis AOT_rel_equiv_def Quotient3_def urrel_quotient3)
  moreover have ‹AOT_model_enc κ' Π"›
    by (metis AOT_model_enc_κ_def Π"_den Π"_prop κ.simps(11) b_def s_prop)
  moreover have ‹AOT_model_valid_in v (Rep_rel Π" x) =
                 AOT_model_valid_in v (Rep_rel Π x)›
             if ‹∃w. AOT_model_concrete w x› for v x
  proof(insert that)
    assume ‹∃w. AOT_model_concrete w x›
    then obtain u where x_def: ‹x = ωκ u›
      by (metis AOT_model_concrete_κ.simps(2,3) κ.exhaust)
    show ‹AOT_model_valid_in v (Rep_rel Π" x) =
          AOT_model_valid_in v (Rep_rel Π x)›
      unfolding x_def
      by (smt (verit, best) AOT_rel_equiv_def Abs_rel_inverse Quotient3_def
            Π"_den Π"_prop Π_den κυ.simps(1) iso_tuple_UNIV_I s_prop
            urrel_quotient3 urrel_to_ωrel_def urrel_to_rel_def)
  qed
  ultimately show ‹∃Π'. AOT_model_denotes Π' ∧ AOT_model_enc κ' Π' ∧
     (∀v x. (∃w. AOT_model_concrete w x) ⟶ AOT_model_valid_in v (Rep_rel Π' x) =
                                            AOT_model_valid_in v (Rep_rel Π x))›
    apply (safe intro!: exI[where x=Π"])
    by auto
qed
end

text‹Products of unary individual terms and individual terms are individual terms.
     A tuple is regular, if at most one element does not denote. I.e. a pair is
     regular, if the first (unary) element denotes and the second is regular (i.e.
     at most one of its recursive tuple elements does not denote), or the first does
     not denote, but the second denotes (i.e. all its recursive tuple elements
     denote).›
instantiation prod :: (AOT_UnaryIndividualTerm, AOT_IndividualTerm) AOT_IndividualTerm
begin
definition AOT_model_regular_prod :: ‹'a×'b ⇒ bool› where
  ‹AOT_model_regular_prod ≡ λ (x,y) . AOT_model_denotes x ∧ AOT_model_regular y ∨
                                     ¬AOT_model_denotes x ∧ AOT_model_denotes y›
definition AOT_model_term_equiv_prod :: ‹'a×'b ⇒ 'a×'b ⇒ bool› where
  ‹AOT_model_term_equiv_prod ≡  λ (x₁,y₁) (x₂,y₂) .
    AOT_model_term_equiv x₁ x₂ ∧ AOT_model_term_equiv y₁ y₂›
function AOT_model_irregular_prod :: ‹('a×'b ⇒ o) ⇒ 'a×'b ⇒ o› where
  AOT_model_irregular_proj2: ‹AOT_model_denotes x ⟹
```

```
    AOT_model_irregular φ (x,y) =
    AOT_model_irregular (λy. φ (SOME x' . AOT_model_term_equiv x x', y)) y›
  | AOT_model_irregular_proj1: ‹¬AOT_model_denotes x ∧ AOT_model_denotes y ⟹
    AOT_model_irregular φ (x,y) =
    AOT_model_irregular (λx. φ (x, SOME y' . AOT_model_term_equiv y y')) x›
  | AOT_model_irregular_prod_generic: ‹¬AOT_model_denotes x ∧ ¬AOT_model_denotes y ⟹
    AOT_model_irregular φ (x,y) =
    (SOME Φ . AOT_model_irregular_spec Φ AOT_model_regular AOT_model_term_equiv)
      φ (x,y)›
  by auto blast
termination using "termination" by blast

instance proof
  obtain x :: 'a and y :: 'b where
    ‹¬AOT_model_denotes x› and ‹¬AOT_model_denotes y›
    by (meson AOT_model_nondenoting_ex AOT_model_denoting_ex)
  thus ‹∃x::'a×'b. ¬AOT_model_denotes x›
    by (auto simp: AOT_model_denotes_prod_def AOT_model_regular_prod_def)
next
  show ‹equivp (AOT_model_term_equiv :: 'a×'b ⇒ 'a×'b ⇒ bool)›
    by (rule equivpI; rule reflpI sympI transpI;
        simp add: AOT_model_term_equiv_prod_def AOT_model_term_equiv_part_equivp
                  equivp_reflp prod.case_eq_if case_prod_unfold equivp_symp)
       (metis equivp_transp[OF AOT_model_term_equiv_part_equivp])
next
  show ‹¬AOT_model_regular x ⟹ ¬ AOT_model_denotes x› for x :: ‹'a×'b›
    by (metis (mono_tags, lifting) AOT_model_denotes_prod_def case_prod_unfold
          AOT_model_irregular_nondenoting AOT_model_regular_prod_def)
next
  fix x y :: ‹'a×'b›
  show ‹AOT_model_term_equiv x y ⟹ AOT_model_denotes x = AOT_model_denotes y›
    by (metis (mono_tags, lifting) AOT_model_denotes_prod_def case_prod_beta
        AOT_model_term_equiv_denotes AOT_model_term_equiv_prod_def )
next
  fix x y :: ‹'a×'b›
  show ‹AOT_model_term_equiv x y ⟹ AOT_model_regular x = AOT_model_regular y›
    by (induct x; induct y;
        simp add: AOT_model_term_equiv_prod_def AOT_model_regular_prod_def)
       (meson AOT_model_term_equiv_denotes AOT_model_term_equiv_regular)
next
  interpret sp: AOT_model_irregular_spec ‹λφ (x::'a×'b) . ε₀ w . False›
                  AOT_model_regular AOT_model_term_equiv
    by (simp add: AOT_model_irregular_spec_def AOT_model_proposition_choice_simp)
  have ex_spec: ‹∃ φ :: ('a×'b ⇒ o) ⇒ 'a×'b ⇒ o .
    AOT_model_irregular_spec φ AOT_model_regular AOT_model_term_equiv›
    using sp.AOT_model_irregular_spec_axioms by blast
  have some_spec: ‹AOT_model_irregular_spec
    (SOME φ :: ('a×'b ⇒ o) ⇒ 'a×'b ⇒ o .
        AOT_model_irregular_spec φ AOT_model_regular AOT_model_term_equiv)
    AOT_model_regular AOT_model_term_equiv›
    using someI_ex[OF ex_spec] by argo
  interpret sp_some: AOT_model_irregular_spec
    ‹SOME φ :: ('a×'b ⇒ o) ⇒ 'a×'b ⇒ o .
        AOT_model_irregular_spec φ AOT_model_regular AOT_model_term_equiv›
    AOT_model_regular AOT_model_term_equiv
    using some_spec by blast
  show ‹AOT_model_irregular_spec (AOT_model_irregular :: ('a×'b ⇒ o) ⇒ 'a×'b ⇒ o)
          AOT_model_regular AOT_model_term_equiv›
  proof
    have ‹¬AOT_model_valid_in w (AOT_model_irregular φ (a, b))›
      for w φ and a :: 'a and b :: 'b
      by (induct arbitrary: φ rule: AOT_model_irregular_prod.induct)
         (auto simp: AOT_model_irregular_false sp_some.AOT_model_irregular_false)
```

```
    thus "¬AOT_model_valid_in w (AOT_model_irregular φ x)" for w φ and x :: ‹'a×'b›
      by (induct x)
  next
    {
      fix x₁ y₁ :: 'a and x₂ y₂ :: 'b and φ :: ‹'a×'b⇒o›
      assume x₁y₁_equiv: ‹AOT_model_term_equiv x₁ y₁›
      moreover assume x₂y₂_equiv: ‹AOT_model_term_equiv x₂ y₂›
      ultimately have xy_equiv: ‹AOT_model_term_equiv (x₁,x₂) (y₁,y₂)›
        by (simp add: AOT_model_term_equiv_prod_def)
      {
        assume ‹AOT_model_denotes x₁›
        moreover hence ‹AOT_model_denotes y₁›
          using AOT_model_term_equiv_denotes AOT_model_term_equiv_regular
                x₁y₁_equiv x₂y₂_equiv by blast
        ultimately have ‹AOT_model_irregular φ (x₁,x₂) =
                         AOT_model_irregular φ (y₁,y₂)›
          using AOT_model_irregular_equiv AOT_model_term_equiv_eps(3)
                x₁y₁_equiv x₂y₂_equiv by fastforce
      }
      moreover {
        assume ‹ AOT_model_denotes x₁ ∧ AOT_model_denotes x₂›
        moreover hence ‹ AOT_model_denotes y₁ ∧ AOT_model_denotes y₂›
          by (meson AOT_model_term_equiv_denotes x₁y₁_equiv x₂y₂_equiv)
        ultimately have ‹AOT_model_irregular φ (x₁,x₂) =
                         AOT_model_irregular φ (y₁,y₂)›
          using AOT_model_irregular_equiv AOT_model_term_equiv_eps(3)
                x₁y₁_equiv x₂y₂_equiv by fastforce
      }
      moreover {
        assume denotes_x: ‹(¬AOT_model_denotes x₁ ∧ ¬AOT_model_denotes x₂)›
        hence denotes_y: ‹(¬AOT_model_denotes y₁ ∧ ¬AOT_model_denotes y₂)›
          by (meson AOT_model_term_equiv_denotes AOT_model_term_equiv_regular
                    x₁y₁_equiv x₂y₂_equiv)
        have eps_eq: ‹Eps (AOT_model_term_equiv x₁) = Eps (AOT_model_term_equiv y₁)›
          by (simp add: AOT_model_term_equiv_eps(3) x₁y₁_equiv)
        have ‹AOT_model_irregular φ (x₁,x₂) = AOT_model_irregular φ (y₁,y₂)›
          using denotes_x denotes_y
          using sp_some.AOT_model_irregular_equiv xy_equiv by auto
      }
      moreover {
        assume denotes_x: ‹¬AOT_model_denotes x₁ ∧ AOT_model_denotes x₂›
        hence denotes_y: ‹¬AOT_model_denotes y₁ ∧ AOT_model_denotes y₂›
          by (meson AOT_model_term_equiv_denotes x₁y₁_equiv x₂y₂_equiv)
        have eps_eq: ‹Eps (AOT_model_term_equiv x₂) = Eps (AOT_model_term_equiv y₂)›
          by (simp add: AOT_model_term_equiv_eps(3) x₂y₂_equiv)
        have ‹AOT_model_irregular φ (x₁,x₂) = AOT_model_irregular φ (y₁,y₂)›
          using denotes_x denotes_y
          using AOT_model_irregular_nondenoting calculation(2) by blast
      }
      ultimately have ‹AOT_model_irregular φ (x₁,x₂) = AOT_model_irregular φ (y₁,y₂)›
        using AOT_model_term_equiv_denotes AOT_model_term_equiv_regular
              sp_some.AOT_model_irregular_equiv x₁y₁_equiv x₂y₂_equiv xy_equiv
        by blast
    } note 0 = this
    show ‹AOT_model_term_equiv x y ⟹
          AOT_model_irregular φ x = AOT_model_irregular φ y›
      for x y :: ‹'a×'b› and φ
      by (induct x; induct y; simp add: AOT_model_term_equiv_prod_def 0)
  next
    fix φ ψ :: ‹'a×'b ⇒ o›
    assume ‹AOT_model_regular x ⟹ φ x = ψ x› for x
    hence ‹φ (x, y) = ψ (x, y)›
      if ‹AOT_model_denotes x ∧ AOT_model_regular y ∨
```

```
          ¬AOT_model_denotes x ∧ AOT_model_denotes y› for x y
      using that unfolding AOT_model_regular_prod_def by simp
    hence ‹AOT_model_irregular φ (x,y) = AOT_model_irregular ψ (x,y)›
      for x :: 'a and y :: 'b
    proof (induct arbitrary: ψ φ rule: AOT_model_irregular_prod.induct)
      case (1 x y φ)
      thus ?case
        apply simp
        by (meson AOT_model_irregular_eqI AOT_model_irregular_nondenoting
                  AOT_model_term_equiv_denotes AOT_model_term_equiv_eps(1))
    next
      case (2 x y φ)
      thus ?case
        apply simp
        by (meson AOT_model_irregular_nondenoting AOT_model_term_equiv_denotes
                  AOT_model_term_equiv_eps(1))
    next
      case (3 x y φ)
      thus ?case
        apply simp
        by (metis (mono_tags, lifting) AOT_model_regular_prod_def case_prod_conv
                                       sp_some.AOT_model_irregular_eqI surj_pair)
    qed
    thus ‹AOT_model_irregular φ x = AOT_model_irregular ψ x› for  x :: ‹'a×'b›
      by (metis surjective_pairing)
  qed
qed
end

text‹Introduction rules for term equivalence on tuple terms.›
lemma AOT_meta_prod_equivI:
  shows "⋀ (a::'a::AOT_UnaryIndividualTerm) x (y :: 'b::AOT_IndividualTerm) .
            AOT_model_term_equiv x y ⟹ AOT_model_term_equiv (a,x) (a,y)"
    and "⋀ (x::'a::AOT_UnaryIndividualTerm) y (b :: 'b::AOT_IndividualTerm) .
            AOT_model_term_equiv x y ⟹ AOT_model_term_equiv (x,b) (y,b)"
    unfolding AOT_model_term_equiv_prod_def
    by (simp add: AOT_model_term_equiv_part_equivp equivp_reflp)+

text‹The type of propositions are trivial instances of terms.›

instantiation o :: AOT_Term
begin
definition AOT_model_denotes_o :: ‹o ⇒ bool› where
  ‹AOT_model_denotes_o ≡ λ_. True›
instance proof
  show ‹∃x::o. AOT_model_denotes x›
    by (simp add: AOT_model_denotes_o_def)
qed
end

text‹AOT's variables are modelled by restricting the type of terms to those terms
     that denote.›
typedef 'a AOT_var = ‹{ x :: 'a::AOT_Term . AOT_model_denotes x }›
  morphisms AOT_term_of_var AOT_var_of_term
  by (simp add: AOT_model_denoting_ex)

text‹Simplify automatically generated theorems and rules.›
declare AOT_var_of_term_induct[induct del]
        AOT_var_of_term_cases[cases del]
        AOT_term_of_var_induct[induct del]
        AOT_term_of_var_cases[cases del]
lemmas AOT_var_of_term_inverse = AOT_var_of_term_inverse[simplified]
   and AOT_var_of_term_inject = AOT_var_of_term_inject[simplified]
```

```
  and AOT_var_of_term_induct =
        AOT_var_of_term_induct[simplified, induct type: AOT_var]
  and AOT_var_of_term_cases =
        AOT_var_of_term_cases[simplified, cases type: AOT_var]
  and AOT_term_of_var = AOT_term_of_var[simplified]
  and AOT_term_of_var_cases =
        AOT_term_of_var_cases[simplified, induct pred: AOT_term_of_var]
  and AOT_term_of_var_induct =
        AOT_term_of_var_induct[simplified, induct pred: AOT_term_of_var]
  and AOT_term_of_var_inverse = AOT_term_of_var_inverse[simplified]
  and AOT_term_of_var_inject = AOT_term_of_var_inject[simplified]

text‹Equivalence by definition is modelled as necessary equivalence.›
consts AOT_model_equiv_def :: ‹o ⇒ o ⇒ bool›
specification(AOT_model_equiv_def)
  AOT_model_equiv_def: ‹AOT_model_equiv_def φ ψ = (∀ v . AOT_model_valid_in v φ =
                                                   AOT_model_valid_in v ψ)›
  by (rule exI[where x=‹λ φ ψ . ∀ v . AOT_model_valid_in v φ =
                                   AOT_model_valid_in v ψ›]) simp

text‹Identity by definition is modelled as identity for denoting terms plus
     co-denoting.›
consts AOT_model_id_def :: ‹('b ⇒ 'a::AOT_Term) ⇒ ('b ⇒ 'a) ⇒ bool›
specification(AOT_model_id_def)
  AOT_model_id_def: ‹(AOT_model_id_def τ σ) = (∀ α . if AOT_model_denotes (σ α)
                                                 then τ α = σ α
                                                 else ¬AOT_model_denotes (τ α))›
  by (rule exI[where x="λ τ σ . ∀ α . if AOT_model_denotes (σ α)
                                      then τ α = σ α
                                      else ¬AOT_model_denotes (τ α)"])
     blast
text‹To reduce definitions by identity without free variables to definitions
     by identity with free variables acting on the unit type, we give the unit type
     a trivial instantiation to @{class AOT_Term}.›
instantiation unit :: AOT_Term
begin
definition AOT_model_denotes_unit :: ‹unit ⇒ bool› where
  ‹AOT_model_denotes_unit ≡ λ_. True›
instance proof qed(simp add: AOT_model_denotes_unit_def)
end

text‹Modally-strict and modally-fragile axioms are as necessary,
     resp. actually valid propositions.›
definition AOT_model_axiom where
  ‹AOT_model_axiom ≡ λ φ . ∀ v . AOT_model_valid_in v φ›
definition AOT_model_act_axiom where
  ‹AOT_model_act_axiom ≡ λ φ . AOT_model_valid_in w₀ φ›

lemma AOT_model_axiomI:
  assumes ‹⋀v . AOT_model_valid_in v φ›
  shows ‹AOT_model_axiom φ›
  unfolding AOT_model_axiom_def using assms ..

lemma AOT_model_act_axiomI:
  assumes ‹AOT_model_valid_in w₀ φ›
  shows ‹AOT_model_act_axiom φ›
  unfolding AOT_model_act_axiom_def using assms .

(*<*)
end
(*>*)
```

A.2. Outer Syntax Commands

```
(*<*)
theory AOT_commands
  imports AOT_model "HOL-Eisbach.Eisbach_Tools"
  keywords "AOT_define" :: thy_decl
       and "AOT_theorem" :: thy_goal
       and "AOT_lemma" :: thy_goal
       and "AOT_act_theorem" :: thy_goal
       and "AOT_act_lemma" :: thy_goal

       and "AOT_axiom" :: thy_goal
       and "AOT_act_axiom" :: thy_goal

       and "AOT_assume" :: prf_asm % "proof"
       and "AOT_have" :: prf_goal % "proof"
       and "AOT_hence" :: prf_goal % "proof"
       and "AOT_modally_strict {" :: prf_open % "proof"
       and "AOT_actually {" :: prf_open % "proof"
       and "AOT_obtain" :: prf_asm_goal % "proof"
       and "AOT_show" :: prf_asm_goal % "proof"
       and "AOT_thus" :: prf_asm_goal % "proof"

       and "AOT_find_theorems" :: diag
       and "AOT_sledgehammer" :: diag
       and "AOT_sledgehammer_only" :: diag
       and "AOT_syntax_print_translations" :: thy_decl
       and "AOT_no_syntax_print_translations" :: thy_decl
begin
(*>*)

section‹Outer Syntax Commands›

nonterminal AOT_prop
nonterminal φ
nonterminal φ'
nonterminal τ
nonterminal τ'
nonterminal "AOT_axiom"
nonterminal "AOT_act_axiom"
ML_file AOT_keys.ML
ML_file AOT_commands.ML
setup‹AOT_Theorems.setup›
setup‹AOT_Definitions.setup›
setup‹AOT_no_atp.setup›

(*<*)
end
(*>*)
```

A.3. Approximation of the Syntax of PLM

```
(*<*)
theory AOT_syntax
  imports AOT_commands
  keywords "AOT_register_variable_names" :: thy_decl
       and "AOT_register_metavariable_names" :: thy_decl
       and "AOT_register_premise_set_names" :: thy_decl
       and "AOT_register_type_constraints" :: thy_decl
     abbrevs "actually" = "𝒜"
         and "neccessarily" = "□"
         and "possibly" = "◇"
         and "the" = "ι"
         and "lambda" = "[λ∙]"
         and "being such that" = "[λ ∙]"
         and "forall" = "∀"
         and "exists" = "∃"
         and "equivalent" = "≡"
         and "not" = "¬"
         and "implies" = "→"
         and "equal" = "="
         and "by definition" = "df"
         and "df" = "df"
         and "denotes" = "↓"
begin
(*>*)

section‹Approximation of the Syntax of PLM›

locale AOT_meta_syntax
begin
notation AOT_model_valid_in ("[_ ⊨ _]")
notation AOT_model_axiom ("□[_]")
notation AOT_model_act_axiom ("𝒜[_]")
end
locale AOT_no_meta_syntax
begin
no_notation AOT_model_valid_in ("[_ ⊨ _]")
no_notation AOT_model_axiom ("□[_]")
no_notation AOT_model_act_axiom ("𝒜[_]")
end

consts AOT_denotes :: ‹'a::AOT_Term ⇒ o›
       AOT_imp :: ‹[o, o] ⇒ o›
       AOT_not :: ‹o ⇒ o›
       AOT_box :: ‹o ⇒ o›
       AOT_act :: ‹o ⇒ o›
       AOT_forall :: ‹('a::AOT_Term ⇒ o) ⇒ o›
       AOT_eq :: ‹'a::AOT_Term ⇒ 'a::AOT_Term ⇒ o›
       AOT_desc :: ‹('a::AOT_UnaryIndividualTerm ⇒ o) ⇒ 'a›
       AOT_exe :: ‹<'a::AOT_IndividualTerm> ⇒ 'a ⇒ o›
       AOT_lambda :: ‹('a::AOT_IndividualTerm ⇒ o) ⇒ <'a>›
       AOT_lambda0 :: ‹o ⇒ o›
       AOT_concrete :: ‹<'a::AOT_UnaryIndividualTerm> AOT_var›

nonterminal κs and Π and Π0 and α and exe_arg and exe_args
        and lambda_args and desc and free_var and free_vars
        and AOT_props and AOT_premises and AOT_world_relative_prop

syntax "_AOT_process_frees" :: ‹φ ⇒ φ'› ("_")
       "_AOT_verbatim" :: ‹any ⇒ φ› (‹«_»›)
       "_AOT_verbatim" :: ‹any ⇒ τ› (‹«_»›)
       "_AOT_quoted" :: ‹φ' ⇒ any› (‹«_»›)
```

```
        "_AOT_quoted" :: ‹τ' ⇒ any› (‹«_»›)
        "" :: ‹φ ⇒ φ› (‹'(_')›)
        "_AOT_process_frees" :: ‹τ ⇒ τ'› ("_")
        "" :: ‹κs ⇒ τ› ("_")
        "" :: ‹Π ⇒ τ› ("_")
        "" :: ‹φ ⇒ τ› ("'(_')")
        "_AOT_term_var" :: ‹id_position ⇒ τ› ("_")
        "_AOT_term_var" :: ‹id_position ⇒ φ› ("_")
        "_AOT_exe_vars" :: ‹id_position ⇒ exe_arg› ("_")
        "_AOT_lambda_vars" :: ‹id_position ⇒ lambda_args› ("_")
        "_AOT_var" :: ‹id_position ⇒ α› ("_")
        "_AOT_vars" :: ‹id_position ⇒ any›
        "_AOT_verbatim" :: ‹any ⇒ α› (‹«_»›)
        "_AOT_valid" :: ‹w ⇒ φ' ⇒ bool› (‹[_ ⊨ _]›)
        "_AOT_denotes" :: ‹τ ⇒ φ› (‹_↓›)
        "_AOT_imp" :: ‹[φ, φ] ⇒ φ› (infixl ‹→› 25)
        "_AOT_not" :: ‹φ ⇒ φ› (‹ _› [50] 50)
        "_AOT_not" :: ‹φ ⇒ φ› (‹¬_› [50] 50)
        "_AOT_box" :: ‹φ ⇒ φ› (‹□_› [49] 54)
        "_AOT_act" :: ‹φ ⇒ φ› (‹𝓐_› [49] 54)
        "_AOT_all" :: ‹α ⇒ φ ⇒ φ› (‹∀_ _› [1,40])
syntax (input)
        "_AOT_all_ellipse"
            :: ‹id_position ⇒ id_position ⇒ φ ⇒ φ› (‹∀_...∀_ _› [1,40])
syntax (output)
        "_AOT_all_ellipse"
            :: ‹id_position ⇒ id_position ⇒ φ ⇒ φ› (‹∀_...∀_'(_')› [1,40])
syntax
        "_AOT_eq" :: ‹[τ, τ] ⇒ φ› (infixl ‹=› 50)
        "_AOT_desc" :: ‹α ⇒ φ ⇒ desc› ("ι__" [1,1000])
        "" :: ‹desc ⇒ κs› ("_")
        "_AOT_lambda" :: ‹lambda_args ⇒ φ ⇒ Π› (‹[λ_ _]›)
        "_explicitRelation" :: ‹τ ⇒ Π› ("[_]")
        "" :: ‹κs ⇒ exe_arg› ("_")
        "" :: ‹exe_arg ⇒ exe_args› ("_")
        "_AOT_exe_args" :: ‹exe_arg ⇒ exe_args ⇒ exe_args› ("__")
        "_AOT_exe_arg_ellipse" :: ‹id_position ⇒ id_position ⇒ exe_arg› ("_..._")
        "_AOT_lambda_arg_ellipse"
            :: ‹id_position ⇒ id_position ⇒ lambda_args› ("_..._")
        "_AOT_term_ellipse" :: ‹id_position ⇒ id_position ⇒ τ› ("_..._")
        "_AOT_exe" :: ‹Π ⇒ exe_args ⇒ φ› (‹__›)
        "_AOT_enc" :: ‹exe_args ⇒ Π ⇒ φ› (‹__›)
        "_AOT_lambda0" :: ‹φ ⇒ Π0› (‹[λ _]›)
        "" :: ‹Π0 ⇒ φ› ("_")
        "" :: ‹Π0 ⇒ τ› ("_")
        "_AOT_concrete" :: ‹Π› (‹E!›)
        "" :: ‹any ⇒ exe_arg› ("«_»")
        "" :: ‹desc ⇒ free_var› ("_")
        "" :: ‹Π ⇒ free_var› ("_")
        "_AOT_appl" :: ‹id_position ⇒ free_vars ⇒ φ› ("_'{_'}")
        "_AOT_appl" :: ‹id_position ⇒ free_vars ⇒ τ› ("_'{_'}")
        "_AOT_appl" :: ‹id_position ⇒ free_vars ⇒ free_vars› ("_'{_'}")
        "_AOT_appl" :: ‹id_position ⇒ free_vars ⇒ free_vars› ("_'{_'}")
        "_AOT_term_var" :: ‹id_position ⇒ free_var› ("_")
        "" :: ‹any ⇒ free_var› ("«_»")
        "" :: ‹free_var ⇒ free_vars› ("_")
        "_AOT_args" :: ‹free_var ⇒ free_vars ⇒ free_vars› ("_,_")
        "_AOT_free_var_ellipse" :: ‹id_position ⇒ id_position ⇒ free_var› ("_..._")
syntax "_AOT_premises"
            :: ‹AOT_world_relative_prop ⇒ AOT_premises ⇒ AOT_premises› (infixr ‹,› 3)
        "_AOT_world_relative_prop" :: "φ ⇒ AOT_world_relative_prop" ("_")
        "" :: "AOT_world_relative_prop ⇒ AOT_premises" ("_")
        "_AOT_prop" :: ‹AOT_world_relative_prop ⇒ AOT_prop› (‹_›)
```

```
      "" :: ‹AOT_prop ⇒ AOT_props› (‹_›)
      "_AOT_derivable" :: "AOT_premises ⇒ φ' ⇒ AOT_prop" (infixl ‹⊢› 2)
      "_AOT_nec_derivable" :: "AOT_premises ⇒ φ' ⇒ AOT_prop" (infixl ‹⊢□› 2)
      "_AOT_theorem" :: "φ' ⇒ AOT_prop" (‹⊢ _›)
      "_AOT_nec_theorem" :: "φ' ⇒ AOT_prop" (‹⊢□ _›)
      "_AOT_equiv_def" :: ‹φ ⇒ φ ⇒ AOT_prop› (infixl ‹≡df› 3)
      "_AOT_axiom" :: "φ' ⇒ AOT_axiom" (‹_›)
      "_AOT_act_axiom" :: "φ' ⇒ AOT_act_axiom" (‹_›)
      "_AOT_axiom" :: "φ' ⇒ AOT_prop" (‹_ ∈ Λ□›)
      "_AOT_act_axiom" :: "φ' ⇒ AOT_prop" (‹_ ∈ Λ›)
      "_AOT_id_def" :: ‹τ ⇒ τ ⇒ AOT_prop› (infixl ‹=df› 3)
      "_AOT_for_arbitrary"
          :: ‹id_position ⇒ AOT_prop ⇒ AOT_prop› (‹for arbitrary _: _› [1000,1] 1)
syntax (output) "_lambda_args" :: ‹any ⇒ patterns ⇒ patterns› ("__")

translations
  "[w ⊨ φ]" => "CONST AOT_model_valid_in w φ"

AOT_syntax_print_translations
  "[w ⊨ φ]" <= "CONST AOT_model_valid_in w φ"

ML_file AOT_syntax.ML

AOT_register_type_constraints
  Individual: ‹_::AOT_UnaryIndividualTerm› ‹_::AOT_IndividualTerm› and
  Proposition: o and
  Relation: ‹‹_::AOT_IndividualTerm›› and
  Term: ‹_::AOT_Term›

AOT_register_variable_names
  Individual: x y z ν μ a b c d and
  Proposition: p q r s and
  Relation: F G H P Q R S and
  Term: α β γ δ

AOT_register_metavariable_names
  Individual: κ and
  Proposition: φ ψ χ ϑ ζ ξ Θ and
  Relation: Π and
  Term: τ σ

AOT_register_premise_set_names Γ Δ Λ

parse_ast_translation‹[
  (syntax_const‹_AOT_var›, K AOT_check_var),
  (syntax_const‹_AOT_exe_vars›, K AOT_split_exe_vars),
  (syntax_const‹_AOT_lambda_vars›, K AOT_split_lambda_args)
]›

translations
  "_AOT_denotes τ" => "CONST AOT_denotes τ"
  "_AOT_imp φ ψ" => "CONST AOT_imp φ ψ"
  "_AOT_not φ" => "CONST AOT_not φ"
  "_AOT_box φ" => "CONST AOT_box φ"
  "_AOT_act φ" => "CONST AOT_act φ"
  "_AOT_eq τ τ'" => "CONST AOT_eq τ τ'"
  "_AOT_lambda0 φ" => "CONST AOT_lambda0 φ"
  "_AOT_concrete" => "CONST AOT_term_of_var (CONST AOT_concrete)"
  "_AOT_lambda α φ" => "CONST AOT_lambda (_abs α φ)"
  "_explicitRelation Π" => "Π"

AOT_syntax_print_translations
  "_AOT_lambda (_lambda_args x y) φ" <= "CONST AOT_lambda (_abs (_pattern x y) φ)"
```

```
  "_AOT_lambda (_lambda_args x y) φ" <= "CONST AOT_lambda (_abs (_patterns x y) φ)"
  "_AOT_lambda x φ" <= "CONST AOT_lambda (_abs x φ)"
  "_lambda_args x (_lambda_args y z)" <= "_lambda_args x (_patterns y z)"
  "_lambda_args (x y z)" <= "_lambda_args (_tuple x (_tuple_arg (_tuple y z)))"


AOT_syntax_print_translations
  "_AOT_imp φ ψ" <= "CONST AOT_imp φ ψ"
  "_AOT_not φ" <= "CONST AOT_not φ"
  "_AOT_box φ" <= "CONST AOT_box φ"
  "_AOT_act φ" <= "CONST AOT_act φ"
  "_AOT_all α φ" <= "CONST AOT_forall (_abs α φ)"
  "_AOT_all α φ" <= "CONST AOT_forall (λα. φ)"
  "_AOT_eq τ τ'" <= "CONST AOT_eq τ τ'"
  "_AOT_desc x φ" <= "CONST AOT_desc (_abs x φ)"
  "_AOT_desc x φ" <= "CONST AOT_desc (λx. φ)"
  "_AOT_lambda0 φ" <= "CONST AOT_lambda0 φ"
  "_AOT_concrete" <= "CONST AOT_term_of_var (CONST AOT_concrete)"

translations
  "_AOT_appl φ (_AOT_args a b)" => "_AOT_appl (φ a) b"
  "_AOT_appl φ a" => "φ a"


parse_translation‹
[
  (syntax_const‹_AOT_var›, parseVar true),
  (syntax_const‹_AOT_vars›, parseVar false),
  (syntax_const‹_AOT_valid›, fn ctxt => fn [w,x] =>
    const‹AOT_model_valid_in› $ w $ x),
  (syntax_const‹_AOT_quoted›, fn ctxt => fn [x] => x),
  (syntax_const‹_AOT_process_frees›, fn ctxt => fn [x] => processFrees ctxt x),
  (syntax_const‹_AOT_world_relative_prop›, fn ctxt => fn [x] => let
    val (x, premises) = processFreesAndPremises ctxt x
    val (world::formulas) = Variable.variant_frees ctxt [x]
        (("v", dummyT)::(map (fn _ => ("φ", dummyT)) premises))
    val term = HOLogic.mk_Trueprop
        (@{const AOT_model_valid_in} $ Free world $ processFrees ctxt x)
    val term = fold (fn (premise,form) => fn trm =>
         @{const "Pure.imp"} $
        HOLogic.mk_Trueprop
          (Const (const_name‹Set.member›, dummyT) $ Free form $ premise) $
          (Term.absfree (Term.dest_Free (dropConstraints premise)) trm $ Free form)
    ) (ListPair.zipEq (premises,formulas)) term
    val term = fold (fn (form) => fn trm =>
         Const (const_name‹Pure.all›, dummyT) $
        (Term.absfree form trm)
    ) formulas term
    val term = Term.absfree world term
    in term end),
  (syntax_const‹_AOT_prop›, fn ctxt => fn [x] => let
    val world = case (AOT_ProofData.get ctxt) of SOME w => w
        | _ => raise Fail "Expected world to be stored in the proof state."
    in x $ world end),
  (syntax_const‹_AOT_theorem›, fn ctxt => fn [x] =>
      HOLogic.mk_Trueprop (@{const AOT_model_valid_in} $ @{const w0} $ x)),
  (syntax_const‹_AOT_axiom›, fn ctxt => fn [x] =>
      HOLogic.mk_Trueprop (@{const AOT_model_axiom} $ x)),
  (syntax_const‹_AOT_act_axiom›, fn ctxt => fn [x] =>
      HOLogic.mk_Trueprop (@{const AOT_model_act_axiom} $ x)),
  (syntax_const‹_AOT_nec_theorem›, fn ctxt => fn [trm] => let
    val world = singleton (Variable.variant_frees ctxt [trm]) ("v", @{typ w})
    val trm = HOLogic.mk_Trueprop (@{const AOT_model_valid_in} $ Free world $ trm)
```

```
    val trm = Term.absfree world trm
    val trm = Const (const_name‹Pure.all›, dummyT) $ trm
    in trm end),
  (syntax_const‹_AOT_derivable›, fn ctxt => fn [x,y] => let
    val world = case (AOT_ProofData.get ctxt) of SOME w => w
      | _ => raise Fail "Expected world to be stored in the proof state."
    in foldPremises world x y end),
  (syntax_const‹_AOT_nec_derivable›, fn ctxt => fn [x,y] => let
    in Const (const_name‹Pure.all›, dummyT) $
       Abs ("v", dummyT, foldPremises (Bound 0) x y) end),
  (syntax_const‹_AOT_for_arbitrary›, fn ctxt => fn [_ $ var $ pos,trm] => let
    val trm = Const (const_name‹Pure.all›, dummyT) $
        (Const ("_constrainAbs", dummyT) $ Term.absfree (Term.dest_Free var) trm $ pos)
    in trm end),
  (syntax_const‹_AOT_equiv_def›, parseEquivDef),
  (syntax_const‹_AOT_exe›, parseExe),
  (syntax_const‹_AOT_enc›, parseEnc)
]
›

parse_ast_translation‹
[
  (syntax_const‹_AOT_exe_arg_ellipse›, parseEllipseList "_AOT_term_vars"),
  (syntax_const‹_AOT_lambda_arg_ellipse›, parseEllipseList "_AOT_vars"),
  (syntax_const‹_AOT_free_var_ellipse›, parseEllipseList "_AOT_term_vars"),
  (syntax_const‹_AOT_term_ellipse›, parseEllipseList "_AOT_term_vars"),
  (syntax_const‹_AOT_all_ellipse›, fn ctx => fn [a,b,c] =>
      Ast.mk_appl (Ast.Constant const_name‹AOT_forall›) [
        Ast.mk_appl (Ast.Constant "_abs") [parseEllipseList "_AOT_vars" ctx [a,b],c]
      ])
]
›

syntax (output)
  "_AOT_individual_term" :: ‹'a ⇒ tuple_args› ("_")
  "_AOT_individual_terms" :: ‹tuple_args ⇒ tuple_args ⇒ tuple_args› ("__")
  "_AOT_relation_term" :: ‹'a ⇒ Π›
  "_AOT_any_term" :: ‹'a ⇒ τ›


print_ast_translation‹AOT_syntax_print_ast_translations[
 (syntax_const‹_AOT_individual_term›, AOT_print_individual_term),
 (syntax_const‹_AOT_relation_term›, AOT_print_relation_term),
 (syntax_const‹_AOT_any_term›, AOT_print_generic_term)
]›

AOT_syntax_print_translations
  "_AOT_individual_terms (_AOT_individual_term x) (_AOT_individual_terms (_tuple y z))"
  <= "_AOT_individual_terms (_tuple x (_tuple_args y z))"
  "_AOT_individual_terms (_AOT_individual_term x) (_AOT_individual_term y)"
  <= "_AOT_individual_terms (_tuple x (_tuple_arg y))"
  "_AOT_individual_terms (_tuple x y)" <= "_AOT_individual_term (_tuple x y)"
  "_AOT_exe (_AOT_relation_term Π) (_AOT_individual_term κ)" <= "CONST AOT_exe Π κ"
  "_AOT_denotes (_AOT_any_term κ)" <= "CONST AOT_denotes κ"

AOT_define AOT_conj :: ‹[φ, φ] ⇒ φ› (infixl ‹&› 35) ‹φ & ψ ≡df ¬(φ → ¬ψ)›
declare "AOT_conj"[AOT del, AOT_defs del]
AOT_define AOT_disj :: ‹[φ, φ] ⇒ φ› (infixl ‹∨› 35) ‹φ ∨ ψ ≡df ¬φ → ψ›
declare "AOT_disj"[AOT del, AOT_defs del]
AOT_define AOT_equiv :: ‹[φ, φ] ⇒ φ› (infix ‹≡› 20) ‹φ ≡ ψ ≡df (φ → ψ) & (ψ → φ)›
declare "AOT_equiv"[AOT del, AOT_defs del]
AOT_define AOT_dia :: ‹φ ⇒ φ› (‹◇_› [49] 54) ‹◇φ ≡df ¬□¬φ›
declare "AOT_dia"[AOT del, AOT_defs del]
```

```

context AOT_meta_syntax
begin
notation AOT_dia ("◇_" [49] 54)
notation AOT_conj (infixl ‹&› 35)
notation AOT_disj (infixl ‹∨› 35)
notation AOT_equiv (infixl ‹≡› 20)
end
context AOT_no_meta_syntax
begin
no_notation AOT_dia ("◇_" [49] 54)
no_notation AOT_conj (infixl ‹&› 35)
no_notation AOT_disj (infixl ‹∨› 35)
no_notation AOT_equiv (infixl ‹≡› 20)
end


print_translation ‹
AOT_syntax_print_translations
 [
  AOT_preserve_binder_abs_tr'
    const_syntax‹AOT_forall›
    syntax_const‹_AOT_all›
    (syntax_const‹_AOT_all_ellipse›, true)
    const_name‹AOT_imp›,
  AOT_binder_trans @{theory} @{binding "AOT_forall_binder"} syntax_const‹_AOT_all›,
  Syntax_Trans.preserve_binder_abs_tr'
    const_syntax‹AOT_desc›
    syntax_const‹_AOT_desc›,
  AOT_binder_trans @{theory} @{binding "AOT_desc_binder"} syntax_const‹_AOT_desc›,
  AOT_preserve_binder_abs_tr'
    const_syntax‹AOT_lambda›
    syntax_const‹_AOT_lambda›
    (syntax_const‹_AOT_lambda_arg_ellipse›, false)
    const_name‹undefined›,
  AOT_binder_trans
    @{theory}
    @{binding "AOT_lambda_binder"}
    syntax_const‹_AOT_lambda›
 ]
›

parse_translation‹
[(syntax_const‹_AOT_id_def›, parseIdDef)]
›

parse_ast_translation‹[
 (syntax_const‹_AOT_all›,
  AOT_restricted_binder const_name‹AOT_forall› const_name‹AOT_imp›),
 (syntax_const‹_AOT_desc›,
  AOT_restricted_binder const_name‹AOT_desc› const_name‹AOT_conj›)
]›

AOT_define AOT_exists :: ‹α ⇒ φ ⇒ φ› ‹«AOT_exists φ» ≡df ¬∀α ¬φ{α}›
declare AOT_exists[AOT del, AOT_defs del]
syntax "_AOT_exists" :: ‹α ⇒ φ ⇒ φ› ("∃_ _" [1,40])

AOT_syntax_print_translations
  "_AOT_exists α φ" <= "CONST AOT_exists (_abs α φ)"
  "_AOT_exists α φ" <= "CONST AOT_exists (λα. φ)"

parse_ast_translation‹
[(syntax_const‹_AOT_exists›,
```

```
  AOT_restricted_binder const_name‹AOT_exists› const_name‹AOT_conj›)]
›

context AOT_meta_syntax
begin
notation AOT_exists (binder "∃" 8)
end
context AOT_no_meta_syntax
begin
no_notation AOT_exists (binder "∃" 8)
end


syntax (input)
   "_AOT_exists_ellipse" :: ‹id_position ⇒ id_position ⇒ φ ⇒ φ› (‹∃_...∃_ _› [1,40])
syntax (output)
   "_AOT_exists_ellipse" :: ‹id_position ⇒ id_position ⇒ φ ⇒ φ› (‹∃_...∃_ '(_')› [1,40])
parse_ast_translation‹[(syntax_const‹_AOT_exists_ellipse›, fn ctx => fn [a,b,c] =>
  Ast.mk_appl (Ast.Constant "AOT_exists")
    [Ast.mk_appl (Ast.Constant "_abs") [parseEllipseList "_AOT_vars" ctx [a,b],c]])]›
print_translation‹AOT_syntax_print_translations [
  AOT_preserve_binder_abs_tr'
    const_syntax‹AOT_exists›
    syntax_const‹_AOT_exists›
    (syntax_const‹_AOT_exists_ellipse›,true) const_name‹AOT_conj›,
  AOT_binder_trans
    @{theory}
    @{binding "AOT_exists_binder"}
    syntax_const‹_AOT_exists›
]›



syntax "_AOT_DDDOT" :: "φ" ("...")
syntax "_AOT_DDDOT" :: "φ" ("...")
parse_translation‹[(syntax_const‹_AOT_DDDOT›, parseDDOT)]›

print_translation‹AOT_syntax_print_translations
[(const_syntax‹Pure.all›, fn ctxt => fn [Abs (_, _,
  Const (const_syntax‹HOL.Trueprop›, _) $
  (Const (const_syntax‹AOT_model_valid_in›, _) $ Bound 0 $ y))] => let
    val y = (Const (syntax_const‹_AOT_process_frees›, dummyT) $ y)
    in (Const (syntax_const‹_AOT_nec_theorem›, dummyT) $ y) end
| [p as Abs (name, _,
  Const (const_syntax‹HOL.Trueprop›, _) $
  (Const (const_syntax‹AOT_model_valid_in›, _) $ w $ y))]
=> (Const (syntax_const‹_AOT_for_arbitrary›, dummyT) $
    (Const ("_bound", dummyT) $ Free (name, dummyT)) $
    (Term.betapply (p, (Const ("_bound", dummyT) $ Free (name, dummyT)))))
),

 (const_syntax‹AOT_model_valid_in›, fn ctxt =>
  fn [w as (Const ("_free", _) $ Free (v, _)), y] => let
    val is_world = (case (AOT_ProofData.get ctxt)
        of SOME (Free (w, _)) => Name.clean w = Name.clean v | _ => false)
    val y = (Const (syntax_const‹_AOT_process_frees›, dummyT) $ y)
    in if is_world then y else Const (syntax_const‹_AOT_valid›, dummyT) $ w $ y end
  | [Const (const_syntax‹w₀›, _), y] => let
    val y = (Const (syntax_const‹_AOT_process_frees›, dummyT) $ y)
    in case (AOT_ProofData.get ctxt) of SOME (Const (const_name‹w₀›, _)) => y |
            _ => Const (syntax_const‹_AOT_theorem›, dummyT) $ y end
  | [Const ("_var", _) $ _, y] => let
    val y = (Const (syntax_const‹_AOT_process_frees›, dummyT) $ y)
```

```
  in Const (syntax_const‹_AOT_nec_theorem›, dummyT) $ y end
 ),
 (const_syntax‹AOT_model_axiom›, fn ctxt => fn [trm] =>
  Const (syntax_const‹_AOT_axiom›, dummyT) $
  (Const (syntax_const‹_AOT_process_frees›, dummyT) $ trm)),
 (const_syntax‹AOT_model_act_axiom›, fn ctxt => fn [trm] =>
  Const (syntax_const‹_AOT_axiom›, dummyT) $
  (Const (syntax_const‹_AOT_process_frees›, dummyT) $ trm)),
(syntax_const‹_AOT_process_frees›, fn _ =>  fn [t] => let
 fun mapAppls (x as Const ("_free", _) $
          Free (_, Type ("_ignore_type", [Type ("fun", _)])))
    = (Const ("_AOT_raw_appl", dummyT) $ x)
  | mapAppls (x as Const ("_free", _) $ Free (_, Type ("fun", _)))
    = (Const ("_AOT_raw_appl", dummyT) $ x)
  | mapAppls (x as Const ("_var", _) $
          Var (_, Type ("_ignore_type", [Type ("fun", _)])))
    = (Const ("_AOT_raw_appl", dummyT) $ x)
  | mapAppls (x as Const ("_var", _) $ Var (_, Type ("fun", _)))
    = (Const ("_AOT_raw_appl", dummyT) $ x)
  | mapAppls (x $ y) = mapAppls x $ mapAppls y
  | mapAppls (Abs (x,y,z)) = Abs (x,y, mapAppls z)
  | mapAppls x = x
 in mapAppls t end
)
]
›

print_ast_translation‹AOT_syntax_print_ast_translations
let
fun handleTermOfVar x kind name = (
let
val _ = case kind of "_free" => () | "_var" => () | "_bound" => () | _ => raise Match
in
 case printVarKind name
  of (SingleVariable name) => Ast.Appl [Ast.Constant kind, Ast.Variable name]
  | (Ellipses (s, e)) =>  Ast.Appl [Ast.Constant "_AOT_free_var_ellipse",
  Ast.Appl [Ast.Constant kind, Ast.Variable s],
  Ast.Appl [Ast.Constant kind, Ast.Variable e]
   ]
 | Verbatim name => Ast.mk_appl (Ast.Constant "_AOT_quoted")
            [Ast.mk_appl (Ast.Constant "_AOT_term_of_var") [x]]
end
)
fun termOfVar ctxt (Ast.Appl [Ast.Constant "_constrain",
   x as Ast.Appl [Ast.Constant kind, Ast.Variable name], _]) = termOfVar ctxt x
 | termOfVar ctxt (x as Ast.Appl [Ast.Constant kind, Ast.Variable name])
   = handleTermOfVar x kind name
 | termOfVar ctxt (x as Ast.Appl [Ast.Constant rep, y]) = (
let
val (restr,_) = Local_Theory.raw_theory_result (fn thy => (
let
val restrs = Symtab.dest (AOT_Restriction.get thy)
val restr = List.find (fn (n,(_,Const (c,t))) => (
 c = rep orelse c = Lexicon.unmark_const rep) | _ => false) restrs
in
(restr,thy)
end
)) ctxt
in
 case restr of SOME r => Ast.Appl [Ast.Constant (const_syntax‹AOT_term_of_var›), y]
 | _ => raise Match
end)
```

```
in
[(const_syntax‹AOT_term_of_var›, fn ctxt => fn [x] => termOfVar ctxt x),
("_AOT_raw_appl", fn ctxt => fn t::a::args => let
fun applyTermOfVar (t as Ast.Appl (Ast.Constant const_syntax‹AOT_term_of_var›::[x]))
    = (case try (termOfVar ctxt) x of SOME y => y | _ => t)
  | applyTermOfVar y = (case try (termOfVar ctxt) y of SOME x => x | _ => y)
val ts = fold (fn a => fn b => Ast.mk_appl (Ast.Constant syntax_const‹_AOT_args›)
               [b,applyTermOfVar a]) args (applyTermOfVar a)
in Ast.mk_appl (Ast.Constant syntax_const‹_AOT_appl›) [t,ts] end)]
end
›

context AOT_meta_syntax
begin
notation AOT_denotes ("_↓")
notation AOT_imp (infixl "→" 25)
notation AOT_not ("¬_" [50] 50)
notation AOT_box ("□_" [49] 54)
notation AOT_act ("𝒜_" [49] 54)
notation AOT_forall (binder "∀" 8)
notation AOT_eq (infixl "=" 50)
notation AOT_desc (binder "ι" 100)
notation AOT_lambda (binder "λ" 100)
notation AOT_lambda0 ("[λ _]")
notation AOT_exe ("⦇_,_⦈")
notation AOT_model_equiv_def (infixl "≡df" 10)
notation AOT_model_id_def (infixl "=df" 10)
notation AOT_term_of_var ("⟨_⟩")
notation AOT_concrete ("E!")
end
context AOT_no_meta_syntax
begin
no_notation AOT_denotes ("_↓")
no_notation AOT_imp (infixl "→" 25)
no_notation AOT_not ("¬_" [50] 50)
no_notation AOT_box ("□_" [49] 54)
no_notation AOT_act ("𝒜_" [49] 54)
no_notation AOT_forall (binder "∀" 8)
no_notation AOT_eq (infixl "=" 50)
no_notation AOT_desc (binder "ι" 100)
no_notation AOT_lambda (binder "λ" 100)
no_notation AOT_lambda0 ("[λ _]")
no_notation AOT_exe ("⦇_,_⦈")
no_notation AOT_model_equiv_def (infixl "≡df" 10)
no_notation AOT_model_id_def (infixl "=df" 10)
no_notation AOT_term_of_var ("⟨_⟩")
no_notation AOT_concrete ("E!")
end

bundle AOT_syntax
begin
declare[[show_AOT_syntax=true, show_question_marks=false, eta_contract=false]]
end

bundle AOT_no_syntax
begin
declare[[show_AOT_syntax=false, show_question_marks=true]]
end

parse_translation‹
[("_AOT_restriction", fn ctxt => fn [Const (name,_)] =>
let
val (restr, ctxt) = ctxt |> Local_Theory.raw_theory_result
```

```
  (fn thy => (Option.map fst (Symtab.lookup (AOT_Restriction.get thy) name), thy))
val restr = case restr of SOME x => x
  | _ => raise Fail ("Unknown restricted type: " ^ name)
in restr end
)]
›

print_translation‹
AOT_syntax_print_translations
[
  (const_syntax‹AOT_model_equiv_def›, fn ctxt => fn [x,y] =>
    Const (syntax_const‹_AOT_equiv_def›, dummyT) $
    (Const (syntax_const‹_AOT_process_frees›, dummyT) $ x) $
    (Const (syntax_const‹_AOT_process_frees›, dummyT) $ y))
]
›

print_translation‹
AOT_syntax_print_translations [
(const_syntax‹AOT_model_id_def›, fn ctxt =>
  fn [lhs as Abs (lhsName, lhsTy, lhsTrm), rhs as Abs (rhsName, rhsTy, rhsTrm)] =>
    let
      val (name,_) = Name.variant lhsName
        (Term.declare_term_names rhsTrm (Term.declare_term_names lhsTrm Name.context));
      val lhs = Term.betapply (lhs, Const ("_bound", dummyT) $ Free (name, lhsTy))
      val rhs = Term.betapply (rhs, Const ("_bound", dummyT) $ Free (name, rhsTy))
    in
      Const (const_syntax‹AOT_model_id_def›, dummyT) $ lhs $ rhs
    end
  | [Const (const_syntax‹case_prod›, _) $ lhs,
     Const (const_syntax‹case_prod›, _) $ rhs] =>
    Const (const_syntax‹AOT_model_id_def›, dummyT) $ lhs $ rhs
  | [Const (const_syntax‹case_unit›, _) $ lhs,
      Const (const_syntax‹case_unit›, _) $ rhs] =>
    Const (const_syntax‹AOT_model_id_def›, dummyT) $ lhs $ rhs
  | [x, y] =>
       Const (syntax_const‹_AOT_id_def›, dummyT) $
         (Const (syntax_const‹_AOT_process_frees›, dummyT) $ x) $
         (Const (syntax_const‹_AOT_process_frees›, dummyT) $ y)
)]›

text‹Special marker for printing propositions as theorems
     and for pretty-printing AOT terms.›
definition print_as_theorem :: ‹o ⇒ bool› where
  ‹print_as_theorem ≡ λ φ . ∀v . [v ⊨ φ]›
lemma print_as_theoremI:
  assumes ‹⋀ v . [v ⊨ φ]›
  shows ‹print_as_theorem φ›
  using assms by (simp add: print_as_theorem_def)
attribute_setup print_as_theorem =
  ‹Scan.succeed (Thm.rule_attribute []
      (K (fn thm => thm RS @{thm print_as_theoremI})))›
  "Print as theorem."
print_translation‹AOT_syntax_print_translations [
  (const_syntax‹print_as_theorem›, fn ctxt => fn [x] =>
   (Const (syntax_const‹_AOT_process_frees›, dummyT) $ x))
]›

definition print_term :: ‹'a ⇒ 'a› where ‹print_term ≡ λ x . x›
syntax "_AOT_print_term" :: ‹τ ⇒ 'a› (‹AOT'_TERM[_]›)
translations
  "_AOT_print_term φ" => "CONST print_term (_AOT_process_frees φ)"
print_translation‹AOT_syntax_print_translations [
```

```
  (const_syntax‹print_term›, fn ctxt => fn [x] =>
    (Const (syntax_const‹_AOT_process_frees›, dummyT) $ x))
]›


(* To enable meta syntax: *)
(* interpretation AOT_meta_syntax. *)
(* To disable meta syntax: *)
interpretation AOT_no_meta_syntax.

(* To enable AOT syntax (takes precedence over meta syntax;
                         can be done locally using "including" or "include"): *)
unbundle AOT_syntax
(* To disable AOT syntax (restoring meta syntax or no syntax;
                          can be done locally using "including" or "include"): *)
(* unbundle AOT_no_syntax *)

(*<*)
end
(*>*)
```

A.4. Semantics

```
(*<*)
theory AOT_semantics
  imports AOT_syntax
begin
(*>*)

section‹Abstract Semantics for AOT›

specification(AOT_denotes)
  ― ‹Relate object level denoting to meta-denoting. AOT's definitions of
      denoting will become derivable at each type.›
  AOT_sem_denotes: ‹[w ⊨ τ↓] = AOT_model_denotes τ›
  by (rule exI[where x=‹λ τ . εₒ w . AOT_model_denotes τ›])
     (simp add: AOT_model_proposition_choice_simp)

lemma AOT_sem_var_induct[induct type: AOT_var]:
  assumes AOT_denoting_term_case: ‹⋀ τ . [v ⊨ τ↓] ⟹ [v ⊨ φ{τ}]›
  shows ‹[v ⊨ φ{α}]›
  by (simp add: AOT_denoting_term_case AOT_sem_denotes AOT_term_of_var)

text‹\linelabel{AOT_imp_spec}›
specification(AOT_imp)
  AOT_sem_imp: ‹[w ⊨ φ → ψ] = ([w ⊨ φ] ⟶ [w ⊨ ψ])›
  by (rule exI[where x=‹λ φ ψ . εₒ w . ([w ⊨ φ] ⟶ [w ⊨ ψ])›])
    (simp add: AOT_model_proposition_choice_simp)

specification(AOT_not)
  AOT_sem_not: ‹[w ⊨ ¬φ] = (¬[w ⊨ φ])›
  by (rule exI[where x=‹λ φ . εₒ w . ¬[w ⊨ φ]›])
     (simp add: AOT_model_proposition_choice_simp)

text‹\linelabel{AOT_box_spec}›
specification(AOT_box)
  AOT_sem_box: ‹[w ⊨ □φ] = (∀ w . [w ⊨ φ])›
  by (rule exI[where x=‹λ φ . εₒ w . ∀ w . [w ⊨ φ]›])
     (simp add: AOT_model_proposition_choice_simp)

text‹\linelabel{AOT_act_spec}›
specification(AOT_act)
  AOT_sem_act: ‹[w ⊨ 𝒜φ] = [w₀ ⊨ φ]›
  by (rule exI[where x=‹λ φ . εₒ w . [w₀ ⊨ φ]›])
     (simp add: AOT_model_proposition_choice_simp)

text‹Derived semantics for basic defined connectives.›
lemma AOT_sem_conj: ‹[w ⊨ φ & ψ] = ([w ⊨ φ] ∧ [w ⊨ ψ])›
  using AOT_conj AOT_model_equiv_def AOT_sem_imp AOT_sem_not by auto
lemma AOT_sem_equiv: ‹[w ⊨ φ ≡ ψ] = ([w ⊨ φ] = [w ⊨ ψ])›
  using AOT_equiv AOT_sem_conj AOT_model_equiv_def AOT_sem_imp by auto
lemma AOT_sem_disj: ‹[w ⊨ φ ∨ ψ] = ([w ⊨ φ] ∨ [w ⊨ ψ])›
  using AOT_disj AOT_model_equiv_def AOT_sem_imp AOT_sem_not by auto
lemma AOT_sem_dia: ‹[w ⊨ ◇φ] = (∃ w . [w ⊨ φ])›
  using AOT_dia AOT_sem_box AOT_model_equiv_def AOT_sem_not by auto

specification(AOT_forall)
  AOT_sem_forall: ‹[w ⊨ ∀α φ{α}] = (∀ τ . [w ⊨ τ↓] ⟶ [w ⊨ φ{τ}])›
  by (rule exI[where x=‹λ op . εₒ w . ∀ τ . [w ⊨ τ↓] ⟶ [w ⊨ «op τ»]›])
     (simp add: AOT_model_proposition_choice_simp)

lemma AOT_sem_exists: ‹[w ⊨ ∃α φ{α}] = (∃ τ . [w ⊨ τ↓] ∧ [w ⊨ φ{τ}])›
  unfolding AOT_exists[unfolded AOT_model_equiv_def, THEN spec]
  by (simp add: AOT_sem_forall AOT_sem_not)
```

```
text‹\linelabel{AOT_eq_spec}›
specification(AOT_eq)
  – ‹Relate identity to denoting identity in the meta-logic. AOT's definitions
     of identity will become derivable at each type.›
  AOT_sem_eq: ‹[w ⊨ τ = τ'] = ([w ⊨ τ↓] ∧ [w ⊨ τ'↓] ∧ τ = τ')›
  by (rule exI[where x=‹λ τ τ' . ε₀ w . [w ⊨ τ↓] ∧ [w ⊨ τ'↓] ∧ τ = τ'›])
     (simp add: AOT_model_proposition_choice_simp)

text‹\linelabel{AOT_desc_spec}›
specification(AOT_desc)
  – ‹Descriptions denote, if there is a unique denoting object satisfying the
     matrix in the actual world.›
  AOT_sem_desc_denotes: ‹[w ⊨ ιx(φ{x})↓] = (∃! κ . [w₀ ⊨ κ↓] ∧ [w₀ ⊨ φ{κ}])›
  – ‹Denoting descriptions satisfy their matrix in the actual world.›
  AOT_sem_desc_prop: ‹[w ⊨ ιx(φ{x})↓] ⟹ [w₀ ⊨ φ{ιx(φ{x})}]›
  – ‹Uniqueness of denoting descriptions.›
  AOT_sem_desc_unique: ‹[w ⊨ ιx(φ{x})↓] ⟹ [w ⊨ κ↓] ⟹ [w₀ ⊨ φ{κ}] ⟹
                        [w ⊨ ιx(φ{x}) = κ]›
proof -
  have ‹∃x::'a . ¬AOT_model_denotes x›
    using AOT_model_nondenoting_ex
    by blast
  text‹Note that we may choose a distinct non-denoting object for each matrix.
       We do this explicitly merely to convince ourselves that our specification
       can still be satisfied.›
  then obtain nondenoting :: ‹('a ⇒ o) ⇒ 'a› where
    nondenoting: ‹∀ φ . ¬AOT_model_denotes (nondenoting φ)›
    by fast
  define desc where
    ‹desc = (λ φ . if (∃! κ . [w₀ ⊨ κ↓] ∧ [w₀ ⊨ φ{κ}])
                   then (THE κ . [w₀ ⊨ κ↓] ∧ [w₀ ⊨ φ{κ}])
                   else nondenoting φ)›
  {
    fix φ :: ‹'a ⇒ o›
    assume ex1: ‹∃! κ . [w₀ ⊨ κ↓] ∧ [w₀ ⊨ φ{κ}]›
    then obtain κ where x_prop: "[w₀ ⊨ κ↓] ∧ [w₀ ⊨ φ{κ}]"
      unfolding AOT_sem_denotes by blast
    moreover have "(desc φ) = κ"
      unfolding desc_def using x_prop ex1 by fastforce
    ultimately have "[w₀ ⊨ «desc φ»↓] ∧ [w₀ ⊨ «φ (desc φ)»]"
      by blast
  } note 1 = this
  moreover {
    fix φ :: ‹'a ⇒ o›
    assume nex1: ‹∄! κ . [w₀ ⊨ κ↓] ∧ [w₀ ⊨ φ{κ}]›
    hence "(desc φ) = nondenoting φ" by (simp add: desc_def AOT_sem_denotes)
    hence "[w ⊨ ¬«desc φ»↓]" for w
      by (simp add: AOT_sem_denotes nondenoting AOT_sem_not)
  }
  ultimately have desc_denotes_simp:
    ‹[w ⊨ «desc φ»↓] = (∃! κ . [w₀ ⊨ κ↓] ∧ [w₀ ⊨ φ{κ}])› for φ w
    by (simp add: AOT_sem_denotes desc_def nondenoting)
  have ‹(∀φ w. [w ⊨ «desc φ»↓] = (∃!κ. [w₀ ⊨ κ↓] ∧ [w₀ ⊨ φ{κ}])) ∧
    (∀φ w. [w ⊨ «desc φ»↓] ⟶ [w₀ ⊨ «φ (desc φ)»]) ∧
    (∀φ w κ. [w ⊨ «desc φ»↓] ⟶ [w ⊨ κ↓] ⟶ [w₀ ⊨ φ{κ}] ⟶
             [w ⊨ «desc φ» = κ])›
    by (insert 1; auto simp: desc_denotes_simp AOT_sem_eq AOT_sem_denotes
                             desc_def nondenoting)
  thus ?thesis
    by (safe intro!: exI[where x=desc]; presburger)
qed
```

```
text‹\linelabel{AOT_exe_lambda_spec}›
specification(AOT_exe AOT_lambda)
  — ‹Truth conditions of exemplification formulas.›
  AOT_sem_exe: ‹[w ⊨ [Π]κ1...κn] = ([w ⊨ Π↓] ∧ [w ⊨ κ1...κn↓] ∧
                                     [w ⊨ «Rep_rel Π κ1κn»])›
  — ‹η-conversion for denoting terms; equivalent to AOT's axiom›
  AOT_sem_lambda_eta: ‹[w ⊨ Π↓] ⟹ [w ⊨ [λν1...νn [Π]ν1...νn] = Π]›
  — ‹β-conversion; equivalent to AOT's axiom›
  AOT_sem_lambda_beta: ‹[w ⊨ [λν1...νn φ{ν1...νn}]↓] ⟹ [w ⊨ κ1...κn↓] ⟹
                         [w ⊨ [λν1...νn φ{ν1...νn}]κ1...κn] = [w ⊨ φ{κ1...κn}]›
  — ‹Necessary and sufficient conditions for relations to denote. Equivalent
      to a theorem of AOT and used to derive the base cases of denoting relations
      (cqt.2).›
  AOT_sem_lambda_denotes: ‹[w ⊨ [λν1...νn φ{ν1...νn}]↓] =
    (∀ v κ1κn κ1'κn' . [v ⊨ κ1...κn↓] ∧ [v ⊨ κ1'...κn'↓] ∧
        (∀ Π v . [v ⊨ Π↓] ⟶ [v ⊨ [Π]κ1...κn] = [v ⊨ [Π]κ1'...κn']) ⟶
                 [v ⊨ φ{κ1...κn}] = [v ⊨ φ{κ1'...κn'}])›
  — ‹Equivalent to AOT's coexistence axiom.›
  AOT_sem_lambda_coex: ‹[w ⊨ [λν1...νn φ{ν1...νn}]↓] ⟹
    (∀ w κ1κn . [w ⊨ κ1...κn↓] ⟶ [w ⊨ φ{κ1...κn}] = [w ⊨ ψ{κ1...κn}]) ⟹
    [w ⊨ [λν1...νn ψ{ν1...νn}]↓]›
  — ‹Only the unary case of the following should hold, but our specification
      has to range over all types. We might move @{const AOT_exe} and
      @{const AOT_lambda} to type classes in the future to solve this.›
  AOT_sem_lambda_eq_prop_eq: ‹«[λν1...νn φ]» = «[λν1...νn ψ]» ⟹ φ = ψ›
  — ‹The following is solely required for validating n-ary relation identity
      and has the danger of implying artifactual theorems. Possibly avoidable
      by moving @{const AOT_exe} and @{const AOT_lambda} to type classes.›
  AOT_sem_exe_denoting: ‹[w ⊨ Π↓] ⟹ AOT_exe Π κs = Rep_rel Π κs›
  — ‹The following is required for validating the base cases of denoting
      relations (cqt.2). A version of this meta-logical identity will
      become derivable in future versions of AOT, so this will ultimately not
      result in artifactual theorems.›
  AOT_sem_exe_equiv: ‹AOT_model_term_equiv x y ⟹ AOT_exe Π x = AOT_exe Π y›
proof -
  have ‹∃ x :: ‹'a› . ¬AOT_model_denotes x›
    by (rule exI[where x=‹Abs_rel (λ x . ε○ w. True)›])
       (meson AOT_model_denotes_rel.abs_eq AOT_model_nondenoting_ex
              AOT_model_proposition_choice_simp)
  define exe :: ‹<'a> ⇒ 'a ⇒ o› where
    ‹exe ≡ λ Π κs . if AOT_model_denotes Π
                    then Rep_rel Π κs
                    else (ε○ w . False)›
  define lambda :: ‹('a⇒o) ⇒ <'a>› where
    ‹lambda ≡ λ φ . if AOT_model_denotes (Abs_rel φ)
      then (Abs_rel φ)
      else
        if (∀ κ κ' w . (AOT_model_denotes κ ∧ AOT_model_term_equiv κ κ') ⟶
                       [w ⊨ «φ κ»] = [w ⊨ «φ κ'»])
        then
          Abs_rel (fix_irregular (λ x . if AOT_model_denotes x
                                        then φ (SOME y . AOT_model_term_equiv x y)
                                        else  (ε○ w . False)))
        else
          Abs_rel φ›
  have fix_irregular_denoting_simp[simp]:
    ‹fix_irregular (λx. if AOT_model_denotes x then φ x else ψ x) κ = φ κ›
    if ‹AOT_model_denotes κ›
    for κ :: 'a and φ ψ
    by (simp add: that fix_irregular_denoting)
  have denoting_eps_cong[cong]:
    ‹[w ⊨ «φ (Eps (AOT_model_term_equiv κ))»] = [w ⊨ «φ κ»]›
    if ‹AOT_model_denotes κ›
```

```
    and ‹∀ κ κ'. AOT_model_denotes κ ∧ AOT_model_term_equiv κ κ' ⟶
                (∀w. [w ⊨ «φ κ»] = [w ⊨ «φ κ'»])›
    for w :: w and κ :: 'a and φ :: ‹'a⇒o›
    using that AOT_model_term_equiv_eps(2) by blast
  have exe_rep_rel: ‹[w ⊨ «exe Π κ1κn»] = ([w ⊨ Π↓] ∧ [w ⊨ κ1...κn↓] ∧
                                            [w ⊨ «Rep_rel Π κ1κn»])› for w Π κ1κn
    by (metis AOT_model_denotes_rel.rep_eq exe_def AOT_sem_denotes
         AOT_model_proposition_choice_simp)
  moreover have ‹[w ⊨ «Π»↓] ⟹ [w ⊨ «lambda (exe Π)» = «Π»]› for Π w
    by (auto simp: Rep_rel_inverse lambda_def AOT_sem_denotes AOT_sem_eq
             AOT_model_denotes_rel_def Abs_rel_inverse exe_def)
  moreover have lambda_denotes_beta:
    ‹[w ⊨ «exe (lambda φ) κ »] = [w ⊨ «φ κ»]›
    if ‹[w ⊨ «lambda φ»↓]› and ‹[w ⊨ «κ»↓]›
    for φ κ w
    using that unfolding exe_def AOT_sem_denotes
    by (auto simp: lambda_def Abs_rel_inverse split: if_split_asm)
  moreover have lambda_denotes_simp:
    ‹[w ⊨ «lambda φ»↓] = (∀ v κ1κn κ1'κn' . [v ⊨ κ1...κn↓] ∧ [v ⊨ κ1'...κn'↓] ∧
      (∀ Π v . [v ⊨ Π↓] ⟶ [v ⊨ «exe Π κ1κn»] = [v ⊨ «exe Π κ1'κn'»]) ⟶
      [v ⊨ φ{κ1...κn}] = [v ⊨ φ{κ1'...κn'}])› for φ w
  proof
    assume ‹[w ⊨ «lambda φ»↓]›
    hence ‹AOT_model_denotes (lambda φ)›
      unfolding AOT_sem_denotes by simp
    moreover have ‹[w ⊨ «φ κ»] ⟹ [w ⊨ «φ κ'»]›
      and ‹[w ⊨ «φ κ'»] ⟹ [w ⊨ «φ κ»]›
      if ‹AOT_model_denotes κ› and ‹AOT_model_term_equiv κ κ'›
      for w κ κ'
      by (metis (no_types, lifting) AOT_model_denotes_rel.abs_eq lambda_def
                                     that calculation)+
    ultimately show ‹∀ v κ1κn κ1'κn' . [v ⊨ κ1...κn↓] ∧ [v ⊨ κ1'...κn'↓] ∧
      (∀ Π v . [v ⊨ Π↓] ⟶ [v ⊨ «exe Π κ1κn»] = [v ⊨ «exe Π κ1'κn'»]) ⟶
      [v ⊨ φ{κ1...κn}] = [v ⊨ φ{κ1'...κn'}]›
      unfolding AOT_sem_denotes
      by (metis (no_types) AOT_sem_denotes lambda_denotes_beta)
  next
    assume ‹∀ v κ1κn κ1'κn' . [v ⊨ κ1...κn↓] ∧ [v ⊨ κ1'...κn'↓] ∧
     (∀ Π v . [v ⊨ Π↓] ⟶ [v ⊨ «exe Π κ1κn»] = [v ⊨ «exe Π κ1'κn'»]) ⟶
     [v ⊨ φ{κ1...κn}] = [v ⊨ φ{κ1'...κn'}]›
    hence ‹[w ⊨ «φ κ»] = [w ⊨ «φ κ'»]›
      if ‹AOT_model_denotes κ ∧ AOT_model_denotes κ' ∧ AOT_model_term_equiv κ κ'›
      for w κ κ'
      using that
      by (auto simp: AOT_sem_denotes)
         (meson AOT_model_term_equiv_rel_equiv AOT_sem_denotes exe_rep_rel)+
    hence ‹[w ⊨ «φ κ»] = [w ⊨ «φ κ'»]›
      if ‹AOT_model_denotes κ ∧ AOT_model_term_equiv κ κ'›
      for w κ κ'
      using that AOT_model_term_equiv_denotes by blast
    hence ‹AOT_model_denotes (lambda φ)›
      by (auto simp: lambda_def Abs_rel_inverse AOT_model_denotes_rel.abs_eq
                     AOT_model_irregular_equiv AOT_model_term_equiv_eps(3)
                     AOT_model_term_equiv_regular fix_irregular_def AOT_sem_denotes
                     AOT_model_term_equiv_denotes AOT_model_proposition_choice_simp
                     AOT_model_irregular_false
               split: if_split_asm
               intro: AOT_model_irregular_eqI)
    thus ‹[w ⊨ «lambda φ»↓]›
      by (simp add: AOT_sem_denotes)
  qed
  moreover have ‹[w ⊨ «lambda ψ»↓]›
    if ‹[w ⊨ «lambda φ»↓]›
```

```
    and ‹∀ w κ1κn . [w ⊨ κ1...κn↓] ⟶ [w ⊨ φ{κ1...κn}] = [w ⊨ ψ{κ1...κn}]›
    for φ ψ w using that unfolding lambda_denotes_simp by auto
  moreover have ‹[w ⊨ Π↓] ⟹ exe Π κs = Rep_rel Π κs› for Π κs w
    by (simp add: exe_def AOT_sem_denotes)
  moreover have ‹lambda (λx. p) = lambda (λx. q) ⟹ p = q› for p q
    unfolding lambda_def
    by (auto split: if_split_asm simp: Abs_rel_inject fix_irregular_def)
       (meson AOT_model_irregular_nondenoting AOT_model_denoting_ex)+
  moreover have ‹AOT_model_term_equiv x y ⟹ exe Π x = exe Π y› for x y Π
    unfolding exe_def
    by (meson AOT_model_denotes_rel.rep_eq)
  note calculation = calculation this
  show ?thesis
    apply (safe intro!: exI[where x=exe] exI[where x=lambda])
    using calculation apply simp_all
    using lambda_denotes_simp by blast+
qed

lemma AOT_model_lambda_denotes:
  ‹AOT_model_denotes (AOT_lambda φ) = (∀ v κ κ' .
    AOT_model_denotes κ ∧ AOT_model_denotes κ' ∧ AOT_model_term_equiv κ κ' ⟶
    [v ⊨ «φ κ»] = [v ⊨ «φ κ'»])›
proof(safe)
  fix v and κ κ' :: 'a
  assume ‹AOT_model_denotes (AOT_lambda φ)›
  hence 1: ‹AOT_model_denotes κ1κn ∧
        AOT_model_denotes κ1'κn' ∧
        (∀Π v. AOT_model_denotes Π ⟶ [v ⊨ [Π]κ1...κn] = [v ⊨ [Π]κ1'...κn'])  ⟶
        [v ⊨ φ{κ1...κn}] = [v ⊨ φ{κ1'...κn'}]› for κ1κn κ1'κn' v
    using AOT_sem_lambda_denotes[simplified AOT_sem_denotes] by blast
  {
    fix v and κ1κn κ1'κn' :: 'a
    assume d: ‹AOT_model_denotes κ1κn ∧ AOT_model_denotes κ1'κn' ∧
               AOT_model_term_equiv κ1κn κ1'κn'›
    hence ‹∀Π w. AOT_model_denotes Π ⟶ [w ⊨ [Π]κ1...κn] = [w ⊨ [Π]κ1'...κn']›
      by (metis AOT_sem_exe_equiv)
    hence ‹[v ⊨ φ{κ1...κn}] = [v ⊨ φ{κ1'...κn'}]› using d 1 by auto
  }
  moreover assume ‹AOT_model_denotes κ›
  moreover assume ‹AOT_model_denotes κ'›
  moreover assume ‹AOT_model_term_equiv κ κ'›
  ultimately show ‹[v ⊨ «φ κ»] ⟹ [v ⊨ «φ κ'»]›
             and ‹[v ⊨ «φ κ'»] ⟹ [v ⊨ «φ κ»]›
    by auto
next
  assume 0: ‹∀ v κ κ' . AOT_model_denotes κ ∧ AOT_model_denotes κ' ∧
                        AOT_model_term_equiv κ κ' ⟶ [v ⊨ «φ κ»] = [v ⊨ «φ κ'»]›
  {
    fix κ1κn κ1'κn' :: 'a
    assume den: ‹AOT_model_denotes κ1κn›
    moreover assume den': ‹AOT_model_denotes κ1'κn'›
    assume ‹∀Π v . AOT_model_denotes Π ⟶
                   [v ⊨ [Π]κ1...κn] = [v ⊨ [Π]κ1'...κn']›
    hence ‹∀Π v . AOT_model_denotes Π ⟶
                   [v ⊨ «Rep_rel Π κ1κn»] = [v ⊨ «Rep_rel Π κ1'κn'»]›
      by (simp add: AOT_sem_denotes AOT_sem_exe den den')
    hence "AOT_model_term_equiv κ1κn κ1'κn'"
      unfolding AOT_model_term_equiv_rel_equiv[OF den, OF den']
      by argo
    hence ‹[v ⊨ φ{κ1...κn}] = [v ⊨ φ{κ1'...κn'}]› for v
      using den den' 0 by blast
  }
  thus ‹AOT_model_denotes (AOT_lambda φ)›
```

```
    unfolding AOT_sem_lambda_denotes[simplified AOT_sem_denotes]
    by blast
qed

specification (AOT_lambda0)
  AOT_sem_lambda0: "AOT_lambda0 φ = φ"
  by (rule exI[where x=‹λx. x›]) simp

specification(AOT_concrete)
  AOT_sem_concrete: ‹[w ⊨ [E!]κ] =
                     AOT_model_concrete w κ›
  by (rule exI[where x=‹AOT_var_of_term (Abs_rel
                          (λ x . ε∘ w . AOT_model_concrete w x))›];
      subst AOT_var_of_term_inverse)
     (auto simp: AOT_model_unary_regular AOT_model_concrete_denotes
                 AOT_model_concrete_equiv AOT_model_regular_κ_def
                 AOT_model_proposition_choice_simp AOT_sem_exe Abs_rel_inverse
                 AOT_model_denotes_rel_def AOT_sem_denotes)

lemma AOT_sem_equiv_defI:
  assumes ‹⋀ v . [v ⊨ φ] ⟹ [v ⊨ ψ]›
      and ‹⋀ v . [v ⊨ ψ] ⟹ [v ⊨ φ]›
    shows ‹AOT_model_equiv_def φ ψ›
  using AOT_model_equiv_def assms by blast

lemma AOT_sem_id_defI:
  assumes ‹⋀ α v . [v ⊨ «σ α»↓] ⟹ [v ⊨ «τ α» = «σ α»]›
  assumes ‹⋀ α v . ¬[v ⊨ «σ α»↓] ⟹ [v ⊨ ¬«τ α»↓]›
  shows ‹AOT_model_id_def τ σ›
  using assms
  unfolding AOT_sem_denotes AOT_sem_eq AOT_sem_not
  using AOT_model_id_def[THEN iffD2]
  by metis

lemma AOT_sem_id_def2I:
  assumes ‹⋀ α β v . [v ⊨ «σ α β»↓] ⟹ [v ⊨ «τ α β» = «σ α β»]›
  assumes ‹⋀ α β v . ¬[v ⊨ «σ α β»↓] ⟹ [v ⊨ ¬«τ α β»↓]›
  shows ‹AOT_model_id_def (case_prod τ) (case_prod σ)›
  apply (rule AOT_sem_id_defI)
  using assms by (auto simp: AOT_sem_conj AOT_sem_not AOT_sem_eq AOT_sem_denotes
      AOT_model_denotes_prod_def)

lemma AOT_sem_id_defE1:
  assumes ‹AOT_model_id_def τ σ›
      and ‹[v ⊨ «σ α»↓]›
    shows ‹[v ⊨ «τ α» = «σ α»]›
  using assms by (simp add: AOT_model_id_def AOT_sem_denotes AOT_sem_eq)

lemma AOT_sem_id_defE2:
  assumes ‹AOT_model_id_def τ σ›
      and ‹¬[v ⊨ «σ α»↓]›
    shows ‹¬[v ⊨ «τ α»↓]›
  using assms by (simp add: AOT_model_id_def AOT_sem_denotes AOT_sem_eq)

lemma AOT_sem_id_def0I:
  assumes ‹⋀ v . [v ⊨ σ↓] ⟹ [v ⊨ τ = σ]›
      and ‹⋀ v . ¬[v ⊨ σ↓] ⟹ [v ⊨ ¬τ↓]›
  shows ‹AOT_model_id_def (case_unit τ) (case_unit σ)›
  apply (rule AOT_sem_id_defI)
  using assms
  by (simp_all add: AOT_sem_conj AOT_sem_eq AOT_sem_not AOT_sem_denotes
                    AOT_model_denotes_unit_def case_unit_Unity)
```

```
lemma AOT_sem_id_def0E1:
  assumes ‹AOT_model_id_def (case_unit τ) (case_unit σ)›
      and ‹[v ⊨ σ↓]›
    shows ‹[v ⊨ τ = σ]›
  by (metis (full_types) AOT_sem_id_defE1 assms(1) assms(2) case_unit_Unity)

lemma AOT_sem_id_def0E2:
  assumes ‹AOT_model_id_def (case_unit τ) (case_unit σ)›
      and ‹¬[v ⊨ σ↓]›
    shows ‹¬[v ⊨ τ↓]›
  by (metis AOT_sem_id_defE2 assms(1) assms(2) case_unit_Unity)

lemma AOT_sem_id_def0E3:
  assumes ‹AOT_model_id_def (case_unit τ) (case_unit σ)›
      and ‹[v ⊨ σ↓]›
    shows ‹[v ⊨ τ↓]›
  using AOT_sem_id_def0E1[OF assms]
  by (simp add: AOT_sem_eq AOT_sem_denotes)

lemma AOT_sem_ordinary_def_denotes: ‹[w ⊨ [λx ◊[E!]x]↓]›
  unfolding AOT_sem_denotes AOT_model_lambda_denotes
  by (auto simp: AOT_sem_dia AOT_model_concrete_equiv
                 AOT_sem_concrete AOT_sem_denotes)
lemma AOT_sem_abstract_def_denotes: ‹[w ⊨ [λx ¬◊[E!]x]↓]›
  unfolding AOT_sem_denotes AOT_model_lambda_denotes
  by (auto simp: AOT_sem_dia AOT_model_concrete_equiv
                 AOT_sem_concrete AOT_sem_denotes AOT_sem_not)

text‹Relation identity is constructed using an auxiliary abstract projection
     mechanism with suitable instantiations for @{typ κ} and products.›
class AOT_RelationProjection =
  fixes AOT_sem_proj_id :: ‹'a::AOT_IndividualTerm ⇒ ('a ⇒ o) ⇒ ('a ⇒ o) ⇒ o›
  assumes AOT_sem_proj_id_prop:
    ‹[v ⊨ Π = Π'] =
     [v ⊨ Π↓ & Π'↓ & ∀α («AOT_sem_proj_id α (λ τ . «[Π]τ») (λ τ . «[Π']τ»)»)]›
      and AOT_sem_proj_id_refl:
    ‹[v ⊨ τ↓] ⟹ [v ⊨ [λν₁...νₙ φ{ν₁...νₙ}] = [λν₁...νₙ φ{ν₁...νₙ}]] ⟹
     [v ⊨ «AOT_sem_proj_id τ φ φ»]›

class AOT_UnaryRelationProjection = AOT_RelationProjection +
  assumes AOT_sem_unary_proj_id:
    ‹AOT_sem_proj_id κ φ ψ = «[λν₁...νₙ φ{ν₁...νₙ}] = [λν₁...νₙ ψ{ν₁...νₙ}]»›

instantiation κ :: AOT_UnaryRelationProjection
begin
definition AOT_sem_proj_id_κ :: ‹κ ⇒ (κ ⇒ o) ⇒ (κ ⇒ o) ⇒ o› where
  ‹AOT_sem_proj_id_κ κ φ ψ = «[λz φ{z}] = [λz ψ{z}]»›
instance proof
  show ‹[v ⊨ Π = Π'] =
         [v ⊨ Π↓ & Π'↓ & ∀α («AOT_sem_proj_id α (λ τ . «[Π]τ») (λ τ . «[Π']τ»)»)]›
    for v and Π Π' :: ‹<κ>›
    unfolding AOT_sem_proj_id_κ_def
    by (simp add: AOT_sem_eq AOT_sem_conj AOT_sem_denotes AOT_sem_forall)
       (metis AOT_sem_denotes AOT_model_denoting_ex AOT_sem_eq AOT_sem_lambda_eta)
next
  show ‹AOT_sem_proj_id κ φ ψ = «[λν₁...νₙ φ{ν₁...νₙ}] = [λν₁...νₙ ψ{ν₁...νₙ}]»›
    for κ :: κ and φ ψ
    unfolding AOT_sem_proj_id_κ_def ..
next
  show ‹[v ⊨ «AOT_sem_proj_id τ φ φ»]›
    if ‹[v ⊨ τ↓]› and ‹[v ⊨ [λν₁...νₙ φ{ν₁...νₙ}] = [λν₁...νₙ φ{ν₁...νₙ}]]›
    for τ :: κ and v φ
    using that by (simp add: AOT_sem_proj_id_κ_def AOT_sem_eq)
```

```
qed
end

instantiation prod ::
  ("{AOT_UnaryRelationProjection, AOT_UnaryIndividualTerm}", AOT_RelationProjection)
  AOT_RelationProjection
begin
definition AOT_sem_proj_id_prod :: ‹'a×'b ⇒ ('a×'b ⇒ o) ⇒ ('a×'b ⇒ o) ⇒ o› where
  ‹AOT_sem_proj_id_prod ≡ λ (x,y) φ ψ . «[λz «φ (z,y)»] = [λz «ψ (z,y)»] &
    «AOT_sem_proj_id y (λ a . φ (x,a)) (λ a . ψ (x,a))»»›
instance proof
  text‹This is the main proof that allows to derive the definition of n-ary
       relation identity. We need to show that our defined projection identity
       implies relation identity for relations on pairs of individual terms.›
  fix v and Π Π' :: ‹<'a×'b>›
  have AOT_meta_proj_denotes1: ‹AOT_model_denotes (Abs_rel (λz. AOT_exe Π (z, β)))›
    if ‹AOT_model_denotes Π› for Π :: ‹<'a×'b>› and β
    using that unfolding AOT_model_denotes_rel.rep_eq
    apply (auto simp: Abs_rel_inverse AOT_meta_prod_equivI(2) AOT_sem_denotes
                      that intro!: AOT_sem_exe_equiv)
    apply (metis AOT_model_denotes_prod_def AOT_sem_exe case_prodD)
    using AOT_model_unary_regular by blast
  {
    fix κ :: 'a and Π :: ‹<'a×'b>›
    assume Π_denotes: ‹AOT_model_denotes Π›
    assume α_denotes: ‹AOT_model_denotes κ›
    hence ‹AOT_exe Π (κ, x) = AOT_exe Π (κ, y)›
      if ‹AOT_model_term_equiv x y› for x y :: 'b
      by (simp add: AOT_meta_prod_equivI(1) AOT_sem_exe_equiv that)
    moreover have ‹AOT_model_denotes κ₁'κₙ'›
                 if ‹[w ⊨ [Π]κ κ₁'...κₙ']› for w κ₁'κₙ'
      by (metis that AOT_model_denotes_prod_def AOT_sem_exe
                AOT_sem_denotes case_prodD)
    moreover {
      fix x :: 'b
      assume x_irregular: ‹¬AOT_model_regular x›
      hence prod_irregular: ‹¬AOT_model_regular (κ, x)›
        by (metis (no_types, lifting) AOT_model_irregular_nondenoting
                                      AOT_model_regular_prod_def case_prodD)
      hence ‹(¬AOT_model_denotes κ ∨ ¬AOT_model_regular x) ∧
             (AOT_model_denotes κ ∨ ¬AOT_model_denotes x)›
        unfolding AOT_model_regular_prod_def by blast
      hence x_nonden: ‹¬AOT_model_regular x›
        by (simp add: α_denotes)
      have ‹Rep_rel Π (κ, x) = AOT_model_irregular (Rep_rel Π) (κ, x)›
        using AOT_model_denotes_rel.rep_eq Π_denotes prod_irregular by blast
      moreover have ‹AOT_model_irregular (Rep_rel Π) (κ, x) =
                     AOT_model_irregular (λz. Rep_rel Π (κ, z)) x›
        using Π_denotes x_irregular prod_irregular x_nonden
        using AOT_model_irregular_prod_generic
        apply (induct arbitrary: Π x rule: AOT_model_irregular_prod.induct)
        by (auto simp: α_denotes AOT_model_irregular_nondenoting
                       AOT_model_regular_prod_def AOT_meta_prod_equivI(2)
                       AOT_model_denotes_rel.rep_eq AOT_model_term_equiv_eps(1)
                 intro!: AOT_model_irregular_eqI)
      ultimately have
        ‹AOT_exe Π (κ, x) = AOT_model_irregular (λz. AOT_exe Π (κ, z)) x›
        unfolding AOT_sem_exe_denoting[simplified AOT_sem_denotes, OF Π_denotes]
        by auto
    }
    ultimately have ‹AOT_model_denotes (Abs_rel (λz. AOT_exe Π (κ, z)))›
      by (simp add: Abs_rel_inverse AOT_model_denotes_rel.rep_eq)
  } note AOT_meta_proj_denotes2 = this
```

```
{
  fix κ1'κn' :: 'b and Π :: <<'a×'b>>
  assume Π_denotes: <AOT_model_denotes Π>
  assume β_denotes: <AOT_model_denotes κ1'κn'>
  hence <AOT_exe Π (x, κ1'κn') = AOT_exe Π (y, κ1'κn')>
    if <AOT_model_term_equiv x y> for x y :: 'a
    by (simp add: AOT_meta_prod_equivI(2) AOT_sem_exe_equiv that)
  moreover have <AOT_model_denotes κ>
             if <[w ⊨ [Π]κ κ1'...κn']> for w κ
    by (metis that AOT_model_denotes_prod_def AOT_sem_exe
              AOT_sem_denotes case_prodD)
  moreover {
    fix x :: 'a
    assume <¬AOT_model_regular x>
    hence <False>
      using AOT_model_unary_regular by blast
    hence
      <AOT_exe Π (x,κ1'κn') = AOT_model_irregular (λz. AOT_exe Π (z,κ1'κn')) x>
      unfolding AOT_sem_exe_denoting[simplified AOT_sem_denotes, OF Π_denotes]
      by auto
  }
  ultimately have <AOT_model_denotes (Abs_rel (λz. AOT_exe Π (z,κ1'κn')))>
    by (simp add: Abs_rel_inverse AOT_model_denotes_rel.rep_eq)
} note AOT_meta_proj_denotes1 = this
{
  assume Π_denotes: <AOT_model_denotes Π>
  assume Π'_denotes: <AOT_model_denotes Π'>
  have Π_proj2_den: <AOT_model_denotes (Abs_rel (λz. Rep_rel Π (α, z)))>
    if <AOT_model_denotes α> for α
    using that AOT_meta_proj_denotes2[OF Π_denotes]
          AOT_sem_exe_denoting[simplified AOT_sem_denotes,OF Π_denotes] by simp
  have Π'_proj2_den: <AOT_model_denotes (Abs_rel (λz. Rep_rel Π' (α, z)))>
    if <AOT_model_denotes α> for α
    using that AOT_meta_proj_denotes2[OF Π'_denotes]
          AOT_sem_exe_denoting[simplified AOT_sem_denotes,OF Π'_denotes] by simp
  have Π_proj1_den: <AOT_model_denotes (Abs_rel (λz. Rep_rel Π (z, α)))>
    if <AOT_model_denotes α> for α
    using that AOT_meta_proj_denotes1[OF Π_denotes]
          AOT_sem_exe_denoting[simplified AOT_sem_denotes,OF Π_denotes] by simp
  have Π'_proj1_den: <AOT_model_denotes (Abs_rel (λz. Rep_rel Π' (z, α)))>
    if <AOT_model_denotes α> for α
    using that AOT_meta_proj_denotes1[OF Π'_denotes]
          AOT_sem_exe_denoting[simplified AOT_sem_denotes,OF Π'_denotes] by simp
  {
    fix κ :: 'a and κ1'κn' :: 'b
    assume <Π = Π'>
    assume <AOT_model_denotes (κ,κ1'κn')>
    hence <AOT_model_denotes κ> and beta_denotes: <AOT_model_denotes κ1'κn'>
      by (auto simp: AOT_model_denotes_prod_def)
    have <AOT_model_denotes «[λz [Π]z κ1'...κn']»>
      by (rule AOT_model_lambda_denotes[THEN iffD2])
         (metis AOT_sem_exe_denoting AOT_meta_prod_equivI(2)
                AOT_model_denotes_rel.rep_eq AOT_sem_denotes
                AOT_sem_exe_denoting Π_denotes)
    moreover have <«[λz [Π]z κ1'...κn']» = «[λz [Π']z κ1'...κn']»>
      by (simp add: <Π = Π'>)
    moreover have <[v ⊨ «AOT_sem_proj_id κ1'κn' (λκ1'κn'. «[Π]κ κ1'...κn'»)
                                              (λκ1'κn'. «[Π']κ κ1'...κn'»)»]>
      unfolding <Π = Π'> using beta_denotes
      by (rule AOT_sem_proj_id_refl[unfolded AOT_sem_denotes];
          simp add: AOT_sem_denotes AOT_sem_eq AOT_model_lambda_denotes)
         (metis AOT_meta_prod_equivI(1) AOT_model_denotes_rel.rep_eq
                AOT_sem_exe AOT_sem_exe_denoting Π'_denotes)
```

```
      ultimately have ‹[v ⊨ «AOT_sem_proj_id (κ,κ1'κn') (λ κ1κn . «[Π]κ1...κn»)
                                                    (λ κ1κn . «[Π']κ1...κn»)»]›
        unfolding AOT_sem_proj_id_prod_def
        by (simp add: AOT_sem_denotes AOT_sem_conj AOT_sem_eq)
    }
    moreover {
      assume ‹⋀ α . AOT_model_denotes α ⟹
        [v ⊨ «AOT_sem_proj_id α (λ κ1κn . «[Π]κ1...κn») (λ κ1κn . «[Π']κ1...κn»)»]›
      hence 0: ‹AOT_model_denotes κ ⟹ AOT_model_denotes κ1'κn' ⟹
              AOT_model_denotes «[λz [Π]z κ1'...κn']» ∧
              AOT_model_denotes «[λz [Π']z κ1'...κn']» ∧
              «[λz [Π]z κ1'...κn']» = «[λz [Π']z κ1'...κn']» ∧
              [v ⊨ «AOT_sem_proj_id κ1'κn' (λκ1κn. «[Π]κ κ1...κn»)
                                          (λκ1κn. «[Π']κ κ1...κn»)»]› for κ κ1'κn'
        unfolding AOT_sem_proj_id_prod_def
        by (auto simp: AOT_sem_denotes AOT_sem_conj AOT_sem_eq
                       AOT_model_denotes_prod_def)
      obtain αden :: 'a and βden :: 'b where
        αden: ‹AOT_model_denotes αden› and βden: ‹AOT_model_denotes βden›
        using AOT_model_denoting_ex by metis
      {
        fix κ :: 'a
        assume αdenotes: ‹AOT_model_denotes κ›
        have 1: ‹[v ⊨ «AOT_sem_proj_id κ1'κn' (λκ1'κn'. «[Π]κ κ1'...κn'»)
                                            (λκ1'κn'. «[Π']κ κ1'...κn'»)»]›
          if ‹AOT_model_denotes κ1'κn'› for κ1'κn'
          using that 0 using αdenotes by blast
        hence ‹[v ⊨ «AOT_sem_proj_id β (λz. Rep_rel Π (κ, z))
                                       (λz. Rep_rel Π' (κ, z))»]›
          if ‹AOT_model_denotes β› for β
          using that
          unfolding AOT_sem_exe_denoting[simplified AOT_sem_denotes, OF Π_denotes]
                    AOT_sem_exe_denoting[simplified AOT_sem_denotes, OF Π'_denotes]
          by blast
        hence ‹Abs_rel (λz. Rep_rel Π (κ, z)) = Abs_rel (λz. Rep_rel Π' (κ, z))›
          using AOT_sem_proj_id_prop[of v ‹Abs_rel (λz. Rep_rel Π (κ, z))›
                                          ‹Abs_rel (λz. Rep_rel Π' (κ, z))›,
                simplified AOT_sem_eq AOT_sem_conj AOT_sem_forall
                           AOT_sem_denotes, THEN iffD2]
              Π_proj2_den[OF αdenotes] Π'_proj2_den[OF αdenotes]
          unfolding AOT_sem_exe_denoting[simplified AOT_sem_denotes, OF Π_denotes]
                    AOT_sem_exe_denoting[simplified AOT_sem_denotes,
                                         OF Π_proj2_den[OF αdenotes]]
                    AOT_sem_exe_denoting[simplified AOT_sem_denotes,
                                         OF Π'_proj2_den[OF αdenotes]]
          by (metis Abs_rel_inverse UNIV_I)
        hence "Rep_rel Π (κ,β) = Rep_rel Π' (κ,β)" for β
          by (simp add: Abs_rel_inject[simplified]) meson
      } note αdenotes = this
      {
        fix κ1'κn' :: 'b
        assume βden: ‹AOT_model_denotes κ1'κn'›
        have 1: ‹«[λz [Π]z κ1'...κn']» = «[λz [Π']z κ1'...κn']»›
          using 0 βden AOT_model_denoting_ex by blast
        hence ‹Abs_rel (λz. Rep_rel Π (z, κ1'κn')) =
               Abs_rel (λz. Rep_rel Π' (z, κ1'κn'))› (is ‹?a = ?b›)
          apply (safe intro!: AOT_sem_proj_id_prop[of v ‹?a› ‹?b›,
                simplified AOT_sem_eq AOT_sem_conj AOT_sem_forall
                AOT_sem_denotes, THEN iffD2, THEN conjunct2, THEN conjunct2]
                Π_proj1_den[OF βden] Π'_proj1_den[OF βden])
          unfolding AOT_sem_exe_denoting[simplified AOT_sem_denotes, OF Π_denotes]
                    AOT_sem_exe_denoting[simplified AOT_sem_denotes, OF Π'_denotes]
                    AOT_sem_exe_denoting[simplified AOT_sem_denotes,
```

```
                                        OF Π_proj1_den[OF βden]]
                    AOT_sem_exe_denoting[simplified AOT_sem_denotes,
                                        OF Π'_proj1_den[OF βden]]
          by (subst (0 1) Abs_rel_inverse; simp?)
             (metis (no_types, lifting) AOT_model_denotes_rel.abs_eq
                            AOT_model_lambda_denotes AOT_sem_denotes AOT_sem_eq
                            AOT_sem_unary_proj_id Π_proj1_den[OF βden])
        hence ‹Rep_rel Π (x,κ1'κn') = Rep_rel Π' (x,κ1'κn')› for x
          by (simp add: Abs_rel_inject)
             metis
      } note βdenotes = this
      {
        fix α :: 'a and β :: 'b
        assume ‹AOT_model_regular (α, β)›
        moreover {
          assume ‹AOT_model_denotes α ∧ AOT_model_regular β›
          hence ‹Rep_rel Π (α,β) = Rep_rel Π' (α,β)›
            using αdenotes by presburger
        }
        moreover {
          assume ‹¬AOT_model_denotes α ∧ AOT_model_denotes β›
          hence ‹Rep_rel Π (α,β) = Rep_rel Π' (α,β)›
            by (simp add: βdenotes)
        }
        ultimately have ‹Rep_rel Π (α,β) = Rep_rel Π' (α,β)›
          by (metis (no_types, lifting) AOT_model_regular_prod_def case_prodD)
      }
      hence ‹Rep_rel Π = Rep_rel Π'›
        using Π_denotes[unfolded AOT_model_denotes_rel.rep_eq,
                       THEN conjunct2, THEN conjunct2, THEN spec, THEN mp]
        using Π'_denotes[unfolded AOT_model_denotes_rel.rep_eq,
                        THEN conjunct2, THEN conjunct2, THEN spec, THEN mp]
        using AOT_model_irregular_eqI[of ‹Rep_rel Π› ‹Rep_rel Π'› ‹(_,_)›]
        using AOT_model_irregular_nondenoting by fastforce
      hence ‹Π = Π'›
        by (rule Rep_rel_inject[THEN iffD1])
    }
    ultimately have ‹Π = Π' = (∀ κ . AOT_model_denotes κ ⟶
        [v ⊨ «AOT_sem_proj_id κ (λ κ1κn . «[Π]κ1...κn»)
                                (λ κ1κn . «[Π']κ1...κn»)»])›
      by auto
  }
  thus ‹[v ⊨ Π = Π'] = [v ⊨ Π↓ & Π'↓ &
      ∀α («AOT_sem_proj_id α (λ κ1κn . «[Π]κ1...κn») (λ κ1κn . «[Π']κ1...κn»)»)]›
    by (auto simp: AOT_sem_eq AOT_sem_denotes AOT_sem_forall AOT_sem_conj)
next
  fix v and φ :: ‹'a×'b⇒o› and τ :: ‹'a×'b›
  assume ‹[v ⊨ τ↓]›
  moreover assume ‹[v ⊨ [λz1...zn «φ z1zn»] = [λz1...zn «φ z1zn»]]›
  ultimately show ‹[v ⊨ «AOT_sem_proj_id τ φ φ»]›
    unfolding AOT_sem_proj_id_prod_def
    using AOT_sem_proj_id_refl[of v "snd τ" "λb. φ (fst τ, b)"]
    by (auto simp: AOT_sem_eq AOT_sem_conj AOT_sem_denotes
                   AOT_model_denotes_prod_def AOT_model_lambda_denotes
                   AOT_meta_prod_equivI)
qed
end

text‹Sanity-check to verify that n-ary relation identity follows.›
lemma ‹[v ⊨ Π = Π'] = [v ⊨ Π↓ & Π'↓ & ∀x∀y([λz [Π]z y] = [λz [Π']z y] &
                                              [λz [Π]x z] = [λz [Π']x z])]›
  for Π :: ‹<κ×κ>›
  by (auto simp: AOT_sem_proj_id_prop[of v Π Π'] AOT_sem_proj_id_prod_def
```

```
                AOT_sem_conj AOT_sem_denotes AOT_sem_forall AOT_sem_unary_proj_id
                AOT_model_denotes_prod_def)
lemma ‹[v ⊨ Π = Π'] = [v ⊨ Π↓ & Π'↓ & ∀x₁∀x₂∀x₃ (
  [λz [Π]z x₂ x₃] = [λz [Π']z x₂ x₃] &
  [λz [Π]x₁ z x₃] = [λz [Π']x₁ z x₃] &
  [λz [Π]x₁ x₂ z] = [λz [Π']x₁ x₂ z])]›
  for Π :: ‹<κ×κ×κ>›
  by (auto simp: AOT_sem_proj_id_prop[of v Π Π'] AOT_sem_proj_id_prod_def
                AOT_sem_conj AOT_sem_denotes AOT_sem_forall AOT_sem_unary_proj_id
                AOT_model_denotes_prod_def)
lemma ‹[v ⊨ Π = Π'] = [v ⊨ Π↓ & Π'↓ & ∀x₁∀x₂∀x₃∀x₄ (
    [λz [Π]z x₂ x₃ x₄] = [λz [Π']z x₂ x₃ x₄] &
    [λz [Π]x₁ z x₃ x₄] = [λz [Π']x₁ z x₃ x₄] &
    [λz [Π]x₁ x₂ z x₄] = [λz [Π']x₁ x₂ z x₄] &
    [λz [Π]x₁ x₂ x₃ z] = [λz [Π']x₁ x₂ x₃ z])]›
  for Π :: ‹<κ×κ×κ×κ>›
  by (auto simp: AOT_sem_proj_id_prop[of v Π Π'] AOT_sem_proj_id_prod_def
                AOT_sem_conj AOT_sem_denotes AOT_sem_forall AOT_sem_unary_proj_id
                AOT_model_denotes_prod_def)

text‹n-ary Encoding is constructed using a similar mechanism as n-ary relation
     identity using an auxiliary notion of projection-encoding.›
class AOT_Enc =
  fixes AOT_enc :: ‹'a ⇒ <'a::AOT_IndividualTerm> ⇒ o›
    and AOT_proj_enc :: ‹'a ⇒ ('a ⇒ o) ⇒ o›
  assumes AOT_sem_enc_denotes:
    ‹[v ⊨ «AOT_enc κ₁κₙ Π»] ⟹ [v ⊨ κ₁...κₙ↓] ∧ [v ⊨ Π↓]›
  assumes AOT_sem_enc_proj_enc:
    ‹[v ⊨ «AOT_enc κ₁κₙ Π»] =
     [v ⊨ Π↓ & «AOT_proj_enc κ₁κₙ (λ κ₁κₙ.  «[Π]κ₁...κₙ»)»]›
  assumes AOT_sem_proj_enc_denotes:
    ‹[v ⊨ «AOT_proj_enc κ₁κₙ φ»] ⟹ [v ⊨ κ₁...κₙ↓]›
  assumes AOT_sem_enc_nec:
    ‹[v ⊨ «AOT_enc κ₁κₙ Π»] ⟹ [w ⊨ «AOT_enc κ₁κₙ Π»]›
  assumes AOT_sem_proj_enc_nec:
    ‹[v ⊨ «AOT_proj_enc κ₁κₙ φ»] ⟹ [w ⊨ «AOT_proj_enc κ₁κₙ φ»]›
  assumes AOT_sem_universal_encoder:
    ‹∃ κ₁κₙ. [v ⊨ κ₁...κₙ↓] ∧ (∀ Π . [v ⊨ Π↓] ⟶ [v ⊨ «AOT_enc κ₁κₙ Π»]) ∧
              (∀ φ . [v ⊨ [λz₁...zₙ φ{z₁...zₙ}]↓] ⟶ [v ⊨ «AOT_proj_enc κ₁κₙ φ»])›

AOT_syntax_print_translations
  "_AOT_enc (_AOT_individual_term κ) (_AOT_relation_term Π)" <= "CONST AOT_enc κ Π"

context AOT_meta_syntax
begin
notation AOT_enc ("⦃_,_⦄")
end
context AOT_no_meta_syntax
begin
no_notation AOT_enc ("⦃_,_⦄")
end

text‹Unary encoding additionally has to satisfy the axioms of unary encoding and
     the definition of property identity.›
class AOT_UnaryEnc = AOT_UnaryIndividualTerm +
  assumes AOT_sem_enc_eq: ‹[v ⊨ Π↓ & Π'↓ & □∀ν (ν[Π] ≡ ν[Π']) → Π = Π']›
      and AOT_sem_A_objects: ‹[v ⊨ ∃x (¬◇[E!]x & ∀F (x[F] ≡ φ{F}))]›
      and AOT_sem_unary_proj_enc: ‹AOT_proj_enc x ψ = AOT_enc x «[λz ψ{z}]»›
      and AOT_sem_nocoder: ‹[v ⊨ [E!]κ] ⟹ ¬[w ⊨ «AOT_enc κ Π»]›
      and AOT_sem_ind_eq: ‹([v ⊨ κ↓] ∧ [v ⊨ κ'↓] ∧ κ = (κ')) =
       (([v ⊨ [λx ◇[E!]x]κ] ∧
         [v ⊨ [λx ◇[E!]x]κ'] ∧
         (∀ v Π . [v ⊨ Π↓] ⟶ [v ⊨ [Π]κ] = [v ⊨ [Π]κ']))
```

```
          ∨ ([v ⊨ [λx ¬◊[E!]x]κ] ∧
             [v ⊨ [λx ¬◊[E!]x]κ'] ∧
             (∀ v Π . [v ⊨ Π↓] ⟶ [v ⊨ κ[Π]] = [v ⊨ κ'[Π]]))))›

        (* only extended models *)
        and AOT_sem_enc_indistinguishable_all:
            ‹AOT_ExtendedModel ⟹
             [v ⊨ [λx ¬◊[E!]x]κ] ⟹
             [v ⊨ [λx ¬◊[E!]x]κ'] ⟹
             (⋀ Π' . [v ⊨ Π'↓] ⟹ (⋀ w . [w ⊨ [Π']κ] = [w ⊨ [Π']κ'])) ⟹
             [v ⊨ Π↓] ⟹
             (⋀ Π' . [v ⊨ Π'↓] ⟹ (⋀ κ₀ . [v ⊨ [λx ◊[E!]x]κ₀] ⟹
                (⋀ w . [w ⊨ [Π']κ₀] = [w ⊨ [Π]κ₀])) ⟹ [v ⊨ κ[Π']]) ⟹
             (⋀ Π' . [v ⊨ Π'↓] ⟹ (⋀ κ₀ . [v ⊨ [λx ◊[E!]x]κ₀] ⟹
                (⋀ w . [w ⊨ [Π']κ₀] = [w ⊨ [Π]κ₀])) ⟹ [v ⊨ κ'[Π']])›
        and AOT_sem_enc_indistinguishable_ex:
            ‹AOT_ExtendedModel ⟹
             [v ⊨ [λx ¬◊[E!]x]κ] ⟹
             [v ⊨ [λx ¬◊[E!]x]κ'] ⟹
             (⋀ Π' . [v ⊨ Π'↓] ⟹ (⋀ w . [w ⊨ [Π']κ] = [w ⊨ [Π']κ'])) ⟹
             [v ⊨ Π↓] ⟹
             ∃ Π' . [v ⊨ Π'↓] ∧ [v ⊨ κ[Π']] ∧
                    (∀ κ₀ . [v ⊨ [λx ◊[E!]x]κ₀] ⟶
                            (∀ w . [w ⊨ [Π']κ₀] = [w ⊨ [Π]κ₀])) ⟹
             ∃ Π' . [v ⊨ Π'↓] ∧ [v ⊨ κ'[Π']] ∧
                    (∀ κ₀ . [v ⊨ [λx ◊[E!]x]κ₀] ⟶
                            (∀ w . [w ⊨ [Π']κ₀] = [w ⊨ [Π]κ₀]))›

text‹We specify encoding to align with the model-construction of encoding.›
consts AOT_sem_enc_κ :: ‹κ ⇒ <κ> ⇒ o›
specification(AOT_sem_enc_κ)
  AOT_sem_enc_κ:
  ‹[v ⊨ «AOT_sem_enc_κ κ Π»] =
   (AOT_model_denotes κ ∧ AOT_model_denotes Π ∧ AOT_model_enc κ Π)›
  by (rule exI[where x=‹λ κ Π . εₒ w . AOT_model_denotes κ ∧ AOT_model_denotes Π ∧
                                        AOT_model_enc κ Π›])
     (simp add: AOT_model_proposition_choice_simp AOT_model_enc_κ_def κ.case_eq_if)

text‹We show that @{typ κ} satisfies the generic properties of n-ary encoding.›
instantiation κ :: AOT_Enc
begin
definition AOT_enc_κ :: ‹κ ⇒ <κ> ⇒ o› where
  ‹AOT_enc_κ ≡ AOT_sem_enc_κ›
definition AOT_proj_enc_κ :: ‹κ ⇒ (κ ⇒ o) ⇒ o› where
  ‹AOT_proj_enc_κ ≡ λ κ φ . AOT_enc κ «[λz «φ z»]»›
lemma AOT_enc_κ_meta:
  ‹[v ⊨ κ[Π]] = (AOT_model_denotes κ ∧ AOT_model_denotes Π ∧ AOT_model_enc κ Π)›
  for κ::κ
  using AOT_sem_enc_κ unfolding AOT_enc_κ_def by auto
instance proof
  fix v and κ :: κ and Π
  show ‹[v ⊨ «AOT_enc κ Π»] ⟹ [v ⊨ κ↓] ∧ [v ⊨ Π↓]›
    unfolding AOT_sem_denotes
    using AOT_enc_κ_meta by blast
next
  fix v and κ :: κ and Π
  show ‹[v ⊨ κ[Π]] = [v ⊨ Π↓ & «AOT_proj_enc κ (λ κ'.  «[Π]κ'»)»]›
  proof
    assume enc: ‹[v ⊨ κ[Π]]›
    hence Π_denotes: ‹AOT_model_denotes Π›
      by (simp add: AOT_enc_κ_meta)
    hence Π_eta_denotes: ‹AOT_model_denotes «[λz [Π]z]»›
      using AOT_sem_denotes AOT_sem_eq AOT_sem_lambda_eta by metis
```

```
    show ‹[v ⊨ Π↓ & «AOT_proj_enc κ (λ κ.  «[Π]κ»)»]›
      using AOT_sem_lambda_eta[simplified AOT_sem_denotes AOT_sem_eq, OF Π_denotes]
      using Π_eta_denotes Π_denotes
      by (simp add: AOT_sem_conj AOT_sem_denotes AOT_proj_enc_κ_def enc)
  next
    assume ‹[v ⊨ Π↓ & «AOT_proj_enc κ (λ κ.  «[Π]κ»)»]›
    hence Π_denotes: "AOT_model_denotes Π" and eta_enc: "[v ⊨ κ[λz [Π]z]]"
      by (auto simp: AOT_sem_conj AOT_sem_denotes AOT_proj_enc_κ_def)
    thus ‹[v ⊨ κ[Π]]›
      using AOT_sem_lambda_eta[simplified AOT_sem_denotes AOT_sem_eq, OF Π_denotes]
      by auto
  qed
next
  show ‹[v ⊨ «AOT_proj_enc κ φ»] ⟹ [v ⊨ κ↓]› for v and κ :: κ and φ
    by (simp add: AOT_enc_κ_meta AOT_sem_denotes AOT_proj_enc_κ_def)
next
  fix v w and κ :: κ and Π
  assume ‹[v ⊨ κ[Π]]›
  thus ‹[w ⊨ κ[Π]]›
    by (simp add: AOT_enc_κ_meta)
next
  fix v w and κ :: κ and φ
  assume ‹[v ⊨ «AOT_proj_enc κ φ»]›
  thus ‹[w ⊨ «AOT_proj_enc κ φ»]›
    by (simp add: AOT_enc_κ_meta AOT_proj_enc_κ_def)
next
  show ‹∃κ::κ. [v ⊨ κ↓] ∧ (∀ Π . [v ⊨ Π↓] ⟶  [v ⊨ κ[Π]]) ∧
              (∀ φ . [v ⊨ [λz φ{z}]↓] ⟶  [v ⊨ «AOT_proj_enc κ φ»])› for v
    by (rule exI[where x=‹ακ UNIV›])
       (simp add: AOT_sem_denotes AOT_enc_κ_meta AOT_model_enc_κ_def
                  AOT_model_denotes_κ_def  AOT_proj_enc_κ_def)
qed
end

text‹We show that @{typ κ} satisfies the properties of unary encoding.›
instantiation κ :: AOT_UnaryEnc
begin
instance proof
  fix v and Π Π' :: ‹<κ>›
  show ‹[v ⊨ Π↓ & Π'↓ & □∀ν (ν[Π] ≡ ν[Π']) → Π = Π']›
    apply (simp add: AOT_sem_forall AOT_sem_eq AOT_sem_imp AOT_sem_equiv
                     AOT_enc_κ_meta AOT_sem_conj AOT_sem_denotes AOT_sem_box)
    using AOT_meta_A_objects_κ by fastforce
next
  fix v and φ:: ‹<κ> ⇒ o›
  show ‹[v ⊨ ∃x (¬◊[E!]x & ∀F (x[F] ≡ φ{F}))]›
    using AOT_model_A_objects[of "λ Π . [v ⊨ φ{Π}]"]
    by (auto simp: AOT_sem_denotes AOT_sem_exists AOT_sem_conj AOT_sem_not
                   AOT_sem_dia AOT_sem_concrete AOT_enc_κ_meta AOT_sem_equiv
                   AOT_sem_forall)
next
  show ‹AOT_proj_enc x ψ = AOT_enc x (AOT_lambda ψ)› for x :: κ and ψ
    by (simp add: AOT_proj_enc_κ_def)
next
  show ‹[v ⊨ [E!]κ] ⟹ ¬ [w ⊨ κ[Π]]› for v w and κ :: κ and Π
    by (simp add: AOT_enc_κ_meta AOT_sem_concrete AOT_model_nocoder)
next
  fix v and κ κ' :: κ
  show ‹([v ⊨ κ↓] ∧ [v ⊨ κ'↓] ∧ κ = κ') =
        (([v ⊨ [λx ◊[E!]x]κ] ∧
          [v ⊨ [λx ◊[E!]x]κ'] ∧
          (∀ v Π . [v ⊨ Π↓] ⟶ [v ⊨ [Π]κ] = [v ⊨ [Π]κ']))
        ∨ ([v ⊨ [λx ¬◊[E!]x]κ] ∧
```

```
             [v ⊨ [λx ¬◇[E!]x]κ'] ∧
             (∀ v Π . [v ⊨ Π↓] ⟶ [v ⊨ κ[Π]] = [v ⊨ κ'[Π]])))›
    (is ‹?lhs = (?ordeq ∨ ?abseq)›)
  proof -
  {
    assume 0: ‹[v ⊨ κ↓] ∧ [v ⊨ κ'↓] ∧ κ = κ'›
    {
      assume ‹is_ωκ κ'›
      hence ‹[v ⊨ [λx ◇[E!]x]κ']›
        apply (subst AOT_sem_lambda_beta[OF AOT_sem_ordinary_def_denotes, of v κ'])
         apply (simp add: "0")
        apply (simp add: AOT_sem_dia)
        using AOT_sem_concrete AOT_model_ω_concrete_in_some_world is_ωκ_def by force
      hence ‹?ordeq› unfolding 0[THEN conjunct2, THEN conjunct2] by auto
    }
    moreover {
      assume ‹is_ακ κ'›
      hence ‹[v ⊨ [λx ¬◇[E!]x]κ']›
        apply (subst AOT_sem_lambda_beta[OF AOT_sem_abstract_def_denotes, of v κ'])
         apply (simp add: "0")
        apply (simp add: AOT_sem_not AOT_sem_dia)
        using AOT_sem_concrete is_ακ_def by force
      hence ‹?abseq› unfolding 0[THEN conjunct2, THEN conjunct2] by auto
    }
    ultimately have ‹?ordeq ∨ ?abseq›
      by (meson "0" AOT_sem_denotes AOT_model_denotes_κ_def κ.exhaust_disc)
  }
  moreover {
    assume ordeq: ‹?ordeq›
    hence κ_denotes: ‹[v ⊨ κ↓]› and κ'_denotes: ‹[v ⊨ κ'↓]›
      by (simp add: AOT_sem_denotes AOT_sem_exe)+
    hence ‹is_ωκ κ› and ‹is_ωκ κ'›
      by (metis AOT_model_concrete_κ.simps(2) AOT_model_denotes_κ_def
                AOT_sem_concrete AOT_sem_denotes AOT_sem_dia AOT_sem_lambda_beta
                AOT_sem_ordinary_def_denotes κ.collapse(2) κ.exhaust_disc ordeq)+
    have denotes: ‹[v ⊨ [λz «εₒ w . κυ z = κυ κ»]↓]›
      unfolding AOT_sem_denotes AOT_model_lambda_denotes
      by (simp add: AOT_model_term_equiv_κ_def)
    hence "[v ⊨ [λz «εₒ w . κυ z = κυ κ»]κ] = [v ⊨ [λz «εₒ w . κυ z = κυ κ»]κ']"
      using ordeq by (simp add: AOT_sem_denotes)
    hence ‹[v ⊨ «κ»↓] ∧ [v ⊨ «κ'»↓] ∧ κ = κ'›
      unfolding AOT_sem_lambda_beta[OF denotes, OF κ_denotes]
                AOT_sem_lambda_beta[OF denotes, OF κ'_denotes]
      using κ'_denotes ‹is_ωκ κ'› ‹is_ωκ κ› is_ωκ_def
            AOT_model_proposition_choice_simp by force
  }
  moreover {
    assume 0: ‹?abseq›
    hence κ_denotes: ‹[v ⊨ κ↓]› and κ'_denotes: ‹[v ⊨ κ'↓]›
      by (simp add: AOT_sem_denotes AOT_sem_exe)+
    hence ‹¬is_ωκ κ› and ‹¬is_ωκ κ'›
      by (metis AOT_model_ω_concrete_in_some_world AOT_model_concrete_κ.simps(1)
                AOT_sem_concrete AOT_sem_dia AOT_sem_exe AOT_sem_lambda_beta
                AOT_sem_not κ.collapse(1) 0)+
    hence ‹is_ακ κ› and ‹is_ακ κ'›
      by (meson AOT_sem_denotes AOT_model_denotes_κ_def κ.exhaust_disc
                κ_denotes κ'_denotes)+
    then obtain x y where κ_def: ‹κ = ακ x› and κ'_def: ‹κ' = ακ y›
      using is_ακ_def by auto
    {
      fix r
      assume ‹r ∈ x›
      hence ‹[v ⊨ κ[«urrel_to_rel r»]]›
```

```
        unfolding κ_def
        unfolding AOT_enc_κ_meta
        unfolding AOT_model_enc_κ_def
        apply (simp add: AOT_model_denotes_κ_def)
        by (metis (mono_tags) AOT_rel_equiv_def Quotient_def urrel_quotient)
      hence ‹[v ⊨ κ'[«urrel_to_rel r»]]›
        using AOT_enc_κ_meta 0 by (metis AOT_sem_enc_denotes)
      hence ‹r ∈ y›
        unfolding κ'_def
        unfolding AOT_enc_κ_meta
        unfolding AOT_model_enc_κ_def
        apply (simp add: AOT_model_denotes_κ_def)
        using Quotient_abs_rep urrel_quotient by fastforce
    }
    moreover {
      fix r
      assume ‹r ∈ y›
      hence ‹[v ⊨ κ'[«urrel_to_rel r»]]›
        unfolding κ'_def
        unfolding AOT_enc_κ_meta
        unfolding AOT_model_enc_κ_def
        apply (simp add: AOT_model_denotes_κ_def)
        by (metis (mono_tags) AOT_rel_equiv_def Quotient_def urrel_quotient)
      hence ‹[v ⊨ κ[«urrel_to_rel r»]]›
        using AOT_enc_κ_meta 0 by (metis AOT_sem_enc_denotes)
      hence ‹r ∈ x›
        unfolding κ_def
        unfolding AOT_enc_κ_meta
        unfolding AOT_model_enc_κ_def
        apply (simp add: AOT_model_denotes_κ_def)
        using Quotient_abs_rep urrel_quotient by fastforce
    }
    ultimately have "x = y" by blast
    hence ‹[v ⊨ κ↓] ∧ [v ⊨ κ'↓] ∧ κ = κ'›
      using κ'_def κ'_denotes κ_def by blast
  }
  ultimately show ?thesis
    unfolding AOT_sem_denotes
    by auto
  qed
(* Only extended model *)
next
  fix v and κ κ' :: κ and Π Π' :: ‹<κ>›
  assume ext: ‹AOT_ExtendedModel›
  assume ‹[v ⊨ [λx ¬◊[E!]x]κ]›
  hence ‹is_ακ κ›
    by (metis AOT_model_ω_concrete_in_some_world AOT_model_concrete_κ.simps(1)
              AOT_model_denotes_κ_def AOT_sem_concrete AOT_sem_denotes AOT_sem_dia
              AOT_sem_exe AOT_sem_lambda_beta AOT_sem_not κ.collapse(1) κ.exhaust_disc)
  hence κ_abs: ‹¬(∃ w . AOT_model_concrete w κ)›
    using is_ακ_def by fastforce
  have κ_den: ‹AOT_model_denotes κ›
    by (simp add: AOT_model_denotes_κ_def κ.distinct_disc(5) ‹is_ακ κ›)
  assume ‹[v ⊨ [λx ¬◊[E!]x]κ']›
  hence ‹is_ακ κ'›
    by (metis AOT_model_ω_concrete_in_some_world AOT_model_concrete_κ.simps(1)
              AOT_model_denotes_κ_def AOT_sem_concrete AOT_sem_denotes AOT_sem_dia
              AOT_sem_exe AOT_sem_lambda_beta AOT_sem_not κ.collapse(1)
              κ.exhaust_disc)
  hence κ'_abs: ‹¬(∃ w . AOT_model_concrete w κ')›
    using is_ακ_def by fastforce
  have κ'_den: ‹AOT_model_denotes κ'›
    by (meson AOT_model_denotes_κ_def κ.distinct_disc(6) ‹is_ακ κ'›)
```

```
  assume ‹[v ⊨ Π'↓] ⟹ [w ⊨ [Π']κ] = [w ⊨ [Π']κ']› for Π' w
  hence indist: ‹[v ⊨ «Rep_rel Π' κ»] = [v ⊨ «Rep_rel Π' κ'»]›
    if ‹AOT_model_denotes Π'› for Π' v
    by (metis AOT_sem_denotes AOT_sem_exe κ'_den κ_den that)
  assume κ_enc_cond: ‹[v ⊨ Π'↓] ⟹
    (⋀ κ₀ w. [v ⊨ [λx ◇[E!]x]κ₀] ⟹
             [w ⊨ [Π']κ₀] = [w ⊨ [Π]κ₀]) ⟹
    [v ⊨ κ[Π']]› for Π'
  assume Π_den': ‹[v ⊨ Π↓]›
  hence Π_den: ‹AOT_model_denotes Π›
    using AOT_sem_denotes by blast
  {
    fix Π' :: ‹<κ>›
    assume Π'_den: ‹AOT_model_denotes Π'›
    hence Π'_den': ‹[v ⊨ Π'↓]›
      by (simp add: AOT_sem_denotes)
    assume 1: ‹∃w. AOT_model_concrete w x ⟹
               [v ⊨ «Rep_rel Π' x»] = [v ⊨ «Rep_rel Π x»]› for v x
    {
      fix κ₀ :: κ and w
      assume ‹[v ⊨ [λx ◇[E!]x]κ₀]›
      hence ‹is_ωκ κ₀›
        by (smt (z3) AOT_model_concrete_κ.simps(2) AOT_model_denotes_κ_def
                     AOT_sem_concrete AOT_sem_denotes AOT_sem_dia AOT_sem_exe
                     AOT_sem_lambda_beta κ.exhaust_disc is_ακ_def)
      then obtain x where x_prop: ‹κ₀ = ωκ x›
        using is_ωκ_def by blast
      have ‹∃w . AOT_model_concrete w (ωκ x)›
        by (simp add: AOT_model_ω_concrete_in_some_world)
      hence ‹[v ⊨ «Rep_rel Π' (ωκ x)»] = [v ⊨ «Rep_rel Π (ωκ x)»]› for v
        using 1 by blast
      hence ‹[w ⊨ [Π']κ₀] = [w ⊨ [Π]κ₀]› unfolding x_prop
        by (simp add: AOT_sem_exe AOT_sem_denotes AOT_model_denotes_κ_def
                      Π'_den Π_den)
    } note 2 = this
    have ‹[v ⊨ κ[Π']]›
      using κ_enc_cond[OF Π'_den', OF 2]
      by metis
    hence ‹AOT_model_enc κ Π'›
      using AOT_enc_κ_meta by blast
  } note κ_enc_cond = this
  hence ‹AOT_model_denotes Π' ⟹
       (⋀v x. ∃w. AOT_model_concrete w x ⟹
              [v ⊨ «Rep_rel Π' x»] = [v ⊨ «Rep_rel Π x»]) ⟹
       AOT_model_enc κ Π'› for Π'
    by blast
  assume Π'_den': ‹[v ⊨ Π'↓]›
  hence Π'_den: ‹AOT_model_denotes Π'›
    using AOT_sem_denotes by blast
  assume ord_indist: ‹[v ⊨ [λx ◇[E!]x]κ₀] ⟹
                     [w ⊨ [Π']κ₀] = [w ⊨ [Π]κ₀]› for κ₀ w
  {
    fix w and κ₀ :: κ
    assume 0: ‹∃w. AOT_model_concrete w κ₀›
    hence ‹[v ⊨ [λx ◇[E!]x]κ₀]›
      using AOT_model_concrete_denotes AOT_sem_concrete AOT_sem_denotes AOT_sem_dia
            AOT_sem_lambda_beta AOT_sem_ordinary_def_denotes by blast
    hence ‹[w ⊨ [Π']κ₀] = [w ⊨ [Π]κ₀]›
      using ord_indist by metis
    hence ‹[w ⊨ «Rep_rel Π' κ₀»] = [w ⊨ «Rep_rel Π κ₀»]›
      by (metis AOT_model_concrete_denotes AOT_sem_denotes AOT_sem_exe Π'_den Π_den 0)
  } note ord_indist = this
  have ‹AOT_model_enc κ' Π'›
```

```
    using AOT_model_enc_indistinguishable_all
            [OF ext, OF κ_den, OF κ_abs, OF κ'_den, OF κ'_abs, OF Π_den]
          indist κ_enc_cond Π'_den ord_indist by blast
  thus ‹[v ⊨ κ'[Π']]›
    using AOT_enc_κ_meta Π'_den κ'_den by blast
next
  fix v and κ κ' :: κ and Π Π' :: ‹<κ>›
  assume ext: ‹AOT_ExtendedModel›
  assume ‹[v ⊨ [λx ¬◊[E!]x]κ]›
  hence ‹is_ακ κ›
    by (metis AOT_model_ω_concrete_in_some_world AOT_model_concrete_κ.simps(1)
              AOT_model_denotes_κ_def AOT_sem_concrete AOT_sem_denotes AOT_sem_dia
              AOT_sem_exe AOT_sem_lambda_beta AOT_sem_not κ.collapse(1)
              κ.exhaust_disc)
  hence κ_abs: ‹¬(∃ w . AOT_model_concrete w κ)›
    using is_ακ_def by fastforce
  have κ_den: ‹AOT_model_denotes κ›
    by (simp add: AOT_model_denotes_κ_def κ.distinct_disc(5) ‹is_ακ κ›)
  assume ‹[v ⊨ [λx ¬◊[E!]x]κ']›
  hence ‹is_ακ κ'›
    by (metis AOT_model_ω_concrete_in_some_world AOT_model_concrete_κ.simps(1)
              AOT_model_denotes_κ_def AOT_sem_concrete AOT_sem_denotes AOT_sem_dia
              AOT_sem_exe AOT_sem_lambda_beta AOT_sem_not κ.collapse(1)
              κ.exhaust_disc)
  hence κ'_abs: ‹¬(∃ w . AOT_model_concrete w κ')›
    using is_ακ_def by fastforce
  have κ'_den: ‹AOT_model_denotes κ'›
    by (meson AOT_model_denotes_κ_def κ.distinct_disc(6) ‹is_ακ κ'›)
  assume ‹[v ⊨ Π'↓] ⟹ [w ⊨ [Π']κ] = [w ⊨ [Π']κ']› for Π' w
  hence indist: ‹[v ⊨ «Rep_rel Π' κ»] = [v ⊨ «Rep_rel Π' κ'»]›
    if ‹AOT_model_denotes Π'› for Π' v
    by (metis AOT_sem_denotes AOT_sem_exe κ'_den κ_den that)
  assume Π_den': ‹[v ⊨ Π↓]›
  hence Π_den: ‹AOT_model_denotes Π›
    using AOT_sem_denotes by blast
  assume ‹∃Π'. [v ⊨ Π'↓] ∧ [v ⊨ κ[Π']] ∧
              (∀κ₀. [v ⊨ [λx ◊[E!]x]κ₀] ⟶
                    (∀w. [w ⊨ [Π']κ₀] = [w ⊨ [Π]κ₀]))›
  then obtain Π' where
      Π'_den: ‹[v ⊨ Π'↓]› and
      Π'_enc: ‹[v ⊨ κ[Π']]› and
      Π'_prop: ‹∀κ₀. [v ⊨ [λx ◊[E!]x]κ₀] ⟶
                    (∀w. [w ⊨ [Π']κ₀] = [w ⊨ [Π]κ₀])›
    by blast
  have ‹AOT_model_denotes Π'›
    using AOT_enc_κ_meta Π'_enc by force
  moreover have ‹AOT_model_enc κ Π'›
    using AOT_enc_κ_meta Π'_enc by blast
  moreover have ‹(∃w. AOT_model_concrete w κ₀) ⟹
                 [v ⊨ «Rep_rel Π' κ₀»] = [v ⊨ «Rep_rel Π κ₀»]› for κ₀ v
  proof -
    assume 0: ‹∃w. AOT_model_concrete w κ₀›
    hence ‹[v ⊨ [λx ◊[E!]x]κ₀]› for v
      using AOT_model_concrete_denotes AOT_sem_concrete AOT_sem_denotes AOT_sem_dia
            AOT_sem_lambda_beta AOT_sem_ordinary_def_denotes by blast
    hence ‹∀w. [w ⊨ [Π']κ₀] = [w ⊨ [Π]κ₀]› using Π'_prop by blast
    thus ‹[v ⊨ «Rep_rel Π' κ₀»] = [v ⊨ «Rep_rel Π κ₀»]›
      by (meson "0" AOT_model_concrete_denotes AOT_sem_denotes AOT_sem_exe Π_den
                calculation(1))
  qed
  ultimately have ‹∃Π'. AOT_model_denotes Π' ∧ AOT_model_enc κ Π' ∧
                          (∀v x. (∃w. AOT_model_concrete w x) ⟶
                           [v ⊨ «Rep_rel Π' x»] = [v ⊨ «Rep_rel Π x»])›
```

```
    by blast
  hence ‹∃Π'. AOT_model_denotes Π' ∧ AOT_model_enc κ' Π' ∧
             (∀v x. (∃w. AOT_model_concrete w x) ⟶
              [v ⊨ «Rep_rel Π' x»] = [v ⊨ «Rep_rel Π x»])›
    using AOT_model_enc_indistinguishable_ex
          [OF ext, OF κ_den, OF κ_abs, OF κ'_den, OF κ'_abs, OF Π_den]
        indist by blast
  then obtain Π" where
      Π"_den: ‹AOT_model_denotes Π"›
      and Π"_enc: ‹AOT_model_enc κ' Π"›
      and Π"_prop: ‹(∃w. AOT_model_concrete w x) ⟹
                    [v ⊨ «Rep_rel Π" x»] = [v ⊨ «Rep_rel Π x»]› for v x
    by blast
  have ‹[v ⊨ Π"↓]›
    by (simp add: AOT_sem_denotes Π"_den)
  moreover have ‹[v ⊨ κ'[Π"]]›
    by (simp add: AOT_enc_κ_meta Π"_den Π"_enc κ'_den)
  moreover have ‹[v ⊨ [λx ◊[E!]x]κ₀] ⟹
                 (∀w. [w ⊨ [Π"]κ₀] = [w ⊨ [Π]κ₀])› for κ₀
  proof -
    assume ‹[v ⊨ [λx ◊[E!]x]κ₀]›
    hence ‹∃w. AOT_model_concrete w κ₀›
      by (metis AOT_sem_concrete AOT_sem_dia AOT_sem_exe AOT_sem_lambda_beta)
    thus ‹∀w. [w ⊨ [Π"]κ₀] = [w ⊨ [Π]κ₀]›
      using Π"_prop
      by (metis AOT_sem_denotes AOT_sem_exe Π"_den Π_den)
  qed
  ultimately show ‹∃Π'. [v ⊨ Π'↓] ∧ [v ⊨ κ'[Π']] ∧
                        (∀κ₀. [v ⊨ [λx ◊[E!]x]κ₀] ⟶
                              (∀w. [w ⊨ [Π']κ₀] = [w ⊨ [Π]κ₀]))›
    by (safe intro!: exI[where x=Π"]) blast+
qed
end

text‹Define encoding for products using projection-encoding.›
instantiation prod :: (AOT_UnaryEnc, AOT_Enc) AOT_Enc
begin
definition AOT_proj_enc_prod :: ‹'a×'b ⇒ ('a×'b ⇒ o) ⇒ o› where
  ‹AOT_proj_enc_prod ≡ λ (κ,κ') φ . «κ[λν «φ (ν,κ')»] &
                                    «AOT_proj_enc κ' (λν. φ (κ,ν))»»›
definition AOT_enc_prod :: ‹'a×'b ⇒ ‹'a×'b› ⇒ o› where
  ‹AOT_enc_prod ≡ λ κ Π . «Π↓ & «AOT_proj_enc κ (λ κ₁'κₙ'.  «[Π]κ₁'...κₙ'»)»»›
instance proof
  show ‹[v ⊨ κ₁...κₙ[Π]] ⟹ [v ⊨ κ₁...κₙ↓] ∧ [v ⊨ Π↓]›
    for v and κ₁κₙ :: ‹'a×'b› and Π
    unfolding AOT_enc_prod_def
    apply (induct κ₁κₙ; simp add: AOT_sem_conj AOT_sem_denotes AOT_proj_enc_prod_def)
    by (metis AOT_sem_denotes AOT_model_denotes_prod_def AOT_sem_enc_denotes
              AOT_sem_proj_enc_denotes case_prodI)
next
  show ‹[v ⊨ κ₁...κₙ[Π]] =
        [v ⊨ «Π»↓ & «AOT_proj_enc κ₁κₙ (λ κ₁κₙ.  «[Π]κ₁...κₙ»)»]›
    for v and κ₁κₙ :: ‹'a×'b› and Π
    unfolding AOT_enc_prod_def ..
next
  show ‹[v ⊨ «AOT_proj_enc κs φ»] ⟹ [v ⊨ «κs»↓]›
    for v and κs :: ‹'a×'b› and φ
    by (metis (mono_tags, lifting)
        AOT_sem_conj AOT_sem_denotes AOT_model_denotes_prod_def
        AOT_sem_enc_denotes AOT_sem_proj_enc_denotes
        AOT_proj_enc_prod_def case_prod_unfold)
next
  fix v w Π and κ₁κₙ :: ‹'a×'b›
```

```
  show ‹[w ⊨ κ1...κn[Π]]› if ‹[v ⊨ κ1...κn[Π]]› for v w Π and κ1κn :: ‹'a×'b›
    by (metis (mono_tags, lifting)
          AOT_enc_prod_def AOT_sem_enc_proj_enc AOT_sem_conj AOT_sem_denotes
          AOT_sem_proj_enc_nec AOT_proj_enc_prod_def case_prod_unfold that)
next
  show ‹[w ⊨ «AOT_proj_enc κ1κn φ»]› if ‹[v ⊨ «AOT_proj_enc κ1κn φ»]›
    for v w φ and κ1κn :: ‹'a×'b›
    by (metis (mono_tags, lifting)
          that AOT_sem_enc_proj_enc AOT_sem_conj AOT_sem_denotes
          AOT_sem_proj_enc_nec AOT_proj_enc_prod_def case_prod_unfold)
next
  fix v
  obtain κ :: 'a where a_prop: ‹[v ⊨ κ↓] ∧ (∀ Π . [v ⊨ Π↓] ⟶  [v ⊨ κ[Π]])›
    using AOT_sem_universal_encoder by blast
  obtain κ1'κn' :: 'b where b_prop:
    ‹[v ⊨ κ1'...κn'↓] ∧ (∀ φ . [v ⊨ [λν1...νn «φ ν1νn»]↓] ⟶
                              [v ⊨ «AOT_proj_enc κ1'κn' φ»])›
    using AOT_sem_universal_encoder by blast
  have ‹AOT_model_denotes «[λν1...νn [«Π»]ν1...νn κ1'...κn']»›
    if ‹AOT_model_denotes Π› for Π :: ‹<'a×'b>›
    unfolding AOT_model_lambda_denotes
    by (metis AOT_meta_prod_equivI(2) AOT_sem_exe_equiv)
  moreover have ‹AOT_model_denotes  «[λν1...νn [«Π»]κ ν1...νn]»›
    if ‹AOT_model_denotes Π› for Π :: ‹<'a×'b>›
    unfolding AOT_model_lambda_denotes
    by (metis AOT_meta_prod_equivI(1) AOT_sem_exe_equiv)
  ultimately have 1: ‹[v ⊨ «(κ,κ1'κn')»↓]›
              and 2: ‹(∀ Π . [v ⊨ Π↓] ⟶  [v ⊨ κ κ1'...κn'[Π]])›
    using a_prop b_prop
    by (auto simp: AOT_sem_denotes AOT_enc_κ_meta AOT_model_enc_κ_def
                   AOT_model_denotes_κ_def AOT_model_denotes_prod_def
                   AOT_enc_prod_def AOT_proj_enc_prod_def AOT_sem_conj)
  have ‹AOT_model_denotes «[λz1...zn «φ (z1zn, κ1'κn')»]»›
    if ‹AOT_model_denotes «[λz1...zm φ{z1...zm}]»› for φ :: ‹'a×'b ⇒ o›
    using that
    unfolding AOT_model_lambda_denotes
    by (metis (no_types, lifting) AOT_sem_denotes AOT_model_denotes_prod_def
                                  AOT_meta_prod_equivI(2) b_prop case_prodI)
  moreover have ‹AOT_model_denotes «[λz1...zn «φ (κ, z1zn)»]»›
    if ‹AOT_model_denotes «[λz1...zm φ{z1...zm}]»› for φ :: ‹'a×'b ⇒ o›
    using that
    unfolding AOT_model_lambda_denotes
    by (metis (no_types, lifting) AOT_sem_denotes AOT_model_denotes_prod_def
                                  AOT_meta_prod_equivI(1) a_prop case_prodI)
  ultimately have 3:
    ‹[v ⊨ «(κ,κ1'κn')»↓] ∧ (∀ φ . [v ⊨ [λz1...zn φ{z1...zn}]↓] ⟶
                                  [v ⊨ «AOT_proj_enc (κ,κ1'κn') φ»])›
    using a_prop b_prop
    by (auto simp: AOT_sem_denotes AOT_enc_κ_meta AOT_model_enc_κ_def
                   AOT_model_denotes_κ_def AOT_enc_prod_def AOT_proj_enc_prod_def
                   AOT_sem_conj AOT_model_denotes_prod_def)
  show ‹∃κ1κn::'a×'b. [v ⊨ κ1...κn↓] ∧ (∀ Π . [v ⊨ Π↓] ⟶ [v ⊨ κ1...κn[Π]]) ∧
                      (∀ φ . [v ⊨ [λz1...zn «φ z1zn»]↓] ⟶
                             [v ⊨ «AOT_proj_enc κ1κn φ»])›
    apply (rule exI[where x=‹(κ,κ1'κn')›]) using 1 2 3 by blast
qed
end

text‹Sanity-check to verify that n-ary encoding follows.›
lemma ‹[v ⊨ κ1κ2[Π]] = [v ⊨ Π↓ & κ1[λν [Π]νκ2] & κ2[λν [Π]κ1ν]]›
  for κ1 :: "'a::AOT_UnaryEnc" and κ2 :: "'b::AOT_UnaryEnc"
  by (simp add: AOT_sem_conj AOT_enc_prod_def AOT_proj_enc_prod_def
                AOT_sem_unary_proj_enc)
```

```
lemma ‹[v ⊨ κ₁κ₂κ₃[Π]] =
       [v ⊨ Π↓ & κ₁[λν [Π]νκ₂κ₃] & κ₂[λν [Π]κ₁νκ₃] & κ₃[λν [Π]κ₁κ₂ν]]›
  for κ₁ κ₂ κ₃ :: "'a::AOT_UnaryEnc"
  by (simp add: AOT_sem_conj AOT_enc_prod_def AOT_proj_enc_prod_def
                AOT_sem_unary_proj_enc)

lemma AOT_sem_vars_denote: ‹[v ⊨ α₁...αₙ↓]›
  by induct simp

text‹Combine the introduced type classes and register them as
     type constraints for individual terms.›
class AOT_κs = AOT_IndividualTerm + AOT_RelationProjection + AOT_Enc
class AOT_κ = AOT_κs + AOT_UnaryIndividualTerm +
  AOT_UnaryRelationProjection + AOT_UnaryEnc

instance κ :: AOT_κ by standard
instance prod :: (AOT_κ, AOT_κs) AOT_κs by standard

AOT_register_type_constraints
  Individual: ‹_::AOT_κ› ‹_::AOT_κs› and
  Relation: ‹‹_::AOT_κs››

text‹We define semantic predicates to capture the conditions of cqt.2 (i.e.
     the base cases of denoting terms) on matrices of @{text λ}-expressions.›
definition AOT_instance_of_cqt_2 :: ‹('a::AOT_κs ⇒ o) ⇒ bool› where
  ‹AOT_instance_of_cqt_2 ≡ λ φ . ∀ x y . AOT_model_denotes x ∧ AOT_model_denotes y ∧
                                         AOT_model_term_equiv x y ⟶ φ x = φ y›
definition AOT_instance_of_cqt_2_exe_arg :: ‹('a::AOT_κs ⇒ 'b::AOT_κs) ⇒ bool› where
  ‹AOT_instance_of_cqt_2_exe_arg ≡ λ φ . ∀ x y .
      AOT_model_denotes x ∧ AOT_model_denotes y ∧ AOT_model_term_equiv x y ⟶
      AOT_model_term_equiv (φ x) (φ y)›

text‹@{text λ}-expressions with a matrix that satisfies our predicate denote.›
lemma AOT_sem_cqt_2:
  assumes ‹AOT_instance_of_cqt_2 φ›
  shows ‹[v ⊨ [λν₁...νₙ φ{ν₁...νₙ}]↓]›
  using assms
  by (metis AOT_instance_of_cqt_2_def AOT_model_lambda_denotes AOT_sem_denotes)

syntax AOT_instance_of_cqt_2 :: ‹id_position ⇒ AOT_prop›
  ("INSTANCE'_OF'_CQT'_2'(_')")

text‹Prove introduction rules for the predicates that match the natural language
     restrictions of the axiom.›
named_theorems AOT_instance_of_cqt_2_intro
lemma AOT_instance_of_cqt_2_intros_const[AOT_instance_of_cqt_2_intro]:
  ‹AOT_instance_of_cqt_2 (λα. φ)›
  by (simp add: AOT_instance_of_cqt_2_def AOT_sem_denotes AOT_model_lambda_denotes)
lemma AOT_instance_of_cqt_2_intros_not[AOT_instance_of_cqt_2_intro]:
  assumes ‹AOT_instance_of_cqt_2 φ›
  shows ‹AOT_instance_of_cqt_2 (λτ. «¬φ{τ}»)›
  using assms
  by (metis (no_types, lifting) AOT_instance_of_cqt_2_def)
lemma AOT_instance_of_cqt_2_intros_imp[AOT_instance_of_cqt_2_intro]:
  assumes ‹AOT_instance_of_cqt_2 φ› and ‹AOT_instance_of_cqt_2 ψ›
  shows ‹AOT_instance_of_cqt_2 (λτ. «φ{τ} → ψ{τ}»)›
  using assms
  by (auto simp: AOT_instance_of_cqt_2_def AOT_sem_denotes
                 AOT_model_lambda_denotes AOT_sem_imp)
lemma AOT_instance_of_cqt_2_intros_box[AOT_instance_of_cqt_2_intro]:
  assumes ‹AOT_instance_of_cqt_2 φ›
  shows ‹AOT_instance_of_cqt_2 (λτ. «□φ{τ}»)›
  using assms
```

```
  by (auto simp: AOT_instance_of_cqt_2_def AOT_sem_denotes
                 AOT_model_lambda_denotes AOT_sem_box)
lemma AOT_instance_of_cqt_2_intros_act[AOT_instance_of_cqt_2_intro]:
  assumes ‹AOT_instance_of_cqt_2 φ›
  shows ‹AOT_instance_of_cqt_2 (λτ. «𝒜φ{τ}»)›
  using assms
  by (auto simp: AOT_instance_of_cqt_2_def AOT_sem_denotes
                 AOT_model_lambda_denotes AOT_sem_act)
lemma AOT_instance_of_cqt_2_intros_diamond[AOT_instance_of_cqt_2_intro]:
  assumes ‹AOT_instance_of_cqt_2 φ›
  shows ‹AOT_instance_of_cqt_2 (λτ. «◊φ{τ}»)›
  using assms
  by (auto simp: AOT_instance_of_cqt_2_def AOT_sem_denotes
                 AOT_model_lambda_denotes AOT_sem_dia)
lemma AOT_instance_of_cqt_2_intros_conj[AOT_instance_of_cqt_2_intro]:
  assumes ‹AOT_instance_of_cqt_2 φ› and ‹AOT_instance_of_cqt_2 ψ›
  shows ‹AOT_instance_of_cqt_2 (λτ. «φ{τ} & ψ{τ}»)›
  using assms
  by (auto simp: AOT_instance_of_cqt_2_def AOT_sem_denotes
                 AOT_model_lambda_denotes AOT_sem_conj)
lemma AOT_instance_of_cqt_2_intros_disj[AOT_instance_of_cqt_2_intro]:
  assumes ‹AOT_instance_of_cqt_2 φ› and ‹AOT_instance_of_cqt_2 ψ›
  shows ‹AOT_instance_of_cqt_2 (λτ. «φ{τ} ∨ ψ{τ}»)›
  using assms
  by (auto simp: AOT_instance_of_cqt_2_def AOT_sem_denotes
                 AOT_model_lambda_denotes AOT_sem_disj)
lemma AOT_instance_of_cqt_2_intros_equib[AOT_instance_of_cqt_2_intro]:
  assumes ‹AOT_instance_of_cqt_2 φ› and ‹AOT_instance_of_cqt_2 ψ›
  shows ‹AOT_instance_of_cqt_2 (λτ. «φ{τ} ≡ ψ{τ}»)›
  using assms
  by (auto simp: AOT_instance_of_cqt_2_def AOT_sem_denotes
                 AOT_model_lambda_denotes AOT_sem_equiv)
lemma AOT_instance_of_cqt_2_intros_forall[AOT_instance_of_cqt_2_intro]:
  assumes ‹⋀ α . AOT_instance_of_cqt_2 (Φ α)›
  shows ‹AOT_instance_of_cqt_2 (λτ. «∀α Φ{α,τ}»)›
  using assms
  by (auto simp: AOT_instance_of_cqt_2_def AOT_sem_denotes
                 AOT_model_lambda_denotes AOT_sem_forall)
lemma AOT_instance_of_cqt_2_intros_exists[AOT_instance_of_cqt_2_intro]:
  assumes ‹⋀ α . AOT_instance_of_cqt_2 (Φ α)›
  shows ‹AOT_instance_of_cqt_2 (λτ. «∃α Φ{α,τ}»)›
  using assms
  by (auto simp: AOT_instance_of_cqt_2_def AOT_sem_denotes
                 AOT_model_lambda_denotes AOT_sem_exists)
lemma AOT_instance_of_cqt_2_intros_exe_arg_self[AOT_instance_of_cqt_2_intro]:
   ‹AOT_instance_of_cqt_2_exe_arg (λx. x)›
  unfolding AOT_instance_of_cqt_2_exe_arg_def AOT_instance_of_cqt_2_def
            AOT_sem_lambda_denotes
  by (auto simp: AOT_model_term_equiv_part_equivp equivp_reflp AOT_sem_denotes)
lemma AOT_instance_of_cqt_2_intros_exe_arg_const[AOT_instance_of_cqt_2_intro]:
     ‹AOT_instance_of_cqt_2_exe_arg (λx. κ)›
  unfolding AOT_instance_of_cqt_2_exe_arg_def AOT_instance_of_cqt_2_def
  by (auto simp: AOT_model_term_equiv_part_equivp equivp_reflp
                 AOT_sem_denotes AOT_sem_lambda_denotes)
lemma AOT_instance_of_cqt_2_intros_exe_arg_fst[AOT_instance_of_cqt_2_intro]:
   ‹AOT_instance_of_cqt_2_exe_arg fst›
  unfolding AOT_instance_of_cqt_2_exe_arg_def AOT_instance_of_cqt_2_def
  by (simp add: AOT_model_term_equiv_prod_def case_prod_beta)
lemma AOT_instance_of_cqt_2_intros_exe_arg_snd[AOT_instance_of_cqt_2_intro]:
   ‹AOT_instance_of_cqt_2_exe_arg snd›
  unfolding AOT_instance_of_cqt_2_exe_arg_def AOT_instance_of_cqt_2_def
  by (simp add: AOT_model_term_equiv_prod_def AOT_sem_denotes AOT_sem_lambda_denotes)
lemma AOT_instance_of_cqt_2_intros_exe_arg_Pair[AOT_instance_of_cqt_2_intro]:
```

```
  assumes ‹AOT_instance_of_cqt_2_exe_arg φ› and ‹AOT_instance_of_cqt_2_exe_arg ψ›
  shows ‹AOT_instance_of_cqt_2_exe_arg (λτ. Pair (φ τ) (ψ τ))›
  using assms
  unfolding AOT_instance_of_cqt_2_exe_arg_def AOT_instance_of_cqt_2_def
            AOT_sem_denotes AOT_sem_lambda_denotes AOT_model_term_equiv_prod_def
            AOT_model_denotes_prod_def
  by auto
lemma AOT_instance_of_cqt_2_intros_desc[AOT_instance_of_cqt_2_intro]:
  assumes ‹⋀z :: 'a::AOT_κ. AOT_instance_of_cqt_2 (Φ z)›
  shows ‹AOT_instance_of_cqt_2_exe_arg (λ κ :: 'b::AOT_κ . «ιz(Φ{z,κ})»)›
proof -
  have 0: ‹⋀ κ κ'. AOT_model_denotes κ ∧ AOT_model_denotes κ' ∧
                   AOT_model_term_equiv κ κ' ⟹
                   Φ z κ = Φ z κ'› for z
    using assms
    unfolding AOT_instance_of_cqt_2_def
              AOT_sem_denotes AOT_model_lambda_denotes by force
  {
    fix κ κ'
    have ‹«ιz(Φ{z,κ})» = «ιz(Φ{z,κ'})»›
      if ‹AOT_model_denotes κ ∧ AOT_model_denotes κ' ∧ AOT_model_term_equiv κ κ'›
      using 0[OF that]
      by auto
    moreover have ‹AOT_model_term_equiv x x› for x :: ‹'a::AOT_κ›
      by (metis AOT_instance_of_cqt_2_exe_arg_def
                AOT_instance_of_cqt_2_intros_exe_arg_const
                AOT_model_A_objects AOT_model_term_equiv_denotes
                AOT_model_term_equiv_eps(1))
    ultimately have ‹AOT_model_term_equiv «ιz(Φ{z,κ})» «ιz(Φ{z,κ'})»›
      if ‹AOT_model_denotes κ ∧ AOT_model_denotes κ' ∧ AOT_model_term_equiv κ κ'›
      using that by simp
  }
  thus ?thesis using 0
    unfolding AOT_instance_of_cqt_2_exe_arg_def
    by simp
qed

lemma AOT_instance_of_cqt_2_intros_exe_const[AOT_instance_of_cqt_2_intro]:
  assumes ‹AOT_instance_of_cqt_2_exe_arg κs›
  shows ‹AOT_instance_of_cqt_2 (λx :: 'b::AOT_κs. AOT_exe Π (κs x))›
  using assms
  unfolding AOT_instance_of_cqt_2_def AOT_sem_denotes AOT_model_lambda_denotes
            AOT_sem_disj AOT_sem_conj
            AOT_sem_not AOT_sem_box AOT_sem_act AOT_instance_of_cqt_2_exe_arg_def
            AOT_sem_equiv AOT_sem_imp AOT_sem_forall AOT_sem_exists AOT_sem_dia
  by (auto intro!: AOT_sem_exe_equiv)
lemma AOT_instance_of_cqt_2_intros_exe_lam[AOT_instance_of_cqt_2_intro]:
  assumes ‹⋀ y . AOT_instance_of_cqt_2 (λx. φ x y)›
    and ‹AOT_instance_of_cqt_2_exe_arg κs›
    shows ‹AOT_instance_of_cqt_2 (λκ₁κₙ :: 'b::AOT_κs.
              «[λν₁...νₙ φ{κ₁...κₙ,ν₁...νₙ}]«κs κ₁κₙ»»)›
proof -
  {
    fix x y :: 'b
    assume ‹AOT_model_denotes x›
    moreover assume ‹AOT_model_denotes y›
    moreover assume ‹AOT_model_term_equiv x y›
    moreover have 1: ‹φ x = φ y›
      using assms calculation unfolding AOT_instance_of_cqt_2_def by blast
    ultimately have ‹AOT_exe (AOT_lambda (φ x)) (κs x) =
                     AOT_exe (AOT_lambda (φ y)) (κs y)›
      unfolding 1
      apply (safe intro!: AOT_sem_exe_equiv)
```

```
      by (metis AOT_instance_of_cqt_2_exe_arg_def assms(2))
  }
  thus ?thesis
  unfolding AOT_instance_of_cqt_2_def
            AOT_instance_of_cqt_2_exe_arg_def
  by blast
qed
lemma AOT_instance_of_cqt_2_intro_prod[AOT_instance_of_cqt_2_intro]:
  assumes ‹⋀ x . AOT_instance_of_cqt_2 (φ x)›
      and ‹⋀ x . AOT_instance_of_cqt_2 (λ z . φ z x)›
  shows ‹AOT_instance_of_cqt_2 (λ(x,y) . φ x y)›
  using assms unfolding AOT_instance_of_cqt_2_def
  by (auto simp add: AOT_model_lambda_denotes AOT_sem_denotes
                AOT_model_denotes_prod_def
                AOT_model_term_equiv_prod_def)

text‹The following are already derivable semantically, but not yet added
     to @{attribute AOT_instance_of_cqt_2_intro}. They will be added with the
     next planned extension of axiom cqt:2.›
named_theorems AOT_instance_of_cqt_2_intro_next
definition AOT_instance_of_cqt_2_enc_arg :: ‹('a::AOT_κs ⇒ 'b::AOT_κs) ⇒ bool› where
  ‹AOT_instance_of_cqt_2_enc_arg ≡ λ φ . ∀ x y z .
      AOT_model_denotes x ∧ AOT_model_denotes y ∧ AOT_model_term_equiv x y ⟶
      AOT_enc (φ x) z = AOT_enc (φ y) z›
definition AOT_instance_of_cqt_2_enc_rel :: ‹('a::AOT_κs ⇒ <'b::AOT_κs>) ⇒ bool› where
  ‹AOT_instance_of_cqt_2_enc_rel ≡ λ φ . ∀ x y z .
      AOT_model_denotes x ∧ AOT_model_denotes y ∧ AOT_model_term_equiv x y ⟶
      AOT_enc z (φ x) = AOT_enc z (φ y)›
lemma AOT_instance_of_cqt_2_intros_enc[AOT_instance_of_cqt_2_intro_next]:
  assumes ‹AOT_instance_of_cqt_2_enc_rel Π› and ‹AOT_instance_of_cqt_2_enc_arg κs›
  shows ‹AOT_instance_of_cqt_2 (λx . AOT_enc (κs x) «[«Π x»]»)›
  using assms
  unfolding AOT_instance_of_cqt_2_def AOT_sem_denotes AOT_model_lambda_denotes
            AOT_instance_of_cqt_2_enc_rel_def AOT_sem_not AOT_sem_box AOT_sem_act
            AOT_sem_dia AOT_sem_conj AOT_sem_disj AOT_sem_equiv AOT_sem_imp
            AOT_sem_forall AOT_sem_exists AOT_instance_of_cqt_2_enc_arg_def
  by fastforce+
lemma AOT_instance_of_cqt_2_enc_arg_intro_const[AOT_instance_of_cqt_2_intro_next]:
  ‹AOT_instance_of_cqt_2_enc_arg (λx. c)›
  unfolding AOT_instance_of_cqt_2_enc_arg_def by simp
lemma AOT_instance_of_cqt_2_enc_arg_intro_desc[AOT_instance_of_cqt_2_intro_next]:
  assumes ‹⋀z :: 'a::AOT_κ. AOT_instance_of_cqt_2 (Φ z)›
  shows ‹AOT_instance_of_cqt_2_enc_arg (λ κ :: 'b::AOT_κ . «ιz(Φ{z,κ})»)›
proof -
  have 0: ‹⋀ κ κ'. AOT_model_denotes κ ∧ AOT_model_denotes κ' ∧
                    AOT_model_term_equiv κ κ' ⟹
                    Φ z κ = Φ z κ'› for z
    using assms
    unfolding AOT_instance_of_cqt_2_def
              AOT_sem_denotes AOT_model_lambda_denotes by force
  {
    fix κ κ'
    have ‹«ιz(Φ{z,κ})» = «ιz(Φ{z,κ'})»›
      if ‹AOT_model_denotes κ ∧ AOT_model_denotes κ' ∧ AOT_model_term_equiv κ κ'›
      using 0[OF that]
      by auto
  }
  thus ?thesis using 0
    unfolding AOT_instance_of_cqt_2_enc_arg_def by meson
qed
lemma AOT_instance_of_cqt_2_enc_rel_intro[AOT_instance_of_cqt_2_intro_next]:
  assumes ‹⋀ κ' . AOT_instance_of_cqt_2 (λκ :: 'a::AOT_κs . φ κ κ')›
  shows ‹AOT_instance_of_cqt_2_enc_rel (λκ :: 'a::AOT_κs. AOT_lambda (λκ'. φ κ κ'))›
```

```
proof -
  {
    fix x y :: 'a and z ::'b
    assume ‹AOT_model_term_equiv x y›
    moreover assume ‹AOT_model_denotes x›
    moreover assume ‹AOT_model_denotes y›
    ultimately have ‹φ x = φ y›
      using assms unfolding AOT_instance_of_cqt_2_def by blast
    hence ‹AOT_enc z (AOT_lambda (φ x)) = AOT_enc z (AOT_lambda (φ y))›
      by simp
  }
  thus ?thesis
    unfolding AOT_instance_of_cqt_2_enc_rel_def by auto
qed

text‹Further restrict unary individual variables to type @{typ κ} (rather
     than class @{class AOT_κ} only) and define being ordinary and being abstract.›
AOT_register_type_constraints
  Individual: ‹κ› ‹_::AOT_κs›

AOT_define AOT_ordinary :: ‹Π› (‹O!›) ‹O! =df [λx ◇E!x]›
declare AOT_ordinary[AOT del, AOT_defs del]
AOT_define AOT_abstract :: ‹Π› (‹A!›) ‹A! =df [λx ¬◇E!x]›
declare AOT_abstract[AOT del, AOT_defs del]

context AOT_meta_syntax
begin
notation AOT_ordinary ("O!")
notation AOT_abstract ("A!")
end
context AOT_no_meta_syntax
begin
no_notation AOT_ordinary ("O!")
no_notation AOT_abstract ("A!")
end

no_translations
  "_AOT_concrete" => "CONST AOT_term_of_var (CONST AOT_concrete)"
parse_translation‹
[(syntax_const‹_AOT_concrete›, fn _ => fn [] =>
  Const (const_name‹AOT_term_of_var›, dummyT)
  $ Const (const_name‹AOT_concrete›, typ‹‹κ› AOT_var›))]
›

text‹Auxiliary lemmata.›
lemma AOT_sem_ordinary: "«O!» = «[λx ◇E!x]»"
  using AOT_ordinary[THEN AOT_sem_id_def0E1] AOT_sem_ordinary_def_denotes
  by (auto simp: AOT_sem_eq)
lemma AOT_sem_abstract: "«A!» = «[λx ¬◇E!x]»"
  using AOT_abstract[THEN AOT_sem_id_def0E1]  AOT_sem_abstract_def_denotes
  by (auto simp: AOT_sem_eq)
lemma AOT_sem_ordinary_denotes: ‹[w ⊨ O!↓]›
  by (simp add: AOT_sem_ordinary AOT_sem_ordinary_def_denotes)
lemma AOT_meta_abstract_denotes: ‹[w ⊨ A!↓]›
  by (simp add: AOT_sem_abstract AOT_sem_abstract_def_denotes)
lemma AOT_model_abstract_ακ: ‹∃ a . κ = ακ a› if ‹[v ⊨ A!κ]›
  using that[unfolded AOT_sem_abstract, simplified
      AOT_meta_abstract_denotes[unfolded AOT_sem_abstract, THEN AOT_sem_lambda_beta,
          OF that[simplified AOT_sem_exe, THEN conjunct2, THEN conjunct1]]]
  apply (simp add: AOT_sem_not AOT_sem_dia AOT_sem_concrete)
  by (metis AOT_model_ω_concrete_in_some_world AOT_model_concrete_κ.simps(1)
            AOT_model_denotes_κ_def AOT_sem_denotes AOT_sem_exe κ.exhaust_disc
            is_ακ_def is_ωκ_def that)
```

```
lemma AOT_model_ordinary_ωκ: ‹∃ a . κ = ωκ a› if ‹[v ⊨ O!κ]›
  using that[unfolded AOT_sem_ordinary, simplified
      AOT_sem_ordinary_denotes[unfolded AOT_sem_ordinary, THEN AOT_sem_lambda_beta,
        OF that[simplified AOT_sem_exe, THEN conjunct2, THEN conjunct1]]]
  apply (simp add: AOT_sem_dia AOT_sem_concrete)
  by (metis AOT_model_concrete_κ.simps(2) AOT_model_concrete_κ.simps(3)
            κ.exhaust_disc is_ακ_def is_ωκ_def is_nullκ_def)
lemma AOT_model_ωκ_ordinary: ‹[v ⊨ O!«ωκ x»]›
  by (metis AOT_model_abstract_ακ AOT_model_denotes_κ_def AOT_sem_abstract
            AOT_sem_denotes AOT_sem_ind_eq AOT_sem_ordinary κ.disc(7) κ.distinct(1))
lemma AOT_model_ακ_ordinary: ‹[v ⊨ A!«ακ x»]›
  by (metis AOT_model_denotes_κ_def AOT_model_ordinary_ωκ AOT_sem_abstract
            AOT_sem_denotes AOT_sem_ind_eq AOT_sem_ordinary κ.disc(8) κ.distinct(1))
AOT_theorem prod_denotesE: assumes ‹«(κ1,κ2)»↓› shows ‹κ1↓ & κ2↓›
  using assms by (simp add: AOT_sem_denotes AOT_sem_conj AOT_model_denotes_prod_def)
declare prod_denotesE[AOT del]
AOT_theorem prod_denotesI: assumes ‹κ1↓ & κ2↓› shows ‹«(κ1,κ2)»↓›
  using assms by (simp add: AOT_sem_denotes AOT_sem_conj AOT_model_denotes_prod_def)
declare prod_denotesI[AOT del]


text‹Prepare the derivation of the additional axioms that are validated by
     our extended models.›
locale AOT_ExtendedModel =
  assumes AOT_ExtendedModel: ‹AOT_ExtendedModel›
begin
lemma AOT_sem_indistinguishable_ord_enc_all:
  assumes Π_den: ‹[v ⊨ Π↓]›
  assumes Ax: ‹[v ⊨ A!x]›
  assumes Ay: ‹[v ⊨ A!y]›
  assumes indist: ‹[v ⊨ ∀F □([F]x ≡ [F]y)]›
  shows
  ‹[v ⊨ ∀G(∀z(O!z → □([G]z ≡ [Π]z)) → x[G])] =
   [v ⊨ ∀G(∀z(O!z → □([G]z ≡ [Π]z)) → y[G])]›
proof -
    have 0: ‹[v ⊨ [λx ¬♢[E!]x]x]›
      using Ax by (simp add: AOT_sem_abstract)
    have 1: ‹[v ⊨ [λx ¬♢[E!]x]y]›
      using Ay by (simp add: AOT_sem_abstract)
    {
      assume ‹[v ⊨ ∀G(∀z (O!z → □([G]z ≡ [Π]z)) → x[G])]›
      hence 3: ‹[v ⊨ ∀G(∀z([λx ♢[E!]x]z → □([G]z ≡ [Π]z)) → x[G])]›
        by (simp add: AOT_sem_ordinary)
      {
        fix Π' :: ‹<κ>›
        assume 1: ‹[v ⊨ Π'↓]›
        assume 2: ‹[v ⊨ [λx ♢[E!]x]z → □([Π']z ≡ [Π]z)]› for z
        have ‹[v ⊨ x[Π']]›
          using 3
          by (auto simp: AOT_sem_forall AOT_sem_imp AOT_sem_box AOT_sem_denotes)
             (metis (no_types, lifting) 1 2 AOT_term_of_var_cases AOT_sem_box
                                        AOT_sem_denotes AOT_sem_imp)
      } note 3 = this
      fix Π' :: ‹<κ>›
      assume Π_den: ‹[v ⊨ Π'↓]›
      assume 4: ‹[v ⊨ ∀z (O!z → □([Π']z ≡ [Π]z))]›
      {
        fix κ0
        assume ‹[v ⊨ [λx ♢[E!]x]κ0]›
        hence ‹[v ⊨ O!κ0]›
          using AOT_sem_ordinary by metis
        moreover have ‹[v ⊨ κ0↓]›
          using calculation by (simp add: AOT_sem_exe)
```

```
          ultimately have ‹[v ⊨ □([Π']κ0 ≡ [Π]κ0)]›
            using 4 by (auto simp: AOT_sem_forall AOT_sem_imp)
        } note 4 = this
        have ‹[v ⊨ y[Π']]›
          apply (rule AOT_sem_enc_indistinguishable_all[OF AOT_ExtendedModel])
          apply (fact 0)
               apply (auto simp: 0 1 Π_den indist[simplified AOT_sem_forall
                                   AOT_sem_box AOT_sem_equiv])
          apply (rule 3)
           apply auto[1]
          using 4
          by (auto simp: AOT_sem_imp AOT_sem_equiv AOT_sem_box)
      }
      moreover {
      {
        assume ‹[v ⊨ ∀G(∀z (O!z → □([G]z ≡ [Π]z)) → y[G])]›
        hence 3: ‹[v ⊨ ∀G(∀z ([λx ◇[E!]x]z → □([G]z ≡ [Π]z)) → y[G])]›
          by (simp add: AOT_sem_ordinary)
        {
          fix Π' :: ‹<κ>›
          assume 1: ‹[v ⊨ Π'↓]›
          assume 2: ‹[v ⊨ [λx ◇[E!]x]z → □([Π']z ≡ [Π]z)]› for z
          have ‹[v ⊨ y[Π']]›
            using 3
            apply (auto simp: AOT_sem_forall AOT_sem_imp AOT_sem_box AOT_sem_denotes)
            by (metis (no_types, lifting) 1 2 AOT_model.AOT_term_of_var_cases
                                         AOT_sem_box AOT_sem_denotes AOT_sem_imp)
        } note 3 = this
        fix Π' :: ‹<κ>›
        assume Π_den: ‹[v ⊨ Π'↓]›
        assume 4: ‹[v ⊨ ∀z (O!z → □([Π']z ≡ [Π]z))]›
        {
          fix κ0
          assume ‹[v ⊨ [λx ◇[E!]x]κ0]›
          hence ‹[v ⊨ O!κ0]›
            using AOT_sem_ordinary by metis
          moreover have ‹[v ⊨ κ0↓]›
            using calculation by (simp add: AOT_sem_exe)
          ultimately have ‹[v ⊨ □([Π']κ0 ≡ [Π]κ0)]›
            using 4 by (auto simp: AOT_sem_forall AOT_sem_imp)
        } note 4 = this
        have ‹[v ⊨ x[Π']]›
          apply (rule AOT_sem_enc_indistinguishable_all[OF AOT_ExtendedModel])
                apply (fact 1)
               apply (auto simp: 0 1 Π_den indist[simplified AOT_sem_forall
                                   AOT_sem_box AOT_sem_equiv])
          apply (rule 3)
           apply auto[1]
          using 4
          by (auto simp: AOT_sem_imp AOT_sem_equiv AOT_sem_box)
      }
    }
    ultimately show ‹[v ⊨ ∀G (∀z (O!z → □([G]z ≡ [Π]z)) → x[G])] =
          [v ⊨ ∀G (∀z (O!z → □([G]z ≡ [Π]z)) → y[G])]›
      by (auto simp: AOT_sem_forall AOT_sem_imp)
qed

lemma AOT_sem_indistinguishable_ord_enc_ex:
  assumes Π_den: ‹[v ⊨ Π↓]›
  assumes Ax: ‹[v ⊨ A!x]›
  assumes Ay: ‹[v ⊨ A!y]›
  assumes indist: ‹[v ⊨ ∀F □([F]x ≡ [F]y)]›
  shows ‹[v ⊨ ∃G(∀z (O!z → □([G]z ≡ [Π]z)) & x[G])] =
```

```
        [v ⊨ ∃G(∀z(O!z → □([G]z ≡ [Π]z)) & y[G])]›
proof -
  have Aux: ‹[v ⊨ [λx ◇[E!]x]κ] = ([v ⊨ [λx ◇[E!]x]κ] ∧ [v ⊨ κ↓])› for v κ
    using AOT_sem_exe by blast
  AOT_modally_strict {
    fix x y
    AOT_assume Π_den: ‹[Π]↓›
    AOT_assume 2: ‹∀F □([F]x ≡ [F]y)›
    AOT_assume ‹A!x›
    AOT_hence 0: ‹[λx ¬◇[E!]x]x›
      by (simp add: AOT_sem_abstract)
    AOT_assume ‹A!y›
    AOT_hence 1: ‹[λx ¬◇[E!]x]y›
      by (simp add: AOT_sem_abstract)
    {
      AOT_assume ‹∃G(∀z (O!z → □([G]z ≡ [Π]z)) & x[G])›
      then AOT_obtain Π'
        where Π'_den: ‹Π'↓›
          and Π'_indist: ‹∀z (O!z → □([Π']z ≡ [Π]z))›
          and x_enc_Π': ‹x[Π']›
        by (meson AOT_sem_conj AOT_sem_exists)
      {
        fix κ₀
        AOT_assume ‹[λx ◇[E!]x]κ₀›
        AOT_hence ‹□([Π']κ₀ ≡ [Π]κ₀)›
          using Π'_indist
          by (auto simp: AOT_sem_exe AOT_sem_imp AOT_sem_exists AOT_sem_conj
                         AOT_sem_ordinary AOT_sem_forall)
      } note 3 = this
      AOT_have ‹∀z ([λx ◇[E!]x]z → □([Π']z ≡ [Π]z))›
        using Π'_indist by (simp add: AOT_sem_ordinary)
      AOT_obtain Π" where
          Π"_den: ‹Π"↓› and
          Π"_indist: ‹[λx ◇[E!]x]κ₀ → □([Π"]κ₀ ≡ [Π]κ₀)› and
          y_enc_Π": ‹y[Π"]› for κ₀
        using AOT_sem_enc_indistinguishable_ex[OF AOT_ExtendedModel,
                OF 0, OF 1, rotated, OF Π_den,
                OF exI[where x=Π'], OF conjI, OF Π'_den, OF conjI,
                OF x_enc_Π', OF allI, OF impI,
                OF 3[simplified AOT_sem_box AOT_sem_equiv], simplified, OF
                2[simplified AOT_sem_forall AOT_sem_equiv AOT_sem_box,
                  THEN spec, THEN mp, THEN spec], simplified]
        unfolding AOT_sem_imp AOT_sem_box AOT_sem_equiv by blast
      {
        AOT_have ‹Π"↓›
            and ‹∀x ([λx ◇[E!]x]x → □([Π"]x ≡ [Π]x))›
            and ‹y[Π"]›
          apply (simp add: Π"_den)
          apply (simp add: AOT_sem_forall Π"_indist)
          by (simp add: y_enc_Π")
      } note 2 = this
      AOT_have ‹∃G(∀z (O!z → □([G]z ≡ [Π]z)) & y[G])›
        apply (auto simp: AOT_sem_exists AOT_sem_ordinary
            AOT_sem_imp AOT_sem_box AOT_sem_forall AOT_sem_equiv AOT_sem_conj)
        using 2[simplified AOT_sem_box AOT_sem_equiv AOT_sem_imp AOT_sem_forall]
        by blast
    }
  } note 0 = this
  AOT_modally_strict {
    {
      fix x y
      AOT_assume Π_den: ‹[Π]↓›
      moreover AOT_assume ‹∀F □([F]x ≡ [F]y)›
```

```
      moreover AOT_have ‹∀F □([F]y ≡ [F]x)›
        using calculation(2)
        by (auto simp: AOT_sem_forall AOT_sem_box AOT_sem_equiv)
      moreover AOT_assume ‹A!x›
      moreover AOT_assume ‹A!y›
      ultimately AOT_have ‹∃G (∀z (O!z → □([G]z ≡ [Π]z)) & x[G]) ≡
                           ∃G (∀z (O!z → □([G]z ≡ [Π]z)) & y[G])›
        using 0 by (auto simp: AOT_sem_equiv)
    }
    have 1: ‹[v ⊨ ∀F □([F]y ≡ [F]x)]›
      using indist
      by (auto simp: AOT_sem_forall AOT_sem_box AOT_sem_equiv)
    thus ‹[v ⊨ ∃G (∀z (O!z → □([G]z ≡ [Π]z)) & x[G])] =
        [v ⊨ ∃G (∀z (O!z → □([G]z ≡ [Π]z)) & y[G])]›
      using assms
      by (auto simp: AOT_sem_imp AOT_sem_conj AOT_sem_equiv 0)
  }
qed
end


(* Collect all theorems that are not in Main and not declared [AOT]
   and store them in a blacklist. *)
setup‹setup_AOT_no_atp›
bundle AOT_no_atp begin declare AOT_no_atp[no_atp] end
(* Can be used as: "including AOT_no_atp sledgehammer" or
   "sledgehammer(del: AOT_no_atp) *)

(*<*)
end
(*>*)
```

A.5. Definitions of AOT

```
theory AOT_Definitions
  imports AOT_semantics
begin

section‹Definitions of AOT›

AOT_theorem "conventions:1": ‹φ & ψ ≡df ¬(φ → ¬ψ)›                    (18.1)
  using AOT_conj.
AOT_theorem "conventions:2": ‹φ ∨ ψ ≡df ¬φ → ψ›                       (18.2)
  using AOT_disj.
AOT_theorem "conventions:3": ‹φ ≡ ψ ≡df (φ → ψ) & (ψ → φ)›            (18.3)
  using AOT_equiv.
AOT_theorem "conventions:4": ‹∃α φ{α} ≡df ¬∀α ¬φ{α}›                  (18.4)
  using AOT_exists.
AOT_theorem "conventions:5": ‹◇φ ≡df ¬□¬φ›                            (18.5)
  using AOT_dia.

declare "conventions:1"[AOT_defs] "conventions:2"[AOT_defs]
        "conventions:3"[AOT_defs] "conventions:4"[AOT_defs]
        "conventions:5"[AOT_defs]

notepad
begin
  fix φ ψ χ
  text‹\linelabel{precedence}›
  have "conventions3[1]": ‹«φ → ψ ≡ ¬ψ → ¬φ» = «(φ → ψ) ≡ (¬ψ → ¬φ)»›   (19)
    by blast
  have "conventions3[2]": ‹«φ & ψ → χ» = «(φ & ψ) → χ»›                 (19)
                      and ‹«φ ∨ ψ → χ» = «(φ ∨ ψ) → χ»›
    by blast+
  have "conventions3[3]": ‹«φ ∨ ψ & χ» = «(φ ∨ ψ) & χ»›                 (19)
                      and ‹«φ & ψ ∨ χ» = «(φ & ψ) ∨ χ»›
     by blast+ ― ‹Note that PLM instead generally uses parenthesis in these cases.›
end


AOT_theorem "existence:1": ‹κ↓ ≡df ∃F [F]κ›                            (20.1)
  by (simp add: AOT_sem_denotes AOT_sem_exists AOT_model_equiv_def)
     (metis AOT_sem_denotes AOT_sem_exe AOT_sem_lambda_beta AOT_sem_lambda_denotes)
AOT_theorem "existence:2": ‹Π↓ ≡df ∃x1...∃xn x1...xn[Π]›                (20.2)
  using AOT_sem_denotes AOT_sem_enc_denotes AOT_sem_universal_encoder
  by (simp add: AOT_sem_denotes AOT_sem_exists AOT_model_equiv_def) blast
AOT_theorem "existence:2[1]": ‹Π↓ ≡df ∃x x[Π]›                          (20.2)
  using "existence:2"[of Π] by simp
AOT_theorem "existence:2[2]": ‹Π↓ ≡df ∃x∃y xy[Π]›                       (20.2)
  using "existence:2"[of Π]
  by (simp add: AOT_sem_denotes AOT_sem_exists AOT_model_equiv_def
                AOT_model_denotes_prod_def)
AOT_theorem "existence:2[3]": ‹Π↓ ≡df ∃x∃y∃z xyz[Π]›                    (20.2)
  using "existence:2"[of Π]
  by (simp add: AOT_sem_denotes AOT_sem_exists AOT_model_equiv_def
                AOT_model_denotes_prod_def)
AOT_theorem "existence:2[4]": ‹Π↓ ≡df ∃x1∃x2∃x3∃x4 x1x2x3x4[Π]›         (20.2)
  using "existence:2"[of Π]
  by (simp add: AOT_sem_denotes AOT_sem_exists AOT_model_equiv_def
                AOT_model_denotes_prod_def)

AOT_theorem "existence:3": ‹φ↓ ≡df [λx φ]↓›                           (20.3)
  by (simp add: AOT_sem_denotes AOT_model_denotes_o_def AOT_model_equiv_def
                AOT_model_lambda_denotes)

```

```
declare "existence:1"[AOT_defs] "existence:2"[AOT_defs] "existence:2[1]"[AOT_defs]
        "existence:2[2]"[AOT_defs] "existence:2[3]"[AOT_defs]
        "existence:2[4]"[AOT_defs] "existence:3"[AOT_defs]


AOT_theorem "oa:1": ‹O! =df [λx ◇E!x]› using AOT_ordinary .                 (22.1)
AOT_theorem "oa:2": ‹A! =df [λx ¬◇E!x]› using AOT_abstract .                (22.2)

declare "oa:1"[AOT_defs] "oa:2"[AOT_defs]

AOT_theorem "identity:1":                                                   (23.1)
  ‹x = y ≡df ([O!]x & [O!]y & □∀F ([F]x ≡ [F]y)) ∨
             ([A!]x & [A!]y & □∀F (x[F] ≡ y[F]))›
  unfolding AOT_model_equiv_def
  using AOT_sem_ind_eq[of _ x y]
  by (simp add: AOT_sem_ordinary AOT_sem_abstract AOT_sem_conj
                AOT_sem_box AOT_sem_equiv AOT_sem_forall AOT_sem_disj AOT_sem_eq
                AOT_sem_denotes)

AOT_theorem "identity:2":                                                   (23.2)
  ‹F = G ≡df F↓ & G↓ & □∀x(x[F] ≡ x[G])›
  using AOT_sem_enc_eq[of _ F G]
  by (auto simp: AOT_model_equiv_def AOT_sem_imp AOT_sem_denotes AOT_sem_eq
                 AOT_sem_conj AOT_sem_forall AOT_sem_box AOT_sem_equiv)

AOT_theorem "identity:3[2]":                                                (23.3)
  ‹F = G ≡df F↓ & G↓ & ∀y([λz [F]zy] = [λz [G]zy] & [λz [F]yz] = [λz [G]yz])›
  by (auto simp: AOT_model_equiv_def AOT_sem_proj_id_prop[of _ F G]
                 AOT_sem_proj_id_prod_def AOT_sem_conj AOT_sem_denotes
                 AOT_sem_forall AOT_sem_unary_proj_id AOT_model_denotes_prod_def)
AOT_theorem "identity:3[3]":                                                (23.3)
  ‹F = G ≡df F↓ & G↓ & ∀y1∀y2([λz [F]zy1y2] = [λz [G]zy1y2] &
                              [λz [F]y1zy2] = [λz [G]y1zy2] &
                              [λz [F]y1y2z] = [λz [G]y1y2z])›
  by (auto simp: AOT_model_equiv_def AOT_sem_proj_id_prop[of _ F G]
                 AOT_sem_proj_id_prod_def AOT_sem_conj AOT_sem_denotes
                 AOT_sem_forall AOT_sem_unary_proj_id AOT_model_denotes_prod_def)
AOT_theorem "identity:3[4]":                                                (23.3)
  ‹F = G ≡df F↓ & G↓ & ∀y1∀y2∀y3([λz [F]zy1y2y3] = [λz [G]zy1y2y3] &
                                [λz [F]y1zy2y3] = [λz [G]y1zy2y3] &
                                [λz [F]y1y2zy3] = [λz [G]y1y2zy3] &
                                [λz [F]y1y2y3z] = [λz [G]y1y2y3z])›
  by (auto simp: AOT_model_equiv_def AOT_sem_proj_id_prop[of _ F G]
                 AOT_sem_proj_id_prod_def AOT_sem_conj AOT_sem_denotes
                 AOT_sem_forall AOT_sem_unary_proj_id AOT_model_denotes_prod_def)
AOT_theorem "identity:3":                                                   (23.3)
  ‹F = G ≡df F↓ & G↓ & ∀x1...∀xn «AOT_sem_proj_id x1xn (λ τ . AOT_exe F τ)
                                                      (λ τ . AOT_exe G τ)»›
  by (auto simp: AOT_model_equiv_def AOT_sem_proj_id_prop[of _ F G]
                 AOT_sem_proj_id_prod_def AOT_sem_conj AOT_sem_denotes
                 AOT_sem_forall AOT_sem_unary_proj_id AOT_model_denotes_prod_def)

AOT_theorem "identity:4":                                                   (23.4)
  ‹p = q ≡df p↓ & q↓ & [λx p] = [λx q]›
  by (auto simp: AOT_model_equiv_def AOT_sem_eq AOT_sem_denotes AOT_sem_conj
                 AOT_model_lambda_denotes AOT_sem_lambda_eq_prop_eq)

declare "identity:1"[AOT_defs] "identity:2"[AOT_defs] "identity:3[2]"[AOT_defs]
        "identity:3[3]"[AOT_defs] "identity:3[4]"[AOT_defs] "identity:3"[AOT_defs]
        "identity:4"[AOT_defs]

AOT_define AOT_nonidentical :: ‹τ ⇒ τ ⇒ φ› (infixl "≠" 50)
  "=-infix": ‹τ ≠ σ ≡df ¬(τ = σ)›                                            (24)
```

```
context AOT_meta_syntax
begin
notation AOT_nonidentical (infixl "≠" 50)
end
context AOT_no_meta_syntax
begin
no_notation AOT_nonidentical (infixl "≠" 50)
end


text‹The following are purely technical pseudo-definitions required due to
     our internal implementation of n-ary relations and ellipses using tuples.›
AOT_theorem tuple_denotes: ‹«(τ,τ')»↓ ≡df τ↓ & τ'↓›
  by (simp add: AOT_model_denotes_prod_def AOT_model_equiv_def
                AOT_sem_conj AOT_sem_denotes)
AOT_theorem tuple_identity_1: ‹«(τ,τ')» = «(σ, σ')» ≡df (τ = σ) & (τ' = σ')›
  by (auto simp: AOT_model_equiv_def AOT_sem_conj AOT_sem_eq
                 AOT_model_denotes_prod_def AOT_sem_denotes)
AOT_theorem tuple_forall: ‹∀α1...∀αn φ{α1...αn} ≡df ∀α1(∀α2...∀αn φ{«(α1, α2αn)»})›
  by (auto simp: AOT_model_equiv_def AOT_sem_forall AOT_sem_denotes
                 AOT_model_denotes_prod_def)
AOT_theorem tuple_exists: ‹∃α1...∃αn φ{α1...αn} ≡df ∃α1(∃α2...∃αn φ{«(α1, α2αn)»})›
  by (auto simp: AOT_model_equiv_def AOT_sem_exists AOT_sem_denotes
                 AOT_model_denotes_prod_def)
declare tuple_denotes[AOT_defs] tuple_identity_1[AOT_defs] tuple_forall[AOT_defs]
        tuple_exists[AOT_defs]

end
```

A.6. Axioms of AOT

```
(*<*)
theory AOT_Axioms
  imports AOT_Definitions
begin
(*>*)

section‹Axioms of PLM›

AOT_axiom "pl:1": ‹φ → (ψ → φ)›                                              (38.1)
  by (auto simp: AOT_sem_imp AOT_model_axiomI)
AOT_axiom "pl:2": ‹(φ → (ψ → χ)) → ((φ → ψ) → (φ → χ))›                     (38.2)
  by (auto simp: AOT_sem_imp AOT_model_axiomI)
AOT_axiom "pl:3": ‹(¬φ → ¬ψ) → ((¬φ → ψ) → φ)›                               (38.3)
  by (auto simp: AOT_sem_imp AOT_sem_not AOT_model_axiomI)

AOT_axiom "cqt:1": ‹∀α φ{α} → (τ↓ → φ{τ})›                                    (39.1)
  by (auto simp: AOT_sem_denotes AOT_sem_forall AOT_sem_imp AOT_model_axiomI)

AOT_axiom "cqt:2[const_var]": ‹α↓›                                             (39.2)
  using AOT_sem_vars_denote by (rule AOT_model_axiomI)
AOT_axiom "cqt:2[lambda]":                                                    (39.2)
  assumes ‹INSTANCE_OF_CQT_2(φ)›
  shows ‹[λν1...νn φ{ν1...νn}]↓›
  by (auto intro!: AOT_model_axiomI AOT_sem_cqt_2[OF assms])
AOT_axiom "cqt:2[lambda0]":                                                   (39.2)
  shows ‹[λ φ]↓›
  by (auto intro!: AOT_model_axiomI
           simp: AOT_sem_lambda_denotes "existence:3"[unfolded AOT_model_equiv_def])

AOT_axiom "cqt:3": ‹∀α (φ{α} → ψ{α}) → (∀α φ{α} → ∀α ψ{α})›                    (39.3)
  by (simp add: AOT_sem_forall AOT_sem_imp AOT_model_axiomI)
AOT_axiom "cqt:4": ‹φ → ∀α φ›                                                  (39.4)
  by (simp add: AOT_sem_forall AOT_sem_imp AOT_model_axiomI)
AOT_axiom "cqt:5:a": ‹[Π]κ1...κn → (Π↓ & κ1...κn↓)›                            (39.5.a)
  by (simp add: AOT_sem_conj AOT_sem_denotes AOT_sem_exe
                AOT_sem_imp AOT_model_axiomI)
AOT_axiom "cqt:5:a[1]": ‹[Π]κ → (Π↓ & κ↓)›                                     (39.5.a)
  using "cqt:5:a" AOT_model_axiomI by blast
AOT_axiom "cqt:5:a[2]": ‹[Π]κ1κ2 → (Π↓ & κ1↓ & κ2↓)›                           (39.5.a)
  by (rule AOT_model_axiomI)
     (metis AOT_model_denotes_prod_def AOT_sem_conj AOT_sem_denotes AOT_sem_exe
            AOT_sem_imp case_prodD)
AOT_axiom "cqt:5:a[3]": ‹[Π]κ1κ2κ3 → (Π↓ & κ1↓ & κ2↓ & κ3↓)›                   (39.5.a)
  by (rule AOT_model_axiomI)
     (metis AOT_model_denotes_prod_def AOT_sem_conj AOT_sem_denotes AOT_sem_exe
            AOT_sem_imp case_prodD)
AOT_axiom "cqt:5:a[4]": ‹[Π]κ1κ2κ3κ4 → (Π↓ & κ1↓ & κ2↓ & κ3↓ & κ4↓)›           (39.5.a)
  by (rule AOT_model_axiomI)
     (metis AOT_model_denotes_prod_def AOT_sem_conj AOT_sem_denotes AOT_sem_exe
            AOT_sem_imp case_prodD)
AOT_axiom "cqt:5:b": ‹κ1...κn[Π] → (Π↓ & κ1...κn↓)›                            (39.5.b)
  using AOT_sem_enc_denotes
  by (auto intro!: AOT_model_axiomI simp: AOT_sem_conj AOT_sem_denotes AOT_sem_imp)+
AOT_axiom "cqt:5:b[1]": ‹κ[Π] → (Π↓ & κ↓)›                                     (39.5.b)
  using "cqt:5:b" AOT_model_axiomI by blast
AOT_axiom "cqt:5:b[2]": ‹κ1κ2[Π] → (Π↓ & κ1↓ & κ2↓)›                           (39.5.b)
  by (rule AOT_model_axiomI)
     (metis AOT_model_denotes_prod_def AOT_sem_conj AOT_sem_denotes
            AOT_sem_enc_denotes AOT_sem_imp case_prodD)
AOT_axiom "cqt:5:b[3]": ‹κ1κ2κ3[Π] → (Π↓ & κ1↓ & κ2↓ & κ3↓)›                   (39.5.b)
  by (rule AOT_model_axiomI)
```

```
    (metis AOT_model_denotes_prod_def AOT_sem_conj AOT_sem_denotes
           AOT_sem_enc_denotes AOT_sem_imp case_prodD)
AOT_axiom "cqt:5:b[4]": ‹κ₁κ₂κ₃κ₄[Π] → (Π↓ & κ₁↓ & κ₂↓ & κ₃↓ & κ₄↓)›      (39.5.b)
  by (rule AOT_model_axiomI)
     (metis AOT_model_denotes_prod_def AOT_sem_conj AOT_sem_denotes
            AOT_sem_enc_denotes AOT_sem_imp case_prodD)

AOT_axiom "l-identity": ‹α = β → (φ{α} → φ{β})›      (41)
  by (rule AOT_model_axiomI)
     (simp add: AOT_sem_eq AOT_sem_imp)

AOT_act_axiom "logic-actual": ‹𝒜φ → φ›      (43)
  by (rule AOT_model_act_axiomI)
     (simp add: AOT_sem_act AOT_sem_imp)

AOT_axiom "logic-actual-nec:1": ‹𝒜¬φ ≡ ¬𝒜φ›      (44.1)
  by (rule AOT_model_axiomI)
     (simp add: AOT_sem_act AOT_sem_equiv AOT_sem_not)
AOT_axiom "logic-actual-nec:2": ‹𝒜(φ → ψ) ≡ (𝒜φ → 𝒜ψ)›      (44.2)
  by (rule AOT_model_axiomI)
     (simp add: AOT_sem_act AOT_sem_equiv AOT_sem_imp)

AOT_axiom "logic-actual-nec:3": ‹𝒜(∀α φ{α}) ≡ ∀α 𝒜φ{α}›      (44.3)
  by (rule AOT_model_axiomI)
     (simp add: AOT_sem_act AOT_sem_equiv AOT_sem_forall AOT_sem_denotes)
AOT_axiom "logic-actual-nec:4": ‹𝒜φ ≡ 𝒜𝒜φ›      (44.4)
  by (rule AOT_model_axiomI)
     (simp add: AOT_sem_act AOT_sem_equiv)

AOT_axiom "qml:1": ‹□(φ → ψ) → (□φ → □ψ)›      (45.1)
  by (rule AOT_model_axiomI)
     (simp add: AOT_sem_box AOT_sem_imp)
AOT_axiom "qml:2": ‹□φ → φ›      (45.2)
  by (rule AOT_model_axiomI)
     (simp add: AOT_sem_box AOT_sem_imp)
AOT_axiom "qml:3": ‹◇φ → □◇φ›      (45.3)
  by (rule AOT_model_axiomI)
     (simp add: AOT_sem_box AOT_sem_dia AOT_sem_imp)

AOT_axiom "qml:4": ‹◇∃x (E!x & ¬𝒜E!x)›      (45.4)
  using AOT_sem_concrete AOT_model_contingent
  by (auto intro!: AOT_model_axiomI
             simp: AOT_sem_box AOT_sem_dia AOT_sem_imp AOT_sem_exists
                   AOT_sem_denotes AOT_sem_conj AOT_sem_not AOT_sem_act
                   AOT_sem_exe)+

AOT_axiom "qml-act:1": ‹𝒜φ → □𝒜φ›      (46.1)
  by (rule AOT_model_axiomI)
     (simp add: AOT_sem_act AOT_sem_box AOT_sem_imp)
AOT_axiom "qml-act:2": ‹□φ ≡ 𝒜□φ›      (46.2)
  by (rule AOT_model_axiomI)
     (simp add: AOT_sem_act AOT_sem_box AOT_sem_equiv)

AOT_axiom descriptions: ‹x = ιx(φ{x}) ≡ ∀z(𝒜φ{z} ≡ z = x)›      (47)
proof (rule AOT_model_axiomI)
  AOT_modally_strict {
    AOT_show ‹x = ιx(φ{x}) ≡ ∀z(𝒜φ{z} ≡ z = x)›
      by (induct; simp add: AOT_sem_equiv AOT_sem_forall AOT_sem_act AOT_sem_eq)
         (metis (no_types, opaque_lifting) AOT_sem_desc_denotes AOT_sem_desc_prop
                                           AOT_sem_denotes)
  }
qed
```

```
AOT_axiom "lambda-predicates:1":                                                      (48.1)
  ‹[λν₁...νₙ φ{ν₁...νₙ}]↓ → [λν₁...νₙ φ{ν₁...νₙ}] = [λμ₁...μₙ φ{μ₁...μₙ}]›
  by (rule AOT_model_axiomI)
     (simp add: AOT_sem_denotes AOT_sem_eq AOT_sem_imp)
AOT_axiom "lambda-predicates:1[zero]": ‹[λ p]↓ → [λ p] = [λ p]›                        (48.1)
  by (rule AOT_model_axiomI)
     (simp add: AOT_sem_denotes AOT_sem_eq AOT_sem_imp)
AOT_axiom "lambda-predicates:2":                                                      (48.2)
  ‹[λx₁...xₙ φ{x₁...xₙ}]↓ → ([λx₁...xₙ φ{x₁...xₙ}]x₁...xₙ ≡ φ{x₁...xₙ})›
  by (rule AOT_model_axiomI)
     (simp add: AOT_sem_equiv AOT_sem_imp AOT_sem_lambda_beta AOT_sem_vars_denote)
AOT_axiom "lambda-predicates:3": ‹[λx₁...xₙ [F]x₁...xₙ] = F›                            (48.3)
  by (rule AOT_model_axiomI)
     (simp add: AOT_sem_lambda_eta AOT_sem_vars_denote)
AOT_axiom "lambda-predicates:3[zero]": ‹[λ p] = p›                                     (48.3)
  by (rule AOT_model_axiomI)
     (simp add: AOT_sem_eq AOT_sem_lambda0 AOT_sem_vars_denote)

AOT_axiom "safe-ext":                                                                 (49)
  ‹([λν₁...νₙ φ{ν₁...νₙ}]↓ & □∀ν₁...∀νₙ (φ{ν₁...νₙ} ≡ ψ{ν₁...νₙ})) →
   [λν₁...νₙ ψ{ν₁...νₙ}]↓›
  using AOT_sem_lambda_coex
  by (auto intro!: AOT_model_axiomI simp: AOT_sem_imp AOT_sem_denotes AOT_sem_conj
                   AOT_sem_equiv AOT_sem_box AOT_sem_forall)
AOT_axiom "safe-ext[2]":                                                              (49)
  ‹([λν₁ν₂ φ{ν₁,ν₂}]↓ & □∀ν₁∀ν₂ (φ{ν₁, ν₂} ≡ ψ{ν₁, ν₂})) →
   [λν₁ν₂ ψ{ν₁,ν₂}]↓›
  using "safe-ext"[where φ="λ(x,y). φ x y"]
  by (simp add: AOT_model_axiom_def AOT_sem_denotes AOT_model_denotes_prod_def
                AOT_sem_forall AOT_sem_imp AOT_sem_conj AOT_sem_equiv AOT_sem_box)
AOT_axiom "safe-ext[3]":                                                              (49)
  ‹([λν₁ν₂ν₃ φ{ν₁,ν₂,ν₃}]↓ & □∀ν₁∀ν₂∀ν₃ (φ{ν₁, ν₂, ν₃} ≡ ψ{ν₁, ν₂, ν₃})) →
   [λν₁ν₂ν₃ ψ{ν₁,ν₂,ν₃}]↓›
  using "safe-ext"[where φ="λ(x,y,z). φ x y z"]
  by (simp add: AOT_model_axiom_def AOT_model_denotes_prod_def AOT_sem_forall
                AOT_sem_denotes AOT_sem_imp AOT_sem_conj AOT_sem_equiv AOT_sem_box)
AOT_axiom "safe-ext[4]":                                                              (49)
  ‹([λν₁ν₂ν₃ν₄ φ{ν₁,ν₂,ν₃,ν₄}]↓ &
    □∀ν₁∀ν₂∀ν₃∀ν₄ (φ{ν₁, ν₂, ν₃, ν₄} ≡ ψ{ν₁, ν₂, ν₃, ν₄})) →
   [λν₁ν₂ν₃ν₄ ψ{ν₁,ν₂,ν₃,ν₄}]↓›
  using "safe-ext"[where φ="λ(x,y,z,w). φ x y z w"]
  by (simp add: AOT_model_axiom_def AOT_model_denotes_prod_def AOT_sem_forall
                AOT_sem_denotes AOT_sem_imp AOT_sem_conj AOT_sem_equiv AOT_sem_box)

AOT_axiom "nary-encoding[2]":                                                         (50)
  ‹x₁x₂[F] ≡ x₁[λy [F]yx₂] & x₂[λy [F]x₁y]›
  by (rule AOT_model_axiomI)
     (simp add: AOT_sem_conj AOT_sem_equiv AOT_enc_prod_def AOT_proj_enc_prod_def
                AOT_sem_unary_proj_enc AOT_sem_vars_denote)
AOT_axiom "nary-encoding[3]":                                                         (50)
  ‹x₁x₂x₃[F] ≡ x₁[λy [F]yx₂x₃] & x₂[λy [F]x₁yx₃] & x₃[λy [F]x₁x₂y]›
  by (rule AOT_model_axiomI)
     (simp add: AOT_sem_conj AOT_sem_equiv AOT_enc_prod_def AOT_proj_enc_prod_def
                AOT_sem_unary_proj_enc AOT_sem_vars_denote)
AOT_axiom "nary-encoding[4]":                                                         (50)
  ‹x₁x₂x₃x₄[F] ≡ x₁[λy [F]yx₂x₃x₄] &
                 x₂[λy [F]x₁yx₃x₄] &
                 x₃[λy [F]x₁x₂yx₄] &
                 x₄[λy [F]x₁x₂x₃y]›
  by (rule AOT_model_axiomI)
     (simp add: AOT_sem_conj AOT_sem_equiv AOT_enc_prod_def AOT_proj_enc_prod_def
                AOT_sem_unary_proj_enc AOT_sem_vars_denote)
```

```
AOT_axiom encoding: ‹x[F] → □x[F]›                                        (51)
  using AOT_sem_enc_nec
  by (auto intro!: AOT_model_axiomI simp: AOT_sem_imp AOT_sem_box)

AOT_axiom nocoder: ‹O!x → ¬∃F x[F]›                                       (52)
  by (auto intro!: AOT_model_axiomI
           simp: AOT_sem_imp AOT_sem_not AOT_sem_exists AOT_sem_ordinary
                 AOT_sem_dia
                AOT_sem_lambda_beta[OF AOT_sem_ordinary_def_denotes,
                                    OF AOT_sem_vars_denote])
     (metis AOT_sem_nocoder)

AOT_axiom "A-objects": ‹∃x (A!x & ∀F(x[F] ≡ φ{F}))›                        (53)
proof(rule AOT_model_axiomI)
  AOT_modally_strict {
    AOT_obtain κ where ‹κ↓ & □¬E!κ & ∀F (κ[F] ≡ φ{F})›
      using AOT_sem_A_objects[of _ φ]
      by (auto simp: AOT_sem_imp AOT_sem_box AOT_sem_forall AOT_sem_exists
                     AOT_sem_conj AOT_sem_not AOT_sem_dia AOT_sem_denotes
                     AOT_sem_equiv) blast
    AOT_thus ‹∃x (A!x & ∀F(x[F] ≡ φ{F}))›
      unfolding AOT_sem_exists
      by (auto intro!: exI[where x=κ]
               simp: AOT_sem_lambda_beta[OF AOT_sem_abstract_def_denotes]
                     AOT_sem_box AOT_sem_dia AOT_sem_not AOT_sem_denotes
                     AOT_var_of_term_inverse AOT_sem_conj
                     AOT_sem_equiv AOT_sem_forall AOT_sem_abstract)
  }
qed

AOT_theorem universal_closure:
  assumes ‹for arbitrary α: φ{α} ∈ Λ□›
  shows ‹∀α φ{α} ∈ Λ□›
  using assms
  by (metis AOT_term_of_var_cases AOT_model_axiom_def AOT_sem_denotes AOT_sem_forall)

AOT_theorem act_closure:
  assumes ‹φ ∈ Λ□›
  shows ‹𝒜φ ∈ Λ□›
  using assms by (simp add: AOT_model_axiom_def AOT_sem_act)

AOT_theorem nec_closure:
  assumes ‹φ ∈ Λ□›
  shows ‹□φ ∈ Λ□›
  using assms by (simp add: AOT_model_axiom_def AOT_sem_box)

AOT_theorem universal_closure_act:
  assumes ‹for arbitrary α: φ{α} ∈ Λ›
  shows ‹∀α φ{α} ∈ Λ›
  using assms
  by (metis AOT_term_of_var_cases AOT_model_act_axiom_def AOT_sem_denotes
            AOT_sem_forall)

text‹The following are not part of PLM and only hold in the extended models.
     They are a generalization of the predecessor axiom.›
context AOT_ExtendedModel
begin
AOT_axiom indistinguishable_ord_enc_all:
  ‹Π↓ & A!x & A!y & ∀F □([F]x ≡ [F]y) →
  ((∀G(∀z(O!z → □([G]z ≡ [Π]z)) → x[G])) ≡
    ∀G(∀z(O!z → □([G]z ≡ [Π]z)) → y[G]))›
  by (rule AOT_model_axiomI)
     (auto simp: AOT_sem_equiv AOT_sem_imp AOT_sem_conj
```

```
                 AOT_sem_indistinguishable_ord_enc_all)
AOT_axiom indistinguishable_ord_enc_ex:
  ‹Π↓ & A!x & A!y & ∀F □([F]x ≡ [F]y) →
  ((∃G(∀z(O!z → □([G]z ≡ [Π]z)) & x[G])) ≡
    ∃G(∀z(O!z → □([G]z ≡ [Π]z)) & y[G])))›
  by (rule AOT_model_axiomI)
     (auto simp: AOT_sem_equiv AOT_sem_imp AOT_sem_conj
                 AOT_sem_indistinguishable_ord_enc_ex)
end

(*<*)
end
(*>*)
```

A.7. The Deductive System PLM

```
(*<*)
theory AOT_PLM
  imports AOT_Axioms
begin
(*>*)

section‹The Deductive System PLM›
text‹\label{PLM: 9}›

(* constrain sledgehammer to the abstraction layer *)
unbundle AOT_no_atp

subsection‹Primitive Rule of PLM: Modus Ponens›
text‹\label{PLM: 9.1}›

AOT_theorem "modus-ponens":                                             (58)
  assumes ‹φ› and ‹φ → ψ›
  shows ‹ψ›
  (* NOTE: semantics needed *)
  using assms by (simp add: AOT_sem_imp)
lemmas MP = "modus-ponens"

subsection‹(Modally Strict) Proofs and Derivations›
text‹\label{PLM: 9.2}›

AOT_theorem "non-con-thm-thm":                                          (62)
  assumes ‹⊢□ φ›
  shows ‹⊢ φ›
  using assms by simp

AOT_theorem "vdash-properties:1[1]":                                    (63.1)
  assumes ‹φ ∈ Λ›
  shows ‹⊢ φ›
  (* NOTE: semantics needed *)
  using assms unfolding AOT_model_act_axiom_def by blast

text‹Convenience attribute for instantiating modally-fragile axioms.›
attribute_setup act_axiom_inst =
  ‹Scan.succeed (Thm.rule_attribute []
    (K (fn thm => thm RS @{thm "vdash-properties:1[1]"})))›
  "Instantiate modally fragile axiom as modally fragile theorem."

AOT_theorem "vdash-properties:1[2]":                                    (63.1)
  assumes ‹φ ∈ Λ□›
  shows ‹⊢□ φ›
  (* NOTE: semantics needed *)
  using assms unfolding AOT_model_axiom_def by blast

text‹Convenience attribute for instantiating modally-strict axioms.›
attribute_setup axiom_inst =
  ‹Scan.succeed (Thm.rule_attribute []
    (K (fn thm => thm RS @{thm "vdash-properties:1[2]"})))›
  "Instantiate axiom as theorem."

text‹Convenience methods and theorem sets for applying "cqt:2".›
method cqt_2_lambda_inst_prover =
  (fast intro: AOT_instance_of_cqt_2_intro)
method "cqt:2[lambda]" =
  (rule "cqt:2[lambda]"[axiom_inst]; cqt_2_lambda_inst_prover)
lemmas "cqt:2" =                                                        (39.2)
  "cqt:2[const_var]"[axiom_inst] "cqt:2[lambda]"[axiom_inst]
```

```
  AOT_instance_of_cqt_2_intro
method "cqt:2" = (safe intro!: "cqt:2")

AOT_theorem "vdash-properties:3":                                    (63.3)
  assumes ‹⊢□ φ›
  shows ‹Γ ⊢ φ›
  using assms by blast

AOT_theorem "vdash-properties:5":                                    (63.5)
  assumes ‹Γ₁ ⊢ φ› and ‹Γ₂ ⊢ φ → ψ›
  shows ‹Γ₁, Γ₂ ⊢ ψ›
  using MP assms by blast

AOT_theorem "vdash-properties:6":                                    (63.6)
  assumes ‹φ› and ‹φ → ψ›
  shows ‹ψ›
  using MP assms by blast

AOT_theorem "vdash-properties:8":                                    (63.8)
  assumes ‹Γ ⊢ φ› and ‹φ ⊢ ψ›
  shows ‹Γ ⊢ ψ›
  using assms by argo

AOT_theorem "vdash-properties:9":                                    (63.9)
  assumes ‹φ›
  shows ‹ψ → φ›
  using MP "pl:1"[axiom_inst] assms by blast

AOT_theorem "vdash-properties:10":                                   (63.10)
  assumes ‹φ → ψ› and ‹φ›
  shows ‹ψ›
  using MP assms by blast
lemmas "→E" = "vdash-properties:10"

subsection‹Two Fundamental Metarules: GEN and RN›
text‹\label{PLM: 9.3}›

AOT_theorem "rule-gen":                                              (66)
  assumes ‹for arbitrary α: φ{α}›
  shows ‹∀α φ{α}›
  (* NOTE: semantics needed *)
  using assms by (metis AOT_var_of_term_inverse AOT_sem_denotes AOT_sem_forall)
lemmas GEN = "rule-gen"

AOT_theorem "RN[prem]":                                              (68)
  assumes ‹Γ ⊢□ φ›
  shows ‹□Γ ⊢□ □φ›
  by (meson AOT_sem_box assms image_iff) (* NOTE: semantics needed *)
AOT_theorem RN:                                                      (68)
  assumes ‹⊢□ φ›
  shows ‹□φ›
  using "RN[prem]" assms by blast

subsection‹The Inferential Role of Definitions›
text‹\label{PLM: 9.4}›

AOT_axiom "df-rules-formulas[1]":                                    (72)
  assumes ‹φ ≡df ψ›
  shows ‹φ → ψ›
  (* NOTE: semantics needed *)
  using assms
  by (auto simp: assms AOT_model_axiomI AOT_model_equiv_def AOT_sem_imp)
AOT_axiom "df-rules-formulas[2]":                                    (72)
```

```
  assumes ‹φ ≡df ψ›
  shows ‹ψ → φ›
  (* NOTE: semantics needed *)
  using assms
  by (auto simp: AOT_model_axiomI AOT_model_equiv_def AOT_sem_imp)
(* NOTE: for convenience also state the above as regular theorems *)
AOT_theorem "df-rules-formulas[3]":                                    (72)
  assumes ‹φ ≡df ψ›
  shows ‹φ → ψ›
  using "df-rules-formulas[1]"[axiom_inst, OF assms].
AOT_theorem "df-rules-formulas[4]":                                    (72)
  assumes ‹φ ≡df ψ›
  shows ‹ψ → φ›
  using "df-rules-formulas[2]"[axiom_inst, OF assms].


AOT_axiom "df-rules-terms[1]":                                         (73)
  assumes ‹τ{α1...αn} =df σ{α1...αn}›
  shows ‹(σ{τ1...τn}↓ → τ{τ1...τn} = σ{τ1...τn}) &
         (¬σ{τ1...τn}↓ → ¬τ{τ1...τn}↓)›
  (* NOTE: semantics needed *)
  using assms
  by (simp add: AOT_model_axiomI AOT_sem_conj AOT_sem_imp AOT_sem_eq
                AOT_sem_not AOT_sem_denotes AOT_model_id_def)
AOT_axiom "df-rules-terms[2]":                                         (73)
  assumes ‹τ =df σ›
  shows ‹(σ↓ → τ = σ) & (¬σ↓ → ¬τ↓)›
  by (metis "df-rules-terms[1]" case_unit_Unity assms)
(* NOTE: for convenience also state the above as regular theorems *)
AOT_theorem "df-rules-terms[3]":                                       (73)
  assumes ‹τ{α1...αn} =df σ{α1...αn}›
  shows ‹(σ{τ1...τn}↓ → τ{τ1...τn} = σ{τ1...τn}) &
         (¬σ{τ1...τn}↓ → ¬τ{τ1...τn}↓)›
  using "df-rules-terms[1]"[axiom_inst, OF assms].
AOT_theorem "df-rules-terms[4]":                                       (73)
  assumes ‹τ =df σ›
  shows ‹(σ↓ → τ = σ) & (¬σ↓ → ¬τ↓)›
  using "df-rules-terms[2]"[axiom_inst, OF assms].

subsection‹The Theory of Negations and Conditionals›
text‹\label{PLM: 9.5}›

AOT_theorem "if-p-then-p": ‹φ → φ›                                     (74)
  by (meson "pl:1"[axiom_inst] "pl:2"[axiom_inst] MP)

AOT_theorem "deduction-theorem":                                       (75)
  assumes ‹φ ⊢ ψ›
  shows ‹φ → ψ›
  (* NOTE: semantics needed *)
  using assms by (simp add: AOT_sem_imp)
lemmas CP = "deduction-theorem"
lemmas "→I" = "deduction-theorem"

AOT_theorem "ded-thm-cor:1":                                           (76.1)
  assumes ‹Γ1 ⊢ φ → ψ› and ‹Γ2 ⊢ ψ → χ›
  shows ‹Γ1, Γ2 ⊢ φ → χ›
  using "→E" "→I" assms by blast
AOT_theorem "ded-thm-cor:2":                                           (76.2)
  assumes ‹Γ1 ⊢ φ → (ψ → χ)› and ‹Γ2 ⊢ ψ›
  shows ‹Γ1, Γ2 ⊢ φ → χ›
  using "→E" "→I" assms by blast

AOT_theorem "ded-thm-cor:3":                                           (76.3)
```

```
  assumes ‹φ → ψ› and ‹ψ → χ›
  shows ‹φ → χ›
  using "→E" "→I" assms by blast
declare "ded-thm-cor:3"[trans]
AOT_theorem "ded-thm-cor:4":                                                    (76.4)
  assumes ‹φ → (ψ → χ)› and ‹ψ›
  shows ‹φ → χ›
  using "→E" "→I" assms by blast

lemmas "Hypothetical Syllogism" = "ded-thm-cor:3"

AOT_theorem "useful-tautologies:1": ‹¬¬φ → φ›                                   (77.1)
  by (metis "pl:3"[axiom_inst] "→I" "Hypothetical Syllogism")
AOT_theorem "useful-tautologies:2": ‹φ → ¬¬φ›                                   (77.2)
  by (metis "pl:3"[axiom_inst] "→I" "ded-thm-cor:4")
AOT_theorem "useful-tautologies:3": ‹¬φ → (φ → ψ)›                              (77.3)
  by (meson "ded-thm-cor:4" "pl:3"[axiom_inst] "→I")
AOT_theorem "useful-tautologies:4": ‹(¬ψ → ¬φ) → (φ → ψ)›                       (77.4)
  by (meson "pl:3"[axiom_inst] "Hypothetical Syllogism" "→I")
AOT_theorem "useful-tautologies:5": ‹(φ → ψ) → (¬ψ → ¬φ)›                       (77.5)
  by (metis "useful-tautologies:4" "Hypothetical Syllogism" "→I")

AOT_theorem "useful-tautologies:6": ‹(φ → ¬ψ) → (ψ → ¬φ)›                       (77.6)
  by (metis "→I" MP "useful-tautologies:4")

AOT_theorem "useful-tautologies:7": ‹(¬φ → ψ) → (¬ψ → φ)›                       (77.7)
  by (metis "→I" MP "useful-tautologies:3" "useful-tautologies:5")

AOT_theorem "useful-tautologies:8": ‹φ → (¬ψ → ¬(φ → ψ))›                       (77.8)
  by (metis "→I" MP "useful-tautologies:5")

AOT_theorem "useful-tautologies:9": ‹(φ → ψ) → ((¬φ → ψ) → ψ)›                  (77.9)
  by (metis "→I" MP "useful-tautologies:6")

AOT_theorem "useful-tautologies:10": ‹(φ → ¬ψ) → ((φ → ψ) → ¬φ)›                (77.10)
  by (metis "→I" MP "pl:3"[axiom_inst])

AOT_theorem "dn-i-e:1":                                                         (78.1)
  assumes ‹φ›
  shows ‹¬¬φ›
  using MP "useful-tautologies:2" assms by blast
lemmas "¬¬I" = "dn-i-e:1"
AOT_theorem "dn-i-e:2":                                                         (78.2)
  assumes ‹¬¬φ›
  shows ‹φ›
  using MP "useful-tautologies:1" assms by blast
lemmas "¬¬E" = "dn-i-e:2"

AOT_theorem "modus-tollens:1":                                                  (79.1)
  assumes ‹φ → ψ› and ‹¬ψ›
  shows ‹¬φ›
  using MP "useful-tautologies:5" assms by blast
AOT_theorem "modus-tollens:2":                                                  (79.2)
  assumes ‹φ → ¬ψ› and ‹ψ›
  shows ‹¬φ›
  using "¬¬I" "modus-tollens:1" assms by blast
lemmas MT = "modus-tollens:1" "modus-tollens:2"

AOT_theorem "contraposition:1[1]":                                              (80.1)
  assumes ‹φ → ψ›
  shows ‹¬ψ → ¬φ›
  using "→I" MT(1) assms by blast
AOT_theorem "contraposition:1[2]":                                              (80.1)
```

```
  assumes ‹¬ψ → ¬φ›
  shows ‹φ → ψ›
  using "→I" "¬¬E" MT(2) assms by blast

AOT_theorem "contraposition:2":                                    (80.2)
  assumes ‹φ → ¬ψ›
  shows ‹ψ → ¬φ›
  using "→I" MT(2) assms by blast

AOT_theorem "reductio-aa:1":                                       (81.1)
  assumes ‹¬φ ⊢ ¬ψ› and ‹¬φ ⊢ ψ›
  shows ‹φ›
  using "→I" "¬¬E" MT(2) assms by blast
AOT_theorem "reductio-aa:2":                                       (81.2)
  assumes ‹φ ⊢ ¬ψ› and ‹φ ⊢ ψ›
  shows ‹¬φ›
  using "reductio-aa:1" assms by blast
lemmas "RAA" = "reductio-aa:1" "reductio-aa:2"

AOT_theorem "exc-mid": ‹φ ∨ ¬φ›                                    (83)
  using "df-rules-formulas[4]" "if-p-then-p" MP
        "conventions:2" by blast

AOT_theorem "non-contradiction": ‹¬(φ & ¬φ)›                       (84)
  using "df-rules-formulas[3]" MT(2) "useful-tautologies:2"
        "conventions:1" by blast

AOT_theorem "con-dis-taut:1": ‹(φ & ψ) → φ›                        (85.1)
  by (meson "→I" "df-rules-formulas[3]" MP RAA(1) "conventions:1")
AOT_theorem "con-dis-taut:2": ‹(φ & ψ) → ψ›                        (85.2)
  by (metis "→I" "df-rules-formulas[3]" MT(2) RAA(2)
            "¬¬E" "conventions:1")
lemmas "Conjunction Simplification" = "con-dis-taut:1" "con-dis-taut:2"

AOT_theorem "con-dis-taut:3": ‹φ → (φ ∨ ψ)›                        (85.3)
  by (meson "contraposition:1[2]" "df-rules-formulas[4]"
            MP "→I" "conventions:2")
AOT_theorem "con-dis-taut:4": ‹ψ → (φ ∨ ψ)›                        (85.4)
  using "Hypothetical Syllogism" "df-rules-formulas[4]"
        "pl:1"[axiom_inst] "conventions:2" by blast
lemmas "Disjunction Addition" = "con-dis-taut:3" "con-dis-taut:4"

AOT_theorem "con-dis-taut:5": ‹φ → (ψ → (φ & ψ))›                  (85.5)
  by (metis "contraposition:2" "Hypothetical Syllogism" "→I"
            "df-rules-formulas[4]" "conventions:1")
lemmas Adjunction = "con-dis-taut:5"

AOT_theorem "con-dis-taut:6": ‹(φ & φ) ≡ φ›                        (85.6)
  by (metis Adjunction "→I" "df-rules-formulas[4]" MP
            "Conjunction Simplification"(1) "conventions:3")
lemmas "Idempotence of &" = "con-dis-taut:6"

AOT_theorem "con-dis-taut:7": ‹(φ ∨ φ) ≡ φ›                        (85.7)
proof -
  {
    AOT_assume ‹φ ∨ φ›
    AOT_hence ‹¬φ → φ›
      using "conventions:2"[THEN "df-rules-formulas[3]"] MP by blast
    AOT_hence ‹φ› using "if-p-then-p" RAA(1) MP by blast
  }
  moreover {
    AOT_assume ‹φ›
    AOT_hence ‹φ ∨ φ› using "Disjunction Addition"(1) MP by blast
```

```
  }
  ultimately AOT_show ‹(φ ∨ φ) ≡ φ›
    using "conventions:3"[THEN "df-rules-formulas[4]"] MP
    by (metis Adjunction "→I")
qed
lemmas "Idempotence of ∨" = "con-dis-taut:7"


AOT_theorem "con-dis-i-e:1":                                    (86.1)
  assumes ‹φ› and ‹ψ›
  shows ‹φ & ψ›
  using Adjunction MP assms by blast
lemmas "&I" = "con-dis-i-e:1"

AOT_theorem "con-dis-i-e:2:a":                                  (86.2.a)
  assumes ‹φ & ψ›
  shows ‹φ›
  using "Conjunction Simplification"(1) MP assms by blast
AOT_theorem "con-dis-i-e:2:b":                                  (86.2.b)
  assumes ‹φ & ψ›
  shows ‹ψ›
  using "Conjunction Simplification"(2) MP assms by blast
lemmas "&E" = "con-dis-i-e:2:a" "con-dis-i-e:2:b"

AOT_theorem "con-dis-i-e:3:a":                                  (86.3.a)
  assumes ‹φ›
  shows ‹φ ∨ ψ›
  using "Disjunction Addition"(1) MP assms by blast
AOT_theorem "con-dis-i-e:3:b":                                  (86.3.b)
  assumes ‹ψ›
  shows ‹φ ∨ ψ›
  using "Disjunction Addition"(2) MP assms by blast
AOT_theorem "con-dis-i-e:3:c":                                  (86.3.c)
  assumes ‹φ ∨ ψ› and ‹φ → χ› and ‹ψ → Θ›
  shows ‹χ ∨ Θ›
  by (metis "con-dis-i-e:3:a" "Disjunction Addition"(2)
            "df-rules-formulas[3]" MT(1) RAA(1)
            "conventions:2" assms)
lemmas "∨I" = "con-dis-i-e:3:a" "con-dis-i-e:3:b" "con-dis-i-e:3:c"

AOT_theorem "con-dis-i-e:4:a":                                  (86.4.a)
  assumes ‹φ ∨ ψ› and ‹φ → χ› and ‹ψ → χ›
  shows ‹χ›
  by (metis MP RAA(2) "df-rules-formulas[3]" "conventions:2" assms)
AOT_theorem "con-dis-i-e:4:b":                                  (86.4.b)
  assumes ‹φ ∨ ψ› and ‹¬φ›
  shows ‹ψ›
  using "con-dis-i-e:4:a" RAA(1) "→I" assms by blast
AOT_theorem "con-dis-i-e:4:c":                                  (86.4.c)
  assumes ‹φ ∨ ψ› and ‹¬ψ›
  shows ‹φ›
  using "con-dis-i-e:4:a" RAA(1) "→I" assms by blast
lemmas "∨E" = "con-dis-i-e:4:a" "con-dis-i-e:4:b" "con-dis-i-e:4:c"

AOT_theorem "raa-cor:1":                                        (87.1)
  assumes ‹¬φ ⊢ ψ & ¬ψ›
  shows ‹φ›
  using "&E" "∨E"(3) "∨I"(2) RAA(2) assms by blast
AOT_theorem "raa-cor:2":                                        (87.2)
  assumes ‹φ ⊢ ψ & ¬ψ›
  shows ‹¬φ›
  using "raa-cor:1" assms by blast
AOT_theorem "raa-cor:3":                                        (87.3)
```

```
  assumes ‹φ› and ‹¬ψ ⊢ ¬φ›
  shows ‹ψ›
  using RAA assms by blast
AOT_theorem "raa-cor:4":                                              (87.4)
  assumes ‹¬φ› and ‹¬ψ ⊢ φ›
  shows ‹ψ›
  using RAA assms by blast
AOT_theorem "raa-cor:5":                                              (87.5)
  assumes ‹φ› and ‹ψ ⊢ ¬φ›
  shows ‹¬ψ›
  using RAA assms by blast
AOT_theorem "raa-cor:6":                                              (87.6)
  assumes ‹¬φ› and ‹ψ ⊢ φ›
  shows ‹¬ψ›
  using RAA assms by blast

AOT_theorem "oth-class-taut:1:a": ‹(φ → ψ) ≡ ¬(φ & ¬ψ)›               (88.1.a)
  by (rule "conventions:3"[THEN "df-rules-formulas[4]", THEN "→E"])
     (metis "&E" "&I" "raa-cor:3" "→I" MP)
AOT_theorem "oth-class-taut:1:b": ‹¬(φ → ψ) ≡ (φ & ¬ψ)›               (88.1.b)
  by (rule "conventions:3"[THEN "df-rules-formulas[4]", THEN "→E"])
     (metis "&E" "&I" "raa-cor:3" "→I" MP)
AOT_theorem "oth-class-taut:1:c": ‹(φ → ψ) ≡ (¬φ ∨ ψ)›                (88.1.c)
  by (rule "conventions:3"[THEN "df-rules-formulas[4]", THEN "→E"])
     (metis "&I" "∨I"(1, 2) "∨E"(3) "→I" MP "raa-cor:1")

AOT_theorem "oth-class-taut:2:a": ‹(φ & ψ) ≡ (ψ & φ)›                 (88.2.a)
  by (rule "conventions:3"[THEN "df-rules-formulas[4]", THEN "→E"])
     (meson "&I" "&E" "→I")
lemmas "Commutativity of &" = "oth-class-taut:2:a"
AOT_theorem "oth-class-taut:2:b": ‹(φ & (ψ & χ)) ≡ ((φ & ψ) & χ)›     (88.2.b)
  by (rule "conventions:3"[THEN "df-rules-formulas[4]", THEN "→E"])
     (metis "&I" "&E" "→I")
lemmas "Associativity of &" = "oth-class-taut:2:b"
AOT_theorem "oth-class-taut:2:c": ‹(φ ∨ ψ) ≡ (ψ ∨ φ)›                 (88.2.c)
  by (rule "conventions:3"[THEN "df-rules-formulas[4]", THEN "→E"])
     (metis "&I" "∨I"(1, 2) "∨E"(1) "→I")
lemmas "Commutativity of ∨" = "oth-class-taut:2:c"
AOT_theorem "oth-class-taut:2:d": ‹(φ ∨ (ψ ∨ χ)) ≡ ((φ ∨ ψ) ∨ χ)›     (88.2.d)
  by (rule "conventions:3"[THEN "df-rules-formulas[4]", THEN "→E"])
     (metis "&I" "∨I"(1, 2) "∨E"(1) "→I")
lemmas "Associativity of ∨" = "oth-class-taut:2:d"
AOT_theorem "oth-class-taut:2:e": ‹(φ ≡ ψ) ≡ (ψ ≡ φ)›                 (88.2.e)
  by (rule "conventions:3"[THEN "df-rules-formulas[4]", THEN "→E"]; rule "&I";
      metis "&I" "df-rules-formulas[4]" "conventions:3" "&E"
            "Hypothetical Syllogism" "→I" "df-rules-formulas[3]")
lemmas "Commutativity of ≡" = "oth-class-taut:2:e"
AOT_theorem "oth-class-taut:2:f": ‹(φ ≡ (ψ ≡ χ)) ≡ ((φ ≡ ψ) ≡ χ)›     (88.2.f)
  using "conventions:3"[THEN "df-rules-formulas[4]"]
        "conventions:3"[THEN "df-rules-formulas[3]"]
        "→I" "→E" "&E" "&I"
  by metis
lemmas "Associativity of ≡" = "oth-class-taut:2:f"

AOT_theorem "oth-class-taut:3:a": ‹φ ≡ φ›                             (88.3.a)
  using "&I" "vdash-properties:6" "if-p-then-p"
        "df-rules-formulas[4]" "conventions:3" by blast
AOT_theorem "oth-class-taut:3:b": ‹φ ≡ ¬¬φ›                           (88.3.b)
  using "&I" "useful-tautologies:1" "useful-tautologies:2" "→E"
        "df-rules-formulas[4]" "conventions:3" by blast
AOT_theorem "oth-class-taut:3:c": ‹¬(φ ≡ ¬φ)›                         (88.3.c)
  by (metis "&E" "→E" RAA "df-rules-formulas[3]" "conventions:3")
```

```
AOT_theorem "oth-class-taut:4:a": ‹(φ → ψ) → ((ψ → χ) → (φ → χ))›          (88.4.a)
  by (metis "→E" "→I")
AOT_theorem "oth-class-taut:4:b": ‹(φ ≡ ψ) ≡ (¬φ ≡ ¬ψ)›          (88.4.b)
  using "conventions:3"[THEN "df-rules-formulas[4]"]
        "conventions:3"[THEN "df-rules-formulas[3]"]
        "→I" "→E" "&E" "&I" RAA by metis
AOT_theorem "oth-class-taut:4:c": ‹(φ ≡ ψ) → ((φ → χ) ≡ (ψ → χ))›          (88.4.c)
  using "conventions:3"[THEN "df-rules-formulas[4]"]
        "conventions:3"[THEN "df-rules-formulas[3]"]
        "→I" "→E" "&E" "&I" by metis
AOT_theorem "oth-class-taut:4:d": ‹(φ ≡ ψ) → ((χ → φ) ≡ (χ → ψ))›          (88.4.d)
  using "conventions:3"[THEN "df-rules-formulas[4]"]
        "conventions:3"[THEN "df-rules-formulas[3]"]
        "→I" "→E" "&E" "&I" by metis
AOT_theorem "oth-class-taut:4:e": ‹(φ ≡ ψ) → ((φ & χ) ≡ (ψ & χ))›          (88.4.e)
  using "conventions:3"[THEN "df-rules-formulas[4]"]
        "conventions:3"[THEN "df-rules-formulas[3]"]
        "→I" "→E" "&E" "&I" by metis
AOT_theorem "oth-class-taut:4:f": ‹(φ ≡ ψ) → ((χ & φ) ≡ (χ & ψ))›          (88.4.f)
  using "conventions:3"[THEN "df-rules-formulas[4]"]
        "conventions:3"[THEN "df-rules-formulas[3]"]
        "→I" "→E" "&E" "&I" by metis
AOT_theorem "oth-class-taut:4:g": ‹(φ ≡ ψ) ≡ ((φ & ψ) ∨ (¬φ & ¬ψ))›          (88.4.g)
proof(safe intro!: "conventions:3"[THEN "df-rules-formulas[4]", THEN "→E"]
                   "&I" "→I"
           dest!: "conventions:3"[THEN "df-rules-formulas[3]", THEN "→E"])
  AOT_show ‹φ & ψ ∨ (¬φ & ¬ψ)› if ‹(φ → ψ) & (ψ → φ)›
    using "&E" "∨I" "→E" "&I" "raa-cor:1" "→I" "∨E" that by metis
next
  AOT_show ‹ψ› if ‹φ & ψ ∨ (¬φ & ¬ψ)› and ‹φ›
    using that "∨E" "&E" "raa-cor:3" by blast
next
  AOT_show ‹φ› if ‹φ & ψ ∨ (¬φ & ¬ψ)› and ‹ψ›
    using that "∨E" "&E" "raa-cor:3" by blast
qed
AOT_theorem "oth-class-taut:4:h": ‹¬(φ ≡ ψ) ≡ ((φ & ¬ψ) ∨ (¬φ & ψ))›          (88.4.h)
proof (safe intro!: "conventions:3"[THEN "df-rules-formulas[4]", THEN "→E"]
                    "&I" "→I")
  AOT_show ‹φ & ¬ψ ∨ (¬φ & ψ)› if ‹¬(φ ≡ ψ)›
    by (metis that "&I" "∨I"(1, 2) "→I" MT(1) "df-rules-formulas[4]"
              "raa-cor:3" "conventions:3")
next
  AOT_show ‹¬(φ ≡ ψ)› if ‹φ & ¬ψ ∨ (¬φ & ψ)›
    by (metis that "&E" "∨E"(2) "→E" "df-rules-formulas[3]"
              "raa-cor:3" "conventions:3")
qed
AOT_theorem "oth-class-taut:5:a": ‹(φ & ψ) ≡ ¬(¬φ ∨ ¬ψ)›          (88.5.a)
  using "conventions:3"[THEN "df-rules-formulas[4]"]
        "→I" "→E" "&E" "&I" "∨I" "∨E" RAA by metis
AOT_theorem "oth-class-taut:5:b": ‹(φ ∨ ψ) ≡ ¬(¬φ & ¬ψ)›          (88.5.b)
  using "conventions:3"[THEN "df-rules-formulas[4]"]
        "→I" "→E" "&E" "&I" "∨I" "∨E" RAA by metis
AOT_theorem "oth-class-taut:5:c": ‹¬(φ & ψ) ≡ (¬φ ∨ ¬ψ)›          (88.5.c)
  using "conventions:3"[THEN "df-rules-formulas[4]"]
        "→I" "→E" "&E" "&I" "∨I" "∨E" RAA by metis
AOT_theorem "oth-class-taut:5:d": ‹¬(φ ∨ ψ) ≡ (¬φ & ¬ψ)›          (88.5.d)
  using "conventions:3"[THEN "df-rules-formulas[4]"]
        "→I" "→E" "&E" "&I" "∨I" "∨E" RAA by metis

lemmas DeMorgan = "oth-class-taut:5:c" "oth-class-taut:5:d"

AOT_theorem "oth-class-taut:6:a":          (88.6.a)
  ‹(φ & (ψ ∨ χ)) ≡ ((φ & ψ) ∨ (φ & χ))›
```

```
  using "conventions:3"[THEN "df-rules-formulas[4]"]
        "→I" "→E" "&E" "&I" "∨I" "∨E" RAA by metis
AOT_theorem "oth-class-taut:6:b":                                          (88.6.b)
  ‹(φ ∨ (ψ & χ)) ≡ ((φ ∨ ψ) & (φ ∨ χ))›
  using "conventions:3"[THEN "df-rules-formulas[4]"]
        "→I" "→E" "&E" "&I" "∨I" "∨E" RAA by metis

AOT_theorem "oth-class-taut:7:a": ‹((φ & ψ) → χ) → (φ → (ψ → χ))›          (88.7.a)
  by (metis "&I" "→E" "→I")
lemmas Exportation = "oth-class-taut:7:a"
AOT_theorem "oth-class-taut:7:b": ‹(φ → (ψ →χ)) → ((φ & ψ) → χ)›           (88.7.b)
  by (metis "&E" "→E" "→I")
lemmas Importation = "oth-class-taut:7:b"

AOT_theorem "oth-class-taut:8:a":                                          (88.8.a)
  ‹(φ → (ψ → χ)) ≡ (ψ → (φ → χ))›
  using "conventions:3"[THEN "df-rules-formulas[4]"] "→I" "→E" "&E" "&I"
  by metis
lemmas Permutation = "oth-class-taut:8:a"
AOT_theorem "oth-class-taut:8:b":                                          (88.8.b)
  ‹(φ → ψ) → ((φ → χ) → (φ → (ψ & χ)))›
  by (metis "&I" "→E" "→I")
lemmas Composition = "oth-class-taut:8:b"
AOT_theorem "oth-class-taut:8:c":                                          (88.8.c)
  ‹(φ → χ) → ((ψ → χ) → ((φ ∨ ψ) → χ))›
  by (metis "∨E"(2) "→E" "→I" RAA(1))
AOT_theorem "oth-class-taut:8:d":                                          (88.8.d)
  ‹((φ → ψ) & (χ → Θ)) → ((φ & χ) → (ψ & Θ))›
  by (metis "&E" "&I" "→E" "→I")
lemmas "Double Composition" = "oth-class-taut:8:d"
AOT_theorem "oth-class-taut:8:e":                                          (88.8.e)
  ‹((φ & ψ) ≡ (φ & χ)) ≡ (φ → (ψ ≡ χ))›
  by (metis "conventions:3"[THEN "df-rules-formulas[4]"]
            "conventions:3"[THEN "df-rules-formulas[3]"]
            "→I" "→E" "&E" "&I")
AOT_theorem "oth-class-taut:8:f":                                          (88.8.f)
  ‹((φ & ψ) ≡ (χ & ψ)) ≡ (ψ → (φ ≡ χ))›
  by (metis "conventions:3"[THEN "df-rules-formulas[4]"]
            "conventions:3"[THEN "df-rules-formulas[3]"]
            "→I" "→E" "&E" "&I")
AOT_theorem "oth-class-taut:8:g":                                          (88.8.g)
  ‹(ψ ≡ χ) → ((φ ∨ ψ) ≡ (φ ∨ χ))›
  by (metis "conventions:3"[THEN "df-rules-formulas[4]"]
            "conventions:3"[THEN "df-rules-formulas[3]"]
            "→I" "→E" "&E" "&I" "∨I" "∨E"(1))
AOT_theorem "oth-class-taut:8:h":                                          (88.8.h)
  ‹(ψ ≡ χ) → ((ψ ∨ φ) ≡ (χ ∨ φ))›
  by (metis "conventions:3"[THEN "df-rules-formulas[4]"]
            "conventions:3"[THEN "df-rules-formulas[3]"]
            "→I" "→E" "&E" "&I" "∨I" "∨E"(1))
AOT_theorem "oth-class-taut:8:i":                                          (88.8.i)
  ‹(φ ≡ (ψ & χ)) → (ψ → (φ ≡ χ))›
  by (metis "conventions:3"[THEN "df-rules-formulas[4]"]
            "conventions:3"[THEN "df-rules-formulas[3]"]
            "→I" "→E" "&E" "&I")

AOT_theorem "intro-elim:1":                                                (89.1)
  assumes ‹φ ∨ ψ› and ‹φ ≡ χ› and ‹ψ ≡ Θ›
  shows ‹χ ∨ Θ›
  by (metis assms "∨I"(1, 2) "∨E"(1) "→I" "→E" "&E"(1)
            "conventions:3"[THEN "df-rules-formulas[3]"])

AOT_theorem "intro-elim:2":                                                (89.2)
```

```
  assumes ‹φ → ψ› and ‹ψ → φ›
  shows ‹φ ≡ ψ›
  by (meson "&I" "conventions:3" "df-rules-formulas[4]" MP assms)
lemmas "≡I" = "intro-elim:2"

AOT_theorem "intro-elim:3:a":                                              (89.3.a)
  assumes ‹φ ≡ ψ› and ‹φ›
  shows ‹ψ›
  by (metis "∨I"(1) "→I" "∨E"(1) "intro-elim:1" assms)
AOT_theorem "intro-elim:3:b":                                              (89.3.b)
  assumes ‹φ ≡ ψ› and ‹ψ›
  shows ‹φ›
  using "intro-elim:3:a" "Commutativity of ≡" assms by blast
AOT_theorem "intro-elim:3:c":                                              (89.3.c)
  assumes ‹φ ≡ ψ› and ‹¬φ›
  shows ‹¬ψ›
  using "intro-elim:3:b" "raa-cor:3" assms by blast
AOT_theorem "intro-elim:3:d":                                              (89.3.d)
  assumes ‹φ ≡ ψ› and ‹¬ψ›
  shows ‹¬φ›
  using "intro-elim:3:a" "raa-cor:3" assms by blast
AOT_theorem "intro-elim:3:e":                                              (89.3.e)
  assumes ‹φ ≡ ψ› and ‹ψ ≡ χ›
  shows ‹φ ≡ χ›
  by (metis "≡I" "→I" "intro-elim:3:a" "intro-elim:3:b" assms)
declare "intro-elim:3:e"[trans]
AOT_theorem "intro-elim:3:f":                                              (89.3.f)
  assumes ‹φ ≡ ψ› and ‹φ ≡ χ›
  shows ‹χ ≡ ψ›
  by (metis "≡I" "→I" "intro-elim:3:a" "intro-elim:3:b" assms)
lemmas "≡E" = "intro-elim:3:a" "intro-elim:3:b" "intro-elim:3:c"
              "intro-elim:3:d" "intro-elim:3:e" "intro-elim:3:f"

declare "Commutativity of ≡"[THEN "≡E"(1), sym]

AOT_theorem "rule-eq-df:1":                                                (90.1)
  assumes ‹φ ≡df ψ›
  shows ‹φ ≡ ψ›
  by (simp add: "≡I" "df-rules-formulas[3]" "df-rules-formulas[4]" assms)
lemmas "≡Df" = "rule-eq-df:1"
AOT_theorem "rule-eq-df:2":                                                (90.2)
  assumes ‹φ ≡df ψ› and ‹φ›
  shows ‹ψ›
  using "≡Df" "≡E"(1) assms by blast
lemmas "≡dfE" = "rule-eq-df:2"
AOT_theorem "rule-eq-df:3":                                                (90.3)
  assumes ‹φ ≡df ψ› and ‹ψ›
  shows ‹φ›
  using "≡Df" "≡E"(2) assms by blast
lemmas "≡dfI" = "rule-eq-df:3"

AOT_theorem  "df-simplify:1":                                              (91.1)
  assumes ‹φ ≡ (ψ & χ)› and ‹ψ›
  shows ‹φ ≡ χ›
  by (metis "&E"(2) "&I" "≡E"(1, 2) "≡I" "→I" assms)
(* Note: this is a slight variation from PLM *)
AOT_theorem  "df-simplify:2":                                              (91.2)
  assumes ‹φ ≡ (ψ & χ)› and ‹χ›
  shows ‹φ ≡ ψ›
  by (metis "&E"(1) "&I" "≡E"(1, 2) "≡I" "→I" assms)
lemmas "≡S" = "df-simplify:1"  "df-simplify:2"

subsection‹The Theory of Quantification›
```

```
text‹\label{PLM: 9.6}›

AOT_theorem "rule-ui:1":                                                          (93.1)
  assumes ‹∀α φ{α}› and ‹τ↓›
  shows ‹φ{τ}›
  using "→E" "cqt:1"[axiom_inst] assms by blast
AOT_theorem "rule-ui:2[const_var]":                                               (93.2)
  assumes ‹∀α φ{α}›
  shows ‹φ{β}›
  by (simp add: "rule-ui:1" "cqt:2[const_var]"[axiom_inst] assms)
AOT_theorem "rule-ui:2[lambda]":                                                  (93.2)
  assumes ‹∀F φ{F}› and ‹INSTANCE_OF_CQT_2(ψ)›
  shows ‹φ{[λν1...νn ψ{ν1...νn}]}›
  by (simp add: "rule-ui:1" "cqt:2[lambda]"[axiom_inst] assms)
AOT_theorem "rule-ui:3":                                                          (93.3)
  assumes ‹∀α φ{α}›
  shows ‹φ{α}›
  by (simp add: "rule-ui:2[const_var]" assms)
lemmas "∀E" = "rule-ui:1" "rule-ui:2[const_var]"
              "rule-ui:2[lambda]" "rule-ui:3"

AOT_theorem "cqt-orig:1[const_var]": ‹∀α φ{α} → φ{β}›                             (95.1)
  by (simp add: "∀E"(2) "→I")
AOT_theorem "cqt-orig:1[lambda]":                                                 (95.1)
  assumes ‹INSTANCE_OF_CQT_2(ψ)›
  shows ‹∀F φ{F} → φ{[λν1...νn ψ{ν1...νn}]}›
  by (simp add: "∀E"(3) "→I" assms)
AOT_theorem "cqt-orig:2": ‹∀α (φ → ψ{α}) → (φ → ∀α ψ{α})›                         (95.2)
  by (metis "→I" GEN "vdash-properties:6" "∀E"(4))
AOT_theorem "cqt-orig:3": ‹∀α φ{α} → φ{α}›                                        (95.3)
  using "cqt-orig:1[const_var]".

AOT_theorem universal:                                                            (96)
  assumes ‹for arbitrary β: φ{β}›
  shows ‹∀α φ{α}›
  using GEN assms .
lemmas "∀I" = universal

(* Generalized mechanism for ∀I followed by ∀E *)
ML‹
fun get_instantiated_allI ctxt varname thm = let
val trm = Thm.concl_of thm
val trm =
  case trm of (@{const Trueprop} $ (@{const AOT_model_valid_in} $ _ $ x)) => x
  | _ => raise Term.TERM ("Expected simple theorem.", [trm])
fun extractVars (Const (const_name‹AOT_term_of_var›, _) $ Var v) =
    (if fst (fst v) = fst varname then [Var v] else [])
  | extractVars (t1 $ t2) = extractVars t1 @ extractVars t2
  | extractVars (Abs (_, _, t)) = extractVars t
  | extractVars _ = []
val vars = extractVars trm
val vars = fold Term.add_vars vars []
val var = hd vars
val trmty =
  case (snd var) of (Type (type_name‹AOT_var›, [t])) => (t)
  | _ => raise Term.TYPE ("Expected variable type.", [snd var], [Var var])
val trm = Abs (Term.string_of_vname (fst var), trmty, Term.abstract_over (
      Const (const_name‹AOT_term_of_var›, Type ("fun", [snd var, trmty]))
       $ Var var, trm))
val trm = Thm.cterm_of (Context.proof_of ctxt) trm
val ty = hd (Term.add_tvars (Thm.prop_of @{thm "∀I"}) [])
val typ = Thm.ctyp_of (Context.proof_of ctxt) trmty
val allthm = Drule.instantiate_normalize (TVars.make [(ty, typ)], Vars.empty) @{thm "∀I"}
```

```
val phi = hd (Term.add_vars (Thm.prop_of allthm) [])
val allthm = Drule.instantiate_normalize (TVars.empty, Vars.make [(phi,trm)]) allthm
in
allthm
end
›

attribute_setup "∀I" =
  ‹Scan.lift (Scan.repeat1 Args.var) » (fn args => Thm.rule_attribute []
  (fn ctxt => fn thm => fold (fn arg => fn thm =>
    thm RS get_instantiated_allI ctxt arg thm) args thm))›
  "Quantify over a variable in a theorem using GEN."

attribute_setup "unvarify" =
  ‹Scan.lift (Scan.repeat1 Args.var) » (fn args => Thm.rule_attribute []
  (fn ctxt => fn thm =>
    let
      fun get_inst_allI arg thm = thm RS get_instantiated_allI ctxt arg thm
      val thm = fold get_inst_allI args thm
      val thm = fold (K (fn thm => thm RS @{thm "∀E"(1)})) args thm
    in
     thm
    end))›
  "Generalize a statement about variables to a statement about denoting terms."

(* Note: rereplace-lem does not apply to the embedding *)

AOT_theorem "cqt-basic:1": ‹∀α∀β φ{α,β} ≡ ∀β∀α φ{α,β}›                    (99.1)
  by (metis "≡I" "∀E"(2) "∀I" "→I")

AOT_theorem "cqt-basic:2":                                                (99.2)
  ‹∀α(φ{α} ≡ ψ{α}) ≡ (∀α(φ{α} → ψ{α}) & ∀α(ψ{α} → φ{α}))›
proof (rule "≡I"; rule "→I")
  AOT_assume ‹∀α(φ{α} ≡ ψ{α})›
  AOT_hence ‹φ{α} ≡ ψ{α}› for α using "∀E"(2) by blast
  AOT_hence ‹φ{α} → ψ{α}› and ‹ψ{α} → φ{α}› for α
    using "≡E"(1,2) "→I" by blast+
  AOT_thus ‹∀α(φ{α} → ψ{α}) & ∀α(ψ{α} → φ{α})›
    by (auto intro: "&I" "∀I")
next
  AOT_assume ‹∀α(φ{α} → ψ{α}) & ∀α(ψ{α} → φ{α})›
  AOT_hence ‹φ{α} → ψ{α}› and ‹ψ{α} → φ{α}› for α
    using "∀E"(2) "&E" by blast+
  AOT_hence ‹φ{α} ≡ ψ{α}› for α
    using "≡I" by blast
  AOT_thus ‹∀α(φ{α} ≡ ψ{α})› by (auto intro: "∀I")
qed

AOT_theorem "cqt-basic:3": ‹∀α(φ{α} ≡ ψ{α}) → (∀α φ{α} ≡ ∀α ψ{α})›      (99.3)
proof(rule "→I")
  AOT_assume ‹∀α(φ{α} ≡ ψ{α})›
  AOT_hence 1: ‹φ{α} ≡ ψ{α}› for α using "∀E"(2) by blast
  {
    AOT_assume ‹∀α φ{α}›
    AOT_hence ‹∀α ψ{α}› using 1 "∀I" "∀E"(4) "≡E" by metis
  }
  moreover {
    AOT_assume ‹∀α ψ{α}›
    AOT_hence ‹∀α φ{α}› using 1 "∀I" "∀E"(4) "≡E" by metis
  }
  ultimately AOT_show ‹∀α φ{α} ≡ ∀α ψ{α}›
    using "≡I" "→I" by auto
qed
```

```
AOT_theorem "cqt-basic:4": ‹∀α(φ{α} & ψ{α}) → (∀α φ{α} & ∀α ψ{α})›    (99.4)
proof(rule "→I")
  AOT_assume 0: ‹∀α(φ{α} & ψ{α})›
  AOT_have ‹φ{α}› and ‹ψ{α}› for α using "∀E"(2) 0 "&E" by blast+
  AOT_thus ‹∀α φ{α} & ∀α ψ{α}›
    by (auto intro: "∀I" "&I")
qed

AOT_theorem "cqt-basic:5": ‹(∀α₁...∀αₙ(φ{α₁...αₙ})) → φ{α₁...αₙ}›    (99.5)
  using "cqt-orig:3" by blast

AOT_theorem "cqt-basic:6": ‹∀α∀α φ{α} ≡ ∀α φ{α}›    (99.6)
  by (meson "≡I" "→I" GEN "cqt-orig:1[const_var]")

AOT_theorem "cqt-basic:7": ‹(φ → ∀α ψ{α}) ≡ ∀α(φ → ψ{α})›    (99.7)
  by (metis "→I" "vdash-properties:6" "rule-ui:3" "≡I" GEN)

AOT_theorem "cqt-basic:8": ‹(∀α φ{α} ∨ ∀α ψ{α}) → ∀α (φ{α} ∨ ψ{α})›    (99.8)
  by (simp add: "∨I"(3) "→I" GEN "cqt-orig:1[const_var]")

AOT_theorem "cqt-basic:9":    (99.9)
  ‹(∀α (φ{α} → ψ{α}) & ∀α (ψ{α} → χ{α})) → ∀α(φ{α} → χ{α})›
proof -
  {
    AOT_assume ‹∀α (φ{α} → ψ{α})›
    moreover AOT_assume ‹∀α (ψ{α} → χ{α})›
    ultimately AOT_have ‹φ{α} → ψ{α}› and ‹ψ{α} → χ{α}› for α
      using "∀E" by blast+
    AOT_hence ‹φ{α} → χ{α}› for α by (metis "→E" "→I")
    AOT_hence ‹∀α(φ{α} → χ{α})› using "∀I" by fast
  }
  thus ?thesis using "&I" "→I" "&E" by meson
qed

AOT_theorem "cqt-basic:10":    (99.10)
  ‹(∀α(φ{α} ≡ ψ{α}) & ∀α(ψ{α} ≡ χ{α})) → ∀α (φ{α} ≡ χ{α})›
proof(rule "→I"; rule "∀I")
  fix β
  AOT_assume ‹∀α(φ{α} ≡ ψ{α}) & ∀α(ψ{α} ≡ χ{α})›
  AOT_hence ‹φ{β} ≡ ψ{β}› and ‹ψ{β} ≡ χ{β}› using "&E" "∀E" by blast+
  AOT_thus ‹φ{β} ≡ χ{β}› using "≡I" "≡E" by blast
qed

AOT_theorem "cqt-basic:11": ‹∀α(φ{α} ≡ ψ{α}) ≡ ∀α (ψ{α} ≡ φ{α})›    (99.11)
proof (rule "≡I"; rule "→I")
  AOT_assume 0: ‹∀α(φ{α} ≡ ψ{α})›
  {
    fix α
    AOT_have ‹φ{α} ≡ ψ{α}› using 0 "∀E" by blast
    AOT_hence ‹ψ{α} ≡ φ{α}› using "≡I" "≡E" "→I" "→E" by metis
  }
  AOT_thus ‹∀α(ψ{α} ≡ φ{α})› using "∀I" by fast
next
  AOT_assume 0: ‹∀α(ψ{α} ≡ φ{α})›
  {
    fix α
    AOT_have ‹ψ{α} ≡ φ{α}› using 0 "∀E" by blast
    AOT_hence ‹φ{α} ≡ ψ{α}› using "≡I" "≡E" "→I" "→E" by metis
  }
  AOT_thus ‹∀α(φ{α} ≡ ψ{α})› using "∀I" by fast
qed
```

```
AOT_theorem "cqt-basic:12": ‹∀α φ{α} → ∀α (ψ{α} → φ{α})›                    (99.12)
  by (simp add: "∀E"(2) "→I" GEN)

AOT_theorem "cqt-basic:13": ‹∀α φ{α} ≡ ∀β φ{β}›                             (99.13)
  using "≡I" "→I" by blast

AOT_theorem "cqt-basic:14":                                                  (99.14)
  ‹(∀α1...∀αn (φ{α1...αn} → ψ{α1...αn})) →
   ((∀α1...∀αn φ{α1...αn}) → (∀α1...∀αn ψ{α1...αn}))›
  using "cqt:3"[axiom_inst] by auto

AOT_theorem "cqt-basic:15":                                                  (99.15)
  ‹(∀α1...∀αn (φ → ψ{α1...αn})) → (φ → (∀α1...∀αn ψ{α1...αn}))›
  using "cqt-orig:2" by auto

AOT_theorem "universal-cor":                                                 (100)
  assumes ‹for arbitrary β: φ{β}›
  shows ‹∀α φ{α}›
  using GEN assms .

AOT_theorem "existential:1":                                                 (101.1)
  assumes ‹φ{τ}› and ‹τ↓›
  shows ‹∃α φ{α}›
proof(rule "raa-cor:1")
  AOT_assume ‹¬∃α φ{α}›
  AOT_hence ‹∀α ¬φ{α}›
    using "≡dfI" "conventions:4" RAA "&I" by blast
  AOT_hence ‹¬φ{τ}› using assms(2) "∀E"(1) "→E" by blast
  AOT_thus ‹φ{τ} & ¬φ{τ}› using assms(1) "&I" by blast
qed

AOT_theorem "existential:2[const_var]":                                      (101.2)
  assumes ‹φ{β}›
  shows ‹∃α φ{α}›
  using "existential:1" "cqt:2[const_var]"[axiom_inst] assms by blast

AOT_theorem "existential:2[lambda]":                                         (101.2)
  assumes ‹φ{[λν1...νn ψ{ν1...νn}]}› and ‹INSTANCE_OF_CQT_2(ψ)›
  shows ‹∃α φ{α}›
  using "existential:1" "cqt:2[lambda]"[axiom_inst] assms by blast
lemmas "∃I" = "existential:1" "existential:2[const_var]"
              "existential:2[lambda]"

AOT_theorem "instantiation":                                                 (102)
  assumes ‹for arbitrary β: φ{β} ⊢ ψ› and ‹∃α φ{α}›
  shows ‹ψ›
  by (metis (no_types, lifting) "≡dfE" GEN "raa-cor:3" "conventions:4" assms)
lemmas "∃E" = "instantiation"

AOT_theorem "cqt-further:1": ‹∀α φ{α} → ∃α φ{α}›                            (103.1)
  using "∀E"(4) "∃I"(2) "→I" by metis

AOT_theorem "cqt-further:2": ‹¬∀α φ{α} → ∃α ¬φ{α}›                          (103.2)
  using "∀I" "∃I"(2) "→I" RAA by metis

AOT_theorem "cqt-further:3": ‹∀α φ{α} ≡ ¬∃α ¬φ{α}›                          (103.3)
  using "∀E"(4) "∃E" "→I" RAA
  by (metis "cqt-further:2" "≡I" "modus-tollens:1")

AOT_theorem "cqt-further:4": ‹¬∃α φ{α} → ∀α ¬φ{α}›                          (103.4)
  using "∀I" "∃I"(2)"→I" RAA by metis

AOT_theorem "cqt-further:5": ‹∃α (φ{α} & ψ{α}) → (∃α φ{α} & ∃α ψ{α})›       (103.5)
```

```
  by (metis (no_types, lifting) "&E" "&I" "∃E" "∃I"(2) "→I")

AOT_theorem "cqt-further:6": ‹∃α (φ{α} ∨ ψ{α}) → (∃α φ{α} ∨ ∃α ψ{α})›      (103.6)
  by (metis (mono_tags, lifting) "∃E" "∃I"(2) "∨E"(3) "∨I"(1, 2) "→I" RAA(2))

(* NOTE: vacuous in the embedding *)
AOT_theorem "cqt-further:7": ‹∃α φ{α} ≡ ∃β φ{β}›      (103.7)
  by (simp add: "oth-class-taut:3:a")

AOT_theorem "cqt-further:8":      (103.8)
  ‹(∀α φ{α} & ∀α ψ{α}) → ∀α (φ{α} ≡ ψ{α})›
  by (metis (mono_tags, lifting) "&E" "≡I" "∀E"(2) "→I" GEN)

AOT_theorem "cqt-further:9":      (103.9)
  ‹(¬∃α φ{α} & ¬∃α ψ{α}) → ∀α (φ{α} ≡ ψ{α})›
  by (metis (mono_tags, lifting) "&E" "≡I" "∃I"(2) "→I" GEN "raa-cor:4")

AOT_theorem "cqt-further:10":      (103.10)
  ‹(∃α φ{α} & ¬∃α ψ{α}) → ¬∀α (φ{α} ≡ ψ{α})›
proof(rule "→I"; rule "raa-cor:2")
  AOT_assume 0: ‹∃α φ{α} & ¬∃α ψ{α}›
  then AOT_obtain α where ‹φ{α}› using "∃E" "&E"(1) by metis
  moreover AOT_assume ‹∀α (φ{α} ≡ ψ{α})›
  ultimately AOT_have ‹ψ{α}› using "∀E"(4) "≡E"(1) by blast
  AOT_hence ‹∃α ψ{α}› using "∃I" by blast
  AOT_thus ‹∃α ψ{α} & ¬∃α ψ{α}› using 0 "&E"(2) "&I" by blast
qed

AOT_theorem "cqt-further:11": ‹∃α∃β φ{α,β} ≡ ∃β∃α φ{α,β}›      (103.11)
  using "≡I" "→I" "∃I"(2) "∃E" by metis

subsection‹Logical Existence, Identity, and Truth›
text‹\label{PLM: 9.7}›

AOT_theorem "log-prop-prop:1": ‹[λ φ]↓›      (104.1)
  using "cqt:2[lambda0]"[axiom_inst] by auto

AOT_theorem "log-prop-prop:2": ‹φ↓›      (104.2)
  by (rule "≡dfI"[OF "existence:3"]) "cqt:2[lambda]"

AOT_theorem "exist-nec": ‹τ↓ → □τ↓›      (106)
proof -
  AOT_have ‹∀β □β↓›
    by (simp add: GEN RN "cqt:2[const_var]"[axiom_inst])
  AOT_thus ‹τ↓ → □τ↓›
    using "cqt:1"[axiom_inst] "→E" by blast
qed

(* TODO: replace this mechanism by a "proof by types" command *)
class AOT_Term_id = AOT_Term +
  assumes "t=t-proper:1"[AOT]: ‹[v ⊨ τ = τ' → τ↓]›      (107.1)
      and "t=t-proper:2"[AOT]: ‹[v ⊨ τ = τ' → τ'↓]›      (107.2)

instance κ :: AOT_Term_id
proof
  AOT_modally_strict {
    AOT_show ‹κ = κ' → κ↓› for κ κ'
    proof(rule "→I")
      AOT_assume ‹κ = κ'›
      AOT_hence ‹O!κ ∨ A!κ›
        by (rule "∨I"(3)[OF "≡dfE"[OF "identity:1"]])
           (meson "→I" "∨I"(1) "&E"(1))+
      AOT_thus ‹κ↓›
```

```
        by (rule "∨E"(1))
          (metis "cqt:5:a"[axiom_inst] "→I" "→E" "&E"(2))+
    qed
  }
next
  AOT_modally_strict {
    AOT_show ‹κ = κ' → κ'↓› for κ κ'
    proof(rule "→I")
      AOT_assume ‹κ = κ'›
      AOT_hence ‹O!κ' ∨ A!κ'›
        by (rule "∨I"(3)[OF "≡dfE"[OF "identity:1"]])
          (meson "→I" "∨I" "&E")+
      AOT_thus ‹κ'↓›
        by (rule "∨E"(1))
          (metis "cqt:5:a"[axiom_inst] "→I" "→E" "&E"(2))+
    qed
  }
qed

instance rel :: (AOT_κs) AOT_Term_id
proof
  AOT_modally_strict {
    AOT_show ‹Π = Π' → Π↓› for Π Π' :: ‹<'a>›
    proof(rule "→I")
      AOT_assume ‹Π = Π'›
      AOT_thus ‹Π↓› using "≡dfE"[OF "identity:3"[of Π Π']] "&E" by blast
    qed
  }
next
  AOT_modally_strict {
    AOT_show ‹Π = Π' → Π'↓› for Π Π' :: ‹<'a>›
    proof(rule "→I")
      AOT_assume ‹Π = Π'›
      AOT_thus ‹Π'↓› using "≡dfE"[OF "identity:3"[of Π Π']] "&E" by blast
    qed
  }
qed

instance o :: AOT_Term_id
proof
  AOT_modally_strict {
    fix φ ψ
    AOT_show ‹φ = ψ → φ↓›
    proof(rule "→I")
      AOT_assume ‹φ = ψ›
      AOT_thus ‹φ↓› using "≡dfE"[OF "identity:4"[of φ ψ]] "&E" by blast
    qed
  }
next
  AOT_modally_strict {
    fix φ ψ
    AOT_show ‹φ = ψ → ψ↓›
    proof(rule "→I")
      AOT_assume ‹φ = ψ›
      AOT_thus ‹ψ↓› using "≡dfE"[OF "identity:4"[of φ ψ]] "&E" by blast
    qed
  }
qed

instance prod :: (AOT_Term_id, AOT_Term_id) AOT_Term_id
proof
  AOT_modally_strict {
    fix τ τ' :: ‹'a×'b›
```

```
    AOT_show ‹τ = τ' → τ↓›
    proof (induct τ; induct τ'; rule "→I")
      fix τ₁ τ₁' :: 'a and τ₂  τ₂' :: 'b
      AOT_assume ‹«(τ₁, τ₂)» = «(τ₁', τ₂')»›
      AOT_hence ‹(τ₁ = τ₁') & (τ₂ = τ₂')› by (metis "≡dfE" tuple_identity_1)
      AOT_hence ‹τ₁↓› and ‹τ₂↓›
        using "t=t-proper:1" "&E" "vdash-properties:10" by blast+
      AOT_thus ‹«(τ₁, τ₂)»↓› by (metis "≡dfI" "&I" tuple_denotes)
    qed
  }
next
  AOT_modally_strict {
    fix τ τ' :: ‹'a×'b›
    AOT_show ‹τ = τ' → τ'↓›
    proof (induct τ; induct τ'; rule "→I")
      fix τ₁ τ₁' :: 'a and τ₂  τ₂' :: 'b
      AOT_assume ‹«(τ₁, τ₂)» = «(τ₁', τ₂')»›
      AOT_hence ‹(τ₁ = τ₁') & (τ₂ = τ₂')› by (metis "≡dfE" tuple_identity_1)
      AOT_hence ‹τ₁'↓› and ‹τ₂'↓›
        using "t=t-proper:2" "&E" "vdash-properties:10" by blast+
      AOT_thus ‹«(τ₁', τ₂')»↓› by (metis "≡dfI" "&I" tuple_denotes)
    qed
  }
qed

(* This is the end of the "proof by types" and
   makes the results available on new theorems *)
AOT_register_type_constraints
  Term: ‹_::AOT_Term_id› ‹_::AOT_Term_id›
AOT_register_type_constraints
  Individual: ‹κ› ‹_::{AOT_κs, AOT_Term_id}›
AOT_register_type_constraints
  Relation: ‹<_::{AOT_κs, AOT_Term_id}>›

AOT_theorem "id-rel-nec-equiv:1":                                          (108.1)
  ‹Π = Π' → □∀x₁...∀xₙ ([Π]x₁...xₙ ≡ [Π']x₁...xₙ)›
proof(rule "→I")
  AOT_assume assumption: ‹Π = Π'›
  AOT_hence ‹Π↓› and ‹Π'↓›
    using "t=t-proper:1" "t=t-proper:2" MP by blast+
  moreover AOT_have ‹∀F∀G (F = G → ((□∀x₁...∀xₙ ([F]x₁...xₙ ≡ [F]x₁...xₙ)) →
                                    □∀x₁...∀xₙ ([F]x₁...xₙ ≡ [G]x₁...xₙ)))›
    apply (rule GEN)+ using "l-identity"[axiom_inst] by force
  ultimately AOT_have ‹Π = Π' → ((□∀x₁...∀xₙ ([Π]x₁...xₙ ≡ [Π]x₁...xₙ)) →
                                  □∀x₁...∀xₙ ([Π]x₁...xₙ ≡ [Π']x₁...xₙ))›
    using "∀E"(1) by blast
  AOT_hence ‹(□∀x₁...∀xₙ ([Π]x₁...xₙ ≡ [Π]x₁...xₙ)) →
              □∀x₁...∀xₙ ([Π]x₁...xₙ ≡ [Π']x₁...xₙ)›
    using assumption "→E" by blast
  moreover AOT_have ‹□∀x₁...∀xₙ ([Π]x₁...xₙ ≡ [Π]x₁...xₙ)›
    by (simp add: RN "oth-class-taut:3:a" "universal-cor")
  ultimately AOT_show ‹□∀x₁...∀xₙ ([Π]x₁...xₙ ≡ [Π']x₁...xₙ)›
    using "→E" by blast
qed

AOT_theorem "id-rel-nec-equiv:2": ‹φ = ψ → □(φ ≡ ψ)›                        (108.2)
proof(rule "→I")
  AOT_assume assumption: ‹φ = ψ›
  AOT_hence ‹φ↓› and ‹ψ↓›
    using "t=t-proper:1" "t=t-proper:2" MP by blast+
  moreover AOT_have ‹∀p∀q (p = q → ((□(p ≡ p) → □(p ≡ q))))›
    apply (rule GEN)+ using "l-identity"[axiom_inst] by force
  ultimately AOT_have ‹φ = ψ → (□(φ ≡ φ) → □(φ ≡ ψ))›
```

```
    using "∀E"(1) by blast
  AOT_hence ‹□(φ ≡ φ) → □(φ ≡ ψ)›
    using assumption "→E" by blast
  moreover AOT_have ‹□(φ ≡ φ)›
    by (simp add: RN "oth-class-taut:3:a" "universal-cor")
  ultimately AOT_show ‹□(φ ≡ ψ)›
    using "→E" by blast
qed

AOT_theorem "rule=E":                                                    (110)
  assumes ‹φ{τ}› and ‹τ = σ›
  shows ‹φ{σ}›
proof -
  AOT_have ‹τ↓› and ‹σ↓›
    using assms(2) "t=t-proper:1" "t=t-proper:2" "→E" by blast+
  moreover AOT_have ‹∀α∀β(α = β → (φ{α} → φ{β}))›
    apply (rule GEN)+ using "l-identity"[axiom_inst] by blast
  ultimately AOT_have ‹τ = σ → (φ{τ} → φ{σ})›
    using "∀E"(1) by blast
  AOT_thus ‹φ{σ}› using assms "→E" by blast
qed

AOT_theorem "propositions-lemma:1": ‹[λ φ] = φ›                          (111.1)
proof -
  AOT_have ‹φ↓› by (simp add: "log-prop-prop:2")
  moreover AOT_have ‹∀p [λ p] = p›
    using "lambda-predicates:3[zero]"[axiom_inst] "∀I" by fast
  ultimately AOT_show ‹[λ φ] = φ›
    using "∀E" by blast
qed

AOT_theorem "propositions-lemma:2": ‹[λ φ] ≡ φ›                          (111.2)
proof -
  AOT_have ‹[λ φ] ≡ [λ φ]› by (simp add: "oth-class-taut:3:a")
  AOT_thus ‹[λ φ] ≡ φ› using "propositions-lemma:1" "rule=E" by blast
qed

text‹propositions-lemma:3 through propositions-lemma:5 hold implicitly›

AOT_theorem "propositions-lemma:6": ‹(φ ≡ ψ) ≡ ([λ φ] ≡ [λ ψ])›          (111.6)
  by (metis "≡E"(1) "≡E"(5) "Associativity of ≡" "propositions-lemma:2")

text‹dr-alphabetic-rules holds implicitly›

AOT_theorem "oa-exist:1": ‹O!↓›                                          (115.1)
proof -
  AOT_have ‹[λx ◇[E!]x]↓› by "cqt:2[lambda]"
  AOT_hence 1: ‹O! = [λx ◇[E!]x]›
    using "df-rules-terms[4]"[OF "oa:1", THEN "&E"(1)] "→E" by blast
  AOT_show ‹O!↓› using "t=t-proper:1"[THEN "→E", OF 1] by simp
qed

AOT_theorem "oa-exist:2": ‹A!↓›                                          (115.2)
proof -
  AOT_have ‹[λx ¬◇[E!]x]↓› by "cqt:2[lambda]"
  AOT_hence 1: ‹A! = [λx ¬◇[E!]x]›
    using "df-rules-terms[4]"[OF "oa:2", THEN "&E"(1)] "→E" by blast
  AOT_show ‹A!↓› using "t=t-proper:1"[THEN "→E", OF 1] by simp
qed

AOT_theorem "oa-exist:3": ‹O!x ∨ A!x›                                    (115.3)
proof(rule "raa-cor:1")
  AOT_assume ‹¬(O!x ∨ A!x)›
```

```
  AOT_hence A: ‹¬O!x› and B: ‹¬A!x›
    using "Disjunction Addition"(1) "modus-tollens:1"
          "∨I"(2) "raa-cor:5" by blast+
  AOT_have C: ‹O! = [λx ◇[E!]x]›
    by (rule "df-rules-terms[4]"[OF "oa:1", THEN "&E"(1), THEN "→E"]) "cqt:2"
  AOT_have D: ‹A! = [λx ¬◇[E!]x]›
    by (rule "df-rules-terms[4]"[OF "oa:2", THEN "&E"(1), THEN "→E"]) "cqt:2"
  AOT_have E: ‹¬[λx ◇[E!]x]x›
    using A C "rule=E" by fast
  AOT_have F: ‹¬[λx ¬◇[E!]x]x›
    using B D "rule=E" by fast
  AOT_have G: ‹[λx ◇[E!]x]x ≡ ◇[E!]x›
    by (rule "lambda-predicates:2"[axiom_inst, THEN "→E"]) "cqt:2"
  AOT_have H: ‹[λx ¬◇[E!]x]x ≡ ¬◇[E!]x›
    by (rule "lambda-predicates:2"[axiom_inst, THEN "→E"]) "cqt:2"
  AOT_show ‹¬◇[E!]x & ¬¬◇[E!]x› using G E "≡E" H F "≡E" "&I" by metis
qed

AOT_theorem "p-identity-thm2:1": ‹F = G ≡ □∀x(x[F] ≡ x[G])›                  (116.1)
proof -
  AOT_have ‹F = G ≡ F↓ & G↓ & □∀x(x[F] ≡ x[G])›
    using "identity:2" "df-rules-formulas[3]" "df-rules-formulas[4]"
          "→E" "&E" "≡I" "→I" by blast
  moreover AOT_have ‹F↓› and ‹G↓›
    by (auto simp: "cqt:2[const_var]"[axiom_inst])
  ultimately AOT_show ‹F = G ≡ □∀x(x[F] ≡ x[G])›
    using "≡S"(1) "&I" by blast
qed

AOT_theorem "p-identity-thm2:2[2]":                                            (116.2)
  ‹F = G ≡ ∀y1([λx [F]xy1] = [λx [G]xy1] & [λx [F]y1x] = [λx [G]y1x])›
proof -
  AOT_have ‹F = G ≡ F↓ & G↓ &
                  ∀y1([λx [F]xy1] = [λx [G]xy1] & [λx [F]y1x] = [λx [G]y1x])›
    using "identity:3[2]" "df-rules-formulas[3]" "df-rules-formulas[4]"
          "→E" "&E" "≡I" "→I" by blast
  moreover AOT_have ‹F↓› and ‹G↓›
    by (auto simp: "cqt:2[const_var]"[axiom_inst])
  ultimately show ?thesis
    using "≡S"(1) "&I" by blast
qed

AOT_theorem "p-identity-thm2:2[3]":                                            (116.2)
  ‹F = G ≡ ∀y1∀y2([λx [F]xy1y2] = [λx [G]xy1y2] &
                  [λx [F]y1xy2] = [λx [G]y1xy2] &
                  [λx [F]y1y2x] = [λx [G]y1y2x])›
proof -
  AOT_have ‹F = G ≡ F↓ & G↓ & ∀y1∀y2([λx [F]xy1y2] = [λx [G]xy1y2] &
                                    [λx [F]y1xy2] = [λx [G]y1xy2] &
                                    [λx [F]y1y2x] = [λx [G]y1y2x])›
    using "identity:3[3]" "df-rules-formulas[3]" "df-rules-formulas[4]"
          "→E" "&E" "≡I" "→I" by blast
  moreover AOT_have ‹F↓› and ‹G↓›
    by (auto simp: "cqt:2[const_var]"[axiom_inst])
  ultimately show ?thesis
    using "≡S"(1) "&I" by blast
qed

AOT_theorem "p-identity-thm2:2[4]":                                            (116.2)
  ‹F = G ≡ ∀y1∀y2∀y3([λx [F]xy1y2y3] = [λx [G]xy1y2y3] &
                     [λx [F]y1xy2y3] = [λx [G]y1xy2y3] &
                     [λx [F]y1y2xy3] = [λx [G]y1y2xy3] &
                     [λx [F]y1y2y3x] = [λx [G]y1y2y3x])›
```

```
proof -
  AOT_have ‹F = G ≡ F↓ & G↓ & ∀y1∀y2∀y3([λx [F]xy1y2y3] = [λx [G]xy1y2y3] &
                                       [λx [F]y1xy2y3] = [λx [G]y1xy2y3] &
                                       [λx [F]y1y2xy3] = [λx [G]y1y2xy3] &
                                       [λx [F]y1y2y3x] = [λx [G]y1y2y3x])›
    using "identity:3[4]" "df-rules-formulas[3]" "df-rules-formulas[4]"
          "→E" "&E" "≡I" "→I" by blast
  moreover AOT_have ‹F↓› and ‹G↓›
    by (auto simp: "cqt:2[const_var]"[axiom_inst])
  ultimately show ?thesis
    using "≡S"(1) "&I" by blast
qed

AOT_theorem "p-identity-thm2:2":                                              (116.2)
  ‹F = G ≡ ∀x1...∀xn «AOT_sem_proj_id x1xn (λ τ . «[F]τ») (λ τ . «[G]τ»)»›
proof -
  AOT_have ‹F = G ≡ F↓ & G↓ &
            ∀x1...∀xn «AOT_sem_proj_id x1xn (λ τ . «[F]τ») (λ τ . «[G]τ»)»›
    using "identity:3" "df-rules-formulas[3]" "df-rules-formulas[4]"
          "→E" "&E" "≡I" "→I" by blast
  moreover AOT_have ‹F↓› and ‹G↓›
    by (auto simp: "cqt:2[const_var]"[axiom_inst])
  ultimately show ?thesis
    using "≡S"(1) "&I" by blast
qed

AOT_theorem "p-identity-thm2:3":                                              (116.3)
  ‹p = q ≡ [λx p] = [λx q]›
proof -
  AOT_have ‹p = q ≡ p↓ & q↓ & [λx p] = [λx q]›
    using "identity:4" "df-rules-formulas[3]" "df-rules-formulas[4]"
          "→E" "&E" "≡I" "→I" by blast
  moreover AOT_have ‹p↓› and ‹q↓›
    by (auto simp: "cqt:2[const_var]"[axiom_inst])
  ultimately show ?thesis
    using "≡S"(1) "&I" by blast
qed

class AOT_Term_id_2 = AOT_Term_id + assumes "id-eq:1": ‹[v ⊨ α = α]›          (117.1)

instance κ :: AOT_Term_id_2
proof
  AOT_modally_strict {
    fix x
    {
      AOT_assume ‹O!x›
      moreover AOT_have ‹□∀F([F]x ≡ [F]x)›
        using RN GEN "oth-class-taut:3:a" by fast
      ultimately AOT_have ‹O!x & O!x & □∀F([F]x ≡ [F]x)› using "&I" by simp
    }
    moreover {
      AOT_assume ‹A!x›
      moreover AOT_have ‹□∀F(x[F] ≡ x[F])›
        using RN GEN "oth-class-taut:3:a" by fast
      ultimately AOT_have ‹A!x & A!x & □∀F(x[F] ≡ x[F])› using "&I" by simp
    }
    ultimately AOT_have ‹(O!x & O!x & □∀F([F]x ≡ [F]x)) ∨
                         (A!x & A!x & □∀F(x[F] ≡ x[F]))›
      using "oa-exist:3" "∨I"(1) "∨I"(2) "∨E"(3) "raa-cor:1" by blast
    AOT_thus ‹x = x›
      using "identity:1"[THEN "df-rules-formulas[4]"] "→E" by blast
  }
qed
```

```
instance rel :: ("{AOT_κs,AOT_Term_id_2}") AOT_Term_id_2
proof
  AOT_modally_strict {
    fix F :: "<'a> AOT_var"
    AOT_have 0: ‹[λx₁...xₙ [F]x₁...xₙ] = F›
      by (simp add: "lambda-predicates:3"[axiom_inst])
    AOT_have ‹[λx₁...xₙ [F]x₁...xₙ]↓›
      by "cqt:2[lambda]"
    AOT_hence ‹[λx₁...xₙ [F]x₁...xₙ] = [λx₁...xₙ [F]x₁...xₙ]›
      using "lambda-predicates:1"[axiom_inst] "→E" by blast
    AOT_show ‹F = F› using "rule=E" 0 by force
  }
qed

instance o :: AOT_Term_id_2
proof
  AOT_modally_strict {
    fix p
    AOT_have 0: ‹[λ p] = p›
      by (simp add: "lambda-predicates:3[zero]"[axiom_inst])
    AOT_have ‹[λ p]↓›
      by (rule "cqt:2[lambda0]"[axiom_inst])
    AOT_hence ‹[λ p] = [λ p]›
      using "lambda-predicates:1[zero]"[axiom_inst] "→E" by blast
    AOT_show ‹p = p› using "rule=E" 0 by force
  }
qed

instance prod :: (AOT_Term_id_2, AOT_Term_id_2) AOT_Term_id_2
proof
  AOT_modally_strict {
    fix α :: ‹('a×'b) AOT_var›
    AOT_show ‹α = α›
    proof (induct)
      AOT_show ‹τ = τ› if ‹τ↓› for τ :: ‹'a×'b›
        using that
      proof (induct τ)
        fix τ₁ :: 'a and τ₂ :: 'b
        AOT_assume ‹«(τ₁,τ₂)»↓›
        AOT_hence ‹τ₁↓› and ‹τ₂↓›
          using "≡dfE" "&E" tuple_denotes by blast+
        AOT_hence ‹τ₁ = τ₁› and ‹τ₂ = τ₂›
          using "id-eq:1"[unvarify α] by blast+
        AOT_thus ‹«(τ₁, τ₂)» = «(τ₁, τ₂)»›
          by (metis "≡dfI" "&I" tuple_identity_1)
      qed
    qed
  }
qed

AOT_register_type_constraints
  Term: ‹_::AOT_Term_id_2› ‹_::AOT_Term_id_2›
AOT_register_type_constraints
  Individual: ‹κ› ‹_::{AOT_κs, AOT_Term_id_2}›
AOT_register_type_constraints
  Relation: ‹<_::{AOT_κs, AOT_Term_id_2}>›

AOT_theorem "id-eq:2": ‹α = β → β = α›                    (117.2)
  by (meson "rule=E" "deduction-theorem")

AOT_theorem "id-eq:3": ‹α = β & β = γ → α = γ›            (117.3)
  using "rule=E" "→I" "&E" by blast
```

```
AOT_theorem "id-eq:4": ‹α = β ≡ ∀γ (α = γ ≡ β = γ)›                    (117.4)
proof (rule "≡I"; rule "→I")
  AOT_assume 0: ‹α = β›
  AOT_hence 1: ‹β = α› using "id-eq:2" "→E" by blast
  AOT_show ‹∀γ (α = γ ≡ β = γ)›
    by (rule GEN) (metis "≡I" "→I" 0 "1" "rule=E")
next
  AOT_assume ‹∀γ (α = γ ≡ β = γ)›
  AOT_hence ‹α = α ≡ β = α› using "∀E"(2) by blast
  AOT_hence ‹α = α → β = α› using "≡E"(1) "→I" by blast
  AOT_hence ‹β = α› using "id-eq:1" "→E" by blast
  AOT_thus ‹α = β› using "id-eq:2" "→E" by blast
qed

AOT_theorem "rule=I:1":                                                  (118.1)
  assumes ‹τ↓›
  shows ‹τ = τ›
proof -
  AOT_have ‹∀α (α = α)›
    by (rule GEN) (metis "id-eq:1")
  AOT_thus ‹τ = τ› using assms "∀E" by blast
qed

AOT_theorem "rule=I:2[const_var]": "α = α"                               (118.2)
  using "id-eq:1".

AOT_theorem "rule=I:2[lambda]":                                          (118.2)
  assumes ‹INSTANCE_OF_CQT_2(φ)›
  shows "[λν1...νn φ{ν1...νn}] = [λν1...νn φ{ν1...νn}]"
proof -
  AOT_have ‹∀α (α = α)›
    by (rule GEN) (metis "id-eq:1")
  moreover AOT_have ‹[λν1...νn φ{ν1...νn}]↓›
    using assms by (rule "cqt:2[lambda]"[axiom_inst])
  ultimately AOT_show ‹[λν1...νn φ{ν1...νn}] = [λν1...νn φ{ν1...νn}]›
    using assms "∀E" by blast
qed

lemmas "=I" = "rule=I:1" "rule=I:2[const_var]" "rule=I:2[lambda]"

AOT_theorem "rule-id-df:1":                                              (120.1)
  assumes ‹τ{α1...αn} =df σ{α1...αn}› and ‹σ{τ1...τn}↓›
  shows ‹τ{τ1...τn} = σ{τ1...τn}›
proof -
  AOT_have ‹σ{τ1...τn}↓ → τ{τ1...τn} = σ{τ1...τn}›
    using "df-rules-terms[3]" assms(1) "&E" by blast
  AOT_thus ‹τ{τ1...τn} = σ{τ1...τn}›
    using assms(2) "→E" by blast
qed

AOT_theorem "rule-id-df:1[zero]":                                        (120.1)
  assumes ‹τ =df σ› and ‹σ↓›
  shows ‹τ = σ›
proof -
  AOT_have ‹σ↓ → τ = σ›
    using "df-rules-terms[4]" assms(1) "&E" by blast
  AOT_thus ‹τ = σ›
    using assms(2) "→E" by blast
qed

AOT_theorem "rule-id-df:2:a":                                            (120.2.a)
  assumes ‹τ{α1...αn} =df σ{α1...αn}› and ‹σ{τ1...τn}↓› and ‹φ{τ{τ1...τn}}›
```

```
  shows ‹φ{σ{τ₁...τₙ}}›
proof -
  AOT_have ‹τ{τ₁...τₙ} = σ{τ₁...τₙ}› using "rule-id-df:1" assms(1,2) by blast
  AOT_thus ‹φ{σ{τ₁...τₙ}}› using assms(3) "rule=E" by blast
qed

AOT_theorem "rule-id-df:2:a[2]":                                          (120.2.a)
  assumes ‹τ{«(α₁,α₂)»} =df σ{«(α₁,α₂)»}›
    and ‹σ{«(τ₁,τ₂)»}↓›
      and ‹φ{τ{«(τ₁,τ₂)»}}›
  shows ‹φ{σ{«(τ₁::'a::AOT_Term_id_2,τ₂::'b::AOT_Term_id_2)»}}›
proof -
  AOT_have ‹τ{«(τ₁,τ₂)»} = σ{«(τ₁,τ₂)»}›
    using "rule-id-df:1" assms(1,2) by auto
  AOT_thus ‹φ{σ{«(τ₁,τ₂)»}}› using assms(3) "rule=E" by blast
qed

AOT_theorem "rule-id-df:2:a[zero]":                                       (120.2.a)
  assumes ‹τ =df σ› and ‹σ↓› and ‹φ{τ}›
  shows ‹φ{σ}›
proof -
  AOT_have ‹τ = σ› using "rule-id-df:1[zero]" assms(1,2) by blast
  AOT_thus ‹φ{σ}› using assms(3) "rule=E" by blast
qed

lemmas "=dfE" = "rule-id-df:2:a" "rule-id-df:2:a[zero]"

AOT_theorem "rule-id-df:2:b":                                             (120.2.b)
  assumes ‹τ{α₁...αₙ} =df σ{α₁...αₙ}› and ‹σ{τ₁...τₙ}↓› and ‹φ{σ{τ₁...τₙ}}›
  shows ‹φ{τ{τ₁...τₙ}}›
proof -
  AOT_have ‹τ{τ₁...τₙ} = σ{τ₁...τₙ}›
    using "rule-id-df:1" assms(1,2) by blast
  AOT_hence ‹σ{τ₁...τₙ} = τ{τ₁...τₙ}›
    using "rule=E" "=I"(1) "t=t-proper:1" "→E" by fast
  AOT_thus ‹φ{τ{τ₁...τₙ}}› using assms(3) "rule=E" by blast
qed

AOT_theorem "rule-id-df:2:b[2]":                                          (120.2.b)
  assumes ‹τ{«(α₁,α₂)»} =df σ{«(α₁,α₂)»}›
      and ‹σ{«(τ₁,τ₂)»}↓›
      and ‹φ{σ{«(τ₁,τ₂)»}}›
  shows ‹φ{τ{«(τ₁::'a::AOT_Term_id_2,τ₂::'b::AOT_Term_id_2)»}}›
proof -
  AOT_have ‹τ{«(τ₁,τ₂)»} = σ{«(τ₁,τ₂)»}›
    using "=I"(1) "rule-id-df:2:a[2]" RAA(1) assms(1,2) "→I" by metis
  AOT_hence ‹σ{«(τ₁,τ₂)»} = τ{«(τ₁,τ₂)»}›
    using "rule=E" "=I"(1) "t=t-proper:1" "→E" by fast
  AOT_thus ‹φ{τ{«(τ₁,τ₂)»}}› using assms(3) "rule=E" by blast
qed

AOT_theorem "rule-id-df:2:b[zero]":                                       (120.2.b)
  assumes ‹τ =df σ› and ‹σ↓› and ‹φ{σ}›
  shows ‹φ{τ}›
proof -
  AOT_have ‹τ = σ› using "rule-id-df:1[zero]" assms(1,2) by blast
  AOT_hence ‹σ = τ›
    using "rule=E" "=I"(1) "t=t-proper:1" "→E" by fast
  AOT_thus ‹φ{τ}› using assms(3) "rule=E" by blast
qed

lemmas "=dfI" = "rule-id-df:2:b" "rule-id-df:2:b[zero]"
```

```
AOT_theorem "free-thms:1": ‹τ↓ ≡ ∃β (β = τ)›                                    (121.1)
  by (metis "∃E" "rule=I:1" "t=t-proper:2" "→I" "∃I"(1) "≡I" "→E")

AOT_theorem "free-thms:2": ‹∀α φ{α} → (∃β (β = τ) → φ{τ})›                      (121.2)
  by (metis "∃E" "rule=E" "cqt:2[const_var]"[axiom_inst] "→I" "∀E"(1))

AOT_theorem "free-thms:3[const_var]": ‹∃β (β = α)›                               (121.3)
  by (meson "∃I"(2) "id-eq:1")

AOT_theorem "free-thms:3[lambda]":                                               (121.3)
  assumes ‹INSTANCE_OF_CQT_2(φ)›
  shows ‹∃β (β = [λν1...νn φ{ν1...νn}])›
  by (meson "=I"(3) assms "cqt:2[lambda]"[axiom_inst] "existential:1")

AOT_theorem "free-thms:4[rel]":                                                  (121.4)
  ‹([Π]κ1...κn ∨ κ1...κn[Π]) → ∃β (β = Π)›
  by (metis "rule=I:1" "&E"(1) "∨E"(1) "cqt:5:a"[axiom_inst]
            "cqt:5:b"[axiom_inst] "→I" "∃I"(1))

AOT_theorem "free-thms:4[vars]":                                                 (121.4)
  ‹([Π]κ1...κn ∨ κ1...κn[Π]) → ∃β1...∃βn (β1...βn = κ1...κn)›
  by (metis "rule=I:1" "&E"(2) "∨E"(1) "cqt:5:a"[axiom_inst]
            "cqt:5:b"[axiom_inst] "→I" "∃I"(1))

AOT_theorem "free-thms:4[1,rel]":                                                (121.4)
  ‹([Π]κ ∨ κ[Π]) → ∃β (β = Π)›
  by (metis "rule=I:1" "&E"(1) "∨E"(1) "cqt:5:a"[axiom_inst]
            "cqt:5:b"[axiom_inst] "→I" "∃I"(1))
AOT_theorem "free-thms:4[1,1]":                                                  (121.4)
  ‹([Π]κ ∨ κ[Π]) → ∃β (β = κ)›
  by (metis "rule=I:1" "&E"(2) "∨E"(1) "cqt:5:a"[axiom_inst]
            "cqt:5:b"[axiom_inst] "→I" "∃I"(1))

AOT_theorem "free-thms:4[2,rel]":                                                (121.4)
  ‹([Π]κ1κ2 ∨ κ1κ2[Π]) → ∃β (β = Π)›
  by (metis "rule=I:1" "&E"(1) "∨E"(1) "cqt:5:a[2]"[axiom_inst]
            "cqt:5:b[2]"[axiom_inst] "→I" "∃I"(1))
AOT_theorem "free-thms:4[2,1]":                                                  (121.4)
  ‹([Π]κ1κ2 ∨ κ1κ2[Π]) → ∃β (β = κ1)›
  by (metis "rule=I:1" "&E" "∨E"(1) "cqt:5:a[2]"[axiom_inst]
            "cqt:5:b[2]"[axiom_inst] "→I" "∃I"(1))
AOT_theorem "free-thms:4[2,2]":                                                  (121.4)
  ‹([Π]κ1κ2 ∨ κ1κ2[Π]) → ∃β (β = κ2)›
  by (metis "rule=I:1" "&E"(2) "∨E"(1) "cqt:5:a[2]"[axiom_inst]
            "cqt:5:b[2]"[axiom_inst] "→I" "∃I"(1))
AOT_theorem "free-thms:4[3,rel]":                                                (121.4)
  ‹([Π]κ1κ2κ3 ∨ κ1κ2κ3[Π]) → ∃β (β = Π)›
  by (metis "rule=I:1" "&E"(1) "∨E"(1) "cqt:5:a[3]"[axiom_inst]
            "cqt:5:b[3]"[axiom_inst] "→I" "∃I"(1))
AOT_theorem "free-thms:4[3,1]":                                                  (121.4)
  ‹([Π]κ1κ2κ3 ∨ κ1κ2κ3[Π]) → ∃β (β = κ1)›
  by (metis "rule=I:1" "&E" "∨E"(1) "cqt:5:a[3]"[axiom_inst]
            "cqt:5:b[3]"[axiom_inst] "→I" "∃I"(1))
AOT_theorem "free-thms:4[3,2]":                                                  (121.4)
  ‹([Π]κ1κ2κ3 ∨ κ1κ2κ3[Π]) → ∃β (β = κ2)›
  by (metis "rule=I:1" "&E" "∨E"(1) "cqt:5:a[3]"[axiom_inst]
            "cqt:5:b[3]"[axiom_inst] "→I" "∃I"(1))
AOT_theorem "free-thms:4[3,3]":                                                  (121.4)
  ‹([Π]κ1κ2κ3 ∨ κ1κ2κ3[Π]) → ∃β (β = κ3)›
  by (metis "rule=I:1" "&E"(2) "∨E"(1) "cqt:5:a[3]"[axiom_inst]
            "cqt:5:b[3]"[axiom_inst] "→I" "∃I"(1))
AOT_theorem "free-thms:4[4,rel]":                                                (121.4)
  ‹([Π]κ1κ2κ3κ4 ∨ κ1κ2κ3κ4[Π]) → ∃β (β = Π)›
```

```
  by (metis "rule=I:1" "&E"(1) "∨E"(1) "cqt:5:a[4]"[axiom_inst]
            "cqt:5:b[4]"[axiom_inst] "→I" "∃I"(1))
AOT_theorem "free-thms:4[4,1]":                                    (121.4)
  ‹([Π]κ₁κ₂κ₃κ₄ ∨ κ₁κ₂κ₃κ₄[Π]) → ∃β (β = κ₁)›
  by (metis "rule=I:1" "&E" "∨E"(1) "cqt:5:a[4]"[axiom_inst]
            "cqt:5:b[4]"[axiom_inst] "→I" "∃I"(1))
AOT_theorem "free-thms:4[4,2]":                                    (121.4)
  ‹([Π]κ₁κ₂κ₃κ₄ ∨ κ₁κ₂κ₃κ₄[Π]) → ∃β (β = κ₂)›
  by (metis "rule=I:1" "&E" "∨E"(1) "cqt:5:a[4]"[axiom_inst]
            "cqt:5:b[4]"[axiom_inst] "→I" "∃I"(1))
AOT_theorem "free-thms:4[4,3]":                                    (121.4)
  ‹([Π]κ₁κ₂κ₃κ₄ ∨ κ₁κ₂κ₃κ₄[Π]) → ∃β (β = κ₃)›
  by (metis "rule=I:1" "&E" "∨E"(1) "cqt:5:a[4]"[axiom_inst]
            "cqt:5:b[4]"[axiom_inst] "→I" "∃I"(1))
AOT_theorem "free-thms:4[4,4]":                                    (121.4)
  ‹([Π]κ₁κ₂κ₃κ₄ ∨ κ₁κ₂κ₃κ₄[Π]) → ∃β (β = κ₄)›
  by (metis "rule=I:1" "&E"(2) "∨E"(1) "cqt:5:a[4]"[axiom_inst]
            "cqt:5:b[4]"[axiom_inst] "→I" "∃I"(1))

AOT_theorem "ex:1:a": ‹∀α α↓›                                      (123.1.a)
  by (rule GEN) (fact "cqt:2[const_var]"[axiom_inst])
AOT_theorem "ex:1:b": ‹∀α∃β(β = α)›                                (123.1.b)
  by (rule GEN) (fact "free-thms:3[const_var]")

AOT_theorem "ex:2:a": ‹□α↓›                                        (123.2.a)
  by (rule RN) (fact "cqt:2[const_var]"[axiom_inst])
AOT_theorem "ex:2:b": ‹□∃β(β = α)›                                 (123.2.b)
  by (rule RN) (fact "free-thms:3[const_var]")

AOT_theorem "ex:3:a": ‹□∀α α↓›                                     (123.3.a)
  by (rule RN) (fact "ex:1:a")
AOT_theorem "ex:3:b": ‹□∀α∃β(β = α)›                               (123.3.b)
  by (rule RN) (fact "ex:1:b")

AOT_theorem "ex:4:a": ‹∀α □α↓›                                     (123.4.a)
  by (rule GEN; rule RN) (fact "cqt:2[const_var]"[axiom_inst])
AOT_theorem "ex:4:b": ‹∀α□∃β(β = α)›                               (123.4.b)
  by (rule GEN; rule RN) (fact "free-thms:3[const_var]")

AOT_theorem "ex:5:a": ‹□∀α □α↓›                                    (123.5.a)
  by (rule RN) (simp add: "ex:4:a")
AOT_theorem "ex:5:b": ‹□∀α□∃β(β = α)›                              (123.5.b)
  by (rule RN) (simp add: "ex:4:b")

AOT_theorem "all-self=:1": ‹□∀α(α = α)›                            (124.1)
  by (rule RN; rule GEN) (fact "id-eq:1")
AOT_theorem "all-self=:2": ‹∀α□(α = α)›                            (124.2)
  by (rule GEN; rule RN) (fact "id-eq:1")

AOT_theorem "id-nec:1": ‹α = β → □(α = β)›                         (125.1)
proof(rule "→I")
  AOT_assume ‹α = β›
  moreover AOT_have ‹□(α = α)›
    by (rule RN) (fact "id-eq:1")
  ultimately AOT_show ‹□(α = β)› using "rule=E" by fast
qed

AOT_theorem "id-nec:2": ‹τ = σ → □(τ = σ)›                         (125.2)
proof(rule "→I")
  AOT_assume asm: ‹τ = σ›
  moreover AOT_have ‹τ↓›
    using calculation "t=t-proper:1" "→E" by blast
  moreover AOT_have ‹□(τ = τ)›
```

```
    using calculation "all-self=:2" "∀E"(1) by blast
  ultimately AOT_show ‹□(τ = σ)› using "rule=E" by fast
qed

AOT_theorem "term-out:1": ‹φ{α} ≡ ∃β (β = α & φ{β})›    (126.1)
proof (rule "≡I"; rule "→I")
  AOT_assume asm: ‹φ{α}›
  AOT_show ‹∃β (β = α & φ{β})›
    by (rule "∃I"(2)[where β=α]; rule "&I")
       (auto simp: "id-eq:1" asm)
next
  AOT_assume 0: ‹∃β (β = α & φ{β})›
  AOT_obtain β where ‹β = α & φ{β}›
    using "∃E"[rotated, OF 0] by blast
  AOT_thus ‹φ{α}› using "&E" "rule=E" by blast
qed

AOT_theorem "term-out:2": ‹τ↓ → (φ{τ} ≡ ∃α(α = τ & φ{α}))›    (126.2)
proof(rule "→I")
  AOT_assume ‹τ↓›
  moreover AOT_have ‹∀α (φ{α} ≡ ∃β (β = α & φ{β}))›
    by (rule GEN) (fact "term-out:1")
  ultimately AOT_show ‹φ{τ} ≡ ∃α(α = τ & φ{α})›
    using "∀E" by blast
qed

AOT_theorem "term-out:3":    (126.3)
  ‹(φ{α} & ∀β(φ{β} → β = α)) ≡ ∀β(φ{β} ≡ β = α)›
  apply (rule "≡I"; rule "→I")
   apply (frule "&E"(1))
   apply (drule "&E"(2))
   apply (rule GEN; rule "≡I"; rule "→I")
  using "rule-ui:2[const_var]" "vdash-properties:5"
    apply blast
   apply (meson "rule=E" "id-eq:1")
  apply (rule "&I")
  using "id-eq:1" "≡E"(2) "rule-ui:3"
   apply blast
  apply (rule GEN; rule "→I")
  using "≡E"(1) "rule-ui:2[const_var]"
  by blast

(* Note: generalized alphabetic variant of the last theorem. *)
AOT_theorem "term-out:4":    (126.4)
  ‹(φ{β} & ∀α(φ{α} → α = β)) ≡ ∀α(φ{α} ≡ α = β)›
  using "term-out:3" .

(* TODO: Provide a nicer mechanism for introducing custom binders. *)
AOT_define AOT_exists_unique :: ‹α ⇒ φ ⇒ φ› "uniqueness:1":    (127.1)
  ‹«AOT_exists_unique φ» ≡df ∃α (φ{α} & ∀β (φ{β} → β = α))›
syntax (input) "_AOT_exists_unique" :: ‹α ⇒ φ ⇒ φ› ("∃!_ _" [1,40])
syntax (output) "_AOT_exists_unique" :: ‹α ⇒ φ ⇒ φ› ("∃!_'(_')" [1,40])
AOT_syntax_print_translations
  "_AOT_exists_unique τ φ" <= "CONST AOT_exists_unique (_abs τ φ)"
syntax
   "_AOT_exists_unique_ellipse" :: ‹id_position ⇒ id_position ⇒ φ ⇒ φ›
   (‹∃!_...∃!_ _› [1,40])
parse_ast_translation‹
[(syntax_const‹_AOT_exists_unique_ellipse›,
  fn ctx => fn [a,b,c] => Ast.mk_appl (Ast.Constant "AOT_exists_unique")
  [parseEllipseList "_AOT_vars" ctx [a,b],c]),
 (syntax_const‹_AOT_exists_unique›,
  AOT_restricted_binder
```

```
    const_name‹AOT_exists_unique›
    const_syntax‹AOT_conj›)]›
print_translation‹AOT_syntax_print_translations [
  AOT_preserve_binder_abs_tr'
    const_syntax‹AOT_exists_unique›
    syntax_const‹_AOT_exists_unique›
    (syntax_const‹_AOT_exists_unique_ellipse›, true)
    const_name‹AOT_conj›,
  AOT_binder_trans
    @{theory}
    @{binding "AOT_exists_unique_binder"}
    syntax_const‹_AOT_exists_unique›
]›


context AOT_meta_syntax
begin
notation AOT_exists_unique (binder "∃!" 20)
end
context AOT_no_meta_syntax
begin
no_notation AOT_exists_unique (binder "∃!" 20)
end

AOT_theorem "uniqueness:2": ‹∃!α φ{α} ≡ ∃α∀β(φ{β} ≡ β = α)›          (127.2)
proof(rule "≡I"; rule "→I")
    AOT_assume ‹∃!α φ{α}›
    AOT_hence ‹∃α (φ{α} & ∀β (φ{β} → β = α))›
      using "uniqueness:1" "≡dfE" by blast
    then AOT_obtain α where ‹φ{α} & ∀β (φ{β} → β = α)›
      using "instantiation"[rotated] by blast
    AOT_hence ‹∀β(φ{β} ≡ β = α)›
      using "term-out:3" "≡E" by blast
    AOT_thus ‹∃α∀β(φ{β} ≡ β = α)›
      using "∃I" by fast
next
    AOT_assume ‹∃α∀β(φ{β} ≡ β = α)›
    then AOT_obtain α where ‹∀β (φ{β} ≡ β = α)›
      using "instantiation"[rotated] by blast
    AOT_hence ‹φ{α} & ∀β (φ{β} → β = α)›
      using "term-out:3" "≡E" by blast
    AOT_hence ‹∃α (φ{α} & ∀β (φ{β} → β = α))›
      using "∃I" by fast
    AOT_thus ‹∃!α φ{α}›
      using "uniqueness:1" "≡dfI" by blast
qed

AOT_theorem "uni-most": ‹∃!α φ{α} → ∀β∀γ((φ{β} & φ{γ}) → β = γ)›          (128)
proof(rule "→I"; rule GEN; rule GEN; rule "→I")
  fix β γ
  AOT_assume ‹∃!α φ{α}›
  AOT_hence ‹∃α∀β(φ{β} ≡ β = α)›
    using "uniqueness:2" "≡E" by blast
  then AOT_obtain α where ‹∀β(φ{β} ≡ β = α)›
    using "instantiation"[rotated] by blast
  moreover AOT_assume ‹φ{β} & φ{γ}›
  ultimately AOT_have ‹β = α› and ‹γ = α›
    using "∀E"(2) "&E" "≡E"(1,2) by blast+
  AOT_thus ‹β = γ›
    by (metis "rule=E" "id-eq:2" "→E")
qed

AOT_theorem "nec-exist-!": ‹∀α(φ{α} → □φ{α}) → (∃!α φ{α} → ∃!α □φ{α})›          (129)
```

```
proof (rule "→I"; rule "→I")
  AOT_assume a: ‹∀α(φ{α} → □φ{α})›
  AOT_assume ‹∃!α φ{α}›
  AOT_hence ‹∃α (φ{α} & ∀β (φ{β} → β = α))›
    using "uniqueness:1" "≡dfE" by blast
  then AOT_obtain α where ξ: ‹φ{α} & ∀β (φ{β} → β = α)›
    using "instantiation"[rotated] by blast
  AOT_have ‹□φ{α}›
    using ξ a "&E" "∀E" "→E" by fast
  moreover AOT_have ‹∀β (□φ{β} → β = α)›
    apply (rule GEN; rule "→I")
    using ξ[THEN "&E"(2), THEN "∀E"(2), THEN "→E"]
          "qml:2"[axiom_inst, THEN "→E"] by blast
  ultimately AOT_have ‹(□φ{α} & ∀β (□φ{β} → β = α))›
    using "&I" by blast
  AOT_thus ‹∃!α □φ{α}›
    using "uniqueness:1" "≡dfI" "∃I" by fast
qed

subsection‹The Theory of Actuality and Descriptions›
text‹\label{PLM: 9.8}›

AOT_theorem "act-cond": ‹𝒜(φ → ψ) → (𝒜φ → 𝒜ψ)›                    (130)
  using "→I" "≡E"(1) "logic-actual-nec:2"[axiom_inst] by blast

AOT_theorem "nec-imp-act": ‹□φ → 𝒜φ›                                (131)
  by (metis "act-cond" "contraposition:1[2]" "≡E"(4)
            "qml:2"[THEN act_closure, axiom_inst]
            "qml-act:2"[axiom_inst] RAA(1) "→E" "→I")

AOT_theorem "act-conj-act:1": ‹𝒜(𝒜φ → φ)›                          (132.1)
  using "→I" "≡E"(2) "logic-actual-nec:2"[axiom_inst]
        "logic-actual-nec:4"[axiom_inst] by blast

AOT_theorem "act-conj-act:2": ‹𝒜(φ → 𝒜φ)›                          (132.2)
  by (metis "→I" "≡E"(2, 4) "logic-actual-nec:2"[axiom_inst]
            "logic-actual-nec:4"[axiom_inst] RAA(1))

AOT_theorem "act-conj-act:3": ‹(𝒜φ & 𝒜ψ) → 𝒜(φ & ψ)›              (132.3)
proof -
  AOT_have ‹□(φ → (ψ → (φ & ψ)))›
    by (rule RN) (fact Adjunction)
  AOT_hence ‹𝒜(φ → (ψ → (φ & ψ)))›
    using "nec-imp-act" "→E" by blast
  AOT_hence ‹𝒜φ → 𝒜(ψ → (φ & ψ))›
    using "act-cond" "→E" by blast
  moreover AOT_have ‹𝒜(ψ → (φ & ψ)) → (𝒜ψ → 𝒜(φ & ψ))›
    by (fact "act-cond")
  ultimately AOT_have ‹𝒜φ → (𝒜ψ → 𝒜(φ & ψ))›
    using "→I" "→E" by metis
  AOT_thus ‹(𝒜φ & 𝒜ψ) → 𝒜(φ & ψ)›
    by (metis Importation "→E")
qed

AOT_theorem "act-conj-act:4": ‹𝒜(𝒜φ ≡ φ)›                          (132.4)
proof -
  AOT_have ‹(𝒜(𝒜φ → φ) & 𝒜(φ → 𝒜φ)) → 𝒜((𝒜φ → φ) & (φ → 𝒜φ))›
    by (fact "act-conj-act:3")
  moreover AOT_have ‹𝒜(𝒜φ → φ) & 𝒜(φ → 𝒜φ)›
    using "&I" "act-conj-act:1" "act-conj-act:2" by simp
  ultimately AOT_have ζ: ‹𝒜((𝒜φ → φ) & (φ → 𝒜φ))›
    using "→E" by blast
  AOT_have ‹𝒜(((𝒜φ → φ) & (φ → 𝒜φ)) → (𝒜φ ≡ φ))›
```

```
    using "conventions:3"[THEN "df-rules-formulas[2]",
                          THEN act_closure, axiom_inst] by blast
  AOT_hence ‹𝒜((𝒜φ → φ) & (φ → 𝒜φ)) → 𝒜(𝒜φ ≡ φ)›
    using "act-cond" "→E" by blast
  AOT_thus ‹𝒜(𝒜φ ≡ φ)› using ζ "→E" by blast
qed

(* TODO: Consider introducing AOT_inductive. *)
inductive arbitrary_actualization for φ where
  ‹arbitrary_actualization φ «𝒜φ»›
| ‹arbitrary_actualization φ «𝒜ψ»› if ‹arbitrary_actualization φ ψ›
declare arbitrary_actualization.cases[AOT]
        arbitrary_actualization.induct[AOT]
        arbitrary_actualization.simps[AOT]
        arbitrary_actualization.intros[AOT]
syntax arbitrary_actualization :: ‹φ' ⇒ φ' ⇒ AOT_prop›
  ("ARBITRARY'_ACTUALIZATION'(_,_')")

notepad
begin
  AOT_modally_strict {
    fix φ
    AOT_have ‹ARBITRARY_ACTUALIZATION(𝒜φ ≡ φ, 𝒜(𝒜φ ≡ φ))›
      using AOT_PLM.arbitrary_actualization.intros by metis
    AOT_have ‹ARBITRARY_ACTUALIZATION(𝒜φ ≡ φ, 𝒜𝒜(𝒜φ ≡ φ))›
      using AOT_PLM.arbitrary_actualization.intros by metis
    AOT_have ‹ARBITRARY_ACTUALIZATION(𝒜φ ≡ φ, 𝒜𝒜𝒜(𝒜φ ≡ φ))›
      using AOT_PLM.arbitrary_actualization.intros by metis
  }
end


AOT_theorem "closure-act:1":                                        (133.1)
  assumes ‹ARBITRARY_ACTUALIZATION(𝒜φ ≡ φ, ψ)›
  shows ‹ψ›
using assms proof(induct)
  case 1
  AOT_show ‹𝒜(𝒜φ ≡ φ)›
    by (simp add: "act-conj-act:4")
next
  case (2 ψ)
  AOT_thus ‹𝒜ψ›
    by (metis arbitrary_actualization.simps "≡E"(1)
              "logic-actual-nec:4"[axiom_inst])
qed

AOT_theorem "closure-act:2": ‹∀α 𝒜(𝒜φ{α} ≡ φ{α})›                  (133.2)
  by (simp add: "act-conj-act:4" "∀I")

AOT_theorem "closure-act:3": ‹𝒜∀α 𝒜(𝒜φ{α} ≡ φ{α})›                 (133.3)
  by (metis (no_types, lifting) "act-conj-act:4" "≡E"(1,2) "∀I"
            "logic-actual-nec:3"[axiom_inst]
            "logic-actual-nec:4"[axiom_inst])

AOT_theorem "closure-act:4": ‹𝒜∀α₁...∀αₙ 𝒜(𝒜φ{α₁...αₙ} ≡ φ{α₁...αₙ})›  (133.4)
  using "closure-act:3" .

AOT_act_theorem "RA[1]":                                            (134)
  assumes ‹⊢ φ›
  shows ‹⊢ 𝒜φ›
  - ‹While this proof is rejected in PLM,
    we merely state it as modally-fragile rule,
    which addresses the concern in PLM.›
```

```
  using "¬¬E" assms "≡E"(3) "logic-actual"[act_axiom_inst]
        "logic-actual-nec:1"[axiom_inst] "modus-tollens:2" by blast
AOT_theorem "RA[2]":                                                    (134)
  assumes ‹⊢□ φ›
  shows ‹⊢□ 𝒜φ›
  — ‹This rule is in fact a consequence of RN and
     does not require an appeal to the semantics itself.›
  using RN assms "nec-imp-act" "vdash-properties:5" by blast
AOT_theorem "RA[3]":
  assumes ‹Γ ⊢□ φ›
  shows ‹𝒜Γ ⊢□ 𝒜φ›
  text‹This rule is only derivable from the semantics,
       but apparently no proof actually relies on it.
       If this turns out to be required, it is valid to derive it from the
       semantics just like RN, but we refrain from doing so, unless necessary.›
  (*  using assms by (meson AOT_sem_act imageI) *)
  oops — ‹discard the rule›

AOT_act_theorem "ANeg:1": ‹¬𝒜φ ≡ ¬φ›                                    (137.1)
  by (simp add: "RA[1]" "contraposition:1[1]" "deduction-theorem"
                "≡I" "logic-actual"[act_axiom_inst])

AOT_act_theorem "ANeg:2": ‹¬𝒜¬φ ≡ φ›                                    (137.2)
  using "ANeg:1" "≡I" "≡E"(5) "useful-tautologies:1"
        "useful-tautologies:2" by blast

AOT_theorem "Act-Basic:1": ‹𝒜φ ∨ 𝒜¬φ›                                  (138.1)
  by (meson "∨I"(1,2) "≡E"(2) "logic-actual-nec:1"[axiom_inst] "raa-cor:1")

AOT_theorem "Act-Basic:2": ‹𝒜(φ & ψ) ≡ (𝒜φ & 𝒜ψ)›                      (138.2)
proof (rule "≡I"; rule "→I")
  AOT_assume ‹𝒜(φ & ψ)›
  moreover AOT_have ‹𝒜((φ & ψ) → φ)›
    by (simp add: "RA[2]" "Conjunction Simplification"(1))
  moreover AOT_have ‹𝒜((φ & ψ) → ψ)›
    by (simp add: "RA[2]" "Conjunction Simplification"(2))
  ultimately AOT_show ‹𝒜φ & 𝒜ψ›
    using "act-cond"[THEN "→E", THEN "→E"] "&I" by metis
next
  AOT_assume ‹𝒜φ & 𝒜ψ›
  AOT_thus ‹𝒜(φ & ψ)›
    using "act-conj-act:3" "vdash-properties:6" by blast
qed

AOT_theorem "Act-Basic:3": ‹𝒜(φ ≡ ψ) ≡ (𝒜(φ → ψ) & 𝒜(ψ → φ))›          (138.3)
proof (rule "≡I"; rule "→I")
  AOT_assume ‹𝒜(φ ≡ ψ)›
  moreover AOT_have ‹𝒜((φ ≡ ψ) → (φ → ψ))›
    by (simp add: "RA[2]" "deduction-theorem" "≡E"(1))
  moreover AOT_have ‹𝒜((φ ≡ ψ) → (ψ → φ))›
    by (simp add: "RA[2]" "deduction-theorem" "≡E"(2))
  ultimately AOT_show ‹𝒜(φ → ψ) & 𝒜(ψ → φ)›
    using "act-cond"[THEN "→E", THEN "→E"] "&I" by metis
next
  AOT_assume ‹𝒜(φ → ψ) & 𝒜(ψ → φ)›
  AOT_hence ‹𝒜((φ → ψ) & (ψ → φ))›
    by (metis "act-conj-act:3" "vdash-properties:10")
  moreover AOT_have ‹𝒜(((φ → ψ) & (ψ → φ)) → (φ ≡ ψ))›
    by (simp add: "conventions:3" "RA[2]" "df-rules-formulas[2]"
                  "vdash-properties:1[2]")
  ultimately AOT_show ‹𝒜(φ ≡ ψ)›
    using "act-cond"[THEN "→E", THEN "→E"] by metis
qed
```

```
AOT_theorem "Act-Basic:4": ‹(𝒜(φ → ψ) & 𝒜(ψ → φ)) ≡ (𝒜φ ≡ 𝒜ψ)›    (138.4)
proof (rule "≡I"; rule "→I")
  AOT_assume 0: ‹𝒜(φ → ψ) & 𝒜(ψ → φ)›
  AOT_show ‹𝒜φ ≡ 𝒜ψ›
    using 0 "&E" "act-cond"[THEN "→E", THEN "→E"] "≡I" "→I" by metis
next
  AOT_assume ‹𝒜φ ≡ 𝒜ψ›
  AOT_thus ‹𝒜(φ → ψ) & 𝒜(ψ → φ)›
    by (metis "→I" "logic-actual-nec:2"[axiom_inst] "≡E"(1,2) "&I")
qed

AOT_theorem "Act-Basic:5": ‹𝒜(φ ≡ ψ) ≡ (𝒜φ ≡ 𝒜ψ)›    (138.5)
  using "Act-Basic:3" "Act-Basic:4" "≡E"(5) by blast

AOT_theorem "Act-Basic:6": ‹𝒜φ ≡ □𝒜φ›    (138.6)
  by (simp add: "≡I" "qml:2"[axiom_inst] "qml-act:1"[axiom_inst])

AOT_theorem "Act-Basic:7": ‹𝒜□φ → □𝒜φ›    (138.7)
  by (metis "Act-Basic:6" "→I" "→E" "≡E"(1,2) "nec-imp-act"
            "qml-act:2"[axiom_inst])

AOT_theorem "Act-Basic:8": ‹□φ → □𝒜φ›    (138.8)
  using "Hypothetical Syllogism" "nec-imp-act" "qml-act:1"[axiom_inst] by blast

AOT_theorem "Act-Basic:9": ‹𝒜(φ ∨ ψ) ≡ (𝒜φ ∨ 𝒜ψ)›    (138.9)
proof (rule "≡I"; rule "→I")
  AOT_assume ‹𝒜(φ ∨ ψ)›
  AOT_thus ‹𝒜φ ∨ 𝒜ψ›
  proof (rule "raa-cor:3")
    AOT_assume ‹¬(𝒜φ ∨ 𝒜ψ)›
    AOT_hence ‹¬𝒜φ & ¬𝒜ψ›
      by (metis "≡E"(1) "oth-class-taut:5:d")
    AOT_hence ‹𝒜¬φ & 𝒜¬ψ›
      using "logic-actual-nec:1"[axiom_inst, THEN "≡E"(2)] "&E" "&I" by metis
    AOT_hence ‹𝒜(¬φ & ¬ψ)›
      using "≡E" "Act-Basic:2" by metis
    moreover AOT_have ‹𝒜((¬φ & ¬ψ) ≡ ¬(φ ∨ ψ))›
      using "RA[2]" "≡E"(6) "oth-class-taut:3:a" "oth-class-taut:5:d" by blast
    moreover AOT_have ‹𝒜(¬φ & ¬ψ) ≡ 𝒜(¬(φ ∨ ψ))›
      using calculation(2) by (metis "Act-Basic:5" "≡E"(1))
    ultimately AOT_have ‹𝒜(¬(φ ∨ ψ))› using "≡E" by blast
    AOT_thus ‹¬𝒜(φ ∨ ψ)›
      using "logic-actual-nec:1"[axiom_inst, THEN "≡E"(1)] by auto
  qed
next
  AOT_assume ‹𝒜φ ∨ 𝒜ψ›
  AOT_thus ‹𝒜(φ ∨ ψ)›
    by (meson "RA[2]" "act-cond" "∨I"(1) "∨E"(1) "Disjunction Addition"(1,2))
qed

AOT_theorem "Act-Basic:10": ‹𝒜∃α φ{α} ≡ ∃α 𝒜φ{α}›    (138.10)
proof -
  AOT_have ϑ: ‹¬𝒜∀α ¬φ{α} ≡ ¬∀α 𝒜¬φ{α}›
    by (rule "oth-class-taut:4:b"[THEN "≡E"(1)])
       (metis "logic-actual-nec:3"[axiom_inst])
  AOT_have ξ: ‹¬∀α 𝒜¬φ{α} ≡ ¬∀α ¬𝒜φ{α}›
    by (rule "oth-class-taut:4:b"[THEN "≡E"(1)])
       (rule "logic-actual-nec:1"[THEN universal_closure,
              axiom_inst, THEN "cqt-basic:3"[THEN "→E"]])
  AOT_have ‹𝒜(∃α φ{α}) ≡ 𝒜(¬∀α ¬φ{α})›
    using "conventions:4"[THEN "df-rules-formulas[1]",
                          THEN act_closure, axiom_inst]
```

```
        "conventions:4"[THEN "df-rules-formulas[2]",
                        THEN act_closure, axiom_inst]
     "Act-Basic:4"[THEN "≡E"(1)] "&I" "Act-Basic:5"[THEN "≡E"(2)] by metis
  also AOT_have ‹... ≡ ¬𝓐∀α ¬φ{α}›
     by (simp add: "logic-actual-nec:1" "vdash-properties:1[2]")
  also AOT_have ‹... ≡ ¬∀α 𝓐 ¬φ{α}› using ϑ by blast
  also AOT_have ‹... ≡ ¬∀α ¬𝓐 φ{α}› using ξ by blast
  also AOT_have ‹... ≡ ∃α 𝓐 φ{α}›
     using "conventions:4"[THEN "≡Df"] by (metis "≡E"(6) "oth-class-taut:3:a")
  finally AOT_show ‹𝓐∃α φ{α} ≡ ∃α 𝓐φ{α}› .
qed


AOT_theorem "Act-Basic:11":                                                    (138.11)
  ‹𝓐∀α(φ{α} ≡ ψ{α}) ≡ ∀α(𝓐φ{α} ≡ 𝓐ψ{α})›
proof(rule "≡I"; rule "→I")
  AOT_assume ‹𝓐∀α(φ{α} ≡ ψ{α})›
  AOT_hence ‹∀α𝓐(φ{α} ≡ ψ{α})›
     using "logic-actual-nec:3"[axiom_inst, THEN "≡E"(1)] by blast
  AOT_hence ‹𝓐(φ{α} ≡ ψ{α})› for α using "∀E" by blast
  AOT_hence ‹𝓐φ{α} ≡ 𝓐ψ{α}› for α by (metis "Act-Basic:5" "≡E"(1))
  AOT_thus ‹∀α(𝓐φ{α} ≡ 𝓐ψ{α})› by (rule "∀I")
next
  AOT_assume ‹∀α(𝓐φ{α} ≡ 𝓐ψ{α})›
  AOT_hence ‹𝓐φ{α} ≡ 𝓐ψ{α}› for α using "∀E" by blast
  AOT_hence ‹𝓐(φ{α} ≡ ψ{α})› for α by (metis "Act-Basic:5" "≡E"(2))
  AOT_hence ‹∀α 𝓐(φ{α} ≡ ψ{α})› by (rule "∀I")
  AOT_thus ‹𝓐∀α(φ{α} ≡ ψ{α})›
     using "logic-actual-nec:3"[axiom_inst, THEN "≡E"(2)] by fast
qed

AOT_act_theorem "act-quant-uniq":                                              (139)
  ‹∀β(𝓐φ{β} ≡ β = α) ≡ ∀β(φ{β} ≡ β = α)›
proof(rule "≡I"; rule "→I")
  AOT_assume ‹∀β(𝓐φ{β} ≡ β = α)›
  AOT_hence ‹𝓐φ{β} ≡ β = α› for β using "∀E" by blast
  AOT_hence ‹φ{β} ≡ β = α› for β
     using "≡I" "→I" "RA[1]" "≡E"(1,2) "logic-actual"[act_axiom_inst] "→E"
     by metis
  AOT_thus ‹∀β(φ{β} ≡ β = α)› by (rule "∀I")
next
  AOT_assume ‹∀β(φ{β} ≡ β = α)›
  AOT_hence ‹φ{β} ≡ β = α› for β using "∀E" by blast
  AOT_hence ‹𝓐φ{β} ≡ β = α› for β
     using "≡I" "→I" "RA[1]" "≡E"(1,2) "logic-actual"[act_axiom_inst] "→E"
     by metis
  AOT_thus ‹∀β(𝓐φ{β} ≡ β = α)› by (rule "∀I")
qed

AOT_act_theorem "fund-cont-desc": ‹x = ιx(φ{x}) ≡ ∀z(φ{z} ≡ z = x)›           (140)
  using descriptions[axiom_inst] "act-quant-uniq" "≡E"(5) by fast

AOT_act_theorem hintikka: ‹x = ιx(φ{x}) ≡ (φ{x} & ∀z (φ{z} → z = x))›        (141)
  using "Commutativity of ≡"[THEN "≡E"(1)] "term-out:3"
        "fund-cont-desc" "≡E"(5) by blast


locale russell_axiom =
  fixes ψ
  assumes ψ_denotes_asm: "[v ⊨ ψ{κ}] ⟹ [v ⊨ κ↓]"
begin
AOT_act_theorem "russell-axiom":                                               (142)
  ‹ψ{ιx φ{x}} ≡ ∃x(φ{x} & ∀z(φ{z} → z = x) & ψ{x})›
```

```
proof -
  AOT_have b: ‹∀x (x = ιx φ{x} ≡ (φ{x} & ∀z(φ{z} → z = x)))›
    using hintikka "∀I" by fast
  show ?thesis
  proof(rule "≡I"; rule "→I")
    AOT_assume c: ‹ψ{ιx φ{x}}›
    AOT_hence d: ‹ιx φ{x}↓›
      using ψ_denotes_asm by blast
    AOT_hence ‹∃y (y = ιx φ{x})›
      by (metis "rule=I:1" "existential:1")
    then AOT_obtain a where a_def: ‹a = ιx φ{x}›
      using "instantiation"[rotated] by blast
    moreover AOT_have ‹a = ιx φ{x} ≡ (φ{a} & ∀z(φ{z} → z = a))›
      using b "∀E" by blast
    ultimately AOT_have ‹φ{a} & ∀z(φ{z} → z = a)›
      using "≡E" by blast
    moreover AOT_have ‹ψ{a}›
    proof -
      AOT_have 1: ‹∀x∀y(x = y → y = x)›
        by (simp add: "id-eq:2" "universal-cor")
      AOT_have ‹a = ιx φ{x} →  ιx φ{x} = a›
        by (rule "∀E"(1)[where τ="«ιx φ{x}»"]; rule "∀E"(2)[where β=a])
           (auto simp: 1 d "universal-cor")
      AOT_thus ‹ψ{a}›
        using a_def c "rule=E" "→E" by blast
    qed
    ultimately AOT_have ‹φ{a} & ∀z(φ{z} → z = a) & ψ{a}› by (rule "&I")
    AOT_thus ‹∃x(φ{x} & ∀z(φ{z} → z = x) & ψ{x})› by (rule "∃I")
  next
    AOT_assume ‹∃x(φ{x} & ∀z(φ{z} → z = x) & ψ{x})›
    then AOT_obtain b where g: ‹φ{b} & ∀z(φ{z} → z = b) & ψ{b}›
      using "instantiation"[rotated] by blast
    AOT_hence h: ‹b = ιx φ{x} ≡ (φ{b} & ∀z(φ{z} → z = b))›
      using b "∀E" by blast
    AOT_have ‹φ{b} & ∀z(φ{z} → z = b)› and j: ‹ψ{b}›
      using g "&E" by blast+
    AOT_hence ‹b = ιx φ{x}› using h "≡E" by blast
    AOT_thus ‹ψ{ιx φ{x}}› using j "rule=E" by blast
  qed
qed
end

interpretation "russell-axiom[exe,1]": russell_axiom ‹λ κ . «[Π]κ»›
  by standard (metis "cqt:5:a[1]"[axiom_inst, THEN "→E"] "&E"(2))
interpretation "russell-axiom[exe,2,1,1]": russell_axiom ‹λ κ . «[Π]κκ'»›
  by standard (metis "cqt:5:a[2]"[axiom_inst, THEN "→E"] "&E")
interpretation "russell-axiom[exe,2,1,2]": russell_axiom ‹λ κ . «[Π]κ'κ»›
  by standard (metis "cqt:5:a[2]"[axiom_inst, THEN "→E"] "&E"(2))
interpretation "russell-axiom[exe,2,2]": russell_axiom ‹λ κ . «[Π]κκ»›
  by standard (metis "cqt:5:a[2]"[axiom_inst, THEN "→E"] "&E"(2))
interpretation "russell-axiom[exe,3,1,1]": russell_axiom ‹λ κ . «[Π]κκ'κ''»›
  by standard (metis "cqt:5:a[3]"[axiom_inst, THEN "→E"] "&E")
interpretation "russell-axiom[exe,3,1,2]": russell_axiom ‹λ κ . «[Π]κ'κκ''»›
  by standard (metis "cqt:5:a[3]"[axiom_inst, THEN "→E"] "&E")
interpretation "russell-axiom[exe,3,1,3]": russell_axiom ‹λ κ . «[Π]κ'κ''κ»›
  by standard (metis "cqt:5:a[3]"[axiom_inst, THEN "→E"] "&E"(2))
interpretation "russell-axiom[exe,3,2,1]": russell_axiom ‹λ κ . «[Π]κκκ'»›
  by standard (metis "cqt:5:a[3]"[axiom_inst, THEN "→E"] "&E")
interpretation "russell-axiom[exe,3,2,2]": russell_axiom ‹λ κ . «[Π]κκ'κ»›
  by standard (metis "cqt:5:a[3]"[axiom_inst, THEN "→E"] "&E"(2))
interpretation "russell-axiom[exe,3,2,3]": russell_axiom ‹λ κ . «[Π]κ'κκ»›
  by standard (metis "cqt:5:a[3]"[axiom_inst, THEN "→E"] "&E"(2))
interpretation "russell-axiom[exe,3,3]": russell_axiom ‹λ κ . «[Π]κκκ»›
```

```
  by standard (metis "cqt:5:a[3]"[axiom_inst, THEN "→E"] "&E"(2))

interpretation "russell-axiom[enc,1]": russell_axiom ‹λ κ . «κ[Π]»›
  by standard (metis "cqt:5:b[1]"[axiom_inst, THEN "→E"] "&E"(2))
interpretation "russell-axiom[enc,2,1]": russell_axiom ‹λ κ . «κκ'[Π]»›
  by standard (metis "cqt:5:b[2]"[axiom_inst, THEN "→E"] "&E")
interpretation "russell-axiom[enc,2,2]": russell_axiom ‹λ κ . «κ'κ[Π]»›
  by standard (metis "cqt:5:b[2]"[axiom_inst, THEN "→E"] "&E"(2))
interpretation "russell-axiom[enc,2,3]": russell_axiom ‹λ κ . «κκ[Π]»›
  by standard (metis "cqt:5:b[2]"[axiom_inst, THEN "→E"] "&E"(2))
interpretation "russell-axiom[enc,3,1,1]": russell_axiom ‹λ κ . «κκ'κ''[Π]»›
  by standard (metis "cqt:5:b[3]"[axiom_inst, THEN "→E"] "&E")
interpretation "russell-axiom[enc,3,1,2]": russell_axiom ‹λ κ . «κ'κκ''[Π]»›
  by standard (metis "cqt:5:b[3]"[axiom_inst, THEN "→E"] "&E")
interpretation "russell-axiom[enc,3,1,3]": russell_axiom ‹λ κ . «κ'κ''κ[Π]»›
  by standard (metis "cqt:5:b[3]"[axiom_inst, THEN "→E"] "&E"(2))
interpretation "russell-axiom[enc,3,2,1]": russell_axiom ‹λ κ . «κκκ'[Π]»›
  by standard (metis "cqt:5:b[3]"[axiom_inst, THEN "→E"] "&E")
interpretation "russell-axiom[enc,3,2,2]": russell_axiom ‹λ κ . «κκ'κ[Π]»›
  by standard (metis "cqt:5:b[3]"[axiom_inst, THEN "→E"] "&E"(2))
interpretation "russell-axiom[enc,3,2,3]": russell_axiom ‹λ κ . «κ'κκ[Π]»›
  by standard (metis "cqt:5:b[3]"[axiom_inst, THEN "→E"] "&E"(2))
interpretation "russell-axiom[enc,3,3]": russell_axiom ‹λ κ . «κκκ[Π]»›
  by standard (metis "cqt:5:b[3]"[axiom_inst, THEN "→E"] "&E"(2))

AOT_act_theorem "!-exists:1": ‹ιx φ{x}↓ ≡ ∃!x φ{x}›                    (143.1)
proof(rule "≡I"; rule "→I")
  AOT_assume ‹ιx φ{x}↓›
  AOT_hence ‹∃y (y = ιx φ{x})› by (metis "rule=I:1" "existential:1")
  then AOT_obtain a where ‹a = ιx φ{x}›
    using "instantiation"[rotated] by blast
  AOT_hence ‹φ{a} & ∀z (φ{z} → z = a)›
    using hintikka "≡E" by blast
  AOT_hence ‹∃x (φ{x} & ∀z (φ{z} → z = x))›
    by (rule "∃I")
  AOT_thus ‹∃!x φ{x}›
    using "uniqueness:1"[THEN "≡dfI"] by blast
next
  AOT_assume ‹∃!x φ{x}›
  AOT_hence ‹∃x (φ{x} & ∀z (φ{z} → z = x))›
    using "uniqueness:1"[THEN "≡dfE"] by blast
  then AOT_obtain b where ‹φ{b} & ∀z (φ{z} → z = b)›
    using "instantiation"[rotated] by blast
  AOT_hence ‹b = ιx φ{x}›
    using hintikka "≡E" by blast
  AOT_thus ‹ιx φ{x}↓›
    by (metis "t=t-proper:2" "vdash-properties:6")
qed

AOT_act_theorem "!-exists:2": ‹∃y(y=ιx φ{x}) ≡ ∃!x φ{x}›                (143.2)
  using "!-exists:1" "free-thms:1" "≡E"(6) by blast

AOT_act_theorem "y-in:1": ‹x = ιx φ{x} → φ{x}›                         (144.1)
  using "&E"(1) "→I" hintikka "≡E"(1) by blast

(* Note: generalized alphabetic variant of the last theorem *)
AOT_act_theorem "y-in:2": ‹z = ιx φ{x} → φ{z}› using "y-in:1".         (144.2)

AOT_act_theorem "y-in:3": ‹ιx φ{x}↓ → φ{ιx φ{x}}›                      (144.3)
proof(rule "→I")
  AOT_assume ‹ιx φ{x}↓›
  AOT_hence ‹∃y (y = ιx φ{x})›
    by (metis "rule=I:1" "existential:1")
```

```
  then AOT_obtain a where ‹a = ιx φ{x}›
    using "instantiation"[rotated] by blast
  moreover AOT_have ‹φ{a}›
    using calculation hintikka "≡E"(1) "&E" by blast
  ultimately AOT_show ‹φ{ιx φ{x}}› using "rule=E" by blast
qed

AOT_act_theorem "y-in:4": ‹∃y (y = ιx φ{x}) → φ{ιx φ{x}}›                (144.4)
  using "y-in:3"[THEN "→E"] "free-thms:1"[THEN "≡E"(2)] "→I" by blast


AOT_theorem "act-quant-nec":                                              (145)
  ‹∀β (𝒜φ{β} ≡ β = α) ≡ ∀β(𝒜𝒜φ{β} ≡ β = α)›
proof(rule "≡I"; rule "→I")
  AOT_assume ‹∀β (𝒜φ{β} ≡ β = α)›
  AOT_hence ‹𝒜φ{β} ≡ β = α› for β using "∀E" by blast
  AOT_hence ‹𝒜𝒜φ{β} ≡ β = α› for β
    by (metis "Act-Basic:5" "act-conj-act:4" "≡E"(1) "≡E"(5))
  AOT_thus ‹∀β(𝒜𝒜φ{β} ≡ β = α)›
    by (rule "∀I")
next
  AOT_assume ‹∀β(𝒜𝒜φ{β} ≡ β = α)›
  AOT_hence ‹𝒜𝒜φ{β} ≡ β = α› for β using "∀E" by blast
  AOT_hence ‹𝒜φ{β} ≡ β = α› for β
    by (metis "Act-Basic:5" "act-conj-act:4" "≡E"(1) "≡E"(6))
  AOT_thus ‹∀β (𝒜φ{β} ≡ β = α)›
    by (rule "∀I")
qed

AOT_theorem "equi-desc-descA:1": ‹x = ιx φ{x} ≡ x = ιx(𝒜φ{x})›          (146.1)
proof -
  AOT_have ‹x = ιx φ{x} ≡ ∀z (𝒜φ{z} ≡ z = x)›
    using descriptions[axiom_inst] by blast
  also AOT_have ‹... ≡ ∀z (𝒜𝒜φ{z} ≡ z = x)›
  proof(rule "≡I"; rule "→I"; rule "∀I")
    AOT_assume ‹∀z (𝒜φ{z} ≡ z = x)›
    AOT_hence ‹𝒜φ{a} ≡ a = x› for a
      using "∀E" by blast
    AOT_thus ‹𝒜𝒜φ{a} ≡ a = x› for a
      by (metis "Act-Basic:5" "act-conj-act:4" "≡E"(1) "≡E"(5))
  next
    AOT_assume ‹∀z (𝒜𝒜φ{z} ≡ z = x)›
    AOT_hence ‹𝒜𝒜φ{a} ≡ a = x› for a
      using "∀E" by blast
    AOT_thus ‹𝒜φ{a} ≡ a = x› for a
      by (metis "Act-Basic:5" "act-conj-act:4" "≡E"(1) "≡E"(6))
  qed
  also AOT_have ‹... ≡ x = ιx(𝒜φ{x})›
    using "Commutativity of ≡"[THEN "≡E"(1)] descriptions[axiom_inst] by fast
  finally show ?thesis .
qed

AOT_theorem "equi-desc-descA:2": ‹ιx φ{x}↓ → ιx φ{x} = ιx(𝒜φ{x})›        (146.2)
proof(rule "→I")
  AOT_assume ‹ιx φ{x}↓›
  AOT_hence ‹∃y (y = ιx φ{x})›
    by (metis "rule=I:1" "existential:1")
  then AOT_obtain a where ‹a = ιx φ{x}›
    using "instantiation"[rotated] by blast
  moreover AOT_have ‹a = ιx(𝒜φ{x})›
    using calculation "equi-desc-descA:1"[THEN "≡E"(1)] by blast
  ultimately AOT_show ‹ιx φ{x} = ιx(𝒜φ{x})›
    using "rule=E" by fast
```

```
qed

AOT_theorem "nec-hintikka-scheme":                                        (147)
  ‹x = ιx φ{x} ≡ 𝒜φ{x} & ∀z(𝒜φ{z} → z = x)›
proof -
  AOT_have ‹x = ιx φ{x} ≡ ∀z(𝒜φ{z} ≡ z = x)›
    using descriptions[axiom_inst] by blast
  also AOT_have ‹... ≡ (𝒜φ{x} & ∀z(𝒜φ{z} → z = x))›
    using "Commutativity of ≡"[THEN "≡E"(1)] "term-out:3" by fast
  finally show ?thesis.
qed

AOT_theorem "equiv-desc-eq:1":                                            (148.1)
  ‹𝒜∀x(φ{x} ≡ ψ{x}) → ∀x (x = ιx φ{x} ≡ x = ιx ψ{x})›
proof(rule "→I"; rule "∀I")
  fix β
  AOT_assume ‹𝒜∀x(φ{x} ≡ ψ{x})›
  AOT_hence ‹𝒜(φ{x} ≡ ψ{x})› for x
    using "logic-actual-nec:3"[axiom_inst, THEN "≡E"(1)] "∀E"(2) by blast
  AOT_hence 0: ‹𝒜φ{x} ≡ 𝒜ψ{x}› for x
    by (metis "Act-Basic:5" "≡E"(1))
  AOT_have ‹β = ιx φ{x} ≡ 𝒜φ{β} & ∀z(𝒜φ{z} → z = β)›
    using "nec-hintikka-scheme" by blast
  also AOT_have ‹... ≡ 𝒜ψ{β} & ∀z(𝒜ψ{z} → z = β)›
  proof (rule "≡I"; rule "→I")
    AOT_assume 1: ‹𝒜φ{β} & ∀z(𝒜φ{z} → z = β)›
    AOT_hence ‹𝒜φ{z} → z = β› for z
      using "&E" "∀E" by blast
    AOT_hence ‹𝒜ψ{z} → z = β› for z
      using 0 "≡E" "→I" "→E" by metis
    AOT_hence ‹∀z(𝒜ψ{z} → z = β)›
      using "∀I" by fast
    moreover AOT_have ‹𝒜ψ{β}›
      using "&E" 0[THEN "≡E"(1)] 1 by blast
    ultimately AOT_show ‹𝒜ψ{β} & ∀z(𝒜ψ{z} → z = β)›
      using "&I" by blast
  next
    AOT_assume 1: ‹𝒜ψ{β} & ∀z(𝒜ψ{z} → z = β)›
    AOT_hence ‹𝒜ψ{z} → z = β› for z
      using "&E" "∀E" by blast
    AOT_hence ‹𝒜φ{z} → z = β› for z
      using 0 "≡E" "→I" "→E" by metis
    AOT_hence ‹∀z(𝒜φ{z} → z = β)›
      using "∀I" by fast
    moreover AOT_have ‹𝒜φ{β}›
      using "&E" 0[THEN "≡E"(2)] 1 by blast
    ultimately AOT_show ‹𝒜φ{β} & ∀z(𝒜φ{z} → z = β)›
      using "&I" by blast
  qed
  also AOT_have ‹... ≡ β = ιx ψ{x}›
    using "Commutativity of ≡"[THEN "≡E"(1)] "nec-hintikka-scheme" by blast
  finally AOT_show ‹β = ιx φ{x} ≡ β = ιx ψ{x}› .
qed

AOT_theorem "equiv-desc-eq:2":                                            (148.2)
  ‹ιx φ{x}↓ & 𝒜∀x(φ{x} ≡ ψ{x}) → ιx φ{x} = ιx ψ{x}›
proof(rule "→I")
  AOT_assume ‹ιx φ{x}↓ & 𝒜∀x(φ{x} ≡ ψ{x})›
  AOT_hence 0: ‹∃y (y = ιx φ{x})› and
            1: ‹∀x (x = ιx φ{x} ≡ x = ιx ψ{x})›
    using "&E" "free-thms:1"[THEN "≡E"(1)] "equiv-desc-eq:1" "→E" by blast+
  then AOT_obtain a where ‹a = ιx φ{x}›
    using "instantiation"[rotated] by blast
```

```
  moreover AOT_have ‹a = ιx ψ{x}›
    using calculation 1 "∀E" "≡E"(1) by fast
  ultimately AOT_show ‹ιx φ{x} = ιx ψ{x}›
    using "rule=E" by fast
qed

AOT_theorem "equiv-desc-eq:3":                                        (148.3)
  ‹ιx φ{x}↓ & □∀x(φ{x} ≡ ψ{x}) → ιx φ{x} = ιx ψ{x}›
  using "→I" "equiv-desc-eq:2"[THEN "→E", OF "&I"] "&E"
        "nec-imp-act"[THEN "→E"] by metis

(* Note: this is a special case of "exist-nec" *)
AOT_theorem "equiv-desc-eq:4": ‹ιx φ{x}↓ → □ιx φ{x}↓›                 (148.4)
proof(rule "→I")
  AOT_assume ‹ιx φ{x}↓›
  AOT_hence ‹∃y (y = ιx φ{x})›
    by (metis "rule=I:1" "existential:1")
  then AOT_obtain a where ‹a = ιx φ{x}›
    using "instantiation"[rotated] by blast
  AOT_thus ‹□ιx φ{x}↓›
    using "ex:2:a" "rule=E" by fast
qed

AOT_theorem "equiv-desc-eq:5": ‹ιx φ{x}↓ → ∃y □(y = ιx φ{x})›        (148.5)
proof(rule "→I")
  AOT_assume ‹ιx φ{x}↓›
  AOT_hence ‹∃y (y = ιx φ{x})›
    by (metis "rule=I:1" "existential:1")
  then AOT_obtain a where ‹a = ιx φ{x}›
    using "instantiation"[rotated] by blast
  AOT_hence ‹□(a = ιx φ{x})›
    by (metis "id-nec:2" "vdash-properties:10")
  AOT_thus ‹∃y □(y = ιx φ{x})›
    by (rule "∃I")
qed

AOT_act_theorem "equiv-desc-eq2:1":                                   (149.1)
  ‹∀x (φ{x} ≡ ψ{x}) → ∀x (x = ιx φ{x} ≡ x = ιx ψ{x})›
  using "→I" "logic-actual"[act_axiom_inst, THEN "→E"]
        "equiv-desc-eq:1"[THEN "→E"]
        "RA[1]" "deduction-theorem" by blast

AOT_act_theorem "equiv-desc-eq2:2":                                   (149.2)
  ‹ιx φ{x}↓ & ∀x (φ{x} ≡ ψ{x}) → ιx φ{x} = ιx ψ{x}›
  using "→I" "logic-actual"[act_axiom_inst, THEN "→E"]
        "equiv-desc-eq:2"[THEN "→E", OF "&I"]
        "RA[1]" "deduction-theorem" "&E" by metis

context russell_axiom
begin
AOT_theorem "nec-russell-axiom":                                      (150)
  ‹ψ{ιx φ{x}} ≡ ∃x(𝒜φ{x} & ∀z(𝒜φ{z} → z = x) & ψ{x})›
proof -
  AOT_have b: ‹∀x (x = ιx φ{x} ≡ (𝒜φ{x} & ∀z(𝒜φ{z} → z = x)))›
    using "nec-hintikka-scheme" "∀I" by fast
  show ?thesis
  proof(rule "≡I"; rule "→I")
    AOT_assume c: ‹ψ{ιx φ{x}}›
    AOT_hence d: ‹ιx φ{x}↓›
      using ψ_denotes_asm by blast
    AOT_hence ‹∃y (y = ιx φ{x})›
      by (metis "rule=I:1" "existential:1")
    then AOT_obtain a where a_def: ‹a = ιx φ{x}›
```

```
      using "instantiation"[rotated] by blast
    moreover AOT_have ‹a = ιx φ{x} ≡ (𝒜φ{a} & ∀z(𝒜φ{z} → z = a))›
      using b "∀E" by blast
    ultimately AOT_have ‹𝒜φ{a} & ∀z(𝒜φ{z} → z = a)›
      using "≡E" by blast
    moreover AOT_have ‹ψ{a}›
    proof -
      AOT_have 1: ‹∀x∀y(x = y → y = x)›
        by (simp add: "id-eq:2" "universal-cor")
      AOT_have ‹a = ιx φ{x} →  ιx φ{x} = a›
        by (rule "∀E"(1)[where τ="«ιx φ{x}»"]; rule "∀E"(2)[where β=a])
           (auto simp: d "universal-cor" 1)
      AOT_thus ‹ψ{a}›
        using a_def c "rule=E" "→E" by metis
    qed
    ultimately AOT_have ‹𝒜φ{a} & ∀z(𝒜φ{z} → z = a) & ψ{a}›
      by (rule "&I")
    AOT_thus ‹∃x(𝒜φ{x} & ∀z(𝒜φ{z} → z = x) & ψ{x})›
      by (rule "∃I")
  next
    AOT_assume ‹∃x(𝒜φ{x} & ∀z(𝒜φ{z} → z = x) & ψ{x})›
    then AOT_obtain b where g: ‹𝒜φ{b} & ∀z(𝒜φ{z} → z = b) & ψ{b}›
      using "instantiation"[rotated] by blast
    AOT_hence h: ‹b = ιx φ{x} ≡ (𝒜φ{b} & ∀z(𝒜φ{z} → z = b))›
      using b "∀E" by blast
    AOT_have ‹𝒜φ{b} & ∀z(𝒜φ{z} → z = b)› and j: ‹ψ{b}›
      using g "&E" by blast+
    AOT_hence ‹b = ιx φ{x}›
      using h "≡E" by blast
    AOT_thus ‹ψ{ιx φ{x}}›
      using j "rule=E" by blast
  qed
qed
end

AOT_theorem "actual-desc:1": ‹ιx φ{x}↓ ≡ ∃!x 𝒜φ{x}›                    (151.1)
proof (rule "≡I"; rule "→I")
  AOT_assume ‹ιx φ{x}↓›
  AOT_hence ‹∃y (y = ιx φ{x})›
    by (metis "rule=I:1" "existential:1")
  then AOT_obtain a where ‹a = ιx φ{x}›
    using "instantiation"[rotated] by blast
  moreover AOT_have ‹a = ιx φ{x} ≡ ∀z(𝒜φ{z} ≡ z = a)›
    using descriptions[axiom_inst] by blast
  ultimately AOT_have ‹∀z(𝒜φ{z} ≡ z = a)›
    using "≡E" by blast
  AOT_hence ‹∃x∀z(𝒜φ{z} ≡ z = x)› by (rule "∃I")
  AOT_thus ‹∃!x 𝒜φ{x}›
    using "uniqueness:2"[THEN "≡E"(2)] by fast
next
  AOT_assume ‹∃!x 𝒜φ{x}›
  AOT_hence ‹∃x∀z(𝒜φ{z} ≡ z = x)›
    using "uniqueness:2"[THEN "≡E"(1)] by fast
  then AOT_obtain a where ‹∀z(𝒜φ{z} ≡ z = a)›
    using "instantiation"[rotated] by blast
  moreover AOT_have ‹a = ιx φ{x} ≡ ∀z(𝒜φ{z} ≡ z = a)›
    using descriptions[axiom_inst] by blast
  ultimately AOT_have ‹a = ιx φ{x}›
    using "≡E" by blast
  AOT_thus ‹ιx φ{x}↓›
    by (metis "t=t-proper:2" "vdash-properties:6")
qed
```

```
AOT_theorem "actual-desc:2": ‹x = ιx φ{x} → 𝒜φ{x}›                                (151.2)
  using "&E"(1) "contraposition:1[2]" "≡E"(1) "nec-hintikka-scheme"
        "reductio-aa:2" "vdash-properties:9" by blast

(* Note: generalized alphabetic variant of the last theorem *)
AOT_theorem "actual-desc:3": ‹z = ιx φ{x} → 𝒜φ{z}›                                (151.3)
  using "actual-desc:2".

AOT_theorem "actual-desc:4": ‹ιx φ{x}↓ → 𝒜φ{ιx φ{x}}›                             (151.4)
proof(rule "→I")
  AOT_assume ‹ιx φ{x}↓›
  AOT_hence ‹∃y (y = ιx φ{x})› by (metis "rule=I:1" "existential:1")
  then AOT_obtain a where ‹a = ιx φ{x}› using "instantiation"[rotated] by blast
  AOT_thus ‹𝒜φ{ιx φ{x}}›
    using "actual-desc:2" "rule=E" "→E" by fast
qed

AOT_theorem "actual-desc:5": ‹ιx φ{x} = ιx ψ{x} → 𝒜∀x(φ{x} ≡ ψ{x})›               (151.5)
proof(rule "→I")
  AOT_assume 0: ‹ιx φ{x} = ιx ψ{x}›
  AOT_hence φ_down: ‹ιx φ{x}↓› and ψ_down: ‹ιx ψ{x}↓›
    using "t=t-proper:1" "t=t-proper:2" "vdash-properties:6" by blast+
  AOT_hence ‹∃y (y = ιx φ{x})› and ‹∃y (y = ιx ψ{x})›
    by (metis "rule=I:1" "existential:1")+
  then AOT_obtain a and b where a_eq: ‹a = ιx φ{x}› and b_eq: ‹b = ιx ψ{x}›
    using "instantiation"[rotated] by metis

  AOT_have ‹∀α∀β (α = β → β = α)›
    by (rule "∀I"; rule "∀I"; rule "id-eq:2")
  AOT_hence ‹∀β (ιx φ{x} = β → β = ιx φ{x})›
    using "∀E" φ_down by blast
  AOT_hence ‹ιx φ{x} = ιx ψ{x} → ιx ψ{x} = ιx φ{x}›
    using "∀E" ψ_down by blast
  AOT_hence 1: ‹ιx ψ{x} = ιx φ{x}› using 0
    "→E" by blast

  AOT_have ‹𝒜φ{x} ≡ 𝒜ψ{x}› for x
  proof(rule "≡I"; rule "→I")
    AOT_assume ‹𝒜φ{x}›
    moreover AOT_have ‹𝒜φ{x} → x = a› for x
      using "nec-hintikka-scheme"[THEN "≡E"(1), OF a_eq, THEN "&E"(2)]
            "∀E" by blast
    ultimately AOT_have ‹x = a›
      using "→E" by blast
    AOT_hence ‹x = ιx φ{x}›
      using a_eq "rule=E" by blast
    AOT_hence ‹x = ιx ψ{x}›
      using 0 "rule=E" by blast
    AOT_thus ‹𝒜ψ{x}›
      by (metis "actual-desc:3" "vdash-properties:6")
  next
    AOT_assume ‹𝒜ψ{x}›
    moreover AOT_have ‹𝒜ψ{x} → x = b› for x
      using "nec-hintikka-scheme"[THEN "≡E"(1), OF b_eq, THEN "&E"(2)]
            "∀E" by blast
    ultimately AOT_have ‹x = b›
      using "→E" by blast
    AOT_hence ‹x = ιx ψ{x}›
      using b_eq "rule=E" by blast
    AOT_hence ‹x = ιx φ{x}›
      using 1 "rule=E" by blast
    AOT_thus ‹𝒜φ{x}›
      by (metis "actual-desc:3" "vdash-properties:6")
```

```
  qed
  AOT_hence ‹𝒜(φ{x} ≡ ψ{x})› for x
    by (metis "Act-Basic:5" "≡E"(2))
  AOT_hence ‹∀x 𝒜(φ{x} ≡ ψ{x})›
    by (rule "∀I")
  AOT_thus ‹𝒜∀x (φ{x} ≡ ψ{x})›
    using "logic-actual-nec:3"[axiom_inst, THEN "≡E"(2)] by fast
qed

AOT_theorem "!box-desc:1": ‹∃!x □φ{x} → ∀y (y = ιx φ{x} → φ{y})›     (152.1)
proof(rule "→I")
  AOT_assume ‹∃!x □φ{x}›
  AOT_hence ζ: ‹∃x (□φ{x} & ∀z (□φ{z} → z = x))›
    using "uniqueness:1"[THEN "≡dfE"] by blast
  then AOT_obtain b where ϑ: ‹□φ{b} & ∀z (□φ{z} → z = b)›
    using "instantiation"[rotated] by blast
  AOT_show ‹∀y (y = ιx φ{x} → φ{y})›
  proof(rule GEN; rule "→I")
    fix y
    AOT_assume ‹y = ιx φ{x}›
    AOT_hence ‹𝒜φ{y} & ∀z (𝒜φ{z} → z = y)›
      using "nec-hintikka-scheme"[THEN "≡E"(1)] by blast
    AOT_hence ‹𝒜φ{b} → b = y›
      using "&E" "∀E" by blast
    moreover AOT_have ‹𝒜φ{b}›
      using ϑ[THEN "&E"(1)]  by (metis "nec-imp-act" "→E")
    ultimately AOT_have ‹b = y›
      using "→E" by blast
    moreover AOT_have ‹φ{b}›
      using ϑ[THEN "&E"(1)]  by (metis "qml:2"[axiom_inst] "→E")
    ultimately AOT_show ‹φ{y}›
      using "rule=E" by blast
  qed
qed

AOT_theorem "!box-desc:2":                                              (152.2)
  ‹∀x (φ{x} → □φ{x}) → (∃!x φ{x} → ∀y (y = ιx φ{x} → φ{y}))›
proof(rule "→I"; rule "→I")
  AOT_assume ‹∀x (φ{x} → □φ{x})›
  moreover AOT_assume ‹∃!x φ{x}›
  ultimately AOT_have ‹∃!x □φ{x}›
    using "nec-exist-!"[THEN "→E", THEN "→E"] by blast
  AOT_thus ‹∀y (y = ιx φ{x} → φ{y})›
    using "!box-desc:1" "→E" by blast
qed

(* Note: vacuous in the embedding. *)
AOT_theorem "dr-alphabetic-thm": ‹ιν φ{ν}↓ → ιν φ{ν} = ιμ φ{μ}›        (153)
  by (simp add: "rule=I:1" "→I")

subsection‹The Theory of Necessity›
text‹\label{PLM: 9.9}›

AOT_theorem "RM:1[prem]":                                               (156.1)
  assumes ‹Γ ⊢□ φ → ψ›
  shows ‹□Γ ⊢□ □φ → □ψ›
proof -
  AOT_have ‹□Γ ⊢□ □(φ → ψ)›
    using "RN[prem]" assms by blast
  AOT_thus ‹□Γ ⊢□ □φ → □ψ›
    by (metis "qml:1"[axiom_inst] "→E")
qed
```

```
AOT_theorem "RM:1":                                                    (156.1)
  assumes ‹⊢□ φ → ψ›
  shows ‹⊢□ □φ → □ψ›
  using "RM:1[prem]" assms by blast

lemmas RM = "RM:1"                                                     (156)

AOT_theorem "RM:2[prem]":                                              (156.2)
  assumes ‹Γ ⊢□ φ → ψ›
  shows ‹□Γ ⊢□ ◊φ → ◊ψ›
proof -
  AOT_have ‹Γ ⊢□ ¬ψ → ¬φ›
    using assms
    by (simp add: "contraposition:1[1]")
  AOT_hence ‹□Γ ⊢□ □¬ψ → □¬φ›
    using "RM:1[prem]" by blast
  AOT_thus ‹□Γ ⊢□ ◊φ → ◊ψ›
    by (meson "≡dfE" "≡dfI" "conventions:5" "→I" "modus-tollens:1")
qed

AOT_theorem "RM:2":                                                    (156.2)
  assumes ‹⊢□ φ → ψ›
  shows ‹⊢□ ◊φ → ◊ψ›
  using "RM:2[prem]" assms by blast

lemmas "RM◊" = "RM:2"

AOT_theorem "RM:3[prem]":                                              (156.3)
  assumes ‹Γ ⊢□ φ ≡ ψ›
  shows ‹□Γ ⊢□ □φ ≡ □ψ›
proof -
  AOT_have ‹Γ ⊢□ φ → ψ› and ‹Γ ⊢□ ψ → φ›
    using assms "≡E" "→I" by metis+
  AOT_hence ‹□Γ ⊢□ □φ → □ψ› and ‹□Γ ⊢□ □ψ → □φ›
    using "RM:1[prem]" by metis+
  AOT_thus ‹□Γ ⊢□ □φ ≡ □ψ›
    by (simp add: "≡I")
qed

AOT_theorem "RM:3":                                                    (156.3)
  assumes ‹⊢□ φ ≡ ψ›
  shows ‹⊢□ □φ ≡ □ψ›
  using "RM:3[prem]" assms by blast

lemmas RE = "RM:3"

AOT_theorem "RM:4[prem]":                                              (156.4)
  assumes ‹Γ ⊢□ φ ≡ ψ›
  shows ‹□Γ ⊢□ ◊φ ≡ ◊ψ›
proof -
  AOT_have ‹Γ ⊢□ φ → ψ› and ‹Γ ⊢□ ψ → φ›
    using assms "≡E" "→I" by metis+
  AOT_hence ‹□Γ ⊢□ ◊φ → ◊ψ› and ‹□Γ ⊢□ ◊ψ → ◊φ›
    using "RM:2[prem]" by metis+
  AOT_thus ‹□Γ ⊢□ ◊φ ≡ ◊ψ›
    by (simp add: "≡I")
qed

AOT_theorem "RM:4":                                                    (156.4)
  assumes ‹⊢□ φ ≡ ψ›
  shows ‹⊢□ ◊φ ≡ ◊ψ›
  using "RM:4[prem]" assms by blast
```

```
lemmas "RE◇" = "RM:4"

AOT_theorem "KBasic:1": ‹□φ → □(ψ → φ)›                                     (157.1)
  by (simp add: RM "pl:1"[axiom_inst])

AOT_theorem "KBasic:2": ‹□¬φ → □(φ → ψ)›                                    (157.2)
  by (simp add: RM "useful-tautologies:3")

AOT_theorem "KBasic:3": ‹□(φ & ψ) ≡ (□φ & □ψ)›                              (157.3)
proof (rule "≡I"; rule "→I")
  AOT_assume ‹□(φ & ψ)›
  AOT_thus ‹□φ & □ψ›
    by (meson RM "&I" "Conjunction Simplification"(1, 2) "→E")
next
  AOT_have ‹□φ → □(ψ → (φ & ψ))›
    by (simp add: "RM:1" Adjunction)
  AOT_hence ‹□φ → (□ψ → □(φ & ψ))›
    by (metis "Hypothetical Syllogism" "qml:1"[axiom_inst])
  moreover AOT_assume ‹□φ & □ψ›
  ultimately AOT_show ‹□(φ & ψ)›
    using "→E" "&E" by blast
qed

AOT_theorem "KBasic:4": ‹□(φ ≡ ψ) ≡ (□(φ → ψ) & □(ψ → φ))›                (157.4)
proof -
  AOT_have ϑ: ‹□((φ → ψ) & (ψ → φ)) ≡ (□(φ → ψ) & □(ψ → φ))›
    by (fact "KBasic:3")
  AOT_modally_strict {
    AOT_have ‹(φ ≡ ψ) ≡ ((φ → ψ) & (ψ → φ))›
      by (fact "conventions:3"[THEN "≡Df"])
  }
  AOT_hence ξ: ‹□(φ ≡ ψ) ≡ □((φ → ψ) & (ψ → φ))›
    by (rule RE)
  with ξ and ϑ AOT_show ‹□(φ ≡ ψ) ≡ (□(φ → ψ) & □(ψ → φ))›
    using "≡E"(5) by blast
qed

AOT_theorem "KBasic:5": ‹(□(φ → ψ) & □(ψ → φ)) → (□φ ≡ □ψ)›              (157.5)
proof -
  AOT_have ‹□(φ → ψ) → (□φ → □ψ)›
    by (fact "qml:1"[axiom_inst])
  moreover AOT_have ‹□(ψ → φ) → (□ψ → □φ)›
    by (fact "qml:1"[axiom_inst])
  ultimately AOT_have ‹(□(φ → ψ) & □(ψ → φ)) → ((□φ → □ψ) & (□ψ → □φ))›
    by (metis "&I" MP "Double Composition")
  moreover AOT_have ‹((□φ → □ψ) & (□ψ → □φ)) → (□φ ≡ □ψ)›
    using "conventions:3"[THEN "≡dfI"] "→I" by blast
  ultimately AOT_show ‹(□(φ → ψ) & □(ψ → φ)) → (□φ ≡ □ψ)›
    by (metis "Hypothetical Syllogism")
qed

AOT_theorem "KBasic:6": ‹□(φ ≡ ψ) → (□φ ≡ □ψ)›                              (157.6)
  using "KBasic:4" "KBasic:5" "deduction-theorem" "≡E"(1) "→E" by blast
AOT_theorem "KBasic:7": ‹((□φ & □ψ) ∨ (□¬φ & □¬ψ)) → □(φ ≡ ψ)›            (157.7)
proof (rule "→I"; drule "∨E"(1); (rule "→I")?)
  AOT_assume ‹□φ & □ψ›
  AOT_hence ‹□φ› and ‹□ψ› using "&E" by blast+
  AOT_hence ‹□(φ → ψ)› and ‹□(ψ → φ)› using "KBasic:1" "→E" by blast+
  AOT_hence ‹□(φ → ψ) & □(ψ → φ)› using "&I" by blast
  AOT_thus ‹□(φ ≡ ψ)›  by (metis "KBasic:4" "≡E"(2))
next
  AOT_assume ‹□¬φ & □¬ψ›
  AOT_hence 0: ‹□(¬φ & ¬ψ)› using "KBasic:3"[THEN "≡E"(2)] by blast
```

```
  AOT_modally_strict {
    AOT_have ‹(¬φ & ¬ψ) → (φ ≡ ψ)›
      by (metis "&E"(1) "&E"(2) "deduction-theorem" "≡I" "reductio-aa:1")
  }
  AOT_hence ‹□(¬φ & ¬ψ) → □(φ ≡ ψ)›
    by (rule RM)
  AOT_thus ‹□(φ ≡ ψ)› using 0 "→E" by blast
qed(auto)

AOT_theorem "KBasic:8": ‹□(φ & ψ) → □(φ ≡ ψ)›                         (157.8)
  by (meson "RM:1" "&E"(1) "&E"(2) "deduction-theorem" "≡I")
AOT_theorem "KBasic:9": ‹□(¬φ & ¬ψ) → □(φ ≡ ψ)›                       (157.9)
  by (metis "RM:1" "&E"(1) "&E"(2) "deduction-theorem" "≡I" "raa-cor:4")
AOT_theorem "KBasic:10": ‹□φ ≡ □¬¬φ›                                  (157.10)
  by (simp add: "RM:3" "oth-class-taut:3:b")
AOT_theorem "KBasic:11": ‹¬□φ ≡ ◇¬φ›                                  (157.11)
proof (rule "≡I"; rule "→I")
  AOT_show ‹◇¬φ› if ‹¬□φ›
    using that "≡dfI" "conventions:5" "KBasic:10" "≡E"(3) by blast
next
  AOT_show ‹¬□φ› if ‹◇¬φ›
    using "≡dfE" "conventions:5" "KBasic:10" "≡E"(4) that by blast
qed
AOT_theorem "KBasic:12": ‹□φ ≡ ¬◇¬φ›                                  (157.12)
proof (rule "≡I"; rule "→I")
  AOT_show ‹¬◇¬φ› if ‹□φ›
    using "¬¬I" "KBasic:11" "≡E"(3) that by blast
next
  AOT_show ‹□φ› if ‹¬◇¬φ›
  using "KBasic:11" "≡E"(1) "reductio-aa:1" that by blast
qed
AOT_theorem "KBasic:13": ‹□(φ → ψ) → (◇φ → ◇ψ)›                       (157.13)
proof -
  AOT_have ‹φ → ψ ⊢□ φ → ψ› by blast
  AOT_hence ‹□(φ → ψ) ⊢□ ◇φ → ◇ψ›
    using "RM:2[prem]" by blast
  AOT_thus ‹□(φ → ψ) → (◇φ → ◇ψ)› using "→I" by blast
qed
lemmas "K◇" = "KBasic:13"
AOT_theorem "KBasic:14": ‹◇□φ ≡ ¬□◇¬φ›                                (157.14)
  by (meson "RE◇" "KBasic:11" "KBasic:12" "≡E"(6) "oth-class-taut:3:a")
AOT_theorem "KBasic:15": ‹(□φ ∨ □ψ) → □(φ ∨ ψ)›                       (157.15)
proof -
  AOT_modally_strict {
    AOT_have ‹φ → (φ ∨ ψ)› and ‹ψ → (φ ∨ ψ)›
      by (auto simp: "Disjunction Addition"(1) "Disjunction Addition"(2))
  }
  AOT_hence ‹□φ → □(φ ∨ ψ)› and ‹□ψ → □(φ ∨ ψ)›
    using RM by blast+
  AOT_thus ‹(□φ ∨ □ψ) → □(φ ∨ ψ)›
    by (metis "∨E"(1) "deduction-theorem")
qed

AOT_theorem "KBasic:16": ‹(□φ & ◇ψ) → ◇(φ & ψ)›                       (157.16)
  by (meson "KBasic:13" "RM:1" Adjunction "Hypothetical Syllogism"
            Importation "→E")

AOT_theorem "rule-sub-lem:1:a":                                       (158.1.a)
  assumes ‹⊢□ □(ψ ≡ χ)›
  shows ‹⊢□ ¬ψ ≡ ¬χ›
  using "qml:2"[axiom_inst, THEN "→E", OF assms]
        "≡E"(1) "oth-class-taut:4:b" by blast
```

```
AOT_theorem "rule-sub-lem:1:b":                                        (158.1.b)
  assumes ‹⊢□ □(ψ ≡ χ)›
  shows ‹⊢□ (ψ → Θ) ≡ (χ → Θ)›
  using "qml:2"[axiom_inst, THEN "→E", OF assms]
  using "oth-class-taut:4:c" "vdash-properties:6" by blast

AOT_theorem "rule-sub-lem:1:c":                                        (158.1.c)
  assumes ‹⊢□ □(ψ ≡ χ)›
  shows ‹⊢□ (Θ → ψ) ≡ (Θ → χ)›
  using "qml:2"[axiom_inst, THEN "→E", OF assms]
  using "oth-class-taut:4:d" "vdash-properties:6" by blast

AOT_theorem "rule-sub-lem:1:d":                                        (158.1.d)
  assumes ‹for arbitrary α: ⊢□ □(ψ{α} ≡ χ{α})›
  shows ‹⊢□ ∀α ψ{α} ≡ ∀α χ{α}›
proof -
  AOT_modally_strict {
    AOT_have ‹∀α (ψ{α} ≡ χ{α})›
      using "qml:2"[axiom_inst, THEN "→E", OF assms] "∀I" by fast
    AOT_hence 0: ‹ψ{α} ≡ χ{α}› for α using "∀E" by blast
    AOT_show ‹∀α ψ{α} ≡ ∀α χ{α}›
    proof (rule "≡I"; rule "→I")
      AOT_assume ‹∀α ψ{α}›
      AOT_hence ‹ψ{α}› for α using "∀E" by blast
      AOT_hence ‹χ{α}› for α using 0 "≡E" by blast
      AOT_thus ‹∀α χ{α}› by (rule "∀I")
    next
      AOT_assume ‹∀α χ{α}›
      AOT_hence ‹χ{α}› for α using "∀E" by blast
      AOT_hence ‹ψ{α}› for α using 0 "≡E" by blast
      AOT_thus ‹∀α ψ{α}› by (rule "∀I")
    qed
  }
qed

AOT_theorem "rule-sub-lem:1:e":                                        (158.1.e)
  assumes ‹⊢□ □(ψ ≡ χ)›
  shows ‹⊢□ [λ ψ] ≡ [λ χ]›
  using "qml:2"[axiom_inst, THEN "→E", OF assms]
  using "≡E"(1) "propositions-lemma:6" by blast

AOT_theorem "rule-sub-lem:1:f":                                        (158.1.f)
  assumes ‹⊢□ □(ψ ≡ χ)›
  shows ‹⊢□ 𝓐ψ ≡ 𝓐χ›
  using "qml:2"[axiom_inst, THEN "→E", OF assms, THEN "RA[2]"]
  by (metis "Act-Basic:5" "≡E"(1))

AOT_theorem "rule-sub-lem:1:g":                                        (158.1.g)
  assumes ‹⊢□ □(ψ ≡ χ)›
  shows ‹⊢□ □ψ ≡ □χ›
  using "KBasic:6" assms "vdash-properties:6" by blast

text‹Note that instead of deriving @{text "rule-sub-lem:2"},
     @{text "rule-sub-lem:3"}, @{text "rule-sub-lem:4"},
     and @{text "rule-sub-nec"}, we construct substitution methods instead.›

class AOT_subst =
  fixes AOT_subst :: "('a ⇒ o) ⇒ bool"
    and AOT_subst_cond :: "'a ⇒ 'a ⇒ bool"
  assumes AOT_subst:
    "AOT_subst φ ⟹ AOT_subst_cond ψ χ ⟹ [v ⊨ «φ ψ» ≡ «φ χ»]"

named_theorems AOT_substI
```

```
instantiation o :: AOT_subst
begin

inductive AOT_subst_o where
  AOT_subst_o_id[AOT_substI]:
    ‹AOT_subst_o (λφ. φ)›
  | AOT_subst_o_const[AOT_substI]:
    ‹AOT_subst_o (λφ. ψ)›
  | AOT_subst_o_not[AOT_substI]:
    ‹AOT_subst_o Θ ⟹ AOT_subst_o (λ φ. «¬Θ{φ}»)›
  | AOT_subst_o_imp[AOT_substI]:
    ‹AOT_subst_o Θ ⟹ AOT_subst_o Ξ ⟹ AOT_subst_o (λ φ. «Θ{φ} → Ξ{φ}»)›
  | AOT_subst_o_lambda0[AOT_substI]:
    ‹AOT_subst_o Θ ⟹ AOT_subst_o (λ φ. (AOT_lambda0 (Θ φ)))›
  | AOT_subst_o_act[AOT_substI]:
    ‹AOT_subst_o Θ ⟹ AOT_subst_o (λ φ. «𝒜Θ{φ}»)›
  | AOT_subst_o_box[AOT_substI]:
    ‹AOT_subst_o Θ ⟹ AOT_subst_o (λ φ. «□Θ{φ}»)›
  | AOT_subst_o_by_def[AOT_substI]:
    ‹(⋀ ψ . AOT_model_equiv_def (Θ ψ) (Ξ ψ)) ⟹
      AOT_subst_o Ξ ⟹ AOT_subst_o Θ›


definition AOT_subst_cond_o where
  ‹AOT_subst_cond_o ≡ λ ψ χ . ∀ v . [v ⊨ ψ ≡ χ]›

instance
proof
  fix ψ χ :: o and φ :: ‹o ⇒ o›
  assume cond: ‹AOT_subst_cond ψ χ›
  assume ‹AOT_subst φ›
  moreover AOT_have ‹⊢□ ψ ≡ χ›
    using cond unfolding AOT_subst_cond_o_def by blast
  ultimately AOT_show ‹⊢□ φ{ψ} ≡ φ{χ}›
  proof (induct arbitrary: ψ χ)
    case AOT_subst_o_id
    thus ?case
      using "≡E"(2) "oth-class-taut:4:b" "rule-sub-lem:1:a" by blast
  next
    case (AOT_subst_o_const ψ)
    thus ?case
      by (simp add: "oth-class-taut:3:a")
  next
    case (AOT_subst_o_not Θ)
    thus ?case
      by (simp add: RN "rule-sub-lem:1:a")
  next
    case (AOT_subst_o_imp Θ Ξ)
    thus ?case
      by (meson RN "≡E"(5) "rule-sub-lem:1:b" "rule-sub-lem:1:c")
  next
    case (AOT_subst_o_lambda0 Θ)
    thus ?case
      by (simp add: RN "rule-sub-lem:1:e")
  next
    case (AOT_subst_o_act Θ)
    thus ?case
      by (simp add: RN "rule-sub-lem:1:f")
  next
    case (AOT_subst_o_box Θ)
    thus ?case
      by (simp add: RN "rule-sub-lem:1:g")
```

```
  next
    case (AOT_subst_o_by_def Θ Ξ)
    AOT_modally_strict {
      AOT_have ‹Ξ{ψ} ≡ Ξ{χ}›
        using AOT_subst_o_by_def by simp
      AOT_thus ‹Θ{ψ} ≡ Θ{χ}›
        using "≡Df"[OF AOT_subst_o_by_def(1), of _ ψ]
              "≡Df"[OF AOT_subst_o_by_def(1), of _ χ]
        by (metis "≡E"(6) "oth-class-taut:3:a")
    }
  qed
qed
end

instantiation "fun" :: (AOT_Term_id_2, AOT_subst) AOT_subst
begin

definition AOT_subst_cond_fun :: ‹('a ⇒ 'b) ⇒ ('a ⇒ 'b) ⇒ bool› where
  ‹AOT_subst_cond_fun ≡ λ φ ψ . ∀ α . AOT_subst_cond (φ (AOT_term_of_var α))
                                                     (ψ (AOT_term_of_var α))›

inductive AOT_subst_fun :: ‹(('a ⇒ 'b) ⇒ o) ⇒ bool› where
  AOT_subst_fun_const[AOT_substI]:
    ‹AOT_subst_fun (λφ. ψ)›
  | AOT_subst_fun_id[AOT_substI]:
    ‹AOT_subst Π ⟹ AOT_subst_fun (λφ. Π (φ (AOT_term_of_var α)))›
  | AOT_subst_fun_all[AOT_substI]:
    ‹AOT_subst Π ⟹ (⋀ α . AOT_subst_fun (Θ (AOT_term_of_var α))) ⟹
     AOT_subst_fun (λφ :: 'a ⇒ 'b. Π «∀α «Θ (α::'a) φ»»)›
  | AOT_subst_fun_not[AOT_substI]:
    ‹AOT_subst Π ⟹ AOT_subst_fun (λφ. «¬«Π φ»»)›
  | AOT_subst_fun_imp[AOT_substI]:
    ‹AOT_subst Π ⟹ AOT_subst Θ ⟹ AOT_subst_fun (λφ. ««Π φ» → «Θ φ»»)›
  | AOT_subst_fun_lambda0[AOT_substI]:
    ‹AOT_subst Θ ⟹ AOT_subst_fun (λ φ. (AOT_lambda0 (Θ φ)))›
  | AOT_subst_fun_act[AOT_substI]:
    ‹AOT_subst Θ ⟹ AOT_subst_fun (λ φ. «𝓐«Θ φ»»)›
  | AOT_subst_fun_box[AOT_substI]:
    ‹AOT_subst Θ ⟹ AOT_subst_fun (λ φ. «□«Θ φ»»)›
  | AOT_subst_fun_def[AOT_substI]:
    ‹(⋀ φ . AOT_model_equiv_def (Θ φ) (Π φ)) ⟹
     AOT_subst_fun Π ⟹ AOT_subst_fun Θ›

instance proof
  fix ψ χ :: ‹'a ⇒ 'b› and φ :: ‹('a ⇒ 'b) ⇒ o›
  assume ‹AOT_subst φ›
  moreover assume cond: ‹AOT_subst_cond ψ χ›
  ultimately AOT_show ‹⊢□ «φ ψ» ≡ «φ χ»›
  proof(induct)
    case (AOT_subst_fun_const ψ)
    then show ?case by (simp add: "oth-class-taut:3:a")
  next
  case (AOT_subst_fun_id Π x)
  then show ?case by (simp add: AOT_subst AOT_subst_cond_fun_def)
  next
  next
  case (AOT_subst_fun_all Π Θ)
  AOT_have ‹⊢□ □(Θ{α, «ψ»} ≡ Θ{α, «χ»})› for α
    using AOT_subst_fun_all.hyps(3) AOT_subst_fun_all.prems RN by presburger
  thus ?case using AOT_subst[OF AOT_subst_fun_all(1)]
    by (simp add: RN "rule-sub-lem:1:d"
                  AOT_subst_cond_fun_def AOT_subst_cond_o_def)
  next
```

```
  case (AOT_subst_fun_not Π)
  then show ?case by (simp add: RN "rule-sub-lem:1:a")
  next
  case (AOT_subst_fun_imp Π Θ)
  then show ?case
    unfolding AOT_subst_cond_fun_def AOT_subst_cond_o_def
    by (meson "≡E"(5) "oth-class-taut:4:c" "oth-class-taut:4:d" "→E")
  next
  case (AOT_subst_fun_lambda0 Θ)
  then show ?case by (simp add: RN "rule-sub-lem:1:e")
  next
  case (AOT_subst_fun_act Θ)
  then show ?case by (simp add: RN "rule-sub-lem:1:f")
  next
  case (AOT_subst_fun_box Θ)
  then show ?case by (simp add: RN "rule-sub-lem:1:g")
  next
  case (AOT_subst_fun_def Θ Π)
  then show ?case
    by (meson "df-rules-formulas[3]" "df-rules-formulas[4]" "≡I" "≡E"(5))
  qed
qed
end

ML‹
fun prove_AOT_subst_tac ctxt = REPEAT (SUBGOAL (fn (trm,_) => let
  fun findHeadConst (Const x) = SOME x
    | findHeadConst (A $ _) = findHeadConst A
    | findHeadConst _ = NONE
  fun findDef (Const (const_name‹AOT_model_equiv_def›, _) $ lhs $ _)
      = findHeadConst lhs
    | findDef (A $ B) = (case findDef A of SOME x => SOME x | _ => findDef B)
    | findDef (Abs (_,_,c)) = findDef c
    | findDef _ = NONE
  val const_opt = (findDef trm)
  val defs = case const_opt of SOME const => List.filter (fn thm => let
      val concl = Thm.concl_of thm
      val thmconst = (findDef concl)
      in case thmconst of SOME (c,_) => fst const = c | _ => false end)
      (AOT_Definitions.get ctxt)
      | _ => []
  val tac = case defs of
            [] => safe_step_tac (ctxt addSIs @{thms AOT_substI}) 1
            | _ => resolve_tac ctxt defs 1
  in tac end) 1)
fun getSubstThm ctxt reversed phi p q = let
val p_ty = Term.type_of p
val abs = HOLogic.mk_Trueprop (@{const AOT_subst(_)} $ phi)
val abs = Syntax.check_term ctxt abs
val substThm = Goal.prove ctxt [] [] abs
  (fn {context=ctxt, prems=_} => prove_AOT_subst_tac ctxt)
val substThm = substThm RS @{thm AOT_subst}
in if reversed then let
  val substThm = Drule.instantiate_normalize
        (TVars.empty, Vars.make [((("χ", 0), p_ty), Thm.cterm_of ctxt p),
        ((("ψ", 0), p_ty), Thm.cterm_of ctxt q)]) substThm
  val substThm = substThm RS @{thm "≡E"(1)}
  in substThm end
else
  let
  val substThm = Drule.instantiate_normalize
        (TVars.empty, Vars.make [((("ψ", 0), p_ty), Thm.cterm_of ctxt p),
        ((("χ", 0), p_ty), Thm.cterm_of ctxt q)]) substThm
```

```
  val substThm = substThm RS @{thm "≡E"(2)}
  in substThm end end
›

method_setup AOT_subst = ‹
Scan.option (Scan.lift (Args.parens (Args.$$$ "reverse"))) --
Scan.lift (Parse.embedded_inner_syntax -- Parse.embedded_inner_syntax) --
Scan.option (Scan.lift (Args.$$$ "for" -- Args.colon) |--
Scan.repeat1 (Scan.lift (Parse.embedded_inner_syntax) --
Scan.option (Scan.lift (Args.$$$ "::" |-- Parse.embedded_inner_syntax))))
>> (fn ((reversed,(raw_p,raw_q)),raw_bounds) => (fn ctxt =>
(Method.SIMPLE_METHOD (Subgoal.FOCUS (fn {context = ctxt, params = _,
  prems = prems, asms = asms, concl = concl, schematics = _} =>
let
val thms = prems
val ctxt' = ctxt
val ctxt = Context_Position.set_visible false ctxt
val raw_bounds = case raw_bounds of SOME bounds => bounds | _ => []

val ctxt = (fold (fn (bound, ty) => fn ctxt =>
  let
    val bound = AOT_read_term @{nonterminal τ'} ctxt bound
    val ty = Option.map (Syntax.read_typ ctxt) ty
    val ctxt = case ty of SOME ty => let
        val bound = Const ("_type_constraint_", Type ("fun", [ty,ty])) $ bound
        val bound = Syntax.check_term ctxt bound
      in Variable.declare_term bound ctxt end | _ => ctxt
  in ctxt end)) raw_bounds ctxt

val p = AOT_read_term @{nonterminal φ'} ctxt raw_p
val p = Syntax.check_term ctxt p
val ctxt = Variable.declare_term p ctxt
val q = AOT_read_term @{nonterminal φ'} ctxt raw_q
val q = Syntax.check_term ctxt q
val ctxt = Variable.declare_term q ctxt

val bounds = (map (fn (bound, _) =>
  Syntax.check_term ctxt (AOT_read_term @{nonterminal τ'} ctxt bound)
)) raw_bounds
val p = fold (fn bound => fn p =>
  Term.abs ("α", Term.type_of bound) (Term.abstract_over (bound,p)))
  bounds p
val p = Syntax.check_term ctxt p
val p_ty = Term.type_of p

val pat = @{const Trueprop} $
  (@{const AOT_model_valid_in} $ Var (("w",0), @{typ w}) $
   (Var (("φ",0), Type (type_name‹fun›, [p_ty, @{typ o}])) $ p))
val univ = Unify.matchers (Context.Proof ctxt) [(pat, Thm.term_of concl)]
val univ = hd (Seq.list_of univ) (* TODO: consider all matches *)
val phi = the (Envir.lookup univ
  (("φ",0), Type (type_name‹fun›, [p_ty, @{typ o}])))

val q = fold (fn bound => fn q =>
  Term.abs ("α", Term.type_of bound) (Term.abstract_over (bound,q))) bounds q
val q = Syntax.check_term ctxt q

(* Reparse to report bounds as fixes. *)
val ctxt = Context_Position.restore_visible ctxt' ctxt
val ctxt' = ctxt
fun unsource str = fst (Input.source_content (Syntax.read_input str))
val (_,ctxt') = Proof_Context.add_fixes (map (fn (str,_) =>
  (Binding.make (unsource str, Position.none), NONE, Mixfix.NoSyn)) raw_bounds)
```

```
  ctxt'
val _ = (map (fn (x,_) =>
  Syntax.check_term ctxt (AOT_read_term @{nonterminal τ'} ctxt' x)))
  raw_bounds
val _ = AOT_read_term @{nonterminal φ'} ctxt' raw_p
val _ = AOT_read_term @{nonterminal φ'} ctxt' raw_q
val reversed = case reversed of SOME _ => true | _ => false
val simpThms = [@{thm AOT_subst_cond_o_def}, @{thm AOT_subst_cond_fun_def}]
in
resolve_tac ctxt [getSubstThm ctxt reversed phi p q] 1
THEN simp_tac (ctxt addsimps simpThms) 1
THEN (REPEAT (resolve_tac ctxt [@{thm allI}] 1))
THEN (TRY (resolve_tac ctxt thms 1))
end
) ctxt 1))))
›

method_setup AOT_subst_def = ‹
Scan.option (Scan.lift (Args.parens (Args.$$$ "reverse"))) -
Attrib.thm
» (fn (reversed,fact) => (fn ctxt =>
(Method.SIMPLE_METHOD (Subgoal.FOCUS (fn {context = ctxt, params = _,
  prems = prems, asms = asms, concl = concl, schematics = _} =>
let
val c = Thm.concl_of fact
val (lhs, rhs) = case c of (const‹Trueprop› $
    (const‹AOT_model_equiv_def› $ lhs $ rhs)) => (lhs, rhs)
  | _ => raise Fail "Definition expected."
val substCond = HOLogic.mk_Trueprop
  (Const (const_name‹AOT_subst_cond›, dummyT) $ lhs $ rhs)
val substCond = Syntax.check_term
  (Proof_Context.set_mode Proof_Context.mode_schematic ctxt)
  substCond
val simpThms = [@{thm AOT_subst_cond_o_def},
  @{thm AOT_subst_cond_fun_def},
  fact RS @{thm "≡Df"}]
val substCondThm = Goal.prove ctxt [] [] substCond
  (fn {context=ctxt, prems=prems} =>
      (SUBGOAL (fn (trm,int) =>
        auto_tac (ctxt addsimps simpThms)) 1))
val substThm = substCondThm RSN (2,@{thm AOT_subst})
in
resolve_tac ctxt [substThm RS
  (case reversed of NONE => @{thm "≡E"(2)} | _ => @{thm "≡E"(1)})] 1
THEN prove_AOT_subst_tac ctxt
THEN (TRY (resolve_tac ctxt prems 1))
end
) ctxt 1))))
›

method_setup AOT_subst_thm = ‹
Scan.option (Scan.lift (Args.parens (Args.$$$ "reverse"))) -
Attrib.thm
» (fn (reversed,fact) => (fn ctxt =>
(Method.SIMPLE_METHOD (Subgoal.FOCUS (fn {context = ctxt, params = _,
  prems = prems, asms = asms, concl = concl, schematics = _} =>
let
val c = Thm.concl_of fact
val (lhs, rhs) = case c of
  (const‹Trueprop› $
   (const‹AOT_model_valid_in› $ _ $
    (const‹AOT_equiv› $ lhs $ rhs))) => (lhs, rhs)
  | _ => raise Fail "Equivalence expected."
```

```
val substCond = HOLogic.mk_Trueprop
  (Const (const_name‹AOT_subst_cond›, dummyT) $ lhs $ rhs)
val substCond = Syntax.check_term
  (Proof_Context.set_mode Proof_Context.mode_schematic ctxt)
  substCond
val simpThms = [@{thm AOT_subst_cond_o_def},
  @{thm AOT_subst_cond_fun_def},
  fact]
val substCondThm = Goal.prove ctxt [] [] substCond
  (fn {context=ctxt, prems=prems} =>
      (SUBGOAL (fn (trm,int) => auto_tac (ctxt addsimps simpThms)) 1))
val substThm = substCondThm RSN (2,@{thm AOT_subst})
in
resolve_tac ctxt [substThm RS
  (case reversed of NONE => @{thm "≡E"(2)} | _ => @{thm "≡E"(1)})] 1
THEN prove_AOT_subst_tac ctxt
THEN (TRY (resolve_tac ctxt prems 1))
end
) ctxt 1))))
›

AOT_theorem "rule-sub-remark:1[1]":                                   (160.1)
  assumes ‹⊢□ A!x ≡ ¬◊E!x› and ‹¬A!x›
  shows ‹¬¬◊E!x›
  by (AOT_subst (reverse) ‹¬◊E!x› ‹A!x›)
     (auto simp: assms)

AOT_theorem "rule-sub-remark:1[2]":                                   (160.1)
  assumes ‹⊢□ A!x ≡ ¬◊E!x› and  ‹¬¬◊E!x›
  shows ‹¬A!x›
  by (AOT_subst ‹A!x› ‹¬◊E!x›)
     (auto simp: assms)

AOT_theorem "rule-sub-remark:2[1]":                                   (160.2)
  assumes ‹⊢□ [R]xy ≡ ([R]xy & ([Q]a ∨ ¬[Q]a))›
      and ‹p → [R]xy›
  shows ‹p → [R]xy & ([Q]a ∨ ¬[Q]a)›
  by (AOT_subst_thm (reverse) assms(1)) (simp add: assms(2))

AOT_theorem "rule-sub-remark:2[2]":                                   (160.2)
  assumes ‹⊢□ [R]xy ≡ ([R]xy & ([Q]a ∨ ¬[Q]a))›
      and ‹p → [R]xy & ([Q]a ∨ ¬[Q]a)›
  shows ‹p → [R]xy›
  by (AOT_subst_thm assms(1)) (simp add: assms(2))

AOT_theorem "rule-sub-remark:3[1]":                                   (160.3)
  assumes ‹for arbitrary x: ⊢□ A!x ≡ ¬◊E!x›
      and ‹∃x A!x›
    shows ‹∃x ¬◊E!x›
  by (AOT_subst (reverse) ‹¬◊E!x› ‹A!x› for: x)
     (auto simp: assms)

AOT_theorem "rule-sub-remark:3[2]":                                   (160.3)
  assumes ‹for arbitrary x: ⊢□ A!x ≡ ¬◊E!x›
      and ‹∃x ¬◊E!x›
    shows ‹∃x A!x›
  by (AOT_subst ‹A!x› ‹¬◊E!x› for: x)
     (auto simp: assms)

AOT_theorem "rule-sub-remark:4[1]":                                   (160.4)
  assumes ‹⊢□ ¬¬[P]x ≡ [P]x› and ‹𝒜¬¬[P]x›
  shows ‹𝒜[P]x›
```

```
  by (AOT_subst_thm (reverse) assms(1)) (simp add: assms(2))

AOT_theorem "rule-sub-remark:4[2]":                                    (160.4)
  assumes ‹⊢□ ¬¬[P]x ≡ [P]x› and ‹𝒜[P]x›
  shows ‹𝒜¬¬[P]x›
  by (AOT_subst_thm assms(1)) (simp add: assms(2))

AOT_theorem "rule-sub-remark:5[1]":                                    (160.5)
  assumes ‹⊢□ (φ → ψ) ≡ (¬ψ → ¬φ)› and ‹□(φ → ψ)›
  shows ‹□(¬ψ → ¬φ)›
  by (AOT_subst_thm (reverse) assms(1)) (simp add: assms(2))

AOT_theorem "rule-sub-remark:5[2]":                                    (160.5)
  assumes ‹⊢□ (φ → ψ) ≡ (¬ψ → ¬φ)› and ‹□(¬ψ → ¬φ)›
  shows ‹□(φ → ψ)›
  by (AOT_subst_thm assms(1)) (simp add: assms(2))

AOT_theorem "rule-sub-remark:6[1]":                                    (160.6)
  assumes ‹⊢□ ψ ≡ χ› and ‹□(φ → ψ)›
  shows ‹□(φ → χ)›
  by (AOT_subst_thm (reverse) assms(1)) (simp add: assms(2))

AOT_theorem "rule-sub-remark:6[2]":                                    (160.6)
  assumes ‹⊢□ ψ ≡ χ› and ‹□(φ → χ)›
  shows ‹□(φ → ψ)›
  by (AOT_subst_thm assms(1)) (simp add: assms(2))

AOT_theorem "rule-sub-remark:7[1]":                                    (160.7)
  assumes ‹⊢□ φ ≡ ¬¬φ› and ‹□(φ → φ)›
  shows ‹□(¬¬φ → φ)›
  by (AOT_subst_thm (reverse) assms(1)) (simp add: assms(2))

AOT_theorem "rule-sub-remark:7[2]":                                    (160.7)
  assumes ‹⊢□ φ ≡ ¬¬φ› and ‹□(¬¬φ → φ)›
  shows ‹□(φ → φ)›
  by (AOT_subst_thm assms(1)) (simp add: assms(2))

AOT_theorem "KBasic2:1": ‹□¬φ ≡ ¬◇φ›                                  (161.1)
  by (meson "conventions:5" "contraposition:2"
            "Hypothetical Syllogism" "df-rules-formulas[3]"
            "df-rules-formulas[4]" "≡I" "useful-tautologies:1")

AOT_theorem "KBasic2:2": ‹◇(φ ∨ ψ) ≡ (◇φ ∨ ◇ψ)›                        (161.2)
proof -
  AOT_have ‹◇(φ ∨ ψ) ≡ ◇¬(¬φ & ¬ψ)›
    by (simp add: "RE◇" "oth-class-taut:5:b")
  also AOT_have ‹... ≡ ¬□(¬φ & ¬ψ)›
    using "KBasic:11" "≡E"(6) "oth-class-taut:3:a" by blast
  also AOT_have ‹... ≡ ¬(□¬φ & □¬ψ)›
    using "KBasic:3" "≡E"(1) "oth-class-taut:4:b" by blast
  also AOT_have ‹... ≡ ¬(¬◇φ & ¬◇ψ)›
    using "KBasic2:1"
    by (AOT_subst ‹□¬φ› ‹¬◇φ›; AOT_subst ‹□¬ψ› ‹¬◇ψ›;
        auto simp: "oth-class-taut:3:a")
  also AOT_have ‹... ≡ ¬¬(◇φ ∨ ◇ψ)›
    using "≡E"(6) "oth-class-taut:3:b" "oth-class-taut:5:b" by blast
  also AOT_have ‹... ≡ ◇φ ∨ ◇ψ›
    by (simp add: "≡I" "useful-tautologies:1" "useful-tautologies:2")
  finally show ?thesis .
qed

AOT_theorem "KBasic2:3": ‹◇(φ & ψ) → (◇φ & ◇ψ)›                        (161.3)
  by (metis "RM◇" "&I" "Conjunction Simplification"(1,2)
```

```
            "→I" "modus-tollens:1" "reductio-aa:1")

AOT_theorem "KBasic2:4": ‹◇(φ → ψ) ≡ (□φ → ◇ψ)›                    (161.4)
proof -
  AOT_have ‹◇(φ → ψ) ≡ ◇(¬φ ∨ ψ)›
    by (AOT_subst ‹φ → ψ› ‹¬φ ∨ ψ›)
       (auto simp: "oth-class-taut:1:c" "oth-class-taut:3:a")
  also AOT_have ‹... ≡ ◇¬φ ∨ ◇ψ›
    by (simp add: "KBasic2:2")
  also AOT_have ‹... ≡ ¬□φ ∨ ◇ψ›
    by (AOT_subst ‹¬□φ› ‹◇¬φ›)
       (auto simp: "KBasic:11" "oth-class-taut:3:a")
  also AOT_have ‹... ≡ □φ → ◇ψ›
    using "≡E"(6) "oth-class-taut:1:c" "oth-class-taut:3:a" by blast
  finally show ?thesis .
qed

AOT_theorem "KBasic2:5": ‹◇◇φ ≡ ¬□□¬φ›                              (161.5)
  using "conventions:5"[THEN "≡Df"]
  by (AOT_subst ‹◇φ› ‹¬□¬φ›;
      AOT_subst ‹◇¬□¬φ› ‹¬□¬¬□¬φ›;
      AOT_subst (reverse) ‹¬¬□¬φ› ‹□¬φ›)
     (auto simp: "oth-class-taut:3:b" "oth-class-taut:3:a")


AOT_theorem "KBasic2:6": ‹□(φ ∨ ψ) → (□φ ∨ ◇ψ)›                    (161.6)
proof(rule "→I"; rule "raa-cor:1")
  AOT_assume ‹□(φ ∨ ψ)›
  AOT_hence ‹□(¬φ → ψ)›
    using "conventions:2"[THEN "≡Df"]
    by (AOT_subst (reverse) ‹¬φ → ψ› ‹φ ∨ ψ›) simp
  AOT_hence 1: ‹◇¬φ → ◇ψ›
    using "KBasic:13" "vdash-properties:10" by blast
  AOT_assume ‹¬(□φ ∨ ◇ψ)›
  AOT_hence ‹¬□φ› and ‹¬◇ψ›
    using "&E" "≡E"(1) "oth-class-taut:5:d" by blast+
  AOT_thus ‹◇ψ & ¬◇ψ›
    using "&I"(1) 1[THEN "→E"] "KBasic:11" "≡E"(4) "raa-cor:3" by blast
qed

AOT_theorem "KBasic2:7": ‹(□(φ ∨ ψ) & ◇¬φ) → ◇ψ›                   (161.7)
proof(rule "→I"; frule "&E"(1); drule "&E"(2))
  AOT_assume ‹□(φ ∨ ψ)›
  AOT_hence 1: ‹□φ ∨ ◇ψ›
    using "KBasic2:6" "∨I"(2) "∨E"(1) by blast
  AOT_assume ‹◇¬φ›
  AOT_hence ‹¬□φ› using "KBasic:11" "≡E"(2) by blast
  AOT_thus ‹◇ψ› using 1 "∨E"(2) by blast
qed

AOT_theorem "T-S5-fund:1": ‹φ → ◇φ›                                (162.1)
  by (meson "≡dfI" "conventions:5" "contraposition:2"
            "Hypothetical Syllogism" "→I" "qml:2"[axiom_inst])
lemmas "T◇" = "T-S5-fund:1"

AOT_theorem "T-S5-fund:2": ‹◇□φ → □φ›                              (162.2)
proof(rule "→I")
  AOT_assume ‹◇□φ›
  AOT_hence ‹¬□◇¬φ›
    using "KBasic:14" "≡E"(4) "raa-cor:3" by blast
  moreover AOT_have ‹◇¬φ → □◇¬φ›
    by (fact "qml:3"[axiom_inst])
  ultimately AOT_have ‹¬◇¬φ›
```

```
    using "modus-tollens:1" by blast
  AOT_thus ‹□φ› using "KBasic:12" "≡E"(2) by blast
qed
lemmas "5◇" = "T-S5-fund:2"

AOT_theorem "Act-Sub:1": ‹𝒜φ ≡ ¬𝒜¬φ›                                   (163.1)
  by (AOT_subst ‹𝒜¬φ› ‹¬𝒜φ›)
     (auto simp: "logic-actual-nec:1"[axiom_inst] "oth-class-taut:3:b")

AOT_theorem "Act-Sub:2": ‹◇φ ≡ 𝒜◇φ›                                    (163.2)
  using "conventions:5"[THEN "≡Df"]
  by (AOT_subst ‹◇φ› ‹¬□¬φ›)
     (metis "deduction-theorem" "≡I" "≡E"(1) "≡E"(2) "≡E"(3)
            "logic-actual-nec:1"[axiom_inst] "qml-act:2"[axiom_inst])

AOT_theorem "Act-Sub:3": ‹𝒜φ → ◇φ›                                     (163.3)
  using "conventions:5"[THEN "≡Df"]
  by (AOT_subst ‹◇φ› ‹¬□¬φ›)
     (metis "Act-Sub:1" "→I" "≡E"(4) "nec-imp-act" "reductio-aa:2" "→E")

AOT_theorem "Act-Sub:4": ‹𝒜φ ≡ ◇𝒜φ›                                    (163.4)
proof (rule "≡I"; rule "→I")
  AOT_assume ‹𝒜φ›
  AOT_thus ‹◇𝒜φ› using "T◇" "vdash-properties:10" by blast
next
  AOT_assume ‹◇𝒜φ›
  AOT_hence ‹¬□¬𝒜φ›
    using "≡dfE" "conventions:5" by blast
  AOT_hence ‹¬□𝒜¬φ›
    by (AOT_subst ‹𝒜¬φ› ‹¬𝒜φ›)
       (simp add: "logic-actual-nec:1"[axiom_inst])
  AOT_thus ‹𝒜φ›
    using "Act-Basic:1" "Act-Basic:6" "∨E"(3) "≡E"(4)
          "reductio-aa:1" by blast
qed

AOT_theorem "Act-Sub:5": ‹◇𝒜φ → 𝒜◇φ›                                   (163.5)
  by (metis "Act-Sub:2" "Act-Sub:3" "Act-Sub:4" "→I" "≡E"(1) "≡E"(2) "→E")

AOT_theorem "S5Basic:1": ‹◇φ ≡ □◇φ›                                    (164.1)
  by (simp add: "≡I" "qml:2"[axiom_inst] "qml:3"[axiom_inst])

AOT_theorem "S5Basic:2": ‹□φ ≡ ◇□φ›                                    (164.2)
  by (simp add: "T◇" "5◇" "≡I")

AOT_theorem "S5Basic:3": ‹φ → □◇φ›                                     (164.3)
  using "T◇" "Hypothetical Syllogism" "qml:3"[axiom_inst] by blast
lemmas "B" = "S5Basic:3"

AOT_theorem "S5Basic:4": ‹◇□φ → φ›                                     (164.4)
  using "5◇" "Hypothetical Syllogism" "qml:2"[axiom_inst] by blast
lemmas "B◇" = "S5Basic:4"

AOT_theorem "S5Basic:5": ‹□φ → □□φ›                                    (164.5)
  using "RM:1" "B" "5◇" "Hypothetical Syllogism" by blast
lemmas "4" = "S5Basic:5"

AOT_theorem "S5Basic:6": ‹□φ ≡ □□φ›                                    (164.6)
  by (simp add: "4" "≡I" "qml:2"[axiom_inst])

AOT_theorem "S5Basic:7": ‹◇◇φ → ◇φ›                                    (164.7)
  using "conventions:5"[THEN "≡Df"] "oth-class-taut:3:b"
  by (AOT_subst ‹◇◇φ› ‹¬□¬◇φ›;
```

```
      AOT_subst ‹◇φ› ‹¬□¬φ›;
      AOT_subst (reverse) ‹¬¬□¬φ› ‹□¬φ›;
      AOT_subst (reverse) ‹□□¬φ› ‹□¬φ›)
     (auto simp: "S5Basic:6" "if-p-then-p")

lemmas "4◇" = "S5Basic:7"

AOT_theorem "S5Basic:8": ‹◇◇φ ≡ ◇φ›                                        (164.8)
  by (simp add: "4◇" "T◇" "≡I")

AOT_theorem "S5Basic:9": ‹□(φ ∨ □ψ) ≡ (□φ ∨ □ψ)›                           (164.9)
  apply (rule "≡I"; rule "→I")
  using "KBasic2:6" "5◇" "∨I"(3) "if-p-then-p" "vdash-properties:10"
   apply blast
  by (meson "KBasic:15" "4" "∨I"(3) "∨E"(1) "Disjunction Addition"(1)
            "con-dis-taut:7" "intro-elim:1" "Commutativity of ∨")

AOT_theorem "S5Basic:10": ‹□(φ ∨ ◇ψ) ≡ (□φ ∨ ◇ψ)›                         (164.10)
proof(rule "≡I"; rule "→I")
  AOT_assume ‹□(φ ∨ ◇ψ)›
  AOT_hence ‹□φ ∨ ◇◇ψ›
    by (meson "KBasic2:6" "∨I"(2) "∨E"(1))
  AOT_thus ‹□φ ∨ ◇ψ›
    by (meson "B◇" "4" "4◇" "T◇" "∨I"(3))
next
  AOT_assume ‹□φ ∨ ◇ψ›
  AOT_hence ‹□φ ∨ □◇ψ›
    by (meson "S5Basic:1" "B◇" "S5Basic:6" "T◇" "5◇" "∨I"(3) "intro-elim:1")
  AOT_thus ‹□(φ ∨ ◇ψ)›
    by (meson "KBasic:15" "∨I"(3) "∨E"(1) "Disjunction Addition"(1,2))
qed

AOT_theorem "S5Basic:11": ‹◇(φ & ◇ψ) ≡ (◇φ & ◇ψ)›                         (164.11)
proof -
  AOT_have ‹◇(φ & ◇ψ) ≡ ◇¬(¬φ ∨ ¬◇ψ)›
    by (AOT_subst ‹φ & ◇ψ› ‹¬(¬φ ∨ ¬◇ψ)›)
       (auto simp: "oth-class-taut:5:a" "oth-class-taut:3:a")
  also AOT_have ‹... ≡ ◇¬(¬φ ∨ □¬ψ)›
    by (AOT_subst ‹□¬ψ› ‹¬◇ψ›)
       (auto simp: "KBasic2:1" "oth-class-taut:3:a")
  also AOT_have ‹... ≡ ¬□(¬φ ∨ □¬ψ)›
    using "KBasic:11" "≡E"(6) "oth-class-taut:3:a" by blast
  also AOT_have ‹... ≡ ¬(□¬φ ∨ □¬ψ)›
    using "S5Basic:9" "≡E"(1) "oth-class-taut:4:b" by blast
  also AOT_have ‹... ≡ ¬(¬◇φ ∨ ¬◇ψ)›
    using "KBasic2:1"
    by (AOT_subst ‹□¬φ› ‹¬◇φ›; AOT_subst ‹□¬ψ› ‹¬◇ψ›)
       (auto simp:  "oth-class-taut:3:a")
  also AOT_have ‹... ≡ ◇φ & ◇ψ›
    using "≡E"(6) "oth-class-taut:3:a" "oth-class-taut:5:a" by blast
  finally show ?thesis .
qed

AOT_theorem "S5Basic:12": ‹◇(φ & □ψ) ≡ (◇φ & □ψ)›                         (164.12)
proof (rule "≡I"; rule "→I")
  AOT_assume ‹◇(φ & □ψ)›
  AOT_hence ‹◇φ & ◇□ψ›
    using "KBasic2:3" "vdash-properties:6" by blast
  AOT_thus ‹◇φ & □ψ›
    using "5◇" "&I" "&E"(1) "&E"(2) "vdash-properties:6" by blast
next
  AOT_assume ‹◇φ & □ψ›
  moreover AOT_have ‹(□□ψ & ◇φ) → ◇(φ & □ψ)›
```

```
    by (AOT_subst ‹φ & □ψ› ‹□ψ & φ›)
       (auto simp: "Commutativity of &" "KBasic:16")
  ultimately AOT_show ‹◇(φ & □ψ)›
    by (metis "4" "&I" "Conjunction Simplification"(1,2) "→E")
qed

AOT_theorem "S5Basic:13": ‹□(φ → □ψ) ≡ □(◇φ → ψ)›                  (164.13)
proof (rule "≡I")
  AOT_modally_strict {
    AOT_have ‹□(φ → □ψ) → (◇φ → ψ)›
      by (meson "KBasic:13" "B◇" "Hypothetical Syllogism" "→I")
  }
  AOT_hence ‹□□(φ → □ψ) → □(◇φ → ψ)›
    by (rule RM)
  AOT_thus  ‹□(φ → □ψ) → □(◇φ → ψ)›
    using "4" "Hypothetical Syllogism" by blast
next
  AOT_modally_strict {
    AOT_have ‹□(◇φ → ψ) → (φ → □ψ)›
      by (meson "B" "Hypothetical Syllogism" "→I" "qml:1"[axiom_inst])
  }
  AOT_hence  ‹□□(◇φ → ψ) → □(φ → □ψ)›
    by (rule RM)
  AOT_thus ‹□(◇φ → ψ) → □(φ → □ψ)›
    using "4" "Hypothetical Syllogism" by blast
qed

AOT_theorem "derived-S5-rules:1":                                     (165.1)
  assumes ‹Γ ⊢□ ◇φ → ψ›
  shows ‹□Γ ⊢□ φ → □ψ›
proof -
  AOT_have ‹□Γ ⊢□ □◇φ → □ψ›
    using assms by (rule "RM:1[prem]")
  AOT_thus ‹□Γ ⊢□ φ → □ψ›
    using "B" "Hypothetical Syllogism" by blast
qed

AOT_theorem "derived-S5-rules:2":                                     (165.2)
  assumes ‹Γ ⊢□ φ → □ψ›
  shows ‹□Γ ⊢□ ◇φ → ψ›
proof -
  AOT_have ‹□Γ ⊢□ ◇φ → ◇□ψ›
    using assms by (rule "RM:2[prem]")
  AOT_thus ‹□Γ ⊢□ ◇φ → ψ›
    using "B◇" "Hypothetical Syllogism" by blast
qed

AOT_theorem "BFs:1": ‹∀α □φ{α} → □∀α φ{α}›                            (166.1)
proof -
  AOT_modally_strict {
    AOT_have ‹◇∀α □φ{α} → ◇□φ{α}› for α
      using "cqt-orig:3" by (rule "RM◇")
    AOT_hence ‹◇∀α □φ{α} → ∀α φ{α}›
      using "B◇" "∀I" "→E" "→I" by metis
  }
  thus ?thesis
    using "derived-S5-rules:1" by blast
qed
lemmas "BF" = "BFs:1"

AOT_theorem "BFs:2": ‹□∀α φ{α} → ∀α □φ{α}›                            (166.2)
proof -
  AOT_have ‹□∀α φ{α} → □φ{α}› for α
```

```
    using RM "cqt-orig:3" by metis
  thus ?thesis
    using "cqt-orig:2"[THEN "→E"] "∀I" by metis
qed
lemmas "CBF" = "BFs:2"

AOT_theorem "BFs:3": ‹◇∃α φ{α} → ∃α ◇φ{α}›                              (166.3)
proof(rule "→I")
  AOT_modally_strict {
    AOT_have ‹□∀α ¬φ{α} ≡ ∀α □¬φ{α}›
      using BF CBF "≡I" by blast
  } note ϑ = this

  AOT_assume ‹◇∃α φ{α}›
  AOT_hence ‹¬□¬(∃α φ{α})›
    using "≡dfE" "conventions:5" by blast
  AOT_hence ‹¬□∀α ¬φ{α}›
    apply (AOT_subst ‹∀α ¬φ{α}› ‹¬(∃α φ{α})›)
    using "≡dfI" "conventions:3" "conventions:4" "&I"
          "contraposition:2" "cqt-further:4"
          "df-rules-formulas[3]" by blast
  AOT_hence ‹¬∀α □¬φ{α}›
    apply (AOT_subst (reverse) ‹∀α □¬φ{α}› ‹□∀α ¬φ{α}›)
    using ϑ by blast
  AOT_hence ‹¬∀α ¬¬□¬φ{α}›
    by (AOT_subst (reverse) ‹¬¬□¬φ{α}› ‹□¬φ{α}› for: α)
       (simp add: "oth-class-taut:3:b")
  AOT_hence ‹∃α ¬□¬φ{α}›
    by (rule "conventions:4"[THEN "≡dfI"])
  AOT_thus ‹∃α ◇φ{α}›
    using "conventions:5"[THEN "≡Df"]
    by (AOT_subst ‹◇φ{α}› ‹¬□¬φ{α}› for: α)
qed
lemmas "BF◇" = "BFs:3"

AOT_theorem "BFs:4": ‹∃α ◇φ{α} → ◇∃α φ{α}›                              (166.4)
proof(rule "→I")
  AOT_assume ‹∃α ◇φ{α}›
  AOT_hence ‹¬∀α ¬◇φ{α}›
    using "conventions:4"[THEN "≡dfE"] by blast
  AOT_hence ‹¬∀α □¬φ{α}›
    using "KBasic2:1"
    by (AOT_subst ‹□¬φ{α}› ‹¬◇φ{α}› for: α)
  moreover AOT_have ‹∀α □¬φ{α} ≡ □∀α ¬φ{α}›
    using "≡I" "BF" "CBF" by metis
  ultimately AOT_have 1: ‹¬□∀α ¬φ{α}›
    using "≡E"(3) by blast
  AOT_show ‹◇∃α φ{α}›
    apply (rule "conventions:5"[THEN "≡dfI"])
    apply (AOT_subst ‹∃α φ{α}› ‹¬∀α ¬φ{α}›)
     apply (simp add: "conventions:4" "≡Df")
    apply (AOT_subst ‹¬¬∀α ¬φ{α}› ‹∀α ¬φ{α}›)
    by (auto simp: 1 "≡I" "useful-tautologies:1" "useful-tautologies:2")
qed
lemmas "CBF◇" = "BFs:4"

AOT_theorem "sign-S5-thm:1": ‹∃α □φ{α} → □∃α φ{α}›                     (167.1)
proof(rule "→I")
  AOT_assume ‹∃α □φ{α}›
  then AOT_obtain α where ‹□φ{α}› using "∃E" by metis
  moreover AOT_have ‹□α↓›
    by (simp add: "ex:1:a" "rule-ui:2[const_var]" RN)
  moreover AOT_have ‹□φ{τ}, □τ↓ ⊢□ □∃α φ{α}› for τ
```

```
  proof -
    AOT_have ‹φ{τ}, τ↓ ⊢□ ∃α φ{α}› using "existential:1" by blast
    AOT_thus ‹□φ{τ}, □τ↓ ⊢□ □∃α φ{α}›
      using "RN[prem]"[where Γ="{φ τ, «τ↓»}", simplified] by blast
  qed
  ultimately AOT_show ‹□∃α φ{α}› by blast
qed
lemmas Buridan = "sign-S5-thm:1"

AOT_theorem "sign-S5-thm:2": ‹◇∀α φ{α} → ∀α ◇φ{α}›                    (167.2)
proof -
  AOT_have ‹∀α (◇∀α φ{α} → ◇φ{α})›
    by (simp add: "RM◇" "cqt-orig:3" "∀I")
  AOT_thus ‹◇∀α φ{α} → ∀α ◇φ{α}›
    using "∀E"(4) "∀I" "→E" "→I" by metis
qed
lemmas "Buridan◇" = "sign-S5-thm:2"

AOT_theorem "sign-S5-thm:3":                                           (167.3)
  ‹◇∃α (φ{α} & ψ{α}) → ◇(∃α φ{α} & ∃α ψ{α})›
  apply (rule "RM:2")
  by (metis (no_types, lifting) "∃E" "&I" "&E"(1) "&E"(2) "→I" "∃I"(2))

AOT_theorem "sign-S5-thm:4": ‹◇∃α (φ{α} & ψ{α}) → ◇∃α φ{α}›          (167.4)
  apply (rule "RM:2")
  by (meson "instantiation" "&E"(1) "→I" "∃I"(2))

AOT_theorem "sign-S5-thm:5":                                           (167.5)
  ‹(□∀α (φ{α} → ψ{α}) & □∀α (ψ{α} → χ{α})) → □∀α (φ{α} → χ{α})›
proof -
  {
    fix φ' ψ' χ'
    AOT_assume ‹⊢□ φ' & ψ' → χ'›
    AOT_hence ‹□φ' & □ψ' → □χ'›
      using "RN[prem]"[where Γ="{φ', ψ'}"] apply simp
      using "&E" "&I" "→E" "→I" by metis
  } note R = this
  show ?thesis by (rule R; fact AOT)
qed

AOT_theorem "sign-S5-thm:6":                                           (167.6)
  ‹(□∀α (φ{α} ≡ ψ{α}) & □∀α(ψ{α} ≡ χ{α})) → □∀α(φ{α} ≡ χ{α})›
proof -
  {
    fix φ' ψ' χ'
    AOT_assume ‹⊢□ φ' & ψ' → χ'›
    AOT_hence ‹□φ' & □ψ' → □χ'›
      using "RN[prem]"[where Γ="{φ', ψ'}"] apply simp
      using "&E" "&I" "→E" "→I" by metis
  } note R = this
  show ?thesis by (rule R; fact AOT)
qed

AOT_theorem "exist-nec2:1": ‹◇τ↓ → τ↓›                                 (168.1)
  using "B◇" "RM◇" "Hypothetical Syllogism" "exist-nec" by blast

AOT_theorem "exists-nec2:2": ‹◇τ↓ ≡ □τ↓›
  by (meson "Act-Sub:3" "Hypothetical Syllogism" "exist-nec"
            "exist-nec2:1" "≡I" "nec-imp-act")

AOT_theorem "exists-nec2:3": ‹¬τ↓ → □¬τ↓›
  using "KBasic2:1" "→I" "exist-nec2:1" "≡E"(2) "modus-tollens:1" by blast
```

```
AOT_theorem "exists-nec2:4": ‹◇¬τ↓ ≡ □¬τ↓›
  by (metis "Act-Sub:3" "KBasic:12" "→I" "exist-nec" "exists-nec2:3"
            "≡I" "≡E"(4) "nec-imp-act" "reductio-aa:1")

AOT_theorem "id-nec2:1": ‹◇α = β → α = β›                                   (169.1)
  using "B◇" "RM◇" "Hypothetical Syllogism" "id-nec:1" by blast

AOT_theorem "id-nec2:2": ‹α ≠ β → □α ≠ β›                                   (169.2)
  apply (AOT_subst ‹α ≠ β› ‹¬(α = β)›)
  using "=-infix"[THEN "≡Df"] apply blast
  using "KBasic2:1" "→I" "id-nec2:1" "≡E"(2) "modus-tollens:1" by blast

AOT_theorem "id-nec2:3": ‹◇α ≠ β → α ≠ β›                                   (169.3)
  apply (AOT_subst ‹α ≠ β› ‹¬(α = β)›)
  using "=-infix"[THEN "≡Df"] apply blast
  by (metis "KBasic:11" "→I" "id-nec:2" "≡E"(3) "reductio-aa:2" "→E")

AOT_theorem "id-nec2:4": ‹◇α = β → □α = β›                                  (169.4)
  using "Hypothetical Syllogism" "id-nec2:1" "id-nec:1" by blast

AOT_theorem "id-nec2:5": ‹◇α ≠ β → □α ≠ β›                                  (169.5)
  using "id-nec2:3" "id-nec2:2" "→I" "→E" by metis

AOT_theorem "sc-eq-box-box:1": ‹□(φ → □φ) ≡ (◇φ → □φ)›                     (170.1)
  apply (rule "≡I"; rule "→I")
  using "KBasic:13" "5◇" "Hypothetical Syllogism" "→E" apply blast
  by (metis "KBasic2:1" "KBasic:1" "KBasic:2" "S5Basic:13" "≡E"(2)
            "raa-cor:5" "→E")

AOT_theorem "sc-eq-box-box:2": ‹(□(φ → □φ) ∨ (◇φ → □φ)) → (◇φ ≡ □φ)›     (170.2)
  by (metis "Act-Sub:3" "KBasic:13" "5◇" "∨E"(2) "→I" "≡I"
            "nec-imp-act" "raa-cor:2" "→E")

AOT_theorem "sc-eq-box-box:3": ‹□(φ → □φ) → (¬□φ ≡ □¬φ)›                   (170.3)
proof (rule "→I"; rule "≡I"; rule "→I")
  AOT_assume ‹□(φ → □φ)›
  AOT_hence ‹◇φ → □φ› using "sc-eq-box-box:1" "≡E" by blast
  moreover AOT_assume ‹¬□φ›
  ultimately AOT_have ‹¬◇φ›
    using "modus-tollens:1" by blast
  AOT_thus ‹□¬φ›
    using "KBasic2:1" "≡E"(2) by blast
next
  AOT_assume ‹□(φ → □φ)›
  moreover AOT_assume ‹□¬φ›
  ultimately AOT_show ‹¬□φ›
    using "modus-tollens:1" "qml:2"[axiom_inst] "→E" by blast
qed

AOT_theorem "sc-eq-box-box:4":                                               (170.4)
  ‹(□(φ → □φ) & □(ψ → □ψ)) → ((□φ ≡ □ψ) → □(φ ≡ ψ))›
proof(rule "→I"; rule "→I")
  AOT_assume θ: ‹□(φ → □φ) & □(ψ → □ψ)›
  AOT_assume ξ: ‹□φ ≡ □ψ›
  AOT_hence ‹(□φ & □ψ) ∨ (¬□φ & ¬□ψ)›
    using "≡E"(4) "oth-class-taut:4:g" "raa-cor:3" by blast
  moreover {
    AOT_assume ‹□φ & □ψ›
    AOT_hence ‹□(φ ≡ ψ)›
      using "KBasic:3" "KBasic:8" "≡E"(2) "vdash-properties:10" by blast
  }
  moreover {
    AOT_assume ‹¬□φ & ¬□ψ›
```

```
    moreover AOT_have ‹¬□φ ≡ □¬φ› and ‹¬□ψ ≡ □¬ψ›
      using ϑ "Conjunction Simplification"(1,2)
            "sc-eq-box-box:3" "→E" by metis+
    ultimately AOT_have ‹□¬φ & □¬ψ›
      by (metis "&I" "Conjunction Simplification"(1,2)
                "≡E"(4) "modus-tollens:1" "raa-cor:3")
    AOT_hence ‹□(φ ≡ ψ)›
      using "KBasic:3" "KBasic:9" "≡E"(2) "→E" by blast
  }
  ultimately AOT_show ‹□(φ ≡ ψ)›
    using "∨E"(2) "reductio-aa:1" by blast
qed

AOT_theorem "sc-eq-box-box:5":                                                        (170.5)
  ‹(□(φ → □φ) & □(ψ → □ψ)) → □((φ ≡ ψ) → □(φ ≡ ψ))›
proof (rule "→I")
  AOT_assume ‹(□(φ → □φ) & □(ψ → □ψ))›
  AOT_hence ‹□(□(φ → □φ) & □(ψ → □ψ))›
    using 4[THEN "→E"] "&E" "&I" "KBasic:3" "≡E"(2) by metis
  moreover AOT_have ‹□(□(φ → □φ) & □(ψ → □ψ)) → □((φ ≡ ψ) → □(φ ≡ ψ))›
  proof (rule RM; rule "→I"; rule "→I")
    AOT_modally_strict {
      AOT_assume A: ‹(□(φ → □φ) & □(ψ → □ψ))›
      AOT_hence ‹φ → □φ› and ‹ψ → □ψ›
        using "&E" "qml:2"[axiom_inst] "→E" by blast+
      moreover AOT_assume ‹φ ≡ ψ›
      ultimately AOT_have ‹□φ ≡ □ψ›
        using "→E" "qml:2"[axiom_inst] "≡E" "≡I" by meson
      moreover AOT_have ‹(□φ ≡ □ψ) → □(φ ≡ ψ)›
        using A "sc-eq-box-box:4" "→E" by blast
      ultimately AOT_show ‹□(φ ≡ ψ)› using "→E" by blast
    }
  qed
  ultimately AOT_show ‹□((φ ≡ ψ) → □(φ ≡ ψ))› using "→E" by blast
qed

AOT_theorem "sc-eq-box-box:6": ‹□(φ → □φ) → ((φ → □ψ) → □(φ → ψ))›                    (170.6)
proof (rule "→I"; rule "→I"; rule "raa-cor:1")
  AOT_assume ‹¬□(φ → ψ)›
  AOT_hence ‹◇¬(φ → ψ)›
    by (metis "KBasic:11" "≡E"(1))
  AOT_hence ‹◇(φ & ¬ψ)›
    by (AOT_subst ‹φ & ¬ψ› ‹¬(φ → ψ)›)
       (meson "Commutativity of ≡" "≡E"(1) "oth-class-taut:1:b")
  AOT_hence ‹◇φ› and 2: ‹◇¬ψ›
    using "KBasic2:3"[THEN "→E"] "&E" by blast+
  moreover AOT_assume ‹□(φ → □φ)›
  ultimately AOT_have ‹□φ›
    by (metis "≡E"(1) "sc-eq-box-box:1" "→E")
  AOT_hence φ
    using "qml:2"[axiom_inst, THEN "→E"] by blast
  moreover AOT_assume ‹φ → □ψ›
  ultimately AOT_have ‹□ψ›
    using "→E" by blast
  moreover AOT_have ‹¬□ψ›
    using 2 "KBasic:12" "¬¬I" "intro-elim:3:d" by blast
  ultimately AOT_show ‹□ψ & ¬□ψ›
    using "&I" by blast
qed

AOT_theorem "sc-eq-box-box:7": ‹□(φ → □φ) → ((φ → 𝒜ψ) → 𝒜(φ → ψ))›                    (170.7)
proof (rule "→I"; rule "→I"; rule "raa-cor:1")
  AOT_assume ‹¬𝒜(φ → ψ)›
```

```
  AOT_hence ‹𝒜¬(φ → ψ)›
    by (metis "Act-Basic:1" "∨E"(2))
  AOT_hence ‹𝒜(φ & ¬ψ)›
    by (AOT_subst ‹φ & ¬ψ› ‹¬(φ → ψ)›)
       (meson "Commutativity of ≡" "≡E"(1) "oth-class-taut:1:b")
  AOT_hence ‹𝒜φ› and 2: ‹𝒜¬ψ›
    using "Act-Basic:2"[THEN "≡E"(1)] "&E" by blast+
  AOT_hence ‹◇φ›
    by (metis "Act-Sub:3" "→E")
  moreover AOT_assume ‹□(φ → □φ)›
  ultimately AOT_have ‹□φ›
    by (metis "≡E"(1) "sc-eq-box-box:1" "→E")
  AOT_hence φ
    using "qml:2"[axiom_inst, THEN "→E"] by blast
  moreover AOT_assume ‹φ → 𝒜ψ›
  ultimately AOT_have ‹𝒜ψ›
    using "→E" by blast
  moreover AOT_have ‹¬𝒜ψ›
    using 2 by (meson "Act-Sub:1" "≡E"(4) "raa-cor:3")
  ultimately AOT_show ‹𝒜ψ & ¬𝒜ψ›
    using "&I" by blast
qed

AOT_theorem "sc-eq-fur:1": ‹◇𝒜φ ≡ □𝒜φ›                                   (172.1)
  using "Act-Basic:6" "Act-Sub:4" "≡E"(6) by blast

AOT_theorem "sc-eq-fur:2": ‹□(φ → □φ) → (𝒜φ ≡ φ)›                        (172.2)
  by (metis "B◇" "Act-Sub:3" "KBasic:13" "T◇" "Hypothetical Syllogism"
            "→I" "≡I" "nec-imp-act")

AOT_theorem "sc-eq-fur:3":                                                (172.3)
  ‹□∀x (φ{x} → □φ{x}) → (∃!x φ{x} → ιx φ{x}↓)›
proof (rule "→I"; rule "→I")
  AOT_assume ‹□∀x (φ{x} → □φ{x})›
  AOT_hence A: ‹∀x □(φ{x} → □φ{x})›
    using CBF "→E" by blast
  AOT_assume ‹∃!x φ{x}›
  then AOT_obtain a where a_def: ‹φ{a} & ∀y (φ{y} → y = a)›
    using "∃E"[rotated 1, OF "uniqueness:1"[THEN "≡dfE"]] by blast
  moreover AOT_have ‹□φ{a}›
    using calculation A "∀E"(2) "qml:2"[axiom_inst] "→E" "&E"(1) by blast
  AOT_hence ‹𝒜φ{a}›
    using "nec-imp-act" "→E" by blast
  moreover AOT_have ‹∀y (𝒜φ{y} → y = a)›
  proof (rule "∀I"; rule "→I")
    fix b
    AOT_assume ‹𝒜φ{b}›
    AOT_hence ‹◇φ{b}›
      using "Act-Sub:3" "→E" by blast
    moreover {
      AOT_have ‹□(φ{b} → □φ{b})›
        using A "∀E"(2) by blast
      AOT_hence ‹◇φ{b} → □φ{b}›
        using "KBasic:13" "5◇" "Hypothetical Syllogism" "→E" by blast
    }
    ultimately AOT_have ‹□φ{b}›
      using "→E" by blast
    AOT_hence ‹φ{b}›
      using "qml:2"[axiom_inst] "→E" by blast
    AOT_thus ‹b = a›
      using a_def[THEN "&E"(2)] "∀E"(2) "→E" by blast
  qed
  ultimately AOT_have ‹𝒜φ{a} & ∀y (𝒜φ{y} → y = a)›
```

```
    using "&I" by blast
  AOT_hence ‹∃x (𝒜φ{x} & ∀y (𝒜φ{y} → y = x))›
    using "∃I" by fast
  AOT_hence ‹∃!x 𝒜φ{x}›
    using "uniqueness:1"[THEN "≡dfI"] by fast
  AOT_thus ‹ιx φ{x}↓›
    using "actual-desc:1"[THEN "≡E"(2)] by blast
qed

AOT_theorem "sc-eq-fur:4":                                          (172.4)
‹□∀x (φ{x} → □φ{x}) → (x = ιx φ{x} ≡ (φ{x} & ∀z (φ{z} → z = x)))›
proof (rule "→I")
  AOT_assume ‹□∀x (φ{x} → □φ{x})›
  AOT_hence ‹∀x □(φ{x} → □φ{x})›
    using CBF "→E" by blast
  AOT_hence A: ‹𝒜φ{α} ≡ φ{α}› for α
    using "sc-eq-fur:2" "∀E" "→E" by fast
  AOT_show ‹x = ιx φ{x} ≡ (φ{x} & ∀z (φ{z} → z = x))›
  proof (rule "≡I"; rule "→I")
    AOT_assume ‹x = ιx φ{x}›
    AOT_hence B: ‹𝒜φ{x} & ∀z (𝒜φ{z} → z = x)›
      using "nec-hintikka-scheme"[THEN "≡E"(1)] by blast
    AOT_show ‹φ{x} & ∀z (φ{z} → z = x)›
    proof (rule "&I"; (rule "∀I"; rule "→I")?)
      AOT_show ‹φ{x}›
        using A B[THEN "&E"(1)] "≡E"(1) by blast
    next
      AOT_show ‹z = x› if ‹φ{z}› for z
        using that B[THEN "&E"(2)] "∀E"(2) "→E" A[THEN "≡E"(2)] by blast
    qed
  next
    AOT_assume B: ‹φ{x} & ∀z (φ{z} → z = x)›
    AOT_have ‹𝒜φ{x} & ∀z (𝒜φ{z} → z = x)›
    proof(rule "&I"; (rule "∀I"; rule "→I")?)
      AOT_show ‹𝒜φ{x}›
        using B[THEN "&E"(1)] A[THEN "≡E"(2)] by blast
    next
      AOT_show ‹b = x› if ‹𝒜φ{b}› for b
        using A[THEN "≡E"(1)] that
              B[THEN "&E"(2), THEN "∀E"(2), THEN "→E"] by blast
    qed
    AOT_thus ‹x = ιx φ{x}›
      using "nec-hintikka-scheme"[THEN "≡E"(2)] by blast
  qed
qed

AOT_theorem "id-act:1": ‹α = β ≡ 𝒜α = β›                            (173.1)
  by (meson "Act-Sub:3" "Hypothetical Syllogism"
            "id-nec2:1" "id-nec:2" "≡I" "nec-imp-act")

AOT_theorem "id-act:2": ‹α ≠ β ≡ 𝒜α ≠ β›                            (173.2)
proof (AOT_subst ‹α ≠ β› ‹¬(α = β)›)
  AOT_modally_strict {
    AOT_show ‹α ≠ β ≡ ¬(α = β)›
      by (simp add: "=-infix" "≡Df")
  }
next
  AOT_show ‹¬(α = β) ≡ 𝒜¬(α = β)›
  proof (safe intro!: "≡I" "→I")
    AOT_assume ‹¬α = β›
    AOT_hence ‹¬𝒜α = β› using "id-act:1" "≡E"(3) by blast
    AOT_thus ‹𝒜¬α = β›
      using "¬¬E" "Act-Sub:1" "≡E"(3) by blast
```

```
  next
    AOT_assume ‹𝒜¬α = β›
    AOT_hence ‹¬𝒜α = β›
      using "¬¬I" "Act-Sub:1" "≡E"(4) by blast
    AOT_thus ‹¬α = β›
      using "id-act:1" "≡E"(4) by blast
  qed
qed

AOT_theorem "A-Exists:1": ‹𝒜∃!α φ{α} ≡ ∃!α 𝒜φ{α}›                    (174.1)
proof -
  AOT_have ‹𝒜∃!α φ{α} ≡ 𝒜∃α∀β (φ{β} ≡ β = α)›
    by (AOT_subst ‹∃!α φ{α}› ‹∃α∀β (φ{β} ≡ β = α)›)
       (auto simp add: "oth-class-taut:3:a" "uniqueness:2")
  also AOT_have ‹... ≡ ∃α 𝒜∀β (φ{β} ≡ β = α)›
    by (simp add: "Act-Basic:10")
  also AOT_have ‹... ≡ ∃α∀β 𝒜(φ{β} ≡ β = α)›
    by (AOT_subst ‹𝒜∀β (φ{β} ≡ β = α)› ‹∀β 𝒜(φ{β} ≡ β = α)› for: α)
       (auto simp: "logic-actual-nec:3"[axiom_inst] "oth-class-taut:3:a")
  also AOT_have ‹... ≡ ∃α∀β (𝒜φ{β} ≡ 𝒜β = α)›
    by (AOT_subst (reverse) ‹𝒜φ{β} ≡ 𝒜β = α›
                            ‹𝒜(φ{β} ≡ β = α)› for: α β :: 'a)
       (auto simp: "Act-Basic:5" "cqt-further:7")
  also AOT_have ‹... ≡ ∃α∀β (𝒜φ{β} ≡ β = α)›
    by (AOT_subst (reverse) ‹𝒜β = α› ‹β = α› for: α β :: 'a)
       (auto simp: "id-act:1" "cqt-further:7")
  also AOT_have ‹... ≡ ∃!α 𝒜φ{α}›
    using "uniqueness:2" "Commutativity of ≡"[THEN "≡E"(1)] by fast
  finally show ?thesis.
qed

AOT_theorem "A-Exists:2": ‹ιx φ{x}↓ ≡ 𝒜∃!x φ{x}›                      (174.2)
  by (AOT_subst ‹𝒜∃!x φ{x}› ‹∃!x 𝒜φ{x}›)
     (auto simp: "actual-desc:1" "A-Exists:1")

AOT_theorem "id-act-desc:1": ‹ιx (x = y)↓›                             (175.1)
proof(rule "existence:1"[THEN "≡dfI"]; rule "∃I")
  AOT_show ‹[λx E!x → E!x]ιx (x = y)›
  proof (rule "russell-axiom[exe,1].nec-russell-axiom"[THEN "≡E"(2)];
         rule "∃I"; (rule "&I")+)
    AOT_show ‹𝒜y = y› by (simp add: "RA[2]" "id-eq:1")
  next
    AOT_show ‹∀z (𝒜z = y → z = y)›
      apply (rule "∀I")
      using "id-act:1"[THEN "≡E"(2)] "→I" by blast
  next
    AOT_show ‹[λx E!x → E!x]y›
    proof (rule "lambda-predicates:2"[axiom_inst, THEN "→E", THEN "≡E"(2)])
      AOT_show ‹[λx E!x → E!x]↓›
        by "cqt:2[lambda]"
    next
      AOT_show ‹E!y → E!y›
        by (simp add: "if-p-then-p")
    qed
  qed
next
  AOT_show ‹[λx E!x → E!x]↓›
    by "cqt:2[lambda]"
qed

AOT_theorem "id-act-desc:2": ‹y = ιx (x = y)›                          (175.2)
  by (rule descriptions[axiom_inst, THEN "≡E"(2)];
      rule "∀I"; rule "id-act:1"[symmetric])
```

```
AOT_theorem "pre-en-eq:1[1]": ‹x1[F] → □x1[F]›                                    (176.1)
  by (simp add: encoding "vdash-properties:1[2]")

AOT_theorem "pre-en-eq:1[2]": ‹x1x2[F] → □x1x2[F]›                                (176.1)
proof (rule "→I")
  AOT_assume ‹x1x2[F]›
  AOT_hence ‹x1[λy [F]yx2]› and ‹x2[λy [F]x1y]›
    using "nary-encoding[2]"[axiom_inst, THEN "≡E"(1)] "&E" by blast+
  moreover AOT_have ‹[λy [F]yx2]↓› by "cqt:2"
  moreover AOT_have ‹[λy [F]x1y]↓› by "cqt:2"
  ultimately AOT_have ‹□x1[λy [F]yx2]› and ‹□x2[λy [F]x1y]›
    using encoding[axiom_inst, unvarify F] "→E" "&I" by blast+
  note A = this
  AOT_hence ‹□(x1[λy [F]yx2] & x2[λy [F]x1y])›
    using "KBasic:3"[THEN "≡E"(2)] "&I" by blast
  AOT_thus ‹□x1x2[F]›
    by (rule "nary-encoding[2]"[axiom_inst, THEN RN,
                                THEN "KBasic:6"[THEN "→E"],
                                THEN "≡E"(2)])
qed

AOT_theorem "pre-en-eq:1[3]": ‹x1x2x3[F] → □x1x2x3[F]›                            (176.1)
proof (rule "→I")
  AOT_assume ‹x1x2x3[F]›
  AOT_hence ‹x1[λy [F]yx2x3]›
        and ‹x2[λy [F]x1yx3]›
        and ‹x3[λy [F]x1x2y]›
    using "nary-encoding[3]"[axiom_inst, THEN "≡E"(1)] "&E" by blast+
  moreover AOT_have ‹[λy [F]yx2x3]↓› by "cqt:2"
  moreover AOT_have ‹[λy [F]x1yx3]↓› by "cqt:2"
  moreover AOT_have ‹[λy [F]x1x2y]↓› by "cqt:2"
  ultimately AOT_have ‹□x1[λy [F]yx2x3]›
                  and ‹□x2[λy [F]x1yx3]›
                  and ‹□x3[λy [F]x1x2y]›
    using encoding[axiom_inst, unvarify F] "→E" by blast+
  note A = this
  AOT_have B: ‹□(x1[λy [F]yx2x3] & x2[λy [F]x1yx3] & x3[λy [F]x1x2y])›
    by (rule "KBasic:3"[THEN "≡E"(2)] "&I" A)+
  AOT_thus ‹□x1x2x3[F]›
    by (rule "nary-encoding[3]"[axiom_inst, THEN RN,
                THEN "KBasic:6"[THEN "→E"], THEN "≡E"(2)])
qed

AOT_theorem "pre-en-eq:1[4]": ‹x1x2x3x4[F] → □x1x2x3x4[F]›                        (176.1)
proof (rule "→I")
  AOT_assume ‹x1x2x3x4[F]›
  AOT_hence ‹x1[λy [F]yx2x3x4]›
        and ‹x2[λy [F]x1yx3x4]›
        and ‹x3[λy [F]x1x2yx4]›
        and ‹x4[λy [F]x1x2x3y]›
    using "nary-encoding[4]"[axiom_inst, THEN "≡E"(1)] "&E" by metis+
  moreover AOT_have ‹[λy [F]yx2x3x4]↓› by "cqt:2"
  moreover AOT_have ‹[λy [F]x1yx3x4]↓› by "cqt:2"
  moreover AOT_have ‹[λy [F]x1x2yx4]↓› by "cqt:2"
  moreover AOT_have ‹[λy [F]x1x2x3y]↓› by "cqt:2"
  ultimately AOT_have ‹□x1[λy [F]yx2x3x4]›
                  and ‹□x2[λy [F]x1yx3x4]›
                  and ‹□x3[λy [F]x1x2yx4]›
                  and ‹□x4[λy [F]x1x2x3y]›
    using "→E" encoding[axiom_inst, unvarify F] by blast+
  note A = this
  AOT_have B: ‹□(x1[λy [F]yx2x3x4] &
```

```
                  x2[λy [F]x1yx3x4] &
                  x3[λy [F]x1x2yx4] &
                  x4[λy [F]x1x2x3y])›
    by (rule "KBasic:3"[THEN "≡E"(2)] "&I" A)+
  AOT_thus ‹□x1x2x3x4[F]›
    by (rule "nary-encoding[4]"[axiom_inst, THEN RN,
              THEN "KBasic:6"[THEN "→E"], THEN "≡E"(2)])
qed

AOT_theorem "pre-en-eq:2[1]": ‹¬x1[F] → □¬x1[F]›                    (176.2)
proof (rule "→I"; rule "raa-cor:1")
  AOT_assume ‹¬□¬x1[F]›
  AOT_hence ‹◊x1[F]›
    by (rule "conventions:5"[THEN "≡dfI"])
  AOT_hence ‹x1[F]›
    by(rule "S5Basic:13"[THEN "≡E"(1), OF  "pre-en-eq:1[1]"[THEN RN],
              THEN "qml:2"[axiom_inst, THEN "→E"], THEN "→E"])
  moreover AOT_assume ‹¬x1[F]›
  ultimately AOT_show ‹x1[F] & ¬x1[F]› by (rule "&I")
qed
AOT_theorem "pre-en-eq:2[2]": ‹¬x1x2[F] → □¬x1x2[F]›                (176.2)
proof (rule "→I"; rule "raa-cor:1")
  AOT_assume ‹¬□¬x1x2[F]›
  AOT_hence ‹◊x1x2[F]›
    by (rule "conventions:5"[THEN "≡dfI"])
  AOT_hence ‹x1x2[F]›
    by(rule "S5Basic:13"[THEN "≡E"(1), OF  "pre-en-eq:1[2]"[THEN RN],
              THEN "qml:2"[axiom_inst, THEN "→E"], THEN "→E"])
  moreover AOT_assume ‹¬x1x2[F]›
  ultimately AOT_show ‹x1x2[F] & ¬x1x2[F]› by (rule "&I")
qed

AOT_theorem "pre-en-eq:2[3]": ‹¬x1x2x3[F] → □¬x1x2x3[F]›            (176.2)
proof (rule "→I"; rule "raa-cor:1")
  AOT_assume ‹¬□¬x1x2x3[F]›
  AOT_hence ‹◊x1x2x3[F]›
    by (rule "conventions:5"[THEN "≡dfI"])
  AOT_hence ‹x1x2x3[F]›
    by(rule "S5Basic:13"[THEN "≡E"(1), OF  "pre-en-eq:1[3]"[THEN RN],
              THEN "qml:2"[axiom_inst, THEN "→E"], THEN "→E"])
  moreover AOT_assume ‹¬x1x2x3[F]›
  ultimately AOT_show ‹x1x2x3[F] & ¬x1x2x3[F]› by (rule "&I")
qed

AOT_theorem "pre-en-eq:2[4]": ‹¬x1x2x3x4[F] → □¬x1x2x3x4[F]›        (176.2)
proof (rule "→I"; rule "raa-cor:1")
  AOT_assume ‹¬□¬x1x2x3x4[F]›
  AOT_hence ‹◊x1x2x3x4[F]›
    by (rule "conventions:5"[THEN "≡dfI"])
  AOT_hence ‹x1x2x3x4[F]›
    by(rule "S5Basic:13"[THEN "≡E"(1), OF  "pre-en-eq:1[4]"[THEN RN],
                         THEN "qml:2"[axiom_inst, THEN "→E"], THEN "→E"])
  moreover AOT_assume ‹¬x1x2x3x4[F]›
  ultimately AOT_show ‹x1x2x3x4[F] & ¬x1x2x3x4[F]› by (rule "&I")
qed

AOT_theorem "en-eq:1[1]": ‹◊x1[F] ≡ □x1[F]›                         (177.1)
  using "pre-en-eq:1[1]"[THEN RN] "sc-eq-box-box:2" "∨I" "→E" by metis
AOT_theorem "en-eq:1[2]": ‹◊x1x2[F] ≡ □x1x2[F]›                     (177.1)
  using "pre-en-eq:1[2]"[THEN RN] "sc-eq-box-box:2" "∨I" "→E" by metis
AOT_theorem "en-eq:1[3]": ‹◊x1x2x3[F] ≡ □x1x2x3[F]›                 (177.1)
  using "pre-en-eq:1[3]"[THEN RN] "sc-eq-box-box:2" "∨I" "→E" by fast
AOT_theorem "en-eq:1[4]": ‹◊x1x2x3x4[F] ≡ □x1x2x3x4[F]›             (177.1)
```

```
  using "pre-en-eq:1[4]"[THEN RN] "sc-eq-box-box:2" "∨I" "→E" by fast

AOT_theorem "en-eq:2[1]": ‹x1[F] ≡ □x1[F]›                                    (177.2)
  by (simp add: "≡I" "pre-en-eq:1[1]" "qml:2"[axiom_inst])
AOT_theorem "en-eq:2[2]": ‹x1x2[F] ≡ □x1x2[F]›                                (177.2)
  by (simp add: "≡I" "pre-en-eq:1[2]" "qml:2"[axiom_inst])
AOT_theorem "en-eq:2[3]": ‹x1x2x3[F] ≡ □x1x2x3[F]›                            (177.2)
  by (simp add: "≡I" "pre-en-eq:1[3]" "qml:2"[axiom_inst])
AOT_theorem "en-eq:2[4]": ‹x1x2x3x4[F] ≡ □x1x2x3x4[F]›                        (177.2)
  by (simp add: "≡I" "pre-en-eq:1[4]" "qml:2"[axiom_inst])

AOT_theorem "en-eq:3[1]": ‹◇x1[F] ≡ x1[F]›                                    (177.3)
  using "T◇" "derived-S5-rules:2"[OF "pre-en-eq:1[1]"] "≡I" by blast
AOT_theorem "en-eq:3[2]": ‹◇x1x2[F] ≡ x1x2[F]›                                (177.3)
  using "T◇" "derived-S5-rules:2"[OF "pre-en-eq:1[2]"] "≡I" by blast
AOT_theorem "en-eq:3[3]": ‹◇x1x2x3[F] ≡ x1x2x3[F]›                            (177.3)
  using "T◇" "derived-S5-rules:2"[OF "pre-en-eq:1[3]"] "≡I" by blast
AOT_theorem "en-eq:3[4]": ‹◇x1x2x3x4[F] ≡ x1x2x3x4[F]›                        (177.3)
  using "T◇" "derived-S5-rules:2"[OF "pre-en-eq:1[4]"] "≡I" by blast

AOT_theorem "en-eq:4[1]":                                                      (177.4)
  ‹(x1[F] ≡ y1[G]) ≡ (□x1[F] ≡ □y1[G])›
  apply (rule "≡I"; rule "→I"; rule "≡I"; rule "→I")
  using "qml:2"[axiom_inst, THEN "→E"] "≡E"(1,2) "en-eq:2[1]" by blast+
AOT_theorem "en-eq:4[2]":                                                      (177.4)
  ‹(x1x2[F] ≡ y1y2[G]) ≡ (□x1x2[F] ≡ □y1y2[G])›
  apply (rule "≡I"; rule "→I"; rule "≡I"; rule "→I")
  using "qml:2"[axiom_inst, THEN "→E"] "≡E"(1,2) "en-eq:2[2]" by blast+
AOT_theorem "en-eq:4[3]":                                                      (177.4)
  ‹(x1x2x3[F] ≡ y1y2y3[G]) ≡ (□x1x2x3[F] ≡ □y1y2y3[G])›
  apply (rule "≡I"; rule "→I"; rule "≡I"; rule "→I")
  using "qml:2"[axiom_inst, THEN "→E"] "≡E"(1,2) "en-eq:2[3]" by blast+
AOT_theorem "en-eq:4[4]":                                                      (177.4)
  ‹(x1x2x3x4[F] ≡ y1y2y3y4[G]) ≡ (□x1x2x3x4[F] ≡ □y1y2y3y4[G])›
  apply (rule "≡I"; rule "→I"; rule "≡I"; rule "→I")
  using "qml:2"[axiom_inst, THEN "→E"] "≡E"(1,2) "en-eq:2[4]" by blast+

AOT_theorem "en-eq:5[1]":                                                      (177.5)
  ‹□(x1[F] ≡ y1[G]) ≡ (□x1[F] ≡ □y1[G])›
  apply (rule "≡I"; rule "→I")
  using "en-eq:4[1]"[THEN "≡E"(1)] "qml:2"[axiom_inst, THEN "→E"]
   apply blast
  using "sc-eq-box-box:4"[THEN "→E", THEN "→E"]
        "&I"[OF "pre-en-eq:1[1]"[THEN RN], OF "pre-en-eq:1[1]"[THEN RN]]
  by blast
AOT_theorem "en-eq:5[2]":                                                      (177.5)
  ‹□(x1x2[F] ≡ y1y2[G]) ≡ (□x1x2[F] ≡ □y1y2[G])›
  apply (rule "≡I"; rule "→I")
  using "en-eq:4[2]"[THEN "≡E"(1)] "qml:2"[axiom_inst, THEN "→E"]
   apply blast
  using "sc-eq-box-box:4"[THEN "→E", THEN "→E"]
        "&I"[OF "pre-en-eq:1[2]"[THEN RN], OF "pre-en-eq:1[2]"[THEN RN]]
  by blast
AOT_theorem "en-eq:5[3]":                                                      (177.5)
  ‹□(x1x2x3[F] ≡ y1y2y3[G]) ≡ (□x1x2x3[F] ≡ □y1y2y3[G])›
  apply (rule "≡I"; rule "→I")
  using "en-eq:4[3]"[THEN "≡E"(1)] "qml:2"[axiom_inst, THEN "→E"]
   apply blast
  using "sc-eq-box-box:4"[THEN "→E", THEN "→E"]
        "&I"[OF "pre-en-eq:1[3]"[THEN RN], OF "pre-en-eq:1[3]"[THEN RN]]
  by blast
AOT_theorem "en-eq:5[4]":                                                      (177.5)
  ‹□(x1x2x3x4[F] ≡ y1y2y3y4[G]) ≡ (□x1x2x3x4[F] ≡ □y1y2y3y4[G])›
```

```
  apply (rule "≡I"; rule "→I")
  using "en-eq:4[4]"[THEN "≡E"(1)] "qml:2"[axiom_inst, THEN "→E"]
   apply blast
  using "sc-eq-box-box:4"[THEN "→E", THEN "→E"]
        "&I"[OF "pre-en-eq:1[4]"[THEN RN], OF "pre-en-eq:1[4]"[THEN RN]]
  by blast

AOT_theorem "en-eq:6[1]":                                                    (177.6)
  ‹(x1[F] ≡ y1[G]) ≡ □(x1[F] ≡ y1[G])›
  using "en-eq:5[1]"[symmetric] "en-eq:4[1]" "≡E"(5) by fast
AOT_theorem "en-eq:6[2]":                                                    (177.6)
  ‹(x1x2[F] ≡ y1y2[G]) ≡ □(x1x2[F] ≡ y1y2[G])›
  using "en-eq:5[2]"[symmetric] "en-eq:4[2]" "≡E"(5) by fast
AOT_theorem "en-eq:6[3]":                                                    (177.6)
  ‹(x1x2x3[F] ≡ y1y2y3[G]) ≡ □(x1x2x3[F] ≡ y1y2y3[G])›
  using "en-eq:5[3]"[symmetric] "en-eq:4[3]" "≡E"(5) by fast
AOT_theorem "en-eq:6[4]":                                                    (177.6)
  ‹(x1x2x3x4[F] ≡ y1y2y3y4[G]) ≡ □(x1x2x3x4[F] ≡ y1y2y3y4[G])›
  using "en-eq:5[4]"[symmetric] "en-eq:4[4]" "≡E"(5) by fast

AOT_theorem "en-eq:7[1]": ‹¬x1[F] ≡ □¬x1[F]›                                 (177.7)
  using "pre-en-eq:2[1]" "qml:2"[axiom_inst] "≡I" by blast
AOT_theorem "en-eq:7[2]": ‹¬x1x2[F] ≡ □¬x1x2[F]›                             (177.7)
  using "pre-en-eq:2[2]" "qml:2"[axiom_inst] "≡I" by blast
AOT_theorem "en-eq:7[3]": ‹¬x1x2x3[F] ≡ □¬x1x2x3[F]›                         (177.7)
  using "pre-en-eq:2[3]" "qml:2"[axiom_inst] "≡I" by blast
AOT_theorem "en-eq:7[4]": ‹¬x1x2x3x4[F] ≡ □¬x1x2x3x4[F]›                     (177.7)
  using "pre-en-eq:2[4]" "qml:2"[axiom_inst] "≡I" by blast

AOT_theorem "en-eq:8[1]": ‹◇¬x1[F] ≡ ¬x1[F]›                                 (177.8)
  using "en-eq:2[1]"[THEN "oth-class-taut:4:b"[THEN "≡E"(1)]]
        "KBasic:11" "≡E"(5)[symmetric] by blast
AOT_theorem "en-eq:8[2]": ‹◇¬x1x2[F] ≡ ¬x1x2[F]›                             (177.8)
  using "en-eq:2[2]"[THEN "oth-class-taut:4:b"[THEN "≡E"(1)]]
        "KBasic:11" "≡E"(5)[symmetric] by blast
AOT_theorem "en-eq:8[3]": ‹◇¬x1x2x3[F] ≡ ¬x1x2x3[F]›                         (177.8)
  using "en-eq:2[3]"[THEN "oth-class-taut:4:b"[THEN "≡E"(1)]]
        "KBasic:11" "≡E"(5)[symmetric] by blast
AOT_theorem "en-eq:8[4]": ‹◇¬x1x2x3x4[F] ≡ ¬x1x2x3x4[F]›                     (177.8)
  using "en-eq:2[4]"[THEN "oth-class-taut:4:b"[THEN "≡E"(1)]]
        "KBasic:11" "≡E"(5)[symmetric] by blast

AOT_theorem "en-eq:9[1]": ‹◇¬x1[F] ≡ □¬x1[F]›                                (177.9)
  using "en-eq:7[1]" "en-eq:8[1]" "≡E"(5) by blast
AOT_theorem "en-eq:9[2]": ‹◇¬x1x2[F] ≡ □¬x1x2[F]›                            (177.9)
  using "en-eq:7[2]" "en-eq:8[2]" "≡E"(5) by blast
AOT_theorem "en-eq:9[3]": ‹◇¬x1x2x3[F] ≡ □¬x1x2x3[F]›                        (177.9)
  using "en-eq:7[3]" "en-eq:8[3]" "≡E"(5) by blast
AOT_theorem "en-eq:9[4]": ‹◇¬x1x2x3x4[F] ≡ □¬x1x2x3x4[F]›                    (177.9)
  using "en-eq:7[4]" "en-eq:8[4]" "≡E"(5) by blast

AOT_theorem "en-eq:10[1]": ‹𝒜x1[F] ≡ x1[F]›                                  (177.10)
  by (metis "Act-Sub:3" "deduction-theorem" "≡I" "≡E"(1)
            "nec-imp-act" "en-eq:3[1]" "pre-en-eq:1[1]")
AOT_theorem "en-eq:10[2]": ‹𝒜x1x2[F] ≡ x1x2[F]›                              (177.10)
  by (metis "Act-Sub:3" "deduction-theorem" "≡I" "≡E"(1)
            "nec-imp-act" "en-eq:3[2]" "pre-en-eq:1[2]")
AOT_theorem "en-eq:10[3]": ‹𝒜x1x2x3[F] ≡ x1x2x3[F]›                          (177.10)
  by (metis "Act-Sub:3" "deduction-theorem" "≡I" "≡E"(1)
            "nec-imp-act" "en-eq:3[3]" "pre-en-eq:1[3]")
AOT_theorem "en-eq:10[4]": ‹𝒜x1x2x3x4[F] ≡ x1x2x3x4[F]›                      (177.10)
  by (metis "Act-Sub:3" "deduction-theorem" "≡I" "≡E"(1)
            "nec-imp-act" "en-eq:3[4]" "pre-en-eq:1[4]")
```

```
AOT_theorem "oa-facts:1": ‹O!x → □O!x›                                    (178.1)
proof(rule "→I")
  AOT_modally_strict {
    AOT_have ‹[λx ◇E!x]x ≡ ◇E!x›
      by (rule "lambda-predicates:2"[axiom_inst, THEN "→E"]) "cqt:2"
  } note ϑ = this
  AOT_assume ‹O!x›
  AOT_hence ‹[λx ◇E!x]x›
    by (rule "=dfE"(2)[OF AOT_ordinary, rotated 1]) "cqt:2"
  AOT_hence ‹◇E!x› using ϑ[THEN "≡E"(1)] by blast
  AOT_hence ‹□◇E!x› using "qml:3"[axiom_inst, THEN "→E"] by blast
  AOT_hence ‹□[λx ◇E!x]x›
    by (AOT_subst ‹[λx ◇E!x]x› ‹◇E!x›)
       (auto simp: ϑ)
  AOT_thus ‹□O!x›
    by (rule "=dfI"(2)[OF AOT_ordinary, rotated 1]) "cqt:2"
qed

AOT_theorem "oa-facts:2": ‹A!x → □A!x›                                    (178.2)
proof(rule "→I")
  AOT_modally_strict {
    AOT_have ‹[λx ¬◇E!x]x ≡ ¬◇E!x›
      by (rule "lambda-predicates:2"[axiom_inst, THEN "→E"]) "cqt:2"
  } note ϑ = this
  AOT_assume ‹A!x›
  AOT_hence ‹[λx ¬◇E!x]x›
    by (rule "=dfE"(2)[OF AOT_abstract, rotated 1]) "cqt:2"
  AOT_hence ‹¬◇E!x› using ϑ[THEN "≡E"(1)] by blast
  AOT_hence ‹□¬E!x› using "KBasic2:1"[THEN "≡E"(2)] by blast
  AOT_hence ‹□□¬E!x› using "4"[THEN "→E"] by blast
  AOT_hence ‹□¬◇E!x›
    using "KBasic2:1"
    by (AOT_subst (reverse) ‹¬◇E!x› ‹□¬E!x›) blast
  AOT_hence ‹□[λx ¬◇E!x]x›
    by (AOT_subst ‹[λx ¬◇E!x]x› ‹¬◇E!x›)
       (auto simp: ϑ)
  AOT_thus ‹□A!x›
    by (rule "=dfI"(2)[OF AOT_abstract, rotated 1]) "cqt:2[lambda]"
qed

AOT_theorem "oa-facts:3": ‹◇O!x → O!x›                                    (178.3)
  using "oa-facts:1" "B◇" "RM◇" "Hypothetical Syllogism" by blast
AOT_theorem "oa-facts:4": ‹◇A!x → A!x›                                    (178.4)
  using "oa-facts:2" "B◇" "RM◇" "Hypothetical Syllogism" by blast

AOT_theorem "oa-facts:5": ‹◇O!x ≡ □O!x›                                   (178.5)
  by (meson "Act-Sub:3" "Hypothetical Syllogism" "≡I" "nec-imp-act"
            "oa-facts:1" "oa-facts:3")

AOT_theorem "oa-facts:6": ‹◇A!x ≡ □A!x›                                   (178.6)
  by (meson "Act-Sub:3" "Hypothetical Syllogism" "≡I" "nec-imp-act"
            "oa-facts:2" "oa-facts:4")

AOT_theorem "oa-facts:7": ‹O!x ≡ 𝒜O!x›                                    (178.7)
  by (meson "Act-Sub:3" "Hypothetical Syllogism" "≡I" "nec-imp-act"
            "oa-facts:1" "oa-facts:3")

AOT_theorem "oa-facts:8": ‹A!x ≡ 𝒜A!x›                                    (178.8)
  by (meson "Act-Sub:3" "Hypothetical Syllogism" "≡I" "nec-imp-act"
            "oa-facts:2" "oa-facts:4")

subsection‹The Theory of Relations›
```

```
text‹\label{PLM: 9.10}›

AOT_theorem "beta-C-meta":                                                    (179)
  ‹[λμ1...μn φ{μ1...μn, ν1...νn}]↓ →
   ([λμ1...μn φ{μ1...μn, ν1...νn}]ν1...νn ≡ φ{ν1...νn, ν1...νn})›
  using "lambda-predicates:2"[axiom_inst] by blast

AOT_theorem "beta-C-cor:1":                                                   (181.1)
  ‹(∀ν1...∀νn([λμ1...μn φ{μ1...μn, ν1...νn}]↓)) →
   ∀ν1...∀νn ([λμ1...μn φ{μ1...μn, ν1...νn}]ν1...νn ≡ φ{ν1...νn, ν1...νn})›
  apply (rule "cqt-basic:14"[where 'a='a, THEN "→E"])
  using "beta-C-meta" "∀I" by fast

AOT_theorem "beta-C-cor:2":                                                   (181.2)
  ‹[λμ1...μn φ{μ1...μn}]↓ →
   ∀ν1...∀νn ([λμ1...μn φ{μ1...μn}]ν1...νn ≡ φ{ν1...νn})›
  apply (rule "→I"; rule "∀I")
  using "beta-C-meta"[THEN "→E"] by fast

(* TODO: add better syntax parsing for INSTANCE_OF_CQT_2 *)
theorem "beta-C-cor:3":                                                       (181.3)
  assumes ‹⋀ν1νn. AOT_instance_of_cqt_2 (φ (AOT_term_of_var ν1νn))›
  shows ‹[v ⊨ ∀ν1...∀νn ([λμ1...μn φ{ν1...νn, μ1...μn}]ν1...νn ≡
                        φ{ν1...νn, ν1...νn})]›
  using "cqt:2[lambda]"[axiom_inst, OF assms]
        "beta-C-cor:1"[THEN "→E"] "∀I" by fast

AOT_theorem "betaC:1:a": ‹[λμ1...μn φ{μ1...μn}]κ1...κn ⊢□ φ{κ1...κn}›        (182.1.a)
proof -
  AOT_modally_strict {
    AOT_assume ‹[λμ1...μn φ{μ1...μn}]κ1...κn›
    moreover AOT_have ‹[λμ1...μn φ{μ1...μn}]↓› and ‹κ1...κn↓›
      using calculation "cqt:5:a"[axiom_inst, THEN "→E"] "&E" by blast+
    ultimately AOT_show ‹φ{κ1...κn}›
      using "beta-C-cor:2"[THEN "→E", THEN "∀E"(1), THEN "≡E"(1)] by blast
  }
qed

AOT_theorem "betaC:1:b": ‹¬φ{κ1...κn} ⊢□ ¬[λμ1...μn φ{μ1...μn}]κ1...κn›      (182.1.b)
  using "betaC:1:a" "raa-cor:3" by blast

lemmas "β→C" = "betaC:1:a" "betaC:1:b"

AOT_theorem "betaC:2:a":                                                      (182.2.a)
  ‹[λμ1...μn φ{μ1...μn}]↓, κ1...κn↓, φ{κ1...κn} ⊢□
   [λμ1...μn φ{μ1...μn}]κ1...κn›
proof -
  AOT_modally_strict {
    AOT_assume 1: ‹[λμ1...μn φ{μ1...μn}]↓›
           and 2: ‹κ1...κn↓›
           and 3: ‹φ{κ1...κn}›
    AOT_hence ‹[λμ1...μn φ{μ1...μn}]κ1...κn›
      using "beta-C-cor:2"[THEN "→E", OF 1, THEN "∀E"(1), THEN "≡E"(2)]
      by blast
  }
  AOT_thus ‹[λμ1...μn φ{μ1...μn}]↓, κ1...κn↓, φ{κ1...κn} ⊢□
            [λμ1...μn φ{μ1...μn}]κ1...κn›
    by blast
qed

AOT_theorem "betaC:2:b":                                                      (182.2.b)
  ‹[λμ1...μn φ{μ1...μn}]↓, κ1...κn↓, ¬[λμ1...μn φ{μ1...μn}]κ1...κn ⊢□
   ¬φ{κ1...κn}›
```

```
  using "betaC:2:a" "raa-cor:3" by blast

lemmas "β←C" = "betaC:2:a" "betaC:2:b"

AOT_theorem "eta-conversion-lemma1:1": ‹Π↓ → [λx1...xn [Π]x1...xn] = Π›          (184.1)
  using "lambda-predicates:3"[axiom_inst] "∀I" "∀E"(1) "→I" by fast

(* Note: generalized alphabetic variant of the last theorem *)
AOT_theorem "eta-conversion-lemma1:2": ‹Π↓ → [λν1...νn [Π]ν1...νn] = Π›          (184.2)
  using "eta-conversion-lemma1:1".

text‹Note: not explicitly part of PLM.›
AOT_theorem id_sym:
  assumes ‹τ = τ'›
  shows ‹τ' = τ›
  using "rule=E"[where φ="λ τ' . «τ' = τ»", rotated 1, OF assms]
        "=I"(1)[OF "t=t-proper:1"[THEN "→E", OF assms]] by auto
declare id_sym[sym]

text‹Note: not explicitly part of PLM.›
AOT_theorem id_trans:
  assumes ‹τ = τ'› and ‹τ' = τ"›
  shows ‹τ = τ"›
  using "rule=E" assms by blast
declare id_trans[trans]

method "ηC" for Π :: ‹<'a::{AOT_Term_id_2,AOT_κs}>› =
  (match conclusion in "[v ⊨ τ{Π} = τ'{Π}]" for v τ τ' ⇒ ‹
   rule "rule=E"[rotated 1, OF "eta-conversion-lemma1:2"
    [THEN "→E", of v "«[Π]»", symmetric]]›)

AOT_theorem "sub-des-lam:1":                                                    (186.1)
  ‹[λz1...zn  χ{z1...zn, ιx φ{x}}]↓ & ιx φ{x} = ιx ψ{x} →
   [λz1...zn χ{z1...zn, ιx φ{x}}] = [λz1...zn χ{z1...zn, ιx ψ{x}}]›
proof(rule "→I")
  AOT_assume A: ‹[λz1...zn  χ{z1...zn, ιx φ{x}}]↓ & ιx φ{x} = ιx ψ{x}›
  AOT_show ‹[λz1...zn χ{z1...zn, ιx φ{x}}] = [λz1...zn χ{z1...zn, ιx ψ{x}}]›
    using "rule=E"[where φ="λ τ . «[λz1...zn χ{z1...zn, ιx φ{x}}] =
                                   [λz1...zn χ{z1...zn, τ}]»",
                 OF "=I"(1)[OF A[THEN "&E"(1)]], OF A[THEN "&E"(2)]]
    by blast
qed

AOT_theorem "sub-des-lam:2":                                                    (186.2)
  ‹ιx φ{x} = ιx ψ{x} → χ{ιx φ{x}} = χ{ιx ψ{x}}› for χ :: ‹κ ⇒ o›
  using "rule=E"[where φ="λ τ . «χ{ιx φ{x}} = χ{τ}»",
                   OF "=I"(1)[OF "log-prop-prop:2"]] "→I" by blast

AOT_theorem "prop-equiv": ‹F = G ≡ ∀x (x[F] ≡ x[G])›                            (187)
proof(rule "≡I"; rule "→I")
  AOT_assume ‹F = G›
  AOT_thus ‹∀x (x[F] ≡ x[G])›
    by (rule "rule=E"[rotated]) (fact "oth-class-taut:3:a"[THEN GEN])
next
  AOT_assume ‹∀x (x[F] ≡ x[G])›
  AOT_hence ‹x[F] ≡ x[G]› for x
    using "∀E" by blast
  AOT_hence ‹□(x[F] ≡ x[G])› for x
    using "en-eq:6[1]"[THEN "≡E"(1)] by blast
  AOT_hence ‹∀x □(x[F] ≡ x[G])›
    by (rule GEN)
  AOT_hence ‹□∀x (x[F] ≡ x[G])›
    using BF[THEN "→E"] by fast
```

```
  AOT_thus "F = G"
    using "p-identity-thm2:1"[THEN "≡E"(2)] by blast
qed

AOT_theorem "relations:1":                                                   (189.1)
  assumes ‹INSTANCE_OF_CQT_2(φ)›
  shows ‹∃F □∀x₁...∀xₙ ([F]x₁...xₙ ≡ φ{x₁...xₙ})›
  apply (rule "∃I"(1)[where τ="«[λx₁...xₙ φ{x₁...xₙ}]»"])
  using "cqt:2[lambda]"[OF assms, axiom_inst]
        "beta-C-cor:2"[THEN "→E", THEN RN] by blast+

AOT_theorem "relations:2":                                                   (189.2)
  assumes ‹INSTANCE_OF_CQT_2(φ)›
  shows ‹∃F □∀x ([F]x ≡ φ{x})›
  using "relations:1" assms by blast

AOT_theorem "block-paradox:1": ‹¬[λx ∃G (x[G] & ¬[G]x)]↓›                    (190.1)
proof(rule "raa-cor:2")
  let ?K="«[λx ∃G (x[G] & ¬[G]x)]»"
  AOT_assume A: ‹«?K»↓›
  AOT_have ‹∃x (A!x & ∀F (x[F] ≡ F = «?K»))›
    using "A-objects"[axiom_inst] by fast
  then AOT_obtain a where ξ: ‹A!a & ∀F (a[F] ≡ F = «?K»)›
    using "∃E"[rotated] by blast
  AOT_show ‹p & ¬p› for p
  proof (rule "∨E"(1)[OF "exc-mid"]; rule "→I")
    AOT_assume B: ‹[«?K»]a›
    AOT_hence ‹∃G (a[G] & ¬[G]a)›
      using "β→C" A by blast
    then AOT_obtain P where ‹a[P] & ¬[P]a›
      using "∃E"[rotated] by blast
    moreover AOT_have ‹P = [«?K»]›
      using ξ[THEN "&E"(2), THEN "∀E"(2), THEN "≡E"(1)]
            calculation[THEN "&E"(1)] by blast
    ultimately AOT_have ‹¬[«?K»]a›
      using "rule=E" "&E"(2) by fast
    AOT_thus ‹p & ¬p›
      using B RAA by blast
  next
    AOT_assume B: ‹¬[«?K»]a›
    AOT_hence ‹¬∃G (a[G] & ¬[G]a)›
      using "β←C" "cqt:2[const_var]"[of a, axiom_inst] A by blast
    AOT_hence C: ‹∀G ¬(a[G] & ¬[G]a)›
      using "cqt-further:4"[THEN "→E"] by blast
    AOT_have ‹∀G (a[G] → [G]a)›
      by (AOT_subst ‹a[G] → [G]a› ‹¬(a[G] & ¬[G]a)› for: G)
         (auto simp: "oth-class-taut:1:a" C)
    AOT_hence ‹a[«?K»] → [«?K»]a›
      using "∀E" A by blast
    moreover AOT_have ‹a[«?K»]›
      using ξ[THEN "&E"(2), THEN "∀E"(1), OF A, THEN "≡E"(2)]
      using "=I"(1)[OF A] by blast
    ultimately AOT_show ‹p & ¬p›
      using B "→E" RAA by blast
  qed
qed

AOT_theorem "block-paradox:2": ‹¬∃F ∀x([F]x ≡ ∃G(x[G] & ¬[G]x))›              (190.2)
proof(rule RAA(2))
  AOT_assume ‹∃F ∀x ([F]x ≡ ∃G (x[G] & ¬[G]x))›
  then AOT_obtain F where F_prop: ‹∀x ([F]x ≡ ∃G (x[G] & ¬[G]x))›
    using "∃E"[rotated] by blast
  AOT_have ‹∃x (A!x & ∀G (x[G] ≡ G = F))›
```

```
    using "A-objects"[axiom_inst] by fast
  then AOT_obtain a where ξ: ‹A!a & ∀G (a[G] ≡ G = F)›
    using "∃E"[rotated] by blast
  AOT_show ‹¬∃F ∀x([F]x ≡ ∃G(x[G] & ¬[G]x))›
  proof (rule "∨E"(1)[OF "exc-mid"]; rule "→I")
    AOT_assume B: ‹[F]a›
    AOT_hence ‹∃G (a[G] & ¬[G]a)›
      using F_prop[THEN "∀E"(2), THEN "≡E"(1)] by blast
    then AOT_obtain P where ‹a[P] & ¬[P]a›
      using "∃E"[rotated] by blast
    moreover AOT_have ‹P = F›
      using ξ[THEN "&E"(2), THEN "∀E"(2), THEN "≡E"(1)]
            calculation[THEN "&E"(1)] by blast
    ultimately AOT_have ‹¬[F]a›
      using "rule=E" "&E"(2) by fast
    AOT_thus ‹¬∃F ∀x([F]x ≡ ∃G(x[G] & ¬[G]x))›
      using B RAA by blast
  next
    AOT_assume B: ‹¬[F]a›
    AOT_hence ‹¬∃G (a[G] & ¬[G]a)›
      using "oth-class-taut:4:b"[THEN "≡E"(1),
              OF F_prop[THEN "∀E"(2)[of _ _ a]], THEN "≡E"(1)]
      by simp
    AOT_hence C: ‹∀G ¬(a[G] & ¬[G]a)›
      using "cqt-further:4"[THEN "→E"] by blast
    AOT_have ‹∀G (a[G] → [G]a)›
      by (AOT_subst ‹a[G] → [G]a› ‹¬(a[G] & ¬[G]a)› for: G)
         (auto simp: "oth-class-taut:1:a" C)
    AOT_hence ‹a[F] → [F]a›
      using "∀E" by blast
    moreover AOT_have ‹a[F]›
      using ξ[THEN "&E"(2), THEN "∀E"(2), of F, THEN "≡E"(2)]
      using "=I"(2) by blast
    ultimately AOT_show ‹¬∃F ∀x([F]x ≡ ∃G(x[G] & ¬[G]x))›
      using B "→E" RAA by blast
  qed
qed(simp)

AOT_theorem "block-paradox:3": ‹¬∀y [λz z = y]↓›                    (190.3)
proof(rule RAA(2))
  AOT_assume ϑ: ‹∀y [λz z = y]↓›
  AOT_have ‹∃x (A!x & ∀F (x[F] ≡ ∃y(F = [λz z = y] & ¬y[F])))›
    using "A-objects"[axiom_inst] by force
  then AOT_obtain a where
    a_prop: ‹A!a & ∀F (a[F] ≡ ∃y (F = [λz z = y] & ¬y[F]))›
    using "∃E"[rotated] by blast
  AOT_have ζ: ‹a[λz z = a] ≡ ∃y ([λz z = a] = [λz z = y] & ¬y[λz z = a])›
    using ϑ[THEN "∀E"(2)] a_prop[THEN "&E"(2), THEN "∀E"(1)] by blast
  AOT_show ‹¬∀y [λz z = y]↓›
  proof (rule "∨E"(1)[OF "exc-mid"]; rule "→I")
    AOT_assume A: ‹a[λz z = a]›
    AOT_hence ‹∃y ([λz z = a] = [λz z = y] & ¬y[λz z = a])›
      using ζ[THEN "≡E"(1)] by blast
    then AOT_obtain b where b_prop: ‹[λz z = a] = [λz z = b] & ¬b[λz z = a]›
      using "∃E"[rotated] by blast
    moreover AOT_have ‹a = a› by (rule "=I")
    moreover AOT_have ‹[λz z = a]↓› using ϑ "∀E" by blast
    moreover AOT_have ‹a↓› using "cqt:2[const_var]"[axiom_inst] .
    ultimately AOT_have ‹[λz z = a]a› using "β←C" by blast
    AOT_hence ‹[λz z = b]a› using "rule=E" b_prop[THEN "&E"(1)] by fast
    AOT_hence ‹a = b› using "β→C" by blast
    AOT_hence ‹b[λz z = a]› using A "rule=E" by fast
    AOT_thus ‹¬∀y [λz z = y]↓› using b_prop[THEN "&E"(2)] RAA by blast
```

```
  next
    AOT_assume A: ‹¬a[λz z = a]›
    AOT_hence ‹¬∃y ([λz z = a] = [λz z = y] & ¬y[λz z = a])›
      using ζ "oth-class-taut:4:b"[THEN "≡E"(1), THEN "≡E"(1)] by blast
    AOT_hence ‹∀y ¬([λz z = a] = [λz z = y] & ¬y[λz z = a])›
      using "cqt-further:4"[THEN "→E"] by blast
    AOT_hence ‹¬([λz z = a] = [λz z = a] & ¬a[λz z = a])›
      using "∀E" by blast
    AOT_hence ‹[λz z = a] = [λz z = a] → a[λz z = a]›
      by (metis "&I" "deduction-theorem" "raa-cor:4")
    AOT_hence ‹a[λz z = a]› using "=I"(1) ϑ[THEN "∀E"(2)] "→E" by blast
    AOT_thus ‹¬∀y [λz z = y]↓› using A RAA by blast
  qed
qed(simp)

AOT_theorem "block-paradox:4": ‹¬∀y ∃F ∀x([F]x ≡ x = y)›                    (190.4)
proof(rule RAA(2))
  AOT_assume ϑ: ‹∀y ∃F ∀x([F]x ≡ x = y)›
  AOT_have ‹∃x (A!x & ∀F (x[F] ≡ ∃z (∀y([F]y ≡ y = z) & ¬z[F])))›
    using "A-objects"[axiom_inst] by force
  then AOT_obtain a where
    a_prop: ‹A!a & ∀F (a[F] ≡ ∃z (∀y([F]y ≡ y = z) & ¬z[F]))›
    using "∃E"[rotated] by blast
  AOT_obtain F where F_prop: ‹∀x ([F]x ≡ x = a)›
    using ϑ[THEN "∀E"(2)] "∃E"[rotated] by blast
  AOT_have ζ: ‹a[F] ≡ ∃z (∀y ([F]y ≡ y = z) & ¬z[F])›
    using a_prop[THEN "&E"(2), THEN "∀E"(2)] by blast
  AOT_show ‹¬∀y ∃F ∀x([F]x ≡ x = y)›
  proof (rule "∨E"(1)[OF "exc-mid"]; rule "→I")
    AOT_assume A: ‹a[F]›
    AOT_hence ‹∃z (∀y ([F]y ≡ y = z) & ¬z[F])›
      using ζ[THEN "≡E"(1)] by blast
    then AOT_obtain b where b_prop: ‹∀y ([F]y ≡ y = b) & ¬b[F]›
      using "∃E"[rotated] by blast
    moreover AOT_have ‹[F]a›
      using F_prop[THEN "∀E"(2), THEN "≡E"(2)] "=I"(2) by blast
    ultimately AOT_have ‹a = b›
      using "∀E"(2) "≡E"(1) "&E" by fast
    AOT_hence ‹a = b›
      using "β→C" by blast
    AOT_hence ‹b[F]›
      using A "rule=E" by fast
    AOT_thus ‹¬∀y ∃F ∀x([F]x ≡ x = y)›
      using b_prop[THEN "&E"(2)] RAA by blast
  next
    AOT_assume A: ‹¬a[F]›
    AOT_hence ‹¬∃z (∀y ([F]y ≡ y = z) & ¬z[F])›
      using ζ "oth-class-taut:4:b"[THEN "≡E"(1), THEN "≡E"(1)] by blast
    AOT_hence ‹∀z ¬(∀y ([F]y ≡ y = z) & ¬z[F])›
      using "cqt-further:4"[THEN "→E"] by blast
    AOT_hence ‹¬(∀y ([F]y ≡ y = a) & ¬a[F])›
      using "∀E" by blast
    AOT_hence ‹∀y ([F]y ≡ y = a) → a[F]›
      by (metis "&I" "deduction-theorem" "raa-cor:4")
    AOT_hence ‹a[F]› using F_prop "→E" by blast
    AOT_thus ‹¬∀y ∃F ∀x([F]x ≡ x = y)›
      using A RAA by blast
  qed
qed(simp)

AOT_theorem "block-paradox:5": ‹¬∃F∀x∀y([F]xy ≡ y = x)›                    (190.5)
proof(rule "raa-cor:2")
  AOT_assume ‹∃F∀x∀y([F]xy ≡ y = x)›
```

```
  then AOT_obtain F where F_prop: ‹∀x∀y([F]xy ≡ y = x)›
    using "∃E"[rotated] by blast
  {
    fix x
    AOT_have 1: ‹∀y([F]xy ≡ y = x)›
      using F_prop "∀E" by blast
    AOT_have 2: ‹[λz [F]xz]↓› by "cqt:2"
    moreover AOT_have ‹∀y([λz [F]xz]y ≡ y = x)›
    proof(rule "∀I")
      fix y
      AOT_have ‹[λz [F]xz]y ≡ [F]xy›
        using "beta-C-meta"[THEN "→E"] 2 by fast
      also AOT_have ‹... ≡ y = x›
        using 1 "∀E" by fast
      finally AOT_show ‹[λz [F]xz]y ≡ y = x›.
    qed
    ultimately AOT_have ‹∃F∀y([F]y ≡ y = x)›
      using "∃I" by fast
  }
  AOT_hence ‹∀x∃F∀y([F]y ≡ y = x)›
    by (rule GEN)
  AOT_thus ‹∀x∃F∀y([F]y ≡ y = x) & ¬∀x∃F∀y([F]y ≡ y = x)›
    using "&I" "block-paradox:4" by blast
qed

AOT_act_theorem "block-paradox2:1":                                       (191.1)
  ‹∀x [G]x → ¬[λx [G]ιy (y = x & ∃H (x[H] & ¬[H]x))]↓›
proof(rule "→I"; rule "raa-cor:2")
  AOT_assume antecedant: ‹∀x [G]x›
  AOT_have Lemma: ‹∀x ([G]ιy(y = x & ∃H (x[H] & ¬[H]x)) ≡ ∃H (x[H] & ¬[H]x))›
  proof(rule GEN)
    fix x
    AOT_have A: ‹[G]ιy (y = x & ∃H (x[H] & ¬[H]x)) ≡
                 ∃!y (y = x & ∃H (x[H] & ¬[H]x))›
    proof(rule "≡I"; rule "→I")
      AOT_assume ‹[G]ιy (y = x & ∃H (x[H] & ¬[H]x))›
      AOT_hence ‹ιy (y = x & ∃H (x[H] & ¬[H]x))↓›
        using "cqt:5:a"[axiom_inst, THEN "→E", THEN "&E"(2)] by blast
      AOT_thus ‹∃!y (y = x & ∃H (x[H] & ¬[H]x))›
        using "!-exists:1"[THEN "≡E"(1)] by blast
    next
      AOT_assume A: ‹∃!y (y = x & ∃H (x[H] & ¬[H]x))›
      AOT_obtain a where a_1: ‹a = x & ∃H (x[H] & ¬[H]x)›
                   and a_2: ‹∀z (z = x & ∃H (x[H] & ¬[H]x) → z = a)›
        using "uniqueness:1"[THEN "≡dfE", OF A] "&E" "∃E"[rotated] by blast
      AOT_have a_3: ‹[G]a›
        using antecedant "∀E" by blast
      AOT_show ‹[G]ιy (y = x & ∃H (x[H] & ¬[H]x))›
        apply (rule "russell-axiom[exe,1].russell-axiom"[THEN "≡E"(2)])
        apply (rule "∃I"(2))
        using a_1 a_2 a_3 "&I" by blast
    qed
    also AOT_have B: ‹... ≡ ∃H (x[H] & ¬[H]x)›
    proof (rule "≡I"; rule "→I")
      AOT_assume A: ‹∃!y (y = x & ∃H (x[H] & ¬[H]x))›
      AOT_obtain a where ‹a = x & ∃H (x[H] & ¬[H]x)›
        using "uniqueness:1"[THEN "≡dfE", OF A] "&E" "∃E"[rotated] by blast
      AOT_thus ‹∃H (x[H] & ¬[H]x)› using "&E" by blast
    next
      AOT_assume ‹∃H (x[H] & ¬[H]x)›
      AOT_hence ‹x = x & ∃H (x[H] & ¬[H]x)›
        using "id-eq:1" "&I" by blast
      moreover AOT_have ‹∀z (z = x & ∃H (x[H] & ¬[H]x) → z = x)›
```

```
        by (simp add: "Conjunction Simplification"(1) "universal-cor")
      ultimately AOT_show ‹∃!y (y = x & ∃H (x[H] & ¬[H]x))›
        using "uniqueness:1"[THEN "≡dfI"] "&I" "∃I"(2) by fast
    qed
    finally AOT_show ‹([G]ιy(y = x & ∃H (x[H] & ¬[H]x)) ≡ ∃H (x[H] & ¬[H]x))› .
  qed

  AOT_assume A: ‹[λx [G]ιy (y = x & ∃H (x[H] & ¬[H]x))]↓›
  AOT_have ϑ: ‹∀x ([λx [G]ιy (y = x & ∃H (x[H] & ¬[H]x))]x ≡
                  [G]ιy(y = x & ∃H (x[H] & ¬[H]x)))›
    using "beta-C-meta"[THEN "→E", OF A] "∀I" by fast
  AOT_have ‹∀x ([λx [G]ιy (y = x & ∃H (x[H] & ¬[H]x))]x ≡ ∃H (x[H] & ¬[H]x))›
    using ϑ Lemma "cqt-basic:10"[THEN "→E"] "&I" by fast
  AOT_hence ‹∃F ∀x ([F]x ≡ ∃H (x[H] & ¬[H]x))›
    using "∃I"(1) A by fast
  AOT_thus ‹(∃F ∀x ([F]x ≡ ∃H (x[H] & ¬[H]x))) &
            (¬∃F ∀x ([F]x ≡ ∃H (x[H] & ¬[H]x)))›
    using "block-paradox:2" "&I" by blast
qed

text‹Note: Strengthens the above to a modally-strict theorem.
           Not explicitly part of PLM.›
AOT_theorem "block-paradox2:1[strict]":                                    (191.1)
  ‹∀x 𝒜[G]x → ¬[λx [G]ιy (y = x & ∃H (x[H] & ¬[H]x))]↓›
proof(rule "→I"; rule "raa-cor:2")
  AOT_assume antecedant: ‹∀x 𝒜[G]x›
  AOT_have Lemma: ‹𝒜∀x ([G]ιy(y = x & ∃H (x[H] & ¬[H]x)) ≡ ∃H (x[H] & ¬[H]x))›
  proof(safe intro!: GEN "Act-Basic:5"[THEN "≡E"(2)]
                     "logic-actual-nec:3"[axiom_inst, THEN "≡E"(2)])
    fix x
    AOT_have A: ‹𝒜[G]ιy (y = x & ∃H (x[H] & ¬[H]x)) ≡
                 ∃!y 𝒜(y = x & ∃H (x[H] & ¬[H]x))›
    proof(rule "≡I"; rule "→I")
      AOT_assume ‹𝒜[G]ιy (y = x & ∃H (x[H] & ¬[H]x))›
      moreover AOT_have ‹□([G]ιy (y = x & ∃H (x[H] & ¬[H]x)) →
                         □ιy (y = x & ∃H (x[H] & ¬[H]x))↓)›
      proof(rule RN; rule "→I")
        AOT_modally_strict {
          AOT_assume ‹[G]ιy (y = x & ∃H (x[H] & ¬[H]x))›
          AOT_hence ‹ιy (y = x & ∃H (x[H] & ¬[H]x))↓›
            using "cqt:5:a"[axiom_inst, THEN "→E", THEN "&E"(2)] by blast
          AOT_thus ‹□ιy (y = x & ∃H (x[H] & ¬[H]x))↓›
            using "exist-nec"[THEN "→E"] by blast
        }
      qed
      ultimately AOT_have ‹𝒜□ιy (y = x & ∃H (x[H] & ¬[H]x))↓›
        using "act-cond"[THEN "→E", THEN "→E"] "nec-imp-act"[THEN "→E"] by blast
      AOT_hence ‹ιy (y = x & ∃H (x[H] & ¬[H]x))↓›
        using "Act-Sub:3" "B◇" "vdash-properties:10" by blast
      AOT_thus ‹∃!y 𝒜(y = x & ∃H (x[H] & ¬[H]x))›
        using "actual-desc:1"[THEN "≡E"(1)] by blast
    next
      AOT_assume A: ‹∃!y 𝒜(y = x & ∃H (x[H] & ¬[H]x))›
      AOT_obtain a where a_1: ‹𝒜(a = x & ∃H (x[H] & ¬[H]x))›
                     and a_2: ‹∀z (𝒜(z = x & ∃H (x[H] & ¬[H]x)) → z = a)›
        using "uniqueness:1"[THEN "≡dfE", OF A] "&E" "∃E"[rotated] by blast
      AOT_have a_3: ‹𝒜[G]a›
        using antecedant "∀E" by blast
      moreover AOT_have ‹a = ιy(y = x & ∃H (x[H] & ¬[H]x))›
        using "nec-hintikka-scheme"[THEN "≡E"(2), OF "&I"] a_1 a_2 by auto
      ultimately AOT_show ‹𝒜[G]ιy (y = x & ∃H (x[H] & ¬[H]x))›
        using "rule=E" by fast
    qed
```

```
    also AOT_have B: ‹... ≡ 𝒜∃H (x[H] & ¬[H]x)›
    proof (rule "≡I"; rule "→I")
      AOT_assume A: ‹∃!y 𝒜(y = x & ∃H (x[H] & ¬[H]x))›
      AOT_obtain a where ‹𝒜(a = x & ∃H (x[H] & ¬[H]x))›
        using "uniqueness:1"[THEN "≡dfE", OF A] "&E" "∃E"[rotated] by blast
      AOT_thus ‹𝒜∃H (x[H] & ¬[H]x)›
        using "Act-Basic:2"[THEN "≡E"(1), THEN "&E"(2)] by blast
    next
      AOT_assume ‹𝒜∃H (x[H] & ¬[H]x)›
      AOT_hence ‹𝒜x = x & 𝒜∃H (x[H] & ¬[H]x)›
        using "id-eq:1" "&I" "RA[2]" by blast
      AOT_hence ‹𝒜(x = x & ∃H (x[H] & ¬[H]x))›
        using "act-conj-act:3" "Act-Basic:2" "≡E" by blast
      moreover AOT_have ‹∀z (𝒜(z = x & ∃H (x[H] & ¬[H]x)) → z = x)›
      proof(safe intro!: GEN "→I")
        fix z
        AOT_assume ‹𝒜(z = x & ∃H (x[H] & ¬[H]x))›
        AOT_hence ‹𝒜(z = x)›
          using "Act-Basic:2"[THEN "≡E"(1), THEN "&E"(1)] by blast
        AOT_thus ‹z = x›
           by (metis "id-act:1" "intro-elim:3:b")
      qed
      ultimately AOT_show ‹∃!y 𝒜(y = x & ∃H (x[H] & ¬[H]x))›
        using "uniqueness:1"[THEN "≡dfI"] "&I" "∃I"(2) by fast
    qed
    finally AOT_show ‹(𝒜[G]ιy(y = x & ∃H (x[H] & ¬[H]x)) ≡ 𝒜∃H (x[H] & ¬[H]x))›.
  qed

  AOT_assume A: ‹[λx [G]ιy (y = x & ∃H (x[H] & ¬[H]x))]↓›
  AOT_hence ‹𝒜[λx [G]ιy (y = x & ∃H (x[H] & ¬[H]x))]↓›
    using "exist-nec" "→E" "nec-imp-act"[THEN "→E"] by blast
  AOT_hence ‹𝒜([λx [G]ιy (y = x & ∃H (x[H] & ¬[H]x))]↓ &
                ∀x ([G]ιy(y = x & ∃H (x[H] & ¬[H]x)) ≡ ∃H (x[H] & ¬[H]x)))›
    using Lemma "Act-Basic:2"[THEN "≡E"(2)] "&I" by blast
  moreover AOT_have ‹𝒜([λx [G]ιy (y = x & ∃H (x[H] & ¬[H]x))]↓ &
                ∀x ([G]ιy(y = x & ∃H (x[H] & ¬[H]x)) ≡ ∃H (x[H] & ¬[H]x)))
      → 𝒜∃p (p & ¬p)›
  proof (rule "logic-actual-nec:2"[axiom_inst, THEN "≡E"(1)];
         rule "RA[2]"; rule "→I")
    AOT_modally_strict {
      AOT_assume 0: ‹[λx [G]ιy (y = x & ∃H (x[H] & ¬[H]x))]↓ &
                ∀x ([G]ιy(y = x & ∃H (x[H] & ¬[H]x)) ≡ ∃H (x[H] & ¬[H]x))›
      AOT_have ‹∃F ∀x ([F]x ≡ ∃G (x[G] & ¬[G]x))›
      proof(rule "∃I"(1))
        AOT_show ‹∀x ([λx [G]ιy (y = x & ∃H (x[H] & ¬[H]x))]x ≡ ∃H (x[H] & ¬[H]x))›
        proof(safe intro!: GEN "≡I" "→I" "β←C" dest!: "β→C")
          fix x
          AOT_assume ‹[G]ιy(y = x & ∃H (x[H] & ¬[H]x))›
          AOT_thus ‹∃H (x[H] & ¬[H]x)›
            using 0 "&E" "∀E"(2) "≡E"(1) by blast
        next
          fix x
          AOT_assume ‹∃H (x[H] & ¬[H]x)›
          AOT_thus ‹[G]ιy(y = x & ∃H (x[H] & ¬[H]x))›
            using 0 "&E" "∀E"(2) "≡E"(2) by blast
        qed(auto intro!: 0[THEN "&E"(1)] "cqt:2")
      next
        AOT_show ‹[λx [G]ιy (y = x & ∃H (x[H] & ¬[H]x))]↓›
          using 0 "&E"(1) by blast
      qed
      AOT_thus ‹∃p (p & ¬p)›
        using "block-paradox:2" "reductio-aa:1" by blast
    }
```

```
  qed
  ultimately AOT_have ‹𝒜∃p (p & ¬p)›
    using "→E" by blast
  AOT_hence ‹∃p 𝒜(p & ¬p)›
    by (metis "Act-Basic:10" "intro-elim:3:a")
  then AOT_obtain p where ‹𝒜(p & ¬p)›
    using "∃E"[rotated] by blast
  moreover AOT_have ‹¬𝒜(p & ¬p)›
    using "non-contradiction"[THEN "RA[2]"]
    by (meson "Act-Sub:1" "¬¬I" "intro-elim:3:d")
  ultimately AOT_show ‹p & ¬p› for p
    by (metis "raa-cor:3")
qed

AOT_act_theorem "block-paradox2:2":                                    (191.2)
  ‹∃G ¬[λx [G]ιy (y = x & ∃H (x[H] & ¬[H]x))]↓›
proof(rule "∃I"(1))
  AOT_have 0: ‹[λx ∀p (p →p)]↓›
    by "cqt:2[lambda]"
  moreover AOT_have ‹∀x [λx ∀p (p →p)]x›
    apply (rule GEN)
    apply (rule "beta-C-cor:2"[THEN "→E", OF 0, THEN "∀E"(2), THEN "≡E"(2)])
    using "if-p-then-p" GEN by fast
  moreover AOT_have ‹∀G (∀x [G]x → ¬[λx [G]ιy (y = x & ∃H (x[H] & ¬[H]x))]↓)›
      using "block-paradox2:1" "∀I" by fast
  ultimately AOT_show ‹¬[λx [λx ∀p (p →p)]ιy (y = x & ∃H (x[H] & ¬[H]x))]↓›
    using "∀E"(1) "→E" by blast
qed("cqt:2[lambda]")

AOT_theorem propositions: ‹∃p □(p ≡ φ)›                                 (192)
proof(rule "∃I"(1))
  AOT_show ‹□(φ ≡ φ)›
    by (simp add: RN "oth-class-taut:3:a")
next
  AOT_show ‹φ↓›
    by (simp add: "log-prop-prop:2")
qed

AOT_theorem "pos-not-equiv-ne:1":                                      (193.1)
  ‹(◇¬∀x₁...∀xₙ ([F]x₁...xₙ ≡ [G]x₁...xₙ)) → F ≠ G›
proof (rule "→I")
  AOT_assume ‹◇¬∀x₁...∀xₙ ([F]x₁...xₙ ≡ [G]x₁...xₙ)›
  AOT_hence ‹¬□∀x₁...∀xₙ ([F]x₁...xₙ ≡ [G]x₁...xₙ)›
    using "KBasic:11"[THEN "≡E"(2)] by blast
  AOT_hence ‹¬(F = G)›
    using "id-rel-nec-equiv:1" "modus-tollens:1" by blast
  AOT_thus ‹F ≠ G›
    using "=-infix"[THEN "≡dfI"] by blast
qed

AOT_theorem "pos-not-equiv-ne:2": ‹(◇¬(φ{F} ≡ φ{G})) → F ≠ G›            (193.2)
proof (rule "→I")
  AOT_modally_strict {
    AOT_have ‹¬(φ{F} ≡ φ{G}) → ¬(F = G)›
    proof (rule "→I"; rule "raa-cor:2")
      AOT_assume 1: ‹F = G›
      AOT_hence ‹φ{F} → φ{G}›
        using "l-identity"[axiom_inst, THEN "→E"] by blast
      moreover {
        AOT_have ‹G = F›
          using 1 id_sym by blast
        AOT_hence ‹φ{G} → φ{F}›
          using "l-identity"[axiom_inst, THEN "→E"] by blast
```

```
      }
      ultimately AOT_have ‹φ{F} ≡ φ{G}›
        using "≡I" by blast
      moreover AOT_assume ‹¬(φ{F} ≡ φ{G})›
      ultimately AOT_show ‹(φ{F} ≡ φ{G}) & ¬(φ{F} ≡ φ{G})›
        using "&I" by blast
    qed
  }
  AOT_hence ‹◇¬(φ{F} ≡ φ{G}) → ◇¬(F = G)›
    using "RM:2[prem]" by blast
  moreover AOT_assume ‹◇¬(φ{F} ≡ φ{G})›
  ultimately AOT_have 0: ‹◇¬(F = G)› using "→E" by blast
  AOT_have ‹◇(F ≠ G)›
    by (AOT_subst ‹F ≠ G› ‹¬(F = G)›)
       (auto simp: "=-infix" "≡Df" 0)
  AOT_thus ‹F ≠ G›
    using "id-nec2:3"[THEN "→E"] by blast
qed

AOT_theorem "pos-not-equiv-ne:2[zero]": ‹(◇¬(φ{p} ≡ φ{q})) → p ≠ q›          (193.2)
proof (rule "→I")
  AOT_modally_strict {
    AOT_have ‹¬(φ{p} ≡ φ{q}) → ¬(p = q)›
    proof (rule "→I"; rule "raa-cor:2")
      AOT_assume 1: ‹p = q›
      AOT_hence ‹φ{p} → φ{q}›
        using "l-identity"[axiom_inst, THEN "→E"] by blast
      moreover {
        AOT_have ‹q = p›
          using 1 id_sym by blast
        AOT_hence ‹φ{q} → φ{p}›
          using "l-identity"[axiom_inst, THEN "→E"] by blast
      }
      ultimately AOT_have ‹φ{p} ≡ φ{q}›
        using "≡I" by blast
      moreover AOT_assume ‹¬(φ{p} ≡ φ{q})›
      ultimately AOT_show ‹(φ{p} ≡ φ{q}) & ¬(φ{p} ≡ φ{q})›
        using "&I" by blast
    qed
  }
  AOT_hence ‹◇¬(φ{p} ≡ φ{q}) → ◇¬(p = q)›
    using "RM:2[prem]" by blast
  moreover AOT_assume ‹◇¬(φ{p} ≡ φ{q})›
  ultimately AOT_have 0: ‹◇¬(p = q)› using "→E" by blast
  AOT_have ‹◇(p ≠ q)›
    by (AOT_subst ‹p ≠ q› ‹¬(p = q)›)
       (auto simp: 0 "=-infix" "≡Df")
  AOT_thus ‹p ≠ q›
    using "id-nec2:3"[THEN "→E"] by blast
qed

AOT_theorem "pos-not-equiv-ne:3":                                         (193.3)
  ‹(¬∀x1...∀xn ([F]x1...xn ≡ [G]x1...xn)) → F ≠ G›
  using "→I" "pos-not-equiv-ne:1"[THEN "→E"] "T◇"[THEN "→E"] by blast

AOT_theorem "pos-not-equiv-ne:4": ‹(¬(φ{F} ≡ φ{G})) → F ≠ G›            (193.4)
  using "→I" "pos-not-equiv-ne:2"[THEN "→E"] "T◇"[THEN "→E"] by blast

AOT_theorem "pos-not-equiv-ne:4[zero]": ‹(¬(φ{p} ≡ φ{q})) → p ≠ q›       (193.4)
  using "→I" "pos-not-equiv-ne:2[zero]"[THEN "→E"]
      "T◇"[THEN "→E"] by blast

AOT_define relation_negation :: "Π ⇒ Π" ("_¯")
```

```
  "df-relation-negation": "[F]⁻ =df [λx1...xn ¬[F]x1...xn]"                    (194)

nonterminal φneg
syntax "" :: "φneg ⇒ τ" ("_")
syntax "" :: "φneg ⇒ φ" ("'(_')")

AOT_define relation_negation_0 :: ‹φ ⇒ φneg› ("'(_')⁻")
  "df-relation-negation[zero]": "(p)⁻ =df [λ ¬p]"                             (194)

AOT_theorem "rel-neg-T:1": ‹[λx1...xn ¬[Π]x1...xn]↓›                          (195.1)
  by "cqt:2[lambda]"

AOT_theorem "rel-neg-T:1[zero]": ‹[λ ¬φ]↓›                                     (195.1)
  using "cqt:2[lambda0]"[axiom_inst] by blast

AOT_theorem "rel-neg-T:2": ‹[Π]⁻ = [λx1...xn ¬[Π]x1...xn]›                    (195.2)
  using "=I"(1)[OF "rel-neg-T:1"]
  by (rule "=dfI"(1)[OF "df-relation-negation", OF "rel-neg-T:1"])

AOT_theorem "rel-neg-T:2[zero]": ‹(φ)⁻ = [λ ¬φ]›                              (195.2)
  using "=I"(1)[OF "rel-neg-T:1[zero]"]
  by (rule "=dfI"(1)[OF "df-relation-negation[zero]", OF "rel-neg-T:1[zero]"])

AOT_theorem "rel-neg-T:3": ‹[Π]⁻↓›                                              (195.3)
  using "=dfI"(1)[OF "df-relation-negation", OF "rel-neg-T:1"]
        "rel-neg-T:1" by blast

AOT_theorem "rel-neg-T:3[zero]": ‹(φ)⁻↓›                                        (195.3)
  using "log-prop-prop:2" by blast

AOT_theorem "thm-relation-negation:1": ‹[F]⁻x1...xn ≡ ¬[F]x1...xn›             (197.1)
proof -
  AOT_have ‹[F]⁻x1...xn ≡ [λx1...xn ¬[F]x1...xn]x1...xn›
    using "rule=E"[rotated, OF "rel-neg-T:2"]
          "rule=E"[rotated, OF "rel-neg-T:2"[THEN id_sym]]
          "→I" "≡I" by fast
  also AOT_have ‹... ≡ ¬[F]x1...xn›
    using "beta-C-meta"[THEN "→E", OF "rel-neg-T:1"] by fast
  finally show ?thesis.
qed

AOT_theorem "thm-relation-negation:2": ‹¬[F]⁻x1...xn ≡ [F]x1...xn›             (197.2)
  apply (AOT_subst ‹[F]x1...xn› ‹¬¬[F]x1...xn›)
   apply (simp add: "oth-class-taut:3:b")
  apply (rule "oth-class-taut:4:b"[THEN "≡E"(1)])
  using "thm-relation-negation:1".

AOT_theorem "thm-relation-negation:3": ‹((p)⁻) ≡ ¬p›                           (197.3)
proof -
  AOT_have ‹(p)⁻ = [λ ¬p]› using "rel-neg-T:2[zero]" by blast
  AOT_hence ‹((p)⁻) ≡ [λ ¬p]›
    using "df-relation-negation[zero]" "log-prop-prop:2"
          "oth-class-taut:3:a" "rule-id-df:2:a" by blast
  also AOT_have ‹[λ ¬p] ≡ ¬p›
    by (simp add: "propositions-lemma:2")
  finally show ?thesis.
qed

AOT_theorem "thm-relation-negation:4": ‹(¬((p)⁻)) ≡ p›                        (197.4)
  using "thm-relation-negation:3"[THEN "≡E"(1)]
        "thm-relation-negation:3"[THEN "≡E"(2)]
        "≡I" "→I" RAA by metis
```

```
AOT_theorem "thm-relation-negation:5": ‹[F] ≠ [F]^-›                      (197.5)
proof -
  AOT_have ‹¬([F] = [F]^-)›
  proof (rule RAA(2))
    AOT_show ‹[F]x1...xn → [F]x1...xn› for x1xn
      using "if-p-then-p".
  next
    AOT_assume ‹[F] = [F]^-›
    AOT_hence ‹[F]^- = [F]› using id_sym by blast
    AOT_hence ‹[F]x1...xn ≡ ¬[F]x1...xn› for x1xn
      using "rule=E" "thm-relation-negation:1" by fast
    AOT_thus ‹¬([F]x1...xn → [F]x1...xn)› for x1xn
      using "≡E" RAA by metis
  qed
  thus ?thesis
    using "≡_dfI" "=-infix" by blast
qed

AOT_theorem "thm-relation-negation:6": ‹p ≠ (p)^-›                        (197.6)
proof -
  AOT_have ‹¬(p = (p)^-)›
  proof (rule RAA(2))
    AOT_show ‹p → p›
      using "if-p-then-p".
  next
    AOT_assume ‹p = (p)^-›
    AOT_hence ‹(p)^- = p› using id_sym by blast
    AOT_hence ‹p ≡ ¬p›
      using "rule=E" "thm-relation-negation:3" by fast
    AOT_thus ‹¬(p → p)›
      using "≡E" RAA by metis
  qed
  thus ?thesis
    using "≡_dfI" "=-infix" by blast
qed

AOT_theorem "thm-relation-negation:7": ‹(p)^- = (¬p)›                      (197.7)
  apply (rule "df-relation-negation[zero]"[THEN "=_dfE"(1)])
  using "cqt:2[lambda0]"[axiom_inst] "rel-neg-T:2[zero]"
        "propositions-lemma:1" id_trans by blast+

AOT_theorem "thm-relation-negation:8": ‹p = q → (¬p) = (¬q)›              (197.8)
proof(rule "→I")
  AOT_assume ‹p = q›
  moreover AOT_have ‹(¬p)↓› using "log-prop-prop:2".
  moreover AOT_have ‹(¬p) = (¬p)› using calculation(2) "=I" by blast
  ultimately AOT_show ‹(¬p) = (¬q)›
    using "rule=E" by fast
qed

AOT_theorem "thm-relation-negation:9": ‹p = q → (p)^- = (q)^-›            (197.9)
proof(rule "→I")
  AOT_assume ‹p = q›
  AOT_hence ‹(¬p) = (¬q)› using "thm-relation-negation:8" "→E" by blast
  AOT_thus ‹(p)^- = (q)^-›
    using "thm-relation-negation:7" id_sym id_trans by metis
qed

AOT_define Necessary :: ‹Π ⇒ φ› ("Necessary'(_')")
  "contingent-properties:1":                                             (198.1)
  ‹Necessary([F]) ≡_df □∀x1...∀xn [F]x1...xn›

AOT_define Necessary0 :: ‹φ ⇒ φ› ("Necessary0'(_')")
```

```
  "contingent-properties:1[zero]":                                      (198.1)
  ‹Necessary0(p) ≡df □p›

AOT_define Impossible :: ‹Π ⇒ φ› ("Impossible'(_')")
  "contingent-properties:2":                                            (198.2)
  ‹Impossible([F]) ≡df F↓ & □∀x1...∀xn ¬[F]x1...xn›

AOT_define Impossible0 :: ‹φ ⇒ φ› ("Impossible0'(_')")
  "contingent-properties:2[zero]":                                      (198.2)
  ‹Impossible0(p) ≡df □¬p›

AOT_define NonContingent :: ‹Π ⇒ φ› ("NonContingent'(_')")
  "contingent-properties:3":                                            (198.3)
  ‹NonContingent([F]) ≡df Necessary([F]) ∨ Impossible([F])›

AOT_define NonContingent0 :: ‹φ ⇒ φ› ("NonContingent0'(_')")
  "contingent-properties:3[zero]":                                      (198.3)
  ‹NonContingent0(p) ≡df Necessary0(p) ∨ Impossible0(p)›

AOT_define Contingent :: ‹Π ⇒ φ› ("Contingent'(_')")
  "contingent-properties:4":                                            (198.4)
  ‹Contingent([F]) ≡df F↓ & ¬(Necessary([F]) ∨ Impossible([F]))›

AOT_define Contingent0 :: ‹φ ⇒ φ› ("Contingent0'(_')")
  "contingent-properties:4[zero]":                                      (198.4)
  ‹Contingent0(p) ≡df ¬(Necessary0(p) ∨ Impossible0(p))›


AOT_theorem "thm-cont-prop:1": ‹NonContingent([F]) ≡ NonContingent([F]⁻)›   (200.1)
proof (rule "≡I"; rule "→I")
  AOT_assume ‹NonContingent([F])›
  AOT_hence ‹Necessary([F]) ∨ Impossible([F])›
    using "≡dfE"[OF "contingent-properties:3"] by blast
  moreover {
    AOT_assume ‹Necessary([F])›
    AOT_hence ‹□(∀x1...∀xn [F]x1...xn)›
      using "≡dfE"[OF "contingent-properties:1"] by blast
    moreover AOT_modally_strict {
      AOT_assume ‹∀x1...∀xn [F]x1...xn›
      AOT_hence ‹[F]x1...xn› for x1xn using "∀E" by blast
      AOT_hence ‹¬[F]⁻x1...xn› for x1xn
        by (meson "≡E"(6) "oth-class-taut:3:a"
                  "thm-relation-negation:2" "≡E"(1))
      AOT_hence ‹∀x1...∀xn ¬[F]⁻x1...xn› using "∀I" by fast
    }
    ultimately AOT_have ‹□(∀x1...∀xn ¬[F]⁻x1...xn)›
      using "RN[prem]"[where Γ="{«∀x1...∀xn [F]x1...xn»}", simplified] by blast
    AOT_hence ‹Impossible([F]⁻)›
      using "≡Df"[OF "contingent-properties:2", THEN "≡S"(1),
                  OF "rel-neg-T:3", THEN "≡E"(2)]
      by blast
  }
  moreover {
    AOT_assume ‹Impossible([F])›
    AOT_hence ‹□(∀x1...∀xn ¬[F]x1...xn)›
      using "≡Df"[OF "contingent-properties:2", THEN "≡S"(1),
                  OF "cqt:2[const_var]"[axiom_inst], THEN "≡E"(1)]
      by blast
    moreover AOT_modally_strict {
      AOT_assume ‹∀x1...∀xn ¬[F]x1...xn›
      AOT_hence ‹¬[F]x1...xn› for x1xn using "∀E" by blast
      AOT_hence ‹[F]⁻x1...xn› for x1xn
        by (meson "≡E"(6) "oth-class-taut:3:a"
```

```
                    "thm-relation-negation:1" "≡E"(1))
      AOT_hence ‹∀x1...∀xn [F]^-x1...xn› using "∀I" by fast
    }
    ultimately AOT_have ‹□(∀x1...∀xn [F]^-x1...xn)›
      using "RN[prem]"[where Γ="{«∀x1...∀xn ¬[F]x1...xn»}"] by blast
    AOT_hence ‹Necessary([F]^-)›
      using "≡dfI"[OF "contingent-properties:1"] by blast
  }
  ultimately AOT_have ‹Necessary([F]^-) ∨ Impossible([F]^-)›
    using "∨E"(1) "∨I" "→I" by metis
  AOT_thus ‹NonContingent([F]^-)›
    using "≡dfI"[OF "contingent-properties:3"] by blast
next
  AOT_assume ‹NonContingent([F]^-)›
  AOT_hence ‹Necessary([F]^-) ∨ Impossible([F]^-)›
    using "≡dfE"[OF "contingent-properties:3"] by blast
  moreover {
    AOT_assume ‹Necessary([F]^-)›
    AOT_hence ‹□(∀x1...∀xn [F]^-x1...xn)›
      using "≡dfE"[OF "contingent-properties:1"] by blast
    moreover AOT_modally_strict {
      AOT_assume ‹∀x1...∀xn [F]^-x1...xn›
      AOT_hence ‹[F]^-x1...xn› for x1xn using "∀E" by blast
      AOT_hence ‹¬[F]x1...xn› for x1xn
        by (meson "≡E"(6) "oth-class-taut:3:a"
                  "thm-relation-negation:1" "≡E"(2))
      AOT_hence ‹∀x1...∀xn ¬[F]x1...xn› using "∀I" by fast
    }
    ultimately AOT_have ‹□∀x1...∀xn ¬[F]x1...xn›
      using "RN[prem]"[where Γ="{«∀x1...∀xn [F]^-x1...xn»}"] by blast
    AOT_hence ‹Impossible([F])›
      using "≡Df"[OF "contingent-properties:2", THEN "≡S"(1),
                  OF "cqt:2[const_var]"[axiom_inst], THEN "≡E"(2)]
      by blast
  }
  moreover {
    AOT_assume ‹Impossible([F]^-)›
    AOT_hence ‹□(∀x1...∀xn ¬[F]^-x1...xn)›
      using "≡Df"[OF "contingent-properties:2", THEN "≡S"(1),
                  OF "rel-neg-T:3", THEN "≡E"(1)]
      by blast
    moreover AOT_modally_strict {
      AOT_assume ‹∀x1...∀xn ¬[F]^-x1...xn›
      AOT_hence ‹¬[F]^-x1...xn› for x1xn using "∀E" by blast
      AOT_hence ‹[F]x1...xn› for x1xn
        using "thm-relation-negation:1"[THEN
                "oth-class-taut:4:b"[THEN "≡E"(1)], THEN "≡E"(1)]
              "useful-tautologies:1"[THEN "→E"] by blast
      AOT_hence ‹∀x1...∀xn [F]x1...xn› using "∀I" by fast
    }
    ultimately AOT_have ‹□(∀x1...∀xn [F]x1...xn)›
      using "RN[prem]"[where Γ="{«∀x1...∀xn ¬[F]^-x1...xn»}"] by blast
    AOT_hence ‹Necessary([F])›
      using "≡dfI"[OF "contingent-properties:1"] by blast
  }
  ultimately AOT_have ‹Necessary([F]) ∨ Impossible([F])›
    using "∨E"(1) "∨I" "→I" by metis
  AOT_thus ‹NonContingent([F])›
    using "≡dfI"[OF "contingent-properties:3"] by blast
qed

AOT_theorem "thm-cont-prop:2": ‹Contingent([F]) ≡ ◇∃x [F]x & ◇∃x ¬[F]x›      (200.2)
proof -
```

```
  AOT_have ‹Contingent([F]) ≡ ¬(Necessary([F]) ∨ Impossible([F]))›
    using "contingent-properties:4"[THEN "≡Df", THEN "≡S"(1),
                                    OF "cqt:2[const_var]"[axiom_inst]]
    by blast
  also AOT_have ‹... ≡ ¬Necessary([F]) & ¬Impossible([F])›
    using "oth-class-taut:5:d" by fastforce
  also AOT_have ‹... ≡ ¬Impossible([F]) & ¬Necessary([F])›
    by (simp add: "Commutativity of &")
  also AOT_have ‹... ≡ ◇∃x [F]x & ¬Necessary([F])›
  proof (rule "oth-class-taut:4:e"[THEN "→E"])
    AOT_have ‹¬Impossible([F]) ≡ ¬□¬ ∃x [F]x›
      apply (rule "oth-class-taut:4:b"[THEN "≡E"(1)])
      apply (AOT_subst ‹∃x [F]x› ‹¬ ∀x ¬[F]x›)
       apply (simp add: "conventions:4" "≡Df")
      apply (AOT_subst (reverse) ‹¬¬∀x ¬[F]x› ‹∀x ¬[F]x›)
       apply (simp add: "oth-class-taut:3:b")
      using "contingent-properties:2"[THEN "≡Df", THEN "≡S"(1),
                                      OF "cqt:2[const_var]"[axiom_inst]]
      by blast
    also AOT_have ‹... ≡ ◇∃x [F]x›
      using "conventions:5"[THEN "≡Df", symmetric] by blast
    finally AOT_show ‹¬Impossible([F]) ≡ ◇∃x [F]x› .
  qed
  also AOT_have ‹... ≡ ◇∃x [F]x & ◇∃x ¬[F]x›
  proof (rule "oth-class-taut:4:f"[THEN "→E"])
    AOT_have ‹¬Necessary([F]) ≡ ¬□¬∃x ¬[F]x›
      apply (rule "oth-class-taut:4:b"[THEN "≡E"(1)])
      apply (AOT_subst ‹∃x ¬[F]x› ‹¬ ∀x ¬¬[F]x›)
       apply (simp add: "conventions:4" "≡Df")
      apply (AOT_subst (reverse) ‹¬¬[F]x› ‹[F]x› for: x)
       apply (simp add: "oth-class-taut:3:b")
      apply (AOT_subst (reverse) ‹¬¬∀x [F]x› ‹∀x [F]x›)
      by (auto simp: "oth-class-taut:3:b" "contingent-properties:1" "≡Df")
    also AOT_have ‹... ≡ ◇∃x ¬[F]x›
      using "conventions:5"[THEN "≡Df", symmetric] by blast
    finally AOT_show ‹¬Necessary([F]) ≡ ◇∃x ¬[F]x›.
  qed
  finally show ?thesis.
qed

AOT_theorem "thm-cont-prop:3":                                      (200.3)
  ‹Contingent([F]) ≡ Contingent([F]⁻)› for F::‹<κ> AOT_var›
proof -
  {
    fix Π :: ‹<κ>›
    AOT_assume ‹Π↓›
    moreover AOT_have ‹∀F (Contingent([F]) ≡ ◇∃x [F]x & ◇∃x ¬[F]x)›
      using "thm-cont-prop:2" GEN by fast
    ultimately  AOT_have ‹Contingent([Π]) ≡ ◇∃x [Π]x & ◇∃x ¬[Π]x›
      using "thm-cont-prop:2" "∀E" by fast
  } note 1 = this
  AOT_have ‹Contingent([F]) ≡ ◇∃x [F]x & ◇∃x ¬[F]x›
    using "thm-cont-prop:2" by blast
  also AOT_have ‹... ≡ ◇∃x ¬[F]x & ◇∃x [F]x›
    by (simp add: "Commutativity of &")
  also AOT_have ‹... ≡ ◇∃x [F]⁻x & ◇∃x [F]x›
    by (AOT_subst ‹[F]⁻x› ‹¬[F]x› for: x)
       (auto simp: "thm-relation-negation:1" "oth-class-taut:3:a")
  also AOT_have ‹... ≡ ◇∃x [F]⁻x & ◇∃x ¬[F]⁻x›
    by (AOT_subst (reverse) ‹[F]x› ‹¬[F]⁻x› for: x)
       (auto simp: "thm-relation-negation:2" "oth-class-taut:3:a")
  also AOT_have ‹... ≡ Contingent([F]⁻)›
    using 1[OF "rel-neg-T:3", symmetric] by blast
```

```
  finally show ?thesis.
qed

AOT_define concrete_if_concrete :: ‹Π› ("L")
  L_def: ‹L =df [λx E!x → E!x]›

AOT_theorem "thm-noncont-e-e:1": ‹Necessary(L)›                                    (201.1)
proof -
  AOT_modally_strict {
    fix x
    AOT_have ‹[λx E!x → E!x]↓› by "cqt:2[lambda]"
    moreover AOT_have ‹x↓› using "cqt:2[const_var]"[axiom_inst] by blast
    moreover AOT_have ‹E!x → E!x› using "if-p-then-p" by blast
    ultimately AOT_have ‹[λx E!x → E!x]x›
      using "β←C" by blast
  }
  AOT_hence 0: ‹□∀x [λx E!x → E!x]x›
    using RN GEN by blast
  show ?thesis
    apply (rule "=dfI"(2)[OF L_def])
     apply "cqt:2[lambda]"
    by (rule "contingent-properties:1"[THEN "≡dfI", OF 0])
qed

AOT_theorem "thm-noncont-e-e:2": ‹Impossible([L]¯)›                                (201.2)
proof -
  AOT_modally_strict {
    fix x

    AOT_have 0: ‹∀F (¬[F]¯x ≡ [F]x)›
      using "thm-relation-negation:2" GEN by fast
    AOT_have ‹¬[λx E!x → E!x]¯x ≡ [λx E!x → E!x]x›
      by (rule 0[THEN "∀E"(1)]) "cqt:2[lambda]"
    moreover {
      AOT_have ‹[λx E!x → E!x]↓› by "cqt:2[lambda]"
      moreover AOT_have ‹x↓› using "cqt:2[const_var]"[axiom_inst] by blast
      moreover AOT_have ‹E!x → E!x› using "if-p-then-p" by blast
      ultimately AOT_have ‹[λx E!x → E!x]x›
        using "β←C" by blast
    }
    ultimately AOT_have ‹¬[λx E!x → E!x]¯x›
      using "≡E" by blast
  }
  AOT_hence 0: ‹□∀x ¬[λx E!x → E!x]¯x›
    using RN GEN by fast
  show ?thesis
    apply (rule "=dfI"(2)[OF L_def])
     apply "cqt:2[lambda]"
    apply (rule "contingent-properties:2"[THEN "≡dfI"]; rule "&I")
     using "rel-neg-T:3"
     apply blast
    using 0
    by blast
qed

AOT_theorem "thm-noncont-e-e:3": ‹NonContingent(L)›                                (201.3)
  using "thm-noncont-e-e:1"
  by (rule "contingent-properties:3"[THEN "≡dfI", OF "∨I"(1)])

AOT_theorem "thm-noncont-e-e:4": ‹NonContingent([L]¯)›                             (201.4)
proof -
  AOT_have 0: ‹∀F (NonContingent([F]) ≡ NonContingent([F]¯))›
    using "thm-cont-prop:1" "∀I" by fast
```

```
  moreover AOT_have 1: ‹L↓›
    by (rule "=dfI"(2)[OF L_def]) "cqt:2[lambda]"+
  AOT_show ‹NonContingent([L]⁻)›
    using "∀E"(1)[OF 0, OF 1, THEN "≡E"(1), OF "thm-noncont-e-e:3"] by blast
qed

AOT_theorem "thm-noncont-e-e:5":                                              (201.5)
  ‹∃F ∃G (F ≠ «G::<κ>» & NonContingent([F]) & NonContingent([G]))›
proof (rule "∃I")+
  {
    AOT_have ‹∀F [F] ≠ [F]⁻›
      using "thm-relation-negation:5" GEN by fast
    moreover AOT_have ‹L↓›
      by (rule "=dfI"(2)[OF L_def]) "cqt:2[lambda]"+
    ultimately AOT_have ‹L ≠ [L]⁻›
      using "∀E" by blast
  }
  AOT_thus ‹L ≠ [L]⁻ & NonContingent(L) & NonContingent([L]⁻)›
    using "thm-noncont-e-e:3" "thm-noncont-e-e:4" "&I" by metis
next
  AOT_show ‹[L]⁻↓›
    using "rel-neg-T:3" by blast
next
  AOT_show ‹L↓›
      by (rule "=dfI"(2)[OF L_def]) "cqt:2[lambda]"+
qed

AOT_theorem "lem-cont-e:1": ‹◇∃x ([F]x & ◇¬[F]x) ≡ ◇∃x (¬[F]x & ◇[F]x)›    (202.1)
proof -
  AOT_have ‹◇∃x ([F]x & ◇¬[F]x) ≡ ∃x ◇([F]x & ◇¬[F]x)›
    using "BF◇" "CBF◇" "≡I" by blast
  also AOT_have ‹... ≡ ∃x (◇[F]x &  ◇¬[F]x)›
    by (AOT_subst ‹◇([F]x & ◇¬[F]x)› ‹◇[F]x &  ◇¬[F]x› for: x)
       (auto simp: "S5Basic:11" "cqt-further:7")
  also AOT_have ‹... ≡ ∃x (◇¬[F]x & ◇[F]x)›
    by (AOT_subst ‹◇¬[F]x & ◇[F]x›  ‹◇[F]x & ◇¬[F]x› for: x)
       (auto simp: "Commutativity of &" "cqt-further:7")
  also AOT_have ‹... ≡ ∃x ◇(¬[F]x & ◇[F]x)›
    by (AOT_subst ‹◇(¬[F]x & ◇[F]x)› ‹◇¬[F]x & ◇[F]x› for: x)
       (auto simp: "S5Basic:11" "oth-class-taut:3:a")
  also AOT_have ‹... ≡ ◇∃x (¬[F]x & ◇[F]x)›
    using "BF◇" "CBF◇" "≡I" by fast
  finally show ?thesis.
qed

AOT_theorem "lem-cont-e:2":                                                   (202.2)
  ‹◇∃x ([F]x & ◇¬[F]x) ≡ ◇∃x ([F]⁻x & ◇¬[F]⁻x)›
proof -
  AOT_have ‹◇∃x ([F]x & ◇¬[F]x) ≡ ◇∃x (¬[F]x & ◇[F]x)›
    using "lem-cont-e:1".
  also AOT_have ‹... ≡ ◇∃x ([F]⁻x & ◇¬[F]⁻x)›
    apply (AOT_subst ‹¬[F]⁻x› ‹[F]x› for: x)
     apply (simp add: "thm-relation-negation:2")
    apply (AOT_subst ‹[F]⁻x› ‹¬[F]x› for: x)
     apply (simp add: "thm-relation-negation:1")
    by (simp add: "oth-class-taut:3:a")
  finally show ?thesis.
qed

AOT_theorem "thm-cont-e:1": ‹◇∃x (E!x & ◇¬E!x)›                              (203.1)
proof (rule "CBF◇"[THEN "→E"])
  AOT_have ‹∃x ◇(E!x & ¬𝒜E!x)›
    using "qml:4"[axiom_inst] "BF◇"[THEN "→E"] by blast
```

```
  then AOT_obtain a where ‹◇(E!a & ¬𝒜E!a)›
    using "∃E"[rotated] by blast
  AOT_hence ϑ: ‹◇E!a & ◇¬𝒜E!a›
    using "KBasic2:3"[THEN "→E"] by blast
  AOT_have ξ: ‹◇E!a & ◇𝒜¬E!a›
    by (AOT_subst  ‹𝒜¬E!a› ‹¬𝒜E!a›)
       (auto simp: "logic-actual-nec:1"[axiom_inst] ϑ)
  AOT_have ζ: ‹◇E!a & 𝒜¬E!a›
    by (AOT_subst ‹𝒜¬E!a› ‹◇𝒜¬E!a›)
       (auto simp add: "Act-Sub:4" ξ)
  AOT_hence ‹◇E!a & ◇¬E!a›
    using "&E" "&I" "Act-Sub:3"[THEN "→E"] by blast
  AOT_hence ‹◇(E!a & ◇¬E!a)›
    using "S5Basic:11"[THEN "≡E"(2)] by simp
  AOT_thus ‹∃x ◇(E!x & ◇¬E!x)›
    using "∃I"(2) by fast
qed

AOT_theorem "thm-cont-e:2": ‹◇∃x (¬E!x & ◇E!x)›                          (203.2)
proof -
  AOT_have ‹∀F (◇∃x ([F]x & ◇¬[F]x) ≡ ◇∃x (¬[F]x & ◇[F]x))›
    using "lem-cont-e:1" GEN by fast
  AOT_hence ‹(◇∃x (E!x & ◇¬E!x) ≡ ◇∃x (¬E!x & ◇E!x))›
    using "∀E"(2) by blast
  thus ?thesis using "thm-cont-e:1" "≡E" by blast
qed

AOT_theorem "thm-cont-e:3": ‹◇∃x E!x›                                      (203.3)
proof (rule "CBF◇"[THEN "→E"])
  AOT_obtain a where ‹◇(E!a & ◇¬E!a)›
    using "∃E"[rotated, OF "thm-cont-e:1"[THEN "BF◇"[THEN "→E"]]] by blast
  AOT_hence ‹◇E!a›
    using "KBasic2:3"[THEN "→E", THEN "&E"(1)] by blast
  AOT_thus ‹∃x ◇E!x› using "∃I" by fast
qed

AOT_theorem "thm-cont-e:4": ‹◇∃x ¬E!x›                                     (203.4)
proof (rule "CBF◇"[THEN "→E"])
  AOT_obtain a where ‹◇(E!a & ◇¬E!a)›
    using "∃E"[rotated, OF "thm-cont-e:1"[THEN "BF◇"[THEN "→E"]]] by blast
  AOT_hence ‹◇◇¬E!a›
    using "KBasic2:3"[THEN "→E", THEN "&E"(2)] by blast
  AOT_hence ‹◇¬E!a›
    using "4◇"[THEN "→E"] by blast
  AOT_thus ‹∃x ◇¬E!x› using "∃I" by fast
qed

AOT_theorem "thm-cont-e:5": ‹Contingent([E!])›                             (203.5)
proof -
  AOT_have ‹∀F (Contingent([F]) ≡ ◇∃x [F]x & ◇∃x ¬[F]x)›
    using "thm-cont-prop:2" GEN by fast
  AOT_hence ‹Contingent([E!]) ≡ ◇∃x E!x & ◇∃x ¬E!x›
    using "∀E"(2) by blast
  thus ?thesis
    using "thm-cont-e:3" "thm-cont-e:4" "≡E"(2) "&I" by blast
qed

AOT_theorem "thm-cont-e:6": ‹Contingent([E!]⁻)›                            (203.6)
proof -
  AOT_have ‹∀F (Contingent([«F::<κ>»]) ≡ Contingent([F]⁻))›
    using "thm-cont-prop:3" GEN by fast
  AOT_hence ‹Contingent([E!]) ≡ Contingent([E!]⁻)›
    using "∀E"(2) by fast
```

```
  thus ?thesis using "thm-cont-e:5" "≡E" by blast
qed

AOT_theorem "thm-cont-e:7":                                                     (203.7)
  ‹∃F∃G (Contingent([«F::<κ>»]) & Contingent([G]) & F ≠ G)›
proof (rule "∃I")+
  AOT_have ‹∀F [«F::<κ>»] ≠ [F]⁻›
    using "thm-relation-negation:5" GEN by fast
  AOT_hence ‹[E!] ≠ [E!]⁻›
    using "∀E" by fast
  AOT_thus ‹Contingent([E!]) & Contingent([E!]⁻) & [E!] ≠ [E!]⁻›
    using "thm-cont-e:5" "thm-cont-e:6" "&I" by metis
next
  AOT_show ‹E!⁻↓›
    by (fact AOT)
qed("cqt:2")

AOT_theorem "property-facts:1":                                                 (204.1)
  ‹NonContingent([F]) → ¬∃G (Contingent([G]) & G = F)›
proof (rule "→I"; rule "raa-cor:2")
  AOT_assume ‹NonContingent([F])›
  AOT_hence 1: ‹Necessary([F]) ∨ Impossible([F])›
    using "contingent-properties:3"[THEN "≡dfE"] by blast
  AOT_assume ‹∃G (Contingent([G]) & G = F)›
  then AOT_obtain G where ‹Contingent([G]) & G = F›
    using "∃E"[rotated] by blast
  AOT_hence ‹Contingent([F])› using "rule=E" "&E" by blast
  AOT_hence ‹¬(Necessary([F]) ∨ Impossible([F]))›
    using "contingent-properties:4"[THEN "≡Df", THEN "≡S"(1),
            OF "cqt:2[const_var]"[axiom_inst], THEN "≡E"(1)] by blast
  AOT_thus ‹(Necessary([F]) ∨ Impossible([F])) &
            ¬(Necessary([F]) ∨ Impossible([F]))›
    using 1 "&I" by blast
qed

AOT_theorem "property-facts:2":                                                 (204.2)
  ‹Contingent([F]) → ¬∃G (NonContingent([G]) & G = F)›
proof (rule "→I"; rule "raa-cor:2")
  AOT_assume ‹Contingent([F])›
  AOT_hence 1: ‹¬(Necessary([F]) ∨ Impossible([F]))›
    using "contingent-properties:4"[THEN "≡Df", THEN "≡S"(1),
            OF "cqt:2[const_var]"[axiom_inst], THEN "≡E"(1)] by blast
  AOT_assume ‹∃G (NonContingent([G]) & G = F)›
  then AOT_obtain G where ‹NonContingent([G]) & G = F›
    using "∃E"[rotated] by blast
  AOT_hence ‹NonContingent([F])›
    using "rule=E" "&E" by blast
  AOT_hence ‹Necessary([F]) ∨ Impossible([F])›
    using "contingent-properties:3"[THEN "≡dfE"] by blast
  AOT_thus ‹(Necessary([F]) ∨ Impossible([F])) &
            ¬(Necessary([F]) ∨ Impossible([F]))›
    using 1 "&I" by blast
qed

AOT_theorem "property-facts:3":                                                 (204.3)
  ‹L ≠ [L]⁻ & L ≠ E! & L ≠ E!⁻ & [L]⁻ ≠ [E!]⁻ & E! ≠ [E!]⁻›
proof -
  AOT_have noneqI: ‹Π ≠ Π'› if ‹φ{Π}› and ‹¬φ{Π'}› for φ and Π Π' :: ‹<κ>›
    apply (rule "=-infix"[THEN "≡dfI"]; rule "raa-cor:2")
    using "rule=E"[where φ=φ and τ=Π and σ = Π'] that "&I" by blast
  AOT_have contingent_denotes: ‹Π↓› if ‹Contingent([Π])› for Π :: ‹<κ>›
    using that "contingent-properties:4"[THEN "≡dfE", THEN "&E"(1)] by blast
  AOT_have not_noncontingent_if_contingent:
```

```
    ‹¬NonContingent([Π])› if ‹Contingent([Π])› for Π :: ‹‹κ››
  proof(rule RAA(2))
    AOT_show ‹¬(Necessary([Π]) ∨ Impossible([Π]))›
      using that "contingent-properties:4"[THEN "≡Df", THEN "≡S"(1),
                     OF contingent_denotes[OF that], THEN "≡E"(1)]
      by blast
  next
    AOT_assume ‹NonContingent([Π])›
    AOT_thus ‹Necessary([Π]) ∨ Impossible([Π])›
      using "contingent-properties:3"[THEN "≡dfE"] by blast
  qed

  show ?thesis
  proof (safe intro!: "&I")
    AOT_show ‹L ≠ [L]⁻›
      apply (rule "=dfI"(2)[OF L_def])
       apply "cqt:2[lambda]"
      apply (rule "∀E"(1)[where φ="λ Π . «Π ≠ [Π]⁻»"])
       apply (rule GEN) apply (fact AOT)
      by "cqt:2[lambda]"
  next
    AOT_show ‹L ≠ E!›
      apply (rule noneqI)
      using "thm-noncont-e-e:3"
            not_noncontingent_if_contingent[OF "thm-cont-e:5"]
      by auto
  next
    AOT_show ‹L ≠ E!⁻›
      apply (rule noneqI)
      using "thm-noncont-e-e:3" apply fast
      apply (rule not_noncontingent_if_contingent)
      apply (rule "∀E"(1)[
            where φ="λ Π . «Contingent([Π]) ≡ Contingent([Π]⁻)»",
            rotated, OF contingent_denotes, THEN "≡E"(1), rotated])
      using "thm-cont-prop:3" GEN apply fast
      using "thm-cont-e:5" by fast+
  next
    AOT_show ‹[L]⁻ ≠ E!⁻›
      apply (rule noneqI)
      using "thm-noncont-e-e:4" apply fast
      apply (rule not_noncontingent_if_contingent)
      apply (rule "∀E"(1)[
            where φ="λ Π . «Contingent([Π]) ≡ Contingent([Π]⁻)»",
            rotated, OF contingent_denotes, THEN "≡E"(1), rotated])
      using "thm-cont-prop:3" GEN apply fast
      using "thm-cont-e:5" by fast+
  next
    AOT_show ‹E! ≠ E!⁻›
      apply (rule "=dfI"(2)[OF L_def])
       apply "cqt:2[lambda]"
      apply (rule "∀E"(1)[where φ="λ Π . «Π ≠ [Π]⁻»"])
       apply (rule GEN) apply (fact AOT)
      by "cqt:2"
  qed
qed

AOT_theorem "thm-cont-propos:1":                                    (205.1)
  ‹NonContingent0(p) ≡ NonContingent0(((p)⁻))›
proof(rule "≡I"; rule "→I")
  AOT_assume ‹NonContingent0(p)›
  AOT_hence ‹Necessary0(p) ∨ Impossible0(p)›
    using "contingent-properties:3[zero]"[THEN "≡dfE"] by blast
  moreover {
```

```
    AOT_assume ‹Necessary0(p)›
    AOT_hence 1: ‹□p›
      using "contingent-properties:1[zero]"[THEN "≡dfE"] by blast
    AOT_have ‹□¬((p)¯)›
      by (AOT_subst ‹¬((p)¯)› ‹p›)
         (auto simp add: 1 "thm-relation-negation:4")
    AOT_hence ‹Impossible0(((p)¯))›
      by (rule "contingent-properties:2[zero]"[THEN "≡dfI"])
  }
  moreover {
    AOT_assume ‹Impossible0(p)›
    AOT_hence 1: ‹□¬p›
      by (rule "contingent-properties:2[zero]"[THEN "≡dfE"])
    AOT_have ‹□((p)¯)›
      by (AOT_subst ‹((p)¯)› ‹¬p›)
         (auto simp: 1 "thm-relation-negation:3")
    AOT_hence ‹Necessary0(((p)¯))›
      by (rule "contingent-properties:1[zero]"[THEN "≡dfI"])
  }
  ultimately AOT_have ‹Necessary0(((p)¯)) ∨ Impossible0(((p)¯))›
    using "∨E"(1) "∨I" "→I" by metis
  AOT_thus ‹NonContingent0(((p)¯))›
    using "contingent-properties:3[zero]"[THEN "≡dfI"] by blast
next
  AOT_assume ‹NonContingent0(((p)¯))›
  AOT_hence ‹Necessary0(((p)¯)) ∨ Impossible0(((p)¯))›
    using "contingent-properties:3[zero]"[THEN "≡dfE"] by blast
  moreover {
    AOT_assume ‹Impossible0(((p)¯))›
    AOT_hence 1: ‹□¬((p)¯)›
      by (rule "contingent-properties:2[zero]"[THEN "≡dfE"])
    AOT_have ‹□p›
      by (AOT_subst (reverse) ‹p› ‹¬((p)¯)›)
         (auto simp: 1 "thm-relation-negation:4")
    AOT_hence ‹Necessary0(p)›
      using "contingent-properties:1[zero]"[THEN "≡dfI"] by blast
  }
  moreover {
    AOT_assume ‹Necessary0(((p)¯))›
    AOT_hence 1: ‹□((p)¯)›
      by (rule "contingent-properties:1[zero]"[THEN "≡dfE"])
    AOT_have ‹□¬p›
      by (AOT_subst (reverse) ‹¬p› ‹((p)¯)›)
         (auto simp: 1 "thm-relation-negation:3")
    AOT_hence ‹Impossible0(p)›
      by (rule "contingent-properties:2[zero]"[THEN "≡dfI"])
  }
  ultimately AOT_have ‹Necessary0(p) ∨ Impossible0(p)›
    using "∨E"(1) "∨I" "→I" by metis
  AOT_thus ‹NonContingent0(p)›
    using "contingent-properties:3[zero]"[THEN "≡dfI"] by blast
qed

AOT_theorem "thm-cont-propos:2": ‹Contingent0(φ) ≡ ◇φ & ◇¬φ›          (205.2)
proof -
  AOT_have ‹Contingent0(φ) ≡ ¬(Necessary0(φ) ∨ Impossible0(φ))›
    using "contingent-properties:4[zero]"[THEN "≡Df"] by simp
  also AOT_have ‹... ≡ ¬Necessary0(φ) & ¬Impossible0(φ)›
    by (fact AOT)
  also AOT_have ‹... ≡ ¬Impossible0(φ) & ¬Necessary0(φ)›
    by (fact AOT)
  also AOT_have ‹... ≡ ◇φ & ◇¬φ›
    apply (AOT_subst ‹◇φ› ‹¬□¬φ›)
```

```
     apply (simp add: "conventions:5" "≡Df")
    apply (AOT_subst ‹Impossible0(φ)› ‹□¬φ›)
     apply (simp add: "contingent-properties:2[zero]" "≡Df")
    apply (AOT_subst (reverse) ‹◊¬φ› ‹¬□φ›)
     apply (simp add: "KBasic:11")
    apply (AOT_subst ‹Necessary0(φ)› ‹□φ›)
     apply (simp add: "contingent-properties:1[zero]" "≡Df")
    by (simp add: "oth-class-taut:3:a")
  finally show ?thesis.
qed

AOT_theorem "thm-cont-propos:3": ‹Contingent0(p) ≡ Contingent0(((p)^-))›      (205.3)
proof -
  AOT_have ‹Contingent0(p) ≡ ◊p & ◊¬p› using "thm-cont-propos:2".
  also AOT_have ‹... ≡ ◊¬p & ◊p› by (fact AOT)
  also AOT_have ‹... ≡ ◊((p)^-) & ◊p›
    by (AOT_subst ‹((p)^-)› ‹¬p›)
       (auto simp: "thm-relation-negation:3" "oth-class-taut:3:a")
  also AOT_have ‹... ≡ ◊((p)^-) & ◊¬((p)^-)›
    by (AOT_subst ‹¬((p)^-)› ‹p›)
       (auto simp: "thm-relation-negation:4" "oth-class-taut:3:a")
  also AOT_have ‹... ≡ Contingent0(((p)^-))›
    using "thm-cont-propos:2"[symmetric] by blast
  finally show ?thesis.
qed

AOT_define noncontingent_prop :: ‹φ› ("p_0")
  p_0_def: "(p_0) =_df (∀x (E!x → E!x))"

AOT_theorem "thm-noncont-propos:1":  ‹Necessary0((p_0))›      (206.1)
proof(rule "contingent-properties:1[zero]"[THEN "≡_dfI"])
  AOT_show ‹□(p_0)›
    apply (rule "=_dfI"(2)[OF p_0_def])
    using "log-prop-prop:2" apply simp
    using "if-p-then-p" RN GEN by fast
qed

AOT_theorem "thm-noncont-propos:2": ‹Impossible0(((p_0)^-))›      (206.2)
proof(rule "contingent-properties:2[zero]"[THEN "≡_dfI"])
  AOT_show ‹□¬((p_0)^-)›
    apply (AOT_subst ‹((p_0)^-)› ‹¬p_0›)
    using "thm-relation-negation:3" GEN "∀E"(1)[rotated, OF "log-prop-prop:2"]
     apply fast
    apply (AOT_subst (reverse) ‹¬¬p_0› ‹p_0›)
     apply (simp add: "oth-class-taut:3:b")
    apply (rule "=_dfI"(2)[OF p_0_def])
    using "log-prop-prop:2" apply simp
    using "if-p-then-p" RN GEN by fast
qed

AOT_theorem "thm-noncont-propos:3": ‹NonContingent0((p_0))›      (206.3)
  apply(rule "contingent-properties:3[zero]"[THEN "≡_dfI"])
  using "thm-noncont-propos:1" "∨I" by blast

AOT_theorem "thm-noncont-propos:4": ‹NonContingent0(((p_0)^-))›      (206.4)
  apply(rule "contingent-properties:3[zero]"[THEN "≡_dfI"])
  using "thm-noncont-propos:2" "∨I" by blast

AOT_theorem "thm-noncont-propos:5":      (206.5)
  ‹∃p∃q (NonContingent0((p)) & NonContingent0((q)) & p ≠ q)›
proof(rule "∃I")+
  AOT_have 0: ‹φ ≠ (φ)^-› for φ
    using "thm-relation-negation:6" "∀I"
```

```
        "∀E"(1)[rotated, OF "log-prop-prop:2"] by fast
  AOT_thus ‹NonContingent0((p₀)) & NonContingent0(((p₀)⁻)) & (p₀) ≠ (p₀)⁻›
    using "thm-noncont-propos:3" "thm-noncont-propos:4" "&I" by auto
qed(auto simp: "log-prop-prop:2")

AOT_act_theorem "no-cnac": ‹¬∃x(E!x & ¬𝒜E!x)›                    (207)
proof(rule "raa-cor:2")
  AOT_assume ‹∃x(E!x & ¬𝒜E!x)›
  then AOT_obtain a where a: ‹E!a & ¬𝒜E!a›
    using "∃E"[rotated] by blast
  AOT_hence ‹𝒜¬E!a›
    using "&E" "logic-actual-nec:1"[axiom_inst, THEN "≡E"(2)] by blast
  AOT_hence ‹¬E!a›
    using "logic-actual"[act_axiom_inst, THEN "→E"] by blast
  AOT_hence ‹E!a & ¬E!a›
    using a "&E" "&I" by blast
  AOT_thus ‹p & ¬p› for p using "raa-cor:1" by blast
qed

AOT_theorem "pos-not-pna:1": ‹¬𝒜∃x (E!x & ¬𝒜E!x)›               (208.1)
proof(rule "raa-cor:2")
  AOT_assume ‹𝒜∃x (E!x & ¬𝒜E!x)›
  AOT_hence ‹∃x 𝒜(E!x & ¬𝒜E!x)›
    using "Act-Basic:10"[THEN "≡E"(1)] by blast
  then AOT_obtain a where ‹𝒜(E!a & ¬𝒜E!a)›
    using "∃E"[rotated] by blast
  AOT_hence 1: ‹𝒜E!a & 𝒜¬𝒜E!a›
    using "Act-Basic:2"[THEN "≡E"(1)] by blast
  AOT_hence ‹¬𝒜𝒜E!a›
    using "&E"(2) "logic-actual-nec:1"[axiom_inst, THEN "≡E"(1)] by blast
  AOT_hence ‹¬𝒜E!a›
    using "logic-actual-nec:4"[axiom_inst, THEN "≡E"(1)] RAA by blast
  AOT_thus ‹p & ¬p› for p using 1[THEN "&E"(1)] "&I" "raa-cor:1" by blast
qed

AOT_theorem "pos-not-pna:2": ‹◇¬∃x(E!x & ¬𝒜E!x)›                (208.2)
proof (rule RAA(1))
  AOT_show ‹¬𝒜∃x (E!x & ¬𝒜E!x)›
    using "pos-not-pna:1" by blast
next
  AOT_assume ‹¬◇¬∃x (E!x & ¬𝒜E!x)›
  AOT_hence ‹□∃x (E!x & ¬𝒜E!x)›
    using "KBasic:12"[THEN "≡E"(2)] by blast
  AOT_thus ‹𝒜∃x (E!x & ¬𝒜E!x)›
    using "nec-imp-act"[THEN "→E"] by blast
qed

AOT_theorem "pos-not-pna:3": ‹∃x (◇E!x & ¬𝒜E!x)›                (208.3)
proof -
  AOT_obtain a where ‹◇(E!a & ¬𝒜E!a)›
    using "qml:4"[axiom_inst] "BF◇"[THEN "→E"] "∃E"[rotated] by blast
  AOT_hence θ: ‹◇E!a› and ξ: ‹◇¬𝒜E!a›
    using "KBasic2:3"[THEN "→E"] "&E" by blast+
  AOT_have ‹¬□𝒜E!a›
    using ξ "KBasic:11"[THEN "≡E"(2)] by blast
  AOT_hence ‹¬𝒜E!a›
    using "Act-Basic:6"[THEN "oth-class-taut:4:b"[THEN "≡E"(1)],
                        THEN "≡E"(2)] by blast
  AOT_hence ‹◇E!a & ¬𝒜E!a› using θ "&I" by blast
  thus ?thesis using "∃I" by fast
qed

AOT_define contingent_prop :: φ ("q₀")
```

```
  q0_def: ‹(q0) =df (∃x (E!x & ¬𝒜E!x))›

AOT_theorem q0_prop: ‹◇q0 & ◇¬q0›
  apply (rule "=dfI"(2)[OF q0_def])
  apply (fact "log-prop-prop:2")
  apply (rule "&I")
   apply (fact "qml:4"[axiom_inst])
  by (fact "pos-not-pna:2")

AOT_theorem "basic-prop:1": ‹Contingent0((q0))›                                   (209.1)
proof(rule "contingent-properties:4[zero]"[THEN "≡dfI"])
  AOT_have ‹¬Necessary0((q0)) & ¬Impossible0((q0))›
  proof (rule "&I";
         rule "=dfI"(2)[OF q0_def];
         (rule "log-prop-prop:2" | rule "raa-cor:2"))
    AOT_assume ‹Necessary0(∃x (E!x & ¬𝒜E!x))›
    AOT_hence ‹□∃x (E!x & ¬𝒜E!x)›
      using "contingent-properties:1[zero]"[THEN "≡dfE"] by blast
    AOT_hence ‹𝒜∃x (E!x & ¬𝒜E!x)›
      using "Act-Basic:8"[THEN "→E"] "qml:2"[axiom_inst, THEN "→E"] by blast
    AOT_thus ‹𝒜∃x (E!x & ¬𝒜E!x) & ¬𝒜∃x (E!x & ¬𝒜E!x)›
      using "pos-not-pna:1" "&I" by blast
  next
    AOT_assume ‹Impossible0(∃x (E!x & ¬𝒜E!x))›
    AOT_hence ‹□¬(∃x (E!x & ¬𝒜E!x))›
      using "contingent-properties:2[zero]"[THEN "≡dfE"] by blast
    AOT_hence ‹¬◇(∃x (E!x & ¬𝒜E!x))›
      using "KBasic2:1"[THEN "≡E"(1)] by blast
    AOT_thus ‹◇(∃x (E!x & ¬𝒜E!x)) & ¬◇(∃x (E!x & ¬𝒜E!x))›
      using "qml:4"[axiom_inst] "&I" by blast
  qed
  AOT_thus ‹¬(Necessary0((q0)) ∨ Impossible0((q0)))›
    using "oth-class-taut:5:d" "≡E"(2) by blast
qed

AOT_theorem "basic-prop:2": ‹∃p Contingent0((p))›                                 (209.2)
  using "∃I"(1)[rotated, OF "log-prop-prop:2"] "basic-prop:1" by blast

AOT_theorem "basic-prop:3": ‹Contingent0(((q0)¯))›                                (209.3)
  apply (AOT_subst ‹((q0)¯)› ‹¬q0›)
   apply (insert "thm-relation-negation:3" "∀I"
              "∀E"(1)[rotated, OF "log-prop-prop:2"]; fast)
  apply (rule "contingent-properties:4[zero]"[THEN "≡dfI"])
  apply (rule "oth-class-taut:5:d"[THEN "≡E"(2)])
  apply (rule "&I")
   apply (rule "contingent-properties:1[zero]"[THEN "df-rules-formulas[3]",
                THEN "useful-tautologies:5"[THEN "→E"], THEN "→E"])
   apply (rule "conventions:5"[THEN "≡dfE"])
   apply (rule "=dfE"(2)[OF q0_def])
    apply (rule "log-prop-prop:2")
   apply (rule q0_prop[THEN "&E"(1)])
  apply (rule "contingent-properties:2[zero]"[THEN "df-rules-formulas[3]",
              THEN "useful-tautologies:5"[THEN "→E"], THEN "→E"])
  apply (rule "conventions:5"[THEN "≡dfE"])
  by (rule q0_prop[THEN "&E"(2)])

AOT_theorem "basic-prop:4":                                                       (209.4)
  ‹∃p∃q (p ≠ q & Contingent0(p) & Contingent0(q))›
proof(rule "∃I")+
  AOT_have 0: ‹φ ≠ (φ)¯› for φ
    using "thm-relation-negation:6" "∀I"
          "∀E"(1)[rotated, OF "log-prop-prop:2"] by fast
  AOT_show ‹(q0) ≠ (q0)¯ & Contingent0(q0) & Contingent0(((q0)¯))›
```

```
    using "basic-prop:1" "basic-prop:3" "&I" 0 by presburger
qed(auto simp: "log-prop-prop:2")

AOT_theorem "proposition-facts:1":                                            (210.1)
  ‹NonContingent0(p) → ¬∃q (Contingent0(q) & q = p)›
proof(rule "→I"; rule "raa-cor:2")
  AOT_assume ‹NonContingent0(p)›
  AOT_hence 1: ‹Necessary0(p) ∨ Impossible0(p)›
    using "contingent-properties:3[zero]"[THEN "≡dfE"] by blast
  AOT_assume ‹∃q (Contingent0(q) & q = p)›
  then AOT_obtain q where ‹Contingent0(q) & q = p›
    using "∃E"[rotated] by blast
  AOT_hence ‹Contingent0(p)›
    using "rule=E" "&E" by fast
  AOT_thus ‹(Necessary0(p) ∨ Impossible0(p)) &
          ¬(Necessary0(p) ∨ Impossible0(p))›
    using "contingent-properties:4[zero]"[THEN "≡dfE"] 1 "&I" by blast
qed

AOT_theorem "proposition-facts:2":                                            (210.2)
  ‹Contingent0(p) → ¬∃q (NonContingent0(q) & q = p)›
proof(rule "→I"; rule "raa-cor:2")
  AOT_assume ‹Contingent0(p)›
  AOT_hence 1: ‹¬(Necessary0(p) ∨ Impossible0(p))›
    using "contingent-properties:4[zero]"[THEN "≡dfE"] by blast
  AOT_assume ‹∃q (NonContingent0(q) & q = p)›
  then AOT_obtain q where ‹NonContingent0(q) & q = p›
    using "∃E"[rotated] by blast
  AOT_hence ‹NonContingent0(p)›
    using "rule=E" "&E" by fast
  AOT_thus ‹(Necessary0(p) ∨ Impossible0(p)) &
          ¬(Necessary0(p) ∨ Impossible0(p))›
    using "contingent-properties:3[zero]"[THEN "≡dfE"] 1 "&I" by blast
qed

AOT_theorem "proposition-facts:3":                                            (210.3)
  ‹(p0) ≠ (p0)‾ & (p0) ≠ (q0) & (p0) ≠ (q0)‾ & (p0)‾ ≠ (q0)‾ & (q0) ≠ (q0)‾›
proof -
  {
    fix χ φ ψ
    AOT_assume ‹χ{φ}›
    moreover AOT_assume ‹¬χ{ψ}›
    ultimately AOT_have ‹¬(χ{φ} ≡ χ{ψ})›
      using RAA "≡E" by metis
    moreover {
      AOT_have ‹∀p∀q ((¬(χ{p} ≡ χ{q})) → p ≠ q)›
        by (rule "∀I"; rule "∀I"; rule "pos-not-equiv-ne:4[zero]")
      AOT_hence ‹((¬(χ{φ} ≡ χ{ψ})) → φ ≠ ψ)›
        using "∀E" "log-prop-prop:2" by blast
    }
    ultimately AOT_have ‹φ ≠ ψ›
      using "→E" by blast
  } note 0 = this
  AOT_have contingent_neg: ‹Contingent0(φ) ≡ Contingent0(((φ)‾))› for φ
    using "thm-cont-propos:3" "∀I"
          "∀E"(1)[rotated, OF "log-prop-prop:2"] by fast
  AOT_have not_noncontingent_if_contingent:
    ‹¬NonContingent0(φ)› if ‹Contingent0(φ)› for φ
    apply (rule "contingent-properties:3[zero]"[THEN "≡Df",
                THEN "oth-class-taut:4:b"[THEN "≡E"(1)], THEN "≡E"(2)])
    using that "contingent-properties:4[zero]"[THEN "≡dfE"] by blast
  show ?thesis
    apply (rule "&I")+
```

```
    using "thm-relation-negation:6" "∀I"
          "∀E"(1)[rotated, OF "log-prop-prop:2"]
        apply fast
       apply (rule 0)
    using "thm-noncont-propos:3" apply fast
       apply (rule not_noncontingent_if_contingent)
       apply (fact AOT)
      apply (rule 0)
    apply (rule "thm-noncont-propos:3")
      apply (rule not_noncontingent_if_contingent)
      apply (rule contingent_neg[THEN "≡E"(1)])
      apply (fact AOT)
     apply (rule 0)
    apply (rule "thm-noncont-propos:4")
      apply (rule not_noncontingent_if_contingent)
      apply (rule contingent_neg[THEN "≡E"(1)])
     apply (fact AOT)
    using "thm-relation-negation:6" "∀I"
          "∀E"(1)[rotated, OF "log-prop-prop:2"] by fast
qed

AOT_define ContingentlyTrue :: ‹φ ⇒ φ› ("ContingentlyTrue'(_')")
  "cont-tf:1": ‹ContingentlyTrue(p) ≡df p & ◇¬p›                          (211.1)

AOT_define ContingentlyFalse :: ‹φ ⇒ φ› ("ContingentlyFalse'(_')")
  "cont-tf:2": ‹ContingentlyFalse(p) ≡df ¬p & ◇p›                         (211.2)

AOT_theorem "cont-true-cont:1":                                           (212.1)
  ‹ContingentlyTrue((p)) → Contingent0((p))›
proof(rule "→I")
  AOT_assume ‹ContingentlyTrue((p))›
  AOT_hence 1: ‹p› and 2: ‹◇¬p› using "cont-tf:1"[THEN "≡dfE"] "&E" by blast+
  AOT_have ‹¬Necessary0((p))›
    apply (rule "contingent-properties:1[zero]"[THEN "≡Df",
                THEN "oth-class-taut:4:b"[THEN "≡E"(1)], THEN "≡E"(2)])
    using 2 "KBasic:11"[THEN "≡E"(2)] by blast
  moreover AOT_have ‹¬Impossible0((p))›
    apply (rule "contingent-properties:2[zero]"[THEN "≡Df",
                THEN "oth-class-taut:4:b"[THEN "≡E"(1)], THEN "≡E"(2)])
    apply (rule "conventions:5"[THEN "≡dfE"])
    using "T◇"[THEN "→E", OF 1].
  ultimately AOT_have ‹¬(Necessary0((p)) ∨ Impossible0((p)))›
    using DeMorgan(2)[THEN "≡E"(2)] "&I" by blast
  AOT_thus ‹Contingent0((p))›
    using "contingent-properties:4[zero]"[THEN "≡dfI"] by blast
qed

AOT_theorem "cont-true-cont:2":                                           (212.2)
  ‹ContingentlyFalse((p)) → Contingent0((p))›
proof(rule "→I")
  AOT_assume ‹ContingentlyFalse((p))›
  AOT_hence 1: ‹¬p› and 2: ‹◇p› using "cont-tf:2"[THEN "≡dfE"] "&E" by blast+
  AOT_have ‹¬Necessary0((p))›
    apply (rule "contingent-properties:1[zero]"[THEN "≡Df",
                THEN "oth-class-taut:4:b"[THEN "≡E"(1)], THEN "≡E"(2)])
    using "KBasic:11"[THEN "≡E"(2)] "T◇"[THEN "→E", OF 1] by blast
  moreover AOT_have ‹¬Impossible0((p))›
    apply (rule "contingent-properties:2[zero]"[THEN "≡Df",
                THEN "oth-class-taut:4:b"[THEN "≡E"(1)], THEN "≡E"(2)])
    apply (rule "conventions:5"[THEN "≡dfE"])
    using 2.
  ultimately AOT_have ‹¬(Necessary0((p)) ∨ Impossible0((p)))›
    using DeMorgan(2)[THEN "≡E"(2)] "&I" by blast
```

```
  AOT_thus ‹Contingent0((p))›
    using "contingent-properties:4[zero]"[THEN "≡dfI"] by blast
qed

AOT_theorem "cont-true-cont:3":                                        (212.3)
  ‹ContingentlyTrue((p)) ≡ ContingentlyFalse(((p)¯))›
proof(rule "≡I"; rule "→I")
  AOT_assume ‹ContingentlyTrue((p))›
  AOT_hence 0: ‹p & ◇¬p› using "cont-tf:1"[THEN "≡dfE"] by blast
  AOT_have 1: ‹ContingentlyFalse(¬p)›
    apply (rule "cont-tf:2"[THEN "≡dfI"])
    apply (AOT_subst (reverse) ‹¬¬p› p)
    by (auto simp: "oth-class-taut:3:b" 0)
  AOT_show ‹ContingentlyFalse(((p)¯))›
    apply (AOT_subst ‹((p)¯)› ‹¬p›)
    by (auto simp: "thm-relation-negation:3" 1)
next
  AOT_assume 1: ‹ContingentlyFalse(((p)¯))›
  AOT_have ‹ContingentlyFalse(¬p)›
    by (AOT_subst (reverse) ‹¬p› ‹((p)¯)›)
       (auto simp: "thm-relation-negation:3" 1)
  AOT_hence ‹¬¬p & ◇¬p› using "cont-tf:2"[THEN "≡dfE"] by blast
  AOT_hence ‹p & ◇¬p›
    using "&I" "&E" "useful-tautologies:1"[THEN "→E"] by metis
  AOT_thus ‹ContingentlyTrue((p))›
    using "cont-tf:1"[THEN "≡dfI"] by blast
qed

AOT_theorem "cont-true-cont:4":                                        (212.4)
  ‹ContingentlyFalse((p)) ≡ ContingentlyTrue(((p)¯))›
proof(rule "≡I"; rule "→I")
  AOT_assume ‹ContingentlyFalse(p)›
  AOT_hence 0: ‹¬p & ◇p›
    using "cont-tf:2"[THEN "≡dfE"] by blast
  AOT_have ‹¬p & ◇¬¬p›
    by (AOT_subst (reverse) ‹¬¬p› p)
       (auto simp: "oth-class-taut:3:b" 0)
  AOT_hence 1: ‹ContingentlyTrue(¬p)›
    by (rule "cont-tf:1"[THEN "≡dfI"])
  AOT_show ‹ContingentlyTrue(((p)¯))›
    by (AOT_subst ‹((p)¯)› ‹¬p›)
       (auto simp: "thm-relation-negation:3" 1)
next
  AOT_assume 1: ‹ContingentlyTrue(((p)¯))›
  AOT_have ‹ContingentlyTrue(¬p)›
    by (AOT_subst (reverse) ‹¬p› ‹((p)¯)›)
       (auto simp add: "thm-relation-negation:3" 1)
  AOT_hence 2: ‹¬p & ◇¬¬p› using "cont-tf:1"[THEN "≡dfE"] by blast
  AOT_have ‹◇p›
    by (AOT_subst p ‹¬¬p›)
       (auto simp add: "oth-class-taut:3:b" 2[THEN "&E"(2)])
  AOT_hence ‹¬p & ◇p› using 2[THEN "&E"(1)] "&I" by blast
  AOT_thus ‹ContingentlyFalse(p)›
    by (rule "cont-tf:2"[THEN "≡dfI"])
qed

AOT_theorem "cont-true-cont:5":                                        (212.5)
  ‹(ContingentlyTrue((p)) & Necessary0((q))) → p ≠ q›
proof (rule "→I"; frule "&E"(1); drule "&E"(2); rule "raa-cor:1")
  AOT_assume ‹ContingentlyTrue((p))›
  AOT_hence ‹◇¬p›
    using "cont-tf:1"[THEN "≡dfE"] "&E" by blast
  AOT_hence 0: ‹¬□p› using "KBasic:11"[THEN "≡E"(2)] by blast
```

```
  AOT_assume ‹Necessary0((q))›
  moreover AOT_assume ‹¬(p ≠ q)›
  AOT_hence ‹p = q›
    using "=-infix"[THEN "≡Df",
                    THEN "oth-class-taut:4:b"[THEN "≡E"(1)],
                    THEN "≡E"(1)]
          "useful-tautologies:1"[THEN "→E"] by blast
  ultimately AOT_have ‹Necessary0((p))› using "rule=E" id_sym by blast
  AOT_hence ‹□p›
    using "contingent-properties:1[zero]"[THEN "≡dfE"] by blast
  AOT_thus ‹□p & ¬□p› using 0 "&I" by blast
qed

AOT_theorem "cont-true-cont:6":                                         (212.6)
  ‹(ContingentlyFalse((p)) & Impossible0((q))) → p ≠ q›
proof (rule "→I"; frule "&E"(1); drule "&E"(2); rule "raa-cor:1")
  AOT_assume ‹ContingentlyFalse((p))›
  AOT_hence ‹◊p›
    using "cont-tf:2"[THEN "≡dfE"] "&E" by blast
  AOT_hence 1: ‹¬□¬p›
    using "conventions:5"[THEN "≡dfE"] by blast
  AOT_assume ‹Impossible0((q))›
  moreover AOT_assume ‹¬(p ≠ q)›
  AOT_hence ‹p = q›
    using "=-infix"[THEN "≡Df",
                    THEN "oth-class-taut:4:b"[THEN "≡E"(1)],
                    THEN "≡E"(1)]
          "useful-tautologies:1"[THEN "→E"] by blast
  ultimately AOT_have ‹Impossible0((p))› using "rule=E" id_sym by blast
  AOT_hence ‹□¬p›
    using "contingent-properties:2[zero]"[THEN "≡dfE"] by blast
  AOT_thus ‹□¬p & ¬□¬p› using 1 "&I" by blast
qed

AOT_act_theorem "q0cf:1": ‹ContingentlyFalse(q0)›                       (213.1)
  apply (rule "cont-tf:2"[THEN "≡dfI"])
  apply (rule "=dfI"(2)[OF q0_def])
   apply (fact "log-prop-prop:2")
  apply (rule "&I")
   apply (fact "no-cnac")
  by (fact "qml:4"[axiom_inst])

AOT_act_theorem "q0cf:2": ‹ContingentlyTrue(((q0)¯))›                   (213.2)
  apply (rule "cont-tf:1"[THEN "≡dfI"])
  apply (rule "=dfI"(2)[OF q0_def])
   apply (fact "log-prop-prop:2")
  apply (rule "&I")
   apply (rule "thm-relation-negation:3"
               [unvarify p, OF "log-prop-prop:2", THEN "≡E"(2)])
     apply (fact "no-cnac")
  apply (rule "rule=E"[rotated,
               OF "thm-relation-negation:7"
                  [unvarify p, OF "log-prop-prop:2", THEN id_sym]])
  apply (AOT_subst (reverse) ‹¬¬(∃x  (E!x & ¬𝒜E!x))› ‹∃x (E!x & ¬𝒜E!x)›)
  by (auto simp: "oth-class-taut:3:b" "qml:4"[axiom_inst])

AOT_theorem "cont-tf-thm:1": ‹∃p ContingentlyTrue((p))›                  (215.1)
proof(rule "∨E"(1)[OF "exc-mid"]; rule "→I"; rule "∃I")
  AOT_assume ‹q0›
  AOT_hence ‹q0 & ◊¬q0› using q0_prop[THEN "&E"(2)] "&I" by blast
  AOT_thus ‹ContingentlyTrue(q0)›
    by (rule "cont-tf:1"[THEN "≡dfI"])
next
```

```
  AOT_assume ‹¬q0›
  AOT_hence ‹¬q0 & ◇q0› using q0_prop[THEN "&E"(1)] "&I" by blast
  AOT_hence ‹ContingentlyFalse(q0)›
    by (rule "cont-tf:2"[THEN "≡dfI"])
  AOT_thus ‹ContingentlyTrue(((q0)-))›
    by (rule "cont-true-cont:4"[unvarify p,
                  OF "log-prop-prop:2", THEN "≡E"(1)])
qed(auto simp: "log-prop-prop:2")


AOT_theorem "cont-tf-thm:2": ‹∃p ContingentlyFalse((p))›          (215.2)
proof(rule "∨E"(1)[OF "exc-mid"]; rule "→I"; rule "∃I")
  AOT_assume ‹q0›
  AOT_hence ‹q0 & ◇¬q0› using q0_prop[THEN "&E"(2)] "&I" by blast
  AOT_hence ‹ContingentlyTrue(q0)›
    by (rule "cont-tf:1"[THEN "≡dfI"])
  AOT_thus ‹ContingentlyFalse(((q0)-))›
    by (rule "cont-true-cont:3"[unvarify p,
                  OF "log-prop-prop:2", THEN "≡E"(1)])
next
  AOT_assume ‹¬q0›
  AOT_hence ‹¬q0 & ◇q0› using q0_prop[THEN "&E"(1)] "&I" by blast
  AOT_thus ‹ContingentlyFalse(q0)›
    by (rule "cont-tf:2"[THEN "≡dfI"])
qed(auto simp: "log-prop-prop:2")

AOT_theorem "property-facts1:1": ‹∃F∃x ([F]x & ◇¬[F]x)›          (217.1)
proof -
  fix x
  AOT_obtain p1 where ‹ContingentlyTrue((p1))›
    using "cont-tf-thm:1" "∃E"[rotated] by blast
  AOT_hence 1: ‹p1 & ◇¬p1› using "cont-tf:1"[THEN "≡dfE"] by blast
  AOT_modally_strict {
    AOT_have ‹for arbitrary p: ⊢□ ([λz p]x ≡ p)›
      by (rule "beta-C-cor:3"[THEN "∀E"(2)]) cqt_2_lambda_inst_prover
    AOT_hence ‹for arbitrary p: ⊢□ □ ([λz p]x ≡ p)›
      by (rule RN)
    AOT_hence ‹∀p □([λz p]x ≡ p)› using GEN by fast
    AOT_hence ‹□([λz p1]x ≡ p1)› using "∀E" by fast
  } note 2 = this
  AOT_hence ‹□([λz p1]x ≡ p1)› using "∀E" by blast
  AOT_hence ‹[λz p1]x›
    using 1[THEN "&E"(1)] "qml:2"[axiom_inst, THEN "→E"] "≡E"(2) by blast
  moreover AOT_have ‹◇¬[λz p1]x›
    using 2[THEN "qml:2"[axiom_inst, THEN "→E"]]
    apply (AOT_subst ‹[λz p1]x› ‹p1›)
    using 1[THEN "&E"(2)] by blast
  ultimately AOT_have ‹[λz p1]x & ◇¬[λz p1]x› using "&I" by blast
  AOT_hence ‹∃x ([λz p1]x & ◇¬[λz p1]x)› using "∃I"(2) by fast
  moreover AOT_have ‹[λz p1]↓› by "cqt:2[lambda]"
  ultimately AOT_show ‹∃F∃x ([F]x & ◇¬[F]x)› by (rule "∃I"(1))
qed

AOT_theorem "property-facts1:2": ‹∃F∃x (¬[F]x & ◇[F]x)›          (217.2)
proof -
  fix x
  AOT_obtain p1 where ‹ContingentlyFalse((p1))›
    using "cont-tf-thm:2" "∃E"[rotated] by blast
  AOT_hence 1: ‹¬p1 & ◇p1› using "cont-tf:2"[THEN "≡dfE"] by blast
  AOT_modally_strict {
    AOT_have ‹for arbitrary p: ⊢□ ([λz p]x ≡ p)›
      by (rule "beta-C-cor:3"[THEN "∀E"(2)]) cqt_2_lambda_inst_prover
    AOT_hence ‹for arbitrary p: ⊢□ (¬[λz p]x ≡ ¬p)›
```

```
      using "oth-class-taut:4:b" "≡E" by blast
    AOT_hence ‹for arbitrary p: ⊢□ □(¬[λz p]x ≡ ¬p)›
      by (rule RN)
    AOT_hence ‹∀p □(¬[λz p]x ≡ ¬p)› using GEN by fast
    AOT_hence ‹□(¬[λz p1]x ≡ ¬p1)› using "∀E" by fast
  } note 2 = this
  AOT_hence ‹□(¬[λz p1]x ≡ ¬p1)› using "∀E" by blast
  AOT_hence 3: ‹¬[λz p1]x›
    using 1[THEN "&E"(1)] "qml:2"[axiom_inst, THEN "→E"] "≡E"(2) by blast
  AOT_modally_strict {
    AOT_have ‹for arbitrary p: ⊢□ ([λz p]x ≡ p)›
      by (rule "beta-C-cor:3"[THEN "∀E"(2)]) cqt_2_lambda_inst_prover
    AOT_hence ‹for arbitrary p: ⊢□ □([λz p]x ≡ p)›
      by (rule RN)
    AOT_hence ‹∀p □([λz p]x ≡ p)› using GEN by fast
    AOT_hence ‹□([λz p1]x ≡ p1)› using "∀E" by fast
  } note 4 = this
  AOT_have ‹◇[λz p1]x›
    using 4[THEN "qml:2"[axiom_inst, THEN "→E"]]
    apply (AOT_subst ‹[λz p1]x› ‹p1›)
    using 1[THEN "&E"(2)] by blast
  AOT_hence ‹¬[λz p1]x & ◇[λz p1]x› using 3 "&I" by blast
  AOT_hence ‹∃x (¬[λz p1]x & ◇[λz p1]x)› using "∃I"(2) by fast
  moreover AOT_have ‹[λz p1]↓› by "cqt:2[lambda]"
  ultimately AOT_show ‹∃F∃x (¬[F]x & ◇[F]x)› by (rule "∃I"(1))
qed

context
begin

private AOT_lemma eqnotnec_123_Aux_ζ: ‹[L]x ≡ (E!x → E!x)›
    apply (rule "=dfI"(2)[OF L_def])
     apply "cqt:2[lambda]"
    apply (rule "beta-C-meta"[THEN "→E"])
  by "cqt:2[lambda]"

private AOT_lemma eqnotnec_123_Aux_ω: ‹[λz φ]x ≡ φ›
    by (rule "beta-C-meta"[THEN "→E"]) "cqt:2[lambda]"

private AOT_lemma eqnotnec_123_Aux_ϑ: ‹φ ≡ ∀x([L]x ≡ [λz φ]x)›
proof(rule "≡I"; rule "→I"; (rule "∀I")?)
  fix x
  AOT_assume 1: ‹φ›
  AOT_have ‹[L]x ≡ (E!x → E!x)› using eqnotnec_123_Aux_ζ.
  also AOT_have ‹... ≡ φ›
    using "if-p-then-p" 1 "≡I" "→I" by simp
  also AOT_have ‹... ≡ [λz φ]x›
    using "Commutativity of ≡"[THEN "≡E"(1)] eqnotnec_123_Aux_ω by blast
  finally AOT_show ‹[L]x ≡ [λz φ]x›.
next
  fix x
  AOT_assume ‹∀x([L]x ≡ [λz φ]x)›
  AOT_hence ‹[L]x ≡ [λz φ]x› using "∀E" by blast
  also AOT_have ‹... ≡ φ› using eqnotnec_123_Aux_ω.
  finally AOT_have ‹φ ≡ [L]x›
    using "Commutativity of ≡"[THEN "≡E"(1)] by blast
  also AOT_have ‹... ≡ E!x → E!x› using eqnotnec_123_Aux_ζ.
  finally AOT_show ‹φ› using "≡E" "if-p-then-p" by fast
qed
private lemmas eqnotnec_123_Aux_ξ =
  eqnotnec_123_Aux_ϑ[THEN "oth-class-taut:4:b"[THEN "≡E"(1)],
    THEN "conventions:3"[THEN "≡Df", THEN "≡E"(1), THEN "&E"(1)],
    THEN "RM◇"]
```

```
private lemmas eqnotnec_123_Aux_ξ' =
  eqnotnec_123_Aux_ϑ[
    THEN "conventions:3"[THEN "≡Df", THEN "≡E"(1), THEN "&E"(1)],
    THEN "RM◇"]

AOT_theorem "eqnotnec:1": ‹∃F∃G(∀x([F]x ≡ [G]x) & ◇¬∀x([F]x ≡ [G]x))›          (219.1)
proof-
  AOT_obtain p_1 where ‹ContingentlyTrue(p_1)›
    using "cont-tf-thm:1" "∃E"[rotated] by blast
  AOT_hence ‹p_1 & ◇¬p_1› using "cont-tf:1"[THEN "≡_dfE"] by blast
  AOT_hence ‹∀x ([L]x ≡ [λz p_1]x) & ◇¬∀x([L]x ≡ [λz p_1]x)›
    apply - apply (rule "&I")
    using "&E" eqnotnec_123_Aux_ϑ[THEN "≡E"(1)]
          eqnotnec_123_Aux_ξ "→E" by fast+
  AOT_hence ‹∃G (∀x([L]x ≡ [G]x) & ◇¬∀x([L]x ≡ [G]x))›
    by (rule "∃I") "cqt:2[lambda]"
  AOT_thus ‹∃F∃G (∀x([F]x ≡ [G]x) & ◇¬∀x([F]x ≡ [G]x))›
    apply (rule "∃I")
    by (rule "=_dfI"(2)[OF L_def]) "cqt:2[lambda]"+
qed

AOT_theorem "eqnotnec:2": ‹∃F∃G(¬∀x([F]x ≡ [G]x) & ◇∀x([F]x ≡ [G]x))›          (219.2)
proof-
  AOT_obtain p_1 where ‹ContingentlyFalse(p_1)›
    using "cont-tf-thm:2" "∃E"[rotated] by blast
  AOT_hence ‹¬p_1 & ◇p_1› using "cont-tf:2"[THEN "≡_dfE"] by blast
  AOT_hence ‹¬∀x ([L]x ≡ [λz p_1]x) & ◇∀x([L]x ≡ [λz p_1]x)›
    apply - apply (rule "&I")
    using eqnotnec_123_Aux_ϑ[THEN "oth-class-taut:4:b"[THEN "≡E"(1)],
                                     THEN "≡E"(1)]
          "&E" eqnotnec_123_Aux_ξ' "→E" by fast+
  AOT_hence ‹∃G (¬∀x([L]x ≡ [G]x) & ◇∀x([L]x ≡ [G]x))›
    by (rule "∃I") "cqt:2[lambda]"
  AOT_thus ‹∃F∃G (¬∀x([F]x ≡ [G]x) & ◇∀x([F]x ≡ [G]x))›
    apply (rule "∃I")
    by (rule "=_dfI"(2)[OF L_def]) "cqt:2[lambda]"+
qed

AOT_theorem "eqnotnec:3": ‹∃F∃G(𝒜¬∀x([F]x ≡ [G]x) & ◇∀x([F]x ≡ [G]x))›          (219.3)
proof-
  AOT_have ‹¬𝒜q_0›
    apply (rule "=_dfI"(2)[OF q_0_def])
     apply (fact "log-prop-prop:2")
    by (fact AOT)
  AOT_hence ‹𝒜¬q_0›
    using "logic-actual-nec:1"[axiom_inst, THEN "≡E"(2)] by blast
  AOT_hence ‹𝒜¬∀x ([L]x ≡ [λz q_0]x)›
    using eqnotnec_123_Aux_ϑ[THEN "oth-class-taut:4:b"[THEN "≡E"(1)],
            THEN "conventions:3"[THEN "≡Df", THEN "≡E"(1), THEN "&E"(1)],
            THEN "RA[2]", THEN "act-cond"[THEN "→E"], THEN "→E"] by blast
  moreover AOT_have ‹◇∀x ([L]x ≡ [λz q_0]x)›
    using eqnotnec_123_Aux_ξ'[THEN "→E"] q_0_prop[THEN "&E"(1)] by blast
  ultimately AOT_have ‹𝒜¬∀x ([L]x ≡ [λz q_0]x) & ◇∀x ([L]x ≡ [λz q_0]x)›
    using "&I" by blast
  AOT_hence ‹∃G (𝒜¬∀x([L]x ≡ [G]x) & ◇∀x([L]x ≡ [G]x))›
    by (rule "∃I") "cqt:2[lambda]"
  AOT_thus ‹∃F∃G (𝒜¬∀x([F]x ≡ [G]x) & ◇∀x([F]x ≡ [G]x))›
    apply (rule "∃I")
    by (rule "=_dfI"(2)[OF L_def]) "cqt:2[lambda]"+
qed

end
```

```
AOT_theorem "eqnotnec:4": ‹∀F∃G(∀x([F]x ≡ [G]x) & ◇¬∀x([F]x ≡ [G]x))›        (219.4)
proof(rule GEN)
  fix F
  AOT_have Aux_A: ‹⊢□ φ → ∀x([F]x ≡ [λz [F]z & φ]x)› for φ
  proof(rule "→I"; rule GEN)
    AOT_modally_strict {
    fix x
    AOT_assume 0: ‹φ›
    AOT_have ‹[λz [F]z & φ]x ≡ [F]x & φ›
      by (rule "beta-C-meta"[THEN "→E"]) "cqt:2[lambda]"
    also AOT_have ‹... ≡ [F]x›
      apply (rule "≡I"; rule "→I")
      using "∨E"(3)[rotated, OF "useful-tautologies:2"[THEN "→E"], OF 0] "&E"
      apply blast
      using 0 "&I" by blast
    finally AOT_show ‹[F]x ≡ [λz [F]z & φ]x›
      using "Commutativity of ≡"[THEN "≡E"(1)] by blast
    }
  qed

  AOT_have Aux_B: ‹⊢□ φ → ∀x([F]x ≡ [λz [F]z & φ ∨ ¬φ]x)› for φ
  proof (rule "→I"; rule GEN)
    AOT_modally_strict {
      fix x
      AOT_assume 0: ‹φ›
      AOT_have ‹[λz ([F]z & φ) ∨ ¬φ]x ≡ (([F]x & φ) ∨ ¬φ)›
        by (rule "beta-C-meta"[THEN "→E"]) "cqt:2[lambda]"
      also AOT_have ‹... ≡ [F]x›
        apply (rule "≡I"; rule "→I")
        using "∨E"(3)[rotated, OF "useful-tautologies:2"[THEN "→E"], OF 0]
              "&E"
         apply blast
        apply (rule "∨I"(1)) using 0 "&I" by blast
      finally AOT_show ‹[F]x ≡ [λz ([F]z & φ) ∨ ¬φ]x›
        using "Commutativity of ≡"[THEN "≡E"(1)] by blast
    }
  qed

  AOT_have Aux_C:
    ‹⊢□ ◇¬φ → ◇¬∀z([λz [F]z & φ]z ≡ [λz [F]z & φ ∨ ¬φ]z)› for φ
  proof(rule "RM◇"; rule "→I"; rule "raa-cor:2")
  AOT_modally_strict {
      AOT_assume 0: ‹¬φ›
      AOT_assume ‹∀z ([λz [F]z & φ]z ≡ [λz [F]z & φ ∨ ¬φ]z)›
      AOT_hence ‹[λz [F]z & φ]z ≡ [λz [F]z & φ ∨ ¬φ]z› for z
        using "∀E" by blast
      moreover AOT_have ‹[λz [F]z & φ]z ≡ [F]z & φ› for z
          by (rule "beta-C-meta"[THEN "→E"]) "cqt:2[lambda]"
      moreover AOT_have ‹[λz ([F]z & φ) ∨ ¬φ]z ≡ (([F]z & φ) ∨ ¬φ)› for z
        by (rule "beta-C-meta"[THEN "→E"]) "cqt:2[lambda]"
      ultimately AOT_have ‹[F]z & φ ≡ (([F]z & φ) ∨ ¬φ)› for z
        using "Commutativity of ≡"[THEN "≡E"(1)] "≡E"(5) by meson
      moreover AOT_have ‹(([F]z & φ) ∨ ¬φ)› for z using 0 "∨I" by blast
      ultimately AOT_have ‹φ› using "≡E" "&E" by metis
      AOT_thus ‹φ & ¬φ› using 0 "&I" by blast
    }
  qed

  AOT_have Aux_D: ‹□∀z ([F]z ≡ [λz [F]z & φ]z) →
      (◇¬∀x ([λz [F]z & φ]x ≡ [λz [F]z & φ ∨ ¬φ]x) ≡
       ◇¬∀x ([F]x ≡ [λz [F]z & φ ∨ ¬φ]x))› for φ
  proof (rule "→I")
    AOT_assume A: ‹□∀z([F]z ≡ [λz [F]z & φ]z)›
```

```
    AOT_show ‹◇¬∀x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x) ≡
             ◇¬∀x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
    proof(rule "≡I"; rule "KBasic:13"[THEN "→E"];
          rule "RN[prem]"[where Γ="{«∀z([F]z ≡ [λz [F]z & ψ]z)»}", simplified];
          (rule "useful-tautologies:5"[THEN "→E"]; rule "→I")?)
      AOT_modally_strict {
        AOT_assume ‹∀z ([F]z ≡ [λz [F]z & ψ]z)›
        AOT_hence 1: ‹[F]z ≡ [λz [F]z & ψ]z› for z
          using "∀E" by blast
        AOT_assume ‹∀x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
        AOT_hence 2: ‹[F]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
          using "∀E" by blast
        AOT_have ‹[λz [F]z & ψ]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
          using "≡E" 1 2 by meson
        AOT_thus ‹∀x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
          by (rule GEN)
      }
    next
      AOT_modally_strict {
        AOT_assume ‹∀z ([F]z ≡ [λz [F]z & ψ]z)›
        AOT_hence 1: ‹[F]z ≡ [λz [F]z & ψ]z› for z
          using "∀E" by blast
        AOT_assume ‹∀x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
        AOT_hence 2: ‹[λz [F]z & ψ]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
          using "∀E" by blast
        AOT_have ‹[F]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
          using 1 2 "≡E" by meson
        AOT_thus ‹ ∀x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
          by (rule GEN)
      }
    qed(auto simp: A)
  qed

  AOT_obtain p₁ where p₁_prop: ‹p₁ & ◇¬p₁›
    using "cont-tf-thm:1" "∃E"[rotated]
          "cont-tf:1"[THEN "≡dfE"] by blast
  {
    AOT_assume 1: ‹□∀x([F]x ≡ [λz [F]z & p₁]x)›
    AOT_have 2: ‹∀x([F]x ≡ [λz [F]z & p₁ ∨ ¬p₁]x)›
      using Aux_B[THEN "→E", OF p₁_prop[THEN "&E"(1)]].
    AOT_have ‹◇¬∀x([λz [F]z & p₁]x ≡ [λz [F]z & p₁ ∨ ¬p₁]x)›
      using Aux_C[THEN "→E", OF p₁_prop[THEN "&E"(2)]].
    AOT_hence 3: ‹◇¬∀x([F]x ≡ [λz [F]z & p₁ ∨ ¬p₁]x)›
      using Aux_D[THEN "→E", OF 1, THEN "≡E"(1)] by blast
    AOT_hence ‹∀x([F]x ≡ [λz [F]z & p₁ ∨ ¬p₁]x) &
               ◇¬∀x([F]x ≡ [λz [F]z & p₁ ∨ ¬p₁]x)›
      using 2 "&I" by blast
    AOT_hence ‹∃G (∀x ([F]x ≡ [G]x) & ◇¬∀x([F]x ≡ [G]x))›
      by (rule "∃I"(1)) "cqt:2[lambda]"
  }
  moreover {
    AOT_assume 2: ‹¬□∀x([F]x ≡ [λz [F]z & p₁]x)›
    AOT_hence ‹◇¬∀x([F]x ≡ [λz [F]z & p₁]x)›
      using "KBasic:11"[THEN "≡E"(1)] by blast
    AOT_hence ‹∀x ([F]x ≡ [λz [F]z & p₁]x) & ◇¬∀x([F]x ≡ [λz [F]z & p₁]x)›
      using Aux_A[THEN "→E", OF p₁_prop[THEN "&E"(1)]] "&I" by blast
    AOT_hence ‹∃G (∀x ([F]x ≡ [G]x) & ◇¬∀x([F]x ≡ [G]x))›
      by (rule "∃I"(1)) "cqt:2[lambda]"
  }
  ultimately AOT_show ‹∃G (∀x ([F]x ≡ [G]x) & ◇¬∀x([F]x ≡ [G]x))›
    using "∨E"(1)[OF "exc-mid"] "→I" by blast
qed
```

```
AOT_theorem "eqnotnec:5": ‹∀F∃G(¬∀x([F]x ≡ [G]x) & ◇∀x([F]x ≡ [G]x))›          (219.5)
proof(rule GEN)
  fix F
  AOT_have Aux_A: ‹⊢□ ◇ψ → ◇∀x([F]x ≡ [λz [F]z & ψ]x)› for ψ
  proof(rule "RM◇"; rule "→I"; rule GEN)
    AOT_modally_strict {
    fix x
    AOT_assume 0: ‹ψ›
    AOT_have ‹[λz [F]z & ψ]x ≡ [F]x & ψ›
      by (rule "beta-C-meta"[THEN "→E"]) "cqt:2[lambda]"
    also AOT_have ‹... ≡ [F]x›
      apply (rule "≡I"; rule "→I")
      using "∨E"(3)[rotated, OF "useful-tautologies:2"[THEN "→E"], OF 0] "&E"
       apply blast
      using 0 "&I" by blast
    finally AOT_show ‹[F]x ≡ [λz [F]z & ψ]x›
      using "Commutativity of ≡"[THEN "≡E"(1)] by blast
    }
  qed

  AOT_have Aux_B: ‹⊢□ ◇ψ → ◇∀x([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)› for ψ
  proof (rule "RM◇"; rule "→I"; rule GEN)
    AOT_modally_strict {
      fix x
      AOT_assume 0: ‹ψ›
      AOT_have ‹[λz ([F]z & ψ) ∨ ¬ψ]x ≡ (([F]x & ψ) ∨ ¬ψ)›
        by (rule "beta-C-meta"[THEN "→E"]) "cqt:2[lambda]"
      also AOT_have ‹... ≡ [F]x›
        apply (rule "≡I"; rule "→I")
        using "∨E"(3)[rotated, OF "useful-tautologies:2"[THEN "→E"], OF 0] "&E"
         apply blast
        apply (rule "∨I"(1)) using 0 "&I" by blast
      finally AOT_show ‹[F]x ≡ [λz ([F]z & ψ) ∨ ¬ψ]x›
        using "Commutativity of ≡"[THEN "≡E"(1)] by blast
    }
  qed

  AOT_have Aux_C: ‹⊢□ ¬ψ → ¬∀z([λz [F]z & ψ]z ≡ [λz [F]z & ψ ∨ ¬ψ]z)› for ψ
  proof(rule "→I"; rule "raa-cor:2")
  AOT_modally_strict {
      AOT_assume 0: ‹¬ψ›
      AOT_assume ‹∀z ([λz [F]z & ψ]z ≡ [λz [F]z & ψ ∨ ¬ψ]z)›
      AOT_hence ‹[λz [F]z & ψ]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
        using "∀E" by blast
      moreover AOT_have ‹[λz [F]z & ψ]z ≡ [F]z & ψ› for z
          by (rule "beta-C-meta"[THEN "→E"]) "cqt:2[lambda]"
      moreover AOT_have ‹[λz ([F]z & ψ) ∨ ¬ψ]z ≡ (([F]z & ψ) ∨ ¬ψ)› for z
        by (rule "beta-C-meta"[THEN "→E"]) "cqt:2[lambda]"
      ultimately AOT_have ‹[F]z & ψ ≡ (([F]z & ψ) ∨ ¬ψ)› for z
        using "Commutativity of ≡"[THEN "≡E"(1)] "≡E"(5) by meson
      moreover AOT_have ‹(([F]z & ψ) ∨ ¬ψ)› for z
        using 0 "∨I" by blast
      ultimately AOT_have ‹ψ› using "≡E" "&E" by metis
      AOT_thus ‹ψ & ¬ψ› using 0 "&I" by blast
    }
  qed

  AOT_have Aux_D: ‹∀z ([F]z ≡ [λz [F]z & ψ]z) →
    (¬∀x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x) ≡
     ¬∀x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x))› for ψ
  proof (rule "→I"; rule "≡I";
          (rule "useful-tautologies:5"[THEN "→E"]; rule "→I")?)
    AOT_modally_strict {
```

```
      AOT_assume ‹∀z ([F]z ≡ [λz [F]z & ψ]z)›
      AOT_hence 1: ‹[F]z ≡ [λz [F]z & ψ]z› for z
        using "∀E" by blast
      AOT_assume ‹∀x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
      AOT_hence 2: ‹[F]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
        using "∀E" by blast
      AOT_have ‹[λz [F]z & ψ]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
        using "≡E" 1 2 by meson
      AOT_thus ‹∀x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
        by (rule GEN)
    }
  next
    AOT_modally_strict {
      AOT_assume ‹∀z ([F]z ≡ [λz [F]z & ψ]z)›
      AOT_hence 1: ‹[F]z ≡ [λz [F]z & ψ]z› for z
        using "∀E" by blast
      AOT_assume ‹∀x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
      AOT_hence 2: ‹[λz [F]z & ψ]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
        using "∀E" by blast
      AOT_have ‹[F]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
        using 1 2 "≡E" by meson
      AOT_thus ‹ ∀x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
        by (rule GEN)
    }
  qed

  AOT_obtain p₁ where p₁_prop: ‹¬p₁ & ◊p₁›
    using "cont-tf-thm:2" "∃E"[rotated] "cont-tf:2"[THEN "≡dfE"] by blast
  {
    AOT_assume 1: ‹∀x([F]x ≡ [λz [F]z & p₁]x)›
    AOT_have 2: ‹◊∀x([F]x ≡ [λz [F]z & p₁ ∨ ¬p₁]x)›
      using Aux_B[THEN "→E", OF p₁_prop[THEN "&E"(2)]].
    AOT_have ‹¬∀x([λz [F]z & p₁]x ≡ [λz [F]z & p₁ ∨ ¬p₁]x)›
      using Aux_C[THEN "→E", OF p₁_prop[THEN "&E"(1)]].
    AOT_hence 3: ‹¬∀x([F]x ≡ [λz [F]z & p₁ ∨ ¬p₁]x)›
      using Aux_D[THEN "→E", OF 1, THEN "≡E"(1)] by blast
    AOT_hence ‹¬∀x([F]x ≡ [λz [F]z & p₁ ∨ ¬p₁]x) &
               ◊∀x([F]x ≡ [λz [F]z & p₁ ∨ ¬p₁]x)›
      using 2 "&I" by blast
    AOT_hence ‹∃G (¬∀x ([F]x ≡ [G]x) & ◊∀x([F]x ≡ [G]x))›
      by (rule "∃I"(1)) "cqt:2[lambda]"
  }
  moreover {
    AOT_assume 2: ‹¬∀x([F]x ≡ [λz [F]z & p₁]x)›
    AOT_hence ‹¬∀x([F]x ≡ [λz [F]z & p₁]x)›
      using "KBasic:11"[THEN "≡E"(1)] by blast
    AOT_hence ‹¬∀x ([F]x ≡ [λz [F]z & p₁]x) &
               ◊∀x([F]x ≡ [λz [F]z & p₁]x)›
      using Aux_A[THEN "→E", OF p₁_prop[THEN "&E"(2)]] "&I" by blast
    AOT_hence ‹∃G (¬∀x ([F]x ≡ [G]x) & ◊∀x([F]x ≡ [G]x))›
      by (rule "∃I"(1)) "cqt:2[lambda]"
  }
  ultimately AOT_show ‹∃G (¬∀x ([F]x ≡ [G]x) & ◊∀x([F]x ≡ [G]x))›
    using "∨E"(1)[OF "exc-mid"] "→I" by blast
qed

AOT_theorem "eqnotnec:6": ‹∀F∃G(𝒜¬∀x([F]x ≡ [G]x) & ◊∀x([F]x ≡ [G]x))›          (219.6)
proof(rule GEN)
  fix F
  AOT_have Aux_A: ‹⊢□ ◊ψ → ◊∀x([F]x ≡ [λz [F]z & ψ]x)› for ψ
  proof(rule "RM◊"; rule "→I"; rule GEN)
    AOT_modally_strict {
    fix x
```

```
  AOT_assume 0: ‹ψ›
  AOT_have ‹[λz [F]z & ψ]x ≡ [F]x & ψ›
    by (rule "beta-C-meta"[THEN "→E"]) "cqt:2[lambda]"
  also AOT_have ‹... ≡ [F]x›
    apply (rule "≡I"; rule "→I")
    using "∨E"(3)[rotated, OF "useful-tautologies:2"[THEN "→E"], OF 0]
          "&E"
     apply blast
    using 0 "&I" by blast
  finally AOT_show ‹[F]x ≡ [λz [F]z & ψ]x›
    using "Commutativity of ≡"[THEN "≡E"(1)] by blast
  }
qed

AOT_have Aux_B: ‹⊢□ ◇ψ → ◇∀x([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)› for ψ
proof (rule "RM◇"; rule "→I"; rule GEN)
  AOT_modally_strict {
    fix x
    AOT_assume 0: ‹ψ›
    AOT_have ‹[λz ([F]z & ψ) ∨ ¬ψ]x ≡ (([F]x & ψ) ∨ ¬ψ)›
      by (rule "beta-C-meta"[THEN "→E"]) "cqt:2[lambda]"
    also AOT_have ‹... ≡ [F]x›
      apply (rule "≡I"; rule "→I")
      using "∨E"(3)[rotated, OF "useful-tautologies:2"[THEN "→E"], OF 0] "&E"
       apply blast
      apply (rule "∨I"(1)) using 0 "&I" by blast
    finally AOT_show ‹[F]x ≡ [λz ([F]z & ψ) ∨ ¬ψ]x›
      using "Commutativity of ≡"[THEN "≡E"(1)] by blast
  }
qed

AOT_have Aux_C:
  ‹⊢□ 𝒜¬ψ → 𝒜¬∀z([λz [F]z & ψ]z ≡ [λz [F]z & ψ ∨ ¬ψ]z)› for ψ
proof(rule "act-cond"[THEN "→E"]; rule "RA[2]"; rule "→I"; rule "raa-cor:2")
AOT_modally_strict {
    AOT_assume 0: ‹¬ψ›
    AOT_assume ‹∀z ([λz [F]z & ψ]z ≡ [λz [F]z & ψ ∨ ¬ψ]z)›
    AOT_hence ‹[λz [F]z & ψ]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
      using "∀E" by blast
    moreover AOT_have ‹[λz [F]z & ψ]z ≡ [F]z & ψ› for z
        by (rule "beta-C-meta"[THEN "→E"]) "cqt:2[lambda]"
    moreover AOT_have ‹[λz ([F]z & ψ) ∨ ¬ψ]z ≡ (([F]z & ψ) ∨ ¬ψ)› for z
      by (rule "beta-C-meta"[THEN "→E"]) "cqt:2[lambda]"
    ultimately AOT_have ‹[F]z & ψ ≡ (([F]z & ψ) ∨ ¬ψ)› for z
      using "Commutativity of ≡"[THEN "≡E"(1)] "≡E"(5) by meson
    moreover AOT_have ‹(([F]z & ψ) ∨ ¬ψ)› for z
      using 0 "∨I" by blast
    ultimately AOT_have ‹ψ› using "≡E" "&E" by metis
    AOT_thus ‹ψ & ¬ψ› using 0 "&I" by blast
  }
qed

AOT_have ‹□(∀z ([F]z ≡ [λz [F]z & ψ]z) →
  (¬∀x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x) ≡
   ¬∀x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)))› for ψ
proof (rule RN; rule "→I")
  AOT_modally_strict {
      AOT_assume ‹∀z ([F]z ≡ [λz [F]z & ψ]z)›
      AOT_thus ‹¬∀x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x) ≡
                ¬∀x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
        apply -
      proof(rule "≡I"; (rule "useful-tautologies:5"[THEN "→E"]; rule "→I")?)
        AOT_assume ‹∀z ([F]z ≡ [λz [F]z & ψ]z)›
```

```
          AOT_hence 1: ‹[F]z ≡ [λz [F]z & ψ]z› for z
            using "∀E" by blast
          AOT_assume ‹∀x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
          AOT_hence 2: ‹[F]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
            using "∀E" by blast
          AOT_have ‹[λz [F]z & ψ]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
            using "≡E" 1 2 by meson
          AOT_thus ‹∀x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
            by (rule GEN)
        next
          AOT_assume ‹∀z ([F]z ≡ [λz [F]z & ψ]z)›
          AOT_hence 1: ‹[F]z ≡ [λz [F]z & ψ]z› for z
            using "∀E" by blast
          AOT_assume ‹∀x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
          AOT_hence 2: ‹[λz [F]z & ψ]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
            using "∀E" by blast
          AOT_have ‹[F]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
            using 1 2 "≡E" by meson
          AOT_thus ‹ ∀x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
            by (rule GEN)
        qed
    }
  qed
  AOT_hence ‹𝓐(∀z ([F]z ≡ [λz [F]z & ψ]z) →
    (¬∀x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x) ≡
     ¬∀x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)))› for ψ
    using "nec-imp-act"[THEN "→E"] by blast
  AOT_hence ‹𝓐∀z ([F]z ≡ [λz [F]z & ψ]z) →
    𝓐(¬∀x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x) ≡
    ¬∀x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x))› for ψ
    using "act-cond"[THEN "→E"] by blast
  AOT_hence Aux_D: ‹𝓐∀z ([F]z ≡ [λz [F]z & ψ]z) →
    (𝓐¬∀x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x) ≡
     𝓐¬∀x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x))› for ψ
    by (auto intro!: "→I" "Act-Basic:5"[THEN "≡E"(1)] dest!: "→E")

  AOT_have ‹¬𝓐q_0›
    apply (rule "=_dfI"(2)[OF q_0_def])
     apply (fact "log-prop-prop:2")
    by (fact AOT)
  AOT_hence q_0_prop_1: ‹𝓐¬q_0›
    using "logic-actual-nec:1"[axiom_inst, THEN "≡E"(2)] by blast
  {
    AOT_assume 1: ‹𝓐∀x([F]x ≡ [λz [F]z & q_0]x)›
    AOT_have 2: ‹◊∀x([F]x ≡ [λz [F]z & q_0 ∨ ¬q_0]x)›
      using Aux_B[THEN "→E", OF q_0_prop[THEN "&E"(1)]].
    AOT_have ‹𝓐¬∀x([λz [F]z & q_0]x ≡ [λz [F]z & q_0 ∨ ¬q_0]x)›
      using Aux_C[THEN "→E", OF q_0_prop_1].
    AOT_hence 3: ‹𝓐¬∀x([F]x ≡ [λz [F]z & q_0 ∨ ¬q_0]x)›
      using Aux_D[THEN "→E", OF 1, THEN "≡E"(1)] by blast
    AOT_hence ‹𝓐¬∀x([F]x ≡ [λz [F]z & q_0 ∨ ¬q_0]x) &
                ◊∀x([F]x ≡ [λz [F]z & q_0 ∨ ¬q_0]x)›
      using 2 "&I" by blast
    AOT_hence ‹∃G (𝓐¬∀x ([F]x ≡ [G]x) & ◊∀x([F]x ≡ [G]x))›
      by (rule "∃I"(1)) "cqt:2[lambda]"
  }
  moreover {
    AOT_assume 2: ‹¬𝓐∀x([F]x ≡ [λz [F]z & q_0]x)›
    AOT_hence ‹𝓐¬∀x([F]x ≡ [λz [F]z & q_0]x)›
      using "logic-actual-nec:1"[axiom_inst, THEN "≡E"(2)] by blast
    AOT_hence ‹𝓐¬∀x ([F]x ≡ [λz [F]z & q_0]x) & ◊∀x([F]x ≡ [λz [F]z & q_0]x)›
      using Aux_A[THEN "→E", OF q_0_prop[THEN "&E"(1)]] "&I" by blast
    AOT_hence ‹∃G (𝓐¬∀x ([F]x ≡ [G]x) & ◊∀x([F]x ≡ [G]x))›
```

```
      by (rule "∃I"(1)) "cqt:2[lambda]"
  }
  ultimately AOT_show ‹∃G (𝒜¬∀x ([F]x ≡ [G]x) & ◊∀x([F]x ≡ [G]x))›
    using "∨E"(1)[OF "exc-mid"] "→I" by blast
qed

AOT_theorem "oa-contingent:1": ‹O! ≠ A!›                                  (220.1)
proof(rule "≡dfI"[OF "=-infix"]; rule "raa-cor:2")
  fix x
  AOT_assume 1: ‹O! = A!›
  AOT_hence ‹[λx ◊E!x] = A!›
    by (rule "=dfE"(2)[OF AOT_ordinary, rotated]) "cqt:2[lambda]"
  AOT_hence ‹[λx ◊E!x] = [λx ¬◊E!x]›
    by (rule "=dfE"(2)[OF AOT_abstract, rotated]) "cqt:2[lambda]"
  moreover AOT_have ‹[λx ◊E!x]x ≡ ◊E!x›
    by (rule "beta-C-meta"[THEN "→E"]) "cqt:2[lambda]"
  ultimately AOT_have ‹[λx ¬◊E!x]x ≡ ◊E!x›
    using "rule=E" by fast
  moreover AOT_have ‹[λx ¬◊E!x]x ≡ ¬◊E!x›
    by (rule "beta-C-meta"[THEN "→E"]) "cqt:2[lambda]"
  ultimately AOT_have ‹◊E!x ≡ ¬◊E!x›
    using "≡E"(6) "Commutativity of ≡"[THEN "≡E"(1)] by blast
  AOT_thus "(◊E!x ≡ ¬◊E!x) & ¬(◊E!x ≡ ¬◊E!x)"
    using "oth-class-taut:3:c" "&I" by blast
qed

AOT_theorem "oa-contingent:2": ‹O!x ≡ ¬A!x›                                (220.2)
proof -
  AOT_have ‹O!x ≡ [λx ◊E!x]x›
    apply (rule "≡I"; rule "→I")
     apply (rule "=dfE"(2)[OF AOT_ordinary])
      apply "cqt:2[lambda]"
     apply argo
    apply (rule  "=dfI"(2)[OF AOT_ordinary])
     apply "cqt:2[lambda]"
    by argo
  also AOT_have ‹... ≡ ◊E!x›
    by (rule "beta-C-meta"[THEN "→E"]) "cqt:2[lambda]"
  also AOT_have ‹... ≡ ¬¬◊E!x›
    using "oth-class-taut:3:b".
  also AOT_have ‹... ≡ ¬[λx ¬◊E!x]x›
    by (rule "beta-C-meta"[THEN "→E",
               THEN "oth-class-taut:4:b"[THEN "≡E"(1)], symmetric])
        "cqt:2"
  also AOT_have ‹... ≡ ¬A!x›
    apply (rule "≡I"; rule "→I")
     apply (rule "=dfI"(2)[OF AOT_abstract])
      apply "cqt:2[lambda]"
     apply argo
    apply (rule "=dfE"(2)[OF AOT_abstract])
     apply "cqt:2[lambda]"
    by argo
  finally show ?thesis.
qed

AOT_theorem "oa-contingent:3": ‹A!x ≡ ¬O!x›                                (220.3)
  by (AOT_subst ‹A!x› ‹¬¬A!x›)
     (auto simp add: "oth-class-taut:3:b" "oa-contingent:2"[THEN
         "oth-class-taut:4:b"[THEN "≡E"(1)], symmetric])

AOT_theorem "oa-contingent:4": ‹Contingent(O!)›                            (220.4)
proof (rule "thm-cont-prop:2"[unvarify F, OF "oa-exist:1", THEN "≡E"(2)];
       rule "&I")
```

```
  AOT_have ‹◇∃x E!x› using "thm-cont-e:3" .
  AOT_hence ‹∃x ◇E!x› using "BF◇"[THEN "→E"] by blast
  then AOT_obtain a where ‹◇E!a› using "∃E"[rotated] by blast
  AOT_hence ‹[λx ◇E!x]a›
    by (rule "beta-C-meta"[THEN "→E", THEN "≡E"(2), rotated]) "cqt:2"
  AOT_hence ‹O!a›
    by (rule "=dfI"(2)[OF AOT_ordinary, rotated]) "cqt:2"
  AOT_hence ‹∃x O!x› using "∃I" by blast
  AOT_thus ‹◇∃x O!x› using "T◇"[THEN "→E"] by blast
next
  AOT_obtain a where ‹A!a›
    using "A-objects"[axiom_inst] "∃E"[rotated] "&E" by blast
  AOT_hence ‹¬O!a› using "oa-contingent:3"[THEN "≡E"(1)] by blast
  AOT_hence ‹∃x ¬O!x› using "∃I" by fast
  AOT_thus ‹◇∃x ¬O!x› using "T◇"[THEN "→E"] by blast
qed

AOT_theorem "oa-contingent:5": ‹Contingent(A!)›                    (220.5)
proof (rule "thm-cont-prop:2"[unvarify F, OF "oa-exist:2", THEN "≡E"(2)];
       rule "&I")
  AOT_obtain a where ‹A!a›
    using "A-objects"[axiom_inst] "∃E"[rotated] "&E" by blast
  AOT_hence ‹∃x A!x› using "∃I" by fast
  AOT_thus ‹◇∃x A!x› using "T◇"[THEN "→E"] by blast
next
  AOT_have ‹◇∃x E!x› using "thm-cont-e:3" .
  AOT_hence ‹∃x ◇E!x› using "BF◇"[THEN "→E"] by blast
  then AOT_obtain a where ‹◇E!a› using "∃E"[rotated] by blast
  AOT_hence ‹[λx ◇E!x]a›
    by (rule "beta-C-meta"[THEN "→E", THEN "≡E"(2), rotated]) "cqt:2[lambda]"
  AOT_hence ‹O!a›
    by (rule "=dfI"(2)[OF AOT_ordinary, rotated]) "cqt:2[lambda]"
  AOT_hence ‹¬A!a› using "oa-contingent:2"[THEN "≡E"(1)] by blast
  AOT_hence ‹∃x ¬A!x› using "∃I" by fast
  AOT_thus ‹◇∃x ¬A!x› using "T◇"[THEN "→E"] by blast
qed

AOT_theorem "oa-contingent:7": ‹O!¯x ≡ ¬A!¯x›                    (220.7)
proof -
  AOT_have ‹O!x ≡ ¬A!x›
    using "oa-contingent:2" by blast
  also AOT_have ‹... ≡ A!¯x›
    using "thm-relation-negation:1"[symmetric, unvarify F, OF "oa-exist:2"].
  finally AOT_have 1: ‹O!x ≡ A!¯x›.

  AOT_have ‹A!x ≡ ¬O!x›
    using "oa-contingent:3" by blast
  also AOT_have ‹... ≡ O!¯x›
    using "thm-relation-negation:1"[symmetric, unvarify F, OF "oa-exist:1"].
  finally AOT_have 2: ‹A!x ≡ O!¯x›.

  AOT_show ‹O!¯x ≡ ¬A!¯x›
    using 1[THEN "oth-class-taut:4:b"[THEN "≡E"(1)]]
          "oa-contingent:3"[of _ x] 2[symmetric]
          "≡E"(5) by blast
qed

AOT_theorem "oa-contingent:6": ‹O!¯ ≠ A!¯›                    (220.6)
proof (rule "=-infix"[THEN "≡dfI"]; rule "raa-cor:2")
  AOT_assume 1: ‹O!¯ = A!¯›
  fix x
  AOT_have ‹A!¯x ≡ O!¯x›
    apply (rule "rule=E"[rotated, OF 1])
```

```
    by (fact "oth-class-taut:3:a")
  AOT_hence ‹A!¯x ≡ ¬A!¯x›
    using "oa-contingent:7" "≡E" by fast
  AOT_thus ‹(A!¯x ≡ ¬A!¯x) & ¬(A!¯x ≡ ¬A!¯x)›
    using "oth-class-taut:3:c" "&I" by blast
qed

AOT_theorem "oa-contingent:8": ‹Contingent(O!¯)›                    (220.8)
  using "thm-cont-prop:3"[unvarify F, OF "oa-exist:1", THEN "≡E"(1),
          OF "oa-contingent:4"].

AOT_theorem "oa-contingent:9": ‹Contingent(A!¯)›                    (220.9)
  using "thm-cont-prop:3"[unvarify F, OF "oa-exist:2", THEN "≡E"(1),
          OF "oa-contingent:5"].

AOT_define WeaklyContingent :: ‹Π ⇒ φ› (‹WeaklyContingent'(_')›)
  "df-cont-nec":                                                    (221)
  ‹WeaklyContingent([F]) ≡df Contingent([F]) & ∀x (◇[F]x → □[F]x)›

AOT_theorem "cont-nec-fact1:1":                                     (222.1)
  ‹WeaklyContingent([F]) ≡ WeaklyContingent([F]¯)›
proof -
  AOT_have ‹WeaklyContingent([F]) ≡ Contingent([F]) & ∀x (◇[F]x → □[F]x)›
    using "df-cont-nec"[THEN "≡Df"] by blast
  also AOT_have ‹... ≡ Contingent([F]¯) & ∀x (◇[F]x → □[F]x)›
    apply (rule "oth-class-taut:8:f"[THEN "≡E"(2)]; rule "→I")
    using "thm-cont-prop:3".
  also AOT_have ‹... ≡ Contingent([F]¯) & ∀x (◇[F]¯x → □[F]¯x)›
  proof (rule "oth-class-taut:8:e"[THEN "≡E"(2)];
          rule "→I"; rule "≡I"; rule "→I"; rule GEN; rule "→I")
    fix x
    AOT_assume 0: ‹∀x (◇[F]x → □[F]x)›
    AOT_assume 1: ‹◇[F]¯x›
    AOT_have ‹◇¬[F]x›
      by (AOT_subst (reverse) ‹¬[F]x› ‹[F]¯x›)
         (auto simp add: "thm-relation-negation:1" 1)
    AOT_hence 2: ‹¬□[F]x›
      using "KBasic:11"[THEN "≡E"(2)] by blast
    AOT_show ‹□[F]¯x›
    proof (rule "raa-cor:1")
      AOT_assume 3: ‹¬□[F]¯x›
      AOT_have ‹¬□¬[F]x›
        by (AOT_subst (reverse) ‹¬[F]x› ‹[F]¯x›)
           (auto simp add: "thm-relation-negation:1" 3)
      AOT_hence ‹◇[F]x›
        using "conventions:5"[THEN "≡dfI"] by simp
      AOT_hence ‹□[F]x› using 0 "∀E" "→E" by fast
      AOT_thus ‹□[F]x & ¬□[F]x› using "&I" 2 by blast
    qed
  next
    fix x
    AOT_assume 0: ‹∀x (◇[F]¯x → □[F]¯x)›
    AOT_assume 1: ‹◇[F]x›
    AOT_have ‹◇¬[F]¯x›
      by (AOT_subst ‹¬[F]¯x› ‹[F]x›)
         (auto simp: "thm-relation-negation:2" 1)
    AOT_hence 2: ‹¬□[F]¯x›
      using "KBasic:11"[THEN "≡E"(2)] by blast
    AOT_show ‹□[F]x›
    proof (rule "raa-cor:1")
      AOT_assume 3: ‹¬□[F]x›
      AOT_have ‹¬□¬[F]¯x›
        by (AOT_subst ‹¬[F]¯x› ‹[F]x›)
```

```
          (auto simp add: "thm-relation-negation:2" 3)
      AOT_hence ‹◇[F]⁻x›
        using "conventions:5"[THEN "≡dfI"] by simp
      AOT_hence ‹□[F]⁻x› using 0 "∀E" "→E" by fast
      AOT_thus ‹□[F]⁻x & ¬□[F]⁻x› using "&I" 2 by blast
    qed
  qed
  also AOT_have ‹... ≡ WeaklyContingent([F]⁻)›
    using "df-cont-nec"[THEN "≡Df", symmetric] by blast
  finally show ?thesis.
qed

AOT_theorem "cont-nec-fact1:2":                                        (222.2)
  ‹(WeaklyContingent([F]) & ¬WeaklyContingent([G])) → F ≠ G›
proof (rule "→I"; rule "=-infix"[THEN "≡dfI"]; rule "raa-cor:2")
  AOT_assume 1: ‹WeaklyContingent([F]) & ¬WeaklyContingent([G])›
  AOT_hence ‹WeaklyContingent([F])› using "&E" by blast
  moreover AOT_assume ‹F = G›
  ultimately AOT_have ‹WeaklyContingent([G])›
    using "rule=E" by blast
  AOT_thus ‹WeaklyContingent([G]) & ¬WeaklyContingent([G])›
    using 1 "&I" "&E" by blast
qed

AOT_theorem "cont-nec-fact2:1": ‹WeaklyContingent(O!)›                 (223.1)
proof (rule "df-cont-nec"[THEN "≡dfI"]; rule "&I")
  AOT_show ‹Contingent(O!)›
    using "oa-contingent:4".
next
  AOT_show ‹∀x (◇[O!]x → □[O!]x)›
    apply (rule GEN; rule "→I")
    using "oa-facts:5"[THEN "≡E"(1)] by blast
qed


AOT_theorem "cont-nec-fact2:2": ‹WeaklyContingent(A!)›                 (223.2)
proof (rule "df-cont-nec"[THEN "≡dfI"]; rule "&I")
  AOT_show ‹Contingent(A!)›
    using "oa-contingent:5".
next
  AOT_show ‹∀x (◇[A!]x → □[A!]x)›
    apply (rule GEN; rule "→I")
    using "oa-facts:6"[THEN "≡E"(1)] by blast
qed

AOT_theorem "cont-nec-fact2:3": ‹¬WeaklyContingent(E!)›                (223.3)
proof (rule "df-cont-nec"[THEN "≡Df",
                          THEN "oth-class-taut:4:b"[THEN "≡E"(1)],
                          THEN "≡E"(2)];
       rule DeMorgan(1)[THEN "≡E"(2)]; rule "∨I"(2); rule "raa-cor:2")
  AOT_have ‹◇∃x (E!x & ¬𝒜E!x)› using "qml:4"[axiom_inst].
  AOT_hence ‹∃x ◇(E!x & ¬𝒜E!x)› using "BF◇"[THEN "→E"] by blast
  then AOT_obtain a where ‹◇(E!a & ¬𝒜E!a)› using "∃E"[rotated] by blast
  AOT_hence 1: ‹◇E!a & ◇¬𝒜E!a› using "KBasic2:3"[THEN "→E"] by simp
  moreover AOT_assume ‹∀x (◇[E!]x → □[E!]x)›
  ultimately AOT_have ‹□E!a› using "&E" "∀E" "→E" by fast
  AOT_hence ‹𝒜E!a› using "nec-imp-act"[THEN "→E"] by blast
  AOT_hence ‹□𝒜E!a› using "qml-act:1"[axiom_inst, THEN "→E"] by blast
  moreover AOT_have ‹¬□𝒜E!a›
    using "KBasic:11"[THEN "≡E"(2)] 1[THEN "&E"(2)] by meson
  ultimately AOT_have ‹□𝒜E!a & ¬□𝒜E!a› using "&I" by blast
  AOT_thus ‹p & ¬p› for p using "raa-cor:1" by blast
qed
```

```
AOT_theorem "cont-nec-fact2:4": ‹¬WeaklyContingent(L)›                    (223.4)
  apply (rule "df-cont-nec"[THEN "≡Df",
                            THEN "oth-class-taut:4:b"[THEN "≡E"(1)],
                            THEN "≡E"(2)];
        rule DeMorgan(1)[THEN "≡E"(2)]; rule "∨I"(1))
  apply (rule "contingent-properties:4"
                 [THEN "≡Df",
                  THEN "oth-class-taut:4:b"[THEN "≡E"(1)],
                  THEN "≡E"(2)])
  apply (rule DeMorgan(1)[THEN "≡E"(2)];
         rule "∨I"(2);
         rule "useful-tautologies:2"[THEN "→E"])
  using "thm-noncont-e-e:3"[THEN "contingent-properties:3"[THEN "≡dfE"]].

AOT_theorem "cont-nec-fact2:5": ‹O! ≠ E! & O! ≠ E!⁻ & O! ≠ L & O! ≠ L⁻›    (223.5)
proof -
  AOT_have 1: ‹L↓›
    by (rule "=dfI"(2)[OF L_def]) "cqt:2[lambda]"+
  {
    fix φ and Π Π' :: ‹<κ>›
    AOT_have A: ‹¬(φ{Π'} ≡ φ{Π})› if  ‹φ{Π}› and ‹¬φ{Π'}›
    proof (rule "raa-cor:2")
      AOT_assume ‹φ{Π'} ≡ φ{Π}›
      AOT_hence ‹φ{Π'}› using that(1) "≡E" by blast
      AOT_thus ‹φ{Π'} & ¬φ{Π'}› using that(2) "&I" by blast
    qed
    AOT_have ‹Π' ≠ Π› if ‹Π↓› and ‹Π'↓› and ‹φ{Π}› and ‹¬φ{Π'}›
      using "pos-not-equiv-ne:4"[unvarify F G, THEN "→E",
                                 OF that(1,2), OF A[OF that(3, 4)]].
  } note 0 = this
  show ?thesis
    apply(safe intro!: "&I"; rule 0)
    apply "cqt:2"
    using "oa-exist:1" apply blast
    using "cont-nec-fact2:3" apply fast
    apply (rule "useful-tautologies:2"[THEN "→E"])
    using "cont-nec-fact2:1" apply fast
    using "rel-neg-T:3" apply fast
    using "oa-exist:1" apply blast
    using "cont-nec-fact1:1"[THEN "oth-class-taut:4:b"[THEN "≡E"(1)],
            THEN "≡E"(1), rotated, OF "cont-nec-fact2:3"] apply fast
    apply (rule "useful-tautologies:2"[THEN "→E"])
    using "cont-nec-fact2:1" apply blast
    apply (rule "=dfI"(2)[OF L_def]; "cqt:2[lambda]")
    using "oa-exist:1" apply fast
    using "cont-nec-fact2:4" apply fast
    apply (rule "useful-tautologies:2"[THEN "→E"])
    using "cont-nec-fact2:1" apply fast
    using "rel-neg-T:3" apply fast
    using "oa-exist:1" apply fast
     apply (rule "cont-nec-fact1:1"[unvarify F,
                   THEN "oth-class-taut:4:b"[THEN "≡E"(1)],
                   THEN "≡E"(1), rotated, OF "cont-nec-fact2:4"])
    apply (rule "=dfI"(2)[OF L_def]; "cqt:2[lambda]")
    apply (rule "useful-tautologies:2"[THEN "→E"])
    using "cont-nec-fact2:1" by blast
qed

AOT_theorem "cont-nec-fact2:6": ‹A! ≠ E! & A! ≠ E!⁻ & A! ≠ L & A! ≠ L⁻›    (223.6)
proof -
  AOT_have 1: ‹L↓›
    by (rule "=dfI"(2)[OF L_def]) "cqt:2[lambda]"+
```

```
  {
    fix φ and Π Π' :: <<κ>>
    AOT_have A: <¬(φ{Π'} ≡ φ{Π})> if  <φ{Π}> and <¬φ{Π'}>
    proof (rule "raa-cor:2")
      AOT_assume <φ{Π'} ≡ φ{Π}>
      AOT_hence <φ{Π'}> using that(1) "≡E" by blast
      AOT_thus <φ{Π'} & ¬φ{Π'}> using that(2) "&I" by blast
    qed
    AOT_have <Π' ≠ Π> if <Π↓> and <Π'↓> and <φ{Π}> and <¬φ{Π'}>
      using "pos-not-equiv-ne:4"[unvarify F G, THEN "→E",
              OF that(1,2), OF A[OF that(3, 4)]].
  } note 0 = this
  show ?thesis
    apply(safe intro!: "&I"; rule 0)
    apply "cqt:2"
    using "oa-exist:2" apply blast
    using "cont-nec-fact2:3" apply fast
    apply (rule "useful-tautologies:2"[THEN "→E"])
    using "cont-nec-fact2:2" apply fast
    using "rel-neg-T:3" apply fast
    using "oa-exist:2" apply blast
    using "cont-nec-fact1:1"[THEN "oth-class-taut:4:b"[THEN "≡E"(1)],
            THEN "≡E"(1), rotated, OF "cont-nec-fact2:3"] apply fast
    apply (rule "useful-tautologies:2"[THEN "→E"])
    using "cont-nec-fact2:2" apply blast
    apply (rule "=dfI"(2)[OF L_def]; "cqt:2[lambda]")
    using "oa-exist:2" apply fast
    using "cont-nec-fact2:4" apply fast
    apply (rule "useful-tautologies:2"[THEN "→E"])
    using "cont-nec-fact2:2" apply fast
    using "rel-neg-T:3" apply fast
    using "oa-exist:2" apply fast
     apply (rule "cont-nec-fact1:1"[unvarify F,
              THEN "oth-class-taut:4:b"[THEN "≡E"(1)],
              THEN "≡E"(1), rotated, OF "cont-nec-fact2:4"])
     apply (rule "=dfI"(2)[OF L_def]; "cqt:2[lambda]")
    apply (rule "useful-tautologies:2"[THEN "→E"])
    using "cont-nec-fact2:2" by blast
qed

AOT_define necessary_or_contingently_false :: <φ ⇒ φ> ("Δ_" [49] 54)
  <Δp ≡df □p ∨ (¬𝒜p & ◇p)>

AOT_theorem sixteen:                                                  (224)
 shows <∃F1∃F2∃F3∃F4∃F5∃F6∃F7∃F8∃F9∃F10∃F11∃F12∃F13∃F14∃F15∃F16 (
  «F1::<κ>» ≠ F2 & F1 ≠ F3 & F1 ≠ F4 & F1 ≠ F5 & F1 ≠ F6 & F1 ≠ F7 &
    F1 ≠ F8 & F1 ≠ F9 & F1 ≠ F10 & F1 ≠ F11 & F1 ≠ F12 & F1 ≠ F13 &
    F1 ≠ F14 & F1 ≠ F15 & F1 ≠ F16 &
  F2 ≠ F3 & F2 ≠ F4 & F2 ≠ F5 & F2 ≠ F6 & F2 ≠ F7 & F2 ≠ F8 &
    F2 ≠ F9 & F2 ≠ F10 & F2 ≠ F11 & F2 ≠ F12 & F2 ≠ F13 & F2 ≠ F14 &
    F2 ≠ F15 & F2 ≠ F16 &
  F3 ≠ F4 & F3 ≠ F5 & F3 ≠ F6 & F3 ≠ F7 & F3 ≠ F8 & F3 ≠ F9 & F3 ≠ F10 &
    F3 ≠ F11 & F3 ≠ F12 & F3 ≠ F13 & F3 ≠ F14 & F3 ≠ F15 & F3 ≠ F16 &
  F4 ≠ F5 & F4 ≠ F6 & F4 ≠ F7 & F4 ≠ F8 & F4 ≠ F9 & F4 ≠ F10 & F4 ≠ F11 &
    F4 ≠ F12 & F4 ≠ F13 & F4 ≠ F14 & F4 ≠ F15 & F4 ≠ F16 &
  F5 ≠ F6 & F5 ≠ F7 & F5 ≠ F8 & F5 ≠ F9 & F5 ≠ F10 & F5 ≠ F11 & F5 ≠ F12 &
    F5 ≠ F13 & F5 ≠ F14 & F5 ≠ F15 & F5 ≠ F16 &
  F6 ≠ F7 & F6 ≠ F8 & F6 ≠ F9 & F6 ≠ F10 & F6 ≠ F11 & F6 ≠ F12 & F6 ≠ F13 &
    F6 ≠ F14 & F6 ≠ F15 & F6 ≠ F16 &
  F7 ≠ F8 & F7 ≠ F9 & F7 ≠ F10 & F7 ≠ F11 & F7 ≠ F12 & F7 ≠ F13 & F7 ≠ F14 &
    F7 ≠ F15 & F7 ≠ F16 &
  F8 ≠ F9 & F8 ≠ F10 & F8 ≠ F11 & F8 ≠ F12 & F8 ≠ F13 & F8 ≠ F14 & F8 ≠ F15 &
    F8 ≠ F16 &
```

```
  F₉ ≠ F₁₀ & F₉ ≠ F₁₁ & F₉ ≠ F₁₂ & F₉ ≠ F₁₃ & F₉ ≠ F₁₄ & F₉ ≠ F₁₅ & F₉ ≠ F₁₆ &
  F₁₀ ≠ F₁₁ & F₁₀ ≠ F₁₂ & F₁₀ ≠ F₁₃ & F₁₀ ≠ F₁₄ & F₁₀ ≠ F₁₅ & F₁₀ ≠ F₁₆ &
  F₁₁ ≠ F₁₂ & F₁₁ ≠ F₁₃ & F₁₁ ≠ F₁₄ & F₁₁ ≠ F₁₅ & F₁₁ ≠ F₁₆ &
  F₁₂ ≠ F₁₃ & F₁₂ ≠ F₁₄ & F₁₂ ≠ F₁₅ & F₁₂ ≠ F₁₆ &
  F₁₃ ≠ F₁₄ & F₁₃ ≠ F₁₅ & F₁₃ ≠ F₁₆ &
  F₁₄ ≠ F₁₅ & F₁₄ ≠ F₁₆ &
  F₁₅ ≠ F₁₆)›
proof -
  AOT_have Delta_pos: ‹Δφ → ◇φ› for φ
  proof(rule "→I")
    AOT_assume ‹Δφ›
    AOT_hence ‹□φ ∨ (¬𝒜φ & ◇φ)›
      using "≡dfE"[OF necessary_or_contingently_false] by blast
    moreover {
      AOT_assume ‹□φ›
      AOT_hence ‹◇φ›
        by (metis "B◇" "T◇" "vdash-properties:10")
    }
    moreover {
      AOT_assume ‹¬𝒜φ & ◇φ›
      AOT_hence ‹◇φ›
        using "&E" by blast
    }
    ultimately AOT_show ‹◇φ›
      by (metis "∨E"(2) "raa-cor:1")
  qed

  AOT_have act_and_not_nec_not_delta: ‹¬Δφ› if ‹𝒜φ› and ‹¬□φ› for φ
    using "≡dfE" "&E"(1) "∨E"(2) necessary_or_contingently_false
          "raa-cor:3" that(1,2) by blast
  AOT_have act_and_pos_not_not_delta: ‹¬Δφ› if ‹𝒜φ› and ‹◇¬φ› for φ
    using "KBasic:11" act_and_not_nec_not_delta "≡E"(2) that(1,2) by blast
  AOT_have impossible_delta: ‹¬Δφ› if ‹¬◇φ› for φ
    using Delta_pos "modus-tollens:1" that by blast
  AOT_have not_act_and_pos_delta: ‹Δφ› if ‹¬𝒜φ› and ‹◇φ› for φ
    by (meson "≡dfI" "&I" "∨I"(2) necessary_or_contingently_false that(1,2))
  AOT_have nec_delta: ‹Δφ› if ‹□φ› for φ
    using "≡dfI" "∨I"(1) necessary_or_contingently_false that by blast

  AOT_obtain a where a_prop: ‹A!a›
    using "A-objects"[axiom_inst] "∃E"[rotated] "&E" by blast
  AOT_obtain b where b_prop: ‹◇[E!]b & ¬𝒜[E!]b›
    using "pos-not-pna:3" using "∃E"[rotated] by blast

  AOT_have b_ord: ‹[O!]b›
  proof(rule "=dfI"(2)[OF AOT_ordinary])
    AOT_show ‹[λx ◇[E!]x]↓› by "cqt:2[lambda]"
  next
    AOT_show ‹[λx ◇[E!]x]b›
    proof (rule "β←C"(1); ("cqt:2[lambda]")?)
      AOT_show ‹b↓› by (rule "cqt:2[const_var]"[axiom_inst])
      AOT_show ‹◇[E!]b› by (fact b_prop[THEN "&E"(1)])
    qed
  qed

  AOT_have nec_not_L_neg: ‹□¬[L¯]x› for x
    using "thm-noncont-e-e:2" "contingent-properties:2"[THEN "≡dfE"] "&E"
          CBF[THEN "→E"] "∀E" by blast
  AOT_have nec_L: ‹□[L]x› for x
    using "thm-noncont-e-e:1" "contingent-properties:1"[THEN "≡dfE"]
      CBF[THEN "→E"] "∀E" by blast

  AOT_have act_ord_b: ‹𝒜[O!]b›
```

```
    using b_ord "≡E"(1) "oa-facts:7" by blast
  AOT_have delta_ord_b: ‹Δ[O!]b›
    by (meson "≡dfI" b_ord "∨I"(1) necessary_or_contingently_false
              "oa-facts:1" "→E")
  AOT_have not_act_ord_a: ‹¬𝒜[O!]a›
    by (meson a_prop "≡E"(1) "≡E"(3) "oa-contingent:3" "oa-facts:7")
  AOT_have not_delta_ord_a: ‹¬Δ[O!]a›
    by (metis Delta_pos "≡E"(4) not_act_ord_a "oa-facts:3" "oa-facts:7"
              "reductio-aa:1" "→E")

  AOT_have not_act_abs_b: ‹¬𝒜[A!]b›
    by (meson b_ord "≡E"(1) "≡E"(3) "oa-contingent:2" "oa-facts:8")
  AOT_have not_delta_abs_b: ‹¬Δ[A!]b›
  proof(rule "raa-cor:2")
    AOT_assume ‹Δ[A!]b›
    AOT_hence ‹◇[A!]b›
      by (metis Delta_pos "vdash-properties:10")
    AOT_thus ‹[A!]b & ¬[A!]b›
      by (metis b_ord "&I" "≡E"(1) "oa-contingent:2"
                "oa-facts:4" "→E")
  qed
  AOT_have act_abs_a: ‹𝒜[A!]a›
    using a_prop "≡E"(1) "oa-facts:8" by blast
  AOT_have delta_abs_a: ‹Δ[A!]a›
    by (metis "≡dfI" a_prop "oa-facts:2" "→E" "∨I"(1)
              necessary_or_contingently_false)

  AOT_have not_act_concrete_b: ‹¬𝒜[E!]b›
    using b_prop "&E"(2) by blast
  AOT_have delta_concrete_b: ‹Δ[E!]b›
  proof (rule "≡dfI"[OF necessary_or_contingently_false];
         rule "∨I"(2); rule "&I")
    AOT_show ‹¬𝒜[E!]b› using b_prop "&E"(2) by blast
  next
    AOT_show ‹◇[E!]b› using b_prop "&E"(1) by blast
  qed
  AOT_have not_act_concrete_a: ‹¬𝒜[E!]a›
  proof (rule "raa-cor:2")
    AOT_assume ‹𝒜[E!]a›
    AOT_hence 1: ‹◇[E!]a› by (metis "Act-Sub:3" "→E")
    AOT_have ‹[A!]a› by (simp add: a_prop)
    AOT_hence ‹[λx ¬◇[E!]x]a›
      by (rule "=dfE"(2)[OF AOT_abstract, rotated]) "cqt:2"
    AOT_hence ‹¬◇[E!]a› using "β→C"(1) by blast
    AOT_thus ‹◇[E!]a & ¬◇[E!]a› using 1 "&I" by blast
  qed
  AOT_have not_delta_concrete_a: ‹¬Δ[E!]a›
  proof (rule "raa-cor:2")
    AOT_assume ‹Δ[E!]a›
    AOT_hence 1: ‹◇[E!]a› by (metis Delta_pos "vdash-properties:10")
    AOT_have ‹[A!]a› by (simp add: a_prop)
    AOT_hence ‹[λx ¬◇[E!]x]a›
      by (rule "=dfE"(2)[OF AOT_abstract, rotated]) "cqt:2[lambda]"
    AOT_hence ‹¬◇[E!]a› using "β→C"(1) by blast
    AOT_thus ‹◇[E!]a & ¬◇[E!]a› using 1 "&I" by blast
  qed

  AOT_have not_act_q_zero: ‹¬𝒜q₀›
    by (meson "log-prop-prop:2" "pos-not-pna:1"
              q₀_def "reductio-aa:1" "rule-id-df:2:a[zero]")
  AOT_have delta_q_zero: ‹Δq₀›
  proof(rule "≡dfI"[OF necessary_or_contingently_false];
        rule "∨I"(2); rule "&I")
```

```
  AOT_show ‹¬𝒜q₀› using not_act_q_zero.
  AOT_show ‹◇q₀› by (meson "&E"(1) q₀_prop)
qed
AOT_have act_not_q_zero: ‹𝒜¬q₀›
  using "Act-Basic:1" "∨E"(2) not_act_q_zero by blast
AOT_have not_delta_not_q_zero: ‹¬Δ¬q₀›
  using "≡dfE" "conventions:5" "Act-Basic:1" act_and_not_nec_not_delta
        "&E"(1) "∨E"(2) not_act_q_zero q₀_prop by blast

AOT_have ‹[L⁻]↓› by (simp add: "rel-neg-T:3")
moreover AOT_have ‹¬𝒜[L⁻]b & ¬Δ[L⁻]b & ¬𝒜[L⁻]a & ¬Δ[L⁻]a›
proof (safe intro!: "&I")
  AOT_show ‹¬𝒜[L⁻]b›
    by (meson "≡E"(1) "logic-actual-nec:1"[axiom_inst] "nec-imp-act"
              nec_not_L_neg "→E")
  AOT_show ‹¬Δ[L⁻]b›
    by (meson Delta_pos "KBasic2:1" "≡E"(1)
              "modus-tollens:1" nec_not_L_neg)
  AOT_show ‹¬𝒜[L⁻]a›
    by (meson "≡E"(1) "logic-actual-nec:1"[axiom_inst]
              "nec-imp-act" nec_not_L_neg "→E")
  AOT_show ‹¬Δ[L⁻]a›
    using Delta_pos "KBasic2:1" "≡E"(1) "modus-tollens:1"
          nec_not_L_neg by blast
qed
ultimately AOT_obtain F₀ where ‹¬𝒜[F₀]b & ¬Δ[F₀]b & ¬𝒜[F₀]a & ¬Δ[F₀]a›
  using "∃I"(1)[rotated, THEN "∃E"[rotated]] by fastforce
AOT_hence ‹¬𝒜[F₀]b› and ‹¬Δ[F₀]b› and ‹¬𝒜[F₀]a› and ‹¬Δ[F₀]a›
  using "&E" by blast+
note props = this

let ?Π = "«[λy [A!]y & q₀]»"
AOT_modally_strict {
  AOT_have ‹[«?Π»]↓› by "cqt:2[lambda]"
} note 1 = this
moreover AOT_have ‹¬𝒜[«?Π»]b & ¬Δ[«?Π»]b & ¬𝒜[«?Π»]a & Δ[«?Π»]a›
proof (safe intro!: "&I"; AOT_subst ‹[λy A!y & q₀]x› ‹A!x & q₀› for: x)
  AOT_show ‹¬𝒜([A!]b & q₀)›
    using "Act-Basic:2" "&E"(1) "≡E"(1) not_act_abs_b "raa-cor:3" by blast
next AOT_show ‹¬Δ([A!]b & q₀)›
    by (metis Delta_pos "KBasic2:3" "&E"(1) "≡E"(4) not_act_abs_b
              "oa-facts:4" "oa-facts:8" "raa-cor:3" "→E")
next AOT_show ‹¬𝒜([A!]a & q₀)›
    using "Act-Basic:2" "&E"(2) "≡E"(1) not_act_q_zero
          "raa-cor:3" by blast
next AOT_show ‹Δ([A!]a & q₀)›
  proof (rule not_act_and_pos_delta)
    AOT_show ‹¬𝒜([A!]a & q₀)›
      using "Act-Basic:2" "&E"(2) "≡E"(4) not_act_q_zero
            "raa-cor:3" by blast
  next AOT_show ‹◇([A!]a & q₀)›
      by (metis "&I" "→E" Delta_pos "KBasic:16" "&E"(1) delta_abs_a
                "≡E"(1) "oa-facts:6" q₀_prop)
  qed
qed(auto simp: "beta-C-meta"[THEN "→E", OF 1])
ultimately AOT_obtain F₁ where ‹¬𝒜[F₁]b & ¬Δ[F₁]b & ¬𝒜[F₁]a & Δ[F₁]a›
  using "∃I"(1)[rotated, THEN "∃E"[rotated]] by fastforce
AOT_hence ‹¬𝒜[F₁]b› and ‹¬Δ[F₁]b› and ‹¬𝒜[F₁]a› and ‹Δ[F₁]a›
  using "&E" by blast+
note props = props this

let ?Π = "«[λy [A!]y & ¬q₀]»"
AOT_modally_strict {
```

```
  AOT_have ‹[«?Π»]↓› by "cqt:2[lambda]"
} note 1 = this
moreover AOT_have ‹¬𝒜[«?Π»]b & ¬Δ[«?Π»]b & 𝒜[«?Π»]a & ¬Δ[«?Π»]a›
proof (safe intro!: "&I"; AOT_subst ‹[λy A!y & ¬q_0]x› ‹A!x & ¬q_0› for: x)
  AOT_show ‹¬𝒜([A!]b & ¬q_0)›
    using "Act-Basic:2" "&E"(1) "≡E"(1) not_act_abs_b "raa-cor:3" by blast
next AOT_show ‹¬Δ([A!]b & ¬q_0)›
    by (meson "RM◇" Delta_pos "Conjunction Simplification"(1) "≡E"(4)
              "modus-tollens:1" not_act_abs_b "oa-facts:4" "oa-facts:8")
next AOT_show ‹𝒜([A!]a & ¬q_0)›
    by (metis "Act-Basic:1" "Act-Basic:2" act_abs_a "&I" "∨E"(2)
              "≡E"(3) not_act_q_zero "raa-cor:3")
next AOT_show ‹¬Δ([A!]a & ¬q_0)›
  proof (rule act_and_not_nec_not_delta)
    AOT_show ‹𝒜([A!]a & ¬q_0)›
      by (metis "Act-Basic:1" "Act-Basic:2" act_abs_a "&I" "∨E"(2)
                "≡E"(3) not_act_q_zero "raa-cor:3")
  next
    AOT_show ‹¬□([A!]a & ¬q_0)›
      by (metis "KBasic2:1" "KBasic:3" "&E"(1) "&E"(2) "≡E"(4)
                q_0_prop "raa-cor:3")
  qed
qed(auto simp: "beta-C-meta"[THEN "→E", OF 1])
ultimately AOT_obtain F_2 where ‹¬𝒜[F_2]b & ¬Δ[F_2]b & 𝒜[F_2]a & ¬Δ[F_2]a›
  using "∃I"(1)[rotated, THEN "∃E"[rotated]] by fastforce
AOT_hence ‹¬𝒜[F_2]b› and ‹¬Δ[F_2]b› and ‹𝒜[F_2]a› and ‹¬Δ[F_2]a›
  using "&E" by blast+
note props = props this

AOT_have abstract_prop: ‹¬𝒜[A!]b & ¬Δ[A!]b & 𝒜[A!]a & Δ[A!]a›
  using act_abs_a "&I" delta_abs_a not_act_abs_b not_delta_abs_b
  by presburger
then AOT_obtain F_3 where ‹¬𝒜[F_3]b & ¬Δ[F_3]b & 𝒜[F_3]a & Δ[F_3]a›
  using "∃I"(1)[rotated, THEN "∃E"[rotated]] "oa-exist:2" by fastforce
AOT_hence ‹¬𝒜[F_3]b› and ‹¬Δ[F_3]b› and ‹𝒜[F_3]a› and ‹Δ[F_3]a›
  using "&E" by blast+
note props = props this

AOT_have ‹¬𝒜[E!]b & Δ[E!]b & ¬𝒜[E!]a & ¬Δ[E!]a›
  by (meson "&I" delta_concrete_b not_act_concrete_a
            not_act_concrete_b not_delta_concrete_a)
then AOT_obtain F_4 where ‹¬𝒜[F_4]b & Δ[F_4]b & ¬𝒜[F_4]a & ¬Δ[F_4]a›
  using "∃I"(1)[rotated, THEN "∃E"[rotated]]
  by fastforce
AOT_hence ‹¬𝒜[F_4]b› and ‹Δ[F_4]b› and ‹¬𝒜[F_4]a› and ‹¬Δ[F_4]a›
  using "&E" by blast+
note props = props this

AOT_modally_strict {
  AOT_have ‹[λy q_0]↓› by "cqt:2[lambda]"
} note 1 = this
moreover AOT_have ‹¬𝒜[λy q_0]b & Δ[λy q_0]b & ¬𝒜[λy q_0]a & Δ[λy q_0]a›
  by (safe intro!: "&I"; AOT_subst ‹[λy q_0]b› ‹q_0› for: b)
     (auto simp: not_act_q_zero delta_q_zero  "beta-C-meta"[THEN "→E", OF 1])
ultimately AOT_obtain F_5 where ‹¬𝒜[F_5]b & Δ[F_5]b & ¬𝒜[F_5]a & Δ[F_5]a›
  using "∃I"(1)[rotated, THEN "∃E"[rotated]]
  by fastforce
AOT_hence ‹¬𝒜[F_5]b› and ‹Δ[F_5]b› and ‹¬𝒜[F_5]a› and ‹Δ[F_5]a›
  using "&E" by blast+
note props = props this

let ?Π = "«[λy [E!]y ∨ ([A!]y & ¬q_0)]»"
AOT_modally_strict {
```

```
    AOT_have ‹[«?Π»]↓› by "cqt:2[lambda]"
  } note 1 = this
  moreover AOT_have ‹¬𝒜[«?Π»]b & Δ[«?Π»]b & 𝒜[«?Π»]a & ¬Δ[«?Π»]a›
  proof(safe intro!: "&I";
        AOT_subst ‹[λy E!y ∨ (A!y & ¬q₀)]x› ‹E!x ∨ (A!x & ¬q₀)› for: x)
    AOT_have ‹𝒜¬([A!]b & ¬q₀)›
      by (metis "Act-Basic:1" "Act-Basic:2" abstract_prop "&E"(1) "∨E"(2)
                "≡E"(1) "raa-cor:3")
    moreover AOT_have ‹¬𝒜[E!]b›
      using b_prop "&E"(2) by blast
    ultimately AOT_have 2: ‹𝒜(¬[E!]b & ¬([A!]b & ¬q₀))›
      by (metis "Act-Basic:2" "Act-Sub:1" "&I" "≡E"(3) "raa-cor:1")
    AOT_have ‹𝒜¬([E!]b ∨ ([A!]b & ¬q₀))›
      by (AOT_subst ‹¬([E!]b ∨ ([A!]b & ¬q₀))› ‹¬[E!]b & ¬([A!]b & ¬q₀)›)
         (auto simp: "oth-class-taut:5:d" 2)
    AOT_thus ‹¬𝒜([E!]b ∨ ([A!]b & ¬q₀))›
      by (metis "¬¬I" "Act-Sub:1" "≡E"(4))
  next
    AOT_show ‹Δ([E!]b ∨ ([A!]b & ¬q₀))›
    proof (rule not_act_and_pos_delta)
      AOT_show ‹¬𝒜([E!]b ∨ ([A!]b & ¬q₀))›
        by (metis "Act-Basic:2" "Act-Basic:9" "∨E"(2) "raa-cor:3"
                  "Conjunction Simplification"(1) "≡E"(4)
                  "modus-tollens:1" not_act_abs_b not_act_concrete_b)
    next
      AOT_show ‹◇([E!]b ∨ ([A!]b & ¬q₀))›
        using "KBasic2:2" b_prop "&E"(1) "∨I"(1) "≡E"(3) "raa-cor:3" by blast
    qed
  next AOT_show ‹𝒜([E!]a ∨ ([A!]a & ¬q₀))›
      by (metis "Act-Basic:1" "Act-Basic:2" "Act-Basic:9" act_abs_a "&I"
                "∨I"(2) "∨E"(2) "≡E"(3) not_act_q_zero "raa-cor:1")
  next AOT_show ‹¬Δ([E!]a ∨ ([A!]a & ¬q₀))›
    proof (rule act_and_not_nec_not_delta)
      AOT_show ‹𝒜([E!]a ∨ ([A!]a & ¬q₀))›
        by (metis "Act-Basic:1" "Act-Basic:2" "Act-Basic:9" act_abs_a "&I"
                  "∨I"(2) "∨E"(2) "≡E"(3) not_act_q_zero "raa-cor:1")
    next
      AOT_have ‹□¬[E!]a›
        by (metis "≡dfI" "conventions:5" "&I" "∨I"(2)
                  necessary_or_contingently_false
                  not_act_concrete_a not_delta_concrete_a "raa-cor:3")
      moreover AOT_have ‹◇¬([A!]a & ¬q₀)›
        by (metis "KBasic2:1" "KBasic:11" "KBasic:3"
                  "&E"(1,2) "≡E"(1) q₀_prop "raa-cor:3")
      ultimately AOT_have ‹◇(¬[E!]a & ¬([A!]a & ¬q₀))›
        by (metis "KBasic:16" "&I" "vdash-properties:10")
      AOT_hence ‹◇¬([E!]a ∨ ([A!]a & ¬q₀))›
        by (metis "RE◇" "≡E"(2) "oth-class-taut:5:d")
      AOT_thus ‹¬□([E!]a ∨ ([A!]a & ¬q₀))›
        by (metis "KBasic:12" "≡E"(1) "raa-cor:3")
    qed
  qed(auto simp: "beta-C-meta"[THEN "→E", OF 1])
  ultimately AOT_obtain F₆ where ‹¬𝒜[F₆]b & Δ[F₆]b & 𝒜[F₆]a & ¬Δ[F₆]a›
    using "∃I"(1)[rotated, THEN "∃E"[rotated]] by fastforce
  AOT_hence ‹¬𝒜[F₆]b› and ‹Δ[F₆]b› and ‹𝒜[F₆]a› and ‹¬Δ[F₆]a›
    using "&E" by blast+
  note props = props this

  let ?Π = "«[λy [A!]y ∨ [E!]y]»"
  AOT_modally_strict {
    AOT_have ‹[«?Π»]↓› by "cqt:2[lambda]"
  } note 1 = this
  moreover AOT_have ‹¬𝒜[«?Π»]b & Δ[«?Π»]b & 𝒜[«?Π»]a & Δ[«?Π»]a›
```

```
proof(safe intro!: "&I"; AOT_subst ‹[λy A!y ∨ E!y]x› ‹A!x ∨ E!x› for: x)
  AOT_show ‹¬𝒜([A!]b ∨ [E!]b)›
    using "Act-Basic:9" "∨E"(2) "≡E"(4) not_act_abs_b
          not_act_concrete_b "raa-cor:3" by blast
next AOT_show ‹Δ([A!]b ∨ [E!]b)›
  proof (rule not_act_and_pos_delta)
    AOT_show ‹¬𝒜([A!]b ∨ [E!]b)›
      using "Act-Basic:9" "∨E"(2) "≡E"(4) not_act_abs_b
            not_act_concrete_b "raa-cor:3" by blast
  next AOT_show ‹◊([A!]b ∨ [E!]b)›
      using "KBasic2:2" b_prop "&E"(1) "∨I"(2) "≡E"(2) by blast
  qed
next AOT_show ‹𝒜([A!]a ∨ [E!]a)›
    by (meson "Act-Basic:9" act_abs_a "∨I"(1) "≡E"(2))
next AOT_show ‹Δ([A!]a ∨ [E!]a) ›
  proof (rule nec_delta)
    AOT_show ‹□([A!]a ∨ [E!]a)›
      by (metis "KBasic:15" act_abs_a act_and_not_nec_not_delta
                "Disjunction Addition"(1) delta_abs_a "raa-cor:3" "→E")
  qed
qed(auto simp: "beta-C-meta"[THEN "→E", OF 1])
ultimately AOT_obtain F7 where ‹¬𝒜[F7]b & Δ[F7]b & 𝒜[F7]a & Δ[F7]a›
  using "∃I"(1)[rotated, THEN "∃E"[rotated]] by fastforce
AOT_hence ‹¬𝒜[F7]b› and ‹Δ[F7]b› and ‹𝒜[F7]a› and ‹Δ[F7]a›
  using "&E" by blast+
note props = props this

let ?Π = "«[λy [O!]y & ¬[E!]y]»"
AOT_modally_strict {
  AOT_have ‹[«?Π»]↓› by "cqt:2[lambda]"
} note 1 = this
moreover AOT_have ‹𝒜[«?Π»]b & ¬Δ[«?Π»]b & ¬𝒜[«?Π»]a & ¬Δ[«?Π»]a›
proof(safe intro!: "&I"; AOT_subst ‹[λy O!y & ¬E!y]x› ‹O!x & ¬E!x› for: x)
  AOT_show ‹𝒜([O!]b & ¬[E!]b)›
    by (metis "Act-Basic:1" "Act-Basic:2" act_ord_b "&I" "∨E"(2)
         "≡E"(3) not_act_concrete_b "raa-cor:3")
next AOT_show ‹¬Δ([O!]b & ¬[E!]b)›
    by (metis (no_types, opaque_lifting) "conventions:5" "Act-Sub:1" "RM:1"
         act_and_not_nec_not_delta "act-conj-act:3"
         act_ord_b b_prop "&I" "&E"(1) "Conjunction Simplification"(2)
         "df-rules-formulas[3]"
         "≡E"(3) "raa-cor:1" "→E")
next AOT_show ‹¬𝒜([O!]a & ¬[E!]a)›
    using "Act-Basic:2" "&E"(1) "≡E"(1) not_act_ord_a "raa-cor:3" by blast
next AOT_have ‹¬◊([O!]a & ¬[E!]a)›
    by (metis "KBasic2:3" "&E"(1) "≡E"(4) not_act_ord_a "oa-facts:3"
         "oa-facts:7" "raa-cor:3" "vdash-properties:10")
  AOT_thus ‹¬Δ([O!]a & ¬[E!]a)›
    by (rule impossible_delta)
qed(auto simp: "beta-C-meta"[THEN "→E", OF 1])
ultimately AOT_obtain F8 where ‹𝒜[F8]b & ¬Δ[F8]b & ¬𝒜[F8]a & ¬Δ[F8]a›
  using "∃I"(1)[rotated, THEN "∃E"[rotated]] by fastforce
AOT_hence ‹𝒜[F8]b› and ‹¬Δ[F8]b› and ‹¬𝒜[F8]a› and ‹¬Δ[F8]a›
  using "&E" by blast+
note props = props this

let ?Π = "«[λy ¬[E!]y & ([O!]y ∨ q0)]»"
AOT_modally_strict {
  AOT_have ‹[«?Π»]↓› by "cqt:2[lambda]"
} note 1 = this
moreover AOT_have ‹𝒜[«?Π»]b & ¬Δ[«?Π»]b & ¬𝒜[«?Π»]a & Δ[«?Π»]a›
proof(safe intro!: "&I";
      AOT_subst ‹[λy ¬E!y & (O!y ∨ q0)]x› ‹¬E!x & (O!x ∨ q0)› for: x)
```

```
  AOT_show ‹𝒜(¬[E!]b & ([O!]b ∨ q₀))›
    by (metis "Act-Basic:1" "Act-Basic:2" "Act-Basic:9" act_ord_b "&I"
              "∨I"(1) "∨E"(2) "≡E"(3) not_act_concrete_b "raa-cor:1")
next AOT_show ‹¬Δ(¬[E!]b & ([O!]b ∨ q₀))›
  proof (rule act_and_pos_not_not_delta)
    AOT_show ‹𝒜(¬[E!]b & ([O!]b ∨ q₀))›
      by (metis "Act-Basic:1" "Act-Basic:2" "Act-Basic:9" act_ord_b "&I"
                "∨I"(1) "∨E"(2) "≡E"(3) not_act_concrete_b "raa-cor:1")
  next
    AOT_show ‹◊¬(¬[E!]b & ([O!]b ∨ q₀))›
    proof (AOT_subst ‹¬(¬[E!]b & ([O!]b ∨ q₀))› ‹[E!]b ∨ ¬([O!]b ∨ q₀)›)
      AOT_modally_strict {
        AOT_show ‹¬(¬[E!]b & ([O!]b ∨ q₀)) ≡ [E!]b ∨ ¬([O!]b ∨ q₀)›
          by (metis "&I" "&E"(1,2) "∨I"(1,2) "∨E"(2)
                    "→I" "≡I" "reductio-aa:1")
      }
    next
      AOT_show ‹◊([E!]b ∨ ¬([O!]b ∨ q₀))›
        using "KBasic2:2" b_prop "&E"(1) "∨I"(1) "≡E"(3)
              "raa-cor:3" by blast
     qed
   qed
 next
   AOT_show ‹¬𝒜(¬[E!]a & ([O!]a ∨ q₀))›
     using "Act-Basic:2" "Act-Basic:9" "&E"(2) "∨E"(3) "≡E"(1)
           not_act_ord_a not_act_q_zero "reductio-aa:2" by blast
 next
   AOT_show ‹Δ(¬[E!]a & ([O!]a ∨ q₀))›
   proof (rule not_act_and_pos_delta)
     AOT_show ‹¬𝒜(¬[E!]a & ([O!]a ∨ q₀))›
       by (metis "Act-Basic:2" "Act-Basic:9" "&E"(2) "∨E"(3) "≡E"(1)
                 not_act_ord_a not_act_q_zero "reductio-aa:2")
   next
     AOT_have ‹□¬[E!]a›
       using "KBasic2:1" "≡E"(2) not_act_and_pos_delta not_act_concrete_a
             not_delta_concrete_a "raa-cor:5" by blast
     moreover AOT_have ‹◊([O!]a ∨ q₀)›
       by (metis "KBasic2:2" "&E"(1) "∨I"(2) "≡E"(3) q₀_prop "raa-cor:3")
     ultimately AOT_show ‹◊(¬[E!]a & ([O!]a ∨ q₀))›
       by (metis "KBasic:16" "&I" "vdash-properties:10")
   qed
 qed(auto simp:  "beta-C-meta"[THEN "→E", OF 1])
ultimately AOT_obtain F₉ where ‹𝒜[F₉]b & ¬Δ[F₉]b & ¬𝒜[F₉]a & Δ[F₉]a›
  using "∃I"(1)[rotated, THEN "∃E"[rotated]] by fastforce
AOT_hence ‹𝒜[F₉]b› and ‹¬Δ[F₉]b› and ‹¬𝒜[F₉]a› and ‹Δ[F₉]a›
  using "&E" by blast+
note props = props this

AOT_modally_strict {
  AOT_have ‹[λy ¬q₀]↓› by "cqt:2[lambda]"
} note 1 = this
moreover AOT_have ‹𝒜[λy ¬q₀]b & ¬Δ[λy ¬q₀]b & 𝒜[λy ¬q₀]a & ¬Δ[λy ¬q₀]a›
  by (safe intro!: "&I"; AOT_subst ‹[λy ¬q₀]x› ‹¬q₀› for: x)
     (auto simp: act_not_q_zero not_delta_not_q_zero
                 "beta-C-meta"[THEN "→E", OF 1])
ultimately AOT_obtain F₁₀ where ‹𝒜[F₁₀]b & ¬Δ[F₁₀]b & 𝒜[F₁₀]a & ¬Δ[F₁₀]a›
  using "∃I"(1)[rotated, THEN "∃E"[rotated]] by fastforce
AOT_hence ‹𝒜[F₁₀]b› and ‹¬Δ[F₁₀]b› and ‹𝒜[F₁₀]a› and ‹¬Δ[F₁₀]a›
  using "&E" by blast+
note props = props this

AOT_modally_strict {
  AOT_have ‹[λy ¬[E!]y]↓› by "cqt:2[lambda]"
```

```
} note 1 = this
moreover AOT_have ‹𝒜[λy ¬[E!]y]b & ¬Δ[λy ¬[E!]y]b &
                   𝒜[λy ¬[E!]y]a & Δ[λy ¬[E!]y]a›
proof (safe intro!: "&I"; AOT_subst ‹[λy ¬[E!]y]x› ‹¬[E!]x› for: x)
  AOT_show ‹𝒜¬[E!]b›
    using "Act-Basic:1" "∨E"(2) not_act_concrete_b by blast
next AOT_show ‹¬Δ¬[E!]b›
    using "≡dfE" "conventions:5" "Act-Basic:1" act_and_not_nec_not_delta
          b_prop "&E"(1) "∨E"(2) not_act_concrete_b by blast
next AOT_show ‹𝒜¬[E!]a›
    using "Act-Basic:1" "∨E"(2) not_act_concrete_a by blast
next AOT_show ‹Δ¬[E!]a›
    using "KBasic2:1" "≡E"(2) nec_delta not_act_and_pos_delta
          not_act_concrete_a not_delta_concrete_a "reductio-aa:1"
    by blast
qed(auto simp: "beta-C-meta"[THEN "→E", OF 1])
ultimately AOT_obtain F11 where ‹𝒜[F11]b & ¬Δ[F11]b & 𝒜[F11]a & Δ[F11]a›
  using "∃I"(1)[rotated, THEN "∃E"[rotated]] by fastforce
AOT_hence ‹𝒜[F11]b› and ‹¬Δ[F11]b› and ‹𝒜[F11]a› and ‹Δ[F11]a›
  using "&E" by blast+
note props = props this

AOT_have ‹𝒜[O!]b & Δ[O!]b & ¬𝒜[O!]a & ¬Δ[O!]a›
  by (simp add: act_ord_b "&I" delta_ord_b not_act_ord_a not_delta_ord_a)
then AOT_obtain F12 where ‹𝒜[F12]b & Δ[F12]b & ¬𝒜[F12]a & ¬Δ[F12]a›
  using "oa-exist:1" "∃I"(1)[rotated, THEN "∃E"[rotated]] by fastforce
AOT_hence ‹𝒜[F12]b› and ‹Δ[F12]b› and ‹¬𝒜[F12]a› and ‹¬Δ[F12]a›
  using "&E" by blast+
note props = props this

let ?Π = "«[λy [O!]y ∨ q0]»"
AOT_modally_strict {
  AOT_have ‹[«?Π»]↓› by "cqt:2[lambda]"
} note 1 = this
moreover AOT_have ‹𝒜[«?Π»]b & Δ[«?Π»]b & ¬𝒜[«?Π»]a & Δ[«?Π»]a›
proof (safe intro!: "&I"; AOT_subst ‹[λy O!y ∨ q0]x› ‹O!x ∨ q0› for: x)
  AOT_show ‹𝒜([O!]b ∨ q0)›
    by (meson "Act-Basic:9" act_ord_b "∨I"(1) "≡E"(2))
next AOT_show ‹Δ([O!]b ∨ q0)›
    by (meson "KBasic:15" b_ord "∨I"(1) nec_delta "oa-facts:1" "→E")
next AOT_show ‹¬𝒜([O!]a ∨ q0)›
    using "Act-Basic:9" "∨E"(2) "≡E"(4) not_act_ord_a
          not_act_q_zero "raa-cor:3" by blast
next AOT_show ‹Δ([O!]a ∨ q0)›
  proof (rule not_act_and_pos_delta)
    AOT_show ‹¬𝒜([O!]a ∨ q0)›
      using "Act-Basic:9" "∨E"(2) "≡E"(4) not_act_ord_a
            not_act_q_zero "raa-cor:3" by blast
  next AOT_show ‹◊([O!]a ∨ q0)›
      using "KBasic2:2" "&E"(1) "∨I"(2) "≡E"(2) q0_prop by blast
  qed
qed(auto simp: "beta-C-meta"[THEN "→E", OF 1])
ultimately AOT_obtain F13 where ‹𝒜[F13]b & Δ[F13]b & ¬𝒜[F13]a & Δ[F13]a›
  using "∃I"(1)[rotated, THEN "∃E"[rotated]] by fastforce
AOT_hence ‹𝒜[F13]b› and ‹Δ[F13]b› and ‹¬𝒜[F13]a› and ‹Δ[F13]a›
  using "&E" by blast+
note props = props this

let ?Π = "«[λy [O!]y ∨ ¬q0]»"
AOT_modally_strict {
   AOT_have ‹[«?Π»]↓› by "cqt:2[lambda]"
} note 1 = this
moreover AOT_have ‹𝒜[«?Π»]b & Δ[«?Π»]b & 𝒜[«?Π»]a & ¬Δ[«?Π»]a›
```

```
proof (safe intro!: "&I"; AOT_subst ‹[λy O!y ∨ ¬q0]x› ‹O!x ∨ ¬q0› for: x)
  AOT_show ‹𝒜([O!]b ∨ ¬q0)›
    by (meson "Act-Basic:9" act_not_q_zero "∨I"(2) "≡E"(2))
next AOT_show ‹Δ([O!]b ∨ ¬q0)›
    by (meson "KBasic:15" b_ord "∨I"(1) nec_delta "oa-facts:1" "→E")
next AOT_show ‹𝒜([O!]a ∨ ¬q0)›
    by (meson "Act-Basic:9" act_not_q_zero "∨I"(2) "≡E"(2))
next AOT_show ‹¬Δ([O!]a ∨ ¬q0)›
  proof(rule act_and_pos_not_not_delta)
    AOT_show ‹𝒜([O!]a ∨ ¬q0)›
      by (meson "Act-Basic:9" act_not_q_zero "∨I"(2) "≡E"(2))
  next
    AOT_have ‹□¬[O!]a›
      using "KBasic2:1" "≡E"(2) not_act_and_pos_delta
            not_act_ord_a not_delta_ord_a "raa-cor:6" by blast
    moreover AOT_have ‹◇q0›
      by (meson "&E"(1) q0_prop)
    ultimately AOT_have 2: ‹◇(¬[O!]a & q0)›
       by (metis "KBasic:16" "&I" "vdash-properties:10")
    AOT_show ‹◇¬([O!]a ∨ ¬q0)›
    proof (AOT_subst (reverse) ‹¬([O!]a ∨ ¬q0)› ‹¬[O!]a & q0›)
      AOT_modally_strict {
        AOT_show ‹¬[O!]a & q0 ≡ ¬([O!]a ∨ ¬q0)›
          by (metis "&I" "&E"(1) "&E"(2) "∨I"(1) "∨I"(2)
                    "∨E"(3) "deduction-theorem" "≡I" "raa-cor:3")
      }
    next
      AOT_show ‹◇(¬[O!]a & q0)›
        using "2" by blast
    qed
  qed
qed(auto simp: "beta-C-meta"[THEN "→E", OF 1])
ultimately AOT_obtain F14 where ‹𝒜[F14]b & Δ[F14]b & 𝒜[F14]a & ¬Δ[F14]a›
  using "∃I"(1)[rotated, THEN "∃E"[rotated]] by fastforce
AOT_hence ‹𝒜[F14]b› and ‹Δ[F14]b› and ‹𝒜[F14]a› and ‹¬Δ[F14]a›
  using "&E" by blast+
note props = props this

AOT_have ‹[L]↓›
  by (rule "=dfI"(2)[OF L_def]) "cqt:2[lambda]"+
moreover AOT_have ‹𝒜[L]b & Δ[L]b & 𝒜[L]a & Δ[L]a›
proof (safe intro!: "&I")
  AOT_show ‹𝒜[L]b›
    by (meson nec_L "nec-imp-act" "vdash-properties:10")
  next AOT_show ‹Δ[L]b› using nec_L nec_delta by blast
  next AOT_show ‹𝒜[L]a› by (meson nec_L "nec-imp-act" "→E")
  next AOT_show ‹Δ[L]a› using nec_L nec_delta by blast
qed
ultimately AOT_obtain F15 where ‹𝒜[F15]b & Δ[F15]b & 𝒜[F15]a & Δ[F15]a›
  using "∃I"(1)[rotated, THEN "∃E"[rotated]] by fastforce
AOT_hence ‹𝒜[F15]b› and ‹Δ[F15]b› and ‹𝒜[F15]a› and ‹Δ[F15]a›
  using "&E" by blast+
note props = props this

show ?thesis
  by (rule "∃I"(2)[where β=F0]; rule "∃I"(2)[where β=F1];
      rule "∃I"(2)[where β=F2]; rule "∃I"(2)[where β=F3];
      rule "∃I"(2)[where β=F4]; rule "∃I"(2)[where β=F5];
      rule "∃I"(2)[where β=F6]; rule "∃I"(2)[where β=F7];
      rule "∃I"(2)[where β=F8]; rule "∃I"(2)[where β=F9];
      rule "∃I"(2)[where β=F10]; rule "∃I"(2)[where β=F11];
      rule "∃I"(2)[where β=F12]; rule "∃I"(2)[where β=F13];
      rule "∃I"(2)[where β=F14]; rule "∃I"(2)[where β=F15];
```

```
        safe intro!: "&I")
       (match conclusion in "[?v ⊨ [F] ≠ [G]]" for F G ⇒ ‹
        match props in A: "[?v ⊨ ¬φ{F}]" for φ ⇒ ‹
        match (φ) in "λa . ?p" ⇒ ‹fail› | "λa . a" ⇒ ‹fail› | _ ⇒ ‹
        match props in B: "[?v ⊨ φ{G}]" ⇒ ‹
        fact "pos-not-equiv-ne:4"[where F=F and G=G and φ=φ, THEN "→E",
                                OF "oth-class-taut:4:h"[THEN "≡E"(2)],
                                OF "Disjunction Addition"(2)[THEN "→E"],
                                OF "&I", OF A, OF B]››››)+
qed

subsection‹The Theory of Objects›
text‹\label{PLM: 9.11}›

AOT_theorem "o-objects-exist:1": ‹□∃x O!x›                                    (225.1)
proof(rule RN)
  AOT_modally_strict {
    AOT_obtain a where ‹◇(E!a & ¬𝒜[E!]a)›
      using "∃E"[rotated, OF "qml:4"[axiom_inst, THEN "BF◇"[THEN "→E"]]]
      by blast
    AOT_hence 1: ‹◇E!a› by (metis "KBasic2:3" "&E"(1) "→E")
    AOT_have ‹[λx ◇[E!]x]a›
    proof (rule "β←C"(1); "cqt:2[lambda]"?)
      AOT_show ‹a↓› using "cqt:2[const_var]"[axiom_inst] by blast
    next
      AOT_show ‹◇E!a› by (fact 1)
    qed
    AOT_hence ‹O!a› by (rule "=dfI"(2)[OF AOT_ordinary, rotated]) "cqt:2"
    AOT_thus ‹∃x [O!]x› by (rule "∃I")
  }
qed

AOT_theorem "o-objects-exist:2": ‹□∃x A!x›                                    (225.2)
proof (rule RN)
  AOT_modally_strict {
    AOT_obtain a where ‹[A!]a›
      using "A-objects"[axiom_inst] "∃E"[rotated] "&E" by blast
    AOT_thus ‹∃x A!x› using "∃I" by blast
  }
qed

AOT_theorem "o-objects-exist:3": ‹□¬∀x O!x›                                   (225.3)
  by (rule RN)
     (metis (no_types, opaque_lifting) "∃E" "cqt-orig:1[const_var]"
        "≡E"(4) "modus-tollens:1" "o-objects-exist:2" "oa-contingent:2"
        "qml:2"[axiom_inst] "reductio-aa:2")

AOT_theorem "o-objects-exist:4": ‹□¬∀x A!x›                                   (225.4)
  by (rule RN)
     (metis (mono_tags, opaque_lifting) "∃E" "cqt-orig:1[const_var]"
        "≡E"(1) "modus-tollens:1" "o-objects-exist:1" "oa-contingent:2"
        "qml:2"[axiom_inst] "→E")

AOT_theorem "o-objects-exist:5": ‹□¬∀x E!x›                                   (225.5)
proof (rule RN; rule "raa-cor:2")
  AOT_modally_strict {
    AOT_assume ‹∀x E!x›
    moreover AOT_obtain a where abs: ‹A!a›
      using "o-objects-exist:2"[THEN "qml:2"[axiom_inst, THEN "→E"]]
            "∃E"[rotated] by blast
    ultimately AOT_have ‹E!a› using "∀E" by blast
    AOT_hence 1: ‹◇E!a› by (metis "T◇" "→E")
    AOT_have ‹[λy ◇E!y]a›
```

```
    proof (rule "β←C"(1); "cqt:2[lambda]"?)
      AOT_show ‹a↓› using "cqt:2[const_var]"[axiom_inst].
    next
      AOT_show ‹◇E!a› by (fact 1)
    qed
    AOT_hence ‹O!a›
      by (rule "=dfI"(2)[OF AOT_ordinary, rotated]) "cqt:2[lambda]"
    AOT_hence ‹¬A!a› by (metis "≡E"(1) "oa-contingent:2")
    AOT_thus ‹p & ¬p› for p using abs by (metis "raa-cor:3")
  }
qed

AOT_theorem partition: ‹¬∃x (O!x & A!x)›                                    (226)
proof(rule "raa-cor:2")
  AOT_assume ‹∃x (O!x & A!x)›
  then AOT_obtain a where ‹O!a & A!a›
    using "∃E"[rotated] by blast
  AOT_thus ‹p & ¬p› for p
    by (metis "&E"(1) "Conjunction Simplification"(2) "≡E"(1)
              "modus-tollens:1" "oa-contingent:2" "raa-cor:3")
qed

AOT_define eq_E :: ‹Π› ("'(=E')")
  "=E": ‹(=E) =df [λxy O!x & O!y & □∀F ([F]x ≡ [F]y)]›                        (227)

syntax "_AOT_eq_E_infix" :: ‹τ ⇒ τ ⇒ φ› (infixl "=E" 50)
translations
  "_AOT_eq_E_infix κ κ'" == "CONST AOT_exe (CONST eq_E) (CONST Pair κ κ')"
print_translation‹
AOT_syntax_print_translations
[(const_syntax‹AOT_exe›, fn ctxt => fn [
  Const (const_name‹eq_E›, _),
  Const (const_syntax‹Pair›, _) $ lhs $ rhs
] => Const (syntax_const‹_AOT_eq_E_infix›, dummyT) $ lhs $ rhs)]›

text‹Note: Not explicitly mentioned as theorem in PLM.›
AOT_theorem "=E[denotes]": ‹[(=E)]↓›                                          (227)
  by (rule "=dfI"(2)[OF "=E"]) "cqt:2[lambda]"+

AOT_theorem "=E-simple:1": ‹x =E y ≡ (O!x & O!y & □∀F ([F]x ≡ [F]y))›        (230.1)
proof -
  AOT_have 1: ‹[λxy [O!]x & [O!]y & □∀F ([F]x ≡ [F]y)]↓› by "cqt:2"
  show ?thesis
    apply (rule "=dfI"(2)[OF "=E"]; "cqt:2[lambda]"?)
    using "beta-C-meta"[THEN "→E", OF 1, unvarify ν1νn, of "(_,_)",
                        OF tuple_denotes[THEN "≡dfI"], OF "&I",
                        OF "cqt:2[const_var]"[axiom_inst],
                        OF "cqt:2[const_var]"[axiom_inst]]
    by fast
qed

AOT_theorem "=E-simple:2": ‹x =E y → x = y›                                  (230.2)
proof (rule "→I")
  AOT_assume ‹x =E y›
  AOT_hence ‹O!x & O!y & □∀F ([F]x ≡ [F]y)›
    using "=E-simple:1"[THEN "≡E"(1)] by blast
  AOT_thus ‹x = y›
    using "≡dfI"[OF "identity:1"] "∨I" by blast
qed

AOT_theorem "id-nec3:1": ‹x =E y ≡ □(x =E y)›                                (231.1)
proof (rule "≡I"; rule "→I")
  AOT_assume ‹x =E y›
```

```
  AOT_hence ‹O!x & O!y & □∀F ([F]x ≡ [F]y)›
    using "=E-simple:1" "≡E" by blast
  AOT_hence ‹□O!x & □O!y & □□∀F ([F]x ≡ [F]y)›
    by (metis "S5Basic:6" "&I" "&E"(1) "&E"(2) "≡E"(4)
              "oa-facts:1" "raa-cor:3" "vdash-properties:10")
  AOT_hence ‹□(O!x & O!y & □∀F ([F]x ≡ [F]y))›
    by (metis "&E"(1) "&E"(2) "≡E"(2) "KBasic:3" "&I")
  AOT_thus ‹□(x =E y)›
    using "=E-simple:1"
    by (AOT_subst ‹x =E y› ‹O!x & O!y & □∀F ([F]x ≡ [F]y)›) auto
next
  AOT_assume ‹□(x =E y)›
  AOT_thus ‹x =E y› using "qml:2"[axiom_inst, THEN "→E"] by blast
qed

AOT_theorem "id-nec3:2": ‹◇(x =E y) ≡ x =E y›                      (231.2)
  by (meson "RE◇" "S5Basic:2" "id-nec3:1" "≡E"(1,5) "Commutativity of ≡")

AOT_theorem "id-nec3:3": ‹◇(x =E y) ≡ □(x =E y)›                   (231.3)
  by (meson "id-nec3:1" "id-nec3:2" "≡E"(5))

syntax "_AOT_non_eq_E" :: ‹Π› ("'(≠E')")
translations
  (Π) "(≠E)" == (Π) "(=E)-"
syntax "_AOT_non_eq_E_infix" :: ‹τ ⇒ τ ⇒ φ› (infixl "≠E" 50)
translations
 "_AOT_non_eq_E_infix κ κ'" ==
 "CONST AOT_exe (CONST relation_negation (CONST eq_E)) (CONST Pair κ κ')"
print_translation‹
AOT_syntax_print_translations
[(const_syntax‹AOT_exe›, fn ctxt => fn [
  Const (const_syntax‹relation_negation›, _) $ Const (const_name‹eq_E›, _),
  Const (const_syntax‹Pair›, _) $ lhs $ rhs
] => Const (syntax_const‹_AOT_non_eq_E_infix›, dummyT) $ lhs $ rhs)]›
AOT_theorem "thm-neg=E": ‹x ≠E y ≡ ¬(x =E y)›                      (233)
proof -
  AOT_have ϑ: ‹[λx1...x2 ¬(=E)x1...x2]↓› by "cqt:2"
  AOT_have ‹x ≠E y ≡ [λx1...x2 ¬(=E)x1...x2]xy›
    by (rule "=dfI"(1)[OF "df-relation-negation", OF ϑ])
       (meson "oth-class-taut:3:a")
  also AOT_have ‹... ≡ ¬(=E)xy›
    by (safe intro!: "beta-C-meta"[THEN "→E", unvarify ν1νn] "cqt:2"
                     tuple_denotes[THEN "≡dfI"] "&I")
  finally show ?thesis.
qed

AOT_theorem "id-nec4:1": ‹x ≠E y ≡ □(x ≠E y)›                      (234.1)
proof -
  AOT_have ‹x ≠E y ≡ ¬(x =E y)› using "thm-neg=E".
  also AOT_have ‹... ≡ ¬◇(x =E y)›
    by (meson "id-nec3:2" "≡E"(1) "Commutativity of ≡" "oth-class-taut:4:b")
  also AOT_have ‹... ≡ □¬(x =E y)›
    by (meson "KBasic2:1" "≡E"(2) "Commutativity of ≡")
  also AOT_have ‹... ≡ □(x ≠E y)›
    by (AOT_subst (reverse) ‹¬(x =E y)› ‹x ≠E y›)
       (auto simp: "thm-neg=E" "oth-class-taut:3:a")
  finally show ?thesis.
qed

AOT_theorem "id-nec4:2": ‹◇(x ≠E y) ≡ (x ≠E y)›                    (234.2)
  by (meson "RE◇" "S5Basic:2" "id-nec4:1" "≡E"(2,5) "Commutativity of ≡")

AOT_theorem "id-nec4:3": ‹◇(x ≠E y) ≡ □(x ≠E y)›                   (234.3)
```

```
  by (meson "id-nec4:1" "id-nec4:2" "≡E"(5))

AOT_theorem "id-act2:1": ‹x =E y ≡ 𝒜x =E y›                          (235.1)
  by (meson "Act-Basic:5" "Act-Sub:2" "RA[2]" "id-nec3:2" "≡E"(1,6))
AOT_theorem "id-act2:2": ‹x ≠E y ≡ 𝒜x ≠E y›                          (235.2)
  by (meson "Act-Basic:5" "Act-Sub:2" "RA[2]" "id-nec4:2" "≡E"(1,6))

AOT_theorem "ord=Eequiv:1": ‹O!x → x =E x›                            (236.1)
proof (rule "→I")
  AOT_assume 1: ‹O!x›
  AOT_show ‹x =E x›
    apply (rule "=dfI"(2)[OF "=E"]) apply "cqt:2[lambda]"
    apply (rule "β←C"(1))
      apply "cqt:2[lambda]"
     apply (simp add: "&I" "cqt:2[const_var]"[axiom_inst] prod_denotesI)
    by (simp add: "1" RN "&I" "oth-class-taut:3:a" "universal-cor")
qed

AOT_theorem "ord=Eequiv:2": ‹x =E y → y =E x›                         (236.2)
proof(rule CP)
  AOT_assume 1: ‹x =E y›
  AOT_hence 2: ‹x = y› by (metis "=E-simple:2" "vdash-properties:10")
  AOT_have ‹O!x› using 1 by (meson "&E"(1) "=E-simple:1" "≡E"(1))
  AOT_hence ‹x =E x› using "ord=Eequiv:1" "→E" by blast
  AOT_thus ‹y =E x› using "rule=E"[rotated, OF 2] by fast
qed

AOT_theorem "ord=Eequiv:3": ‹(x =E y & y =E z) → x =E z›               (236.3)
proof (rule CP)
  AOT_assume 1: ‹x =E y & y =E z›
  AOT_hence ‹x = y & y = z›
    by (metis "&I" "&E"(1) "&E"(2) "=E-simple:2" "vdash-properties:6")
  AOT_hence ‹x = z› by (metis "id-eq:3" "vdash-properties:6")
  moreover AOT_have ‹x =E x›
    using 1[THEN "&E"(1)] "&E"(1) "=E-simple:1" "≡E"(1)
          "ord=Eequiv:1" "→E" by blast
  ultimately AOT_show ‹x =E z›
    using "rule=E" by fast
qed

AOT_theorem "ord-=E=:1": ‹(O!x ∨ O!y) → □(x = y ≡ x =E y)›            (237.1)
proof(rule CP)
  AOT_assume ‹O!x ∨ O!y›
  moreover {
    AOT_assume ‹O!x›
    AOT_hence ‹□O!x› by (metis "oa-facts:1" "vdash-properties:10")
    moreover {
      AOT_modally_strict {
        AOT_have ‹O!x → (x = y ≡ x =E y)›
        proof (rule "→I"; rule "≡I"; rule "→I")
          AOT_assume ‹O!x›
          AOT_hence ‹x =E x› by (metis "ord=Eequiv:1" "→E")
          moreover AOT_assume ‹x = y›
          ultimately AOT_show ‹x =E y› using "rule=E" by fast
        next
          AOT_assume ‹x =E y›
          AOT_thus ‹x = y› by (metis "=E-simple:2" "→E")
        qed
      }
      AOT_hence ‹□O!x → □(x = y ≡ x =E y)› by (metis "RM:1")
    }
    ultimately AOT_have ‹□(x = y ≡ x =E y)› using "→E" by blast
  }
```

```
  moreover {
    AOT_assume ‹O!y›
    AOT_hence ‹□O!y› by (metis "oa-facts:1" "vdash-properties:10")
    moreover {
      AOT_modally_strict {
        AOT_have ‹O!y → (x = y ≡ x =E y)›
        proof (rule "→I"; rule "≡I"; rule "→I")
          AOT_assume ‹O!y›
          AOT_hence ‹y =E y› by (metis "ord=Eequiv:1" "→E")
          moreover AOT_assume ‹x = y›
          ultimately AOT_show ‹x =E y› using "rule=E" id_sym by fast
        next
          AOT_assume ‹x =E y›
          AOT_thus ‹x = y› by (metis "=E-simple:2" "→E")
        qed
      }
      AOT_hence ‹□O!y → □(x = y ≡ x =E y)› by (metis "RM:1")
    }
    ultimately AOT_have ‹□(x = y ≡ x =E y)› using "→E" by blast
  }
  ultimately AOT_show ‹□(x = y ≡ x =E y)› by (metis "∨E"(3) "raa-cor:1")
qed

AOT_theorem "ord-=E=:2": ‹O!y → [λx x = y]↓›                          (237.2)
proof (rule "→I"; rule "safe-ext"[axiom_inst, THEN "→E"]; rule "&I")
  AOT_show ‹[λx x =E y]↓› by "cqt:2[lambda]"
next
  AOT_assume ‹O!y›
  AOT_hence 1: ‹□(x = y ≡ x =E y)› for x
    using "ord-=E=:1" "→E" "∨I" by blast
  AOT_have ‹□(x =E y ≡ x = y)› for x
    by (AOT_subst ‹x =E y ≡ x = y› ‹x = y ≡ x =E y›)
       (auto simp add: "Commutativity of ≡" 1)
  AOT_hence ‹∀x □(x =E y ≡ x = y)› by (rule GEN)
  AOT_thus ‹□∀x (x =E y ≡ x = y)› by (rule BF[THEN "→E"])
qed


AOT_theorem "ord-=E=:3": ‹[λxy O!x & O!y & x = y]↓›                    (237.3)
proof (rule "safe-ext[2]"[axiom_inst, THEN "→E"]; rule "&I")
  AOT_show ‹[λxy O!x & O!y & x =E y]↓› by "cqt:2[lambda]"
next
  AOT_show ‹□∀x∀y ([O!]x & [O!]y & x =E y ≡ [O!]x & [O!]y & x = y)›
  proof (rule RN; rule GEN; rule GEN; rule "≡I"; rule "→I")
    AOT_modally_strict {
      AOT_show ‹[O!]x & [O!]y & x = y› if ‹[O!]x & [O!]y & x =E y› for x y
        by (metis "&I" "&E"(1) "Conjunction Simplification"(2) "=E-simple:2"
                  "modus-tollens:1" "raa-cor:1" that)
    }
  next
    AOT_modally_strict {
      AOT_show ‹[O!]x & [O!]y & x =E y› if ‹[O!]x & [O!]y & x = y› for x y
        apply(safe intro!: "&I")
          apply (metis that[THEN "&E"(1), THEN "&E"(1)])
         apply (metis that[THEN "&E"(1), THEN "&E"(2)])
        using "rule=E"[rotated, OF that[THEN "&E"(2)]]
              "ord=Eequiv:1"[THEN "→E", OF that[THEN "&E"(1), THEN "&E"(1)]]
        by fast
    }
  qed
qed

AOT_theorem "ind-nec": ‹∀F ([F]x ≡ [F]y) → □∀F ([F]x ≡ [F]y)›         (238)
```

```
proof(rule "→I")
  AOT_assume ‹∀F ([F]x ≡ [F]y)›
  moreover AOT_have ‹[λx □∀F ([F]x ≡ [F]y)]↓› by "cqt:2[lambda]"
  ultimately AOT_have ‹[λx □∀F ([F]x ≡ [F]y)]x ≡ [λx □∀F ([F]x ≡ [F]y)]y›
    using "∀E" by blast
  moreover AOT_have ‹[λx □∀F ([F]x ≡ [F]y)]y›
    apply (rule "β←C"(1))
      apply "cqt:2[lambda]"
     apply (fact "cqt:2[const_var]"[axiom_inst])
    by (simp add: RN GEN "oth-class-taut:3:a")
  ultimately AOT_have ‹[λx □∀F ([F]x ≡ [F]y)]x› using "≡E" by blast
  AOT_thus ‹□∀F ([F]x ≡ [F]y)›
    using "β→C"(1) by blast
qed

AOT_theorem "ord=E:1": ‹(O!x & O!y) → (∀F ([F]x ≡ [F]y) → x =E y)›          (239.1)
proof (rule "→I"; rule "→I")
  AOT_assume ‹∀F ([F]x ≡ [F]y)›
  AOT_hence ‹□∀F ([F]x ≡ [F]y)›
    using "ind-nec"[THEN "→E"] by blast
  moreover AOT_assume ‹O!x & O!y›
  ultimately AOT_have ‹O!x & O!y & □∀F ([F]x ≡ [F]y)›
    using "&I" by blast
  AOT_thus ‹x =E y› using "=E-simple:1"[THEN "≡E"(2)] by blast
qed

AOT_theorem "ord=E:2": ‹(O!x & O!y) → (∀F ([F]x ≡ [F]y) → x = y)›           (239.2)
proof (rule "→I"; rule "→I")
  AOT_assume ‹O!x & O!y›
  moreover AOT_assume ‹∀F ([F]x ≡ [F]y)›
  ultimately AOT_have ‹x =E y›
    using "ord=E:1" "→E" by blast
  AOT_thus ‹x = y› using "=E-simple:2"[THEN "→E"] by blast
qed

AOT_theorem "ord=E2:1":                                                      (240.1)
  ‹(O!x & O!y) → (x ≠ y ≡ [λz z =E x] ≠ [λz z =E y])›
proof (rule "→I"; rule "≡I"; rule "→I";
        rule "≡dfI"[OF "=-infix"]; rule "raa-cor:2")
  AOT_assume 0: ‹O!x & O!y›
  AOT_assume ‹x ≠ y›
  AOT_hence 1: ‹¬(x = y)› using "≡dfE"[OF "=-infix"] by blast
  AOT_assume ‹[λz z =E x] = [λz z =E y]›
  moreover AOT_have ‹[λz z =E x]x›
    apply (rule "β←C"(1))
      apply "cqt:2[lambda]"
     apply (fact "cqt:2[const_var]"[axiom_inst])
    using "ord=Eequiv:1"[THEN "→E", OF 0[THEN "&E"(1)]].
  ultimately AOT_have ‹[λz z =E y]x› using "rule=E" by fast
  AOT_hence ‹x =E y› using "β→C"(1) by blast
  AOT_hence ‹x = y› by (metis "=E-simple:2" "vdash-properties:6")
  AOT_thus ‹x = y & ¬(x = y)› using 1 "&I" by blast
next
  AOT_assume ‹[λz z =E x] ≠ [λz z =E y]›
  AOT_hence 0: ‹¬([λz z =E x] = [λz z =E y])›
    using "≡dfE"[OF "=-infix"] by blast
  AOT_have ‹[λz z =E x]↓› by "cqt:2[lambda]"
  AOT_hence ‹[λz z =E x] = [λz z =E x]›
    by (metis "rule=I:1")
  moreover AOT_assume ‹x = y›
  ultimately AOT_have ‹[λz z =E x] = [λz z =E y]›
    using "rule=E" by fast
  AOT_thus ‹[λz z =E x] = [λz z =E y] & ¬([λz z =E x] = [λz z =E y])›
```

```
    using 0 "&I" by blast
qed

AOT_theorem "ord=E2:2":                                                    (240.2)
  ‹(O!x & O!y) → (x ≠ y ≡ [λz z = x] ≠ [λz z = y])›
proof (rule "→I"; rule "≡I"; rule "→I";
       rule "≡dfI"[OF "=-infix"]; rule "raa-cor:2")
  AOT_assume 0: ‹O!x & O!y›
  AOT_assume ‹x ≠ y›
  AOT_hence 1: ‹¬(x = y)› using "≡dfE"[OF "=-infix"] by blast
  AOT_assume ‹[λz z = x] = [λz z = y]›
  moreover AOT_have ‹[λz z = x]x›
    apply (rule "β←C"(1))
    apply (fact "ord-=E=:2"[THEN "→E", OF 0[THEN "&E"(1)]])
     apply (fact "cqt:2[const_var]"[axiom_inst])
    by (simp add: "id-eq:1")
  ultimately AOT_have ‹[λz z = y]x› using "rule=E" by fast
  AOT_hence ‹x = y› using "β→C"(1) by blast
  AOT_thus ‹x = y & ¬(x = y)› using 1 "&I" by blast
next
  AOT_assume 0: ‹O!x & O!y›
  AOT_assume ‹[λz z = x] ≠ [λz z = y]›
  AOT_hence 1: ‹¬([λz z = x] = [λz z = y])›
    using "≡dfE"[OF "=-infix"] by blast
  AOT_have ‹[λz z = x]↓›
    by (fact "ord-=E=:2"[THEN "→E", OF 0[THEN "&E"(1)]])
  AOT_hence ‹[λz z = x] = [λz z = x]›
    by (metis "rule=I:1")
  moreover AOT_assume ‹x = y›
  ultimately AOT_have ‹[λz z = x] = [λz z = y]›
    using "rule=E" by fast
  AOT_thus ‹[λz z = x] = [λz z = y] & ¬([λz z = x] = [λz z = y])›
    using 1 "&I" by blast
qed

AOT_theorem ordnecfail: ‹O!x → □¬∃F x[F]›                                   (241)
  by (meson "RM:1" "→I" nocoder[axiom_inst] "oa-facts:1" "→E")

AOT_theorem "ab-obey:1": ‹(A!x & A!y) → (∀F (x[F] ≡ y[F]) → x = y)›        (242.1)
proof (rule "→I"; rule "→I")
  AOT_assume 1: ‹A!x & A!y›
  AOT_assume ‹∀F (x[F] ≡ y[F])›
  AOT_hence ‹x[F] ≡ y[F]› for F using "∀E" by blast
  AOT_hence ‹□(x[F] ≡ y[F])› for F by (metis "en-eq:6[1]" "≡E"(1))
  AOT_hence ‹∀F □(x[F] ≡ y[F])› by (rule GEN)
  AOT_hence ‹□∀F (x[F] ≡ y[F])› by (rule BF[THEN "→E"])
  AOT_thus ‹x = y›
    using "≡dfI"[OF "identity:1", OF "∨I"(2)] 1 "&I" by blast
qed

AOT_theorem "ab-obey:2":                                                   (242.2)
  ‹(∃F (x[F] & ¬y[F]) ∨ ∃F (y[F] & ¬x[F])) → x ≠ y›
proof (rule "→I"; rule "≡dfI"[OF "=-infix"]; rule "raa-cor:2")
  AOT_assume 1: ‹x = y›
  AOT_assume ‹∃F (x[F] & ¬y[F]) ∨ ∃F (y[F] & ¬x[F])›
  moreover {
    AOT_assume ‹∃F (x[F] & ¬y[F])›
    then AOT_obtain F where ‹x[F] & ¬y[F]›
      using "∃E"[rotated] by blast
    moreover AOT_have ‹y[F]›
      using calculation[THEN "&E"(1)] 1 "rule=E" by fast
    ultimately AOT_have ‹p & ¬p› for p
      by (metis "Conjunction Simplification"(2) "modus-tollens:2" "raa-cor:3")
```

```
  }
  moreover {
    AOT_assume ‹∃F (y[F] & ¬x[F])›
    then AOT_obtain F where ‹y[F] & ¬x[F]›
      using "∃E"[rotated] by blast
    moreover AOT_have ‹¬y[F]›
      using calculation[THEN "&E"(2)] 1 "rule=E" by fast
    ultimately AOT_have ‹p & ¬p› for p
      by (metis "Conjunction Simplification"(1) "modus-tollens:1" "raa-cor:3")
  }
  ultimately AOT_show ‹p & ¬p› for p
    by (metis "∨E"(3) "raa-cor:1")
qed

AOT_theorem "encoders-are-abstract": ‹∃F x[F] → A!x›                    (243)
  by (meson "deduction-theorem" "≡E"(2) "modus-tollens:2" nocoder
            "oa-contingent:3" "vdash-properties:1[2]")

AOT_theorem "denote=:1": ‹∀H∃x x[H]›                                    (244.1)
  by (rule GEN; rule "existence:2[1]"[THEN "≡dfE"]; "cqt:2")

AOT_theorem "denote=:2": ‹∀G∃x1...∃xn x1...xn[H]›                       (244.2)
  by (rule GEN; rule "existence:2"[THEN "≡dfE"]; "cqt:2")

AOT_theorem "denote=:2[2]": ‹∀G∃x1∃x2 x1x2[H]›                          (244.2)
  by (rule GEN; rule "existence:2[2]"[THEN "≡dfE"]; "cqt:2")

AOT_theorem "denote=:2[3]": ‹∀G∃x1∃x2∃x3 x1x2x3[H]›                     (244.2)
  by (rule GEN; rule "existence:2[3]"[THEN "≡dfE"]; "cqt:2")

AOT_theorem "denote=:2[4]": ‹∀G∃x1∃x2∃x3∃x4 x1x2x3x4[H]›                (244.2)
  by (rule GEN; rule "existence:2[4]"[THEN "≡dfE"]; "cqt:2")

AOT_theorem "denote=:3": ‹∃x x[Π] ≡ ∃H (H = Π)›                         (244.3)
  using "existence:2[1]" "free-thms:1" "≡E"(2,5)
        "Commutativity of ≡" "≡Df" by blast

AOT_theorem "denote=:4": ‹(∃x1...∃xn x1...xn[Π]) ≡ ∃H (H = Π)›          (244.4)
  using "existence:2" "free-thms:1" "≡E"(6) "≡Df" by blast

AOT_theorem "denote=:4[2]": ‹(∃x1∃x2 x1x2[Π]) ≡ ∃H (H = Π)›             (244.4)
  using "existence:2[2]" "free-thms:1" "≡E"(6) "≡Df" by blast

AOT_theorem "denote=:4[3]": ‹(∃x1∃x2∃x3 x1x2x3[Π]) ≡ ∃H (H = Π)›        (244.4)
  using "existence:2[3]" "free-thms:1" "≡E"(6) "≡Df" by blast

AOT_theorem "denote=:4[4]": ‹(∃x1∃x2∃x3∃x4 x1x2x3x4[Π]) ≡ ∃H (H = Π)›   (244.4)
  using "existence:2[4]" "free-thms:1" "≡E"(6) "≡Df" by blast

AOT_theorem "A-objects!": ‹∃!x (A!x & ∀F (x[F] ≡ φ{F}))›               (247)
proof (rule "uniqueness:1"[THEN "≡dfI"])
  AOT_obtain a where a_prop: ‹A!a & ∀F (a[F] ≡ φ{F})›
    using "A-objects"[axiom_inst] "∃E"[rotated] by blast
  AOT_have ‹(A!β & ∀F (β[F] ≡ φ{F})) → β = a› for β
  proof (rule "→I")
    AOT_assume β_prop: ‹[A!]β & ∀F (β[F] ≡ φ{F})›
    AOT_hence ‹β[F] ≡ φ{F}› for F
      using "∀E" "&E" by blast
    AOT_hence ‹β[F] ≡ a[F]› for F
      using a_prop[THEN "&E"(2)] "∀E" "≡E"(2,5)
            "Commutativity of ≡" by fast
    AOT_hence ‹∀F (β[F] ≡ a[F])› by (rule GEN)
    AOT_thus ‹β = a›
```

```
    using "ab-obey:1"[THEN "→E",
                OF "&I"[OF β_prop[THEN "&E"(1)], OF a_prop[THEN "&E"(1)]],
                THEN "→E"] by blast
  qed
  AOT_hence ‹∀β ((A!β & ∀F (β[F] ≡ φ{F})) → β = a)› by (rule GEN)
  AOT_thus ‹∃α ([A!]α & ∀F (α[F] ≡ φ{F}) &
                ∀β ([A!]β & ∀F (β[F] ≡ φ{F}) → β = α))›
    using "∃I" using a_prop "&I" by fast
qed

AOT_theorem "obj-oth:1": ‹∃!x (A!x & ∀F (x[F] ≡ [F]y))›                    (248.1)
  using "A-objects!" by fast

AOT_theorem "obj-oth:2": ‹∃!x (A!x & ∀F (x[F] ≡ [F]y & [F]z))›             (248.2)
  using "A-objects!" by fast

AOT_theorem "obj-oth:3": ‹∃!x (A!x & ∀F (x[F] ≡ [F]y ∨ [F]z))›             (248.3)
  using "A-objects!" by fast

AOT_theorem "obj-oth:4": ‹∃!x (A!x & ∀F (x[F] ≡ □[F]y))›                   (248.4)
  using "A-objects!" by fast

AOT_theorem "obj-oth:5": ‹∃!x (A!x & ∀F (x[F] ≡ F = G))›                   (248.5)
  using "A-objects!" by fast

AOT_theorem "obj-oth:6": ‹∃!x (A!x & ∀F (x[F] ≡ □∀y([G]y → [F]y)))›        (248.6)
  using "A-objects!" by fast

AOT_theorem "A-descriptions": ‹ιx (A!x & ∀F (x[F] ≡ φ{F}))↓›               (249)
  by (rule "A-Exists:2"[THEN "≡E"(2)]; rule "RA[2]"; rule "A-objects!")

AOT_act_theorem "thm-can-terms2":                                          (251)
  ‹y = ιx(A!x & ∀F (x[F] ≡ φ{F})) → (A!y & ∀F (y[F] ≡ φ{F}))›
  using "y-in:2" by blast

AOT_theorem "can-ab2": ‹y = ιx(A!x & ∀F (x[F] ≡ φ{F})) →  A!y›             (252)
proof(rule "→I")
  AOT_assume ‹y = ιx(A!x & ∀F (x[F] ≡ φ{F}))›
  AOT_hence ‹𝒜(A!y & ∀F (y[F] ≡ φ{F}))›
    using "actual-desc:2"[THEN "→E"] by blast
  AOT_hence ‹𝒜A!y› by (metis "Act-Basic:2" "&E"(1) "≡E"(1))
  AOT_thus ‹A!y› by (metis "≡E"(2) "oa-facts:8")
qed

AOT_act_theorem "desc-encode:1": ‹ιx(A!x & ∀F (x[F] ≡ φ{F}))[F] ≡ φ{F}›     (253.1)
proof -
  AOT_have ‹ιx(A!x & ∀F (x[F] ≡ φ{F}))↓›
    by (simp add: "A-descriptions")
  AOT_hence ‹A!ιx(A!x & ∀F (x[F] ≡ φ{F})) &
             ∀F(ιx(A!x & ∀F (x[F] ≡ φ{F}))[F] ≡ φ{F})›
    using "y-in:3"[THEN "→E"] by blast
  AOT_thus ‹ιx(A!x & ∀F (x[F] ≡ φ{F}))[F] ≡ φ{F}›
    using "&E" "∀E" by blast
qed

AOT_act_theorem "desc-encode:2": ‹ιx(A!x & ∀F (x[F] ≡ φ{F}))[G] ≡ φ{G}›     (253.2)
  using "desc-encode:1".

AOT_theorem "desc-nec-encode:1":                                           (255.1)
  ‹ιx (A!x & ∀F (x[F] ≡ φ{F}))[F] ≡ 𝒜φ{F}›
proof -
  AOT_have 0: ‹ιx(A!x & ∀F (x[F] ≡ φ{F}))↓›
    by (simp add: "A-descriptions")
```

```
  AOT_hence ‹𝒜(A!ιx(A!x & ∀F (x[F] ≡ φ{F})) &
             ∀F(ιx(A!x & ∀F (x[F] ≡ φ{F}))[F] ≡ φ{F}))›
    using "actual-desc:4"[THEN "→E"] by blast
  AOT_hence ‹𝒜∀F (ιx(A!x & ∀F (x[F] ≡ φ{F}))[F] ≡ φ{F})›
    using "Act-Basic:2" "&E"(2) "≡E"(1) by blast
  AOT_hence ‹∀F 𝒜(ιx(A!x & ∀F (x[F] ≡ φ{F}))[F] ≡ φ{F})›
    using "≡E"(1) "logic-actual-nec:3" "vdash-properties:1[2]" by blast
  AOT_hence ‹𝒜(ιx(A!x & ∀F (x[F] ≡ φ{F}))[F] ≡ φ{F})›
    using "∀E" by blast
  AOT_hence ‹𝒜ιx(A!x & ∀F (x[F] ≡ φ{F}))[F] ≡ 𝒜φ{F}›
    using "Act-Basic:5" "≡E"(1) by blast
  AOT_thus ‹ιx(A!x & ∀F (x[F] ≡ φ{F}))[F] ≡ 𝒜φ{F}›
    using "en-eq:10[1]"[unvarify x₁, OF 0] "≡E"(6) by blast
qed

AOT_theorem "desc-nec-encode:2":                                                    (255.2)
  ‹ιx (A!x & ∀F (x[F] ≡ φ{F}))[G] ≡ 𝒜φ{G}›
  using "desc-nec-encode:1".

AOT_theorem "Box-desc-encode:1": ‹□φ{G} → ιx(A!x & ∀F (x[F] ≡ φ{G}))[G]›          (256.1)
  by (rule "→I"; rule "desc-nec-encode:2"[THEN "≡E"(2)])
     (meson "nec-imp-act" "vdash-properties:10")

AOT_theorem "Box-desc-encode:2":                                                    (256.2)
‹□φ{G} → □(ιx(A!x & ∀F (x[F] ≡ φ{G}))[G] ≡ φ{G})›
proof(rule CP)
  AOT_assume ‹□φ{G}›
  AOT_hence ‹□□φ{G}› by (metis "S5Basic:6" "≡E"(1))
  moreover AOT_have ‹□□φ{G} → □(ιx(A!x & ∀F (x[F] ≡ φ{G}))[G] ≡ φ{G})›
  proof (rule RM; rule "→I")
    AOT_modally_strict {
      AOT_assume 1: ‹□φ{G}›
      AOT_hence ‹ιx(A!x & ∀F (x[F] ≡ φ{G}))[G]›
        using "Box-desc-encode:1" "→E" by blast
      moreover AOT_have ‹φ{G}›
        using 1 by (meson "qml:2"[axiom_inst] "→E")
      ultimately AOT_show ‹ιx(A!x & ∀F (x[F] ≡ φ{G}))[G] ≡ φ{G}›
        using "→I" "≡I" by simp
    }
  qed
  ultimately AOT_show ‹□(ιx(A!x & ∀F (x[F] ≡ φ{G}))[G] ≡ φ{G})›
    using "→E" by blast
qed

definition rigid_condition where
  ‹rigid_condition φ ≡ ∀v . [v ⊨ ∀α (φ{α} → □φ{α})]›
syntax rigid_condition :: ‹id_position ⇒ AOT_prop› ("RIGID'_CONDITION'(_')")

AOT_theorem "strict-can:1[E]":                                                      (257.1)
  assumes ‹RIGID_CONDITION(φ)›
  shows ‹∀α (φ{α} → □φ{α})›
  using assms[unfolded rigid_condition_def] by auto

AOT_theorem "strict-can:1[I]":                                                      (257.1)
  assumes ‹⊢□ ∀α (φ{α} → □φ{α})›
  shows ‹RIGID_CONDITION(φ)›
  using assms rigid_condition_def by auto

AOT_theorem "box-phi-a:1":                                                          (258.1)
  assumes ‹RIGID_CONDITION(φ)›
  shows ‹(A!x  & ∀F (x[F] ≡ φ{F})) → □(A!x & ∀F (x[F] ≡ φ{F}))›
proof (rule "→I")
  AOT_assume a: ‹A!x & ∀F (x[F] ≡ φ{F})›
```

```
  AOT_hence b: ‹□A!x›
    by (metis "Conjunction Simplification"(1) "oa-facts:2" "→E")
  AOT_have ‹x[F] ≡ φ{F}› for F
    using a[THEN "&E"(2)] "∀E" by blast
  moreover AOT_have ‹□(x[F] → □x[F])› for F
    by (meson "pre-en-eq:1[1]" RN)
  moreover AOT_have ‹□(φ{F} → □φ{F})› for F
    using RN "strict-can:1[E]"[OF assms] "∀E" by blast
  ultimately AOT_have ‹□(x[F] ≡ φ{F})› for F
    using "sc-eq-box-box:5" "qml:2"[axiom_inst, THEN "→E"] "→E" "&I" by metis
  AOT_hence ‹∀F □(x[F] ≡ φ{F})› by (rule GEN)
  AOT_hence ‹□∀F (x[F] ≡ φ{F})› by (rule BF[THEN "→E"])
  AOT_thus ‹□([A!]x & ∀F (x[F] ≡ φ{F}))›
    using b "KBasic:3" "≡S"(1) "≡E"(2) by blast
qed

AOT_theorem "box-phi-a:2":                                                    (258.2)
  assumes ‹RIGID_CONDITION(φ)›
  shows ‹y = ιx(A!x & ∀F (x[F] ≡ φ{F})) → (A!y & ∀F (y[F] ≡ φ{F}))›
proof(rule "→I")
  AOT_assume ‹y = ιx(A!x & ∀F (x[F] ≡ φ{F}))›
  AOT_hence ‹𝒜(A!y & ∀F (y[F] ≡ φ{F}))›
    using "actual-desc:2"[THEN "→E"] by fast
  AOT_hence abs: ‹𝒜A!y› and ‹𝒜∀F (y[F] ≡ φ{F})›
    using "Act-Basic:2" "&E" "≡E"(1) by blast+
  AOT_hence ‹∀F 𝒜(y[F] ≡ φ{F})›
    by (metis "≡E"(1) "logic-actual-nec:3" "vdash-properties:1[2]")
  AOT_hence ‹𝒜(y[F] ≡ φ{F})› for F
    using "∀E" by blast
  AOT_hence ‹𝒜y[F] ≡ 𝒜φ{F}› for F
    by (metis "Act-Basic:5" "≡E"(1))
  AOT_hence ‹y[F] ≡ φ{F}› for F
    using "sc-eq-fur:2"[THEN "→E",
            OF "strict-can:1[E]"[OF assms,
                THEN "∀E"(2)[where β=F], THEN RN]]
    by (metis "en-eq:10[1]" "≡E"(6))
  AOT_hence ‹∀F (y[F] ≡ φ{F})› by (rule GEN)
  AOT_thus ‹[A!]y & ∀F (y[F] ≡ φ{F})›
    using abs "&I" "≡E"(2) "oa-facts:8" by blast
qed

AOT_theorem "box-phi-a:3":                                                    (258.3)
  assumes ‹RIGID_CONDITION(φ)›
  shows ‹ιx(A!x & ∀F (x[F] ≡ φ{F}))[F] ≡ φ{F}›
  using "desc-nec-encode:2"
    "sc-eq-fur:2"[THEN "→E",
        OF "strict-can:1[E]"[OF assms,
          THEN "∀E"(2)[where β=F], THEN RN]]
    "≡E"(5) by blast

AOT_define Null :: ‹τ ⇒ φ› ("Null'(_')")
  "df-null-uni:1": ‹Null(x) ≡df A!x & ¬∃F x[F]›                               (260.1)

AOT_define Universal :: ‹τ ⇒ φ› ("Universal'(_')")
  "df-null-uni:2": ‹Universal(x) ≡df A!x & ∀F x[F]›                           (260.2)

AOT_theorem "null-uni-uniq:1": ‹∃!x Null(x)›                                   (261.1)
proof (rule "uniqueness:1"[THEN "≡dfI"])
  AOT_obtain a where a_prop: ‹A!a & ∀F (a[F] ≡ ¬(F = F))›
    using "A-objects"[axiom_inst] "∃E"[rotated] by fast
  AOT_have a_null: ‹¬a[F]› for F
  proof (rule "raa-cor:2")
    AOT_assume ‹a[F]›
```

```
    AOT_hence ‹¬(F = F)› using a_prop[THEN "&E"(2)] "∀E" "≡E" by blast
    AOT_hence ‹F = F & ¬(F = F)› by (metis "id-eq:1" "raa-cor:3")
    AOT_thus ‹p & ¬p› for p  by (metis "raa-cor:1")
  qed
  AOT_have ‹Null(a) & ∀β (Null(β) → β = a)›
  proof (rule "&I")
    AOT_have ‹¬∃F a[F]›
      using a_null by (metis "instantiation" "reductio-aa:1")
    AOT_thus ‹Null(a)›
      using "df-null-uni:1"[THEN "≡dfI"] a_prop[THEN "&E"(1)] "&I" by metis
  next
    AOT_show ‹∀β (Null(β) → β = a)›
    proof (rule GEN; rule "→I")
      fix β
      AOT_assume a: ‹Null(β)›
      AOT_hence ‹¬∃F β[F]›
        using "df-null-uni:1"[THEN "≡dfE"] "&E" by blast
      AOT_hence β_null: ‹¬β[F]› for F
        by (metis "existential:2[const_var]" "reductio-aa:1")
      AOT_have ‹∀F (β[F] ≡ a[F])›
        apply (rule GEN; rule "≡I"; rule CP)
        using "raa-cor:3" β_null a_null by blast+
      moreover AOT_have ‹A!β›
        using a "df-null-uni:1"[THEN "≡dfE"] "&E" by blast
      ultimately AOT_show ‹β = a›
        using a_prop[THEN "&E"(1)] "ab-obey:1"[THEN "→E", THEN "→E"]
              "&I" by blast
    qed
  qed
  AOT_thus ‹∃α (Null(α) & ∀β (Null(β) → β = α))›
    using "∃I"(2) by fast
qed

AOT_theorem "null-uni-uniq:2": ‹∃!x Universal(x)›                    (261.2)
proof (rule "uniqueness:1"[THEN "≡dfI"])
  AOT_obtain a where a_prop: ‹A!a & ∀F (a[F] ≡ F = F)›
    using "A-objects"[axiom_inst] "∃E"[rotated] by fast
  AOT_hence aF: ‹a[F]› for F using "&E" "∀E" "≡E" "id-eq:1" by fast
  AOT_hence ‹Universal(a)›
    using "df-null-uni:2"[THEN "≡dfI"] "&I" a_prop[THEN "&E"(1)] GEN by blast
  moreover AOT_have ‹∀β (Universal(β) → β = a)›
  proof (rule GEN; rule "→I")
    fix β
    AOT_assume ‹Universal(β)›
    AOT_hence abs_β: ‹A!β› and ‹β[F]› for F
      using "df-null-uni:2"[THEN "≡dfE"] "&E" "∀E" by blast+
    AOT_hence ‹β[F] ≡ a[F]› for F
      using aF by (metis "deduction-theorem" "≡I")
    AOT_hence ‹∀F (β[F] ≡ a[F])› by (rule GEN)
    AOT_thus ‹β = a›
      using a_prop[THEN "&E"(1)] "ab-obey:1"[THEN "→E", THEN "→E"]
            "&I" abs_β by blast
  qed
  ultimately AOT_show ‹∃α (Universal(α) & ∀β (Universal(β) → β = α))›
    using "&I" "∃I" by fast
qed

AOT_theorem "null-uni-uniq:3": ‹ιx Null(x)↓›                          (261.3)
  using "A-Exists:2" "RA[2]" "≡E"(2) "null-uni-uniq:1" by blast

AOT_theorem "null-uni-uniq:4": ‹ιx Universal(x)↓›                     (261.4)
  using "A-Exists:2" "RA[2]" "≡E"(2) "null-uni-uniq:2" by blast
```

```
AOT_define Null_object :: ‹κ_s› (‹a_∅›)
  "df-null-uni-terms:1": ‹a_∅ =_df ιx Null(x)›                                          (262.1)

AOT_define Universal_object :: ‹κ_s› (‹a_V›)
  "df-null-uni-terms:2": ‹a_V =_df ιx Universal(x)›                                     (262.2)

AOT_theorem "null-uni-facts:1": ‹Null(x) → □Null(x)›                                  (263.1)
proof (rule "→I")
  AOT_assume ‹Null(x)›
  AOT_hence x_abs: ‹A!x› and x_null: ‹¬∃F x[F]›
    using "df-null-uni:1"[THEN "≡_dfE"] "&E" by blast+
  AOT_have ‹¬x[F]› for F using x_null
    using "existential:2[const_var]" "reductio-aa:1"
    by metis
  AOT_hence ‹□¬x[F]› for F by (metis "en-eq:7[1]" "≡E"(1))
  AOT_hence ‹∀F □¬x[F]› by (rule GEN)
  AOT_hence ‹□∀F ¬x[F]› by (rule BF[THEN "→E"])
  moreover AOT_have ‹□∀F ¬x[F] → □¬∃F x[F]›
    apply (rule RM)
    by (metis (full_types) "instantiation" "cqt:2[const_var]"[axiom_inst]
              "→I" "reductio-aa:1" "rule-ui:1")
  ultimately AOT_have ‹□¬∃F x[F]›
    by (metis "→E")
  moreover AOT_have ‹□A!x› using x_abs
    using "oa-facts:2" "vdash-properties:10" by blast
  ultimately AOT_have r: ‹□(A!x & ¬∃F x[F])›
    by (metis "KBasic:3" "&I" "≡E"(3) "raa-cor:3")
  AOT_show ‹□Null(x)›
    by (AOT_subst ‹Null(x)› ‹A!x & ¬∃F x[F]›)
       (auto simp: "df-null-uni:1" "≡Df" r)
qed

AOT_theorem "null-uni-facts:2": ‹Universal(x) → □Universal(x)›                       (263.2)
proof (rule "→I")
  AOT_assume ‹Universal(x)›
  AOT_hence x_abs: ‹A!x› and x_univ: ‹∀F x[F]›
    using "df-null-uni:2"[THEN "≡_dfE"] "&E" by blast+
  AOT_have ‹x[F]› for F using x_univ "∀E" by blast
  AOT_hence ‹□x[F]› for F by (metis "en-eq:2[1]" "≡E"(1))
  AOT_hence ‹∀F □x[F]› by (rule GEN)
  AOT_hence ‹□∀F x[F]› by (rule BF[THEN "→E"])
  moreover AOT_have ‹□A!x› using x_abs
    using "oa-facts:2" "vdash-properties:10" by blast
  ultimately AOT_have r: ‹□(A!x & ∀F x[F])›
    by (metis "KBasic:3" "&I" "≡E"(3) "raa-cor:3")
  AOT_show ‹□Universal(x)›
    by (AOT_subst ‹Universal(x)› ‹A!x & ∀F x[F]›)
       (auto simp add: "df-null-uni:2" "≡Df" r)
qed

AOT_theorem "null-uni-facts:3": ‹Null(a_∅)›                                            (263.3)
  apply (rule "=_dfI"(2)[OF "df-null-uni-terms:1"])
   apply (simp add: "null-uni-uniq:3")
  using "actual-desc:4"[THEN "→E", OF "null-uni-uniq:3"]
    "sc-eq-fur:2"[THEN "→E",
        OF "null-uni-facts:1"[unvarify x, THEN RN, OF "null-uni-uniq:3"],
        THEN "≡E"(1)]
  by blast

AOT_theorem "null-uni-facts:4": ‹Universal(a_V)›                                       (263.4)
  apply (rule "=_dfI"(2)[OF "df-null-uni-terms:2"])
   apply (simp add: "null-uni-uniq:4")
  using "actual-desc:4"[THEN "→E", OF "null-uni-uniq:4"]
```

```
    "sc-eq-fur:2"[THEN "→E",
        OF "null-uni-facts:2"[unvarify x, THEN RN, OF "null-uni-uniq:4"],
        THEN "≡E"(1)]
  by blast

AOT_theorem "null-uni-facts:5": ‹a_∅ ≠ a_V›                                    (263.5)
proof (rule "=_dfI"(2)[OF "df-null-uni-terms:1", OF "null-uni-uniq:3"];
    rule "=_dfI"(2)[OF "df-null-uni-terms:2", OF "null-uni-uniq:4"];
    rule "≡_dfI"[OF "=-infix"];
    rule "raa-cor:2")
  AOT_obtain x where nullx: ‹Null(x)›
    by (metis "instantiation" "df-null-uni-terms:1" "existential:1"
              "null-uni-facts:3" "null-uni-uniq:3" "rule-id-df:2:b[zero]")
  AOT_hence act_null: ‹𝒜Null(x)›
    by (metis "nec-imp-act" "null-uni-facts:1" "→E")
  AOT_assume ‹ιx Null(x) = ιx Universal(x)›
  AOT_hence ‹𝒜∀x(Null(x) ≡ Universal(x))›
    using "actual-desc:5"[THEN "→E"] by blast
  AOT_hence ‹∀x 𝒜(Null(x) ≡ Universal(x))›
    by (metis "≡E"(1) "logic-actual-nec:3" "vdash-properties:1[2]")
  AOT_hence ‹𝒜Null(x) ≡ 𝒜Universal(x)›
    using "Act-Basic:5" "≡E"(1) "rule-ui:3" by blast
  AOT_hence ‹𝒜Universal(x)› using act_null "≡E" by blast
  AOT_hence ‹Universal(x)›
    by (metis RN "≡E"(1) "null-uni-facts:2" "sc-eq-fur:2" "→E")
  AOT_hence ‹∀F x[F]› using "≡_dfE"[OF "df-null-uni:2"] "&E" by metis
  moreover AOT_have ‹¬∃F x[F]›
    using nullx "≡_dfE"[OF "df-null-uni:1"] "&E" by metis
  ultimately AOT_show ‹p & ¬p› for p
    by (metis "cqt-further:1" "raa-cor:3" "→E")
qed

AOT_theorem "null-uni-facts:6": ‹a_∅ = ιx(A!x & ∀F (x[F] ≡ F ≠ F))›           (263.6)
proof (rule "ab-obey:1"[unvarify x y, THEN "→E", THEN "→E"])
  AOT_show ‹ιx([A!]x & ∀F (x[F] ≡ F ≠ F))↓›
    by (simp add: "A-descriptions")
next
  AOT_show ‹a_∅↓›
    by (rule "=_dfI"(2)[OF "df-null-uni-terms:1", OF "null-uni-uniq:3"])
       (simp add: "null-uni-uniq:3")
next
  AOT_have ‹ιx([A!]x & ∀F (x[F] ≡ F ≠ F))↓›
    by (simp add: "A-descriptions")
  AOT_hence 1: ‹ιx([A!]x & ∀F (x[F] ≡ F ≠ F)) = ιx([A!]x & ∀F (x[F] ≡ F ≠ F))›
    using "rule=I:1" by blast
  AOT_show ‹[A!]a_∅ & [A!]ιx([A!]x & ∀F (x[F] ≡ F ≠ F))›
    apply (rule "=_dfI"(2)[OF "df-null-uni-terms:1", OF "null-uni-uniq:3"];
           rule "&I")
     apply (meson "≡_dfE" "Conjunction Simplification"(1)
                  "df-null-uni:1" "df-null-uni-terms:1" "null-uni-facts:3"
                  "null-uni-uniq:3" "rule-id-df:2:a[zero]" "→E")
    using "can-ab2"[unvarify y, OF "A-descriptions", THEN "→E", OF 1].
next
  AOT_show ‹∀F (a_∅[F] ≡ ιx([A!]x & ∀F (x[F] ≡ F ≠ F))[F])›
  proof (rule GEN)
    fix F
    AOT_have ‹¬a_∅[F]›
      by (rule "=_dfI"(2)[OF "df-null-uni-terms:1", OF "null-uni-uniq:3"])
         (metis (no_types, lifting) "≡_dfE" "&E"(2) "∨I"(2) "∨E"(3) "∃I"(2)
                "df-null-uni:1" "df-null-uni-terms:1" "null-uni-facts:3"
                "raa-cor:2" "rule-id-df:2:a[zero]"
                "russell-axiom[enc,1].ψ_denotes_asm")
    moreover AOT_have ‹¬ιx([A!]x & ∀F (x[F] ≡ F ≠ F))[F]›
```

```
    proof(rule "raa-cor:2")
      AOT_assume 0: ‹ιx([A!]x & ∀F (x[F] ≡ F ≠ F))[F]›
      AOT_hence ‹𝒜(F ≠ F)›
        using "desc-nec-encode:2"[THEN "≡E"(1), OF 0] by blast
      moreover AOT_have ‹¬𝒜(F ≠ F)›
        using "≡dfE" "id-act:2" "id-eq:1" "≡E"(2)
              "=-infix" "raa-cor:3" by blast
      ultimately AOT_show ‹𝒜(F ≠ F) & ¬𝒜(F ≠ F)› by (rule "&I")
    qed
    ultimately AOT_show ‹a∅[F] ≡ ιx([A!]x & ∀F (x[F] ≡ F ≠ F))[F]›
      using "deduction-theorem" "≡I" "raa-cor:4" by blast
  qed
qed

AOT_theorem "null-uni-facts:7": ‹aV = ιx(A!x & ∀F (x[F] ≡ F = F))›          (263.7)
proof (rule "ab-obey:1"[unvarify x y, THEN "→E", THEN "→E"])
  AOT_show ‹ιx([A!]x & ∀F (x[F] ≡ F = F))↓›
    by (simp add: "A-descriptions")
next
  AOT_show ‹aV↓›
    by (rule "=dfI"(2)[OF "df-null-uni-terms:2", OF "null-uni-uniq:4"])
       (simp add: "null-uni-uniq:4")
next
  AOT_have ‹ιx([A!]x & ∀F (x[F] ≡ F = F))↓›
    by (simp add: "A-descriptions")
  AOT_hence 1: ‹ιx([A!]x & ∀F (x[F] ≡ F = F)) = ιx([A!]x & ∀F (x[F] ≡ F = F))›
    using "rule=I:1" by blast
  AOT_show ‹[A!]aV & [A!]ιx([A!]x & ∀F (x[F] ≡ F = F))›
    apply (rule "=dfI"(2)[OF "df-null-uni-terms:2", OF "null-uni-uniq:4"];
           rule "&I")
     apply (meson "≡dfE" "Conjunction Simplification"(1) "df-null-uni:2"
                  "df-null-uni-terms:2" "null-uni-facts:4" "null-uni-uniq:4"
                  "rule-id-df:2:a[zero]" "→E")
    using "can-ab2"[unvarify y, OF "A-descriptions", THEN "→E", OF 1].
next
  AOT_show ‹∀F (aV[F] ≡ ιx([A!]x & ∀F (x[F] ≡ F = F))[F])›
  proof (rule GEN)
    fix F
    AOT_have ‹aV[F]›
      apply (rule "=dfI"(2)[OF "df-null-uni-terms:2", OF "null-uni-uniq:4"])
      using "≡dfE" "&E"(2) "df-null-uni:2" "df-null-uni-terms:2"
            "null-uni-facts:4" "null-uni-uniq:4" "rule-id-df:2:a[zero]"
            "rule-ui:3" by blast
    moreover AOT_have ‹ιx([A!]x & ∀F (x[F] ≡ F = F))[F]›
      using "RA[2]" "desc-nec-encode:2" "id-eq:1" "≡E"(2) by fastforce
    ultimately AOT_show ‹aV[F] ≡ ιx([A!]x & ∀F (x[F] ≡ F = F))[F]›
      using "deduction-theorem" "≡I" by simp
  qed
qed

AOT_theorem "aclassical:1":                                                   (265.1)
‹∀R∃x∃y(A!x & A!y & x ≠ y & [λz [R]zx] = [λz [R]zy])›
proof(rule GEN)
  fix R
  AOT_obtain a where a_prop:
    ‹A!a & ∀F (a[F] ≡ ∃y(A!y & F = [λz [R]zy] & ¬y[F]))›
    using "A-objects"[axiom_inst] "∃E"[rotated] by fast
  AOT_have a_enc: ‹a[λz [R]za]›
  proof (rule "raa-cor:1")
    AOT_assume 0: ‹¬a[λz [R]za]›
    AOT_hence ‹¬∃y(A!y & [λz [R]za] = [λz [R]zy] & ¬y[λz [R]za])›
      by (rule a_prop[THEN "&E"(2), THEN "∀E"(1)[where τ="«[λz [R]za]»"],
                 THEN "oth-class-taut:4:b"[THEN "≡E"(1)],
```

```
                THEN "≡E"(1), rotated])
         "cqt:2[lambda]"
    AOT_hence ‹∀y ¬(A!y & [λz [R]za] = [λz [R]zy] & ¬y[λz [R]za])›
      using "cqt-further:4" "vdash-properties:10" by blast
    AOT_hence ‹¬(A!a & [λz [R]za] = [λz [R]za] & ¬a[λz [R]za])›
      using "∀E" by blast
    AOT_hence ‹(A!a & [λz [R]za] = [λz [R]za]) → a[λz [R]za]›
      by (metis "&I" "deduction-theorem" "raa-cor:3")
    moreover AOT_have ‹[λz [R]za] = [λz [R]za]›
      by (rule "=I") "cqt:2[lambda]"
    ultimately AOT_have ‹a[λz [R]za]›
      using a_prop[THEN "&E"(1)] "→E" "&I" by blast
    AOT_thus ‹a[λz [R]za] & ¬a[λz [R]za]›
      using 0 "&I" by blast
  qed
  AOT_hence ‹∃y(A!y & [λz [R]za] = [λz [R]zy] & ¬y[λz [R]za])›
    by (rule a_prop[THEN "&E"(2), THEN "∀E"(1), THEN "≡E"(1), rotated])
       "cqt:2"
  then AOT_obtain b where b_prop:
    ‹A!b & [λz [R]za] = [λz [R]zb] & ¬b[λz [R]za]›
    using "∃E"[rotated] by blast
  AOT_have ‹a ≠ b›
    apply (rule "≡dfI"[OF "=-infix"])
    using a_enc b_prop[THEN "&E"(2)]
    using "¬¬I" "rule=E" id_sym "≡E"(4) "oth-class-taut:3:a"
          "raa-cor:3" "reductio-aa:1" by fast
  AOT_hence ‹A!a & A!b & a ≠ b & [λz [R]za] = [λz [R]zb]›
    using b_prop "&E" a_prop "&I" by meson
  AOT_hence ‹∃y (A!a & A!y & a ≠ y & [λz [R]za] = [λz [R]zy])› by (rule "∃I")
  AOT_thus ‹∃x∃y (A!x & A!y & x ≠ y & [λz [R]zx] = [λz [R]zy])› by (rule "∃I")
qed

AOT_theorem "aclassical:2":                                               (265.2)
  ‹∀R∃x∃y(A!x & A!y & x ≠ y & [λz [R]xz] = [λz [R]yz])›
proof(rule GEN)
  fix R
  AOT_obtain a where a_prop:
    ‹A!a & ∀F (a[F] ≡ ∃y(A!y & F = [λz [R]yz] & ¬y[F]))›
    using "A-objects"[axiom_inst] "∃E"[rotated] by fast
  AOT_have a_enc: ‹a[λz [R]az]›
  proof (rule "raa-cor:1")
    AOT_assume 0: ‹¬a[λz [R]az]›
    AOT_hence ‹¬∃y(A!y & [λz [R]az] = [λz [R]yz] & ¬y[λz [R]az])›
      by (rule a_prop[THEN "&E"(2), THEN "∀E"(1)[where τ="«[λz [R]az]»"],
                THEN "oth-class-taut:4:b"[THEN "≡E"(1)],
                THEN "≡E"(1), rotated])
         "cqt:2[lambda]"
    AOT_hence ‹∀y ¬(A!y & [λz [R]az] = [λz [R]yz] & ¬y[λz [R]az])›
      using "cqt-further:4" "vdash-properties:10" by blast
    AOT_hence ‹¬(A!a & [λz [R]az] = [λz [R]az] & ¬a[λz [R]az])›
      using "∀E" by blast
    AOT_hence ‹(A!a & [λz [R]az] = [λz [R]az]) → a[λz [R]az]›
      by (metis "&I" "deduction-theorem" "raa-cor:3")
    moreover AOT_have ‹[λz [R]az] = [λz [R]az]›
      by (rule "=I") "cqt:2[lambda]"
    ultimately AOT_have ‹a[λz [R]az]›
      using a_prop[THEN "&E"(1)] "→E" "&I" by blast
    AOT_thus ‹a[λz [R]az] & ¬a[λz [R]az]›
      using 0 "&I" by blast
  qed
  AOT_hence ‹∃y(A!y & [λz [R]az] = [λz [R]yz] & ¬y[λz [R]az])›
    by (rule a_prop[THEN "&E"(2), THEN "∀E"(1), THEN "≡E"(1), rotated])
       "cqt:2"
```

```
  then AOT_obtain b where b_prop:
    ‹A!b & [λz [R]az] = [λz [R]bz] & ¬b[λz [R]az]›
    using "∃E"[rotated] by blast
  AOT_have ‹a ≠ b›
    apply (rule "≡dfI"[OF "=-infix"])
    using a_enc b_prop[THEN "&E"(2)]
    using "¬¬I" "rule=E" id_sym "≡E"(4) "oth-class-taut:3:a"
          "raa-cor:3" "reductio-aa:1" by fast
  AOT_hence ‹A!a & A!b & a ≠ b & [λz [R]az] = [λz [R]bz]›
    using b_prop "&E" a_prop "&I" by meson
  AOT_hence ‹∃y (A!a & A!y & a ≠ y & [λz [R]az] = [λz [R]yz])› by (rule "∃I")
  AOT_thus ‹∃x∃y (A!x & A!y & x ≠ y & [λz [R]xz] = [λz [R]yz])› by (rule "∃I")
qed

AOT_theorem "aclassical:3":                                              (265.3)
  ‹∀F∃x∃y(A!x & A!y & x ≠ y & [λ [F]x] = [λ [F]y])›
proof(rule GEN)
  fix R
  AOT_obtain a where a_prop:
    ‹A!a & ∀F (a[F] ≡ ∃y(A!y & F = [λz [R]y] & ¬y[F]))›
    using "A-objects"[axiom_inst] "∃E"[rotated] by fast
  AOT_have den: ‹[λz [R]a]↓› by "cqt:2[lambda]"
  AOT_have a_enc: ‹a[λz [R]a]›
  proof (rule "raa-cor:1")
    AOT_assume 0: ‹¬a[λz [R]a]›
    AOT_hence ‹¬∃y(A!y & [λz [R]a] = [λz [R]y] & ¬y[λz [R]a])›
      by (safe intro!: a_prop[THEN "&E"(2), THEN "∀E"(1)[where τ=‹«[λz [R]a]»›],
                  THEN "oth-class-taut:4:b"[THEN "≡E"(1)],
                  THEN "≡E"(1), rotated] "cqt:2")
    AOT_hence ‹∀y ¬(A!y & [λz [R]a] = [λz [R]y] & ¬y[λz [R]a])›
      using "cqt-further:4" "→E" by blast
    AOT_hence ‹¬(A!a & [λz [R]a] = [λz [R]a] & ¬a[λz [R]a])› using "∀E" by blast
    AOT_hence ‹(A!a & [λz [R]a] = [λz [R]a]) → a[λz [R]a]›
      by (metis "&I" "deduction-theorem" "raa-cor:3")
    AOT_hence ‹a[λz [R]a]›
      using a_prop[THEN "&E"(1)] "→E" "&I"
      by (metis "rule=I:1" den)
    AOT_thus ‹a[λz [R]a] & ¬a[λz [R]a]›  by (metis "0" "raa-cor:3")
  qed
  AOT_hence ‹∃y(A!y & [λz [R]a] = [λz [R]y] & ¬y[λz [R]a])›
    by (rule a_prop[THEN "&E"(2), THEN "∀E"(1), OF den, THEN "≡E"(1), rotated])
  then AOT_obtain b where b_prop: ‹A!b & [λz [R]a] = [λz [R]b] & ¬b[λz [R]a]›
    using "∃E"[rotated] by blast
  AOT_have 1: ‹a ≠ b›
    apply (rule "≡dfI"[OF "=-infix"])
    using a_enc b_prop[THEN "&E"(2)]
    using "¬¬I" "rule=E" id_sym "≡E"(4) "oth-class-taut:3:a"
          "raa-cor:3" "reductio-aa:1" by fast
  AOT_have a: ‹[λ [R]a] = ([R]a)›
    apply (rule "lambda-predicates:3[zero]"[axiom_inst, unvarify p])
    by (meson "log-prop-prop:2")
  AOT_have b: ‹[λ [R]b] = ([R]b)›
    apply (rule "lambda-predicates:3[zero]"[axiom_inst, unvarify p])
    by (meson "log-prop-prop:2")
  AOT_have ‹[λ [R]a] = [λ [R]b]›
    apply (rule "rule=E"[rotated, OF a[THEN id_sym]])
    apply (rule "rule=E"[rotated, OF b[THEN id_sym]])
    apply (rule "identity:4"[THEN "≡dfI", OF "&I", rotated])
    using b_prop "&E" apply blast
    apply (safe intro!: "&I")
    by (simp add: "log-prop-prop:2")+
  AOT_hence ‹A!a & A!b & a ≠ b & [λ [R]a] = [λ [R]b]›
    using 1 a_prop[THEN "&E"(1)] b_prop[THEN "&E"(1), THEN "&E"(1)]
```

```
          "&I" by auto
  AOT_hence ‹∃y (A!a & A!y & a ≠ y & [λ [R]a] = [λ [R]y])› by (rule "∃I")
  AOT_thus ‹∃x∃y (A!x & A!y & x ≠ y & [λ [R]x] = [λ [R]y])› by (rule "∃I")
qed

AOT_theorem aclassical2: ‹∃x∃y (A!x & A!y & x ≠ y & ∀F ([F]x ≡ [F]y))›          (266)
proof -
  AOT_have ‹∃x ∃y ([A!]x & [A!]y & x ≠ y &
                [λz [λxy ∀F ([F]x ≡ [F]y)]zx] =
                [λz [λxy ∀F ([F]x ≡ [F]y)]zy])›
    by (rule "aclassical:1"[THEN "∀E"(1)[where τ="«[λxy ∀F ([F]x ≡ [F]y)]»"]])
       "cqt:2"
  then AOT_obtain x where ‹∃y ([A!]x & [A!]y & x ≠ y &
                [λz [λxy ∀F ([F]x ≡ [F]y)]zx] =
                [λz [λxy ∀F ([F]x ≡ [F]y)]zy])›
    using "∃E"[rotated] by blast
  then AOT_obtain y where 0: ‹([A!]x & [A!]y & x ≠ y &
                [λz [λxy ∀F ([F]x ≡ [F]y)]zx] =
                [λz [λxy ∀F ([F]x ≡ [F]y)]zy])›
    using "∃E"[rotated] by blast
  AOT_have ‹[λz [λxy ∀F ([F]x ≡ [F]y)]zx]x›
    by (auto intro!: "β←C"(1) "cqt:2";
        simp add: "&I" "ex:1:a" prod_denotesI "rule-ui:3"
                  "oth-class-taut:3:a" "universal-cor")
  AOT_hence ‹[λz [λxy ∀F ([F]x ≡ [F]y)]zy]x›
    by (rule "rule=E"[rotated, OF 0[THEN "&E"(2)]])
  AOT_hence ‹[λxy ∀F ([F]x ≡ [F]y)]xy›
    by (rule "β→C"(1))
  AOT_hence ‹∀F ([F]x ≡ [F]y)›
    using "β→C"(1) old.prod.case by fast
  AOT_hence ‹[A!]x & [A!]y & x ≠ y & ∀F ([F]x ≡ [F]y)›
    using 0 "&E" "&I" by blast
  AOT_hence ‹∃y ([A!]x & [A!]y & x ≠ y & ∀F ([F]x ≡ [F]y))› by (rule "∃I")
  AOT_thus ‹∃x∃y ([A!]x & [A!]y & x ≠ y & ∀F ([F]x ≡ [F]y))› by (rule "∃I"(2))
qed

AOT_theorem "kirchner-thm:1":                                                 (268.1)
  ‹[λx φ{x}]↓ ≡ □∀x∀y(∀F([F]x ≡ [F]y) → (φ{x} ≡ φ{y}))›
proof(rule "≡I"; rule "→I")
  AOT_assume ‹[λx φ{x}]↓›
  AOT_hence ‹□[λx φ{x}]↓› by (metis "exist-nec" "vdash-properties:10")
  moreover AOT_have ‹□[λx φ{x}]↓ → □∀x∀y(∀F([F]x ≡ [F]y) → (φ{x} ≡ φ{y}))›
  proof (rule "RM:1"; rule "→I"; rule GEN; rule GEN; rule "→I")
    AOT_modally_strict {
      fix x y
      AOT_assume 0: ‹[λx φ{x}]↓›
      moreover AOT_assume ‹∀F([F]x ≡ [F]y)›
      ultimately AOT_have ‹[λx φ{x}]x ≡ [λx φ{x}]y›
        using "∀E" by blast
      AOT_thus ‹(φ{x} ≡ φ{y})›
        using "beta-C-meta"[THEN "→E", OF 0] "≡E"(6) by meson
    }
  qed
  ultimately AOT_show ‹□∀x∀y(∀F([F]x ≡ [F]y) → (φ{x} ≡ φ{y}))›
    using "→E" by blast
next
  AOT_have ‹□∀x∀y(∀F([F]x ≡ [F]y) → (φ{x} ≡ φ{y})) →
            □∀y(∃x(∀F([F]x ≡ [F]y) & φ{x}) ≡ φ{y})›
  proof(rule "RM:1"; rule "→I"; rule GEN)
    AOT_modally_strict {
      AOT_assume ‹∀x∀y(∀F([F]x ≡ [F]y) → (φ{x} ≡ φ{y}))›
      AOT_hence indisc: ‹φ{x} ≡ φ{y}› if ‹∀F([F]x ≡ [F]y)› for x y
        using "∀E"(2) "→E" that by blast
```

```
      AOT_show ‹(∃x(∀F([F]x ≡ [F]y) & φ{x}) ≡ φ{y})› for y
      proof (rule "raa-cor:1")
        AOT_assume ‹¬(∃x(∀F([F]x ≡ [F]y) & φ{x}) ≡ φ{y})›
        AOT_hence ‹(∃x(∀F([F]x ≡ [F]y) & φ{x}) & ¬φ{y}) ∨
                   (¬(∃x(∀F([F]x ≡ [F]y) & φ{x})) & φ{y})›
          using "≡E"(1) "oth-class-taut:4:h" by blast
        moreover {
          AOT_assume 0: ‹∃x(∀F([F]x ≡ [F]y) & φ{x}) & ¬φ{y}›
          AOT_obtain a where ‹∀F([F]a ≡ [F]y) & φ{a}›
            using "∃E"[rotated, OF 0[THEN "&E"(1)]]  by blast
          AOT_hence ‹φ{y}›
            using indisc[THEN "≡E"(1)] "&E" by blast
          AOT_hence ‹p & ¬p› for p
            using 0[THEN "&E"(2)] "&I" "raa-cor:3" by blast
        }
        moreover {
          AOT_assume 0: ‹(¬(∃x(∀F([F]x ≡ [F]y) & φ{x})) & φ{y})›
          AOT_hence ‹∀x ¬(∀F([F]x ≡ [F]y) & φ{x})›
            using "&E"(1) "cqt-further:4" "→E" by blast
          AOT_hence ‹¬(∀F([F]y ≡ [F]y) & φ{y})›
            using "∀E" by blast
          AOT_hence ‹¬∀F([F]y ≡ [F]y) ∨ ¬φ{y}›
            using "≡E"(1) "oth-class-taut:5:c" by blast
          moreover AOT_have ‹∀F([F]y ≡ [F]y)›
            by (simp add: "oth-class-taut:3:a" "universal-cor")
          ultimately AOT_have ‹¬φ{y}› by (metis "¬¬I" "∨E"(2))
          AOT_hence ‹p & ¬p› for p
            using 0[THEN "&E"(2)] "&I" "raa-cor:3" by blast
        }
        ultimately AOT_show ‹p & ¬p› for p
          using "∨E"(3) "raa-cor:1" by blast
      qed
    }
  qed
  moreover AOT_assume ‹□∀x∀y(∀F([F]x ≡ [F]y) → (φ{x} ≡ φ{y}))›
  ultimately AOT_have ‹□∀y(∃x(∀F([F]x ≡ [F]y) & φ{x}) ≡ φ{y})›
    using "→E" by blast
  AOT_thus ‹[λx φ{x}]↓›
    by (rule "safe-ext"[axiom_inst, THEN "→E", OF "&I", rotated]) "cqt:2"
qed

AOT_theorem "kirchner-thm:2":                                              (268.2)
  ‹[λx1...xn φ{x1...xn}]↓ ≡ □∀x1...∀xn∀y1...∀yn
    (∀F([F]x1...xn ≡ [F]y1...yn) → (φ{x1...xn} ≡ φ{y1...yn}))›
proof(rule "≡I"; rule "→I")
  AOT_assume ‹[λx1...xn φ{x1...xn}]↓›
  AOT_hence ‹□[λx1...xn φ{x1...xn}]↓› by (metis "exist-nec" "→E")
  moreover AOT_have ‹□[λx1...xn φ{x1...xn}]↓ → □∀x1...∀xn∀y1...∀yn
    (∀F([F]x1...xn ≡ [F]y1...yn) → (φ{x1...xn} ≡ φ{y1...yn}))›
  proof (rule "RM:1"; rule "→I"; rule GEN; rule GEN; rule "→I")
    AOT_modally_strict {
      fix x1xn y1yn :: ‹'a AOT_var›
      AOT_assume 0: ‹[λx1...xn φ{x1...xn}]↓›
      moreover AOT_assume ‹∀F([F]x1...xn ≡ [F]y1...yn)›
      ultimately AOT_have ‹[λx1...xn φ{x1...xn}]x1...xn ≡
                           [λx1...xn φ{x1...xn}]y1...yn›
        using "∀E" by blast
      AOT_thus ‹(φ{x1...xn} ≡ φ{y1...yn})›
        using "beta-C-meta"[THEN "→E", OF 0] "≡E"(6) by meson
    }
  qed
  ultimately AOT_show ‹□∀x1...∀xn∀y1...∀yn(
    ∀F([F]x1...xn ≡ [F]y1...yn) → (φ{x1...xn} ≡ φ{y1...yn})
```

```
  )›
    using "→E" by blast
next
  AOT_have ‹
    □(∀x1...∀xn∀y1...∀yn
      (∀F([F]x1...xn ≡ [F]y1...yn) → (φ{x1...xn} ≡ φ{y1...yn})))
    → □∀y1...∀yn
        ((∃x1...∃xn(∀F([F]x1...xn ≡ [F]y1...yn) & φ{x1...xn})) ≡
         φ{y1...yn})›
  proof(rule "RM:1"; rule "→I"; rule GEN)
    AOT_modally_strict {
      AOT_assume ‹∀x1...∀xn∀y1...∀yn
        (∀F([F]x1...xn ≡ [F]y1...yn) → (φ{x1...xn} ≡ φ{y1...yn}))›
      AOT_hence indisc: ‹φ{x1...xn} ≡ φ{y1...yn}›
        if ‹∀F([F]x1...xn ≡ [F]y1...yn)› for x1xn y1yn
        using "∀E"(2) "→E" that by blast
      AOT_show ‹(∃x1...∃xn(∀F([F]x1...xn ≡ [F]y1...yn) & φ{x1...xn})) ≡
                φ{y1...yn}› for y1yn
      proof (rule "raa-cor:1")
        AOT_assume ‹¬((∃x1...∃xn(∀F([F]x1...xn ≡ [F]y1...yn) & φ{x1...xn})) ≡
                    φ{y1...yn})›
        AOT_hence ‹((∃x1...∃xn(∀F([F]x1...xn ≡ [F]y1...yn)
                    & φ{x1...xn}))
                    & ¬φ{y1...yn}) ∨
                  (¬(∃x1...∃xn(∀F([F]x1...xn ≡ [F]y1...yn) & φ{x1...xn}))
                   & φ{y1...yn})›
          using "≡E"(1) "oth-class-taut:4:h" by blast
        moreover {
          AOT_assume 0: ‹(∃x1...∃xn(∀F([F]x1...xn ≡ [F]y1...yn) & φ{x1...xn}))
                        & ¬φ{y1...yn}›
          AOT_obtain a1an where ‹∀F([F]a1...an ≡ [F]y1...yn) & φ{a1...an}›
            using "∃E"[rotated, OF 0[THEN "&E"(1)]]  by blast
          AOT_hence ‹φ{y1...yn}›
            using indisc[THEN "≡E"(1)] "&E" by blast
          AOT_hence ‹p & ¬p› for p
            using 0[THEN "&E"(2)] "&I" "raa-cor:3" by blast
        }
        moreover {
          AOT_assume 0: ‹¬(∃x1...∃xn(∀F([F]x1...xn ≡ [F]y1...yn) & φ{x1...xn}))
                        & φ{y1...yn}›
          AOT_hence ‹∀x1...∀xn ¬(∀F([F]x1...xn ≡ [F]y1...yn) & φ{x1...xn})›
            using "&E"(1) "cqt-further:4" "→E" by blast
          AOT_hence ‹¬(∀F([F]y1...yn ≡ [F]y1...yn) & φ{y1...yn})›
            using "∀E" by blast
          AOT_hence ‹¬∀F([F]y1...yn ≡ [F]y1...yn) ∨ ¬φ{y1...yn}›
            using "≡E"(1) "oth-class-taut:5:c" by blast
          moreover AOT_have ‹∀F([F]y1...yn ≡ [F]y1...yn)›
            by (simp add: "oth-class-taut:3:a" "universal-cor")
          ultimately AOT_have ‹¬φ{y1...yn}›
            by (metis "¬¬I" "∨E"(2))
          AOT_hence ‹p & ¬p› for p
            using 0[THEN "&E"(2)] "&I" "raa-cor:3" by blast
        }
        ultimately AOT_show ‹p & ¬p› for p
          using "∨E"(3) "raa-cor:1" by blast
      qed
    }
  qed
  moreover AOT_assume ‹□∀x1...∀xn∀y1...∀yn
    (∀F([F]x1...xn ≡ [F]y1...yn) → (φ{x1...xn} ≡ φ{y1...yn}))›
  ultimately AOT_have ‹□∀y1...∀yn
    ((∃x1...∃xn(∀F([F]x1...xn ≡ [F]y1...yn) & φ{x1...xn})) ≡
     φ{y1...yn})›
```

```
    using "→E" by blast
  AOT_thus ‹[λx1...xn φ{x1...xn}]↓›
    by (rule "safe-ext"[axiom_inst, THEN "→E", OF "&I", rotated]) "cqt:2"
qed

AOT_theorem "kirchner-thm-cor:1":                                        (269.1)
  ‹[λx φ{x}]↓ → ∀x∀y(∀F([F]x ≡ [F]y) → □(φ{x} ≡ φ{y}))›
proof(rule "→I"; rule GEN; rule GEN; rule "→I")
  fix x y
  AOT_assume ‹[λx φ{x}]↓›
  AOT_hence ‹□∀x∀y (∀F ([F]x ≡ [F]y) → (φ{x} ≡ φ{y}))›
    by (rule "kirchner-thm:1"[THEN "≡E"(1)])
  AOT_hence ‹∀x□∀y (∀F ([F]x ≡ [F]y) → (φ{x} ≡ φ{y}))›
    using CBF[THEN "→E"] by blast
  AOT_hence ‹□∀y (∀F ([F]x ≡ [F]y) → (φ{x} ≡ φ{y}))›
    using "∀E" by blast
  AOT_hence ‹∀y □(∀F ([F]x ≡ [F]y) → (φ{x} ≡ φ{y}))›
    using CBF[THEN "→E"] by blast
  AOT_hence ‹□(∀F ([F]x ≡ [F]y) → (φ{x} ≡ φ{y}))›
    using "∀E" by blast
  AOT_hence ‹□∀F ([F]x ≡ [F]y) → □(φ{x} ≡ φ{y})›
    using "qml:1"[axiom_inst] "vdash-properties:6" by blast
  moreover AOT_assume ‹∀F([F]x ≡ [F]y)›
  ultimately AOT_show ‹□(φ{x} ≡ φ{y})› using "→E" "ind-nec" by blast
qed

AOT_theorem "kirchner-thm-cor:2":                                        (269.2)
  ‹[λx1...xn φ{x1...xn}]↓ → ∀x1...∀xn∀y1...∀yn
    (∀F([F]x1...xn ≡ [F]y1...yn) → □(φ{x1...xn} ≡ φ{y1...yn}))›
proof(rule "→I"; rule GEN; rule GEN; rule "→I")
  fix x1xn y1yn
  AOT_assume ‹[λx1...xn φ{x1...xn}]↓›
  AOT_hence 0: ‹□∀x1...∀xn∀y1...∀yn
    (∀F ([F]x1...xn ≡ [F]y1...yn) → (φ{x1...xn} ≡ φ{y1...yn}))›
    by (rule "kirchner-thm:2"[THEN "≡E"(1)])
  AOT_have ‹∀x1...∀xn∀y1...∀yn
    □(∀F ([F]x1...xn ≡ [F]y1...yn) → (φ{x1...xn} ≡ φ{y1...yn}))›
  proof(rule GEN; rule GEN)
    fix x1xn y1yn
    AOT_show ‹□(∀F ([F]x1...xn ≡ [F]y1...yn) → (φ{x1...xn} ≡ φ{y1...yn}))›
      apply (rule "RM:1"[THEN "→E", rotated, OF 0]; rule "→I")
      using "∀E" by blast
  qed
  AOT_hence ‹∀y1...∀yn □(∀F ([F]x1...xn ≡ [F]y1...yn) →
                          (φ{x1...xn} ≡ φ{y1...yn}))›
    using "∀E" by blast
  AOT_hence ‹□(∀F ([F]x1...xn ≡ [F]y1...yn) → (φ{x1...xn} ≡ φ{y1...yn}))›
    using "∀E" by blast
  AOT_hence ‹□(∀F ([F]x1...xn ≡ [F]y1...yn) → (φ{x1...xn} ≡ φ{y1...yn}))›
    using "∀E" by blast
  AOT_hence 0: ‹□∀F ([F]x1...xn ≡ [F]y1...yn) → □(φ{x1...xn} ≡ φ{y1...yn})›
    using "qml:1"[axiom_inst] "vdash-properties:6" by blast
  moreover AOT_assume ‹∀F([F]x1...xn ≡ [F]y1...yn)›
  moreover AOT_have ‹[λx1...xn □∀F ([F]x1...xn ≡ [F]y1...yn)]↓› by "cqt:2"
  ultimately AOT_have ‹[λx1...xn □∀F ([F]x1...xn ≡ [F]y1...yn)]x1...xn ≡
                      [λx1...xn □∀F ([F]x1...xn ≡ [F]y1...yn)]y1...yn›
    using "∀E" by blast
  moreover AOT_have ‹[λx1...xn □∀F ([F]x1...xn ≡ [F]y1...yn)]y1...yn›
    apply (rule "β←C"(1))
      apply "cqt:2[lambda]"
     apply (fact "cqt:2[const_var]"[axiom_inst])
    by (simp add: RN GEN "oth-class-taut:3:a")
  ultimately AOT_have ‹[λx1...xn □∀F ([F]x1...xn ≡ [F]y1...yn)]x1...xn›
```

```
    using "≡E"(2) by blast
  AOT_hence ‹□∀F ([F]x_1...x_n ≡ [F]y_1...y_n)›
    using "β→C"(1) by blast
  AOT_thus ‹□(φ{x_1...x_n} ≡ φ{y_1...y_n})› using "→E" 0 by blast
qed

subsection‹Propositional Properties›
text‹\label{PLM: 9.12}›

AOT_define propositional :: ‹Π ⇒ φ› (‹Propositional'(_')›)
  "prop-prop1": ‹Propositional([F]) ≡df ∃p(F = [λy p])›                    (270)

AOT_theorem "prop-prop2:1": ‹∀p [λy p]↓›                                   (271.1)
  by (rule GEN) "cqt:2[lambda]"

AOT_theorem "prop-prop2:2": ‹[λν φ]↓›                                      (271.2)
  by "cqt:2[lambda]"

AOT_theorem "prop-prop2:3": ‹F = [λy p] → □∀x([F]x ≡ p)›                   (271.3)
proof (rule "→I")
  AOT_assume 0: ‹F = [λy p]›
  AOT_show ‹□∀x([F]x ≡ p)›
    by (rule "rule=E"[rotated, OF 0[symmetric]];
        rule RN; rule GEN; rule "beta-C-meta"[THEN "→E"])
       "cqt:2[lambda]"
qed

AOT_theorem "prop-prop2:4": ‹Propositional([F]) → □Propositional([F])›    (271.4)
proof(rule "→I")
  AOT_assume ‹Propositional([F])›
  AOT_hence ‹∃p(F = [λy p])›
    using "≡dfE"[OF "prop-prop1"] by blast
  then AOT_obtain p where ‹F = [λy p]›
    using "∃E"[rotated] by blast
  AOT_hence ‹□(F = [λy p])›
    using "id-nec:2" "modus-tollens:1" "raa-cor:3" by blast
  AOT_hence ‹∃p □(F = [λy p])›
    using "∃I" by fast
  AOT_hence 0: ‹□∃p (F = [λy p])›
    by (metis Buridan "vdash-properties:10")
  AOT_thus ‹□Propositional([F])›
    using "prop-prop1"[THEN "≡Df"]
    by (AOT_subst ‹Propositional([F])› ‹∃p (F = [λy p])›) auto
qed

AOT_define indicriminate :: ‹Π ⇒ φ› ("Indiscriminate'(_')")
  "prop-indis": ‹Indiscriminate([F]) ≡df F↓ & □(∃x [F]x → ∀x [F]x)›        (272)

AOT_theorem "prop-in-thm": ‹Propositional([Π]) → Indiscriminate([Π])›     (273)
proof(rule "→I")
  AOT_assume ‹Propositional([Π])›
  AOT_hence ‹∃p Π = [λy p]› using "≡dfE"[OF "prop-prop1"] by blast
  then AOT_obtain p where Π_def: ‹Π = [λy p]› using "∃E"[rotated] by blast
  AOT_show ‹Indiscriminate([Π])›
  proof (rule "≡dfI"[OF "prop-indis"]; rule "&I")
    AOT_show ‹Π↓›
      using Π_def by (meson "t=t-proper:1" "vdash-properties:6")
  next
    AOT_show ‹□(∃x [Π]x → ∀x [Π]x)›
    proof (rule "rule=E"[rotated, OF Π_def[symmetric]];
           rule RN; rule "→I"; rule GEN)
      AOT_modally_strict {
        AOT_assume ‹∃x [λy p]x›
```

```
      then AOT_obtain a where ‹[λy p]a› using "∃E"[rotated] by blast
      AOT_hence 0: ‹p› by (metis "β→C"(1))
      AOT_show ‹[λy p]x› for x
        apply (rule "β←C"(1))
          apply "cqt:2[lambda]"
         apply (fact "cqt:2[const_var]"[axiom_inst])
        by (fact 0)
    }
  qed
 qed
qed

AOT_theorem "prop-in-f:1": ‹Necessary([F]) → Indiscriminate([F])›          (274.1)
proof (rule "→I")
  AOT_assume ‹Necessary([F])›
  AOT_hence 0: ‹□∀x1...∀xn [F]x1...xn›
    using "≡dfE"[OF "contingent-properties:1"] by blast
  AOT_show ‹Indiscriminate([F])›
    by (rule "≡dfI"[OF "prop-indis"])
       (metis "0" "KBasic:1" "&I" "ex:1:a" "rule-ui:2[const_var]" "→E")
qed

AOT_theorem "prop-in-f:2": ‹Impossible([F]) → Indiscriminate([F])›          (274.2)
proof (rule "→I")
  AOT_modally_strict {
    AOT_have ‹∀x ¬[F]x → (∃x [F]x → ∀x [F]x)›
      by (metis "∃E" "cqt-orig:3" "Hypothetical Syllogism" "→I" "raa-cor:3")
  }
  AOT_hence 0: ‹□∀x ¬[F]x → □(∃x [F]x → ∀x [F]x)›
    by (rule "RM:1")
  AOT_assume ‹Impossible([F])›
  AOT_hence ‹□∀x ¬[F]x›
    using "≡dfE"[OF "contingent-properties:2"] "&E" by blast
  AOT_hence 1: ‹□(∃x [F]x → ∀x [F]x)›
    using 0 "→E" by blast
  AOT_show ‹Indiscriminate([F])›
    by (rule "≡dfI"[OF "prop-indis"]; rule "&I")
       (simp add: "ex:1:a" "rule-ui:2[const_var]" 1)+
qed

AOT_theorem "prop-in-f:3:a": ‹¬Indiscriminate([E!])›          (274.3.a)
proof(rule "raa-cor:2")
  AOT_assume ‹Indiscriminate([E!])›
  AOT_hence 0: ‹□(∃x [E!]x → ∀x [E!]x)›
    using "≡dfE"[OF "prop-indis"] "&E" by blast
  AOT_hence ‹◊∃x [E!]x → ◊∀x [E!]x›
    using "KBasic:13" "vdash-properties:10" by blast
  moreover AOT_have ‹◊∃x [E!]x›
    by (simp add: "thm-cont-e:3")
  ultimately AOT_have ‹◊∀x [E!]x›
    by (metis "vdash-properties:6")
  AOT_thus ‹p & ¬p› for p
    by (metis "≡dfE" "conventions:5" "o-objects-exist:5" "reductio-aa:1")
qed

AOT_theorem "prop-in-f:3:b": ‹¬Indiscriminate([E!]¯)›          (274.3.b)
proof (rule "rule=E"[rotated, OF "rel-neg-T:2"[symmetric]];
       rule "raa-cor:2")
  AOT_assume ‹Indiscriminate([λx ¬[E!]x])›
  AOT_hence 0: ‹□(∃x [λx ¬[E!]x]x → ∀x [λx ¬[E!]x]x)›
    using "≡dfE"[OF "prop-indis"] "&E" by blast
  AOT_hence ‹□∃x [λx ¬[E!]x]x → □∀x [λx ¬[E!]x]x›
    using "→E" "qml:1" "vdash-properties:1[2]" by blast
```

```
  moreover AOT_have ‹□∃x [λx ¬[E!]x]x›
    apply (AOT_subst ‹[λx ¬E!x]x› ‹¬E!x› for: x)
    apply (rule "beta-C-meta"[THEN "→E"])
     apply "cqt:2"
    by (metis (full_types) "B◊" RN "T◊" "cqt-further:2"
                           "o-objects-exist:5" "→E")
  ultimately AOT_have 1: ‹□∀x [λx ¬[E!]x]x›
    by (metis "vdash-properties:6")
  AOT_hence ‹□∀x ¬[E!]x›
    by (AOT_subst (reverse) ‹¬[E!]x›  ‹[λx ¬[E!]x]x› for: x)
       (auto intro!: "cqt:2" "beta-C-meta"[THEN "→E"])
  AOT_hence ‹∀x □¬[E!]x› by (metis "CBF" "vdash-properties:10")
  moreover AOT_obtain a where abs_a: ‹O!a›
    using "∃E" "o-objects-exist:1" "qml:2"[axiom_inst] "→E" by blast
  ultimately AOT_have ‹□¬[E!]a› using "∀E" by blast
  AOT_hence 2: ‹¬◊[E!]a› by (metis "≡dfE" "conventions:5" "reductio-aa:1")
  AOT_have ‹A!a›
    apply (rule "=dfI"(2)[OF AOT_abstract])
     apply "cqt:2[lambda]"
    apply (rule "β←C"(1))
      apply "cqt:2[lambda]"
    using "cqt:2[const_var]"[axiom_inst] apply blast
    by (fact 2)
  AOT_thus ‹p & ¬p› for p using abs_a
    by (metis "≡E"(1) "oa-contingent:2" "reductio-aa:1")
qed

AOT_theorem "prop-in-f:3:c": ‹¬Indiscriminate(O!)›                 (274.3.c)
proof(rule "raa-cor:2")
  AOT_assume ‹Indiscriminate(O!)›
  AOT_hence 0: ‹□(∃x O!x → ∀x O!x)›
    using "≡dfE"[OF "prop-indis"] "&E" by blast
  AOT_hence ‹□∃x O!x → □∀x O!x›
    using "qml:1"[axiom_inst] "vdash-properties:6" by blast
  moreover AOT_have ‹□∃x O!x›
    using "o-objects-exist:1" by blast
  ultimately AOT_have ‹□∀x O!x›
    by (metis "vdash-properties:6")
  AOT_thus ‹p & ¬p› for p
    by (metis "o-objects-exist:3" "qml:2"[axiom_inst] "raa-cor:3" "→E")
qed

AOT_theorem "prop-in-f:3:d": ‹¬Indiscriminate(A!)›                 (274.3.d)
proof(rule "raa-cor:2")
  AOT_assume ‹Indiscriminate(A!)›
  AOT_hence 0: ‹□(∃x A!x → ∀x A!x)›
    using "≡dfE"[OF "prop-indis"] "&E" by blast
  AOT_hence ‹□∃x A!x → □∀x A!x›
    using "qml:1"[axiom_inst] "vdash-properties:6" by blast
  moreover AOT_have ‹□∃x A!x›
    using "o-objects-exist:2" by blast
  ultimately AOT_have ‹□∀x A!x›
    by (metis "vdash-properties:6")
  AOT_thus ‹p & ¬p› for p
    by (metis "o-objects-exist:4" "qml:2"[axiom_inst] "raa-cor:3" "→E")
qed

AOT_theorem "prop-in-f:4:a": ‹¬Propositional(E!)›                  (274.4.a)
  using "modus-tollens:1" "prop-in-f:3:a" "prop-in-thm" by blast

AOT_theorem "prop-in-f:4:b": ‹¬Propositional(E!¯)›                 (274.4.b)
  using "modus-tollens:1" "prop-in-f:3:b" "prop-in-thm" by blast
```

```
AOT_theorem "prop-in-f:4:c": ‹¬Propositional(O!)›                    (274.4.c)
  using "modus-tollens:1" "prop-in-f:3:c" "prop-in-thm" by blast

AOT_theorem "prop-in-f:4:d": ‹¬Propositional(A!)›                    (274.4.d)
  using "modus-tollens:1" "prop-in-f:3:d" "prop-in-thm" by blast

AOT_theorem "prop-prop-nec:1": ‹◊∃p (F = [λy p]) → ∃p(F = [λy p])›   (275.1)
proof(rule "→I")
  AOT_assume ‹◊∃p (F = [λy p])›
  AOT_hence ‹∃p ◊(F = [λy p])›
    by (metis "BF◊" "→E")
  then AOT_obtain p where ‹◊(F = [λy p])›
    using "∃E"[rotated] by blast
  AOT_hence ‹F = [λy p]›
    by (metis "derived-S5-rules:2" emptyE "id-nec:2" "→E")
  AOT_thus ‹∃p(F = [λy p])› by (rule "∃I")
qed

AOT_theorem "prop-prop-nec:2": ‹∀p (F ≠ [λy p]) → □∀p(F ≠ [λy p])›   (275.2)
proof(rule "→I")
  AOT_assume ‹∀p (F ≠ [λy p])›
  AOT_hence ‹(F ≠ [λy p])› for p
    using "∀E" by blast
  AOT_hence ‹□(F ≠ [λy p])› for p
    by (rule "id-nec2:2"[unvarify β, THEN "→E", rotated]) "cqt:2"
  AOT_hence ‹∀p □(F ≠ [λy p])› by (rule GEN)
  AOT_thus ‹□∀p (F ≠ [λy p])› using BF[THEN "→E"] by fast
qed

AOT_theorem "prop-prop-nec:3": ‹∃p (F = [λy p]) → □∃p(F = [λy p])›   (275.3)
proof(rule "→I")
  AOT_assume ‹∃p (F = [λy p])›
  then AOT_obtain p where ‹(F = [λy p])› using "∃E"[rotated] by blast
  AOT_hence ‹□(F = [λy p])› by (metis "id-nec:2" "→E")
  AOT_hence ‹∃p□(F = [λy p])› by (rule "∃I")
  AOT_thus ‹□∃p(F = [λy p])› by (metis Buridan "→E")
qed

AOT_theorem "prop-prop-nec:4": ‹◊∀p (F ≠ [λy p]) → ∀p(F ≠ [λy p])›   (275.4)
proof(rule "→I")
  AOT_assume ‹◊∀p (F ≠ [λy p])›
  AOT_hence ‹∀p ◊(F ≠ [λy p])› by (metis "Buridan◊" "→E")
  AOT_hence ‹◊(F ≠ [λy p])› for p
    using "∀E" by blast
  AOT_hence ‹F ≠ [λy p]› for p
    by (rule "id-nec2:3"[unvarify β, THEN "→E", rotated]) "cqt:2"
  AOT_thus ‹∀p (F ≠ [λy p])› by (rule GEN)
qed

AOT_theorem "enc-prop-nec:1":                                          (276.1)
  ‹◊∀F (x[F] → ∃p(F = [λy p])) → ∀F(x[F] → ∃p (F = [λy p]))›
proof(rule "→I"; rule GEN; rule "→I")
  fix F
  AOT_assume ‹◊∀F (x[F] → ∃p(F = [λy p]))›
  AOT_hence ‹∀F ◊(x[F] → ∃p(F = [λy p]))›
    using "Buridan◊" "vdash-properties:10" by blast
  AOT_hence 0: ‹◊(x[F] → ∃p(F = [λy p]))› using "∀E" by blast
  AOT_assume ‹x[F]›
  AOT_hence ‹□x[F]› by (metis "en-eq:2[1]" "≡E"(1))
  AOT_hence ‹◊∃p(F = [λy p])›
    using 0 by (metis "KBasic2:4" "≡E"(1) "vdash-properties:10")
  AOT_thus ‹∃p(F = [λy p])›
    using "prop-prop-nec:1"[THEN "→E"] by blast
```

```
qed

AOT_theorem "enc-prop-nec:2":                                              (276.2)
  ‹∀F (x[F] → ∃p(F = [λy p])) → □∀F(x[F] → ∃p (F = [λy p]))›
  using "derived-S5-rules:1"[where Γ="{}", simplified, OF "enc-prop-nec:1"]
  by blast

(*<*)
end
(*>*)
```

A.8. Basic Logical Objects

```
(*<*)
theory AOT_BasicLogicalObjects
  imports AOT_PLM
begin
(*>*)

section‹Basic Logical Objects›
(* Note: so far only the parts required for possible world theory are implemented *)

AOT_define TruthValueOf :: ‹τ ⇒ φ ⇒ φ› (‹TruthValueOf'(_,_')›)
  "tv-p": ‹TruthValueOf(x,p) ≡df A!x & ∀F (x[F] ≡ ∃q((q ≡ p) & F = [λy q]))›        (281)

AOT_theorem "p-has-!tv:1": ‹∃x TruthValueOf(x,p)›                                   (283.1)
  using  "tv-p"[THEN "≡Df"]
  by (AOT_subst ‹TruthValueOf(x,p)›
                ‹A!x & ∀F (x[F] ≡ ∃q((q ≡ p) & F = [λy q]))› for: x)
     (simp add: "A-objects"[axiom_inst])


AOT_theorem "p-has-!tv:2": ‹∃!x TruthValueOf(x,p)›                                  (283.2)
  using  "tv-p"[THEN "≡Df"]
  by (AOT_subst ‹TruthValueOf(x,p)›
                ‹A!x & ∀F (x[F] ≡ ∃q((q ≡ p) & F = [λy q]))› for: x)
     (simp add: "A-objects!")


AOT_theorem "uni-tv": ‹ιx TruthValueOf(x,p)↓›                                       (284)
  using "A-Exists:2" "RA[2]" "≡E"(2) "p-has-!tv:2" by blast

AOT_define TheTruthValueOf :: ‹φ ⇒ κs› (‹∘_› [100] 100)
  "the-tv-p": ‹∘p =df ιx TruthValueOf(x,p)›                                         (285)

AOT_define PropEnc :: ‹τ ⇒ φ ⇒ φ› (infixl ‹Σ› 40)
  "prop-enc": ‹xΣp ≡df x↓ & x[λy p]›                                                (286)

AOT_theorem "tv-id:1": ‹∘p = ιx (A!x & ∀F (x[F] ≡ ∃q((q ≡ p) & F = [λy q])))›       (287.1)
proof -
  AOT_have ‹□∀x(TruthValueOf(x,p)  ≡ A!x & ∀F (x[F] ≡ ∃q((q ≡ p) & F = [λy q])))›
    by (rule RN; rule GEN; rule "tv-p"[THEN "≡Df"])
  AOT_hence ‹ιx TruthValueOf(x,p) = ιx (A!x & ∀F (x[F] ≡ ∃q((q ≡ p) & F = [λy q])))›
    using "equiv-desc-eq:3"[THEN "→E", OF "&I", OF "uni-tv"] by simp
  thus ?thesis
    using "=dfI"(1)[OF "the-tv-p", OF "uni-tv"] by fast
qed

AOT_theorem "tv-id:2": ‹∘pΣp›                                                       (287.2)
proof -
  AOT_modally_strict {
    AOT_have ‹(p ≡ p) & [λy p] = [λy p]›
      by (auto simp: "prop-prop2:2" "rule=I:1" intro!: "≡I" "→I" "&I")
    AOT_hence ‹∃q ((q ≡ p) & [λy p] = [λy q])›
      using "∃I" by fast
  }
  AOT_hence ‹𝒜∃q ((q ≡ p) & [λy p] = [λy q])›
    using "RA[2]" by blast
  AOT_hence ‹ιx(A!x & ∀F (x[F] ≡ ∃q ((q ≡ p) & F = [λy q])))[λy p]›
    by (safe intro!: "desc-nec-encode:1"[unvarify F, THEN "≡E"(2)] "cqt:2")
  AOT_hence ‹ιx(A!x & ∀F (x[F] ≡ ∃q ((q ≡ p) & F = [λy q])))Σp›
    by (safe intro!: "prop-enc"[THEN "≡dfI"] "&I" "A-descriptions")
  AOT_thus ‹∘pΣp›
    by (rule "rule=E"[rotated, OF "tv-id:1"[symmetric]])
```

```
qed

(* TODO more theorems *)

AOT_theorem "TV-lem1:1":                                                    (292.1)
  ‹p ≡ ∀F(∃q (q & F = [λy q]) ≡ ∃q((q ≡ p) & F = [λy q]))›
proof(safe intro!: "≡I" "→I" GEN)
  fix F
  AOT_assume ‹∃q (q & F = [λy q])›
  then AOT_obtain q where ‹q & F = [λy q]› using "∃E"[rotated] by blast
  moreover AOT_assume p
  ultimately AOT_have ‹(q ≡ p) & F = [λy q]›
    by (metis "&I" "&E"(1) "&E"(2) "deduction-theorem" "≡I")
  AOT_thus ‹∃q ((q ≡ p) & F = [λy q])› by (rule "∃I")
next
  fix F
  AOT_assume ‹∃q ((q ≡ p) & F = [λy q])›
  then AOT_obtain q where ‹(q ≡ p) & F = [λy q]› using "∃E"[rotated] by blast
  moreover AOT_assume p
  ultimately AOT_have ‹q & F = [λy q]›
    by (metis "&I" "&E"(1) "&E"(2) "≡E"(2))
  AOT_thus ‹∃q (q & F = [λy q])› by (rule "∃I")
next
  AOT_assume ‹∀F (∃q (q & F = [λy q]) ≡ ∃q ((q ≡ p) & F = [λy q]))›
  AOT_hence ‹∃q (q & [λy p] = [λy q]) ≡ ∃q ((q ≡ p) & [λy p] = [λy q])›
    using "∀E"(1)[rotated, OF "prop-prop2:2"] by blast
  moreover AOT_have ‹∃q ((q ≡ p) & [λy p] = [λy q])›
    by (rule "∃I"(2)[where β=p])
       (simp add: "rule=I:1" "&I" "oth-class-taut:3:a" "prop-prop2:2")
  ultimately AOT_have ‹∃q (q & [λy p] = [λy q])› using "≡E"(2) by blast
  then AOT_obtain q where ‹q & [λy p] = [λy q]› using "∃E"[rotated] by blast
  AOT_thus ‹p›
    using "rule=E" "&E"(1) "&E"(2) id_sym "≡E"(2) "p-identity-thm2:3" by fast
qed

AOT_theorem "TV-lem1:2":                                                    (292.2)
  ‹¬p ≡ ∀F(∃q (¬q & F = [λy q]) ≡ ∃q((q ≡ p) & F = [λy q]))›
proof(safe intro!: "≡I" "→I" GEN)
  fix F
  AOT_assume ‹∃q (¬q & F = [λy q])›
  then AOT_obtain q where ‹¬q & F = [λy q]› using "∃E"[rotated] by blast
  moreover AOT_assume ‹¬p›
  ultimately AOT_have ‹(q ≡ p) & F = [λy q]›
    by (metis "&I" "&E"(1) "&E"(2) "deduction-theorem" "≡I" "raa-cor:3")
  AOT_thus ‹∃q ((q ≡ p) & F = [λy q])› by (rule "∃I")
next
  fix F
  AOT_assume ‹∃q ((q ≡ p) & F = [λy q])›
  then AOT_obtain q where ‹(q ≡ p) & F = [λy q]› using "∃E"[rotated] by blast
  moreover AOT_assume ‹¬p›
  ultimately AOT_have ‹¬q & F = [λy q]›
    by (metis "&I" "&E"(1) "&E"(2) "≡E"(1) "raa-cor:3")
  AOT_thus ‹∃q (¬q & F = [λy q])› by (rule "∃I")
next
  AOT_assume ‹∀F (∃q (¬q & F = [λy q]) ≡ ∃q ((q ≡ p) & F = [λy q]))›
  AOT_hence ‹∃q (¬q & [λy p] = [λy q]) ≡ ∃q ((q ≡ p) & [λy p] = [λy q])›
    using "∀E"(1)[rotated, OF "prop-prop2:2"] by blast
  moreover AOT_have ‹∃q ((q ≡ p) & [λy p] = [λy q])›
    by (rule "∃I"(2)[where β=p])
       (simp add: "rule=I:1" "&I" "oth-class-taut:3:a" "prop-prop2:2")
  ultimately AOT_have ‹∃q (¬q & [λy p] = [λy q])› using "≡E"(2) by blast
  then AOT_obtain q where ‹¬q & [λy p] = [λy q]› using "∃E"[rotated] by blast
  AOT_thus ‹¬p›
```

```
    using "rule=E" "&E"(1) "&E"(2) id_sym "≡E"(2) "p-identity-thm2:3" by fast
qed


AOT_define TruthValue :: ‹τ ⇒ φ› (‹TruthValue'(_')›)
  "T-value": ‹TruthValue(x) ≡df ∃p (TruthValueOf(x,p))›                    (293)

(* TODO more theorems *)

AOT_act_theorem "T-lem:1": ‹TruthValueOf(∘p, p)›                          (290.1)
proof -
  AOT_have ϑ: ‹∘p = ιx TruthValueOf(x, p)›
    using "rule-id-df:1" "the-tv-p" "uni-tv" by blast
  moreover AOT_have ‹∘p↓›
    using "t=t-proper:1" calculation "vdash-properties:10" by blast
  ultimately show ?thesis by (metis "rule=E" id_sym "vdash-properties:10" "y-in:3")
qed

AOT_act_theorem "T-lem:2": ‹∀F (∘p[F] ≡ ∃q((q ≡ p) & F = [λy q]))›       (290.2)
  using "T-lem:1"[THEN "tv-p"[THEN "≡dfE"], THEN "&E"(2)].

AOT_act_theorem "T-lem:3": ‹∘pΣr ≡ (r ≡ p)›                               (290.3)
proof -
  AOT_have ϑ: ‹∘p[λy r] ≡ ∃q ((q ≡ p) & [λy r] = [λy q])›
    using "T-lem:2"[THEN "∀E"(1), OF "prop-prop2:2"].
  show ?thesis
  proof(rule "≡I"; rule "→I")
    AOT_assume ‹∘pΣr›
    AOT_hence ‹∘p[λy r]› by (metis "≡dfE" "&E"(2) "prop-enc")
    AOT_hence ‹∃q ((q ≡ p) & [λy r] = [λy q])› using ϑ "≡E"(1) by blast
    then AOT_obtain q where ‹(q ≡ p) & [λy r] = [λy q]› using "∃E"[rotated] by blast
    moreover AOT_have ‹r = q› using calculation
      using "&E"(2) "≡E"(2) "p-identity-thm2:3" by blast
    ultimately AOT_show ‹r ≡ p›
      by (metis "rule=E" "&E"(1) "≡E"(6) "oth-class-taut:3:a")
  next
    AOT_assume ‹r ≡ p›
    moreover AOT_have ‹[λy r] = [λy r]›
      by (simp add: "rule=I:1" "prop-prop2:2")
    ultimately AOT_have ‹(r ≡ p) & [λy r] = [λy r]› using "&I" by blast
    AOT_hence ‹∃q ((q ≡ p) & [λy r] = [λy q])› by (rule "∃I"(2)[where β=r])
    AOT_hence ‹∘p[λy r]› using ϑ "≡E"(2) by blast
    AOT_thus ‹∘pΣr›
      by (metis "≡dfI" "&I" "prop-enc" "russell-axiom[enc,1].ψ_denotes_asm")
  qed
qed

AOT_act_theorem "T-lem:4": ‹TruthValueOf(x, p) ≡ x = ∘p›                  (290.4)
proof -
  AOT_have ‹∀x (x = ιx TruthValueOf(x, p) ≡ ∀z (TruthValueOf(z, p) ≡ z = x))›
    by (simp add: "fund-cont-desc" GEN)
  moreover AOT_have ‹∘p↓›
    using "≡dfE" "tv-id:2" "&E"(1) "prop-enc" by blast
  ultimately AOT_have
    ‹(∘p = ιx TruthValueOf(x, p)) ≡ ∀z (TruthValueOf(z, p) ≡ z = ∘p)›
    using "∀E"(1) by blast
  AOT_hence ‹∀z (TruthValueOf(z, p) ≡ z = ∘p)›
    using "≡E"(1) "rule-id-df:1" "the-tv-p" "uni-tv" by blast
  AOT_thus ‹TruthValueOf(x, p) ≡ x = ∘p› using "∀E"(2) by blast
qed


(* TODO more theorems *)
```

```
AOT_theorem "TV-lem2:1":                                                    (295.1)
  ‹(A!x & ∀F (x[F] ≡ ∃q (q & F = [λy q]))) → TruthValue(x)›
proof(safe intro!: "→I" "T-value"[THEN "≡dfI"] "tv-p"[THEN "≡dfI"]
                   "∃I"(1)[rotated, OF "log-prop-prop:2"])
  AOT_assume ‹[A!]x & ∀F (x[F] ≡ ∃q (q & F = [λy q]))›
  AOT_thus ‹[A!]x & ∀F (x[F] ≡ ∃q ((q ≡ (∀p (p → p))) & F = [λy q]))›
    apply (AOT_subst ‹∃q ((q ≡ (∀p (p → p))) & F = [λy q])›
                     ‹∃q (q & F = [λy q])› for: F :: ‹<κ>›)
     apply (AOT_subst ‹q ≡ ∀p (p →p)› ‹q› for: q)
    apply (metis (no_types, lifting) "→I" "≡I" "≡E"(2) GEN)
    by (auto simp: "cqt-further:7")
qed

AOT_theorem "TV-lem2:2":                                                    (295.2)
  ‹(A!x & ∀F (x[F] ≡ ∃q (¬q & F = [λy q]))) → TruthValue(x)›
proof(safe intro!: "→I" "T-value"[THEN "≡dfI"] "tv-p"[THEN "≡dfI"]
                   "∃I"(1)[rotated, OF "log-prop-prop:2"])
  AOT_assume ‹[A!]x & ∀F (x[F] ≡ ∃q (¬q & F = [λy q]))›
  AOT_thus ‹[A!]x & ∀F (x[F] ≡ ∃q ((q ≡ (∃p (p & ¬p))) & F = [λy q]))›
    apply (AOT_subst ‹∃q ((q ≡ (∃p (p & ¬p))) & F = [λy q])›
                     ‹∃q (¬q & F = [λy q])› for: F :: ‹<κ>›)
     apply (AOT_subst ‹q ≡ ∃p (p & ¬p)› ‹¬q› for: q)
      apply (metis (no_types, lifting)
        "→I" "∃E" "≡E"(1) "≡I" "raa-cor:1" "raa-cor:3")
    by (auto simp add: "cqt-further:7")
qed

AOT_define TheTrue :: κs (‹⊤›)
  "the-true:1": ‹⊤ =df ιx (A!x & ∀F (x[F] ≡ ∃p(p & F = [λy p])))›          (296.1)
AOT_define TheFalse :: κs (‹⊥›)
  "the-true:2": ‹⊥ =df ιx (A!x & ∀F (x[F] ≡ ∃p(¬p & F = [λy p])))›         (296.2)


AOT_theorem "the-true:3": ‹⊤ ≠ ⊥›                                           (296.3)
proof(safe intro!: "ab-obey:2"[unvarify x y, THEN "→E", rotated 2, OF "∨I"(1)]
                   "∃I"(1)[where τ=‹«[λx ∀q(q → q)]»›] "&I" "prop-prop2:2")
  AOT_have false_def: ‹⊥ = ιx (A!x & ∀F (x[F] ≡ ∃p(¬p & F = [λy p])))›
    by (simp add: "A-descriptions" "rule-id-df:1[zero]" "the-true:2")
  moreover AOT_show false_den: ‹⊥↓›
    by (meson "→E" "t=t-proper:1" "A-descriptions"
              "rule-id-df:1[zero]" "the-true:2")
  ultimately AOT_have false_prop: ‹𝒜(A!⊥ & ∀F (⊥[F] ≡ ∃p(¬p & F = [λy p])))›
    using "nec-hintikka-scheme"[unvarify x, THEN "≡E"(1), THEN "&E"(1)] by blast
  AOT_hence ‹𝒜∀F (⊥[F] ≡ ∃p(¬p & F = [λy p]))›
    using "Act-Basic:2" "&E"(2) "≡E"(1) by blast
  AOT_hence ‹∀F 𝒜(⊥[F] ≡ ∃p(¬p & F = [λy p]))›
    using "≡E"(1) "logic-actual-nec:3"[axiom_inst] by blast
  AOT_hence false_enc_cond:
    ‹𝒜(⊥[λx ∀q(q → q)] ≡ ∃p(¬p & [λx ∀q(q → q)] = [λy p]))›
    using "∀E"(1)[rotated, OF "prop-prop2:2"] by blast

  AOT_have true_def: ‹⊤ = ιx (A!x & ∀F (x[F] ≡ ∃p(p & F = [λy p])))›
    by (simp add: "A-descriptions" "rule-id-df:1[zero]" "the-true:1")
  moreover AOT_show true_den: ‹⊤↓›
    by (meson "t=t-proper:1" "A-descriptions" "rule-id-df:1[zero]" "the-true:1" "→E")
  ultimately AOT_have true_prop: ‹𝒜(A!⊤ & ∀F (⊤[F] ≡ ∃p(p & F = [λy p])))›
    using "nec-hintikka-scheme"[unvarify x, THEN "≡E"(1), THEN "&E"(1)]  by blast
  AOT_hence ‹𝒜∀F (⊤[F] ≡ ∃p(p & F = [λy p]))›
    using "Act-Basic:2" "&E"(2) "≡E"(1) by blast
  AOT_hence ‹∀F 𝒜(⊤[F] ≡ ∃p(p & F = [λy p]))›
    using "≡E"(1) "logic-actual-nec:3"[axiom_inst] by blast
  AOT_hence ‹𝒜(⊤[λx ∀q(q → q)] ≡ ∃p(p & [λx ∀q(q → q)] = [λy p]))›
```

```
    using "∀E"(1)[rotated, OF "prop-prop2:2"] by blast
  moreover AOT_have ‹𝒜∃p(p & [λx ∀q(q → q)] = [λy p])›
    by (safe intro!: "nec-imp-act"[THEN "→E"] RN "∃I"(1)[where τ="«∀q(q → q)»"] "&I"
                     GEN "→I" "log-prop-prop:2" "rule=I:1" "prop-prop2:2")
  ultimately AOT_have ‹𝒜(⊤[λx ∀q(q → q)])›
    using "Act-Basic:5" "≡E"(1,2) by blast
  AOT_thus ‹⊤[λx ∀q(q → q)]›
    using "en-eq:10[1]"[unvarify x₁ F, THEN "≡E"(1)] true_den "prop-prop2:2" by blast

  AOT_show ‹¬⊥[λx ∀q(q → q)]›
  proof(rule "raa-cor:2")
    AOT_assume ‹⊥[λx ∀q(q → q)]›
    AOT_hence ‹𝒜⊥[λx ∀q(q → q)]›
      using "en-eq:10[1]"[unvarify x₁ F, THEN "≡E"(2)]
            false_den "prop-prop2:2" by blast
    AOT_hence ‹𝒜∃p(¬p & [λx ∀q(q → q)] = [λy p])›
      using false_enc_cond "Act-Basic:5" "≡E"(1) by blast
    AOT_hence ‹∃p 𝒜(¬p & [λx ∀q(q → q)] = [λy p])›
      using "Act-Basic:10" "≡E"(1) by blast
    then AOT_obtain p where p_prop: ‹𝒜(¬p & [λx ∀q(q → q)] = [λy p])›
      using "∃E"[rotated] by blast
    AOT_hence ‹𝒜[λx ∀q(q → q)] = [λy p]›
      by (metis "Act-Basic:2" "&E"(2) "≡E"(1))
    AOT_hence ‹[λx ∀q(q → q)] = [λy p]›
      using "id-act:1"[unvarify α β, THEN "≡E"(2)] "prop-prop2:2" by blast
    AOT_hence ‹(∀q(q → q)) = p›
      using "p-identity-thm2:3"[unvarify p, THEN "≡E"(2)]
            "log-prop-prop:2" by blast
    moreover AOT_have ‹𝒜¬p› using p_prop
      using "Act-Basic:2" "&E"(1) "≡E"(1) by blast
    ultimately AOT_have ‹𝒜¬∀q(q → q)›
      by (metis "Act-Sub:1" "≡E"(1,2) "raa-cor:3" "rule=E")
    moreover AOT_have ‹¬𝒜¬∀q(q → q)›
      by (meson "Act-Sub:1" "RA[2]" "if-p-then-p" "≡E"(1) "universal-cor")
    ultimately AOT_show ‹𝒜¬∀q(q → q) & ¬𝒜¬∀q(q → q)›
      using "&I" by blast
  qed
qed

AOT_act_theorem "T-T-value:1": ‹TruthValue(⊤)›                                   (297.1)
proof -
  AOT_have true_def: ‹⊤ = ιx (A!x & ∀F (x[F] ≡ ∃p(p & F = [λy p])))›
    by (simp add: "A-descriptions" "rule-id-df:1[zero]" "the-true:1")
  AOT_hence true_den: ‹⊤↓›
    using "t=t-proper:1" "vdash-properties:6" by blast
  AOT_show ‹TruthValue(⊤)›
    using "y-in:2"[unvarify z, OF true_den, THEN "→E", OF true_def]
          "TV-lem2:1"[unvarify x, OF true_den, THEN "→E"] by blast
qed

AOT_act_theorem "T-T-value:2": ‹TruthValue(⊥)›                                   (297.2)
proof -
  AOT_have false_def: ‹⊥ = ιx (A!x & ∀F (x[F] ≡ ∃p(¬p & F = [λy p])))›
    by (simp add: "A-descriptions" "rule-id-df:1[zero]" "the-true:2")
  AOT_hence false_den: ‹⊥↓›
    using "t=t-proper:1" "vdash-properties:6" by blast
  AOT_show ‹TruthValue(⊥)›
    using "y-in:2"[unvarify z, OF false_den, THEN "→E", OF false_def]
          "TV-lem2:2"[unvarify x, OF false_den, THEN "→E"] by blast
qed

AOT_theorem "two-T": ‹∃x∃y(TruthValue(x) & TruthValue(y) & x ≠ y &                 (298)
                          ∀z (TruthValue(z) → z = x ∨ z = y))›
```

```
proof -
  AOT_obtain a where a_prop: ‹A!a & ∀F (a[F] ≡ ∃p (p & F = [λy p]))›
    using "A-objects"[axiom_inst] "∃E"[rotated] by fast
  AOT_obtain b where b_prop: ‹A!b & ∀F (b[F] ≡ ∃p (¬p & F = [λy p]))›
    using "A-objects"[axiom_inst] "∃E"[rotated] by fast
  AOT_obtain p where p: p
    by (metis "log-prop-prop:2" "raa-cor:3" "rule-ui:1" "universal-cor")
  show ?thesis
  proof(rule "∃I"(2)[where β=a]; rule "∃I"(2)[where β=b];
        safe intro!: "&I" GEN "→I")
    AOT_show ‹TruthValue(a)›
      using "TV-lem2:1" a_prop "vdash-properties:10" by blast
  next
    AOT_show ‹TruthValue(b)›
      using "TV-lem2:2" b_prop "vdash-properties:10" by blast
  next
    AOT_show ‹a ≠ b›
    proof(rule "ab-obey:2"[THEN "→E", OF "∨I"(1)])
      AOT_show ‹∃F (a[F] & ¬b[F])›
      proof(rule "∃I"(1)[where τ="«[λy p]»"]; rule "&I" "prop-prop2:2")
        AOT_show ‹a[λy p]›
          by(safe intro!: "∃I"(2)[where β=p] "&I" p "rule=I:1"[OF "prop-prop2:2"]
              a_prop[THEN "&E"(2), THEN "∀E"(1), THEN "≡E"(2), OF "prop-prop2:2"])
      next
        AOT_show ‹¬b[λy p]›
        proof (rule "raa-cor:2")
          AOT_assume ‹b[λy p]›
          AOT_hence ‹∃q (¬q & [λy p] = [λy q])›
            using "∀E"(1)[rotated, OF "prop-prop2:2", THEN "≡E"(1)]
                  b_prop[THEN "&E"(2)] by fast
          then AOT_obtain q where ‹¬q & [λy p] = [λy q]›
            using "∃E"[rotated] by blast
          AOT_hence ‹¬p›
            by (metis "rule=E" "&E"(1) "&E"(2) "deduction-theorem" "≡I"
                      "≡E"(2) "p-identity-thm2:3" "raa-cor:3")
          AOT_thus ‹p & ¬p› using p "&I" by blast
        qed
      qed
    qed
  next
    fix z
    AOT_assume ‹TruthValue(z)›
    AOT_hence ‹∃p (TruthValueOf(z, p))›
      by (metis "≡dfE" "T-value")
    then AOT_obtain p where ‹TruthValueOf(z, p)› using "∃E"[rotated] by blast
    AOT_hence z_prop: ‹A!z & ∀F (z[F] ≡ ∃q ((q ≡ p) & F = [λy q]))›
      using "≡dfE" "tv-p" by blast
    {
      AOT_assume p: ‹p›
      AOT_have ‹z = a›
      proof(rule "ab-obey:1"[THEN "→E", THEN "→E", OF "&I",
                            OF z_prop[THEN "&E"(1)], OF a_prop[THEN "&E"(1)]];
            rule GEN)
        fix G
        AOT_have ‹z[G] ≡ ∃q ((q ≡ p) & G = [λy q])›
          using z_prop[THEN "&E"(2)] "∀E"(2) by blast
        also AOT_have ‹∃q ((q ≡ p) & G = [λy q]) ≡ ∃q (q & G = [λy q])›
          using "TV-lem1:1"[THEN "≡E"(1), OF p, THEN "∀E"(2)[where β=G], symmetric].
        also AOT_have ‹... ≡ a[G]›
          using a_prop[THEN "&E"(2), THEN "∀E"(2)[where β=G], symmetric].
        finally AOT_show ‹z[G] ≡ a[G]›.
      qed
      AOT_hence ‹z = a ∨ z = b› by (rule "∨I")
```

```
    }
    moreover {
      AOT_assume notp: ‹¬p›
      AOT_have ‹z = b›
      proof(rule "ab-obey:1"[THEN "→E", THEN "→E", OF "&I",
                             OF z_prop[THEN "&E"(1)], OF b_prop[THEN "&E"(1)]];
            rule GEN)
        fix G
        AOT_have ‹z[G] ≡ ∃q ((q ≡ p) & G = [λy q])›
          using z_prop[THEN "&E"(2)] "∀E"(2) by blast
        also AOT_have ‹∃q ((q ≡ p) & G = [λy q]) ≡ ∃q (¬q & G = [λy q])›
          using "TV-lem1:2"[THEN "≡E"(1), OF notp, THEN "∀E"(2), symmetric].
        also AOT_have ‹... ≡ b[G]›
          using b_prop[THEN "&E"(2), THEN "∀E"(2), symmetric].
        finally AOT_show ‹z[G] ≡ b[G]›.
      qed
      AOT_hence ‹z = a ∨ z = b› by (rule "∨I")
    }
    ultimately AOT_show ‹z = a ∨ z = b›
      by (metis "reductio-aa:1")
  qed
qed

AOT_act_theorem "valueof-facts:1": ‹TruthValueOf(x, p) → (p ≡ x = ⊤)›             (299.1)
proof(safe intro!: "→I" dest!: "tv-p"[THEN "≡dfE"])
  AOT_assume ϑ: ‹[A!]x & ∀F (x[F] ≡ ∃q ((q ≡ p) & F = [λy q]))›
  AOT_have a: ‹A!⊤›
    using "∃E" "T-T-value:1" "T-value" "&E"(1) "≡dfE" "tv-p" by blast
  AOT_have true_def: ‹⊤ = ιx (A!x & ∀F (x[F] ≡ ∃p(p & F = [λy p])))›
    by (simp add: "A-descriptions" "rule-id-df:1[zero]" "the-true:1")
  AOT_hence true_den: ‹⊤↓›
    using "t=t-proper:1" "vdash-properties:6" by blast
  AOT_have b: ‹∀F (⊤[F] ≡ ∃q (q & F = [λy q]))›
    using "y-in:2"[unvarify z, OF true_den, THEN "→E", OF true_def] "&E" by blast
  AOT_show ‹p ≡ x = ⊤›
  proof(safe intro!: "≡I" "→I")
    AOT_assume p
    AOT_hence ‹∀F (∃q (q & F = [λy q]) ≡ ∃q ((q ≡ p) & F = [λy q]))›
      using "TV-lem1:1"[THEN "≡E"(1)] by blast
    AOT_hence ‹∀F(⊤[F] ≡ ∃q ((q ≡ p) & F = [λy q]))›
      using b "cqt-basic:10"[THEN "→E", OF "&I", OF b] by fast
    AOT_hence c: ‹∀F(∃q((q ≡ p) & F = [λy q]) ≡ ⊤[F])›
      using "cqt-basic:11"[THEN "≡E"(1)] by fast
    AOT_hence ‹∀F (x[F] ≡ ⊤[F])›
      using "cqt-basic:10"[THEN "→E", OF "&I", OF ϑ[THEN "&E"(2)]] by fast
    AOT_thus ‹x = ⊤›
      by (rule "ab-obey:1"[unvarify y, OF true_den, THEN "→E", THEN "→E",
                           OF "&I", OF ϑ[THEN "&E"(1)], OF a])
  next
    AOT_assume ‹x = ⊤›
    AOT_hence d: ‹∀F (⊤[F] ≡ ∃q ((q ≡ p) & F = [λy q]))›
      using "rule=E" ϑ[THEN "&E"(2)] by fast
    AOT_have ‹∀F (∃q (q & F = [λy q]) ≡ ∃q ((q ≡ p) & F = [λy q]))›
      using "cqt-basic:10"[THEN "→E", OF "&I",
              OF b[THEN "cqt-basic:11"[THEN "≡E"(1)]], OF d].
    AOT_thus p using "TV-lem1:1"[THEN "≡E"(2)] by blast
  qed
qed

AOT_act_theorem "valueof-facts:2": ‹TruthValueOf(x, p) → (¬p ≡ x = ⊥)›            (299.2)
proof(safe intro!: "→I" dest!: "tv-p"[THEN "≡dfE"])
  AOT_assume ϑ: ‹[A!]x & ∀F (x[F] ≡ ∃q ((q ≡ p) & F = [λy q]))›
  AOT_have a: ‹A!⊥›
```

```
    using "∃E" "T-T-value:2" "T-value" "&E"(1) "≡dfE" "tv-p" by blast
  AOT_have false_def: ‹⊥ = ιx (A!x & ∀F (x[F] ≡ ∃p(¬p & F = [λy p])))›
    by (simp add: "A-descriptions" "rule-id-df:1[zero]" "the-true:2")
  AOT_hence false_den: ‹⊥↓›
    using "t=t-proper:1" "vdash-properties:6" by blast
  AOT_have b: ‹∀F (⊥[F] ≡ ∃q (¬q & F = [λy q]))›
    using "y-in:2"[unvarify z, OF false_den, THEN "→E", OF false_def] "&E" by blast
  AOT_show ‹¬p ≡ x = ⊥›
  proof(safe intro!: "≡I" "→I")
    AOT_assume ‹¬p›
    AOT_hence ‹∀F (∃q (¬q & F = [λy q]) ≡ ∃q ((q ≡ p) & F = [λy q]))›
      using "TV-lem1:2"[THEN "≡E"(1)] by blast
    AOT_hence ‹∀F(⊥[F] ≡ ∃q ((q ≡ p) & F = [λy q]))›
      using b "cqt-basic:10"[THEN "→E", OF "&I", OF b] by fast
    AOT_hence c: ‹∀F(∃q((q ≡ p) & F = [λy q]) ≡ ⊥[F])›
      using "cqt-basic:11"[THEN "≡E"(1)] by fast
    AOT_hence ‹∀F (x[F] ≡ ⊥[F])›
      using "cqt-basic:10"[THEN "→E", OF "&I", OF ϑ[THEN "&E"(2)]] by fast
    AOT_thus ‹x = ⊥›
      by (rule "ab-obey:1"[unvarify y, OF false_den, THEN "→E", THEN "→E",
                           OF "&I", OF ϑ[THEN "&E"(1)], OF a])
  next
    AOT_assume ‹x = ⊥›
    AOT_hence d: ‹∀F (⊥[F] ≡ ∃q ((q ≡ p) & F = [λy q]))›
      using "rule=E" ϑ[THEN "&E"(2)] by fast
    AOT_have ‹∀F (∃q (¬q & F = [λy q]) ≡ ∃q ((q ≡ p) & F = [λy q]))›
      using "cqt-basic:10"[THEN "→E", OF "&I",
                 OF b[THEN "cqt-basic:11"[THEN "≡E"(1)]], OF d].
    AOT_thus ‹¬p› using "TV-lem1:2"[THEN "≡E"(2)] by blast
  qed
qed

AOT_act_theorem "q-True:1": ‹p ≡ (∘p = ⊤)›                                    (300.1)
  apply (rule "valueof-facts:1"[unvarify x, THEN "→E", rotated, OF "T-lem:1"])
  using "≡dfE" "tv-id:2" "&E"(1) "prop-enc" by blast

AOT_act_theorem "q-True:2": ‹¬p ≡ (∘p = ⊥)›                                   (300.2)
  apply (rule "valueof-facts:2"[unvarify x, THEN "→E", rotated, OF "T-lem:1"])
  using "≡dfE" "tv-id:2" "&E"(1) "prop-enc" by blast

AOT_act_theorem "q-True:3": ‹p ≡ ⊤Σp›                                         (300.3)
proof(safe intro!: "≡I" "→I")
  AOT_assume p
  AOT_hence ‹∘p = ⊤› by (metis "≡E"(1) "q-True:1")
  moreover AOT_have ‹∘pΣp›
    by (simp add: "tv-id:2")
  ultimately AOT_show ‹⊤Σp›
    using "rule=E" "T-lem:4" by fast
next
  AOT_have true_def: ‹⊤ = ιx (A!x & ∀F (x[F] ≡ ∃p(p & F = [λy p])))›
    by (simp add: "A-descriptions" "rule-id-df:1[zero]" "the-true:1")
  AOT_hence true_den: ‹⊤↓›
    using "t=t-proper:1" "vdash-properties:6" by blast
  AOT_have b: ‹∀F (⊤[F] ≡ ∃q (q & F = [λy q]))›
    using "y-in:2"[unvarify z, OF true_den, THEN "→E", OF true_def] "&E" by blast

  AOT_assume ‹⊤Σp›
  AOT_hence ‹⊤[λy p]› by (metis "≡dfE" "&E"(2) "prop-enc")
  AOT_hence ‹∃q (q & [λy p] = [λy q])›
    using b[THEN "∀E"(1), OF "prop-prop2:2", THEN "≡E"(1)] by blast
  then AOT_obtain q where ‹q & [λy p] = [λy q]› using "∃E"[rotated] by blast
  AOT_thus ‹p›
    using "rule=E" "&E"(1) "&E"(2) id_sym "≡E"(2) "p-identity-thm2:3" by fast
```

```
qed


AOT_act_theorem "q-True:5": ‹¬p ≡ ⊥Σp›                                         (300.5)
proof(safe intro!: "≡I" "→I")
  AOT_assume ‹¬p›
  AOT_hence ‹◦p = ⊥› by (metis "≡E"(1) "q-True:2")
  moreover AOT_have ‹◦pΣp›
    by (simp add: "tv-id:2")
  ultimately AOT_show ‹⊥Σp›
    using "rule=E" "T-lem:4" by fast
next
  AOT_have false_def: ‹⊥ = ιx (A!x & ∀F (x[F] ≡ ∃p(¬p & F = [λy p])))›
    by (simp add: "A-descriptions" "rule-id-df:1[zero]" "the-true:2")
  AOT_hence false_den: ‹⊥↓›
    using "t=t-proper:1" "vdash-properties:6" by blast
  AOT_have b: ‹∀F (⊥[F] ≡ ∃q (¬q & F = [λy q]))›
    using "y-in:2"[unvarify z, OF false_den, THEN "→E", OF false_def] "&E" by blast

  AOT_assume ‹⊥Σp›
  AOT_hence ‹⊥[λy p]› by (metis "≡dfE" "&E"(2) "prop-enc")
  AOT_hence ‹∃q (¬q & [λy p] = [λy q])›
    using b[THEN "∀E"(1), OF "prop-prop2:2", THEN "≡E"(1)] by blast
  then AOT_obtain q where ‹¬q & [λy p] = [λy q]› using "∃E"[rotated] by blast
  AOT_thus ‹¬p›
    using "rule=E" "&E"(1) "&E"(2) id_sym "≡E"(2) "p-identity-thm2:3" by fast
qed

AOT_act_theorem "q-True:4": ‹p ≡ ¬(⊥Σp)›                                       (300.4)
  using "q-True:5"
  by (metis "deduction-theorem" "≡I" "≡E"(2) "≡E"(4) "raa-cor:3")

AOT_act_theorem "q-True:6": ‹¬p ≡ ¬(⊤Σp)›                                      (300.6)
  using "≡E"(1) "oth-class-taut:4:b" "q-True:3" by blast

AOT_define ExtensionOf :: ‹τ ⇒ φ ⇒ φ› (‹ExtensionOf'(_,_')›)
  "exten-p": ‹ExtensionOf(x,p) ≡df A!x &                                       (301)
                                     ∀F (x[F] → Propositional([F])) &
                                     ∀q ((xΣq) ≡ (q ≡ p))›

AOT_theorem "extof-e": ‹ExtensionOf(x, p) ≡ TruthValueOf(x, p)›                (302)
proof (safe intro!: "≡I" "→I" "tv-p"[THEN "≡dfI"] "exten-p"[THEN "≡dfI"]
            dest!: "tv-p"[THEN "≡dfE"] "exten-p"[THEN "≡dfE"])
  AOT_assume 1: ‹[A!]x & ∀F (x[F] → Propositional([F])) & ∀q (x Σ q ≡ (q ≡ p))›
  AOT_hence ϑ: ‹[A!]x & ∀F (x[F] → ∃q(F = [λy q])) & ∀q (x Σ q ≡ (q ≡ p))›
    by (AOT_subst ‹∃q(F = [λy q])› ‹Propositional([F])› for: F :: ‹<κ>›)
       (auto simp add: "df-rules-formulas[3]" "df-rules-formulas[4]"
                       "≡I" "prop-prop1")
  AOT_show ‹[A!]x & ∀F (x[F] ≡ ∃q ((q ≡ p) & F = [λy q]))›
  proof(safe intro!: "&I" GEN 1[THEN "&E"(1), THEN "&E"(1)] "≡I" "→I")
    fix F
    AOT_assume 0: ‹x[F]›
    AOT_hence ‹∃q (F = [λy q])›
      using ϑ[THEN "&E"(1), THEN "&E"(2)] "∀E"(2) "→E" by blast
    then AOT_obtain q where q_prop: ‹F = [λy q]› using "∃E"[rotated] by blast
    AOT_hence ‹x[λy q]› using 0 "rule=E" by blast
    AOT_hence ‹xΣq› by (metis "≡dfI" "&I" "ex:1:a" "prop-enc" "rule-ui:3")
    AOT_hence ‹q ≡ p› using ϑ[THEN "&E"(2)] "∀E"(2) "≡E"(1) by blast
    AOT_hence ‹(q ≡ p) & F = [λy q]› using q_prop "&I" by blast
    AOT_thus ‹∃q ((q ≡ p) & F = [λy q])› by (rule "∃I")
  next
    fix F
    AOT_assume ‹∃q ((q ≡ p) & F = [λy q])›
```

```
    then AOT_obtain q where q_prop: ‹(q ≡ p) & F = [λy q]›
      using "∃E"[rotated] by blast
    AOT_hence ‹xΣq› using ϑ[THEN "&E"(2)] "∀E"(2) "&E" "≡E"(2) by blast
    AOT_hence ‹x[λy q]› by (metis "≡dfE" "&E"(2) "prop-enc")
    AOT_thus ‹x[F]› using q_prop[THEN "&E"(2), symmetric] "rule=E" by blast
  qed
next
  AOT_assume 0: ‹[A!]x & ∀F (x[F] ≡ ∃q ((q ≡ p) & F = [λy q]))›
  AOT_show ‹[A!]x & ∀F (x[F] → Propositional([F])) & ∀q (x Σ q ≡ (q ≡ p))›
  proof(safe intro!: "&I" 0[THEN "&E"(1)] GEN "→I")
    fix F
    AOT_assume ‹x[F]›
    AOT_hence ‹∃q ((q ≡ p) & F = [λy q])›
      using 0[THEN "&E"(2)] "∀E"(2) "≡E"(1) by blast
    then AOT_obtain q where ‹(q ≡ p) & F = [λy q]›
      using "∃E"[rotated] by blast
    AOT_hence ‹F = [λy q]› using "&E"(2) by blast
    AOT_hence ‹∃q F = [λy q]› by (rule "∃I")
    AOT_thus ‹Propositional([F])› by (metis "≡dfI" "prop-prop1")
  next
    AOT_show ‹xΣr ≡ (r ≡ p)› for r
    proof(rule "≡I"; rule "→I")
      AOT_assume ‹xΣr›
      AOT_hence ‹x[λy r]› by (metis "≡dfE" "&E"(2) "prop-enc")
      AOT_hence ‹∃q ((q ≡ p) & [λy r] = [λy q])›
        using 0[THEN "&E"(2), THEN "∀E"(1), OF "prop-prop2:2", THEN "≡E"(1)] by blast
      then AOT_obtain q where ‹(q ≡ p) & [λy r] = [λy q]›
        using "∃E"[rotated] by blast
      AOT_thus ‹r ≡ p›
        by (metis "rule=E" "&E"(1,2) id_sym "≡E"(2) "Commutativity of ≡"
                  "p-identity-thm2:3")
    next
      AOT_assume ‹r ≡ p›
      AOT_hence ‹(r ≡ p) & [λy r] = [λy r]›
        by (metis "rule=I:1" "≡S"(1) "≡E"(2) "Commutativity of &" "prop-prop2:2")
      AOT_hence ‹∃q ((q ≡ p) & [λy r] = [λy q])› by (rule "∃I")
      AOT_hence ‹x[λy r]›
        using 0[THEN "&E"(2), THEN "∀E"(1), OF "prop-prop2:2", THEN "≡E"(2)] by blast
      AOT_thus ‹xΣr› by (metis "≡dfI" "&I" "ex:1:a" "prop-enc" "rule-ui:3")
    qed
  qed
qed

AOT_theorem "ext-p-tv:1": ‹∃!x ExtensionOf(x, p)›                          (303.1)
  by (AOT_subst ‹ExtensionOf(x, p)› ‹TruthValueOf(x, p)› for: x)
     (auto simp: "extof-e" "p-has-!tv:2")

AOT_theorem "ext-p-tv:2": ‹ιx(ExtensionOf(x, p))↓›                         (303.2)
  using "A-Exists:2" "RA[2]" "ext-p-tv:1" "≡E"(2) by blast

AOT_theorem "ext-p-tv:3": ‹ιx(ExtensionOf(x, p)) = ◦p›                     (303.3)
proof -
  AOT_have 0: ‹𝒜∀x(ExtensionOf(x, p) ≡ TruthValueOf(x,p))›
    by (rule "RA[2]"; rule GEN; rule "extof-e")
  AOT_have 1: ‹◦p = ιx TruthValueOf(x,p)›
    using "rule-id-df:1" "the-tv-p" "uni-tv" by blast
  show ?thesis
    apply (rule "equiv-desc-eq:1"[THEN "→E", OF 0, THEN "∀E"(1)[where τ=‹«◦p»›],
                                  THEN "≡E"(2), symmetric])
    using "1" "t=t-proper:1" "vdash-properties:10" apply blast
    by (fact 1)
qed
```

```
(*<*)end(*>*)
```

A.9. Restricted Variables

```
(*<*)
theory AOT_RestrictedVariables
  imports AOT_PLM
  keywords "AOT_register_rigid_restricted_type" :: thy_goal
       and "AOT_register_restricted_type" :: thy_goal
begin
(*>*)

section‹Restricted Variables›

locale AOT_restriction_condition =
  fixes ψ :: ‹'a::AOT_Term_id_2 ⇒ o›
  assumes "res-var:2"[AOT]: ‹[v ⊨ ∃α ψ{α}]›                                (330.2)
  assumes "res-var:3"[AOT]: ‹[v ⊨ ψ{τ} → τ↓]›                              (330.3)

ML‹
fun register_restricted_type (name:string, restriction:string) thy =
let
val ctxt = thy
val ctxt = setupStrictWorld ctxt
val trm = Syntax.check_term ctxt (AOT_read_term @{nonterminal φ'} ctxt restriction)
val free = case (Term.add_frees trm []) of [f] => f |
    _ => raise Term.TERM ("Expected single free variable.", [trm])
val trm = Term.absfree free trm
val localeTerm = Const (const_name‹AOT_restriction_condition›, dummyT) $ trm
val localeTerm = Syntax.check_term ctxt localeTerm
fun after_qed thms thy = let
val st = Interpretation.global_interpretation
  (([(@{locale AOT_restriction_condition}, ((name, true),
            (Expression.Named [("ψ", trm)], [])))], [])) [] thy
val st = Proof.refine_insert (flat thms) st
val thy = Proof.global_immediate_proof st

val thy = Local_Theory.background_theory
  (AOT_Constraints.map (Symtab.update
    (name, (term_of (snd free), term_of (snd free))))) thy
val thy = Local_Theory.background_theory
  (AOT_Restriction.map (Symtab.update
    (name, (trm, Const (const_name‹AOT_term_of_var›, dummyT))))) thy

in thy end
in
Proof.theorem NONE after_qed [[(HOLogic.mk_Trueprop localeTerm, [])]] ctxt
end

val _ =
  Outer_Syntax.command
    command_keyword‹AOT_register_restricted_type›
    "Register a restricted type."
    (((Parse.short_ident -| Parse.$$$ ":") - Parse.term) >>
    (Toplevel.local_theory_to_proof NONE NONE o register_restricted_type));
›

locale AOT_rigid_restriction_condition = AOT_restriction_condition +
  assumes rigid[AOT]: ‹[v ⊨ ∀α(ψ{α} → □ψ{α})]›
begin
lemma rigid_condition[AOT]: ‹[v ⊨ □(ψ{α} → □ψ{α})]›
  using rigid[THEN "∀E"(2)] RN by simp
lemma type_set_nonempty[AOT_no_atp, no_atp]: ‹∃x . x ∈ { α . [w₀ ⊨ ψ{α}]}›
  by (metis "instantiation" mem_Collect_eq "res-var:2")
end
```

```
locale AOT_restricted_type = AOT_rigid_restriction_condition +
  fixes Rep and Abs
  assumes AOT_restricted_type_definition[AOT_no_atp]:
    ‹type_definition Rep Abs { α . [w₀ ⊨ ψ{α}]}›
begin

AOT_theorem restricted_var_condition: ‹ψ{«AOT_term_of_var (Rep α)»}›
proof -
  interpret type_definition Rep Abs "{ α . [w₀ ⊨ ψ{α}]}"
    using AOT_restricted_type_definition.
  AOT_actually {
    AOT_have ‹«AOT_term_of_var (Rep α)»↓› and ‹ψ{«AOT_term_of_var (Rep α)»}›
      using AOT_sem_imp Rep "res-var:3" by auto
  }
  moreover AOT_actually {
    AOT_have ‹ψ{α} → □ψ{α}› for α
      using AOT_sem_box rigid_condition by presburger
    AOT_hence ‹ψ{τ} → □ψ{τ}› if ‹τ↓› for τ
      by (metis AOT_model.AOT_term_of_var_cases AOT_sem_denotes that)
  }
  ultimately AOT_show ‹ψ{«AOT_term_of_var (Rep α)»}›
    using AOT_sem_box AOT_sem_imp by blast
qed
lemmas "ψ" = restricted_var_condition

AOT_theorem GEN: assumes ‹for arbitrary α: φ{«AOT_term_of_var (Rep α)»}›
  shows ‹∀α (ψ{α} → φ{α})›
proof(rule GEN; rule "→I")
  interpret type_definition Rep Abs "{ α . [w₀ ⊨ ψ{α}]}"
    using AOT_restricted_type_definition.
  fix α
  AOT_assume ‹ψ{α}›
  AOT_hence ‹𝓐ψ{α}›
    by (metis AOT_model_axiom_def AOT_sem_box AOT_sem_imp act_closure rigid_condition)
  hence 0: ‹[w₀ ⊨ ψ{α}]› by (metis AOT_sem_act)
  {
    fix τ
    assume α_def: ‹α = Rep τ›
    AOT_have ‹φ{α}›
      unfolding α_def
      using assms by blast
  }
  AOT_thus ‹φ{α}›
    using Rep_cases[simplified, OF 0]
    by blast
qed
lemmas "∀I" = GEN

end


lemma AOT_restricted_type_intro[AOT_no_atp, no_atp]:
  assumes ‹type_definition Rep Abs { α . [w₀ ⊨ ψ{α}]}›
      and ‹AOT_rigid_restriction_condition ψ›
  shows ‹AOT_restricted_type ψ Rep Abs›
  by (auto intro!: assms AOT_restricted_type_axioms.intro AOT_restricted_type.intro)



ML‹
fun register_rigid_restricted_type (name:string, restriction:string) thy =
let
```

```
val ctxt = thy
val ctxt = setupStrictWorld ctxt
val trm = Syntax.check_term ctxt (AOT_read_term @{nonterminal φ'} ctxt restriction)
val free = case (Term.add_frees trm []) of [f] => f
  | _ => raise Term.TERM ("Expected single free variable.", [trm])
val trm = Term.absfree free trm
val localeTerm = HOLogic.mk_Trueprop
  (Const (const_name‹AOT_rigid_restriction_condition›, dummyT) $ trm)
val localeTerm = Syntax.check_prop ctxt localeTerm
val int_bnd = Binding.concealed (Binding.qualify true "internal" (Binding.name name))
val bnds = {Rep_name = Binding.qualify true name (Binding.name "Rep"),
            Abs_name = Binding.qualify true "Abs" int_bnd,
            type_definition_name = Binding.qualify true "type_definition" int_bnd}

fun after_qed witts thy = let
  val thms = (map (Element.conclude_witness ctxt) (flat witts))

  val typeset = HOLogic.mk_Collect ("α", dummyT,
    const‹AOT_model_valid_in› $ const‹w₀› $
    (trm $ (Const (const_name‹AOT_term_of_var›, dummyT) $ Bound 0)))
  val typeset = Syntax.check_term thy typeset
  val nonempty_thm = Drule.OF
    (@{thm AOT_rigid_restriction_condition.type_set_nonempty}, thms)

  val ((_,st),thy) = Typedef.add_typedef {overloaded=true}
    (Binding.name name, [], Mixfix.NoSyn) typeset (SOME bnds)
    (fn ctxt => (Tactic.resolve_tac ctxt ([nonempty_thm]) 1)) thy
  val ({rep_type = _, abs_type = _, Rep_name = Rep_name, Abs_name = Abs_name,
        axiom_name = _},
     {inhabited = _, type_definition = type_definition, Rep = _,
      Rep_inverse = _, Abs_inverse = _, Rep_inject = _, Abs_inject = _,
      Rep_cases = _, Abs_cases = _, Rep_induct = _, Abs_induct = _}) = st

  val locale_thm = Drule.OF (@{thm AOT_restricted_type_intro}, type_definition::thms)

  val st = Interpretation.global_interpretation (([(@{locale AOT_restricted_type},
    ((name, true), (Expression.Named [
      ("ψ", trm),
      ("Rep", Const (Rep_name, dummyT)),
      ("Abs", Const (Abs_name, dummyT))], [])))
    ], [])) [] thy

  val st = Proof.refine_insert [locale_thm] st
  val thy = Proof.global_immediate_proof st

  val thy = Local_Theory.background_theory (AOT_Constraints.map (
    Symtab.update (name, (term_of (snd free), term_of (snd free))))) thy
  val thy = Local_Theory.background_theory (AOT_Restriction.map (
    Symtab.update (name, (trm, Const (Rep_name, dummyT))))) thy

  in thy end
in
Element.witness_proof after_qed [[localeTerm]] thy
end

val _ =
  Outer_Syntax.command
    command_keyword‹AOT_register_rigid_restricted_type›
    "Register a restricted type."
    (((Parse.short_ident --| Parse.$$$ ":") -- Parse.term) >>
    (Toplevel.local_theory_to_proof NONE NONE o register_rigid_restricted_type));
›
```

```
(* Generalized mechanism for "AOT_restricted_type.∀I" followed by ∀E *)
ML‹
fun get_instantiated_allI ctxt varname thm = let
val trm = Thm.concl_of thm
val trm = case trm of (@{const Trueprop} $ (@{const AOT_model_valid_in} $ _ $ x)) => x
                      | _ => raise Term.TERM ("Expected simple theorem.", [trm])
fun extractVars (Const (const_name‹AOT_term_of_var›, t) $ (Const rep $ Var v)) =
    (if fst (fst v) = fst varname
     then [Const (const_name‹AOT_term_of_var›, t) $ (Const rep $ Var v)]
     else []) (* TODO: care about the index *)
  | extractVars (t1 $ t2) = extractVars t1 @ extractVars t2
  | extractVars (Abs (_, _, t)) = extractVars t
  | extractVars _ = []
val vars = extractVars trm
val vartrm = hd vars
val vars = fold Term.add_vars vars []
val var = hd vars
val trmty = (case vartrm of (Const (_, Type ("fun", [_, ty])) $ _) => ty
                       | _ => raise Match)
val varty = snd var
val tyname = fst (Term.dest_Type varty)
val b = tyname^".∀I" (* TODO: better way to find the theorem *)
val thms = fst (Context.map_proof_result (fn ctxt => (Attrib.eval_thms ctxt
    [(Facts.Named ((b,Position.none),NONE),[])], ctxt)) ctxt)
val allthm = (case thms of (thm::_) => thm
    | _ => raise Fail "Unknown restricted type.")
val trm = Abs (Term.string_of_vname (fst var), trmty, Term.abstract_over (vartrm, trm))
val trm = Thm.cterm_of (Context.proof_of ctxt) trm
val phi = hd (Term.add_vars (Thm.prop_of allthm) [])
val allthm = Drule.instantiate_normalize (TVars.empty, Vars.make [(phi,trm)]) allthm
in
allthm
end
›

(* TODO: unconstraining multiple variables does not work yet *)
attribute_setup "unconstrain" =
  ‹Scan.lift (Scan.repeat1 Args.var) » (fn args => Thm.rule_attribute []
  (fn ctxt => fn thm =>
    let
    val thm = fold (fn arg => fn thm => thm RS get_instantiated_allI ctxt arg thm)
                   args thm
    val thm = fold (fn _ => fn thm => thm RS @{thm "∀E"(2)}) args thm
    in
     thm
    end))›
  "Generalize a statement about restricted variables to a statement about
   unrestricted variables with explicit restriction condition."



context AOT_restricted_type
begin

AOT_theorem "rule-ui":                                                        (93)
  assumes ‹∀α(ψ{α} → φ{α})›
  shows ‹φ{«AOT_term_of_var (Rep α)»}›
proof -
  AOT_have ‹φ{α}› if ‹ψ{α}› for α using assms[THEN "∀E"(2), THEN "→E"] that by blast
  moreover AOT_have ‹ψ{«AOT_term_of_var (Rep α)»}›
    by (auto simp: ψ)
  ultimately show ?thesis by blast
qed
```

```
lemmas "∀E" = "rule-ui"

AOT_theorem "instantiation":                                                    (102)
  assumes ‹for arbitrary β: φ{«AOT_term_of_var (Rep β)»} ⊢ χ› and ‹∃α (ψ{α} & φ{α})›
  shows ‹χ›
proof -
  AOT_have ‹φ{«AOT_term_of_var (Rep α)»} → χ› for α
    using assms(1)
    by (simp add: "deduction-theorem")
  AOT_hence 0: ‹∀α (ψ{α} → (φ{α} → χ))›
    using GEN by simp
  moreover AOT_obtain α where ‹ψ{α} & φ{α}› using assms(2) "∃E"[rotated] by blast
  ultimately AOT_show ‹χ› using "AOT_PLM.∀E"(2)[THEN "→E", THEN "→E"] "&E" by fast
qed
lemmas "∃E" = "instantiation"

AOT_theorem existential: assumes ‹φ{«AOT_term_of_var (Rep β)»}›                 (101)
  shows ‹∃ α (ψ{α} & φ{α})›
  by (meson AOT_restricted_type.ψ AOT_restricted_type_axioms assms
            "&I" "existential:2[const_var]")
lemmas "∃I" = existential
end


context AOT_rigid_restriction_condition
begin

AOT_theorem "res-var-bound-reas[1]":                                            (334)
  ‹∀α(ψ{α} → ∀β φ{α, β}) ≡ ∀β∀α (ψ{α} → φ{α, β})›
proof(safe intro!: "≡I" "→I" GEN)
  fix β α
  AOT_assume ‹∀α (ψ{α} → ∀β φ{α, β})›
  AOT_hence ‹ψ{α} → ∀β φ{α, β}› using "∀E"(2) by blast
  moreover AOT_assume ‹ψ{α}›
  ultimately AOT_have ‹∀β φ{α, β}› using "→E" by blast
  AOT_thus ‹φ{α, β}› using "∀E"(2) by blast
next
  fix α β
  AOT_assume ‹∀β∀α(ψ{α} → φ{α, β})›
  AOT_hence ‹∀α(ψ{α} → φ{α, β})› using "∀E"(2) by blast
  AOT_hence ‹ψ{α} → φ{α, β}› using "∀E"(2) by blast
  moreover AOT_assume ‹ψ{α}›
  ultimately AOT_show ‹φ{α, β}› using "→E" by blast
qed

AOT_theorem "res-var-bound-reas[BF]":                                           (334)
  ‹∀α(ψ{α} → □φ{α}) → □∀α(ψ{α} → φ{α})›
proof(safe intro!: "→I")
  AOT_assume ‹∀α(ψ{α} → □φ{α})›
  AOT_hence ‹ψ{α} → □φ{α}› for α
    using "∀E"(2) by blast
  AOT_hence ‹□(ψ{α} → φ{α})› for α
    by (metis "sc-eq-box-box:6" rigid_condition "vdash-properties:6")
  AOT_hence ‹∀α □(ψ{α} → φ{α})›
    by (rule GEN)
  AOT_thus ‹□∀α (ψ{α} → φ{α})›
    by (metis "BF" "vdash-properties:6")
qed

AOT_theorem "res-var-bound-reas[CBF]":                                          (334)
‹□∀α(ψ{α} → φ{α}) → ∀α(ψ{α} → □φ{α})›
proof(safe intro!: "→I" GEN)
  fix α
```

```
  AOT_assume ‹□∀α (ψ{α} → φ{α})›
  AOT_hence ‹∀α □(ψ{α} → φ{α})›
    by (metis "CBF" "vdash-properties:6")
  AOT_hence 1: ‹□(ψ{α} → φ{α})›
    using "∀E"(2) by blast
  AOT_assume ‹ψ{α}›
  AOT_hence ‹□ψ{α}›
    by (metis "B◇" "T◇" rigid_condition "vdash-properties:6")
  AOT_thus ‹□φ{α}›
    using 1 "qml:1"[axiom_inst, THEN "→E", THEN "→E"] by blast
qed

AOT_theorem "res-var-bound-reas[2]":                                    (334)
‹∀α (ψ{α} → 𝒜φ{α}) → 𝒜∀α (ψ{α} → φ{α})›
proof(safe intro!: "→I")
  AOT_assume ‹∀α (ψ{α} → 𝒜φ{α})›
  AOT_hence ‹ψ{α} → 𝒜φ{α}› for α
    using "∀E"(2) by blast
  AOT_hence ‹𝒜(ψ{α} → φ{α})› for α
    by (metis "sc-eq-box-box:7" rigid_condition "vdash-properties:6")
  AOT_hence ‹∀α 𝒜(ψ{α} → φ{α})›
    by (rule GEN)
  AOT_thus ‹𝒜∀α(ψ{α} → φ{α})›
    by (metis "≡E"(2) "logic-actual-nec:3"[axiom_inst])
qed


AOT_theorem "res-var-bound-reas[3]":                                    (334)
  ‹𝒜∀α (ψ{α} → φ{α}) → ∀α (ψ{α} → 𝒜φ{α})›
proof(safe intro!: "→I" GEN)
  fix α
  AOT_assume ‹𝒜∀α (ψ{α} → φ{α})›
  AOT_hence ‹∀α 𝒜(ψ{α} → φ{α})›
    by (metis "≡E"(1) "logic-actual-nec:3"[axiom_inst])
  AOT_hence 1: ‹𝒜(ψ{α} → φ{α})› by (metis "rule-ui:3")
  AOT_assume ‹ψ{α}›
  AOT_hence ‹𝒜ψ{α}›
    by (metis "nec-imp-act" "qml:2"[axiom_inst] rigid_condition "→E")
  AOT_thus ‹𝒜φ{α}›
    using 1 by (metis "act-cond" "→E")
qed

AOT_theorem "res-var-bound-reas[Buridan]":                              (334)
  ‹∃α (ψ{α} & □φ{α}) → □∃α (ψ{α} & φ{α})›
proof (rule "→I")
  AOT_assume ‹∃α (ψ{α} & □φ{α})›
  then AOT_obtain α where ‹ψ{α} & □φ{α}›
    using "∃E"[rotated] by blast
  AOT_hence ‹□(ψ{α} & φ{α})›
    by (metis "KBasic:11" "KBasic:3" "T◇" "&I" "&E"(1) "&E"(2)
              "≡E"(2) "reductio-aa:1" rigid_condition "vdash-properties:6")
  AOT_hence ‹∃α □(ψ{α} & φ{α})›
    by (rule "∃I")
  AOT_thus ‹□∃α (ψ{α} & φ{α})›
    by (rule Buridan[THEN "→E"])
qed

AOT_theorem "res-var-bound-reas[BF◇]":                                  (334)
  ‹◇∃α (ψ{α} & φ{α}) → ∃α (ψ{α} & ◇φ{α})›
proof(rule "→I")
  AOT_assume ‹◇∃α (ψ{α} & φ{α})›
  AOT_hence ‹∃α ◇(ψ{α} & φ{α})›
    using "BF◇"[THEN "→E"] by blast
```

```
  then AOT_obtain α where ‹◇(ψ{α} & φ{α})›
    using "∃E"[rotated] by blast
  AOT_hence ‹◇ψ{α}› and ‹◇φ{α}›
    using "KBasic2:3" "&E" "→E" by blast+
  moreover AOT_have ‹ψ{α}›
    using calculation rigid_condition by (metis "B◇" "K◇" "→E")
  ultimately AOT_have ‹ψ{α} & ◇φ{α}›
    using "&I" by blast
  AOT_thus ‹∃α (ψ{α} & ◇φ{α})›
    by (rule "∃I")
qed

AOT_theorem "res-var-bound-reas[CBF◇]":                                    (334)
  ‹∃α (ψ{α} & ◇φ{α}) → ◇∃α (ψ{α} & φ{α})›
proof(rule "→I")
  AOT_assume ‹∃α (ψ{α} & ◇φ{α})›
  then AOT_obtain α where ‹ψ{α} & ◇φ{α}›
    using "∃E"[rotated] by blast
  AOT_hence ‹□ψ{α}› and ‹◇φ{α}›
    using rigid_condition[THEN "qml:2"[axiom_inst, THEN "→E"], THEN "→E"] "&E" by blast+
  AOT_hence ‹◇(ψ{α} & φ{α})›
    by (metis "KBasic:16" "con-dis-taut:5" "→E")
  AOT_hence ‹∃α ◇(ψ{α} & φ{α})›
    by (rule "∃I")
  AOT_thus ‹◇∃α (ψ{α} & φ{α})›
    using "CBF◇"[THEN "→E"] by fast
qed

AOT_theorem "res-var-bound-reas[A-Exists:1]":                              (334)
  ‹𝒜∃!α (ψ{α} & φ{α}) ≡ ∃!α (ψ{α} & 𝒜φ{α})›
proof(safe intro!: "≡I" "→I")
  AOT_assume ‹𝒜∃!α (ψ{α} & φ{α})›
  AOT_hence ‹∃!α 𝒜(ψ{α} & φ{α})›
    using "A-Exists:1"[THEN "≡E"(1)] by blast
  AOT_hence ‹∃!α (𝒜ψ{α} & 𝒜φ{α})›
    apply(AOT_subst ‹𝒜ψ{α} & 𝒜φ{α}› ‹𝒜(ψ{α} & φ{α})› for: α)
     apply (meson "Act-Basic:2" "intro-elim:3:f" "oth-class-taut:3:a")
    by simp
  AOT_thus ‹∃!α (ψ{α} & 𝒜φ{α})›
    apply (AOT_subst ‹ψ{α}› ‹𝒜ψ{α}› for: α)
    using "Commutativity of ≡" "intro-elim:3:b" "sc-eq-fur:2"
          "→E" rigid_condition by blast
next
  AOT_assume ‹∃!α (ψ{α} & 𝒜φ{α})›
  AOT_hence ‹∃!α (𝒜ψ{α} & 𝒜φ{α})›
    apply (AOT_subst ‹𝒜ψ{α}› ‹ψ{α}› for: α)
     apply (meson "sc-eq-fur:2" "→E" rigid_condition)
    by simp
  AOT_hence ‹∃!α 𝒜(ψ{α} & φ{α})›
    apply (AOT_subst ‹𝒜(ψ{α} & φ{α})› ‹𝒜ψ{α} & 𝒜φ{α}› for: α)
     using "Act-Basic:2" apply presburger
     by simp
  AOT_thus ‹𝒜∃!α (ψ{α} & φ{α})›
     by (metis "A-Exists:1" "intro-elim:3:b")
qed

end

(*<*)
end
(*>*)
```

A.10. Extended Relation Comprehension

```
theory AOT_ExtendedRelationComprehension
  imports AOT_RestrictedVariables
begin

section‹Extended Relation Comprehension›

text‹This theory depends on choosing extended models.›
interpretation AOT_ExtendedModel by (standard; auto)

text‹Auxiliary lemma: negations of denoting relations denote.›
AOT_theorem negation_denotes: ‹[λx φ{x}]↓ → [λx ¬φ{x}]↓›
proof(rule "→I")
  AOT_assume 0: ‹[λx φ{x}]↓›
  AOT_show ‹[λx ¬φ{x}]↓›
  proof (rule "safe-ext"[axiom_inst, THEN "→E", OF "&I"])
    AOT_show ‹[λx ¬[λx φ{x}]x]↓› by "cqt:2"
  next
    AOT_have ‹□[λx φ{x}]↓›
      using 0 "exist-nec"[THEN "→E"] by blast
    moreover AOT_have ‹□[λx φ{x}]↓ → □∀x (¬[λx φ{x}]x ≡ ¬φ{x})›
      by(rule RM; safe intro!: GEN "≡I" "→I" "β→C"(2) "β←C"(2) "cqt:2")
    ultimately AOT_show ‹□∀x (¬[λx φ{x}]x ≡ ¬φ{x})›
      using "→E" by blast
  qed
qed

text‹Auxiliary lemma: conjunctions of denoting relations denote.›
AOT_theorem conjunction_denotes: ‹[λx φ{x}]↓ & [λx ψ{x}]↓ → [λx φ{x} & ψ{x}]↓›
proof(rule "→I")
  AOT_assume 0: ‹[λx φ{x}]↓ & [λx ψ{x}]↓›
  AOT_show ‹[λx φ{x} & ψ{x}]↓›
  proof (rule "safe-ext"[axiom_inst, THEN "→E", OF "&I"])
    AOT_show ‹[λx [λx φ{x}]x & [λx ψ{x}]x]↓› by "cqt:2"
  next
    AOT_have ‹□([λx φ{x}]↓ & [λx ψ{x}]↓)›
      using 0 "exist-nec"[THEN "→E"] "&E"
            "KBasic:3" "df-simplify:2" "intro-elim:3:b" by blast
    moreover AOT_have
      ‹□([λx φ{x}]↓ & [λx ψ{x}]↓) → □∀x ([λx φ{x}]x & [λx ψ{x}]x ≡ φ{x} & ψ{x})›
      by(rule RM; auto intro!: GEN "≡I" "→I" "cqt:2" "&I"
                       intro: "β←C"
                        dest: "&E" "β→C")
    ultimately AOT_show ‹□∀x ([λx φ{x}]x & [λx ψ{x}]x ≡ φ{x} & ψ{x})›
      using "→E" by blast
  qed
qed

AOT_register_rigid_restricted_type
  Ordinary: ‹O!κ›
proof
  AOT_modally_strict {
    AOT_show ‹∃x O!x›
      by (meson "B◇" "T◇" "o-objects-exist:1" "→E")
  }
next
  AOT_modally_strict {
    AOT_show ‹O!κ → κ↓› for κ
      by (simp add: "→I" "cqt:5:a[1]"[axiom_inst, THEN "→E", THEN "&E"(2)])
  }
next
  AOT_modally_strict {
```

```
    AOT_show ‹∀α(O!α → □O!α)›
      by (simp add: GEN "oa-facts:1")
  }
qed

AOT_register_variable_names
  Ordinary: u v r t s

text‹In PLM this is defined in the Natural Numbers chapter,
     but since it is helpful for stating the comprehension principles,
     we already define it here.›
AOT_define eqE :: ‹τ ⇒ τ ⇒ φ› (infixl ‹≡E› 50)
  eqE: ‹F ≡E G ≡df F↓ & G↓ & ∀u ([F]u ≡ [G]u)›                    (738)

text‹Derive existence claims about relations from the axioms.›
AOT_theorem denotes_all: ‹[λx ∀G (□G ≡E F → x[G])]↓›
    and denotes_all_neg: ‹[λx ∀G (□G ≡E F → ¬x[G])]↓›
proof -
  AOT_have Aux: ‹∀F (□F ≡E G → (x[F] ≡ x[G])), ¬(x[G] ≡ y[G])
    ⊢□ ∃F([F]x & ¬[F]y)› for x y G
  proof -
    AOT_modally_strict {
    AOT_assume 0: ‹∀F (□F ≡E G → (x[F] ≡ x[G]))›
    AOT_assume G_prop: ‹¬(x[G] ≡ y[G])›
    {
      AOT_assume ‹¬∃F([F]x & ¬[F]y)›
      AOT_hence 0: ‹∀F ¬([F]x & ¬[F]y)›
        by (metis "cqt-further:4" "→E")
      AOT_have ‹∀F ([F]x ≡ [F]y)›
      proof (rule GEN; rule "≡I"; rule "→I")
        fix F
        AOT_assume ‹[F]x›
        moreover AOT_have ‹¬([F]x & ¬[F]y)›
          using 0[THEN "∀E"(2)] by blast
        ultimately AOT_show ‹[F]y›
          by (metis "&I" "raa-cor:1")
      next
        fix F
        AOT_assume ‹[F]y›
        AOT_hence ‹¬[λx ¬[F]x]y›
          by (metis "¬¬I" "β→C"(2))
        moreover AOT_have ‹¬([λx ¬[F]x]x & ¬[λx ¬[F]x]y)›
          apply (rule 0[THEN "∀E"(1)]) by "cqt:2[lambda]"
        ultimately AOT_have 1: ‹¬[λx ¬[F]x]x›
          by (metis "&I" "raa-cor:3")
        {
          AOT_assume ‹¬[F]x›
          AOT_hence ‹[λx ¬[F]x]x›
            by (auto intro!: "β←C"(1) "cqt:2")
          AOT_hence ‹p & ¬p› for p
            using 1 by (metis "raa-cor:3")
        }
        AOT_thus ‹[F]x› by (metis "raa-cor:1")
      qed
      AOT_hence ‹□∀F ([F]x ≡ [F]y)›
        using "ind-nec"[THEN "→E"] by blast
      AOT_hence ‹∀F □([F]x ≡ [F]y)›
        by (metis "CBF" "→E")
    } note indistI = this
    {
      AOT_assume G_prop: ‹x[G] & ¬y[G]›
      AOT_hence Ax: ‹A!x›
        using "&E"(1) "∃I"(2) "→E" "encoders-are-abstract" by blast
```

```

      {
        AOT_assume Ay: ‹A!y›
        {
          fix F
          {
            AOT_assume ‹∀u□([F]u ≡ [G]u)›
            AOT_hence ‹□∀u([F]u ≡ [G]u)›
              using "Ordinary.res-var-bound-reas[BF]"[THEN "→E"] by simp
            AOT_hence ‹□F ≡E G›
              by (AOT_subst ‹F ≡E G› ‹∀u ([F]u ≡ [G]u)›)
                 (auto intro!: "eqE"[THEN "≡Df", THEN "≡S"(1), OF "&I"] "cqt:2")
            AOT_hence ‹x[F] ≡ x[G]›
              using 0[THEN "∀E"(2)] "≡E" "→E" by meson
            AOT_hence ‹x[F]›
              using G_prop "&E" "≡E" by blast
          }
          AOT_hence ‹∀u□([F]u ≡ [G]u) → x[F]›
            by (rule "→I")
        }
        AOT_hence xprop: ‹∀F(∀u□([F]u ≡ [G]u) → x[F])›
          by (rule GEN)
        moreover AOT_have yprop: ‹¬∀F(∀u□([F]u ≡ [G]u) → y[F])›
        proof (rule "raa-cor:2")
          AOT_assume ‹∀F(∀u□([F]u ≡ [G]u) → y[F])›
          AOT_hence ‹∀F(□∀u([F]u ≡ [G]u) → y[F])›
            apply (AOT_subst ‹□∀u([F]u ≡ [G]u)› ‹∀u□([F]u ≡ [G]u)› for: F)
            using "Ordinary.res-var-bound-reas[BF]"
                  "Ordinary.res-var-bound-reas[CBF]"
                  "intro-elim:2" apply presburger
            by simp
          AOT_hence A: ‹∀F(□F ≡E G → y[F])›
            by (AOT_subst ‹F ≡E G› ‹∀u ([F]u ≡ [G]u)› for: F)
               (auto intro!: "eqE"[THEN "≡Df", THEN "≡S"(1), OF "&I"] "cqt:2")
          moreover AOT_have ‹□G ≡E G›
            by (auto intro!: "eqE"[THEN "≡dfI"] "cqt:2" RN "&I" GEN "→I" "≡I")
          ultimately AOT_have ‹y[G]› using "∀E"(2) "→E" by blast
          AOT_thus ‹p & ¬p› for p using G_prop "&E" by (metis "raa-cor:3")
        qed
        AOT_have ‹∃F([F]x & ¬[F]y)›
        proof(rule "raa-cor:1")
          AOT_assume ‹¬∃F([F]x & ¬[F]y)›
          AOT_hence indist: ‹∀F □([F]x ≡ [F]y)› using indistI by blast
          AOT_have ‹∀F(∀u□([F]u ≡ [G]u) → y[F])›
            using indistinguishable_ord_enc_all[axiom_inst, THEN "→E", OF "&I",
                      OF "&I", OF "&I", OF "cqt:2[const_var]"[axiom_inst],
                      OF Ax, OF Ay, OF indist, THEN "≡E"(1), OF xprop].
          AOT_thus ‹∀F(∀u□([F]u ≡ [G]u) → y[F]) & ¬∀F(∀u□([F]u ≡ [G]u) → y[F])›
            using yprop "&I" by blast
        qed
      }
      moreover {
        AOT_assume notAy: ‹¬A!y›
        AOT_have ‹∃F([F]x & ¬[F]y)›
          apply (rule "∃I"(1)[where τ=‹«A!»›])
          using Ax notAy "&I" apply blast
          by (simp add: "oa-exist:2")
      }
      ultimately AOT_have ‹∃F([F]x & ¬[F]y)›
        by (metis "raa-cor:1")
    }
    moreover {
      AOT_assume G_prop: ‹¬x[G] & y[G]›
```

```
AOT_hence Ay: ‹A!y›
  by (meson "&E"(2) "encoders-are-abstract" "existential:2[const_var]" "→E")
AOT_hence notOy: ‹¬O!y›
  using "≡E"(1) "oa-contingent:3" by blast
{
  AOT_assume Ax: ‹A!x›
  {
    fix F
    {
      AOT_assume ‹□∀u([F]u ≡ [G]u)›
      AOT_hence ‹□F ≡E G›
        by (AOT_subst ‹F ≡E G› ‹∀u([F]u ≡ [G]u)›)
           (auto intro!: "eqE"[THEN "≡Df", THEN "≡S"(1), OF "&I"] "cqt:2")
      AOT_hence ‹x[F] ≡ x[G]›
        using 0[THEN "∀E"(2)] "≡E" "→E" by meson
      AOT_hence ‹¬x[F]›
        using G_prop "&E" "≡E" by blast
    }
    AOT_hence ‹□∀u([F]u ≡ [G]u) → ¬x[F]›
      by (rule "→I")
  }
  AOT_hence x_prop: ‹∀F(□∀u([F]u ≡ [G]u) → ¬x[F])›
    by (rule GEN)
  AOT_have x_prop: ‹¬∃F(∀u□([F]u ≡ [G]u) & x[F])›
  proof (rule "raa-cor:2")
    AOT_assume ‹∃F(∀u □([F]u ≡ [G]u) & x[F])›
    then AOT_obtain F where F_prop: ‹∀u □([F]u ≡ [G]u) & x[F]›
      using "∃E"[rotated] by blast
    AOT_have ‹□([F]u ≡ [G]u)› for u
      using F_prop[THEN "&E"(1), THEN "Ordinary.∀E"].
    AOT_hence ‹∀u □([F]u ≡ [G]u)›
      by (rule Ordinary.GEN)
    AOT_hence ‹□∀u([F]u ≡ [G]u)›
      by (metis "Ordinary.res-var-bound-reas[BF]" "→E")
    AOT_hence ‹¬x[F]›
      using x_prop[THEN "∀E"(2), THEN "→E"] by blast
    AOT_thus ‹p & ¬p› for p
      using F_prop[THEN "&E"(2)] by (metis "raa-cor:3")
  qed
  AOT_have y_prop: ‹∃F(∀u □([F]u ≡ [G]u) & y[F])›
  proof (rule "raa-cor:1")
    AOT_assume ‹¬∃F (∀u □([F]u ≡ [G]u) & y[F])›
    AOT_hence 0: ‹∀F ¬(∀u □([F]u ≡ [G]u) & y[F])›
      using "cqt-further:4"[THEN "→E"] by blast
    {
      fix F
      {
        AOT_assume ‹∀u □([F]u ≡ [G]u)›
        AOT_hence ‹¬y[F]›
          using 0[THEN "∀E"(2)] "&I" "raa-cor:1" by meson
      }
      AOT_hence ‹(∀u □([F]u ≡ [G]u) → ¬y[F])›
        by (rule "→I")
    }
    AOT_hence A: ‹∀F(∀u □([F]u ≡ [G]u) → ¬y[F])›
      by (rule GEN)
    moreover AOT_have ‹∀u □([G]u ≡ [G]u)›
      by (simp add: RN "oth-class-taut:3:a" "universal-cor" "→I")
    ultimately AOT_have ‹¬y[G]›
      using "∀E"(2) "→E" by blast
    AOT_thus ‹p & ¬p› for p
      using G_prop "&E" by (metis "raa-cor:3")
  qed
```

```
     AOT_have ‹∃F([F]x & ¬[F]y)›
     proof(rule "raa-cor:1")
      AOT_assume ‹¬∃F([F]x & ¬[F]y)›
      AOT_hence indist: ‹∀F □([F]x ≡ [F]y)›
       using indistI by blast
      AOT_thus ‹∃F(∀u □([F]u ≡ [G]u) & x[F]) & ¬∃F(∀u □([F]u ≡ [G]u) & x[F])›
       using indistinguishable_ord_enc_ex[axiom_inst, THEN "→E", OF "&I",
           OF "&I", OF "&I", OF "cqt:2[const_var]"[axiom_inst],
           OF Ax, OF Ay, OF indist, THEN "≡E"(2), OF y_prop]
         x_prop "&I" by blast
     qed
    }
    moreover {
     AOT_assume notAx: ‹¬A!x›
     AOT_hence Ox: ‹O!x›
      using "∨E"(3) "oa-exist:3" by blast
     AOT_have ‹∃F([F]x & ¬[F]y)›
      apply (rule "∃I"(1)[where τ=‹«O!»›])
      using Ox notOy "&I" apply blast
      by (simp add: "oa-exist:1")
    }
    ultimately AOT_have ‹∃F([F]x & ¬[F]y)›
     by (metis "raa-cor:1")
   }
   ultimately AOT_show ‹∃F([F]x & ¬[F]y)›
    using G_prop by (metis "&I" "→I" "≡I" "raa-cor:1")
  }
  qed

  AOT_modally_strict {
   fix x y
   AOT_assume indist: ‹∀F ([F]x ≡ [F]y)›
   AOT_hence nec_indist: ‹□∀F ([F]x ≡ [F]y)›
    using "ind-nec" "vdash-properties:10" by blast
   AOT_hence indist_nec: ‹∀F □([F]x ≡ [F]y)›
    using "CBF" "vdash-properties:10" by blast
   AOT_assume 0: ‹∀G (□G ≡E F → x[G])›
   AOT_hence 1: ‹∀G (□∀u ([G]u ≡ [F]u) → x[G])›
    by (AOT_subst (reverse) ‹∀u ([G]u ≡ [F]u)› ‹G ≡E F› for: G)
      (auto intro!: "eqE"[THEN "≡Df", THEN "≡S"(1), OF "&I"] "cqt:2")
   AOT_have ‹x[F]›
    by (safe intro!: 1[THEN "∀E"(2), THEN "→E"] GEN "→I" RN "≡I")
   AOT_have ‹∀G (□G ≡E F → y[G])›
   proof(rule "raa-cor:1")
    AOT_assume ‹¬∀G (□G ≡E F → y[G])›
    AOT_hence ‹∃G ¬(□G ≡E F → y[G])›
     using "cqt-further:2" "→E" by blast
    then AOT_obtain G where G_prop: ‹¬(□G ≡E F → y[G])›
     using "∃E"[rotated] by blast
    AOT_hence 1: ‹□G ≡E F & ¬y[G]›
     by (metis "≡E"(1) "oth-class-taut:1:b")
    AOT_have xG: ‹x[G]›
     using 0[THEN "∀E"(2), THEN "→E"] 1[THEN "&E"(1)] by blast
    AOT_hence ‹x[G] & ¬y[G]›
     using 1[THEN "&E"(2)] "&I" by blast
    AOT_hence B: ‹¬(x[G] ≡ y[G])›
     using "&E"(2) "≡E"(1) "reductio-aa:1" xG by blast
    {
     fix H
     {
      AOT_assume ‹□H ≡E G›
      AOT_hence ‹□(H ≡E G & G ≡E F)›
       using 1 by (metis "KBasic:3" "con-dis-i-e:1" "con-dis-i-e:2:a"
```

```
                            "intro-elim:3:b")
        moreover AOT_have ‹□(H ≡E G & G ≡E F) → □(H ≡E F)›
        proof(rule RM)
          AOT_modally_strict {
            AOT_show ‹H ≡E G & G ≡E F → H ≡E F›
            proof (safe intro!: "→I" "eqE"[THEN "≡dfI"] "&I" "cqt:2" Ordinary.GEN)
              fix u
              AOT_assume ‹H ≡E G & G ≡E F›
              AOT_hence ‹∀u ([H]u ≡ [G]u)› and ‹∀u ([G]u ≡ [F]u)›
                using "eqE"[THEN "≡dfE"] "&E" by blast+
              AOT_thus ‹[H]u ≡ [F]u›
                by (auto dest!: "Ordinary.∀E" dest: "≡E")
            qed
          }
        qed
        ultimately AOT_have ‹□(H ≡E F)›
          using "→E" by blast
        AOT_hence ‹x[H]›
          using 0[THEN "∀E"(2)] "→E" by blast
        AOT_hence ‹x[H] ≡ x[G]›
          using xG "≡I" "→I" by blast
      }
      AOT_hence ‹□H ≡E G → (x[H] ≡ x[G])› by (rule "→I")
    }
    AOT_hence A: ‹∀H(□H ≡E G → (x[H] ≡ x[G]))›
      by (rule GEN)
    then AOT_obtain F where F_prop: ‹[F]x & ¬[F]y›
      using Aux[OF A, OF B] "∃E"[rotated] by blast
    moreover AOT_have ‹[F]y›
      using indist[THEN "∀E"(2), THEN "≡E"(1), OF F_prop[THEN "&E"(1)]].
    AOT_thus ‹p & ¬p› for p
      using F_prop[THEN "&E"(2)] by (metis "raa-cor:3")
  qed
} note 0 = this
AOT_modally_strict {
  fix x y
  AOT_assume ‹∀F ([F]x ≡ [F]y)›
  moreover AOT_have ‹∀F ([F]y ≡ [F]x)›
    by (metis calculation "cqt-basic:11" "≡E"(2))
  ultimately AOT_have ‹∀G (□G ≡E F → x[G]) ≡ ∀G (□G ≡E F → y[G])›
    using 0 "≡I" "→I" by auto
} note 1 = this
AOT_show ‹[λx ∀G (□G ≡E F → x[G])]↓›
  by (safe intro!: RN GEN "→I" 1 "kirchner-thm:2"[THEN "≡E"(2)])

AOT_modally_strict {
  fix x y
  AOT_assume indist: ‹∀F ([F]x ≡ [F]y)›
  AOT_hence nec_indist: ‹□∀F ([F]x ≡ [F]y)›
    using "ind-nec" "vdash-properties:10" by blast
  AOT_hence indist_nec: ‹∀F □([F]x ≡ [F]y)›
    using "CBF" "vdash-properties:10" by blast
  AOT_assume 0: ‹∀G (□G ≡E F → ¬x[G])›
  AOT_hence 1: ‹∀G (□∀u ([G]u ≡ [F]u) → ¬x[G])›
    by (AOT_subst (reverse) ‹∀u ([G]u ≡ [F]u)› ‹G ≡E F› for: G)
       (auto intro!: "eqE"[THEN "≡Df", THEN "≡S"(1), OF "&I"] "cqt:2")
  AOT_have ‹¬x[F]›
    by (safe intro!: 1[THEN "∀E"(2), THEN "→E"] GEN "→I" RN "≡I")
  AOT_have ‹∀G (□G ≡E F → ¬y[G])›
  proof(rule "raa-cor:1")
    AOT_assume ‹¬∀G (□G ≡E F → ¬y[G])›
    AOT_hence ‹∃G ¬(□G ≡E F → ¬y[G])›
      using "cqt-further:2" "→E" by blast
```

```
      then AOT_obtain G where G_prop: ‹¬(□G ≡E F → ¬y[G])›
        using "∃E"[rotated] by blast
      AOT_hence 1: ‹□G ≡E F & ¬¬y[G]›
        by (metis "≡E"(1) "oth-class-taut:1:b")
      AOT_hence yG: ‹y[G]›
        using G_prop "→I" "raa-cor:3" by blast
      moreover AOT_hence 12: ‹¬x[G]›
        using 0[THEN "∀E"(2), THEN "→E"] 1[THEN "&E"(1)] by blast
      ultimately AOT_have ‹¬x[G] & y[G]›
        using "&I" by blast
      AOT_hence B: ‹¬(x[G] ≡ y[G])›
        by (metis "12" "≡E"(3) "raa-cor:3" yG)
      {
        fix H
        {
          AOT_assume 3: ‹□H ≡E G›
          AOT_hence ‹□(H ≡E G & G ≡E F)›
            using 1
            by (metis "KBasic:3" "con-dis-i-e:1" "→I" "intro-elim:3:b"
                      "reductio-aa:1" G_prop)
          moreover AOT_have ‹□(H ≡E G & G ≡E F) → □(H ≡E F)›
          proof (rule RM)
            AOT_modally_strict {
              AOT_show ‹H ≡E G & G ≡E F → H ≡E F›
              proof (safe intro!: "→I" "eqE"[THEN "≡dfI"] "&I" "cqt:2" Ordinary.GEN)
                fix u
                AOT_assume ‹H ≡E G & G ≡E F›
                AOT_hence ‹∀u ([H]u ≡ [G]u)› and ‹∀u ([G]u ≡ [F]u)›
                  using "eqE"[THEN "≡dfE"] "&E" by blast+
                AOT_thus ‹[H]u ≡ [F]u›
                  by (auto dest!: "Ordinary.∀E" dest: "≡E")
              qed
            }
          qed
          ultimately AOT_have ‹□(H ≡E F)›
            using "→E" by blast
          AOT_hence ‹¬x[H]›
            using 0[THEN "∀E"(2)] "→E" by blast
          AOT_hence ‹x[H] ≡ x[G]›
            using 12 "≡I" "→I" by (metis "raa-cor:3")
        }
        AOT_hence ‹□H ≡E G → (x[H] ≡ x[G])›
          by (rule "→I")
      }
      AOT_hence A: ‹∀H(□H ≡E G → (x[H] ≡ x[G]))›
        by (rule GEN)
      then AOT_obtain F where F_prop: ‹[F]x & ¬[F]y›
        using Aux[OF A, OF B] "∃E"[rotated] by blast
      moreover AOT_have ‹[F]y›
        using indist[THEN "∀E"(2), THEN "≡E"(1), OF F_prop[THEN "&E"(1)]].
      AOT_thus ‹p & ¬p› for p
        using F_prop[THEN "&E"(2)] by (metis "raa-cor:3")
    qed
  } note 0 = this
  AOT_modally_strict {
    fix x y
    AOT_assume ‹∀F ([F]x ≡ [F]y)›
    moreover AOT_have ‹∀F ([F]y ≡ [F]x)›
      by (metis calculation "cqt-basic:11" "≡E"(2))
    ultimately AOT_have ‹∀G (□G ≡E F → ¬x[G]) ≡ ∀G (□G ≡E F → ¬y[G])›
      using 0 "≡I" "→I" by auto
  } note 1 = this
  AOT_show ‹[λx ∀G (□G ≡E F → ¬x[G])]↓›
```

```
    by (safe intro!: RN GEN "→I" 1 "kirchner-thm:2"[THEN "≡E"(2)])
qed

text‹Reformulate the existence claims in terms of their negations.›

AOT_theorem denotes_ex: ‹[λx ∃G (□G ≡E F & x[G])]↓›
proof (rule "safe-ext"[axiom_inst, THEN "→E", OF "&I"])
  AOT_show ‹[λx ¬∀G (□G ≡E F → ¬x[G])]↓›
    using denotes_all_neg[THEN negation_denotes[THEN "→E"]].
next
  AOT_show ‹□∀x (¬∀G (□G ≡E F → ¬x[G]) ≡ ∃G (□G ≡E F & x[G]))›
    by (AOT_subst  ‹□G ≡E F & x[G]› ‹¬(□G ≡E F → ¬x[G])› for: G x)
       (auto simp: "conventions:1" "rule-eq-df:1"
             intro: "oth-class-taut:4:b"[THEN "≡E"(2)]
                    "intro-elim:3:f"[OF "cqt-further:3", OF "oth-class-taut:3:b"]
             intro!: RN GEN)
qed

AOT_theorem denotes_ex_neg: ‹[λx ∃G (□G ≡E F & ¬x[G])]↓›
proof (rule "safe-ext"[axiom_inst, THEN "→E", OF "&I"])
  AOT_show ‹[λx ¬∀G (□G ≡E F → x[G])]↓›
    using denotes_all[THEN negation_denotes[THEN "→E"]].
next
  AOT_show ‹□∀x (¬∀G (□G ≡E F → x[G]) ≡ ∃G (□G ≡E F & ¬x[G]))›
    by (AOT_subst (reverse) ‹□G ≡E F & ¬x[G]› ‹¬(□G ≡E F → x[G])› for: G x)
       (auto simp: "oth-class-taut:1:b"
             intro: "oth-class-taut:4:b"[THEN "≡E"(2)]
                    "intro-elim:3:f"[OF "cqt-further:3", OF "oth-class-taut:3:b"]
             intro!: RN GEN)
qed

text‹Derive comprehension principles.›

AOT_theorem Comprehension_1:
  shows ‹□∀F∀G(□G ≡E F → (φ{F} ≡ φ{G})) → [λx ∃F (φ{F} & x[F])]↓›
proof(rule "→I")
  AOT_assume assm: ‹□∀F∀G(□G ≡E F → (φ{F} ≡ φ{G}))›
  AOT_modally_strict {
    fix x y
    AOT_assume 0: ‹∀F∀G (□G ≡E F → (φ{F} ≡ φ{G}))›
    AOT_assume indist: ‹∀F ([F]x ≡ [F]y)›
    AOT_assume x_prop: ‹∃F (φ{F} & x[F])›
    then AOT_obtain F where F_prop: ‹φ{F} & x[F]›
      using "∃E"[rotated] by blast
    AOT_hence ‹□F ≡E F & x[F]›
      by (auto intro!: RN eqE[THEN "≡dfI"] "&I" "cqt:2" GEN "≡I" "→I" dest: "&E")
    AOT_hence ‹∃G(□G ≡E F & x[G])›
      by (rule "∃I")
    AOT_hence ‹[λx ∃G(□G ≡E F & x[G])]x›
      by (safe intro!: "β←C" denotes_ex "cqt:2")
    AOT_hence ‹[λx ∃G(□G ≡E F & x[G])]y›
      using indist[THEN "∀E"(1), OF denotes_ex, THEN "≡E"(1)] by blast
    AOT_hence ‹∃G(□G ≡E F & y[G])›
      using "β→C" by blast
    then AOT_obtain G where ‹□G ≡E F & y[G]›
      using "∃E"[rotated] by blast
    AOT_hence ‹φ{G} & y[G]›
      using 0[THEN "∀E"(2), THEN "∀E"(2), THEN "→E", THEN "≡E"(1)]
            F_prop[THEN "&E"(1)] "&E" "&I" by blast
    AOT_hence ‹∃F (φ{F} & y[F])›
      by (rule "∃I")
  } note 1 = this
  AOT_modally_strict {
```

```
    AOT_assume 0: ‹∀F∀G (□G ≡E F → (φ{F} ≡ φ{G}))›
    {
      fix x y
      {
        AOT_assume ‹∀F ([F]x ≡ [F]y)›
        moreover AOT_have ‹∀F ([F]y ≡ [F]x)›
          by (metis calculation "cqt-basic:11" "≡E"(1))
        ultimately AOT_have ‹∃F (φ{F} & x[F]) ≡ ∃F (φ{F} & y[F])›
          using 0 1[OF 0] "≡I" "→I" by simp
      }
      AOT_hence ‹∀F ([F]x ≡ [F]y) → (∃F (φ{F} & x[F]) ≡ ∃F (φ{F} & y[F]))›
        using "→I" by blast
    }
    AOT_hence ‹∀x∀y(∀F ([F]x ≡ [F]y) → (∃F (φ{F} & x[F]) ≡  ∃F (φ{F} & y[F])))›
      by (auto intro!: GEN)
  } note 1 = this
  AOT_hence ‹⊢□ ∀F∀G (□G ≡E F → (φ{F} ≡ φ{G})) →
                ∀x∀y(∀F ([F]x ≡ [F]y) → (∃F (φ{F} & x[F]) ≡ ∃F (φ{F} & y[F])))›
    by (rule "→I")
  AOT_hence ‹□∀F∀G (□G ≡E F → (φ{F} ≡ φ{G})) →
              □∀x∀y(∀F ([F]x ≡ [F]y) → (∃F (φ{F} & x[F]) ≡ ∃F (φ{F} & y[F])))›
    by (rule RM)
  AOT_hence ‹□∀x∀y(∀F ([F]x ≡ [F]y) → (∃F (φ{F} & x[F]) ≡ ∃F (φ{F} & y[F])))›
    using "→E" assm by blast
  AOT_thus ‹[λx ∃F (φ{F} & x[F])]↓›
    by (safe intro!: "kirchner-thm:2"[THEN "≡E"(2)])
qed

AOT_theorem Comprehension_2:
  shows ‹□∀F∀G(□G ≡E F → (φ{F} ≡ φ{G})) → [λx ∃F (φ{F} & ¬x[F])]↓›
proof(rule "→I")
  AOT_assume assm: ‹□∀F∀G(□G ≡E F → (φ{F} ≡ φ{G}))›
  AOT_modally_strict {
    fix x y
    AOT_assume 0: ‹∀F∀G (□G ≡E F → (φ{F} ≡ φ{G}))›
    AOT_assume indist: ‹∀F ([F]x ≡ [F]y)›
    AOT_assume x_prop: ‹∃F (φ{F} & ¬x[F])›
    then AOT_obtain F where F_prop: ‹φ{F} & ¬x[F]›
      using "∃E"[rotated] by blast
    AOT_hence ‹□F ≡E F & ¬x[F]›
      by (auto intro!: RN eqE[THEN "≡dfI"] "&I" "cqt:2" GEN "≡I" "→I" dest: "&E")
    AOT_hence ‹∃G(□G ≡E F & ¬x[G])›
      by (rule "∃I")
    AOT_hence ‹[λx ∃G(□G ≡E F & ¬x[G])]x›
      by (safe intro!: "β←C" denotes_ex_neg "cqt:2")
    AOT_hence ‹[λx ∃G(□G ≡E F & ¬x[G])]y›
      using indist[THEN "∀E"(1), OF denotes_ex_neg, THEN "≡E"(1)] by blast
    AOT_hence ‹∃G(□G ≡E F & ¬y[G])›
      using "β→C" by blast
    then AOT_obtain G where ‹□G ≡E F & ¬y[G]›
      using "∃E"[rotated] by blast
    AOT_hence ‹φ{G} & ¬y[G]›
      using 0[THEN "∀E"(2), THEN "∀E"(2), THEN "→E", THEN "≡E"(1)]
        F_prop[THEN "&E"(1)] "&E" "&I" by blast
    AOT_hence ‹∃F (φ{F} & ¬y[F])›
      by (rule "∃I")
  } note 1 = this
  AOT_modally_strict {
    AOT_assume 0: ‹∀F∀G (□G ≡E F → (φ{F} ≡ φ{G}))›
    {
      fix x y
      {
        AOT_assume ‹∀F ([F]x ≡ [F]y)›
```

```
        moreover AOT_have ‹∀F ([F]y ≡ [F]x)›
          by (metis calculation "cqt-basic:11" "≡E"(1))
        ultimately AOT_have ‹∃F (φ{F} & ¬x[F]) ≡ ∃F (φ{F} & ¬y[F])›
          using 0 1[OF 0] "≡I" "→I" by simp
      }
      AOT_hence ‹∀F ([F]x ≡ [F]y) → (∃F (φ{F} & ¬x[F]) ≡ ∃F (φ{F} & ¬y[F]))›
        using "→I" by blast
    }
    AOT_hence ‹∀x∀y(∀F ([F]x ≡ [F]y) → (∃F (φ{F} & ¬x[F]) ≡  ∃F (φ{F} & ¬y[F])))›
      by (auto intro!: GEN)
  } note 1 = this
  AOT_hence ‹⊢□ ∀F∀G (□G ≡E F → (φ{F} ≡ φ{G})) →
                  ∀x∀y(∀F ([F]x ≡ [F]y) → (∃F (φ{F} & ¬x[F]) ≡ ∃F (φ{F} & ¬y[F])))›
    by (rule "→I")
  AOT_hence ‹□∀F∀G (□G ≡E F → (φ{F} ≡ φ{G})) →
                □∀x∀y(∀F ([F]x ≡ [F]y) → (∃F (φ{F} & ¬x[F]) ≡ ∃F (φ{F} & ¬y[F])))›
    by (rule RM)
  AOT_hence ‹□∀x∀y(∀F ([F]x ≡ [F]y) → (∃F (φ{F} & ¬x[F]) ≡ ∃F (φ{F} & ¬y[F])))›
    using "→E" assm by blast
  AOT_thus ‹[λx ∃F (φ{F} & ¬x[F])]↓›
    by (safe intro!: "kirchner-thm:2"[THEN "≡E"(2)])
qed

text‹Derived variants of the comprehension principles above.›

AOT_theorem Comprehension_1':
  shows ‹□∀F∀G(□G ≡E F → (φ{F} ≡ φ{G})) → [λx ∀F (x[F] → φ{F})]↓›
proof(rule "→I")
  AOT_assume ‹□∀F∀G(□G ≡E F → (φ{F} ≡ φ{G}))›
  AOT_hence 0: ‹□∀F∀G(□G ≡E F → (¬φ{F} ≡ ¬φ{G}))›
    by (AOT_subst (reverse) ‹¬φ{F} ≡ ¬φ{G}› ‹φ{F} ≡ φ{G}› for: F G)
       (auto simp: "oth-class-taut:4:b")
  AOT_show ‹[λx ∀F (x[F] → φ{F})]↓›
  proof(rule "safe-ext"[axiom_inst, THEN "→E", OF "&I"])
    AOT_show ‹[λx ¬∃F(¬φ{F} & x[F])]↓›
      using Comprehension_1[THEN "→E", OF 0, THEN negation_denotes[THEN "→E"]].
  next
    AOT_show ‹□∀x (¬∃F (¬φ{F} & x[F]) ≡ ∀F (x[F] → φ{F}))›
      by (AOT_subst (reverse) ‹¬φ{F} & x[F]› ‹¬(x[F] → φ{F})› for: F x)
         (auto simp: "oth-class-taut:1:b"[THEN "intro-elim:3:e",
                                          OF "oth-class-taut:2:a"]
               intro: "intro-elim:3:f"[OF "cqt-further:3", OF "oth-class-taut:3:a",
                                       symmetric]
               intro!: RN GEN)
  qed
qed

AOT_theorem Comprehension_2':
  shows ‹□∀F∀G(□G ≡E F → (φ{F} ≡ φ{G})) → [λx ∀F (φ{F} → x[F])]↓›
proof(rule "→I")
  AOT_assume 0: ‹□∀F∀G(□G ≡E F → (φ{F} ≡ φ{G}))›
  AOT_show ‹[λx ∀F (φ{F} → x[F])]↓›
  proof(rule "safe-ext"[axiom_inst, THEN "→E", OF "&I"])
    AOT_show ‹[λx ¬∃F(φ{F} & ¬x[F])]↓›
      using Comprehension_2[THEN "→E", OF 0, THEN negation_denotes[THEN "→E"]].
  next
    AOT_show ‹□∀x (¬∃F (φ{F} & ¬x[F]) ≡ ∀F (φ{F} → x[F]))›
      by (AOT_subst (reverse) ‹φ{F} & ¬x[F]› ‹¬(φ{F} → x[F])› for: F x)
         (auto simp: "oth-class-taut:1:b"
               intro: "intro-elim:3:f"[OF "cqt-further:3", OF "oth-class-taut:3:a",
                                       symmetric]
               intro!: RN GEN)
  qed
```

```
qed

text‹Derive a combined comprehension principles.›

AOT_theorem Comprehension_3:
  ‹□∀F∀G(□G ≡E F → (φ{F} ≡ φ{G})) → [λx ∀F (x[F] ≡ φ{F})]↓›
proof(rule "→I")
  AOT_assume 0: ‹□∀F∀G(□G ≡E F → (φ{F} ≡ φ{G}))›
  AOT_show ‹[λx ∀F (x[F] ≡ φ{F})]↓›
  proof(rule "safe-ext"[axiom_inst, THEN "→E", OF "&I"])
    AOT_show ‹[λx ∀F (x[F] → φ{F}) & ∀F (φ{F} → x[F])]↓›
      by (safe intro!: conjunction_denotes[THEN "→E", OF "&I"]
                      Comprehension_1'[THEN "→E"]
                      Comprehension_2'[THEN "→E"] 0)
  next
    AOT_show ‹□∀x (∀F (x[F] → φ{F}) & ∀F (φ{F} → x[F]) ≡ ∀F (x[F] ≡ φ{F}))›
      by (auto intro!: RN GEN "≡I" "→I" "&I" dest: "&E" "∀E"(2) "→E" "≡E"(1,2))
  qed
qed

notepad
begin
text‹Verify that the original axioms are equivalent to @{thm denotes_ex}
     and @{thm denotes_ex_neg}.›
AOT_modally_strict {
  fix x y H
  AOT_have ‹A!x & A!y & ∀F □([F]x ≡ [F]y) →
  (∀G (∀z (O!z → □([G]z ≡ [H]z)) → x[G]) ≡
   ∀G (∀z (O!z → □([G]z ≡ [H]z)) → y[G]))›
  proof(rule "→I")
    {
      fix x y
      AOT_assume ‹A!x›
      AOT_assume ‹A!y›
      AOT_assume indist: ‹∀F □([F]x ≡ [F]y)›
      AOT_assume ‹∀G (∀u □([G]u ≡ [H]u) → x[G])›
      AOT_hence ‹∀G (□∀u ([G]u ≡ [H]u) → x[G])›
        using "Ordinary.res-var-bound-reas[BF]" "Ordinary.res-var-bound-reas[CBF]"
              "intro-elim:2"
        by (AOT_subst ‹□∀u ([G]u ≡ [H]u)› ‹∀u □([G]u ≡ [H]u)› for: G) auto
      AOT_hence ‹∀G (□G ≡E H → x[G])›
        by (AOT_subst ‹G ≡E H› ‹∀u ([G]u ≡ [H]u)› for: G)
           (safe intro!: "eqE"[THEN "≡Df", THEN "≡S"(1), OF "&I"] "cqt:2")
      AOT_hence ‹¬∃G (□G ≡E H & ¬x[G])›
        by (AOT_subst (reverse) ‹(□G ≡E H & ¬x[G])› ‹¬(□G ≡E H → x[G])› for: G)
          (auto simp: "oth-class-taut:1:b" "cqt-further:3"[THEN "≡E"(1)])
      AOT_hence ‹¬[λx ∃G (□G ≡E H & ¬x[G])]x›
        by (auto intro: "β→C")
      AOT_hence ‹¬[λx ∃G (□G ≡E H & ¬x[G])]y›
        using indist[THEN "∀E"(1), OF denotes_ex_neg,
                     THEN "qml:2"[axiom_inst, THEN "→E"],
                     THEN "≡E"(3)] by blast
      AOT_hence ‹¬∃G (□G ≡E H & ¬y[G])›
        by (safe intro!: "β←C" denotes_ex_neg "cqt:2")
      AOT_hence ‹∀G ¬(□G ≡E H & ¬y[G])›
        using "cqt-further:4"[THEN "→E"] by blast
      AOT_hence ‹∀G(□G ≡E H → y[G])›
        by (AOT_subst ‹□G ≡E H → y[G]› ‹¬(□G ≡E H & ¬y[G])› for: G)
           (auto simp: "oth-class-taut:1:a")
      AOT_hence ‹∀G (□∀u([G]u ≡ [H]u) → y[G])›
        by (AOT_subst (reverse) ‹∀u ([G]u ≡ [H]u)› ‹G ≡E H› for: G)
           (safe intro!: "eqE"[THEN "≡Df", THEN "≡S"(1), OF "&I"] "cqt:2")
      AOT_hence ‹∀G (∀u □([G]u ≡ [H]u) → y[G])›
```

```
        using "Ordinary.res-var-bound-reas[BF]" "Ordinary.res-var-bound-reas[CBF]"
              "intro-elim:2"
        by (AOT_subst ‹∀u □([G]u ≡ [H]u)› ‹□∀u ([G]u ≡ [H]u)› for: G) auto
    } note 0 = this
    AOT_assume ‹A!x & A!y & ∀F □([F]x ≡ [F]y)›
    AOT_hence ‹A!x› and ‹A!y› and ‹∀F □([F]x ≡ [F]y)›
      using "&E" by blast+
    moreover AOT_have ‹∀F □([F]y ≡ [F]x)›
      using calculation(3)
      apply (safe intro!: CBF[THEN "→E"] dest!: BF[THEN "→E"])
      using "RM:3" "cqt-basic:11" "intro-elim:3:b" by fast
    ultimately AOT_show ‹∀G (∀u □([G]u ≡ [H]u) → x[G]) ≡
                         ∀G (∀u □([G]u ≡ [H]u) → y[G])›
      using 0 by (auto intro!: "≡I" "→I")
  qed

  AOT_have ‹A!x & A!y & ∀F □([F]x ≡ [F]y) →
  (∃G (∀z (O!z → □([G]z ≡ [H]z)) & x[G]) ≡ ∃G (∀z (O!z → □([G]z ≡ [H]z)) & y[G]))›
  proof(rule "→I")
    {
      fix x y
      AOT_assume ‹A!x›
      AOT_assume ‹A!y›
      AOT_assume indist: ‹∀F □([F]x ≡ [F]y)›
      AOT_assume x_prop: ‹∃G (∀u □([G]u ≡ [H]u) & x[G])›
      AOT_hence ‹∃G (□∀u ([G]u ≡ [H]u) & x[G])›
        using "Ordinary.res-var-bound-reas[BF]" "Ordinary.res-var-bound-reas[CBF]"
              "intro-elim:2"
        by (AOT_subst ‹□∀u ([G]u ≡ [H]u)› ‹∀u □([G]u ≡ [H]u)› for: G) auto
      AOT_hence ‹∃G (□G ≡E H & x[G])›
        by (AOT_subst ‹G ≡E H› ‹∀u ([G]u ≡ [H]u)› for: G)
           (safe intro!: "eqE"[THEN "≡Df", THEN "≡S"(1), OF "&I"] "cqt:2")
      AOT_hence ‹[λx ∃G (□G ≡E H & x[G])]x›
        by (safe intro!: "β←C" denotes_ex "cqt:2")
      AOT_hence ‹[λx ∃G (□G ≡E H & x[G])]y›
        using indist[THEN "∀E"(1), OF denotes_ex,
                     THEN "qml:2"[axiom_inst, THEN "→E"],
                     THEN "≡E"(1)] by blast
      AOT_hence ‹∃G (□G ≡E H & y[G])›
        by (rule "β→C")
      AOT_hence ‹∃G (□∀u ([G]u ≡ [H]u) & y[G])›
        by (AOT_subst (reverse) ‹∀u ([G]u ≡ [H]u)› ‹G ≡E H› for: G)
           (safe intro!: "eqE"[THEN "≡Df", THEN "≡S"(1), OF "&I"] "cqt:2")
      AOT_hence ‹∃G (∀u □([G]u ≡ [H]u) & y[G])›
        using "Ordinary.res-var-bound-reas[BF]"
              "Ordinary.res-var-bound-reas[CBF]"
              "intro-elim:2"
        by (AOT_subst ‹∀u □([G]u ≡ [H]u)› ‹□∀u ([G]u ≡ [H]u)› for: G) auto
    } note 0 = this
    AOT_assume ‹A!x & A!y & ∀F □([F]x ≡ [F]y)›
    AOT_hence ‹A!x› and ‹A!y› and ‹∀F □([F]x ≡ [F]y)›
      using "&E" by blast+
    moreover AOT_have ‹∀F □([F]y ≡ [F]x)›
      using calculation(3)
      apply (safe intro!: CBF[THEN "→E"] dest!: BF[THEN "→E"])
      using "RM:3" "cqt-basic:11" "intro-elim:3:b" by fast
    ultimately AOT_show ‹∃G (∀u □([G]u ≡ [H]u) & x[G]) ≡
                         ∃G (∀u □([G]u ≡ [H]u) & y[G])›
      using 0 by (auto intro!: "≡I" "→I")
  qed
}
end
end
```

A.11. Possible Worlds

```
(*<*)
theory AOT_PossibleWorlds
  imports AOT_PLM AOT_BasicLogicalObjects AOT_RestrictedVariables
begin
(*>*)

section‹Possible Worlds›

AOT_define Situation :: ‹τ ⇒ φ› (‹Situation'(_')›)
  situations: ‹Situation(x) ≡df A!x & ∀F (x[F] → Propositional([F]))›        (456)

AOT_theorem "T-sit": ‹TruthValue(x) → Situation(x)›        (457)
proof(rule "→I")
  AOT_assume ‹TruthValue(x)›
  AOT_hence ‹∃p TruthValueOf(x,p)›
    using "T-value"[THEN "≡dfE"] by blast
  then AOT_obtain p where ‹TruthValueOf(x,p)› using "∃E"[rotated] by blast
  AOT_hence ϑ: ‹A!x & ∀F (x[F] ≡ ∃q((q ≡ p) & F = [λy q]))›
    using "tv-p"[THEN "≡dfE"] by blast
  AOT_show ‹Situation(x)›
  proof(rule situations[THEN "≡dfI"]; safe intro!: "&I" GEN "→I" ϑ[THEN "&E"(1)])
    fix F
    AOT_assume ‹x[F]›
    AOT_hence ‹∃q((q ≡ p) & F = [λy q])›
      using ϑ[THEN "&E"(2), THEN "∀E"(2)[where β=F], THEN "≡E"(1)] by argo
    then AOT_obtain q where ‹(q ≡ p) & F = [λy q]› using "∃E"[rotated] by blast
    AOT_hence ‹∃p F = [λy p]› using "&E"(2) "∃I"(2) by metis
    AOT_thus ‹Propositional([F])›
      by (metis "≡dfI" "prop-prop1")
  qed
qed

AOT_theorem "possit-sit:1": ‹Situation(x) ≡ □Situation(x)›        (458.1)
proof(rule "≡I"; rule "→I")
  AOT_assume ‹Situation(x)›
  AOT_hence 0: ‹A!x & ∀F (x[F] → Propositional([F]))›
    using situations[THEN "≡dfE"] by blast
  AOT_have 1: ‹□(A!x & ∀F (x[F] → Propositional([F])))›
  proof(rule "KBasic:3"[THEN "≡E"(2)]; rule "&I")
    AOT_show ‹□A!x› using 0[THEN "&E"(1)] by (metis "oa-facts:2"[THEN "→E"])
  next
    AOT_have ‹∀F (x[F] → Propositional([F])) → □∀F (x[F] → Propositional([F]))›
      by (AOT_subst ‹Propositional([F])› ‹∃p (F = [λy p])› for: F :: ‹<κ>›)
         (auto simp: "prop-prop1" "≡Df" "enc-prop-nec:2")
    AOT_thus ‹□∀F (x[F] → Propositional([F]))›
      using 0[THEN "&E"(2)] "→E" by blast
  qed
  AOT_show ‹□Situation(x)›
    by (AOT_subst ‹Situation(x)› ‹A!x & ∀F (x[F] → Propositional([F]))›)
       (auto simp: 1 "≡Df" situations)
next
  AOT_show ‹Situation(x)› if ‹□Situation(x)›
    using "qml:2"[axiom_inst, THEN "→E", OF that].
qed

AOT_theorem "possit-sit:2": ‹◇Situation(x) ≡ Situation(x)›        (458.2)
  using "possit-sit:1"
  by (metis "RE◇" "S5Basic:2" "≡E"(1) "≡E"(5) "Commutativity of ≡")

AOT_theorem "possit-sit:3": ‹◇Situation(x) ≡ □Situation(x)›        (458.3)
  using "possit-sit:1" "possit-sit:2" by (meson "≡E"(5))
```

```
AOT_theorem "possit-sit:4": ‹𝒜Situation(x) ≡ Situation(x)›     (458.4)
  by (meson "Act-Basic:5" "Act-Sub:2" "RA[2]" "≡E"(1) "≡E"(6) "possit-sit:2")

AOT_theorem "possit-sit:5": ‹Situation(∘p)›     (458.5)
proof (safe intro!: situations[THEN "≡dfI"] "&I" GEN "→I" "prop-prop1"[THEN "≡dfI"])
  AOT_have ‹∃F ∘p[F]›
    using "tv-id:2"[THEN "prop-enc"[THEN "≡dfE"], THEN "&E"(2)]
          "existential:1" "prop-prop2:2" by blast
  AOT_thus ‹A!∘p›
    by (safe intro!: "encoders-are-abstract"[unvarify x, THEN "→E"]
                     "t=t-proper:2"[THEN "→E", OF "ext-p-tv:3"])
next
  fix F
  AOT_assume ‹∘p[F]›
  AOT_hence ‹ιx(A!x & ∀F (x[F] ≡ ∃q ((q ≡ p) & F = [λy q])))[F]›
    using "tv-id:1" "rule=E" by fast
  AOT_hence ‹𝒜∃q ((q ≡ p) & F = [λy q])›
    using "≡E"(1) "desc-nec-encode:1" by fast
  AOT_hence ‹∃q 𝒜((q ≡ p) & F = [λy q])›
    by (metis "Act-Basic:10" "≡E"(1))
  then AOT_obtain q where ‹𝒜((q ≡ p) & F = [λy q])› using "∃E"[rotated] by blast
  AOT_hence ‹𝒜F = [λy q]› by (metis "Act-Basic:2" "con-dis-i-e:2:b" "intro-elim:3:a")
  AOT_hence ‹F = [λy q]›
    using "id-act:1"[unvarify β, THEN "≡E"(2)] by (metis "prop-prop2:2")
  AOT_thus ‹∃p F = [λy p]›
    using "∃I" by fast
qed

AOT_theorem "possit-sit:6": ‹Situation(⊤)›     (458.6)
proof -
  AOT_have true_def: ‹⊢□ ⊤ = ιx (A!x & ∀F (x[F] ≡ ∃p(p & F = [λy p])))›
    by (simp add: "A-descriptions" "rule-id-df:1[zero]" "the-true:1")
  AOT_hence true_den: ‹⊢□ ⊤↓›
    using "t=t-proper:1" "vdash-properties:6" by blast
  AOT_have ‹𝒜TruthValue(⊤)›
    using "actual-desc:2"[unvarify x, OF true_den, THEN "→E", OF true_def]
    using "TV-lem2:1"[unvarify x, OF true_den, THEN "RA[2]",
                      THEN "act-cond"[THEN "→E"], THEN "→E"]
    by blast
  AOT_hence ‹𝒜Situation(⊤)›
    using "T-sit"[unvarify x, OF true_den, THEN "RA[2]",
                  THEN "act-cond"[THEN "→E"], THEN "→E"] by blast
  AOT_thus ‹Situation(⊤)›
    using "possit-sit:4"[unvarify x, OF true_den, THEN "≡E"(1)] by blast
qed

AOT_theorem "possit-sit:7": ‹Situation(⊥)›     (458.7)
proof -
  AOT_have true_def: ‹⊢□ ⊥ = ιx (A!x & ∀F (x[F] ≡ ∃p(¬p & F = [λy p])))›
    by (simp add: "A-descriptions" "rule-id-df:1[zero]" "the-true:2")
  AOT_hence true_den: ‹⊢□ ⊥↓›
    using "t=t-proper:1" "vdash-properties:6" by blast
  AOT_have ‹𝒜TruthValue(⊥)›
    using "actual-desc:2"[unvarify x, OF true_den, THEN "→E", OF true_def]
    using "TV-lem2:2"[unvarify x, OF true_den, THEN "RA[2]",
                      THEN "act-cond"[THEN "→E"], THEN "→E"]
    by blast
  AOT_hence ‹𝒜Situation(⊥)›
    using "T-sit"[unvarify x, OF true_den, THEN "RA[2]",
                  THEN "act-cond"[THEN "→E"], THEN "→E"] by blast
  AOT_thus ‹Situation(⊥)›
    using "possit-sit:4"[unvarify x, OF true_den, THEN "≡E"(1)] by blast
```

```
qed

AOT_register_rigid_restricted_type
  Situation: ‹Situation(κ)›
proof
  AOT_modally_strict {
    fix p
    AOT_obtain x where ‹TruthValueOf(x,p)›
      by (metis "instantiation" "p-has-!tv:1")
    AOT_hence ‹∃p TruthValueOf(x,p)› by (rule "∃I")
    AOT_hence ‹TruthValue(x)› by (metis "≡dfI" "T-value")
    AOT_hence ‹Situation(x)› using "T-sit"[THEN "→E"] by blast
    AOT_thus ‹∃x Situation(x)› by (rule "∃I")
  }
next
  AOT_modally_strict {
    AOT_show ‹Situation(κ) → κ↓› for κ
    proof (rule "→I")
      AOT_assume ‹Situation(κ)›
      AOT_hence ‹A!κ› by (metis "≡dfE" "&E"(1) situations)
      AOT_thus ‹κ↓› by (metis "russell-axiom[exe,1].ψ_denotes_asm")
    qed
  }
next
  AOT_modally_strict {
    AOT_show ‹∀α(Situation(α) → □Situation(α))›
      using "possit-sit:1"[THEN "conventions:3"[THEN "≡dfE"],
                           THEN "&E"(1)] GEN by fast
  }
qed

AOT_register_variable_names
  Situation: s

AOT_define TruthInSituation :: ‹τ ⇒ φ ⇒ φ› ("(_ ⊨/ _)" [100, 40] 100)
  "true-in-s": ‹s ⊨ p ≡df sΣp›                                          (459)

notepad
begin
  (* Verify precedence. *)
  fix x p q
  have ‹«x ⊨ p → q» = «(x ⊨ p) → q»›
    by simp
  have ‹«x ⊨ p & q» = «(x ⊨ p) & q»›
    by simp
  have ‹«x ⊨ ¬p» = «x ⊨ (¬p)»›
    by simp
  have ‹«x ⊨ □p» = «x ⊨ (□p)»›
    by simp
  have ‹«x ⊨ 𝒜p» = «x ⊨ (𝒜p)»›
    by simp
  have ‹«□x ⊨ p» = «□(x ⊨ p)»›
    by simp
  have ‹«¬x ⊨ p» = «¬(x ⊨ p)»›
    by simp
end


AOT_theorem lem1: ‹Situation(x) → (x ⊨ p ≡ x[λy p])›                   (460)
proof (rule "→I"; rule "≡I"; rule "→I")
  AOT_assume ‹Situation(x)›
  AOT_assume ‹x ⊨ p›
  AOT_hence ‹xΣp›
```

```
    using "true-in-s"[THEN "≡dfE"] "&E" by blast
  AOT_thus ‹x[λy p]› using "prop-enc"[THEN "≡dfE"] "&E" by blast
next
  AOT_assume 1: ‹Situation(x)›
  AOT_assume ‹x[λy p]›
  AOT_hence ‹xΣp›
    using "prop-enc"[THEN "≡dfI", OF "&I", OF "cqt:2"(1)] by blast
  AOT_thus ‹x ⊨ p›
    using "true-in-s"[THEN "≡dfI"] 1 "&I" by blast
qed

AOT_theorem "lem2:1": ‹s ⊨ p ≡ □s ⊨ p›                                    (462.1)
proof -
  AOT_have sit: ‹Situation(s)›
    by (simp add: Situation.ψ)
  AOT_have ‹s ⊨ p ≡ s[λy p]›
    using lem1[THEN "→E", OF sit] by blast
  also AOT_have ‹... ≡ □s[λy p]›
    by (rule "en-eq:2[1]"[unvarify F]) "cqt:2[lambda]"
  also AOT_have ‹... ≡ □s ⊨ p›
    using lem1[THEN RM, THEN "→E", OF "possit-sit:1"[THEN "≡E"(1), OF sit]]
    by (metis "KBasic:6" "≡E"(2) "Commutativity of ≡" "→E")
  finally show ?thesis.
qed

AOT_theorem "lem2:2": ‹◇s ⊨ p ≡ s ⊨ p›                                    (462.2)
proof -
  AOT_have ‹□(s ⊨ p → □s ⊨ p)›
    using "possit-sit:1"[THEN "≡E"(1), OF Situation.ψ]
          "lem2:1"[THEN "conventions:3"[THEN "≡dfE", THEN "&E"(1)]]
          RM[OF "→I", THEN "→E"] by blast
  thus ?thesis by (metis "B◇" "S5Basic:13" "T◇" "≡I" "≡E"(1) "→E")
qed

AOT_theorem "lem2:3": ‹◇s ⊨ p ≡ □s ⊨ p›                                   (462.3)
  using "lem2:1" "lem2:2" by (metis "≡E"(5))

AOT_theorem "lem2:4": ‹𝒜(s ⊨ p) ≡ s ⊨ p›                                  (462.4)
proof -
  AOT_have ‹□(s ⊨ p → □s ⊨ p)›
    using "possit-sit:1"[THEN "≡E"(1), OF Situation.ψ]
      "lem2:1"[THEN "conventions:3"[THEN "≡dfE", THEN "&E"(1)]]
      RM[OF "→I", THEN "→E"] by blast
  thus ?thesis
    using "sc-eq-fur:2"[THEN "→E"] by blast
qed

AOT_theorem "lem2:5": ‹¬s ⊨ p ≡ □¬s ⊨ p›                                  (462.5)
  by (metis "KBasic2:1" "contraposition:1[2]" "→I" "≡I" "≡E"(3) "≡E"(4) "lem2:2")

AOT_theorem "sit-identity": ‹s = s' ≡ ∀p(s ⊨ p ≡ s' ⊨ p)›                  (463)
proof(rule "≡I"; rule "→I")
  AOT_assume ‹s = s'›
  moreover AOT_have ‹∀p(s ⊨ p ≡ s ⊨ p)›
    by (simp add: "oth-class-taut:3:a" "universal-cor")
  ultimately AOT_show ‹∀p(s ⊨ p ≡ s' ⊨ p)›
    using "rule=E" by fast
next
  AOT_assume a: ‹∀p (s ⊨ p ≡ s' ⊨ p)›
  AOT_show ‹s = s'›
  proof(safe intro!: "ab-obey:1"[THEN "→E", THEN "→E"] "&I" GEN "≡I" "→I")
    AOT_show ‹A!s› using Situation.ψ "≡dfE" "&E"(1) situations by blast
  next
```

```
    AOT_show ‹A!s'› using Situation.ψ "≡dfE" "&E"(1) situations by blast
  next
    fix F
    AOT_assume 0: ‹s[F]›
    AOT_hence ‹∃p (F = [λy p])›
      using Situation.ψ[THEN situations[THEN "≡dfE"], THEN "&E"(2),
                        THEN "∀E"(2)[where β=F], THEN "→E"]
            "prop-prop1"[THEN "≡dfE"] by blast
    then AOT_obtain p where F_def: ‹F = [λy p]›
      using "∃E" by metis
    AOT_hence ‹s[λy p]›
      using 0 "rule=E" by blast
    AOT_hence ‹s ⊨ p›
      using lem1[THEN "→E", OF Situation.ψ, THEN "≡E"(2)] by blast
    AOT_hence ‹s' ⊨ p›
      using a[THEN "∀E"(2)[where β=p], THEN "≡E"(1)] by blast
    AOT_hence ‹s'[λy p]›
      using lem1[THEN "→E", OF Situation.ψ, THEN "≡E"(1)] by blast
    AOT_thus ‹s'[F]›
      using F_def[symmetric] "rule=E" by blast
  next
    fix F
    AOT_assume 0: ‹s'[F]›
    AOT_hence ‹∃p (F = [λy p])›
      using Situation.ψ[THEN situations[THEN "≡dfE"], THEN "&E"(2),
                        THEN "∀E"(2)[where β=F], THEN "→E"]
            "prop-prop1"[THEN "≡dfE"] by blast
    then AOT_obtain p where F_def: ‹F = [λy p]›
      using "∃E" by metis
    AOT_hence ‹s'[λy p]›
      using 0 "rule=E" by blast
    AOT_hence ‹s' ⊨ p›
      using lem1[THEN "→E", OF Situation.ψ, THEN "≡E"(2)] by blast
    AOT_hence ‹s ⊨ p›
      using a[THEN "∀E"(2)[where β=p], THEN "≡E"(2)] by blast
    AOT_hence ‹s[λy p]›
      using lem1[THEN "→E", OF Situation.ψ, THEN "≡E"(1)] by blast
    AOT_thus ‹s[F]›
      using F_def[symmetric] "rule=E" by blast
  qed
qed

AOT_define PartOfSituation :: ‹τ ⇒ τ ⇒ φ› (infixl ‹⊴› 80)
  "sit-part-whole": ‹s ⊴ s' ≡df ∀p (s ⊨ p → s' ⊨ p)›                    (464)

AOT_theorem "part:1": ‹s ⊴ s›                                           (465.1)
  by (rule "sit-part-whole"[THEN "≡dfI"])
     (safe intro!: "&I" Situation.ψ GEN "→I")

AOT_theorem "part:2": ‹s ⊴ s' & s ≠ s' → ¬(s' ⊴ s)›                   (465.2)
proof(rule "→I"; frule "&E"(1); drule "&E"(2); rule "raa-cor:2")
  AOT_assume 0: ‹s ⊴ s'›
  AOT_hence a: ‹s ⊨ p → s' ⊨ p› for p
    using "∀E"(2) "sit-part-whole"[THEN "≡dfE"] "&E" by blast
  AOT_assume ‹s' ⊴ s›
  AOT_hence b: ‹s' ⊨ p → s ⊨ p› for p
    using "∀E"(2) "sit-part-whole"[THEN "≡dfE"] "&E" by blast
  AOT_have ‹∀p (s ⊨ p ≡ s' ⊨ p)›
    using a b by (simp add: "≡I" "universal-cor")
  AOT_hence 1: ‹s = s'›
    using "sit-identity"[THEN "≡E"(2)] by metis
  AOT_assume ‹s ≠ s'›
  AOT_hence ‹¬(s = s')›
```

```
    by (metis "≡dfE" "=-infix")
  AOT_thus ‹s = s' & ¬(s = s')›
    using 1 "&I" by blast
qed

AOT_theorem "part:3": ‹s ⊴ s' & s' ⊴ s" → s ⊴ s"›                    (465.3)
proof(rule "→I"; frule "&E"(1); drule "&E"(2);
      safe intro!: "&I" GEN "→I" "sit-part-whole"[THEN "≡dfI"] Situation.ψ)
  fix p
  AOT_assume ‹s ⊨ p›
  moreover AOT_assume ‹s ⊴ s'›
  ultimately AOT_have ‹s' ⊨ p›
    using "sit-part-whole"[THEN "≡dfE", THEN "&E"(2),
                           THEN "∀E"(2)[where β=p], THEN "→E"] by blast
  moreover AOT_assume ‹s' ⊴ s"›
  ultimately AOT_show ‹s" ⊨ p›
    using "sit-part-whole"[THEN "≡dfE", THEN "&E"(2),
                           THEN "∀E"(2)[where β=p], THEN "→E"] by blast
qed

AOT_theorem "sit-identity2:1": ‹s = s' ≡ s ⊴ s' & s' ⊴ s›            (466.1)
proof (safe intro!: "≡I" "&I" "→I")
  AOT_show ‹s ⊴ s'› if ‹s = s'›
    using "rule=E" "part:1" that by blast
next
  AOT_show ‹s' ⊴ s› if ‹s = s'›
    using "rule=E" "part:1" that[symmetric] by blast
next
  AOT_assume ‹s ⊴ s' & s' ⊴ s›
  AOT_thus ‹s = s'› using "part:2"[THEN "→E", OF "&I"]
    by (metis "≡dfI" "&E"(1) "&E"(2) "=-infix" "raa-cor:3")
qed

AOT_theorem "sit-identity2:2": ‹s = s' ≡ ∀s" (s" ⊴ s ≡ s" ⊴ s')›     (466.2)
proof(safe intro!: "≡I" "→I" Situation.GEN "sit-identity"[THEN "≡E"(2)]
                   GEN[where 'a=o])
  AOT_show ‹s" ⊴ s'› if ‹s" ⊴ s› and ‹s = s'› for s"
    using "rule=E" that by blast
next
  AOT_show ‹s" ⊴ s› if ‹s" ⊴ s'› and ‹s = s'› for s"
    using "rule=E" id_sym that by blast
next
  AOT_show ‹s' ⊨ p› if ‹s ⊨ p› and ‹∀s" (s" ⊴ s ≡ s" ⊴ s')› for p
    using "sit-part-whole"[THEN "≡dfE", THEN "&E"(2),
            OF that(2)[THEN "Situation.∀E", THEN "≡E"(1), OF "part:1"],
            THEN "∀E"(2), THEN "→E", OF that(1)].
next
  AOT_show ‹s ⊨ p› if ‹s' ⊨ p› and ‹∀s" (s" ⊴ s ≡ s" ⊴ s')› for p
    using "sit-part-whole"[THEN "≡dfE", THEN "&E"(2),
        OF that(2)[THEN "Situation.∀E", THEN "≡E"(2), OF "part:1"],
        THEN "∀E"(2), THEN "→E", OF that(1)].
qed

AOT_define Persistent :: ‹φ ⇒ φ› (‹Persistent'(_')›)
  persistent: ‹Persistent(p) ≡df ∀s (s ⊨ p → ∀s' (s ⊴ s' → s' ⊨ p))›  (467)

AOT_theorem "pers-prop": ‹∀p Persistent(p)›                           (468)
  by (safe intro!: GEN[where 'a=o] Situation.GEN persistent[THEN "≡dfI"] "→I")
     (simp add: "sit-part-whole"[THEN "≡dfE", THEN "&E"(2), THEN "∀E"(2), THEN "→E"])

AOT_define NullSituation :: ‹τ ⇒ φ› (‹NullSituation'(_')›)
  "df-null-trivial:1": ‹NullSituation(s) ≡df ¬∃p s ⊨ p›               (469.1)
```

```
AOT_define TrivialSituation :: ‹τ ⇒ φ› (‹TrivialSituation'(_')›)
  "df-null-trivial:2": ‹TrivialSituation(s) ≡df ∀p s ⊨ p›                (469.2)

AOT_theorem "thm-null-trivial:1": ‹∃!x NullSituation(x)›                  (470.1)
proof (AOT_subst ‹NullSituation(x)› ‹A!x & ∀F (x[F] ≡ F ≠ F)› for: x)
  AOT_modally_strict {
    AOT_show ‹NullSituation(x) ≡ A!x & ∀F (x[F] ≡ F ≠ F)› for x
    proof (safe intro!: "≡I" "→I" "df-null-trivial:1"[THEN "≡dfI"]
                dest!: "df-null-trivial:1"[THEN "≡dfE"])
      AOT_assume 0: ‹Situation(x) & ¬∃p x ⊨ p›
      AOT_have 1: ‹A!x›
        using 0[THEN "&E"(1), THEN situations[THEN "≡dfE"], THEN "&E"(1)].
      AOT_have 2: ‹x[F] → ∃p F = [λy p]› for F
        using 0[THEN "&E"(1), THEN situations[THEN "≡dfE"],
                THEN "&E"(2), THEN "∀E"(2)]
        by (metis "≡dfE" "→I" "prop-prop1" "→E")
      AOT_show ‹A!x & ∀F (x[F] ≡ F ≠ F)›
      proof (safe intro!: "&I" 1 GEN "≡I" "→I")
        fix F
        AOT_assume ‹x[F]›
        moreover AOT_obtain p where ‹F = [λy p]›
          using calculation 2[THEN "→E"] "∃E"[rotated] by blast
        ultimately AOT_have ‹x[λy p]›
          by (metis "rule=E")
        AOT_hence ‹x ⊨ p›
          using lem1[THEN "→E", OF 0[THEN "&E"(1)], THEN "≡E"(2)] by blast
        AOT_hence ‹∃p (x ⊨ p)›
          by (rule "∃I")
        AOT_thus ‹F ≠ F›
          using 0[THEN "&E"(2)] "raa-cor:1" "&I" by blast
      next
        fix F :: ‹<κ> AOT_var›
        AOT_assume ‹F ≠ F›
        AOT_hence ‹¬(F = F)› by (metis "≡dfE" "=-infix")
        moreover AOT_have ‹F = F›
          by (simp add: "id-eq:1")
        ultimately AOT_show ‹x[F]› using "&I" "raa-cor:1" by blast
      qed
    next
      AOT_assume 0: ‹A!x & ∀F (x[F] ≡ F ≠ F)›
      AOT_hence ‹x[F] ≡ F ≠ F› for F
        using "∀E" "&E" by blast
      AOT_hence 1: ‹¬x[F]› for F
        using "≡dfE" "id-eq:1" "=-infix" "reductio-aa:1" "≡E"(1) by blast
      AOT_show ‹Situation(x) & ¬∃p x ⊨ p›
      proof (safe intro!: "&I" situations[THEN "≡dfI"] 0[THEN "&E"(1)] GEN "→I")
        AOT_show ‹Propositional([F])› if ‹x[F]› for F
          using that 1 "&I" "raa-cor:1" by fast
      next
        AOT_show ‹¬∃p x ⊨ p›
        proof(rule "raa-cor:2")
          AOT_assume ‹∃p x ⊨ p›
          then AOT_obtain p where ‹x ⊨ p› using "∃E"[rotated] by blast
          AOT_hence ‹x[λy p]›
            using "≡dfE" "&E"(1) "≡E"(1) lem1 "modus-tollens:1"
                  "raa-cor:3" "true-in-s" by fast
          moreover AOT_have ‹¬x[λy p]›
            by (rule 1[unvarify F]) "cqt:2[lambda]"
          ultimately AOT_show ‹p & ¬p› for p using "&I" "raa-cor:1" by blast
        qed
      qed
    qed
  }
```

```
next
  AOT_show ‹∃!x ([A!]x & ∀F (x[F] ≡ F ≠ F))›
    by (simp add: "A-objects!")
qed


AOT_theorem "thm-null-trivial:2": ‹∃!x TrivialSituation(x)›                    (470.2)
proof (AOT_subst ‹TrivialSituation(x)› ‹A!x & ∀F (x[F] ≡ ∃p F = [λy p])› for: x)
  AOT_modally_strict {
    AOT_show ‹TrivialSituation(x) ≡ A!x & ∀F (x[F] ≡ ∃p F = [λy p])› for x
    proof (safe intro!: "≡I" "→I" "df-null-trivial:2"[THEN "≡dfI"]
                 dest!: "df-null-trivial:2"[THEN "≡dfE"])
      AOT_assume 0: ‹Situation(x) & ∀p x ⊨ p›
      AOT_have 1: ‹A!x›
        using 0[THEN "&E"(1), THEN situations[THEN "≡dfE"], THEN "&E"(1)].
      AOT_have 2: ‹x[F] → ∃p F = [λy p]› for F
        using 0[THEN "&E"(1), THEN situations[THEN "≡dfE"],
                THEN "&E"(2), THEN "∀E"(2)]
        by (metis "≡dfE" "deduction-theorem" "prop-prop1" "→E")
      AOT_show ‹A!x & ∀F (x[F] ≡ ∃p F = [λy p])›
      proof (safe intro!: "&I" 1 GEN "≡I" "→I" 2)
        fix F
        AOT_assume ‹∃p F = [λy p]›
        then AOT_obtain p where ‹F = [λy p]›
          using "∃E"[rotated] by blast
        moreover AOT_have ‹x ⊨ p›
          using 0[THEN "&E"(2)] "∀E" by blast
        ultimately AOT_show ‹x[F]›
          by (metis 0 "rule=E" "&E"(1) id_sym "≡E"(2) lem1
                    "Commutativity of ≡" "→E")
      qed
    next
      AOT_assume 0: ‹A!x & ∀F (x[F] ≡ ∃p F = [λy p])›
      AOT_hence 1: ‹x[F] ≡ ∃p F = [λy p]› for F
        using "∀E" "&E" by blast
      AOT_have 2: ‹Situation(x)›
      proof (safe intro!: "&I" situations[THEN "≡dfI"] 0[THEN "&E"(1)] GEN "→I")
        AOT_show ‹Propositional([F])› if ‹x[F]› for F
          using 1[THEN "≡E"(1), OF that]
          by (metis "≡dfI" "prop-prop1")
      qed
      AOT_show ‹Situation(x) & ∀p (x ⊨ p)›
      proof (safe intro!: "&I" 2 0[THEN "&E"(1)] GEN "→I")
        AOT_have ‹x[λy p] ≡ ∃q [λy p] = [λy q]› for p
          by (rule 1[unvarify F, where τ="«[λy p]»"]) "cqt:2[lambda]"
        moreover AOT_have ‹∃q [λy p] = [λy q]› for p
          by (rule "∃I"(2)[where β=p])
             (simp add: "rule=I:1" "prop-prop2:2")
        ultimately AOT_have ‹x[λy p]› for p by (metis "≡E"(2))
        AOT_thus ‹x ⊨ p› for p
          by (metis "2" "≡E"(2) lem1 "→E")
      qed
    qed
  }
next
  AOT_show ‹∃!x ([A!]x & ∀F (x[F] ≡ ∃p F = [λy p]))›
    by (simp add: "A-objects!")
qed

AOT_theorem "thm-null-trivial:3": ‹ιx NullSituation(x)↓›                      (470.3)
  by (meson "A-Exists:2" "RA[2]" "≡E"(2) "thm-null-trivial:1")

AOT_theorem "thm-null-trivial:4": ‹ιx TrivialSituation(x)↓›                   (470.4)
```

```
  using "A-Exists:2" "RA[2]" "≡E"(2) "thm-null-trivial:2" by blast

AOT_define TheNullSituation :: ‹κs› (‹s∅›)
  "df-the-null-sit:1": ‹s∅ =df ιx NullSituation(x)›                    (471.1)

AOT_define TheTrivialSituation :: ‹κs› (‹sV›)
  "df-the-null-sit:2": ‹sV =df ιx TrivialSituation(x)›                 (471.2)

AOT_theorem "null-triv-sc:1": ‹NullSituation(x) → □NullSituation(x)›  (472.1)
proof(safe intro!: "→I" dest!: "df-null-trivial:1"[THEN "≡dfE"];
      frule "&E"(1); drule "&E"(2))
  AOT_assume 1: ‹¬∃p (x ⊨ p)›
  AOT_assume 0: ‹Situation(x)›
  AOT_hence ‹□Situation(x)› by (metis "≡E"(1) "possit-sit:1")
  moreover AOT_have ‹□¬∃p (x ⊨ p)›
  proof(rule "raa-cor:1")
    AOT_assume ‹¬□¬∃p (x ⊨ p)›
    AOT_hence ‹◇∃p (x ⊨ p)›
      by (metis "≡dfI" "conventions:5")
    AOT_hence ‹∃p ◇(x ⊨ p)› by (metis "BF◇" "→E")
    then AOT_obtain p where ‹◇(x ⊨ p)› using "∃E"[rotated] by blast
    AOT_hence ‹x ⊨ p›
      by (metis "≡E"(1) "lem2:2"[unconstrain s, THEN "→E", OF 0])
    AOT_hence ‹∃p x ⊨ p› using "∃I" by fast
    AOT_thus ‹∃p x ⊨ p & ¬∃p x ⊨ p› using 1 "&I" by blast
  qed
  ultimately AOT_have 2: ‹□(Situation(x) & ¬∃p x ⊨ p)›
    by (metis "KBasic:3" "&I" "≡E"(2))
  AOT_show ‹□NullSituation(x)›
    by (AOT_subst ‹NullSituation(x)› ‹Situation(x) & ¬∃p x ⊨ p›)
       (auto simp: "df-null-trivial:1" "≡Df" 2)
qed


AOT_theorem "null-triv-sc:2": ‹TrivialSituation(x) → □TrivialSituation(x)›  (472.2)
proof(safe intro!: "→I" dest!: "df-null-trivial:2"[THEN "≡dfE"];
      frule "&E"(1); drule "&E"(2))
  AOT_assume 0: ‹Situation(x)›
  AOT_hence 1: ‹□Situation(x)› by (metis "≡E"(1) "possit-sit:1")
  AOT_assume ‹∀p x ⊨ p›
  AOT_hence ‹x ⊨ p› for p
    using "∀E" by blast
  AOT_hence ‹□x ⊨ p› for p
    using  0 "≡E"(1) "lem2:1"[unconstrain s, THEN "→E"] by blast
  AOT_hence ‹∀p □x ⊨ p›
    by (rule GEN)
  AOT_hence ‹□∀p x ⊨ p›
    by (rule BF[THEN "→E"])
  AOT_hence 2: ‹□(Situation(x) & ∀p x ⊨ p)›
    using 1 by (metis "KBasic:3" "&I" "≡E"(2))
  AOT_show ‹□TrivialSituation(x)›
    by (AOT_subst ‹TrivialSituation(x)› ‹Situation(x) & ∀p x ⊨ p›)
       (auto simp: "df-null-trivial:2" "≡Df" 2)
qed

AOT_theorem "null-triv-sc:3": ‹NullSituation(s∅)›                        (472.3)
  by (safe intro!: "df-the-null-sit:1"[THEN "=dfI"(2)] "thm-null-trivial:3"
       "rule=I:1"[OF "thm-null-trivial:3"]
       "!box-desc:2"[THEN "→E", THEN "→E", rotated, OF "thm-null-trivial:1",
                     OF "∀I", OF "null-triv-sc:1", THEN "∀E"(1), THEN "→E"])

AOT_theorem "null-triv-sc:4": ‹TrivialSituation(sV)›                     (472.4)
  by (safe intro!: "df-the-null-sit:2"[THEN "=dfI"(2)] "thm-null-trivial:4"
```

```
      "rule=I:1"[OF "thm-null-trivial:4"]
      "!box-desc:2"[THEN "→E", THEN "→E", rotated, OF "thm-null-trivial:2",
                    OF "∀I", OF "null-triv-sc:2", THEN "∀E"(1), THEN "→E"])

AOT_theorem "null-triv-facts:1": ‹NullSituation(x) ≡ Null(x)›                    (473.1)
proof (safe intro!: "≡I" "→I" "df-null-uni:1"[THEN "≡dfI"]
                    "df-null-trivial:1"[THEN "≡dfI"]
            dest!: "df-null-uni:1"[THEN "≡dfE"] "df-null-trivial:1"[THEN "≡dfE"])
  AOT_assume 0: ‹Situation(x) & ¬∃p x ⊨ p›
  AOT_have 1: ‹x[F] → ∃p F = [λy p]› for F
    using 0[THEN "&E"(1), THEN situations[THEN "≡dfE"], THEN "&E"(2), THEN "∀E"(2)]
    by (metis "≡dfE" "deduction-theorem" "prop-prop1" "→E")
  AOT_show ‹A!x & ¬∃F x[F]›
  proof (safe intro!: "&I" 0[THEN "&E"(1), THEN situations[THEN "≡dfE"],
                               THEN "&E"(1)];
         rule "raa-cor:2")
    AOT_assume ‹∃F x[F]›
    then AOT_obtain F where F_prop: ‹x[F]›
      using "∃E"[rotated] by blast
    AOT_hence ‹∃p F = [λy p]›
      using 1[THEN "→E"] by blast
    then AOT_obtain p where ‹F = [λy p]›
      using "∃E"[rotated] by blast
    AOT_hence ‹x[λy p]›
      by (metis "rule=E" F_prop)
    AOT_hence ‹x ⊨ p›
      using lem1[THEN "→E", OF 0[THEN "&E"(1)], THEN "≡E"(2)] by blast
    AOT_hence ‹∃p x ⊨ p›
      by (rule "∃I")
    AOT_thus ‹∃p x ⊨ p & ¬∃p x ⊨ p›
      using 0[THEN "&E"(2)] "&I" by blast
  qed
next
  AOT_assume 0: ‹A!x & ¬∃F x[F]›
  AOT_have ‹Situation(x)›
    apply (rule situations[THEN "≡dfI", OF "&I", OF 0[THEN "&E"(1)]]; rule GEN)
    using 0[THEN "&E"(2)] by (metis "→I" "existential:2[const_var]" "raa-cor:3")
  moreover AOT_have ‹¬∃p x ⊨ p›
  proof (rule "raa-cor:2")
    AOT_assume ‹∃p x ⊨ p›
    then AOT_obtain p where ‹x ⊨ p› by (metis "instantiation")
    AOT_hence ‹x[λy p]› by (metis "≡dfE" "&E"(2) "prop-enc" "true-in-s")
    AOT_hence ‹∃F x[F]› by (rule "∃I") "cqt:2[lambda]"
    AOT_thus ‹∃F x[F] & ¬∃F x[F]› using 0[THEN "&E"(2)] "&I" by blast
  qed
  ultimately AOT_show ‹Situation(x) & ¬∃p x ⊨ p› using "&I" by blast
qed

AOT_theorem "null-triv-facts:2": ‹s∅ = a∅›                                        (473.2)
  apply (rule "=dfI"(2)[OF "df-the-null-sit:1"])
   apply (fact "thm-null-trivial:3")
  apply (rule "=dfI"(2)[OF "df-null-uni-terms:1"])
   apply (fact "null-uni-uniq:3")
  apply (rule "equiv-desc-eq:3"[THEN "→E"])
  apply (rule "&I")
   apply (fact "thm-null-trivial:3")
  by (rule RN; rule GEN; rule "null-triv-facts:1")

AOT_theorem "null-triv-facts:3": ‹sV ≠ aV›                                        (473.3)
proof(rule "=-infix"[THEN "≡dfI"])
  AOT_have ‹Universal(aV)›
    by (simp add: "null-uni-facts:4")
  AOT_hence 0: ‹aV[A!]›
```

```
    using "df-null-uni:2"[THEN "≡_dfE"] "&E" "∀E"(1)
    by (metis "cqt:5:a" "vdash-properties:10" "vdash-properties:1[2]")
  moreover AOT_have 1: ‹¬s_V[A!]›
  proof(rule "raa-cor:2")
    AOT_have ‹Situation(s_V)›
      using "≡_dfE" "&E"(1) "df-null-trivial:2" "null-triv-sc:4" by blast
    AOT_hence ‹∀F (s_V[F] → Propositional([F]))›
      by (metis "≡_dfE" "&E"(2) situations)
    moreover AOT_assume ‹s_V[A!]›
    ultimately AOT_have ‹Propositional(A!)›
      using "∀E"(1)[rotated, OF "oa-exist:2"] "→E" by blast
    AOT_thus ‹Propositional(A!) & ¬Propositional(A!)›
      using "prop-in-f:4:d" "&I" by blast
  qed
  AOT_show ‹¬(s_V = a_V)›
  proof (rule "raa-cor:2")
    AOT_assume ‹s_V = a_V›
    AOT_hence ‹s_V[A!]› using 0 "rule=E" id_sym by fast
    AOT_thus ‹s_V[A!] & ¬s_V[A!]› using 1 "&I" by blast
  qed
qed

definition ConditionOnPropositionalProperties :: ‹(<κ> ⇒ o) ⇒ bool› where
  "cond-prop": ‹ConditionOnPropositionalProperties ≡ λ φ . ∀ v .             (474)
                         [v ⊨ ∀F (φ{F} → Propositional([F]))]›

syntax ConditionOnPropositionalProperties :: ‹id_position ⇒ AOT_prop›
  ("CONDITION'_ON'_PROPOSITIONAL'_PROPERTIES'(_')")

AOT_theorem "cond-prop[E]":                                                   (474)
  assumes ‹CONDITION_ON_PROPOSITIONAL_PROPERTIES(φ)›
  shows ‹∀F (φ{F} → Propositional([F]))›
  using assms[unfolded "cond-prop"] by auto

AOT_theorem "cond-prop[I]":                                                   (474)
  assumes ‹⊢□ ∀F (φ{F} → Propositional([F]))›
  shows ‹CONDITION_ON_PROPOSITIONAL_PROPERTIES(φ)›
  using assms "cond-prop" by metis

AOT_theorem "pre-comp-sit":                                                   (475)
  assumes ‹CONDITION_ON_PROPOSITIONAL_PROPERTIES(φ)›
  shows ‹(Situation(x) & ∀F (x[F] ≡ φ{F})) ≡ (A!x & ∀F (x[F] ≡ φ{F}))›
proof(rule "≡I"; rule "→I")
  AOT_assume ‹Situation(x) & ∀F (x[F] ≡ φ{F})›
  AOT_thus ‹A!x & ∀F (x[F] ≡ φ{F})›
    using "&E" situations[THEN "≡_dfE"] "&I" by blast
next
  AOT_assume 0: ‹A!x & ∀F (x[F] ≡ φ{F})›
  AOT_show ‹Situation(x) & ∀F (x[F] ≡ φ{F})›
  proof (safe intro!: situations[THEN "≡_dfI"] "&I")
    AOT_show ‹A!x› using 0[THEN "&E"(1)].
  next
    AOT_show ‹∀F (x[F] → Propositional([F]))›
    proof(rule GEN; rule "→I")
      fix F
      AOT_assume ‹x[F]›
      AOT_hence ‹φ{F}›
        using 0[THEN "&E"(2)] "∀E" "≡E" by blast
      AOT_thus ‹Propositional([F])›
        using "cond-prop[E]"[OF assms] "∀E" "→E" by blast
    qed
  next
    AOT_show ‹∀F (x[F] ≡ φ{F})› using 0 "&E" by blast
```

```
  qed
qed

AOT_theorem "comp-sit:1":                                                    (476.1)
  assumes ‹CONDITION_ON_PROPOSITIONAL_PROPERTIES(φ)›
  shows ‹∃s ∀F(s[F] ≡ φ{F})›
  by (AOT_subst ‹Situation(x) & ∀F(x[F] ≡ φ{F})› ‹A!x & ∀F (x[F] ≡ φ{F})› for: x)
     (auto simp: "pre-comp-sit"[OF assms] "A-objects"[where φ=φ, axiom_inst])

AOT_theorem "comp-sit:2":                                                    (476.2)
  assumes ‹CONDITION_ON_PROPOSITIONAL_PROPERTIES(φ)›
  shows ‹∃!s ∀F(s[F] ≡ φ{F})›
  by (AOT_subst ‹Situation(x) & ∀F(x[F] ≡ φ{F})› ‹A!x & ∀F (x[F] ≡ φ{F})› for: x)
     (auto simp: assms "pre-comp-sit"  "pre-comp-sit"[OF assms] "A-objects!")

AOT_theorem "can-sit-desc:1":                                                (477.1)
  assumes ‹CONDITION_ON_PROPOSITIONAL_PROPERTIES(φ)›
  shows ‹ιs(∀F (s[F] ≡ φ{F}))↓›
  using "comp-sit:2"[OF assms] "A-Exists:2" "RA[2]" "≡E"(2) by blast

AOT_theorem "can-sit-desc:2":                                                (477.2)
  assumes ‹CONDITION_ON_PROPOSITIONAL_PROPERTIES(φ)›
  shows ‹ιs(∀F (s[F] ≡ φ{F})) = ιx(A!x & ∀F (x[F] ≡ φ{F}))›
  by (auto intro!: "equiv-desc-eq:2"[THEN "→E", OF "&I",
                                   OF "can-sit-desc:1"[OF assms]]
                   "RA[2]" GEN "pre-comp-sit"[OF assms])

AOT_theorem "strict-sit":                                                    (478)
  assumes ‹RIGID_CONDITION(φ)›
      and ‹CONDITION_ON_PROPOSITIONAL_PROPERTIES(φ)›
    shows ‹y = ιs(∀F (s[F] ≡ φ{F})) → ∀F (y[F] ≡ φ{F})›
  using "rule=E"[rotated, OF "can-sit-desc:2"[OF assms(2), symmetric]]
        "box-phi-a:2"[OF assms(1)] "→E" "→I" "&E" by fast

(* TODO: exercise (479) sit-lit *)

AOT_define actual :: ‹τ ⇒ φ› (‹Actual'(_')›)                                (481)
  ‹Actual(s) ≡df ∀p (s ⊨ p → p)›

AOT_theorem "act-and-not-pos": ‹∃s (Actual(s) & ◇¬Actual(s))›               (482)
proof -
  AOT_obtain q₁ where q₁_prop: ‹q₁ & ◇¬q₁›
    by (metis "≡dfE" "instantiation" "cont-tf:1" "cont-tf-thm:1")
  AOT_have ‹∃s (∀F (s[F] ≡ F = [λy q₁]))›
  proof (safe intro!: "comp-sit:1" "cond-prop[I]" GEN "→I")
    AOT_modally_strict {
      AOT_show ‹Propositional([F])› if ‹F = [λy q₁]› for F
        using "≡dfI" "existential:2[const_var]" "prop-prop1" that by fastforce
    }
  qed
  then AOT_obtain s₁ where s_prop: ‹∀F (s₁[F] ≡ F = [λy q₁])›
    using "Situation.∃E"[rotated] by meson
  AOT_have ‹Actual(s₁)›
  proof(safe intro!: actual[THEN "≡dfI"] "&I" GEN "→I" s_prop Situation.ψ)
    fix p
    AOT_assume ‹s₁ ⊨ p›
    AOT_hence ‹s₁[λy p]›
      by (metis "≡dfE" "&E"(2) "prop-enc" "true-in-s")
    AOT_hence ‹[λy p] = [λy q₁]›
      by (rule s_prop[THEN "∀E"(1), THEN "≡E"(1), rotated]) "cqt:2[lambda]"
    AOT_hence ‹p = q₁› by (metis "≡E"(2) "p-identity-thm2:3")
    AOT_thus ‹p› using q₁_prop[THEN "&E"(1)] "rule=E" id_sym by fast
  qed
```

```
  moreover AOT_have ‹◇¬Actual(s1)›
  proof(rule "raa-cor:1"; drule "KBasic:12"[THEN "≡E"(2)])
    AOT_assume ‹□Actual(s1)›
    AOT_hence ‹□(Situation(s1) & ∀p (s1 ⊨ p → p))›
      using actual[THEN "≡Df", THEN "conventions:3"[THEN "≡dfE"],
                   THEN "&E"(1), THEN RM, THEN "→E"] by blast
    AOT_hence ‹□∀p (s1 ⊨ p → p)›
      by (metis "RM:1" "Conjunction Simplification"(2) "→E")
    AOT_hence ‹∀p □(s1 ⊨ p → p)›
      by (metis "CBF" "vdash-properties:10")
    AOT_hence ‹□(s1 ⊨ q1 → q1)›
      using "∀E" by blast
    AOT_hence ‹□s1 ⊨ q1 → □q1›
      by (metis "→E" "qml:1" "vdash-properties:1[2]")
    moreover AOT_have ‹s1 ⊨ q1›
      using s_prop[THEN "∀E"(1), THEN "≡E"(2),
                   THEN lem1[THEN "→E", OF Situation.ψ, THEN "≡E"(2)]]
            "rule=I:1" "prop-prop2:2" by blast
    ultimately AOT_have ‹□q1›
      using "≡dfE" "&E"(1) "≡E"(1) "lem2:1" "true-in-s" "→E" by fast
    AOT_thus ‹◇¬q1 & ¬◇¬q1›
      using "KBasic:12"[THEN "≡E"(1)] q1_prop[THEN "&E"(2)] "&I" by blast
  qed
  ultimately AOT_have ‹(Actual(s1) & ◇¬Actual(s1))›
    using s_prop "&I" by blast
  thus ?thesis
    by (rule "Situation.∃I")
qed

AOT_theorem "actual-s:1": ‹∃s Actual(s)›                                  (484.1)
proof -
  AOT_obtain s where ‹(Actual(s) & ◇¬Actual(s))›
    using "act-and-not-pos" "Situation.∃E"[rotated] by meson
  AOT_hence ‹Actual(s)› using "&E" "&I" by metis
  thus ?thesis by (rule "Situation.∃I")
qed

AOT_theorem "actual-s:2": ‹∃s ¬Actual(s)›                                 (484.2)
proof(rule "∃I"(1)[where τ=‹«sV»›]; (rule "&I")?)
  AOT_show ‹Situation(sV)›
    using "≡dfE" "&E"(1) "df-null-trivial:2" "null-triv-sc:4" by blast
next
  AOT_show ‹¬Actual(sV)›
  proof(rule "raa-cor:2")
    AOT_assume 0: ‹Actual(sV)›
    AOT_obtain p1 where notp1: ‹¬p1›
      by (metis "∃E" "∃I"(1) "log-prop-prop:2" "non-contradiction")
    AOT_have ‹sV ⊨ p1›
      using "null-triv-sc:4"[THEN "≡dfE"[OF "df-null-trivial:2"], THEN "&E"(2)]
            "∀E" by blast
    AOT_hence ‹p1›
      using 0[THEN actual[THEN "≡dfE"], THEN "&E"(2), THEN "∀E"(2), THEN "→E"]
      by blast
    AOT_thus ‹p & ¬p› for p using notp1 by (metis "raa-cor:3")
  qed
next
  AOT_show ‹sV↓›
    using "df-the-null-sit:2" "rule-id-df:2:b[zero]" "thm-null-trivial:4" by blast
qed

AOT_theorem "actual-s:3": ‹∃p∀s(Actual(s) → ¬s ⊨ p)›                     (484.3)
proof -
  AOT_obtain p1 where notp1: ‹¬p1›
```

```
    by (metis "∃E" "∃I"(1) "log-prop-prop:2" "non-contradiction")
  AOT_have ‹∀s (Actual(s) → ¬(s ⊨ p₁))›
  proof (rule Situation.GEN; rule "→I"; rule "raa-cor:2")
    fix s
    AOT_assume ‹Actual(s)›
    moreover AOT_assume ‹s ⊨ p₁›
    ultimately AOT_have ‹p₁›
      using actual[THEN "≡dfE", THEN "&E"(2), THEN "∀E"(2), THEN "→E"] by blast
    AOT_thus ‹p₁ & ¬p₁›
      using notp₁ "&I" by simp
  qed
  thus ?thesis by (rule "∃I")
qed

AOT_theorem comp:                                                        (485)
  ‹∃s (s' ⊴ s & s" ⊴ s & ∀s"' (s' ⊴ s"' & s" ⊴ s"' → s ⊴ s"'))›
proof -
  have cond_prop: ‹ConditionOnPropositionalProperties (λ Π . «s'[Π] ∨ s"[Π]»)›
  proof(safe intro!: "cond-prop[I]" GEN "oth-class-taut:8:c"[THEN "→E", THEN "→E"];
        rule "→I")
    AOT_modally_strict {
      fix F
      AOT_have ‹Situation(s')›
        by (simp add: Situation.restricted_var_condition)
      AOT_hence ‹s'[F] → Propositional([F])›
        using "situations"[THEN "≡dfE", THEN "&E"(2), THEN "∀E"(2)] by blast
      moreover AOT_assume ‹s'[F]›
      ultimately AOT_show ‹Propositional([F])›
        using "→E" by blast
    }
  next
    AOT_modally_strict {
      fix F
      AOT_have ‹Situation(s")›
        by (simp add: Situation.restricted_var_condition)
      AOT_hence ‹s"[F] → Propositional([F])›
        using "situations"[THEN "≡dfE", THEN "&E"(2), THEN "∀E"(2)] by blast
      moreover AOT_assume ‹s"[F]›
      ultimately AOT_show ‹Propositional([F])›
        using "→E" by blast
    }
  qed
  AOT_obtain s₃ where ϑ: ‹∀F (s₃[F] ≡ s'[F] ∨ s"[F])›
    using "comp-sit:1"[OF cond_prop] "Situation.∃E"[rotated] by meson
  AOT_have ‹s' ⊴ s₃ & s" ⊴ s₃ & ∀s"' (s' ⊴ s"' & s" ⊴ s"' → s₃ ⊴ s"')›
  proof(safe intro!: "&I" "≡dfI"[OF "true-in-s"] "≡dfI"[OF "prop-enc"]
                     "Situation.GEN" "GEN"[where 'a=o] "→I"
                     "sit-part-whole"[THEN "≡dfI"]
                     Situation.ψ "cqt:2[const_var]"[axiom_inst])
    fix p
    AOT_assume ‹s' ⊨ p›
    AOT_hence ‹s'[λx p]›
      by (metis "&E"(2) "prop-enc" "≡dfE" "true-in-s")
    AOT_thus ‹s₃[λx p]›
      using ϑ[THEN "∀E"(1),OF "prop-prop2:2", THEN "≡E"(2), OF "∨I"(1)] by blast
  next
    fix p
    AOT_assume ‹s" ⊨ p›
    AOT_hence ‹s"[λx p]›
      by (metis "&E"(2) "prop-enc" "≡dfE" "true-in-s")
    AOT_thus ‹s₃[λx p]›
      using ϑ[THEN "∀E"(1),OF "prop-prop2:2", THEN "≡E"(2), OF "∨I"(2)] by blast
  next
```

```
      fix s p
      AOT_assume 0: ‹s' ⊴ s & s" ⊴ s›
      AOT_assume ‹s3 ⊨ p›
      AOT_hence ‹s3[λx p]›
        by (metis "&E"(2) "prop-enc" "≡dfE" "true-in-s")
      AOT_hence ‹s'[λx p] ∨ s"[λx p]›
        using ϑ[THEN "∀E"(1),OF "prop-prop2:2", THEN "≡E"(1)] by blast
      moreover {
        AOT_assume ‹s'[λx p]›
        AOT_hence ‹s' ⊨ p›
          by (safe intro!: "prop-enc"[THEN "≡dfI"] "true-in-s"[THEN "≡dfI"] "&I"
                           Situation.ψ "cqt:2[const_var]"[axiom_inst])
        moreover AOT_have ‹s' ⊨ p → s ⊨ p›
          using "sit-part-whole"[THEN "≡dfE", THEN "&E"(2)] 0[THEN "&E"(1)]
                "∀E"(2) by blast
        ultimately AOT_have ‹s ⊨ p›
          using "→E" by blast
        AOT_hence ‹s[λx p]›
          using "true-in-s"[THEN "≡dfE"] "prop-enc"[THEN "≡dfE"] "&E" by blast
      }
      moreover {
        AOT_assume ‹s"[λx p]›
        AOT_hence ‹s" ⊨ p›
          by (safe intro!: "prop-enc"[THEN "≡dfI"] "true-in-s"[THEN "≡dfI"] "&I"
                           Situation.ψ "cqt:2[const_var]"[axiom_inst])
        moreover AOT_have ‹s" ⊨ p → s ⊨ p›
          using "sit-part-whole"[THEN "≡dfE", THEN "&E"(2)] 0[THEN "&E"(2)]
                "∀E"(2) by blast
        ultimately AOT_have ‹s ⊨ p›
          using "→E" by blast
        AOT_hence ‹s[λx p]›
          using "true-in-s"[THEN "≡dfE"] "prop-enc"[THEN "≡dfE"] "&E" by blast
      }
      ultimately AOT_show ‹s[λx p]›
        by (metis "∨E"(1) "→I")
    qed
    thus ?thesis
      using "Situation.∃I" by fast
  qed

  AOT_theorem "act-sit:1": ‹Actual(s) → (s ⊨ p → [λy p]s)›                    (486.1)
  proof (safe intro!: "→I")
    AOT_assume ‹Actual(s)›
    AOT_hence p if ‹s ⊨ p›
      using actual[THEN "≡dfE", THEN "&E"(2), THEN "∀E"(2), THEN "→E"] that by blast
    moreover AOT_assume ‹s ⊨ p›
    ultimately AOT_have p by blast
    AOT_thus ‹[λy p]s›
      by (safe intro!: "β←C"(1) "cqt:2")
  qed

  AOT_theorem "act-sit:2":                                                    (486.2)
    ‹(Actual(s') & Actual(s")) → ∃x (Actual(x) & s' ⊴ x & s" ⊴ x)›
  proof(rule "→I"; frule "&E"(1); drule "&E"(2))
    AOT_assume act_s': ‹Actual(s')›
    AOT_assume act_s": ‹Actual(s")›
    have "cond-prop": ‹ConditionOnPropositionalProperties                       (474)
                       (λ Π . «∃p (Π = [λy p] & (s' ⊨ p ∨ s" ⊨ p))»)›
    proof (safe intro!: "cond-prop[I]"  "∀I" "→I" "prop-prop1"[THEN "≡dfI"])
      AOT_modally_strict {
        fix β
        AOT_assume ‹∃p (β = [λy p] & (s' ⊨ p ∨ s" ⊨ p))›
        then AOT_obtain p where ‹β = [λy p]› using "∃E"[rotated] "&E" by blast
```

```
      AOT_thus ‹∃p β = [λy p]› by (rule "∃I")
    }
  qed
  have rigid: ‹rigid_condition (λ Π . «∃p (Π = [λy p] & (s' ⊨ p ∨ s" ⊨ p))»)›
  proof(safe intro!: "strict-can:1[I]" "→I" GEN)
    AOT_modally_strict {
      fix F
      AOT_assume ‹∃p (F = [λy p] & (s' ⊨ p ∨ s" ⊨ p))›
      then AOT_obtain p₁ where p₁_prop: ‹F = [λy p₁] & (s' ⊨ p₁ ∨ s" ⊨ p₁)›
        using "∃E"[rotated] by blast
      AOT_hence ‹□(F = [λy p₁])›
        using "&E"(1) "id-nec:2" "vdash-properties:10" by blast
      moreover AOT_have ‹□(s' ⊨ p₁ ∨ s" ⊨ p₁)›
      proof(rule "∨E"; (rule "→I"; rule "KBasic:15"[THEN "→E"])?)
        AOT_show ‹s' ⊨ p₁ ∨ s" ⊨ p₁› using p₁_prop "&E" by blast
      next
        AOT_show ‹□s' ⊨ p₁ ∨ □s" ⊨ p₁› if ‹s' ⊨ p₁›
          apply (rule "∨I"(1))
          using "≡dfE" "&E"(1) "≡E"(1) "lem2:1" that "true-in-s" by blast
      next
        AOT_show ‹□s' ⊨ p₁ ∨ □s" ⊨ p₁› if ‹s" ⊨ p₁›
          apply (rule "∨I"(2))
          using "≡dfE" "&E"(1) "≡E"(1) "lem2:1" that "true-in-s" by blast
      qed
      ultimately AOT_have ‹□(F = [λy p₁] & (s' ⊨ p₁ ∨ s" ⊨ p₁))›
        by (metis "KBasic:3" "&I" "≡E"(2))
      AOT_hence ‹∃p □(F = [λy p] & (s' ⊨ p ∨ s" ⊨ p))› by (rule "∃I")
      AOT_thus ‹□∃p (F = [λy p] & (s' ⊨ p ∨ s" ⊨ p))›
        using Buridan[THEN "→E"] by fast
    }
  qed

  AOT_have desc_den: ‹ιs(∀F (s[F] ≡ ∃p (F = [λy p] & (s' ⊨ p ∨ s" ⊨ p))))↓›
    by (rule "can-sit-desc:1"[OF "cond-prop"])
  AOT_obtain x₀
    where x₀_prop1: ‹x₀ = ιs(∀F (s[F] ≡ ∃p (F = [λy p] & (s' ⊨ p ∨ s" ⊨ p))))›
    by (metis (no_types, lifting) "∃E" "rule=I:1" desc_den "∃I"(1) id_sym)
  AOT_hence x₀_sit: ‹Situation(x₀)›
    using "actual-desc:3"[THEN "→E"] "Act-Basic:2" "&E"(1) "≡E"(1)
          "possit-sit:4" by blast

  AOT_have 1: ‹∀F (x₀[F] ≡ ∃p (F = [λy p] & (s' ⊨ p ∨ s" ⊨ p)))›
    using "strict-sit"[OF rigid, OF "cond-prop", THEN "→E", OF x₀_prop1].
  AOT_have 2: ‹(x₀ ⊨ p) ≡ (s' ⊨ p ∨ s" ⊨ p)› for p
  proof (rule "≡I"; rule "→I")
    AOT_assume ‹x₀ ⊨ p›
    AOT_hence ‹x₀[λy p]› using lem1[THEN "→E", OF x₀_sit, THEN "≡E"(1)] by blast
    then AOT_obtain q where ‹[λy p] = [λy q] & (s' ⊨ q ∨ s" ⊨ q)›
      using 1[THEN "∀E"(1)[where τ="«[λy p]»"], OF "prop-prop2:2", THEN "≡E"(1)]
            "∃E"[rotated] by blast
    AOT_thus ‹s' ⊨ p ∨ s" ⊨ p›
      by (metis "rule=E" "&E"(1) "&E"(2) "∨I"(1) "∨I"(2)
                "∨E"(1) "deduction-theorem" id_sym "≡E"(2) "p-identity-thm2:3")
  next
    AOT_assume ‹s' ⊨ p ∨ s" ⊨ p›
    AOT_hence ‹[λy p] = [λy p] & (s' ⊨ p ∨ s" ⊨ p)›
      by (metis "rule=I:1" "&I" "prop-prop2:2")
    AOT_hence ‹∃q ([λy p] = [λy q] & (s' ⊨ q ∨ s" ⊨ q))›
      by (rule "∃I")
    AOT_hence ‹x₀[λy p]›
      using 1[THEN "∀E"(1), OF "prop-prop2:2", THEN "≡E"(2)] by blast
    AOT_thus ‹x₀ ⊨ p›
      by (metis "≡dfI" "&I" "ex:1:a" "prop-enc" "rule-ui:2[const_var]"
```

```
                x0_sit "true-in-s")
  qed

  AOT_have ‹Actual(x0) & s' ⊴ x0 & s" ⊴ x0›
  proof(safe intro!: "→I" "&I" "∃I"(1) actual[THEN "≡dfI"] x0_sit GEN
                     "sit-part-whole"[THEN "≡dfI"])
    fix p
    AOT_assume ‹x0 ⊨ p›
    AOT_hence ‹s' ⊨ p ∨ s" ⊨ p›
      using 2 "≡E"(1) by metis
    AOT_thus ‹p›
      using act_s' act_s"
            actual[THEN "≡dfE", THEN "&E"(2), THEN "∀E"(2), THEN "→E"]
      by (metis "∨E"(3) "reductio-aa:1")
  next
    AOT_show ‹x0 ⊨ p› if ‹s' ⊨ p› for p
      using 2[THEN "≡E"(2), OF "∨I"(1), OF that].
  next
    AOT_show ‹x0 ⊨ p› if ‹s" ⊨ p› for p
      using 2[THEN "≡E"(2), OF "∨I"(2), OF that].
  next
    AOT_show ‹Situation(s')›
      using act_s'[THEN actual[THEN "≡dfE"]] "&E" by blast
  next
    AOT_show ‹Situation(s")›
      using act_s"[THEN actual[THEN "≡dfE"]] "&E" by blast
  qed
  AOT_thus ‹∃x (Actual(x) & s' ⊴ x & s" ⊴ x)›
    by (rule "∃I")
qed

AOT_define Consistent :: ‹τ ⇒ φ› (‹Consistent'(_')›)
  cons: ‹Consistent(s) ≡df ¬∃p (s ⊨ p & s ⊨ ¬p)›                        (487)

AOT_theorem "sit-cons": ‹Actual(s) → Consistent(s)›                      (489)
proof(safe intro!: "→I" cons[THEN "≡dfI"] "&I" Situation.ψ
            dest!: actual[THEN "≡dfE"]; frule "&E"(1); drule "&E"(2))
  AOT_assume 0: ‹∀p (s ⊨ p → p)›
  AOT_show ‹¬∃p (s ⊨ p & s ⊨ ¬p)›
  proof (rule "raa-cor:2")
    AOT_assume ‹∃p (s ⊨ p & s ⊨ ¬p)›
    then AOT_obtain p where ‹s ⊨ p & s ⊨ ¬p›
      using "∃E"[rotated] by blast
    AOT_hence ‹p & ¬p›
      using 0[THEN "∀E"(1)[where τ=‹«¬p»›, THEN "→E"], OF "log-prop-prop:2"]
            0[THEN "∀E"(2)[where β=p], THEN "→E"] "&E" "&I" by blast
    AOT_thus ‹p & ¬p› for p by (metis "raa-cor:1")
  qed
qed

AOT_theorem "cons-rigid:1": ‹¬Consistent(s) ≡ □¬Consistent(s)›           (490.1)
proof (rule "≡I"; rule "→I")
  AOT_assume ‹¬Consistent(s)›
  AOT_hence ‹∃p (s ⊨ p & s ⊨ ¬p)›
    using cons[THEN "≡dfI", OF "&I", OF Situation.ψ]
    by (metis "raa-cor:3")
  then AOT_obtain p where p_prop: ‹s ⊨ p & s ⊨ ¬p›
    using "∃E"[rotated] by blast
  AOT_hence ‹□s ⊨ p›
    using "&E"(1) "≡E"(1) "lem2:1" by blast
  moreover AOT_have ‹□s ⊨ ¬p›
    using p_prop "T◊" "&E" "≡E"(1)
      "modus-tollens:1" "raa-cor:3" "lem2:3"[unvarify p]
```

```
      "log-prop-prop:2" by metis
  ultimately AOT_have ‹□(s ⊨ p & s ⊨ ¬p)›
    by (metis "KBasic:3" "&I" "≡E"(2))
  AOT_hence ‹∃p □(s ⊨ p & s ⊨ ¬p)›
    by (rule "∃I")
  AOT_hence ‹□∃p(s ⊨ p & s ⊨ ¬p)›
    by (metis Buridan "vdash-properties:10")
  AOT_thus ‹□¬Consistent(s)›
    apply (rule "qml:1"[axiom_inst, THEN "→E", THEN "→E", rotated])
    apply (rule RN)
    using "≡dfE" "&E"(2) cons "deduction-theorem" "raa-cor:3" by blast
next
  AOT_assume ‹□¬Consistent(s)›
  AOT_thus ‹¬Consistent(s)› using "qml:2"[axiom_inst, THEN "→E"] by auto
qed

AOT_theorem "cons-rigid:2": ‹◇Consistent(x) ≡ Consistent(x)›                (490.2)
proof(rule "≡I"; rule "→I")
  AOT_assume 0: ‹◇Consistent(x)›
  AOT_have ‹◇(Situation(x) & ¬∃p (x ⊨ p & x ⊨ ¬p))›
    apply (AOT_subst ‹Situation(x) & ¬∃p (x ⊨ p & x ⊨ ¬p)› ‹Consistent(x)›)
     using cons "≡E"(2) "Commutativity of ≡" "≡Df" apply blast
    by (simp add: 0)
  AOT_hence ‹◇Situation(x)› and 1: ‹◇¬∃p (x ⊨ p & x ⊨ ¬p)›
    using "RM◇" "Conjunction Simplification"(1) "Conjunction Simplification"(2)
          "modus-tollens:1" "raa-cor:3" by blast+
  AOT_hence 2: ‹Situation(x)› by (metis "≡E"(1) "possit-sit:2")
  AOT_have 3: ‹¬□∃p (x ⊨ p & x ⊨ ¬p)›
    using 2 using 1 "KBasic:11" "≡E"(2) by blast
  AOT_show ‹Consistent(x)›
  proof (rule "raa-cor:1")
    AOT_assume ‹¬Consistent(x)›
    AOT_hence ‹∃p (x ⊨ p & x ⊨ ¬p)›
      using 0 "≡dfE" "conventions:5" 2 "cons-rigid:1"[unconstrain s, THEN "→E"]
            "modus-tollens:1" "raa-cor:3" "≡E"(4) by meson
    then AOT_obtain p where ‹x ⊨ p› and 4: ‹x ⊨ ¬p›
      using "∃E"[rotated] "&E" by blast
    AOT_hence ‹□x ⊨ p›
      by (metis "2" "≡E"(1) "lem2:1"[unconstrain s, THEN "→E"])
    moreover AOT_have ‹□x ⊨ ¬p›
      using 4 "lem2:1"[unconstrain s, unvarify p, THEN "→E"]
      by (metis 2 "≡E"(1) "log-prop-prop:2")
    ultimately AOT_have ‹□(x ⊨ p & x ⊨ ¬p)›
      by (metis "KBasic:3" "&I" "≡E"(3) "raa-cor:3")
    AOT_hence ‹∃p □(x ⊨ p & x ⊨ ¬p)›
      by (metis "existential:1" "log-prop-prop:2")
    AOT_hence ‹□∃p (x ⊨ p & x ⊨ ¬p)›
      by (metis Buridan "vdash-properties:10")
    AOT_thus ‹p & ¬p› for p
      using 3 "&I"  by (metis "raa-cor:3")
  qed
next
  AOT_show ‹◇Consistent(x)› if ‹Consistent(x)›
    using "T◇" that "vdash-properties:10" by blast
qed

AOT_define possible :: ‹τ ⇒ φ› (‹Possible'(_')›)
  pos: ‹Possible(s) ≡df ◇Actual(s)›                                         (491)

AOT_theorem "sit-pos:1": ‹Actual(s) → Possible(s)›                         (492.1)
  apply(rule "→I"; rule pos[THEN "≡dfI"]; rule "&I")
  apply (meson "≡dfE" actual "&E"(1))
  using "T◇" "vdash-properties:10" by blast
```

```
AOT_theorem "sit-pos:2": ‹∃p ((s ⊨ p) & ¬◊p) → ¬Possible(s)›                                (492.2)
proof(rule "→I")
  AOT_assume ‹∃p ((s ⊨ p) & ¬◊p)›
  then AOT_obtain p where a: ‹(s ⊨ p) & ¬◊p›
    using "∃E"[rotated] by blast
  AOT_hence ‹□(s ⊨ p)›
    using "&E" by (metis "T◊" "≡E"(1) "lem2:3" "vdash-properties:10")
  moreover AOT_have ‹□¬p›
    using a[THEN "&E"(2)] by (metis "KBasic2:1" "≡E"(2))
  ultimately AOT_have ‹□(s ⊨ p & ¬p)›
    by (metis "KBasic:3" "&I" "≡E"(3) "raa-cor:3")
  AOT_hence ‹∃p □(s ⊨ p & ¬p)›
    by (rule "∃I")
  AOT_hence 1: ‹□∃q (s ⊨ q & ¬q)›
    by (metis Buridan "vdash-properties:10")
  AOT_have ‹□¬∀q (s ⊨ q → q)›
    apply (AOT_subst ‹s ⊨ q → q› ‹¬(s ⊨ q & ¬q)› for: q)
     apply (simp add: "oth-class-taut:1:a")
    apply (AOT_subst ‹¬∀q ¬(s ⊨ q & ¬q)› ‹∃q (s ⊨ q & ¬q)›)
    by (auto simp: "conventions:4" "df-rules-formulas[3]" "df-rules-formulas[4]" "≡I" 1)
  AOT_hence 0: ‹¬◊∀q (s ⊨ q → q)›
    by (metis "≡dfE" "conventions:5" "raa-cor:3")
  AOT_show ‹¬Possible(s)›
    apply (AOT_subst ‹Possible(s)› ‹Situation(s) & ◊Actual(s)›)
     apply (simp add: pos "≡Df")
    apply (AOT_subst ‹Actual(s)› ‹Situation(s) & ∀q (s ⊨ q → q)›)
     using actual "≡Df" apply presburger
    by (metis "0" "KBasic2:3" "&E"(2) "raa-cor:3" "vdash-properties:10")
qed

AOT_theorem "pos-cons-sit:1": ‹Possible(s) → Consistent(s)›                                (493.1)
  by (auto simp: "sit-cons"[THEN "RM◊", THEN "→E",
                            THEN "cons-rigid:2"[THEN "≡E"(1)]]
           intro!: "→I" dest!: pos[THEN "≡dfE"] "&E"(2))

AOT_theorem "pos-cons-sit:2": ‹∃s (Consistent(s) & ¬Possible(s))›                          (493.2)
proof -
  AOT_obtain q1 where ‹q1 & ◊¬q1›
    using "≡dfE" "instantiation" "cont-tf:1" "cont-tf-thm:1" by blast
  have "cond-prop": ‹ConditionOnPropositionalProperties                                      (474)
                      (λ Π . «Π = [λy q1 & ¬q1]»)›
    by (auto intro!: "cond-prop[I]" GEN "→I" "prop-prop1"[THEN "≡dfI"]
                     "∃I"(1)[where τ=‹«q1 & ¬q1»›, rotated, OF "log-prop-prop:2"])
  have rigid: ‹rigid_condition (λ Π . «Π = [λy q1 & ¬q1]»)›
    by (auto intro!: "strict-can:1[I]" GEN "→I" simp: "id-nec:2"[THEN "→E"])

  AOT_obtain x where x_prop: ‹x = ιs (∀F (s[F] ≡ F = [λy q1 & ¬q1]))›
    using "ex:1:b"[THEN "∀E"(1), OF "can-sit-desc:1", OF "cond-prop"]
          "∃E"[rotated] by blast
  AOT_hence 0: ‹𝒜(Situation(x) & ∀F (x[F] ≡ F = [λy q1 & ¬q1]))›
    using "→E" "actual-desc:2" by blast
  AOT_hence ‹𝒜(Situation(x))› by (metis "Act-Basic:2" "&E"(1) "≡E"(1))
  AOT_hence s_sit: ‹Situation(x)› by (metis "≡E"(1) "possit-sit:4")
  AOT_have s_enc_prop: ‹∀F (x[F] ≡ F = [λy q1 & ¬q1])›
    using "strict-sit"[OF rigid, OF "cond-prop", THEN "→E", OF x_prop].
  AOT_hence ‹x[λy q1 & ¬q1]›
    using "∀E"(1)[rotated, OF "prop-prop2:2"]
          "rule=I:1"[OF "prop-prop2:2"] "≡E" by blast
  AOT_hence ‹x ⊨ (q1 & ¬q1)›
    using lem1[THEN "→E", OF s_sit, unvarify p, THEN "≡E"(2), OF "log-prop-prop:2"]
    by blast
  AOT_hence ‹□(x ⊨ (q1 & ¬q1))›
```

```
    using "lem2:1"[unconstrain s, THEN "→E", OF s_sit, unvarify p,
                   OF "log-prop-prop:2", THEN "≡E"(1)] by blast
  moreover AOT_have ‹□(x ⊨ (q1 & ¬q1) → ¬Actual(x))›
  proof(rule RN; rule "→I"; rule "raa-cor:2")
    AOT_modally_strict {
      AOT_assume ‹Actual(x)›
      AOT_hence ‹∀p (x ⊨ p → p)›
        using actual[THEN "≡dfE", THEN "&E"(2)] by blast
      moreover AOT_assume ‹x ⊨ (q1 & ¬q1)›
      ultimately AOT_show ‹q1 & ¬q1›
        using "∀E"(1)[rotated, OF "log-prop-prop:2"] "→E" by metis
    }
  qed
  ultimately AOT_have nec_not_actual_s: ‹□¬Actual(x)›
    using "qml:1"[axiom_inst, THEN "→E", THEN "→E"] by blast
  AOT_have 1: ‹¬∃p (x ⊨ p & x ⊨ ¬p)›
  proof (rule "raa-cor:2")
    AOT_assume ‹∃p (x ⊨ p & x ⊨ ¬p)›
    then AOT_obtain p where ‹x ⊨ p & x ⊨ ¬p›
      using "∃E"[rotated] by blast
    AOT_hence ‹x[λy p] & x[λy ¬p]›
      using lem1[unvarify p, THEN "→E", OF "log-prop-prop:2",
                 OF s_sit, THEN "≡E"(1)] "&I" "&E" by metis
    AOT_hence ‹[λy p] = [λy q1 & ¬q1]› and ‹[λy ¬p] = [λy q1 & ¬q1]›
      by (auto intro!: "prop-prop2:2" s_enc_prop[THEN "∀E"(1), THEN "≡E"(1)]
               elim: "&E")
    AOT_hence i: ‹[λy p] = [λy ¬p]› by (metis "rule=E" id_sym)
    {
      AOT_assume 0: ‹p›
      AOT_have ‹[λy p]x› for x
        by (auto intro!: "β←C"(1) "cqt:2" 0)
      AOT_hence ‹[λy ¬p]x› for x using i "rule=E" by fast
      AOT_hence ‹¬p›
        using "β→C"(1) by auto
    }
    moreover {
      AOT_assume 0: ‹¬p›
      AOT_have ‹[λy ¬p]x› for x
        by (auto intro!: "β←C"(1) "cqt:2" 0)
      AOT_hence ‹[λy p]x› for x using i[symmetric] "rule=E" by fast
      AOT_hence ‹p›
        using "β→C"(1) by auto
    }
    ultimately AOT_show ‹p & ¬p› for p by (metis "raa-cor:1" "raa-cor:3")
  qed
  AOT_have 2: ‹¬Possible(x)›
  proof(rule "raa-cor:2")
    AOT_assume ‹Possible(x)›
    AOT_hence ‹◇Actual(x)›
      by (metis "≡dfE" "&E"(2) pos)
    moreover AOT_have ‹¬◇Actual(x)› using nec_not_actual_s
      using "≡dfE" "conventions:5" "reductio-aa:2" by blast
    ultimately AOT_show ‹◇Actual(x) & ¬◇Actual(x)› by (rule "&I")
  qed
  show ?thesis
    by(rule "∃I"(2)[where β=x]; safe intro!: "&I" 2 s_sit cons[THEN "≡dfI"] 1)
qed

AOT_theorem "sit-classical:1": ‹∀p (s ⊨ p ≡ p) → ∀q(s ⊨ ¬q ≡ ¬s ⊨ q)›          (494.1)
proof(rule "→I"; rule GEN)
  fix q
  AOT_assume ‹∀p (s ⊨ p ≡ p)›
  AOT_hence ‹s ⊨ q ≡ q› and ‹s ⊨ ¬q ≡ ¬q›
```

```
    using "∀E"(1)[rotated, OF "log-prop-prop:2"] by blast+
  AOT_thus ‹s ⊨ ¬q ≡ ¬s ⊨ q›
    by (metis "deduction-theorem" "≡I" "≡E"(1) "≡E"(2) "≡E"(4))
qed

AOT_theorem "sit-classical:2":                                    (494.2)
  ‹∀p (s ⊨ p ≡ p) → ∀q∀r((s ⊨ (q → r)) ≡ (s ⊨ q → s ⊨ r))›
proof(rule "→I"; rule GEN; rule GEN)
  fix q r
  AOT_assume ‹∀p (s ⊨ p ≡ p)›
  AOT_hence ϑ: ‹s ⊨ q ≡ q› and ξ: ‹s ⊨ r ≡ r› and ζ: ‹(s ⊨ (q → r)) ≡ (q → r)›
    using "∀E"(1)[rotated, OF "log-prop-prop:2"] by blast+
  AOT_show ‹(s ⊨ (q → r)) ≡ (s ⊨ q → s ⊨ r)›
  proof (safe intro!: "≡I" "→I")
    AOT_assume ‹s ⊨ (q → r)›
    moreover AOT_assume ‹s ⊨ q›
    ultimately AOT_show ‹s ⊨ r›
      using ϑ ξ ζ by (metis "≡E"(1) "≡E"(2) "vdash-properties:10")
  next
    AOT_assume ‹s ⊨ q → s ⊨ r›
    AOT_thus ‹s ⊨ (q → r)›
      using ϑ ξ ζ by (metis "deduction-theorem" "≡E"(1) "≡E"(2) "→E")
  qed
qed

AOT_theorem "sit-classical:3":                                    (494.3)
  ‹∀p (s ⊨ p ≡ p) → ((s ⊨ ∀α φ{α}) ≡ ∀α s ⊨ φ{α})›
proof (rule "→I")
  AOT_assume ‹∀p (s ⊨ p ≡ p)›
  AOT_hence ϑ: ‹s ⊨ φ{α} ≡ φ{α}› and ξ: ‹s ⊨ ∀α φ{α} ≡ ∀α φ{α}› for α
    using "∀E"(1)[rotated, OF "log-prop-prop:2"] by blast+
  AOT_show ‹s ⊨ ∀α φ{α} ≡ ∀α s ⊨ φ{α}›
  proof (safe intro!: "≡I" "→I" GEN)
    fix α
    AOT_assume ‹s ⊨ ∀α φ{α}›
    AOT_hence ‹φ{α}› using ξ "∀E"(2) "≡E"(1) by blast
    AOT_thus ‹s ⊨ φ{α}› using ϑ "≡E"(2) by blast
  next
    AOT_assume ‹∀α s ⊨ φ{α}›
    AOT_hence ‹s ⊨ φ{α}› for α using "∀E"(2) by blast
    AOT_hence ‹φ{α}› for α using ϑ "≡E"(1) by blast
    AOT_hence ‹∀α φ{α}› by (rule GEN)
    AOT_thus ‹s ⊨ ∀α φ{α}› using ξ "≡E"(2) by blast
  qed
qed

AOT_theorem "sit-classical:4": ‹∀p (s ⊨ p ≡ p) → ∀q (s ⊨ □q → □s ⊨ q)›     (494.4)
proof(rule "→I"; rule GEN; rule "→I")
  fix q
  AOT_assume ‹∀p (s ⊨ p ≡ p)›
  AOT_hence ϑ: ‹s ⊨ q ≡ q› and ξ: ‹s ⊨ □q ≡ □q›
    using "∀E"(1)[rotated, OF "log-prop-prop:2"] by blast+
  AOT_assume ‹s ⊨ □q›
  AOT_hence ‹□q› using ξ "≡E"(1) by blast
  AOT_hence ‹q› using "qml:2"[axiom_inst, THEN "→E"] by blast
  AOT_hence ‹s ⊨ q› using ϑ "≡E"(2) by blast
  AOT_thus ‹□s ⊨ q› using "≡dfE" "&E"(1) "≡E"(1) "lem2:1" "true-in-s" by blast
qed

AOT_theorem "sit-classical:5":                                    (494.5)
  ‹∀p (s ⊨ p ≡ p) → ∃q(□(s ⊨ q) & ¬(s ⊨ □ q))›
proof (rule "→I")
  AOT_obtain r where A: ‹r› and ‹◇¬r›
```

```
    by (metis "&E"(1) "&E"(2) "≡dfE" "instantiation" "cont-tf:1" "cont-tf-thm:1")
  AOT_hence B: ‹¬□r›
    using "KBasic:11" "≡E"(2) by blast
  moreover AOT_assume asm: ‹∀ p (s ⊨ p ≡ p)›
  AOT_hence ‹s ⊨ r›
    using "∀E"(2) A "≡E"(2) by blast
  AOT_hence 1: ‹□s ⊨ r›
    using "≡dfE" "&E"(1) "≡E"(1) "lem2:1" "true-in-s" by blast
  AOT_have ‹s ⊨ ¬□r›
    using asm[THEN "∀E"(1)[rotated, OF "log-prop-prop:2"], THEN "≡E"(2)] B by blast
  AOT_hence ‹¬s ⊨ □r›
    using "sit-classical:1"[THEN "→E", OF asm,
              THEN "∀E"(1)[rotated, OF "log-prop-prop:2"], THEN "≡E"(1)] by blast
  AOT_hence ‹□s ⊨ r & ¬s ⊨ □r›
    using 1 "&I" by blast
  AOT_thus ‹∃r (□s ⊨ r & ¬s ⊨ □r)›
    by (rule "∃I")
qed

AOT_theorem "sit-classical:6":                                                    (494.6)
  ‹∃s ∀p (s ⊨ p ≡ p)›
proof -
  have "cond-prop": ‹ConditionOnPropositionalProperties                             (474)
                        (λ Π . «∃q (q & Π = [λy q])»)›
  proof (safe intro!: "cond-prop[I]" GEN "→I")
    fix F
    AOT_modally_strict {
      AOT_assume ‹∃q (q & F = [λy q])›
      then AOT_obtain q where ‹q & F = [λy q]›
        using "∃E"[rotated] by blast
      AOT_hence ‹F = [λy q]›
        using "&E" by blast
      AOT_hence ‹∃q F = [λy q]›
        by (rule "∃I")
      AOT_thus ‹Propositional([F])›
        by (metis "≡dfI" "prop-prop1")
    }
  qed
  AOT_have ‹∃s ∀F (s[F] ≡ ∃q (q & F = [λy q]))›
    using "comp-sit:1"[OF "cond-prop"].
  then AOT_obtain s₀ where s₀_prop: ‹∀F (s₀[F] ≡ ∃q (q & F = [λy q]))›
    using "Situation.∃E"[rotated] by meson
  AOT_have ‹∀p (s₀ ⊨ p ≡ p)›
  proof(safe intro!: GEN "≡I" "→I")
    fix p
    AOT_assume ‹s₀ ⊨ p›
    AOT_hence ‹s₀[λy p]›
      using lem1[THEN "→E", OF Situation.ψ, THEN "≡E"(1)] by blast
    AOT_hence ‹∃q (q & [λy p] = [λy q])›
      using s₀_prop[THEN "∀E"(1)[rotated, OF "prop-prop2:2"], THEN "≡E"(1)] by blast
    then AOT_obtain q₁ where q₁_prop: ‹q₁ & [λy p] = [λy q₁]›
      using "∃E"[rotated] by blast
    AOT_hence ‹p = q₁›
      by (metis "&E"(2) "≡E"(2) "p-identity-thm2:3")
    AOT_thus ‹p›
      using q₁_prop[THEN "&E"(1)] "rule=E" id_sym by fast
  next
    fix p
    AOT_assume ‹p›
    moreover AOT_have ‹[λy p] = [λy p]›
      by (simp add: "rule=I:1"[OF "prop-prop2:2"])
    ultimately AOT_have ‹p & [λy p] = [λy p]›
      using "&I" by blast
```

```
    AOT_hence ‹∃q (q & [λy p] = [λy q])›
      by (rule "∃I")
    AOT_hence ‹s0[λy p]›
      using s0_prop[THEN "∀E"(1)[rotated, OF "prop-prop2:2"], THEN "≡E"(2)] by blast
    AOT_thus ‹s0 ⊨ p›
      using lem1[THEN "→E", OF Situation.ψ, THEN "≡E"(2)] by blast
  qed
  AOT_hence ‹∀p (s0 ⊨ p ≡ p)›
    using "&I" by blast
  AOT_thus ‹∃s ∀p (s ⊨ p ≡ p)›
    by (rule "Situation.∃I")
qed

AOT_define PossibleWorld :: ‹τ ⇒ φ› (‹PossibleWorld'(_')›)
  "world:1": ‹PossibleWorld(x) ≡df Situation(x) & ◊∀p(x ⊨ p ≡ p)›          (496.1)

AOT_theorem "world:2": ‹∃x PossibleWorld(x)›                               (496.2)
proof -
  AOT_obtain s where s_prop: ‹∀p (s ⊨ p ≡ p)›
    using "sit-classical:6" "Situation.∃E"[rotated] by meson
  AOT_have ‹∀p (s ⊨ p ≡ p)›
  proof(safe intro!: GEN "≡I" "→I")
    fix p
    AOT_assume ‹s ⊨ p›
    AOT_thus ‹p›
      using s_prop[THEN "∀E"(2), THEN "≡E"(1)] by blast
  next
    fix p
    AOT_assume ‹p›
    AOT_thus ‹s ⊨ p›
      using s_prop[THEN "∀E"(2), THEN "≡E"(2)] by blast
  qed
  AOT_hence ‹◊∀p (s ⊨ p ≡ p)›
    by (metis "T◊"[THEN "→E"])
  AOT_hence ‹◊∀p (s ⊨ p ≡ p)›
    using s_prop "&I" by blast
  AOT_hence ‹PossibleWorld(s)›
    using "world:1"[THEN "≡dfI"] Situation.ψ "&I" by blast
  AOT_thus ‹∃x PossibleWorld(x)›
    by (rule "∃I")
qed

AOT_theorem "world:3": ‹PossibleWorld(κ) → κ↓›                            (496.3)
proof (rule "→I")
  AOT_assume ‹PossibleWorld(κ)›
  AOT_hence ‹Situation(κ)›
    using "world:1"[THEN "≡dfE"] "&E" by blast
  AOT_hence ‹A!κ›
    by (metis "≡dfE" "&E"(1) situations)
  AOT_thus ‹κ↓›
    by (metis "russell-axiom[exe,1].ψ_denotes_asm")
qed

AOT_theorem "rigid-pw:1": ‹PossibleWorld(x) ≡ □PossibleWorld(x)›           (497.1)
proof(safe intro!: "≡I" "→I")
  AOT_assume ‹PossibleWorld(x)›
  AOT_hence ‹Situation(x) & ◊∀p(x ⊨ p ≡ p)›
    using "world:1"[THEN "≡dfE"] by blast
  AOT_hence ‹□Situation(x) & □◊∀p(x ⊨ p ≡ p)›
    by (metis "S5Basic:1" "&I" "&E"(1) "&E"(2) "≡E"(1) "possit-sit:1")
  AOT_hence 0: ‹□(Situation(x) & ◊∀p(x ⊨ p ≡ p))›
    by (metis "KBasic:3" "≡E"(2))
  AOT_show ‹□PossibleWorld(x)›
```

```
    by (AOT_subst ‹PossibleWorld(x)› ‹Situation(x) & ◇∀p(x ⊨ p ≡ p)›)
       (auto simp: "≡Df" "world:1" 0)
next
  AOT_show ‹PossibleWorld(x)› if ‹□PossibleWorld(x)›
    using that "qml:2"[axiom_inst, THEN "→E"] by blast
qed

AOT_theorem "rigid-pw:2": ‹◇PossibleWorld(x) ≡ PossibleWorld(x)›        (497.2)
  using "rigid-pw:1"
  by (meson "RE◇" "S5Basic:2" "≡E"(2) "≡E"(6) "Commutativity of ≡")

AOT_theorem "rigid-pw:3": ‹◇PossibleWorld(x) ≡ □PossibleWorld(x)›       (497.3)
  using "rigid-pw:1" "rigid-pw:2" by (meson "≡E"(5))

AOT_theorem "rigid-pw:4": ‹𝒜PossibleWorld(x) ≡ PossibleWorld(x)›        (497.4)
  by (metis "Act-Sub:3" "→I" "≡I" "≡E"(6) "nec-imp-act" "rigid-pw:1" "rigid-pw:2")

AOT_register_rigid_restricted_type
  PossibleWorld: ‹PossibleWorld(κ)›
proof
  AOT_modally_strict {
    AOT_show ‹∃x PossibleWorld(x)› using "world:2".
  }
next
  AOT_modally_strict {
    AOT_show ‹PossibleWorld(κ) → κ↓› for κ using "world:3".
  }
next
  AOT_modally_strict {
    AOT_show ‹∀α(PossibleWorld(α) → □PossibleWorld(α))›
      by (meson GEN "→I" "≡E"(1) "rigid-pw:1")
  }
qed
AOT_register_variable_names
  PossibleWorld: w

AOT_theorem "world-pos": ‹Possible(w)›                                    (500)
proof (safe intro!: "≡dfE"[OF "world:1", OF PossibleWorld.ψ, THEN "&E"(1)]
                    pos[THEN "≡dfI"] "&I" )
  AOT_have ‹◇∀p (w ⊨ p ≡ p)›
    using "world:1"[THEN "≡dfE", OF PossibleWorld.ψ, THEN "&E"(2)].
  AOT_hence ‹◇∀p (w ⊨ p → p)›
  proof (rule "RM◇"[THEN "→E", rotated]; safe intro!: "→I" GEN)
    AOT_modally_strict {
      fix p
      AOT_assume ‹∀p (w ⊨ p ≡ p)›
      AOT_hence ‹w ⊨ p ≡ p› using "∀E"(2) by blast
      moreover AOT_assume ‹w ⊨ p›
      ultimately AOT_show p using "≡E"(1) by blast
    }
  qed
  AOT_hence 0: ‹◇(Situation(w) & ∀p (w ⊨ p → p))›
    using "world:1"[THEN "≡dfE", OF PossibleWorld.ψ, THEN "&E"(1),
                    THEN "possit-sit:1"[THEN "≡E"(1)]]
    by (metis "KBasic:16" "&I" "vdash-properties:10")
  AOT_show ‹◇Actual(w)›
    by (AOT_subst ‹Actual(w)› ‹Situation(w) & ∀p (w ⊨ p → p)›)
       (auto simp: actual "≡Df" 0)
qed

AOT_theorem "world-cons:1": ‹Consistent(w)›                              (501.1)
  using "world-pos"
  using "pos-cons-sit:1"[unconstrain s, THEN "→E", THEN "→E"]
```

```
  by (meson "≡dfE" "&E"(1) pos)

AOT_theorem "world-cons:2": ‹¬TrivialSituation(w)›                    (501.2)
proof(rule "raa-cor:2")
  AOT_assume ‹TrivialSituation(w)›
  AOT_hence ‹Situation(w) & ∀p w ⊨ p›
    using "df-null-trivial:2"[THEN "≡dfE"] by blast
  AOT_hence 0: ‹□w ⊨ (∃p (p & ¬p))›
    using "&E"
    by (metis "Buridan◇" "T◇" "&E"(2) "≡E"(1) "lem2:3"[unconstrain s, THEN "→E"]
              "log-prop-prop:2" "rule-ui:1" "universal-cor" "→E")
  AOT_have ‹◇∀p (w ⊨ p ≡ p)›
    using PossibleWorld.ψ "world:1"[THEN "≡dfE", THEN "&E"(2)] by metis
  AOT_hence ‹∀p ◇(w ⊨ p ≡ p)›
    using "Buridan◇"[THEN "→E"] by blast
  AOT_hence ‹◇(w ⊨ (∃p (p & ¬p)) ≡ (∃p (p & ¬p)))›
    by (metis "log-prop-prop:2" "rule-ui:1")
  AOT_hence ‹◇(w ⊨ (∃p (p & ¬p)) → (∃p (p & ¬p)))›
    using "RM◇"[THEN "→E"] "→I" "≡E"(1) by meson
  AOT_hence ‹◇(∃p (p & ¬p))› using 0
    by (metis "KBasic2:4" "≡E"(1) "→E")
  moreover AOT_have ‹¬◇(∃p (p & ¬p))›
    by (metis "instantiation" "KBasic2:1" RN "≡E"(1) "raa-cor:2")
  ultimately AOT_show ‹◇(∃p (p & ¬p)) & ¬◇(∃p (p & ¬p))›
    using "&I" by blast
qed

AOT_theorem "rigid-truth-at:1": ‹w ⊨ p ≡ □w ⊨ p›                      (502.1)
  using "lem2:1"[unconstrain s, THEN "→E",
                 OF PossibleWorld.ψ[THEN "world:1"[THEN "≡dfE"], THEN "&E"(1)]].

AOT_theorem "rigid-truth-at:2": ‹◇w ⊨ p ≡ w ⊨ p›                      (502.2)
  using "lem2:2"[unconstrain s, THEN "→E",
                 OF PossibleWorld.ψ[THEN "world:1"[THEN "≡dfE"], THEN "&E"(1)]].

AOT_theorem "rigid-truth-at:3": ‹◇w ⊨ p ≡ □w ⊨ p›                     (502.3)
  using "lem2:3"[unconstrain s, THEN "→E",
                 OF PossibleWorld.ψ[THEN "world:1"[THEN "≡dfE"], THEN "&E"(1)]].

AOT_theorem "rigid-truth-at:4": ‹𝒜w ⊨ p ≡ w ⊨ p›                      (502.4)
  using "lem2:4"[unconstrain s, THEN "→E",
                 OF PossibleWorld.ψ[THEN "world:1"[THEN "≡dfE"], THEN "&E"(1)]].

AOT_theorem "rigid-truth-at:5": ‹¬w ⊨ p ≡ □¬w ⊨ p›                    (502.5)
  using "lem2:5"[unconstrain s, THEN "→E",
                 OF PossibleWorld.ψ[THEN "world:1"[THEN "≡dfE"], THEN "&E"(1)]].

AOT_define Maximal :: ‹τ ⇒ φ› (‹Maximal'(_')›)
  max: ‹Maximal(s) ≡df ∀p (s ⊨ p ∨ s ⊨ ¬p)›                           (503)

AOT_theorem "world-max": ‹Maximal(w)›                                   (504)
proof(safe intro!: PossibleWorld.ψ[THEN "≡dfE"[OF "world:1"], THEN "&E"(1)]
                   GEN "≡dfI"[OF max] "&I" )
  fix q
  AOT_have ‹◇(w ⊨ q ∨ w ⊨ ¬q)›
  proof(rule "RM◇"[THEN "→E"]; (rule "→I")?)
    AOT_modally_strict {
      AOT_assume ‹∀p (w ⊨ p ≡ p)›
      AOT_hence ‹w ⊨ q ≡ q› and ‹w ⊨ ¬q ≡ ¬q›
        using "∀E"(1)[rotated, OF "log-prop-prop:2"] by blast+
      AOT_thus ‹w ⊨ q ∨ w ⊨ ¬q›
        by (metis "∨I"(1) "∨I"(2) "≡E"(3) "reductio-aa:1")
    }
```

```
  next
    AOT_show ‹◇∀p (w ⊨ p ≡ p)›
      using PossibleWorld.ψ[THEN "≡dfE"[OF "world:1"], THEN "&E"(2)].
  qed
  AOT_hence ‹◇w ⊨ q ∨ ◇w ⊨ ¬q›
    using "KBasic2:2"[THEN "≡E"(1)] by blast
  AOT_thus ‹w ⊨ q ∨ w ⊨ ¬q›
    using "lem2:2"[unconstrain s, THEN "→E", unvarify p,
                    OF PossibleWorld.ψ[THEN "≡dfE"[OF "world:1"], THEN "&E"(1)],
                    THEN "≡E"(1), OF "log-prop-prop:2"]
    by (metis "∨I"(1) "∨I"(2) "∨E"(3) "raa-cor:2")
qed

AOT_theorem "world=maxpos:1": ‹Maximal(x) → □Maximal(x)›                    (505.1)
proof (AOT_subst ‹Maximal(x)› ‹Situation(x) & ∀p (x ⊨ p ∨ x ⊨ ¬p)›;
       safe intro!: max "≡Df" "→I"; frule "&E"(1); drule "&E"(2))
  AOT_assume sit_x: ‹Situation(x)›
  AOT_hence nec_sit_x: ‹□Situation(x)›
    by (metis "≡E"(1) "possit-sit:1")
  AOT_assume ‹∀p (x ⊨ p ∨ x ⊨ ¬p)›
  AOT_hence ‹x ⊨ p ∨ x ⊨ ¬p› for p
    using "∀E"(1)[rotated, OF "log-prop-prop:2"] by blast
  AOT_hence ‹□x ⊨ p ∨ □x ⊨ ¬p› for p
    using "lem2:1"[unconstrain s, THEN "→E", OF sit_x, unvarify p,
                    OF "log-prop-prop:2", THEN "≡E"(1)]
    by (metis "∨I"(1) "∨I"(2) "∨E"(2) "raa-cor:1")
  AOT_hence ‹□(x ⊨ p ∨ x ⊨ ¬p)› for p
    by (metis "KBasic:15" "→E")
  AOT_hence ‹∀p □(x ⊨ p ∨ x ⊨ ¬p)›
    by (rule GEN)
  AOT_hence ‹□∀p (x ⊨ p ∨ x ⊨ ¬p)›
    by (rule BF[THEN "→E"])
  AOT_thus ‹□(Situation(x) & ∀p (x ⊨ p ∨ x ⊨ ¬p))›
    using nec_sit_x by (metis "KBasic:3" "&I" "≡E"(2))
qed

AOT_theorem "world=maxpos:2": ‹PossibleWorld(x) ≡ Maximal(x) & Possible(x)›   (505.2)
proof(safe intro!: "≡I" "→I" "&I" "world-pos"[unconstrain w, THEN "→E"]
                   "world-max"[unconstrain w, THEN "→E"];
      frule "&E"(2); drule "&E"(1))
  AOT_assume pos_x: ‹Possible(x)›
  AOT_have ‹◇(Situation(x) & ∀p(x ⊨ p → p))›
    apply (AOT_subst (reverse) ‹Situation(x) & ∀p(x ⊨ p → p)› ‹Actual(x)›)
     using actual "≡Df" apply presburger
    using "≡dfE" "&E"(2) pos pos_x by blast
  AOT_hence 0: ‹◇∀p(x ⊨ p → p)›
    by (metis "KBasic2:3" "&E"(2) "vdash-properties:6")
  AOT_assume max_x: ‹Maximal(x)›
  AOT_hence sit_x: ‹Situation(x)› by (metis "≡dfE" max_x "&E"(1) max)
  AOT_have ‹□Maximal(x)› using "world=maxpos:1"[THEN "→E", OF max_x] by simp
  moreover AOT_have ‹□Maximal(x) → □(∀p(x ⊨ p → p) → ∀p (x ⊨ p ≡ p))›
  proof(safe intro!: "→I" RM GEN)
    AOT_modally_strict {
      fix p
      AOT_assume 0: ‹Maximal(x)›
      AOT_assume 1: ‹∀p (x ⊨ p → p)›
      AOT_show ‹x ⊨ p ≡ p›
      proof(safe intro!: "≡I" "→I" 1[THEN "∀E"(2), THEN "→E"]; rule "raa-cor:1")
        AOT_assume ‹¬x ⊨ p›
        AOT_hence ‹x ⊨ ¬p›
          using 0[THEN "≡dfE"[OF max], THEN "&E"(2), THEN "∀E"(2)]
                1 by (metis "∨E"(2))
        AOT_hence ‹¬p›
```

```
          using 1[THEN "∀E"(1), OF "log-prop-prop:2", THEN "→E"] by blast
        moreover AOT_assume p
        ultimately AOT_show ‹p & ¬p› using "&I" by blast
      qed
    }
  qed
  ultimately AOT_have ‹□(∀p(x ⊨ p → p) → ∀p (x ⊨ p ≡ p))›
    using "→E" by blast
  AOT_hence ‹◇∀p(x ⊨ p → p) → ◇∀p(x ⊨ p ≡ p)›
    by (metis "KBasic:13"[THEN "→E"])
  AOT_hence ‹◇∀p(x ⊨ p ≡ p)›
    using 0 "→E" by blast
  AOT_thus ‹PossibleWorld(x)›
    using "≡dfI"[OF "world:1", OF "&I", OF sit_x] by blast
qed

AOT_define NecImpl :: ‹φ ⇒ φ ⇒ φ› (infixl ‹⥤› 26)
  "nec-impl-p:1": ‹p ⥤ q ≡df □(p → q)›                                    (507.1)
AOT_define NecEquiv :: ‹φ ⇒ φ ⇒ φ› (infixl ‹⤄› 21)
  "nec-impl-p:2": ‹p ⤄ q ≡df (p ⥤ q) & (q ⥤ p)›                            (507.2)

AOT_theorem "nec-equiv-nec-im": ‹p ⤄ q ≡ □(p ≡ q)›                          (508)
proof(safe intro!: "≡I" "→I")
  AOT_assume ‹p ⤄ q›
  AOT_hence ‹(p ⥤ q)› and ‹(q ⥤ p)›
    using "nec-impl-p:2"[THEN "≡dfE"] "&E" by blast+
  AOT_hence ‹□(p → q)› and ‹□(q → p)›
    using "nec-impl-p:1"[THEN "≡dfE"] by blast+
  AOT_thus ‹□(p ≡ q)› by (metis "KBasic:4" "&I" "≡E"(2))
next
  AOT_assume ‹□(p ≡ q)›
  AOT_hence ‹□(p → q)› and ‹□(q → p)›
    using "KBasic:4" "&E" "≡E"(1) by blast+
  AOT_hence ‹(p ⥤ q)› and ‹(q ⥤ p)›
    using "nec-impl-p:1"[THEN "≡dfI"] by blast+
  AOT_thus ‹p ⤄ q›
    using "nec-impl-p:2"[THEN "≡dfI"] "&I" by blast
qed

(* TODO: PLM: discuss these; still not in PLM *)
AOT_theorem world_closed_lem_1_a:
  ‹(s ⊨ (φ & ψ)) → (∀p (s ⊨ p ≡ p) → (s ⊨ φ & s ⊨ ψ))›
proof(safe intro!: "→I")
  AOT_assume ‹∀ p (s ⊨ p ≡ p)›
  AOT_hence ‹s ⊨ (φ & ψ) ≡ (φ & ψ)› and ‹s ⊨ φ ≡ φ› and ‹s ⊨ ψ ≡ ψ›
    using "∀E"(1)[rotated, OF "log-prop-prop:2"] by blast+
  moreover AOT_assume ‹s ⊨ (φ & ψ)›
  ultimately AOT_show ‹s ⊨ φ & s ⊨ ψ›
    by (metis "&I" "&E"(1) "&E"(2) "≡E"(1) "≡E"(2))
qed

AOT_theorem world_closed_lem_1_b:
  ‹(s ⊨ φ & (φ → q)) → (∀p (s ⊨ p ≡ p) → s ⊨ q)›
proof(safe intro!: "→I")
  AOT_assume ‹∀ p (s ⊨ p ≡ p)›
  AOT_hence ‹s ⊨ φ ≡ φ› for φ
    using "∀E"(1)[rotated, OF "log-prop-prop:2"] by blast
  moreover AOT_assume ‹s ⊨ φ & (φ → q)›
  ultimately AOT_show ‹s ⊨ q›
    by (metis "&E"(1) "&E"(2) "≡E"(1) "≡E"(2) "→E")
qed

AOT_theorem world_closed_lem_1_c:
```

```
  ‹(s ⊨ φ & s ⊨ (φ → ψ)) → (∀p (s ⊨ p ≡ p) → s ⊨ ψ)›
proof(safe intro!: "→I")
  AOT_assume ‹∀ p (s ⊨ p ≡ p)›
  AOT_hence ‹s ⊨ φ ≡ φ› for φ
    using "∀E"(1)[rotated, OF "log-prop-prop:2"] by blast
  moreover AOT_assume ‹s ⊨ φ & s ⊨ (φ → ψ)›
  ultimately AOT_show ‹s ⊨ ψ›
    by (metis "&E"(1) "&E"(2) "≡E"(1) "≡E"(2) "→E")
qed

AOT_theorem "world-closed-lem:1[0]":                                   (509.1)
  ‹q → (∀p (s ⊨ p ≡ p) → s ⊨ q)›
  by (meson "→I" "≡E"(2) "log-prop-prop:2" "rule-ui:1")

AOT_theorem "world-closed-lem:1[1]":                                   (509.1)
  ‹s ⊨ p_1 & (p_1 → q) → (∀p (s ⊨ p ≡ p) → s ⊨ q)›
  using world_closed_lem_1_b.

AOT_theorem "world-closed-lem:1[2]":                                   (509.1)
  ‹s ⊨ p_1 & s ⊨ p_2 & ((p_1 & p_2) → q) → (∀p (s ⊨ p ≡ p) → s ⊨ q)›
  using world_closed_lem_1_b world_closed_lem_1_a
  by (metis (full_types) "&I" "&E" "→I" "→E")

AOT_theorem "world-closed-lem:1[3]":                                   (509.1)
  ‹s ⊨ p_1 & s ⊨ p_2 & s ⊨ p_3 & ((p_1 & p_2 & p_3) → q) → (∀p (s ⊨ p ≡ p) → s ⊨ q)›
  using world_closed_lem_1_b world_closed_lem_1_a
  by (metis (full_types) "&I" "&E" "→I" "→E")

AOT_theorem "world-closed-lem:1[4]":                                   (509.1)
  ‹s ⊨ p_1 & s ⊨ p_2 & s ⊨ p_3 & s ⊨ p_4 & ((p_1 & p_2 & p_3 & p_4) → q) →
   (∀p (s ⊨ p ≡ p) → s ⊨ q)›
  using world_closed_lem_1_b world_closed_lem_1_a
  by (metis (full_types) "&I" "&E" "→I" "→E")

AOT_theorem "coherent:1": ‹w ⊨ ¬p ≡ ¬w ⊨ p›                           (512.1)
proof(safe intro!: "≡I" "→I")
  AOT_assume 1: ‹w ⊨ ¬p›
  AOT_show ‹¬w ⊨ p›
  proof(rule "raa-cor:2")
    AOT_assume ‹w ⊨ p›
    AOT_hence ‹w ⊨ p & w ⊨ ¬p› using 1 "&I" by blast
    AOT_hence ‹∃q (w ⊨ q & w ⊨ ¬q)› by (rule "∃I")
    moreover AOT_have ‹¬∃q (w ⊨ q & w ⊨ ¬q)›
      using "world-cons:1"[THEN "≡_dfE"[OF cons], THEN "&E"(2)].
    ultimately AOT_show ‹∃q (w ⊨ q & w ⊨ ¬q) & ¬∃q (w ⊨ q & w ⊨ ¬q)›
      using "&I" by blast
  qed
next
  AOT_assume ‹¬w ⊨ p›
  AOT_thus ‹w ⊨ ¬p›
    using "world-max"[THEN "≡_dfE"[OF max], THEN "&E"(2)]
    by (metis "∨E"(2) "log-prop-prop:2" "rule-ui:1")
qed

AOT_theorem "coherent:2": ‹w ⊨ p ≡ ¬w ⊨ ¬p›                           (512.2)
  by (metis "coherent:1" "deduction-theorem" "≡I" "≡E"(1) "≡E"(2) "raa-cor:3")

AOT_theorem "act-world:1": ‹∃w ∀p (w ⊨ p ≡ p)›                        (514.1)
proof -
  AOT_obtain s where s_prop: ‹∀p (s ⊨ p ≡ p)›
    using "sit-classical:6" "Situation.∃E"[rotated] by meson
  AOT_hence ‹◊∀p (s ⊨ p ≡ p)›
    by (metis "T◊" "vdash-properties:10")
```

```
  AOT_hence ‹PossibleWorld(s)›
    using "world:1"[THEN "≡dfI"] Situation.ψ "&I" by blast
  AOT_hence ‹PossibleWorld(s) & ∀p (s ⊨ p ≡ p)›
    using "&I" s_prop by blast
  thus ?thesis by (rule "∃I")
qed

AOT_theorem "act-world:2": ‹∃!w Actual(w)›                                (514.2)
proof -
  AOT_obtain w where w_prop: ‹∀p (w ⊨ p ≡ p)›
    using "act-world:1" "PossibleWorld.∃E"[rotated] by meson
  AOT_have sit_s: ‹Situation(w)›
    using PossibleWorld.ψ "world:1"[THEN "≡dfE", THEN "&E"(1)] by blast
  show ?thesis
  proof (safe intro!: "uniqueness:1"[THEN "≡dfI"] "∃I"(2) "&I" GEN "→I"
                      PossibleWorld.ψ actual[THEN "≡dfI"] sit_s
                      "sit-identity"[unconstrain s, unconstrain s', THEN "→E",
                                     THEN "→E", THEN "≡E"(2)] "≡I"
                      w_prop[THEN "∀E"(2), THEN "≡E"(1)])
    AOT_show ‹PossibleWorld(w)› using PossibleWorld.ψ.
  next
    AOT_show ‹Situation(w)›
      by (simp add: sit_s)
  next
    fix y p
    AOT_assume w_asm: ‹PossibleWorld(y) & Actual(y)›
    AOT_assume ‹w ⊨ p›
    AOT_hence p: ‹p›
      using w_prop[THEN "∀E"(2), THEN "≡E"(1)] by blast
    AOT_show ‹y ⊨ p›
    proof(rule "raa-cor:1")
      AOT_assume ‹¬y ⊨ p›
      AOT_hence ‹y ⊨ ¬p›
        by (metis "coherent:1"[unconstrain w, THEN "→E"] "&E"(1) "≡E"(2) w_asm)
      AOT_hence ‹¬p›
        using w_asm[THEN "&E"(2), THEN actual[THEN "≡dfE"], THEN "&E"(2),
                    THEN "∀E"(1), rotated, OF "log-prop-prop:2"]
              "→E" by blast
      AOT_thus ‹p & ¬p› using p "&I" by blast
    qed
  next
    AOT_show ‹w ⊨ p› if ‹y ⊨ p› and ‹PossibleWorld(y) & Actual(y)› for p y
      using that(2)[THEN "&E"(2), THEN actual[THEN "≡dfE"], THEN "&E"(2),
                    THEN "∀E"(2), THEN "→E", OF that(1)]
            w_prop[THEN "∀E"(2), THEN "≡E"(2)] by blast
  next
    AOT_show ‹Situation(y)› if ‹PossibleWorld(y) & Actual(y)› for y
      using that[THEN "&E"(1)] "world:1"[THEN "≡dfE", THEN "&E"(1)] by blast
  next
    AOT_show ‹Situation(w)›
      using sit_s by blast
  qed(simp)
qed

AOT_theorem "pre-walpha": ‹ιw Actual(w)↓›                                   (516)
  using "A-Exists:2" "RA[2]" "act-world:2" "≡E"(2) by blast

AOT_define TheActualWorld :: ‹κs› (‹wα›)
  "w-alpha": ‹wα =df ιw Actual(w)›                                         (517)

(* TODO: not in PLM *)
AOT_theorem true_in_truth_act_true: ‹⊤ ⊨ p ≡ 𝒜p›
proof(safe intro!: "≡I" "→I")
```

```
  AOT_have true_def: ‹⊢□ ⊤ = ιx (A!x & ∀F (x[F] ≡ ∃p(p & F = [λy p])))›
    by (simp add: "A-descriptions" "rule-id-df:1[zero]" "the-true:1")
  AOT_hence true_den: ‹⊢□ ⊤↓›
    using "t=t-proper:1" "vdash-properties:6" by blast
  {
    AOT_assume ‹⊤ ⊨ p›
    AOT_hence ‹⊤[λy p]›
      by (metis "≡dfE" "con-dis-i-e:2:b" "prop-enc" "true-in-s")
    AOT_hence ‹ιx(A!x & ∀F (x[F] ≡ ∃q (q & F = [λy q])))[λy p]›
      using "rule=E" true_def true_den by fast
    AOT_hence ‹𝒜∃q (q & [λy p] = [λy q])›
      using "≡E"(1) "desc-nec-encode:1"[unvarify F] "prop-prop2:2" by fast
    AOT_hence ‹∃q 𝒜(q & [λy p] = [λy q])›
      by (metis "Act-Basic:10" "≡E"(1))
    then AOT_obtain q where ‹𝒜(q & [λy p] = [λy q])›
      using "∃E"[rotated] by blast
    AOT_hence actq: ‹𝒜q› and ‹𝒜[λy p] = [λy q]›
      using "Act-Basic:2" "intro-elim:3:a" "&E" by blast+
    AOT_hence ‹[λy p] = [λy q]›
      using "id-act:1"[unvarify α β, THEN "≡E"(2)] "prop-prop2:2" by blast
    AOT_hence ‹p = q›
      by (metis "intro-elim:3:b" "p-identity-thm2:3")
    AOT_thus ‹𝒜p›
      using actq "rule=E" id_sym by blast
  }
  {
    AOT_assume ‹𝒜p›
    AOT_hence ‹𝒜(p & [λy p] = [λy p])›
      by (auto intro!: "Act-Basic:2"[THEN "≡E"(2)] "&I"
               intro: "RA[2]" "=I"(1)[OF "prop-prop2:2"])
    AOT_hence ‹∃q 𝒜(q & [λy p] = [λy q])›
      using "∃I" by fast
    AOT_hence ‹𝒜∃q (q & [λy p] = [λy q])›
      by (metis "Act-Basic:10" "≡E"(2))
    AOT_hence ‹ιx(A!x & ∀F (x[F] ≡ ∃q (q & F = [λy q])))[λy p]›
      using "≡E"(2) "desc-nec-encode:1"[unvarify F] "prop-prop2:2" by fast
    AOT_hence ‹⊤[λy p]›
      using "rule=E" true_def true_den id_sym by fast
    AOT_thus ‹⊤ ⊨ p›
      by (safe intro!: "true-in-s"[THEN "≡dfI"] "&I" "possit-sit:6"
                       "prop-enc"[THEN "≡dfI"] true_den)
  }
qed

AOT_theorem "T-world": ‹⊤ = wα›                                          (518)
proof -
  AOT_have true_den: ‹⊢□ ⊤↓›
    using "Situation.res-var:3" "possit-sit:6" "→E" by blast
  AOT_have ‹𝒜∀p (⊤ ⊨ p → p)›
  proof (safe intro!: "logic-actual-nec:3"[axiom_inst, THEN "≡E"(2)] GEN
                      "logic-actual-nec:2"[axiom_inst, THEN "≡E"(2)] "→I")
    fix p
    AOT_assume ‹𝒜⊤ ⊨ p›
    AOT_hence ‹⊤ ⊨ p›
      using "lem2:4"[unconstrain s, unvarify β, OF true_den,
                     THEN "→E", OF "possit-sit:6"] "≡E"(1) by blast
    AOT_thus ‹𝒜p› using true_in_truth_act_true "≡E"(1) by blast
  qed
  moreover AOT_have ‹𝒜(Situation(κ) & ∀p (κ ⊨ p → p)) → 𝒜Actual(κ)› for κ
    using actual[THEN "≡Df", THEN "conventions:3"[THEN "≡dfE", THEN "&E"(2)],
                 THEN "RA[2]", THEN "act-cond"[THEN "→E"]].
  ultimately AOT_have act_act_true: ‹𝒜Actual(⊤)›
    using "possit-sit:4"[unvarify x, OF true_den, THEN "≡E"(2), OF "possit-sit:6"]
```

```
          "Act-Basic:2"[THEN "≡E"(2), OF "&I"] "→E" by blast
  AOT_hence ‹◇Actual(⊤)› by (metis "Act-Sub:3" "vdash-properties:10")
  AOT_hence ‹Possible(⊤)›
    by (safe intro!: pos[THEN "≡dfI"] "&I" "possit-sit:6")
  moreover AOT_have ‹Maximal(⊤)›
  proof (safe intro!: max[THEN "≡dfI"] "&I" "possit-sit:6" GEN)
    fix p
    AOT_have ‹𝒜p ∨ 𝒜¬p›
      by (simp add: "Act-Basic:1")
    moreover AOT_have ‹⊤ ⊨ p› if ‹𝒜p›
        using that true_in_truth_act_true[THEN "≡E"(2)] by blast
    moreover AOT_have ‹⊤ ⊨ ¬p› if ‹𝒜¬p›
      using that true_in_truth_act_true[unvarify p, THEN "≡E"(2)]
            "log-prop-prop:2" by blast
    ultimately AOT_show ‹⊤ ⊨ p ∨ ⊤ ⊨ ¬p›
      using "∨I"(3) "→I" by blast
  qed
  ultimately AOT_have ‹PossibleWorld(⊤)›
    by (safe intro!: "world=maxpos:2"[unvarify x, OF true_den, THEN "≡E"(2)] "&I")
  AOT_hence ‹𝒜PossibleWorld(⊤)›
    using "rigid-pw:4"[unvarify x, OF true_den, THEN "≡E"(2)] by blast
  AOT_hence 1: ‹𝒜(PossibleWorld(⊤) & Actual(⊤))›
    using act_act_true "Act-Basic:2" "df-simplify:2" "intro-elim:3:b" by blast
  AOT_have ‹w_α = ιw(Actual(w))›
    using "rule-id-df:1[zero]"[OF "w-alpha", OF "pre-walpha"] by simp
  moreover AOT_have w_act_den: ‹w_α↓›
    using calculation "t=t-proper:1" "→E" by blast
  ultimately AOT_have ‹∀z (𝒜(PossibleWorld(z) & Actual(z)) → z = w_α)›
    using "nec-hintikka-scheme"[unvarify x] "≡E"(1) "&E" by blast
  AOT_thus ‹⊤ = w_α›
    using "∀E"(1)[rotated, OF true_den] 1 "→E" by blast
qed

AOT_act_theorem "truth-at-alpha:1": ‹p ≡ w_α = ιx (ExtensionOf(x, p))›      (519.1)
  by (metis "rule=E" "T-world" "deduction-theorem" "ext-p-tv:3" id_sym "≡I"
            "≡E"(1) "≡E"(2) "q-True:1")

AOT_act_theorem "truth-at-alpha:2": ‹p ≡ w_α ⊨ p›      (519.2)
proof -
  AOT_have ‹PossibleWorld(w_α)›
    using "&E"(1) "pre-walpha" "rule-id-df:2:b[zero]" "→E"
          "w-alpha" "y-in:3" by blast
  AOT_hence sit_w_alpha: ‹Situation(w_α)›
    by (metis "≡dfE" "&E"(1) "world:1")
  AOT_have w_alpha_den: ‹w_α↓›
    using "pre-walpha" "rule-id-df:2:b[zero]" "w-alpha" by blast
  AOT_have ‹p ≡ ⊤Σp›
    using "q-True:3" by force
  moreover AOT_have ‹⊤ = w_α›
    using "T-world" by auto
  ultimately AOT_have ‹p ≡ w_αΣp›
    using "rule=E" by fast
  moreover AOT_have ‹w_α Σ p ≡ w_α ⊨ p›
    using lem1[unvarify x, OF w_alpha_den, THEN "→E", OF sit_w_alpha]
    using "≡S"(1) "≡E"(1) "Commutativity of ≡" "≡Df" sit_w_alpha "true-in-s" by blast
  ultimately AOT_show ‹p ≡ w_α ⊨ p›
    by (metis "≡E"(5))
qed

AOT_theorem "alpha-world:1": ‹PossibleWorld(w_α)›      (520.1)
proof -
  AOT_have 0: ‹w_α = ιw Actual(w)›
    using "pre-walpha" "rule-id-df:1[zero]" "w-alpha" by blast
```

```
  AOT_hence walpha_den: ‹wα↓›
    by (metis "t=t-proper:1" "vdash-properties:6")
  AOT_have ‹𝒜(PossibleWorld(wα) & Actual(wα))›
    by (rule "actual-desc:2"[unvarify x, OF walpha_den, THEN "→E"]) (fact 0)
  AOT_hence ‹𝒜PossibleWorld(wα)›
    by (metis "Act-Basic:2" "&E"(1) "≡E"(1))
  AOT_thus ‹PossibleWorld(wα)›
    using "rigid-pw:4"[unvarify x, OF walpha_den, THEN "≡E"(1)]
    by blast
qed

AOT_theorem "alpha-world:2": ‹Maximal(wα)›                                    (520.2)
proof -
  AOT_have ‹wα↓›
    using "pre-walpha" "rule-id-df:2:b[zero]" "w-alpha" by blast
  then AOT_obtain x where x_def: ‹x = wα›
    by (metis "instantiation" "rule=I:1" "existential:1" id_sym)
  AOT_hence ‹PossibleWorld(x)› using "alpha-world:1" "rule=E" id_sym by fast
  AOT_hence ‹Maximal(x)› by (metis "world-max"[unconstrain w, THEN "→E"])
  AOT_thus ‹Maximal(wα)› using x_def "rule=E" by blast
qed

AOT_theorem "t-at-alpha-strict": ‹wα ⊨ p ≡ 𝒜p›                                 (521)
proof -
  AOT_have 0: ‹wα = ιw Actual(w)›
    using "pre-walpha" "rule-id-df:1[zero]" "w-alpha" by blast
  AOT_hence walpha_den: ‹wα↓›
    by (metis "t=t-proper:1" "vdash-properties:6")
  AOT_have 1: ‹𝒜(PossibleWorld(wα) & Actual(wα))›
    by (rule "actual-desc:2"[unvarify x, OF walpha_den, THEN "→E"]) (fact 0)
  AOT_have walpha_sit: ‹Situation(wα)›
    by (meson "≡dfE" "alpha-world:2" "&E"(1) max)
  {
    fix p
    AOT_have 2: ‹Situation(x) → (𝒜x ⊨ p ≡ x ⊨ p)› for x
      using "lem2:4"[unconstrain s] by blast
    AOT_assume ‹wα ⊨ p›
    AOT_hence θ: ‹𝒜wα ⊨ p›
      using 2[unvarify x, OF walpha_den, THEN "→E", OF walpha_sit, THEN "≡E"(2)]
      by argo
    AOT_have 3: ‹𝒜Actual(wα)›
      using "1" "Act-Basic:2" "&E"(2) "≡E"(1) by blast
    AOT_have ‹𝒜(Situation(wα) & ∀q(wα ⊨ q → q))›
      apply (AOT_subst (reverse) ‹Situation(wα) & ∀q(wα ⊨ q → q)› ‹Actual(wα)›)
       using actual "≡Df" apply blast
      by (fact 3)
    AOT_hence ‹𝒜∀q(wα ⊨ q → q)› by (metis "Act-Basic:2" "&E"(2) "≡E"(1))
    AOT_hence ‹∀q 𝒜(wα ⊨ q → q)›
      using "logic-actual-nec:3"[axiom_inst, THEN "≡E"(1)] by blast
    AOT_hence ‹𝒜(wα ⊨ p → p)› using "∀E"(2) by blast
    AOT_hence ‹𝒜(wα ⊨ p) → 𝒜p› by (metis "act-cond" "vdash-properties:10")
    AOT_hence ‹𝒜p› using θ "→E" by blast
  }
  AOT_hence 2: ‹wα ⊨ p → 𝒜p› for p by (rule "→I")
  AOT_have walpha_sit: ‹Situation(wα)›
    using "≡dfE" "alpha-world:2" "&E"(1) max by blast
  show ?thesis
  proof(safe intro!: "≡I" "→I" 2)
    AOT_assume actp: ‹𝒜p›
    AOT_show ‹wα ⊨ p›
    proof(rule "raa-cor:1")
      AOT_assume ‹¬wα ⊨ p›
      AOT_hence ‹wα ⊨ ¬p›
```

```
        using "alpha-world:2"[THEN max[THEN "≡dfE"], THEN "&E"(2),
                               THEN "∀E"(1), OF "log-prop-prop:2"]
        by (metis "∨E"(2))
      AOT_hence ‹𝒜¬p›
        using 2[unvarify p, OF "log-prop-prop:2", THEN "→E"] by blast
      AOT_hence ‹¬𝒜p› by (metis "¬¬I" "Act-Sub:1" "≡E"(4))
      AOT_thus ‹𝒜p & ¬𝒜p› using actp "&I" by blast
    qed
  qed
qed

AOT_act_theorem "not-act": ‹w ≠ wα → ¬Actual(w)›                          (522)
proof (rule "→I"; rule "raa-cor:2")
  AOT_assume ‹w ≠ wα›
  AOT_hence 0: ‹¬(w = wα)› by (metis "≡dfE" "=-infix")
  AOT_have walpha_den: ‹wα↓›
    using "pre-walpha" "rule-id-df:2:b[zero]" "w-alpha" by blast
  AOT_have walpha_sit: ‹Situation(wα)›
    using "≡dfE" "alpha-world:2" "&E"(1) max by blast
  AOT_assume act_w: ‹Actual(w)›
  AOT_hence w_sit: ‹Situation(w)› by (metis "≡dfE" actual "&E"(1))
  AOT_have sid: ‹Situation(x') → (w = x' ≡ ∀p (w ⊨ p ≡ x' ⊨ p))› for x'
    using "sit-identity"[unconstrain s', unconstrain s, THEN "→E", OF w_sit]
    by blast
  AOT_have ‹w = wα›
  proof(safe intro!: GEN sid[unvarify x', OF walpha_den, THEN "→E",
                              OF walpha_sit, THEN "≡E"(2)] "≡I" "→I")
    fix p
    AOT_assume ‹w ⊨ p›
    AOT_hence ‹p›
      using actual[THEN "≡dfE", OF act_w, THEN "&E"(2), THEN "∀E"(2), THEN "→E"]
      by blast
    AOT_hence ‹𝒜p›
      by (metis "RA[1]")
    AOT_thus ‹wα ⊨ p›
      using "t-at-alpha-strict"[THEN "≡E"(2)] by blast
  next
    fix p
    AOT_assume ‹wα ⊨ p›
    AOT_hence ‹𝒜p›
      using "t-at-alpha-strict"[THEN "≡E"(1)] by blast
    AOT_hence p: ‹p›
      using "logic-actual"[act_axiom_inst, THEN "→E"] by blast
    AOT_show ‹w ⊨ p›
    proof(rule "raa-cor:1")
      AOT_assume ‹¬w ⊨ p›
      AOT_hence ‹w ⊨ ¬p›
        by (metis "coherent:1" "≡E"(2))
      AOT_hence ‹¬p›
        using actual[THEN "≡dfE", OF act_w, THEN "&E"(2), THEN "∀E"(1),
                     OF "log-prop-prop:2", THEN "→E"] by blast
      AOT_thus ‹p & ¬p› using p "&I" by blast
    qed
  qed
  AOT_thus ‹w = wα & ¬(w = wα)› using 0 "&I" by blast
qed

AOT_act_theorem "w-alpha-part": ‹Actual(s) ≡ s ⊴ wα›                       (523)
proof(safe intro!: "≡I" "→I" "sit-part-whole"[THEN "≡dfI"] "&I" GEN
           dest!:  "sit-part-whole"[THEN "≡dfE"])
  AOT_show ‹Situation(s)› if ‹Actual(s)›
    using "≡dfE" actual "&E"(1) that by blast
next
```

```
  AOT_show ‹Situation(w_α)›
    using "≡dfE" "alpha-world:2" "&E"(1) max by blast
next
  fix p
  AOT_assume ‹Actual(s)›
  moreover AOT_assume ‹s ⊨ p›
  ultimately AOT_have ‹p›
    using actual[THEN "≡dfE", THEN "&E"(2), THEN "∀E"(2), THEN "→E"] by blast
  AOT_thus ‹w_α ⊨ p›
     by (metis "≡E"(1) "truth-at-alpha:2")
next
  AOT_assume 0: ‹Situation(s) & Situation(w_α) & ∀p (s ⊨ p → w_α ⊨ p)›
  AOT_hence ‹s ⊨ p → w_α ⊨ p› for p
    using "&E" "∀E"(2) by blast
  AOT_hence ‹s ⊨ p → p› for p
    by (metis "→I" "≡E"(2) "truth-at-alpha:2" "→E")
  AOT_hence ‹∀p (s ⊨ p → p)› by (rule GEN)
  AOT_thus ‹Actual(s)›
    using actual[THEN "≡dfI", OF "&I", OF 0[THEN "&E"(1), THEN "&E"(1)]] by blast
qed

AOT_act_theorem "act-world2:1": ‹w_α ⊨ p ≡ [λy p]w_α›                      (524.1)
  apply (AOT_subst ‹[λy p]w_α› p)
   apply (rule "beta-C-meta"[THEN "→E", OF "prop-prop2:2", unvarify ν1νn])
  using "pre-walpha" "rule-id-df:2:b[zero]" "w-alpha" apply blast
  using "≡E"(2) "Commutativity of ≡" "truth-at-alpha:2" by blast

AOT_act_theorem "act-world2:2": ‹p ≡ w_α ⊨ [λy p]w_α›                      (524.2)
proof -
  AOT_have ‹p ≡ [λy p]w_α›
    apply (rule "beta-C-meta"[THEN "→E", OF "prop-prop2:2",
                                unvarify ν1νn, symmetric])
    using "pre-walpha" "rule-id-df:2:b[zero]" "w-alpha" by blast
  also AOT_have ‹... ≡ w_α ⊨ [λy p]w_α›
    by (meson "log-prop-prop:2" "rule-ui:1" "truth-at-alpha:2" "universal-cor")
  finally show ?thesis.
qed

AOT_theorem "fund-lem:1": ‹◇p → ◇∃w (w ⊨ p)›                             (525.1)
proof (rule "RM◇"; rule "→I"; rule "raa-cor:1")
  AOT_modally_strict {
    AOT_obtain w where w_prop: ‹∀q (w ⊨ q ≡ q)›
      using "act-world:1" "PossibleWorld.∃E"[rotated] by meson
    AOT_assume p: ‹p›
    AOT_assume 0: ‹¬∃w (w ⊨ p)›
    AOT_have ‹∀w ¬(w ⊨ p)›
      apply (AOT_subst ‹PossibleWorld(x) → ¬x ⊨ p›
                       ‹¬(PossibleWorld(x) & x ⊨ p)› for: x)
      apply (metis "&I" "&E"(1) "&E"(2) "→I" "≡I" "modus-tollens:2")
      using "0" "cqt-further:4" "vdash-properties:10" by blast
    AOT_hence ‹¬(w ⊨ p)›
      using PossibleWorld.ψ "rule-ui:3" "→E" by blast
    AOT_hence ‹¬p›
      using w_prop[THEN "∀E"(2), THEN "≡E"(2)]
      by (metis "raa-cor:3")
    AOT_thus ‹p & ¬p›
      using p "&I" by blast
  }
qed

AOT_theorem "fund-lem:2": ‹◇∃w (w ⊨ p) → ∃w (w ⊨ p)›                     (525.2)
proof (rule "→I")
  AOT_assume ‹◇∃w (w ⊨ p)›
```

```
  AOT_hence ‹∃w ◊(w ⊨ p)›
    using "PossibleWorld.res-var-bound-reas[BF◊]"[THEN "→E"] by auto
  then AOT_obtain w where ‹◊(w ⊨ p)›
    using "PossibleWorld.∃E"[rotated] by meson
  moreover AOT_have ‹Situation(w)›
    by (metis "≡dfE" "&E"(1) pos "world-pos")
  ultimately AOT_have ‹w ⊨ p›
    using "lem2:2"[unconstrain s, THEN "→E"]  "≡E" by blast
  AOT_thus ‹∃w w ⊨ p›
    by (rule "PossibleWorld.∃I")
qed

AOT_theorem "fund-lem:3": ‹p → ∀s(∀q (s ⊨ q ≡ q) → s ⊨ p)›                (525.3)
proof(safe intro!: "→I" Situation.GEN)
  fix s
  AOT_assume ‹p›
  moreover AOT_assume ‹∀q (s ⊨ q ≡ q)›
  ultimately AOT_show ‹s ⊨ p›
    using "∀E"(2) "≡E"(2) by blast
qed

AOT_theorem "fund-lem:4": ‹□p → □∀s(∀q (s ⊨ q ≡ q) → s ⊨ p)›              (525.4)
  using "fund-lem:3" by (rule RM)

AOT_theorem "fund-lem:5": ‹□∀s φ{s} → ∀s □φ{s}›                           (525.5)
proof(safe intro!: "→I" Situation.GEN)
  fix s
  AOT_assume ‹□∀s φ{s}›
  AOT_hence ‹∀s □φ{s}›
    using "Situation.res-var-bound-reas[CBF]"[THEN "→E"] by blast
  AOT_thus ‹□φ{s}›
    using "Situation.∀E" by fast
qed

text‹Note: not explicit in PLM.›
AOT_theorem "fund-lem:5[world]": ‹□∀w φ{w} → ∀w □φ{w}›                    (525.5)
proof(safe intro!: "→I" PossibleWorld.GEN)
  fix w
  AOT_assume ‹□∀w φ{w}›
  AOT_hence ‹∀w □φ{w}›
    using "PossibleWorld.res-var-bound-reas[CBF]"[THEN "→E"] by blast
  AOT_thus ‹□φ{w}›
    using "PossibleWorld.∀E" by fast
qed

AOT_theorem "fund-lem:6": ‹∀w w ⊨ p → □∀w w ⊨ p›                          (525.6)
proof(rule "→I")
  AOT_assume ‹∀w (w ⊨ p)›
  AOT_hence 1: ‹PossibleWorld(w) → (w ⊨ p)› for w
    using "∀E"(2) by blast
  AOT_show ‹□∀w w ⊨ p›
  proof(rule "raa-cor:1")
    AOT_assume ‹¬□∀w w ⊨ p›
    AOT_hence ‹◊¬∀w w ⊨ p›
      by (metis "KBasic:11" "≡E"(1))
    AOT_hence ‹◊∃x (¬(PossibleWorld(x) → x ⊨ p))›
      apply (rule "RM◊"[THEN "→E", rotated])
      by (simp add: "cqt-further:2")
    AOT_hence ‹∃x ◊(¬(PossibleWorld(x) → x ⊨ p))›
      by (metis "BF◊" "vdash-properties:10")
    then AOT_obtain x where x_prop: ‹◊¬(PossibleWorld(x) → x ⊨ p)›
      using "∃E"[rotated] by blast
    AOT_have ‹◊(PossibleWorld(x) & ¬x ⊨ p)›
```

```
        apply (AOT_subst ‹PossibleWorld(x) & ¬x ⊨ p›
                         ‹¬(PossibleWorld(x) → x ⊨ p)›)
         apply (meson "≡E"(6) "oth-class-taut:1:b" "oth-class-taut:3:a")
        by(fact x_prop)
      AOT_hence 2: ‹◇PossibleWorld(x) & ◇¬x ⊨ p›
        by (metis "KBasic2:3" "vdash-properties:10")
      AOT_hence ‹PossibleWorld(x)›
        using "&E"(1) "≡E"(1) "rigid-pw:2" by blast
      AOT_hence ‹□(x ⊨ p)›
        using 2[THEN "&E"(2)]  1[unconstrain w, THEN "→E"] "→E"
              "rigid-truth-at:1"[unconstrain w, THEN "→E"]
        by (metis "≡E"(1))
      moreover AOT_have ‹¬□(x ⊨ p)›
        using 2[THEN "&E"(2)] by (metis "¬¬I" "KBasic:12" "≡E"(4))
      ultimately AOT_show ‹p & ¬p› for p
        by (metis "raa-cor:3")
    qed
  qed

  AOT_theorem "fund-lem:7": ‹□∀w(w ⊨ p) → □p›                    (525.7)
  proof(rule RM; rule "→I")
    AOT_modally_strict {
      AOT_obtain w where w_prop: ‹∀p (w ⊨ p ≡ p)›
        using "act-world:1" "PossibleWorld.∃E"[rotated] by meson
      AOT_assume ‹∀w (w ⊨ p)›
      AOT_hence ‹w ⊨ p›
        using "PossibleWorld.∀E" by fast
      AOT_thus ‹p›
        using w_prop[THEN "∀E"(2), THEN "≡E"(1)] by blast
    }
  qed

  AOT_theorem "fund:1": ‹◇p ≡ ∃w w ⊨ p›                          (526.1)
  proof (rule "≡I"; rule "→I")
    AOT_assume ‹◇p›
    AOT_thus ‹∃w w ⊨ p›
      by (metis "fund-lem:1" "fund-lem:2" "→E")
  next
    AOT_assume ‹∃w w ⊨ p›
    then AOT_obtain w where w_prop: ‹w ⊨ p›
      using "PossibleWorld.∃E"[rotated] by meson
    AOT_hence ‹◇∀p (w ⊨ p ≡ p)›
      using "world:1"[THEN "≡dfE", THEN "&E"(2)] PossibleWorld.ψ "&E" by blast
    AOT_hence ‹∀p ◇(w ⊨ p ≡ p)›
      by (metis "Buridan◇" "→E")
    AOT_hence 1: ‹◇(w ⊨ p ≡ p)›
      by (metis "log-prop-prop:2" "rule-ui:1")
    AOT_have ‹◇((w ⊨ p → p) & (p → w ⊨ p))›
      apply (AOT_subst ‹(w ⊨ p → p) & (p → w ⊨ p)› ‹w ⊨ p ≡ p›)
       apply (meson "conventions:3" "≡E"(6) "oth-class-taut:3:a" "≡Df")
      by (fact 1)
    AOT_hence ‹◇(w ⊨ p → p)›
      by (metis "RM◇" "Conjunction Simplification"(1) "→E")
    moreover AOT_have ‹□(w ⊨ p)›
      using w_prop by (metis "≡E"(1) "rigid-truth-at:1")
    ultimately AOT_show ‹◇p›
      by (metis "KBasic2:4" "≡E"(1) "→E")
  qed

  AOT_theorem "fund:2": ‹□p ≡ ∀w (w ⊨ p)›                        (526.2)
  proof -
    AOT_have 0: ‹∀w (w ⊨ ¬p ≡ ¬w ⊨ p)›
      apply (rule PossibleWorld.GEN)
```

```
    using "coherent:1" by blast
  AOT_have ‹◇¬p ≡ ∃w (w ⊨ ¬p)›
    using "fund:1"[unvarify p, OF "log-prop-prop:2"] by blast
  also AOT_have ‹... ≡ ∃w ¬(w ⊨ p)›
  proof(safe intro!: "≡I" "→I")
    AOT_assume ‹∃w w ⊨ ¬p›
    then AOT_obtain w where w_prop: ‹w ⊨ ¬p›
      using "PossibleWorld.∃E"[rotated] by meson
    AOT_hence ‹¬w ⊨ p›
      using 0[THEN "PossibleWorld.∀E", THEN "≡E"(1)] "&E" by blast
    AOT_thus ‹∃w ¬w ⊨ p›
      by (rule "PossibleWorld.∃I")
  next
    AOT_assume ‹∃w ¬w ⊨ p›
    then AOT_obtain w where w_prop: ‹¬w ⊨ p›
      using "PossibleWorld.∃E"[rotated] by meson
    AOT_hence ‹w ⊨ ¬p›
      using 0[THEN "∀E"(2), THEN "→E", THEN "≡E"(1)] "&E"
      by (metis "coherent:1" "≡E"(2))
    AOT_thus ‹∃w w ⊨ ¬p›
      by (rule "PossibleWorld.∃I")
  qed
  finally AOT_have ‹¬◇¬p ≡ ¬∃w ¬w ⊨ p›
    by (meson "≡E"(1) "oth-class-taut:4:b")
  AOT_hence ‹□p ≡ ¬∃w ¬w ⊨ p›
    by (metis "KBasic:12" "≡E"(5))
  also AOT_have ‹... ≡ ∀w w ⊨ p›
  proof(safe intro!: "≡I" "→I")
    AOT_assume ‹¬∃w ¬w ⊨ p›
    AOT_hence 0: ‹∀x (¬(PossibleWorld(x) & ¬x ⊨ p))›
      by (metis "cqt-further:4" "→E")
    AOT_show ‹∀w w ⊨ p›
      apply (AOT_subst ‹PossibleWorld(x) → x ⊨ p›
                       ‹¬(PossibleWorld(x) & ¬x ⊨ p)› for: x)
       using "oth-class-taut:1:a" apply presburger
      by (fact 0)
  next
    AOT_assume 0: ‹∀w w ⊨ p›
    AOT_have ‹∀x (¬(PossibleWorld(x) & ¬x ⊨ p))›
      by (AOT_subst (reverse) ‹¬(PossibleWorld(x) & ¬x ⊨ p)›
                              ‹PossibleWorld(x) → x ⊨ p› for: x)
         (auto simp: "oth-class-taut:1:a" 0)
    AOT_thus ‹¬∃w ¬w ⊨ p›
      by (metis "∃E" "raa-cor:3" "rule-ui:3")
  qed
  finally AOT_show ‹□p ≡ ∀w w ⊨ p›.
qed

AOT_theorem "fund:3": ‹¬◇p ≡ ¬∃w w ⊨ p›                                    (526.3)
  by (metis (full_types) "contraposition:1[1]" "→I" "fund:1" "≡I" "≡E"(1,2))

AOT_theorem "fund:4": ‹¬□p ≡ ∃w ¬w ⊨p›                                      (526.4)
  apply (AOT_subst ‹∃w ¬w ⊨ p› ‹¬ ∀w w ⊨ p›)
   apply (AOT_subst ‹PossibleWorld(x) → x ⊨ p›
                    ‹¬(PossibleWorld(x) & ¬x ⊨ p)› for: x)
  by (auto simp add: "oth-class-taut:1:a" "conventions:4" "≡Df" RN
                     "fund:2" "rule-sub-lem:1:a")

AOT_theorem "nec-dia-w:1": ‹□p ≡ ∃w w ⊨ □p›                                 (527.1)
proof -
  AOT_have ‹□p ≡ ◇□p›
    using "S5Basic:2" by blast
  also AOT_have ‹... ≡ ∃w w ⊨ □p›
```

```
    using "fund:1"[unvarify p, OF "log-prop-prop:2"] by blast
  finally show ?thesis.
qed

AOT_theorem "nec-dia-w:2": ‹□p ≡ ∀w w ⊨ □p›                                    (527.2)
proof -
  AOT_have ‹□p ≡ □□p›
    using 4 "qml:2"[axiom_inst] "≡I" by blast
  also AOT_have ‹... ≡ ∀w w ⊨ □p›
    using "fund:2"[unvarify p, OF "log-prop-prop:2"] by blast
  finally show ?thesis.
qed

AOT_theorem "nec-dia-w:3": ‹◇p ≡ ∃w w ⊨ ◇p›                                    (527.3)
proof -
  AOT_have ‹◇p ≡ ◇◇p›
    by (simp add: "4◇" "T◇" "≡I")
  also AOT_have ‹... ≡ ∃w w ⊨ ◇p›
    using "fund:1"[unvarify p, OF "log-prop-prop:2"] by blast
  finally show ?thesis.
qed

AOT_theorem "nec-dia-w:4": ‹◇p ≡ ∀w w ⊨ ◇p›                                    (527.4)
proof -
  AOT_have ‹◇p ≡ □◇p›
    by (simp add: "S5Basic:1")
  also AOT_have ‹... ≡ ∀w w ⊨ ◇p›
    using "fund:2"[unvarify p, OF "log-prop-prop:2"] by blast
  finally show ?thesis.
qed

AOT_theorem "conj-dist-w:1": ‹w ⊨ (p & q) ≡ ((w ⊨ p) & (w ⊨ q))›              (528.1)
proof(safe intro!: "≡I" "→I")
  AOT_assume ‹w ⊨ (p & q)›
  AOT_hence 0: ‹□w ⊨ (p & q)›
    using "rigid-truth-at:1"[unvarify p, THEN "≡E"(1), OF "log-prop-prop:2"]
    by blast
  AOT_modally_strict {
    AOT_have ‹∀p (w ⊨ p ≡ p) → ((w ⊨ (φ & ψ)) → (w ⊨ φ & w ⊨ ψ))› for w φ ψ
    proof(safe intro!: "→I")
      AOT_assume ‹∀ p (w ⊨ p ≡ p)›
      AOT_hence ‹w ⊨ (φ & ψ) ≡ (φ & ψ)› and ‹w ⊨ φ ≡ φ› and ‹w ⊨ ψ ≡ ψ›
        using "∀E"(1)[rotated, OF "log-prop-prop:2"] by blast+
      moreover AOT_assume ‹w ⊨ (φ & ψ)›
      ultimately AOT_show ‹w ⊨ φ & w ⊨ ψ›
        by (metis "&I" "&E"(1) "&E"(2) "≡E"(1) "≡E"(2))
    qed
  }
  AOT_hence ‹◇∀p (w ⊨ p ≡ p) → ◇(w ⊨ (φ & ψ) → w ⊨ φ & w ⊨ ψ)› for w φ ψ
    by (rule "RM◇")
  moreover AOT_have pos: ‹◇∀p (w ⊨ p ≡ p)›                                    (491)
    using "world:1"[THEN "≡dfE", OF PossibleWorld.ψ] "&E" by blast
  ultimately AOT_have ‹◇(w ⊨ (p & q) → w ⊨ p & w ⊨ q)› using "→E" by blast
  AOT_hence ‹◇(w ⊨ p) & ◇(w ⊨ q)›
    by (metis 0 "KBasic2:3" "KBasic2:4" "≡E"(1) "vdash-properties:10")
  AOT_thus ‹w ⊨ p & w ⊨ q›
    using "rigid-truth-at:2"[unvarify p, THEN "≡E"(1), OF "log-prop-prop:2"]
          "&E" "&I" by meson
next
  AOT_assume ‹w ⊨ p & w ⊨ q›
  AOT_hence ‹□w ⊨ p & □w ⊨ q›
    using "rigid-truth-at:1"[unvarify p, THEN "≡E"(1), OF "log-prop-prop:2"]
          "&E" "&I" by blast
```

```
  AOT_hence 0: ‹□(w ⊨ p & w ⊨ q)›
    by (metis "KBasic:3" "≡E"(2))
  AOT_modally_strict {
    AOT_have ‹∀p (w ⊨ p ≡ p) → ((w ⊨ φ & w ⊨ ψ) → (w ⊨ (φ & ψ)))› for w φ ψ
    proof(safe intro!: "→I")
      AOT_assume ‹∀ p (w ⊨ p ≡ p)›
      AOT_hence ‹w ⊨ (φ & ψ) ≡ (φ & ψ)› and ‹w ⊨ φ ≡ φ› and ‹w ⊨ ψ ≡ ψ›
        using "∀E"(1)[rotated, OF "log-prop-prop:2"] by blast+
      moreover AOT_assume ‹w ⊨ φ & w ⊨ ψ›
      ultimately AOT_show ‹w ⊨ (φ & ψ)›
        by (metis "&I" "&E"(1) "&E"(2) "≡E"(1) "≡E"(2))
    qed
  }
  AOT_hence ‹◊∀p (w ⊨ p ≡ p) → ◊((w ⊨ φ & w ⊨ ψ) → w ⊨ (φ & ψ))› for w φ ψ
    by (rule "RM◊")
  moreover AOT_have pos: ‹◊∀p (w ⊨ p ≡ p)›                                   (491)
    using "world:1"[THEN "≡dfE", OF PossibleWorld.ψ] "&E" by blast
  ultimately AOT_have ‹◊((w ⊨ p & w ⊨ q) → w ⊨ (p & q))›
    using "→E" by blast
  AOT_hence ‹◊(w ⊨ (p & q))›
    by (metis 0 "KBasic2:4" "≡E"(1) "vdash-properties:10")
  AOT_thus ‹w ⊨ (p & q)›
    using "rigid-truth-at:2"[unvarify p, THEN "≡E"(1), OF "log-prop-prop:2"]
    by blast
qed

AOT_theorem "conj-dist-w:2": ‹w ⊨ (p → q) ≡ ((w ⊨ p) → (w ⊨ q))›        (528.2)
proof(safe intro!: "≡I" "→I")
  AOT_assume ‹w ⊨ (p → q)›
  AOT_hence 0: ‹□w ⊨ (p → q)›
    using "rigid-truth-at:1"[unvarify p, THEN "≡E"(1), OF "log-prop-prop:2"]
    by blast
  AOT_assume ‹w ⊨ p›
  AOT_hence 1: ‹□w ⊨ p›
    by (metis "T◊" "≡E"(1) "rigid-truth-at:3" "→E")
  AOT_modally_strict {
    AOT_have ‹∀p (w ⊨ p ≡ p) → ((w ⊨ (φ → ψ)) → (w ⊨ φ → w ⊨ ψ))› for w φ ψ
    proof(safe intro!: "→I")
      AOT_assume ‹∀ p (w ⊨ p ≡ p)›
      AOT_hence ‹w ⊨ (φ → ψ) ≡ (φ → ψ)› and ‹w ⊨ φ ≡ φ› and ‹w ⊨ ψ ≡ ψ›
        using "∀E"(1)[rotated, OF "log-prop-prop:2"] by blast+
      moreover AOT_assume ‹w ⊨ (φ → ψ)›
      moreover AOT_assume ‹w ⊨ φ›
      ultimately AOT_show ‹w ⊨ ψ›
        by (metis "≡E"(1) "≡E"(2) "→E")
    qed
  }
  AOT_hence ‹◊∀p (w ⊨ p ≡ p) → ◊(w ⊨ (φ → ψ) → (w ⊨ φ → w ⊨ ψ))› for w φ ψ
    by (rule "RM◊")
  moreover AOT_have pos: ‹◊∀p (w ⊨ p ≡ p)›                                   (491)
    using "world:1"[THEN "≡dfE", OF PossibleWorld.ψ] "&E" by blast
  ultimately AOT_have ‹◊(w ⊨ (p → q) → (w ⊨ p → w ⊨ q))›
    using "→E" by blast
  AOT_hence ‹◊(w ⊨ p → w ⊨ q)›
    by (metis 0 "KBasic2:4" "≡E"(1) "→E")
  AOT_hence ‹◊w ⊨ q›
    by (metis 1 "KBasic2:4" "≡E"(1) "→E")
  AOT_thus ‹w ⊨ q›
    using "rigid-truth-at:2"[unvarify p, THEN "≡E"(1), OF "log-prop-prop:2"]
          "&E" "&I" by meson
next
  AOT_assume ‹w ⊨ p → w ⊨ q›
  AOT_hence ‹¬(w ⊨ p) ∨ w ⊨ q›
```

```
    by (metis "∨I"(1) "∨I"(2) "reductio-aa:1" "→E")
  AOT_hence ‹w ⊨ ¬p ∨ w ⊨ q›
    by (metis "coherent:1" "∨I"(1) "∨I"(2) "∨E"(2) "≡E"(2) "reductio-aa:1")
  AOT_hence 0: ‹□(w ⊨ ¬p ∨ w ⊨ q)›
    using "rigid-truth-at:1"[unvarify p, THEN "≡E"(1), OF "log-prop-prop:2"]
    by (metis "KBasic:15" "∨I"(1) "∨I"(2) "∨E"(2) "reductio-aa:1" "→E")
  AOT_modally_strict {
    AOT_have ‹∀p (w ⊨ p ≡ p) → ((w ⊨ ¬φ ∨ w ⊨ ψ) → (w ⊨ (φ → ψ)))› for w φ ψ
    proof(safe intro!: "→I")
      AOT_assume ‹∀ p (w ⊨ p ≡ p)›
      moreover AOT_assume ‹w ⊨ ¬φ ∨ w ⊨ ψ›
      ultimately AOT_show ‹w ⊨ (φ → ψ)›
        by (metis "∨E"(2) "→I" "≡E"(1) "≡E"(2) "log-prop-prop:2"
                  "reductio-aa:1" "rule-ui:1")
    qed
  }
  AOT_hence ‹◊∀p (w ⊨ p ≡ p) → ◊((w ⊨ ¬φ ∨ w ⊨ ψ) → w ⊨ (φ → ψ))› for w φ ψ
    by (rule "RM◊")
  moreover AOT_have pos: ‹◊∀p (w ⊨ p ≡ p)›                                    (491)
    using "world:1"[THEN "≡dfE", OF PossibleWorld.ψ] "&E" by blast
  ultimately AOT_have ‹◊((w ⊨ ¬p ∨ w ⊨ q) → w ⊨ (p → q))›
    using "→E" by blast
  AOT_hence ‹◊(w ⊨ (p → q))›
    by (metis 0 "KBasic2:4" "≡E"(1) "→E")
  AOT_thus ‹w ⊨ (p → q)›
    using "rigid-truth-at:2"[unvarify p, THEN "≡E"(1), OF "log-prop-prop:2"]
    by blast
qed

AOT_theorem "conj-dist-w:3": ‹w ⊨ (p ∨ q) ≡ ((w ⊨ p) ∨ (w ⊨ q))›            (528.3)
proof(safe intro!: "≡I" "→I")
  AOT_assume ‹w ⊨ (p ∨ q)›
  AOT_hence 0: ‹□w ⊨ (p ∨ q)›
    using "rigid-truth-at:1"[unvarify p, THEN "≡E"(1), OF "log-prop-prop:2"]
    by blast
  AOT_modally_strict {
    AOT_have ‹∀p (w ⊨ p ≡ p) → ((w ⊨ (φ ∨ ψ)) → (w ⊨ φ ∨ w ⊨ ψ))› for w φ ψ
    proof(safe intro!: "→I")
      AOT_assume ‹∀ p (w ⊨ p ≡ p)›
      AOT_hence ‹w ⊨ (φ ∨ ψ) ≡ (φ ∨ ψ)› and ‹w ⊨ φ ≡ φ› and ‹w ⊨ ψ ≡ ψ›
        using "∀E"(1)[rotated, OF "log-prop-prop:2"] by blast+
      moreover AOT_assume ‹w ⊨ (φ ∨ ψ)›
      ultimately AOT_show ‹w ⊨ φ ∨ w ⊨ ψ›
        by (metis "∨I"(1) "∨I"(2) "∨E"(3) "≡E"(1) "≡E"(2) "reductio-aa:1")
    qed
  }
  AOT_hence ‹◊∀p (w ⊨ p ≡ p) → ◊(w ⊨ (φ ∨ ψ) → (w ⊨ φ ∨ w ⊨ ψ))› for w φ ψ
    by (rule "RM◊")
  moreover AOT_have pos: ‹◊∀p (w ⊨ p ≡ p)›                                    (491)
    using "world:1"[THEN "≡dfE", OF PossibleWorld.ψ] "&E" by blast
  ultimately AOT_have ‹◊(w ⊨ (p ∨ q) → (w ⊨ p ∨ w ⊨ q))› using "→E" by blast
  AOT_hence ‹◊(w ⊨ p ∨ w ⊨ q)›
    by (metis 0 "KBasic2:4" "≡E"(1) "vdash-properties:10")
  AOT_hence ‹◊w ⊨ p ∨ ◊w ⊨ q›
    using "KBasic2:2"[THEN "≡E"(1)] by blast
  AOT_thus ‹w ⊨ p ∨ w ⊨ q›
    using "rigid-truth-at:2"[unvarify p, THEN "≡E"(1), OF "log-prop-prop:2"]
    by (metis "∨I"(1) "∨I"(2) "∨E"(2) "reductio-aa:1")
next
  AOT_assume ‹w ⊨ p ∨ w ⊨ q›
  AOT_hence 0: ‹□(w ⊨ p ∨ w ⊨ q)›
    using "rigid-truth-at:1"[unvarify p, THEN "≡E"(1), OF "log-prop-prop:2"]
    by (metis "KBasic:15" "∨I"(1) "∨I"(2) "∨E"(2) "reductio-aa:1" "→E")
```

```
  AOT_modally_strict {
    AOT_have ‹∀p (w ⊨ p ≡ p) → ((w ⊨ φ ∨ w ⊨ ψ) → (w ⊨ (φ ∨ ψ)))› for w φ ψ
    proof(safe intro!: "→I")
      AOT_assume ‹∀ p (w ⊨ p ≡ p)›
      moreover AOT_assume ‹w ⊨ φ ∨ w ⊨ ψ›
      ultimately AOT_show ‹w ⊨ (φ ∨ ψ)›
        by (metis "∨I"(1) "∨I"(2) "∨E"(2) "≡E"(1) "≡E"(2)
                  "log-prop-prop:2" "reductio-aa:1" "rule-ui:1")
    qed
  }
  AOT_hence ‹◊∀p (w ⊨ p ≡ p) → ◊((w ⊨ φ ∨ w ⊨ ψ) → w ⊨ (φ ∨ ψ))› for w φ ψ
    by (rule "RM◊")
  moreover AOT_have pos: ‹◊∀p (w ⊨ p ≡ p)›                                  (491)
    using "world:1"[THEN "≡dfE", OF PossibleWorld.ψ] "&E" by blast
  ultimately AOT_have ‹◊((w ⊨ p ∨ w ⊨ q) → w ⊨ (p ∨ q))›
    using "→E" by blast
  AOT_hence ‹◊(w ⊨ (p ∨ q))›
    by (metis 0 "KBasic2:4" "≡E"(1) "→E")
  AOT_thus ‹w ⊨ (p ∨ q)›
    using "rigid-truth-at:2"[unvarify p, THEN "≡E"(1), OF "log-prop-prop:2"]
    by blast
qed

AOT_theorem "conj-dist-w:4": ‹w ⊨ (p ≡ q) ≡ ((w ⊨ p) ≡ (w ⊨ q))›          (528.4)
proof(rule "≡I"; rule "→I")
  AOT_assume ‹w ⊨ (p ≡ q)›
  AOT_hence 0: ‹□w ⊨ (p ≡ q)›
    using "rigid-truth-at:1"[unvarify p, THEN "≡E"(1), OF "log-prop-prop:2"]
    by blast
  AOT_modally_strict {
    AOT_have ‹∀p (w ⊨ p ≡ p) → ((w ⊨ (φ ≡ ψ)) → (w ⊨ φ ≡ w ⊨ ψ))› for w φ ψ
    proof(safe intro!: "→I")
      AOT_assume ‹∀ p (w ⊨ p ≡ p)›
      AOT_hence ‹w ⊨ (φ ≡ ψ) ≡ (φ ≡ ψ)› and ‹w ⊨ φ ≡ φ› and ‹w ⊨ ψ ≡ ψ›
        using "∀E"(1)[rotated, OF "log-prop-prop:2"] by blast+
      moreover AOT_assume ‹w ⊨ (φ ≡ ψ)›
      ultimately AOT_show ‹w ⊨ φ ≡ w ⊨ ψ›
        by (metis "≡E"(2) "≡E"(5) "Commutativity of ≡")
    qed
  }
  AOT_hence ‹◊∀p (w ⊨ p ≡ p) → ◊(w ⊨ (φ ≡ ψ) → (w ⊨ φ ≡ w ⊨ ψ))› for w φ ψ
    by (rule "RM◊")
  moreover AOT_have pos: ‹◊∀p (w ⊨ p ≡ p)›                                  (491)
    using "world:1"[THEN "≡dfE", OF PossibleWorld.ψ] "&E" by blast
  ultimately AOT_have ‹◊(w ⊨ (p ≡ q) → (w ⊨ p ≡ w ⊨ q))›
    using "→E" by blast
  AOT_hence 1: ‹◊(w ⊨ p ≡ w ⊨ q)›
    by (metis 0 "KBasic2:4" "≡E"(1) "vdash-properties:10")
  AOT_have ‹◊((w ⊨ p → w ⊨ q) & (w ⊨ q → w ⊨ p))›
    apply (AOT_subst ‹(w ⊨ p → w ⊨ q) & (w ⊨ q → w ⊨ p)› ‹w ⊨ p ≡ w ⊨ q›)
     apply (meson "≡dfE" "conventions:3" "→I" "df-rules-formulas[4]" "≡I")
    by (fact 1)
  AOT_hence 2: ‹◊(w ⊨ p → w ⊨ q) & ◊(w ⊨ q → w ⊨ p)›
    by (metis "KBasic2:3" "vdash-properties:10")
  AOT_have ‹◊(¬w ⊨ p ∨ w ⊨ q)› and ‹◊(¬w ⊨ q ∨ w ⊨ p)›
     apply (AOT_subst (reverse) ‹¬w ⊨ p ∨ w ⊨ q› ‹w ⊨ p → w ⊨ q›)
      apply (simp add: "oth-class-taut:1:c")
     apply (fact 2[THEN "&E"(1)])
    apply (AOT_subst (reverse) ‹¬w ⊨ q ∨ w ⊨ p› ‹w ⊨ q → w ⊨ p›)
     apply (simp add: "oth-class-taut:1:c")
    by (fact 2[THEN "&E"(2)])
  AOT_hence ‹◊(¬w ⊨ p) ∨ ◊w ⊨ q› and ‹◊¬w ⊨ q ∨ ◊w ⊨ p›
    using "KBasic2:2" "≡E"(1) by blast+
```

```
  AOT_hence ‹¬□w ⊨ p ∨ ◇w ⊨ q› and ‹¬□w ⊨ q ∨ ◇w ⊨ p›
    by (metis "KBasic:11" "∨I"(1) "∨I"(2) "∨E"(2) "≡E"(2) "raa-cor:1")+
  AOT_thus ‹w ⊨ p ≡ w ⊨ q›
    using "rigid-truth-at:2"[unvarify p, THEN "≡E"(1), OF "log-prop-prop:2"]
    by (metis "¬¬I" "T◇" "∨E"(2) "→I" "≡I" "≡E"(1) "rigid-truth-at:3")
next
  AOT_have ‹□PossibleWorld(w)›
    using "≡E"(1) "rigid-pw:1" PossibleWorld.ψ by blast
  moreover {
    fix p
    AOT_modally_strict {
      AOT_have ‹PossibleWorld(w) → (w ⊨ p → □w ⊨ p)›
        using "rigid-truth-at:1" "→I"
        by (metis "≡E"(1))
    }
    AOT_hence ‹□PossibleWorld(w) → □(w ⊨ p → □w ⊨ p)›
      by (rule RM)
  }
  ultimately AOT_have 1: ‹□(w ⊨ p → □w ⊨ p)› for p
    by (metis "→E")
  AOT_assume ‹w ⊨ p ≡ w ⊨ q›
  AOT_hence 0: ‹□(w ⊨ p ≡ w ⊨ q)›
    using "sc-eq-box-box:5"[THEN "→E", THEN "qml:2"[axiom_inst, THEN "→E"],
                            THEN "→E", OF "&I"]
          by (metis "1")
  AOT_modally_strict {
    AOT_have ‹∀p (w ⊨ p ≡ p) → ((w ⊨ φ ≡ w ⊨ ψ) → (w ⊨ (φ ≡ ψ)))› for w φ ψ
    proof(safe intro!: "→I")
      AOT_assume ‹∀ p (w ⊨ p ≡ p)›
      moreover AOT_assume ‹w ⊨ φ ≡ w ⊨ ψ›
      ultimately AOT_show ‹w ⊨ (φ ≡ ψ)›
        by (metis "≡E"(2) "≡E"(6) "log-prop-prop:2" "rule-ui:1")
    qed
  }
  AOT_hence ‹◇∀p (w ⊨ p ≡ p) → ◇((w ⊨ φ ≡ w ⊨ ψ) → w ⊨ (φ ≡ ψ))› for w φ ψ
    by (rule "RM◇")
  moreover AOT_have pos: ‹◇∀p (w ⊨ p ≡ p)›                                    (491)
    using "world:1"[THEN "≡dfE", OF PossibleWorld.ψ] "&E" by blast
  ultimately AOT_have ‹◇((w ⊨ p ≡  w ⊨ q) → w ⊨ (p ≡ q))›
    using "→E" by blast
  AOT_hence ‹◇(w ⊨ (p ≡ q))›
    by (metis 0 "KBasic2:4" "≡E"(1) "→E")
  AOT_thus ‹w ⊨ (p ≡ q)›
    using "rigid-truth-at:2"[unvarify p, THEN "≡E"(1), OF "log-prop-prop:2"]
    by blast
qed

AOT_theorem "conj-dist-w:5": ‹w ⊨ (∀α φ{α}) ≡ (∀ α (w ⊨ φ{α}))›              (528.5)
proof(safe intro!: "≡I" "→I" GEN)
  AOT_assume ‹w ⊨ (∀α φ{α})›
  AOT_hence 0: ‹□w ⊨ (∀α φ{α})›
    using "rigid-truth-at:1"[unvarify p, THEN "≡E"(1), OF "log-prop-prop:2"]
    by blast
  AOT_modally_strict {
    AOT_have ‹∀p (w ⊨ p ≡ p) → ((w ⊨ (∀α φ{α})) → (∀α w ⊨ φ{α}))› for w
    proof(safe intro!: "→I" GEN)
      AOT_assume ‹∀p (w ⊨ p ≡ p)›
      moreover AOT_assume ‹w ⊨ (∀α φ{α})›
      ultimately AOT_show ‹w ⊨ φ{α}› for α
        by (metis "≡E"(1) "≡E"(2) "log-prop-prop:2" "rule-ui:1" "rule-ui:3")
    qed
  }
  AOT_hence ‹◇∀p (w ⊨ p ≡ p) → ◇(w ⊨ (∀α φ{α}) → (∀α w ⊨ φ{α}))› for w
```

```
    by (rule "RM◇")
  moreover AOT_have pos: ‹◇∀p (w ⊨ p ≡ p)›                                (491)
    using "world:1"[THEN "≡dfE", OF PossibleWorld.ψ] "&E" by blast
  ultimately AOT_have ‹◇(w ⊨ (∀α φ{α}) → (∀α w ⊨ φ{α}))› using "→E" by blast
  AOT_hence ‹◇(∀α w ⊨ φ{α})›
    by (metis 0 "KBasic2:4" "≡E"(1) "→E")
  AOT_hence ‹∀α ◇w ⊨ φ{α}›
    by (metis "Buridan◇" "→E")
  AOT_thus ‹w ⊨ φ{α}› for α
    using "rigid-truth-at:2"[unvarify p, THEN "≡E"(1), OF "log-prop-prop:2"]
          "∀E"(2) by blast
next
  AOT_assume ‹∀α w ⊨ φ{α}›
  AOT_hence ‹w ⊨ φ{α}› for α using "∀E"(2) by blast
  AOT_hence ‹□w ⊨ φ{α}› for α
    using "rigid-truth-at:1"[unvarify p, THEN "≡E"(1), OF "log-prop-prop:2"]
          "&E" "&I" by blast
  AOT_hence ‹∀α □w ⊨ φ{α}› by (rule GEN)
  AOT_hence 0: ‹□∀α w ⊨ φ{α}› by (rule BF[THEN "→E"])
  AOT_modally_strict {
    AOT_have ‹∀p (w ⊨ p ≡ p) → ((∀α w ⊨ φ{α}) → (w ⊨ (∀α φ{α})))› for w
    proof(safe intro!: "→I")
      AOT_assume ‹∀ p (w ⊨ p ≡ p)›
      moreover AOT_assume ‹∀α w ⊨ φ{α}›
      ultimately AOT_show ‹w ⊨ (∀α φ{α})›
        by (metis "≡E"(1) "≡E"(2) "log-prop-prop:2" "rule-ui:1"
                  "rule-ui:3" "universal-cor")
    qed
  }
  AOT_hence ‹◇∀p (w ⊨ p ≡ p) → ◇((∀α w ⊨ φ{α}) → w ⊨ (∀α φ{α}))› for w
    by (rule "RM◇")
  moreover AOT_have pos: ‹◇∀p (w ⊨ p ≡ p)›                                (491)
    using "world:1"[THEN "≡dfE", OF PossibleWorld.ψ] "&E" by blast
  ultimately AOT_have ‹◇((∀α w ⊨ φ{α}) → w ⊨ (∀α φ{α}))›
    using "→E" by blast
  AOT_hence ‹◇(w ⊨ (∀α φ{α}))›
    by (metis 0 "KBasic2:4" "≡E"(1) "→E")
  AOT_thus ‹w ⊨ (∀α φ{α})›
    using "rigid-truth-at:2"[unvarify p, THEN "≡E"(1), OF "log-prop-prop:2"]
    by blast
qed

AOT_theorem "conj-dist-w:6": ‹w ⊨ (∃α φ{α}) ≡ (∃ α (w ⊨ φ{α}))›         (528.6)
proof(safe intro!: "≡I" "→I" GEN)
  AOT_assume ‹w ⊨ (∃α φ{α})›
  AOT_hence 0: ‹□w ⊨ (∃α φ{α})›
    using "rigid-truth-at:1"[unvarify p, THEN "≡E"(1), OF "log-prop-prop:2"]
    by blast
  AOT_modally_strict {
    AOT_have ‹∀p (w ⊨ p ≡ p) → ((w ⊨ (∃α φ{α})) → (∃α w ⊨ φ{α}))› for w
    proof(safe intro!: "→I" GEN)
      AOT_assume ‹∀p (w ⊨ p ≡ p)›
      moreover AOT_assume ‹w ⊨ (∃α φ{α})›
      ultimately AOT_show ‹∃ α (w ⊨ φ{α})›
        by (metis "∃E" "∃I"(2) "≡E"(1,2) "log-prop-prop:2" "rule-ui:1")
    qed
  }
  AOT_hence ‹◇∀p (w ⊨ p ≡ p) → ◇(w ⊨ (∃α φ{α}) → (∃α w ⊨ φ{α}))› for w
    by (rule "RM◇")
  moreover AOT_have pos: ‹◇∀p (w ⊨ p ≡ p)›                                (491)
    using "world:1"[THEN "≡dfE", OF PossibleWorld.ψ] "&E" by blast
  ultimately AOT_have ‹◇(w ⊨ (∃α φ{α}) → (∃α w ⊨ φ{α}))› using "→E" by blast
  AOT_hence ‹◇(∃α w ⊨ φ{α})›
```

```
    by (metis 0 "KBasic2:4" "≡E"(1) "→E")
  AOT_hence ‹∃α ◇w ⊨ φ{α}›
    by (metis "BF◇" "→E")
  then AOT_obtain α where ‹◇w ⊨ φ{α}›
    using "∃E"[rotated] by blast
  AOT_hence ‹w ⊨ φ{α}›
    using "rigid-truth-at:2"[unvarify p, THEN "≡E"(1), OF "log-prop-prop:2"] by blast
  AOT_thus ‹∃ α w ⊨ φ{α}› by (rule "∃I")
next
  AOT_assume ‹∃α w ⊨ φ{α}›
  then AOT_obtain α where ‹w ⊨ φ{α}› using "∃E"[rotated] by blast
  AOT_hence ‹□w ⊨ φ{α}›
    using "rigid-truth-at:1"[unvarify p, THEN "≡E"(1), OF "log-prop-prop:2"]
          "&E" "&I" by blast
  AOT_hence ‹∃α □w ⊨ φ{α}›
    by (rule "∃I")
  AOT_hence 0: ‹□∃α w ⊨ φ{α}›
    by (metis Buridan "→E")
  AOT_modally_strict {
    AOT_have ‹∀p (w ⊨ p ≡ p) → ((∃α w ⊨ φ{α}) → (w ⊨ (∃α φ{α})))› for w
    proof(safe intro!: "→I")
      AOT_assume ‹∀ p (w ⊨ p ≡ p)›
      moreover AOT_assume ‹∃α w ⊨ φ{α}›
      then AOT_obtain α where ‹w ⊨ φ{α}›
        using "∃E"[rotated] by blast
      ultimately AOT_show ‹w ⊨ (∃α φ{α})›
        by (metis "∃I"(2) "≡E"(1,2) "log-prop-prop:2" "rule-ui:1")
    qed
  }
  AOT_hence ‹◇∀p (w ⊨ p ≡ p) → ◇((∃α w ⊨ φ{α}) → w ⊨ (∃α φ{α}))› for w
    by (rule "RM◇")
  moreover AOT_have pos: ‹◇∀p (w ⊨ p ≡ p)›                                        (491)
    using "world:1"[THEN "≡dfE", OF PossibleWorld.ψ] "&E" by blast
  ultimately AOT_have ‹◇((∃α w ⊨ φ{α}) → w ⊨ (∃α φ{α}))›
    using "→E" by blast
  AOT_hence ‹◇(w ⊨ (∃α φ{α}))›
    by (metis 0 "KBasic2:4" "≡E"(1) "→E")
  AOT_thus ‹w ⊨ (∃α φ{α})›
    using "rigid-truth-at:2"[unvarify p, THEN "≡E"(1), OF "log-prop-prop:2"]
    by blast
qed

AOT_theorem "conj-dist-w:7": ‹(w ⊨ □p) → □w ⊨ p›                                  (528.7)
proof(rule "→I")
  AOT_assume ‹w ⊨ □p›
  AOT_hence ‹∃w w ⊨ □p› by (rule "PossibleWorld.∃I")
  AOT_hence ‹◇□p› using "fund:1"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(2)]
    by blast
  AOT_hence ‹□p›
    by (metis "5◇" "→E")
  AOT_hence 1: ‹□□p›
    by (metis "S5Basic:6" "≡E"(1))
  AOT_have ‹□∀w w ⊨ p›
    by (AOT_subst (reverse) ‹∀w w ⊨ p› ‹□p›)
      (auto simp add: "fund:2" 1)
  AOT_hence ‹∀w □w ⊨ p›
    using "fund-lem:5[world]"[THEN "→E"] by simp
  AOT_thus ‹□w ⊨ p›
    using "→E" "PossibleWorld.∀E" by fast
qed

AOT_theorem "conj-dist-w:8": ‹∃w∃p((□w ⊨ p) & ¬w ⊨ □p)›                           (528.8)
proof -
```

```
  AOT_obtain r where A: r and ‹◇¬r›
    by (metis "&E"(1) "&E"(2) "≡dfE" "∃E" "cont-tf:1" "cont-tf-thm:1")
  AOT_hence B: ‹¬□r›
    by (metis "KBasic:11" "≡E"(2))
  AOT_have ‹◇r›
    using A "T◇"[THEN "→E"] by simp
  AOT_hence ‹∃w w ⊨ r›
    using "fund:1"[THEN "≡E"(1)] by blast
  then AOT_obtain w where w: ‹w ⊨ r›
    using "PossibleWorld.∃E"[rotated] by meson
  AOT_hence ‹□w ⊨ r›
    by (metis "T◇" "≡E"(1) "rigid-truth-at:3" "vdash-properties:10")
  moreover AOT_have ‹¬w ⊨ □r›
  proof(rule "raa-cor:2")
    AOT_assume ‹w ⊨ □r›
    AOT_hence ‹∃w w ⊨ □r›
      by (rule "PossibleWorld.∃I")
    AOT_hence ‹□r›
      by (metis "≡E"(2) "nec-dia-w:1")
    AOT_thus ‹□r & ¬□r›
      using B "&I" by blast
  qed
  ultimately AOT_have ‹□w ⊨ r & ¬w ⊨ □r›
    by (rule "&I")
  AOT_hence ‹∃p (□w ⊨ p & ¬w ⊨ □p)›
    by (rule "∃I")
  thus ?thesis
    by (rule "PossibleWorld.∃I")
qed

AOT_theorem "conj-dist-w:9": ‹(◇w ⊨ p) → w ⊨ ◇p›                    (528.9)
proof(rule "→I"; rule "raa-cor:1")
  AOT_assume ‹◇w ⊨ p›
  AOT_hence 0: ‹w ⊨ p›
    by (metis "≡E"(1) "rigid-truth-at:2")
  AOT_assume ‹¬w ⊨ ◇p›
  AOT_hence 1: ‹w ⊨ ¬◇p›
    using "coherent:1"[unvarify p, THEN "≡E"(2), OF "log-prop-prop:2"] by blast
  moreover AOT_have ‹w ⊨ (¬◇p → ¬p)›
    using "T◇"[THEN "contraposition:1[1]", THEN RN]
          "fund:2"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(1), THEN "∀E"(2),
                   THEN "→E", rotated, OF PossibleWorld.ψ]
          by blast
  ultimately AOT_have ‹w ⊨ ¬p›
    using "conj-dist-w:2"[unvarify p q, OF "log-prop-prop:2", OF "log-prop-prop:2",
                          THEN "≡E"(1), THEN "→E"]
    by blast
  AOT_hence ‹w ⊨ p & w ⊨ ¬p› using 0 "&I" by blast
  AOT_thus ‹p & ¬p›
    by (metis "coherent:1" "Conjunction Simplification"(1,2) "≡E"(4)
              "modus-tollens:1" "raa-cor:3")
qed

AOT_theorem "conj-dist-w:10": ‹∃w∃p((w ⊨ ◇p) & ¬◇w ⊨ p)›              (528.10)
proof -
  AOT_obtain w where w: ‹∀p (w ⊨ p ≡ p)›
    using "act-world:1" "PossibleWorld.∃E"[rotated] by meson
  AOT_obtain r where ‹¬r› and ‹◇r›
    using "cont-tf-thm:2" "cont-tf:2"[THEN "≡dfE"] "&E" "∃E"[rotated] by metis
  AOT_hence ‹w ⊨ ¬r› and 0: ‹w ⊨ ◇r›
    using w[THEN "∀E"(1), OF "log-prop-prop:2", THEN "≡E"(2)] by blast+
  AOT_hence ‹¬w ⊨ r› using "coherent:1"[THEN "≡E"(1)] by blast
  AOT_hence ‹¬◇w ⊨ r› by (metis "≡E"(4) "rigid-truth-at:2")
```

```
  AOT_hence ‹w ⊨ ◊r & ¬◊w ⊨ r› using 0 "&I" by blast
  AOT_hence ‹∃p (w ⊨ ◊p & ¬◊w ⊨ p)› by (rule "∃I")
  thus ?thesis by (rule "PossibleWorld.∃I")
qed

AOT_theorem "two-worlds-exist:1": ‹∃p(ContingentlyTrue(p)) → ∃w (¬Actual(w))›     (530.1)
proof(rule "→I")
  AOT_assume ‹∃p ContingentlyTrue(p)›
  then AOT_obtain p where ‹ContingentlyTrue(p)›
    using "∃E"[rotated] by blast
  AOT_hence p: ‹p & ◊¬p›
    by (metis "≡dfE" "cont-tf:1")
  AOT_hence ‹∃w w ⊨ ¬p›
    using "fund:1"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(1)] "&E" by blast
  then AOT_obtain w where w: ‹w ⊨ ¬p›
    using "PossibleWorld.∃E"[rotated] by meson
  AOT_have ‹¬Actual(w)›
  proof(rule "raa-cor:2")
    AOT_assume ‹Actual(w)›
    AOT_hence ‹w ⊨ p›
      using p[THEN "&E"(1)] actual[THEN "≡dfE", THEN "&E"(2)]
      by (metis "log-prop-prop:2" "raa-cor:3" "rule-ui:1" "→E" w)
    moreover AOT_have ‹¬(w ⊨ p)›
      by (metis "coherent:1" "≡E"(4) "reductio-aa:2" w)
    ultimately AOT_show ‹w ⊨ p & ¬(w ⊨ p)›
      using "&I" by blast
  qed
  AOT_thus ‹∃w ¬Actual(w)›
    by (rule "PossibleWorld.∃I")
qed


AOT_theorem "two-worlds-exist:2": ‹∃p(ContingentlyFalse(p)) → ∃w (¬Actual(w))›     (530.2)
proof(rule "→I")
  AOT_assume ‹∃p ContingentlyFalse(p)›
  then AOT_obtain p where ‹ContingentlyFalse(p)›
    using "∃E"[rotated] by blast
  AOT_hence p: ‹¬p & ◊p›
    by (metis "≡dfE" "cont-tf:2")
  AOT_hence ‹∃w w ⊨ p›
    using "fund:1"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(1)] "&E" by blast
  then AOT_obtain w where w: ‹w ⊨ p›
    using "PossibleWorld.∃E"[rotated] by meson
  moreover AOT_have ‹¬Actual(w)›
  proof(rule "raa-cor:2")
    AOT_assume ‹Actual(w)›
    AOT_hence ‹w ⊨ ¬p›
      using p[THEN "&E"(1)] actual[THEN "≡dfE", THEN "&E"(2)]
      by (metis "log-prop-prop:2" "raa-cor:3" "rule-ui:1" "→E" w)
    moreover AOT_have ‹¬(w ⊨ p)›
      using calculation by (metis "coherent:1" "≡E"(4) "reductio-aa:2")
    AOT_thus ‹w ⊨ p & ¬(w ⊨ p)›
      using "&I" w by metis
  qed
  AOT_thus ‹∃w ¬Actual(w)›
    by (rule "PossibleWorld.∃I")
qed

AOT_theorem "two-worlds-exist:3": ‹∃w ¬Actual(w)›     (530.3)
  using "cont-tf-thm:1" "two-worlds-exist:1" "→E" by blast

AOT_theorem "two-worlds-exist:4": ‹∃w∃w'(w ≠ w')›     (530.4)
proof -
```

```
  AOT_obtain w where w: ‹Actual(w)›
    using "act-world:2"[THEN "uniqueness:1"[THEN "≡dfE"],
                        THEN "cqt-further:5"[THEN "→E"]]
          "PossibleWorld.∃E"[rotated] "&E"
    by blast
  moreover AOT_obtain w' where w': ‹¬Actual(w')›
    using "two-worlds-exist:3" "PossibleWorld.∃E"[rotated] by meson
  AOT_have ‹¬(w = w')›
  proof(rule "raa-cor:2")
    AOT_assume ‹w = w'›
    AOT_thus ‹p & ¬p› for p
      using w w' "&E" by (metis "rule=E" "raa-cor:3")
  qed
  AOT_hence ‹w ≠ w'›
    by (metis "≡dfI" "=-infix")
  AOT_hence ‹∃w' w ≠ w'›
    by (rule "PossibleWorld.∃I")
  thus ?thesis
    by (rule "PossibleWorld.∃I")
qed

(* TODO: more theorems *)

AOT_theorem "w-rel:1": ‹[λx φ{x}]↓ → [λx w ⊨ φ{x}]↓›        (552.1)
proof(rule "→I")
  AOT_assume ‹[λx φ{x}]↓›
  AOT_hence ‹□[λx φ{x}]↓›
    by (metis "exist-nec" "→E")
  moreover AOT_have
    ‹□[λx φ{x}]↓ → □∀x∀y(∀F([F]x ≡ [F]y) → ((w ⊨ φ{x}) ≡ ( w ⊨ φ{y})))›
  proof (rule RM; rule "→I"; rule GEN; rule GEN; rule "→I")
    AOT_modally_strict {
      fix x y
      AOT_assume ‹[λx φ{x}]↓›
      AOT_hence ‹∀x∀y (∀F ([F]x ≡ [F]y) → □(φ{x} ≡ φ{y}))›
        using "&E" "kirchner-thm-cor:1"[THEN "→E"] by blast
      AOT_hence ‹∀F ([F]x ≡ [F]y) → □(φ{x} ≡ φ{y})›
        using "∀E"(2) by blast
      moreover AOT_assume ‹∀F ([F]x ≡ [F]y)›
      ultimately AOT_have ‹□(φ{x} ≡ φ{y})›
        using "→E" by blast
      AOT_hence ‹∀w (w ⊨ (φ{x} ≡ φ{y}))›
        using "fund:2"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(1)] by blast
      AOT_hence ‹w ⊨ (φ{x} ≡ φ{y})›
        using "∀E"(2) using PossibleWorld.ψ "→E" by blast
      AOT_thus ‹(w ⊨ φ{x}) ≡ (w ⊨ φ{y})›
        using "conj-dist-w:4"[unvarify p q, OF "log-prop-prop:2",
                              OF "log-prop-prop:2", THEN "≡E"(1)] by blast
    }
  qed
  ultimately AOT_have ‹□∀x∀y(∀F([F]x ≡ [F]y) → ((w ⊨ φ{x}) ≡ ( w ⊨ φ{y})))›
    using "→E" by blast
  AOT_thus ‹[λx w ⊨ φ{x}]↓›
    using "kirchner-thm:1"[THEN "≡E"(2)] by fast
qed

AOT_theorem "w-rel:2": ‹[λx1...xn φ{x1...xn}]↓ → [λx1...xn w ⊨ φ{x1...xn}]↓›        (552.2)
proof(rule "→I")
  AOT_assume ‹[λx1...xn φ{x1...xn}]↓›
  AOT_hence ‹□[λx1...xn φ{x1...xn}]↓›
    by (metis "exist-nec" "→E")
  moreover AOT_have ‹□[λx1...xn φ{x1...xn}]↓ → □∀x1...∀xn∀y1...∀yn(
    ∀F([F]x1...xn ≡ [F]y1...yn) → ((w ⊨ φ{x1...xn}) ≡ ( w ⊨ φ{y1...yn})))›
```

```
  proof (rule RM; rule "→I"; rule GEN; rule GEN; rule "→I")
    AOT_modally_strict {
      fix x1xn y1yn
      AOT_assume ‹[λx1...xn φ{x1...xn}]↓›
      AOT_hence ‹∀x1...∀xn∀y1...∀yn (
        ∀F ([F]x1...xn ≡ [F]y1...yn) → □(φ{x1...xn} ≡ φ{y1...yn}))›
        using "&E" "kirchner-thm-cor:2"[THEN "→E"] by blast
      AOT_hence ‹∀F ([F]x1...xn ≡ [F]y1...yn) → □(φ{x1...xn} ≡ φ{y1...yn})›
        using "∀E"(2) by blast
      moreover AOT_assume ‹∀F ([F]x1...xn ≡ [F]y1...yn)›
      ultimately AOT_have ‹□(φ{x1...xn} ≡ φ{y1...yn})›
        using "→E" by blast
      AOT_hence ‹∀w (w ⊨ (φ{x1...xn} ≡ φ{y1...yn}))›
        using "fund:2"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(1)] by blast
      AOT_hence ‹w ⊨ (φ{x1...xn} ≡ φ{y1...yn})›
        using "∀E"(2) using PossibleWorld.ψ "→E" by blast
      AOT_thus ‹(w ⊨ φ{x1...xn}) ≡ (w ⊨ φ{y1...yn})›
        using "conj-dist-w:4"[unvarify p q, OF "log-prop-prop:2",
                                OF "log-prop-prop:2", THEN "≡E"(1)] by blast
    }
  qed
  ultimately AOT_have ‹□∀x1...∀xn∀y1...∀yn(
      ∀F([F]x1...xn ≡ [F]y1...yn) → ((w ⊨ φ{x1...xn}) ≡ ( w ⊨ φ{y1...yn})))›
    using "→E" by blast
  AOT_thus ‹[λx1...xn w ⊨ φ{x1...xn}]↓›
    using "kirchner-thm:2"[THEN "≡E"(2)] by fast
qed

AOT_theorem "w-rel:3": ‹[λx1...xn w ⊨ [F]x1...xn]↓›                    (552.3)
  by (rule "w-rel:2"[THEN "→E"]) "cqt:2[lambda]"

AOT_define WorldIndexedRelation :: ‹Π ⇒ τ ⇒ Π› (‹_›)
  "w-index": ‹[F]w =df [λx1...xn w ⊨ [F]x1...xn]›                      (553)

AOT_define Rigid :: ‹τ ⇒ φ› (‹Rigid'(_')›)
  "df-rigid-rel:1":                                                     (554.1)
‹Rigid(F) ≡df F↓ & □∀x1...∀xn([F]x1...xn → □[F]x1...xn)›

AOT_define Rigidifies :: ‹τ ⇒ τ ⇒ φ› (‹Rigidifies'(_,_')›)
  "df-rigid-rel:2":                                                     (554.2)
‹Rigidifies(F, G) ≡df Rigid(F) & ∀x1...∀xn([F]x1...xn ≡ [G]x1...xn)›

AOT_theorem "rigid-der:1": ‹[[F]w]x1...xn ≡ w ⊨ [F]x1...xn›              (556.1)
  apply (rule "rule-id-df:2:b[2]"[where τ="λ (Π, κ). «[Π]κ»" and
                                  σ="λ(Π, κ). «[λx1...xn κ ⊨ [Π]x1...xn]»",
                            simplified, OF "w-index"])
   apply (fact "w-rel:3")
  apply (rule "beta-C-meta"[THEN "→E"])
  by (fact "w-rel:3")

AOT_theorem "rigid-der:2": ‹Rigid([G]w)›                                (556.2)
proof(safe intro!: "≡dfI"[OF "df-rigid-rel:1"] "&I")
  AOT_show ‹[G]w↓›
    by (rule "rule-id-df:2:b[2]"[where τ="λ (Π, κ). «[Π]κ»" and
                                  σ="λ(Π, κ). «[λx1...xn κ ⊨ [Π]x1...xn]»",
                            simplified, OF "w-index"])
         (fact "w-rel:3")+
next
  AOT_have ‹□∀x1...∀xn ([[G]w]x1...xn → □[[G]w]x1...xn)›
  proof(rule RN; safe intro!: "→I" GEN)
    AOT_modally_strict {
      AOT_have assms: ‹PossibleWorld(w)› using PossibleWorld.ψ.
      AOT_hence nec_pw_w: ‹□PossibleWorld(w)›
```

```
        using "≡E"(1) "rigid-pw:1" by blast
      fix x₁xₙ
      AOT_assume ‹[[G]w]x₁...xₙ›
      AOT_hence ‹[λx₁...xₙ w ⊨ [G]x₁...xₙ]x₁...xₙ›
        using "rule-id-df:2:a[2]"[where τ="λ (Π, κ). «[Π]κ»" and
                                        σ="λ(Π, κ). «[λx₁...xₙ κ ⊨ [Π]x₁...xₙ]»",
                                        simplified, OF "w-index", OF "w-rel:3"]
        by fast
      AOT_hence ‹w ⊨ [G]x₁...xₙ›
        by (metis "β→C"(1))
      AOT_hence ‹□w ⊨ [G]x₁...xₙ›
        using "rigid-truth-at:1"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(1)]
        by blast
      moreover AOT_have ‹□w ⊨ [G]x₁...xₙ → □[λx₁...xₙ w ⊨ [G]x₁...xₙ]x₁...xₙ›
      proof (rule RM; rule "→I")
        AOT_modally_strict {
          AOT_assume ‹w ⊨ [G]x₁...xₙ›
          AOT_thus ‹[λx₁...xₙ w ⊨ [G]x₁...xₙ]x₁...xₙ›
            by (auto intro!: "β←C"(1) simp: "w-rel:3" "cqt:2")
        }
      qed
      ultimately AOT_have 1: ‹□[λx₁...xₙ w ⊨ [G]x₁...xₙ]x₁...xₙ›
        using "→E" by blast
      AOT_show ‹□[[G]w]x₁...xₙ›
        by (rule "rule-id-df:2:b[2]"[where τ="λ (Π, κ). «[Π]κ»" and
                                          σ="λ(Π, κ). «[λx₁...xₙ κ ⊨ [Π]x₁...xₙ]»",
                                     simplified, OF "w-index"])
            (auto simp: 1 "w-rel:3")
    }
  qed
  AOT_thus ‹□∀x₁...∀xₙ ([[G]w]x₁...xₙ → □[[G]w]x₁...xₙ)›
    using "→E" by blast
qed

AOT_theorem "rigid-der:3": ‹∃F Rigidifies(F, G)›                              (556.3)
proof -
  AOT_obtain w where w: ‹∀p (w ⊨ p ≡ p)›
    using "act-world:1" "PossibleWorld.∃E"[rotated] by meson
  show ?thesis
  proof (rule "∃I"(1)[where τ=‹«[G]w»›])
    AOT_show ‹Rigidifies([G]w, [G])›
    proof(safe intro!: "≡dfI"[OF "df-rigid-rel:2"] "&I" GEN)
      AOT_show ‹Rigid([G]w)›
        using "rigid-der:2" by blast
    next
      fix x₁xₙ
      AOT_have ‹[[G]w]x₁...xₙ ≡ [λx₁...xₙ w ⊨ [G]x₁...xₙ]x₁...xₙ›
      proof(rule "≡I"; rule "→I")
        AOT_assume ‹[[G]w]x₁...xₙ›
        AOT_thus ‹[λx₁...xₙ w ⊨ [G]x₁...xₙ]x₁...xₙ›
          by (rule "rule-id-df:2:a[2]"
                      [where τ="λ (Π, κ). «[Π]κ»" and
                             σ="λ(Π, κ). «[λx₁...xₙ κ ⊨ [Π]x₁...xₙ]»",
                       simplified, OF "w-index", OF "w-rel:3"])
      next
        AOT_assume ‹[λx₁...xₙ w ⊨ [G]x₁...xₙ]x₁...xₙ›
        AOT_thus ‹[[G]w]x₁...xₙ›
          by (rule "rule-id-df:2:b[2]"
                      [where τ="λ (Π, κ). «[Π]κ»" and
                             σ="λ(Π, κ). «[λx₁...xₙ κ ⊨ [Π]x₁...xₙ]»",
                       simplified, OF "w-index", OF "w-rel:3"])
      qed
      also AOT_have ‹... ≡ w ⊨ [G]x₁...xₙ›
```

```
        by (rule "beta-C-meta"[THEN "→E"])
           (fact "w-rel:3")
      also AOT_have ‹... ≡ [G]x1...xn›
        using w[THEN "∀E"(1), OF "log-prop-prop:2"] by blast
      finally AOT_show ‹[[G]w]x1...xn ≡ [G]x1...xn›.
    qed
  next
    AOT_show ‹[G]w↓›
      by (rule "rule-id-df:2:b[2]"[where τ="λ (Π, κ). «[Π]κ»"
                                    and σ="λ(Π, κ). «[λx1...xn κ ⊨ [Π]x1...xn]»",
                                  simplified, OF "w-index"])
         (auto simp: "w-rel:3")
  qed
qed

AOT_theorem "rigid-rel-thms:1":                                                     (557.1)
  ‹□(∀x1...∀xn ([F]x1...xn → □[F]x1...xn)) ≡ ∀x1...∀xn(◇[F]x1...xn → □[F]x1...xn)›
proof(safe intro!: "≡I" "→I" GEN)
  fix x1xn
  AOT_assume ‹□∀x1...∀xn ([F]x1...xn → □[F]x1...xn)›
  AOT_hence ‹∀x1...∀xn □([F]x1...xn → □[F]x1...xn)›
    by (metis "→E" GEN RM "cqt-orig:3")
  AOT_hence ‹□([F]x1...xn → □[F]x1...xn)›
    using "∀E"(2) by blast
  AOT_hence ‹◇[F]x1...xn → □[F]x1...xn›
    by (metis "≡E"(1) "sc-eq-box-box:1")
  moreover AOT_assume ‹◇[F]x1...xn›
  ultimately AOT_show ‹□[F]x1...xn›
    using "→E" by blast
next
  AOT_assume ‹∀x1...∀xn (◇[F]x1...xn → □[F]x1...xn)›
  AOT_hence ‹◇[F]x1...xn → □[F]x1...xn› for x1xn
    using "∀E"(2) by blast
  AOT_hence ‹□([F]x1...xn → □[F]x1...xn)› for x1xn
    by (metis "≡E"(2) "sc-eq-box-box:1")
  AOT_hence 0: ‹∀x1...∀xn □([F]x1...xn → □[F]x1...xn)›
    by (rule GEN)
  AOT_thus ‹□(∀x1...∀xn ([F]x1...xn → □[F]x1...xn))›
    using "BF" "vdash-properties:10" by blast
qed

AOT_theorem "rigid-rel-thms:2":                                                     (557.2)
  ‹□(∀x1...∀xn ([F]x1...xn → □[F]x1...xn)) ≡ ∀x1...∀xn(□[F]x1...xn ∨ □¬[F]x1...xn)›
proof(safe intro!: "≡I" "→I")
  AOT_assume ‹□(∀x1...∀xn ([F]x1...xn → □[F]x1...xn))›
  AOT_hence 0: ‹∀x1...∀xn □([F]x1...xn → □[F]x1...xn)›
    using CBF[THEN "→E"] by blast
  AOT_show ‹∀x1...∀xn(□[F]x1...xn ∨ □¬[F]x1...xn)›
  proof(rule GEN)
    fix x1xn
    AOT_have 1: ‹□([F]x1...xn → □[F]x1...xn)›
      using 0[THEN "∀E"(2)].
    AOT_hence 2: ‹◇[F]x1...xn → [F]x1...xn›
      using "B◇" "Hypothetical Syllogism" "K◇" "vdash-properties:10" by blast
    AOT_have ‹[F]x1...xn ∨ ¬[F]x1...xn›
      using "exc-mid".
    moreover {
      AOT_assume ‹[F]x1...xn›
      AOT_hence ‹□[F]x1...xn›
        using 1[THEN "qml:2"[axiom_inst, THEN "→E"], THEN "→E"] by blast
    }
    moreover {
      AOT_assume 3: ‹¬[F]x1...xn›
```

```
      AOT_have ‹□¬[F]x1...xn›
      proof(rule "raa-cor:1")
        AOT_assume ‹¬□¬[F]x1...xn›
        AOT_hence ‹◇[F]x1...xn›
          by (AOT_subst_def "conventions:5")
        AOT_hence ‹[F]x1...xn› using 2[THEN "→E"] by blast
        AOT_thus ‹[F]x1...xn & ¬[F]x1...xn›
          using 3 "&I" by blast
      qed
    }
    ultimately AOT_show ‹□[F]x1...xn ∨ □¬[F]x1...xn›
      by (metis "∨I"(1,2) "raa-cor:1")
  qed
next
  AOT_assume 0: ‹∀x1...∀xn(□[F]x1...xn ∨ □¬[F]x1...xn)›
  {
    fix x1xn
    AOT_have ‹□[F]x1...xn ∨ □¬[F]x1...xn› using 0[THEN "∀E"(2)] by blast
    moreover {
      AOT_assume ‹□[F]x1...xn›
      AOT_hence ‹□□[F]x1...xn›
        using "S5Basic:6"[THEN "≡E"(1)] by blast
      AOT_hence ‹□([F]x1...xn → □[F]x1...xn)›
        using "KBasic:1"[THEN "→E"] by blast
    }
    moreover {
      AOT_assume ‹□¬[F]x1...xn›
      AOT_hence ‹□([F]x1...xn → □[F]x1...xn)›
        using "KBasic:2"[THEN "→E"] by blast
    }
    ultimately AOT_have ‹□([F]x1...xn → □[F]x1...xn)›
      using "con-dis-i-e:4:b" "raa-cor:1" by blast
  }
  AOT_hence ‹∀x1...∀xn □([F]x1...xn → □[F]x1...xn)›
    by (rule GEN)
  AOT_thus ‹□(∀x1...∀xn ([F]x1...xn → □[F]x1...xn))›
    using BF[THEN "→E"] by fast
qed

AOT_theorem "rigid-rel-thms:3": ‹Rigid(F) ≡ ∀x1...∀xn (□[F]x1...xn ∨ □¬[F]x1...xn)›        (557.3)
  by (AOT_subst_thm "df-rigid-rel:1"[THEN "≡Df", THEN "≡S"(1), OF "cqt:2"(1)];
      AOT_subst_thm "rigid-rel-thms:2")
     (simp add: "oth-class-taut:3:a")

(*<*)
end
(*>*)
```

A.12. Natural Numbers

```
(*<*)
theory AOT_NaturalNumbers
  imports AOT_PossibleWorlds AOT_ExtendedRelationComprehension
  abbrevs one-to-one = ‹1-1›
      and onto = ‹onto›
begin
(*>*)

section‹Natural Numbers›

AOT_define CorrelatesOneToOne :: ‹τ ⇒ τ ⇒ τ ⇒ φ› (‹_ |: _ 1-1⟷ _›)
  "1-1-cor": ‹R |: F 1-1⟷ G ≡df R↓ & F↓ & G↓ &                              (723)
                                  ∀x ([F]x → ∃!y([G]y & [R]xy)) &
                                  ∀y ([G]y → ∃!x([F]x & [R]xy))›

AOT_define MapsTo :: ‹τ ⇒ τ ⇒ τ ⇒ φ› (‹_ |: _ ⟶ _›)
  "fFG:1": ‹R |: F ⟶ G ≡df R↓ & F↓ & G↓ & ∀x ([F]x → ∃!y([G]y & [R]xy))›    (725.1)

AOT_define MapsToOneToOne :: ‹τ ⇒ τ ⇒ τ ⇒ φ› (‹_ |: _ 1-1⟶ _›)
  "fFG:2": ‹R |: F 1-1⟶ G ≡df                                               (725.2)
      R |: F ⟶ G & ∀x∀y∀z (([F]x & [F]y & [G]z) → ([R]xz & [R]yz → x = y))›

AOT_define MapsOnto :: ‹τ ⇒ τ ⇒ τ ⇒ φ› (‹_ |: _ ⟶onto _›)
  "fFG:3": ‹R |: F ⟶onto G ≡df R |: F ⟶ G & ∀y ([G]y → ∃x([F]x & [R]xy))›    (725.3)

AOT_define MapsOneToOneOnto :: ‹τ ⇒ τ ⇒ τ ⇒ φ› (‹_ |: _ 1-1⟶onto _›)
  "fFG:4": ‹R |: F 1-1⟶onto G ≡df R |: F 1-1⟶ G & R |: F ⟶onto G›           (725.4)

AOT_theorem "eq-1-1": ‹R |: F 1-1⟷ G ≡ R |: F 1-1⟶onto G›                   (726)
proof(rule "≡I"; rule "→I")
  AOT_assume ‹R |: F 1-1⟷ G›
  AOT_hence A: ‹∀x ([F]x → ∃!y([G]y & [R]xy))›
        and B: ‹∀y ([G]y → ∃!x([F]x & [R]xy))›
    using "≡dfE"[OF "1-1-cor"] "&E" by blast+
  AOT_have C: ‹R |: F ⟶ G›
  proof (rule "≡dfI"[OF "fFG:1"]; rule "&I")
    AOT_show ‹R↓ & F↓ & G↓›
      using "cqt:2[const_var]"[axiom_inst] "&I" by metis
  next
    AOT_show ‹∀x ([F]x → ∃!y([G]y & [R]xy))› by (rule A)
  qed
  AOT_show ‹R |: F 1-1⟶onto G›
  proof (rule "≡dfI"[OF "fFG:4"]; rule "&I")
    AOT_show ‹R |: F 1-1⟶ G›
    proof (rule "≡dfI"[OF "fFG:2"]; rule "&I")
      AOT_show ‹R |: F ⟶ G› using C.
    next
      AOT_show ‹∀x∀y∀z ([F]x & [F]y & [G]z → ([R]xz & [R]yz → x = y))›
      proof(rule GEN; rule GEN; rule GEN; rule "→I"; rule "→I")
        fix x y z
        AOT_assume 1: ‹[F]x & [F]y & [G]z›
        moreover AOT_assume 2: ‹[R]xz & [R]yz›
        ultimately AOT_have 3: ‹∃!x ([F]x & [R]xz)›
          using B "&E" "∀E" "→E" by fast
        AOT_show ‹x = y›
          by (rule "uni-most"[THEN "→E", OF 3, THEN "∀E"(2)[where β=x],
                              THEN "∀E"(2)[where β=y], THEN "→E"])
             (metis "&I" "&E" 1 2)
      qed
    qed
  next
```

```
    AOT_show ‹R |: F ⟶onto G›
    proof (rule "≡dfI"[OF "fFG:3"]; rule "&I")
      AOT_show ‹R |: F ⟶ G› using C.
    next
      AOT_show ‹∀y ([G]y → ∃x ([F]x & [R]xy))›
      proof(rule GEN; rule "→I")
        fix y
        AOT_assume ‹[G]y›
        AOT_hence ‹∃!x ([F]x & [R]xy)›
          using B[THEN "∀E"(2), THEN "→E"] by blast
        AOT_hence ‹∃x ([F]x & [R]xy & ∀β (([F]β & [R]βy) → β = x))›
          using "uniqueness:1"[THEN "≡dfE"] by blast
        then AOT_obtain x where ‹[F]x & [R]xy›
          using "∃E"[rotated] "&E" by blast
        AOT_thus ‹∃x ([F]x & [R]xy)› by (rule "∃I")
      qed
    qed
  qed
next
  AOT_assume ‹R |: F 1-1⟶onto G›
  AOT_hence ‹R |: F 1-1⟶ G› and ‹R |: F ⟶onto G›
    using "≡dfE"[OF "fFG:4"] "&E" by blast+
  AOT_hence C: ‹R |: F ⟶ G›
    and D: ‹∀x∀y∀z ([F]x & [F]y & [G]z → ([R]xz & [R]yz → x = y))›
    and E: ‹∀y ([G]y → ∃x ([F]x & [R]xy))›
    using "≡dfE"[OF "fFG:2"] "≡dfE"[OF "fFG:3"] "&E" by blast+
  AOT_show ‹R |: F 1-1⟷ G›
  proof(rule "1-1-cor"[THEN "≡dfI"]; safe intro!: "&I" "cqt:2[const_var]"[axiom_inst])
    AOT_show ‹∀x ([F]x → ∃!y ([G]y & [R]xy))›
      using "≡dfE"[OF "fFG:1", OF C] "&E" by blast
  next
    AOT_show ‹∀y ([G]y → ∃!x ([F]x & [R]xy))›
    proof (rule "GEN"; rule "→I")
      fix y
      AOT_assume 0: ‹[G]y›
      AOT_hence ‹∃x ([F]x & [R]xy)›
        using E "∀E" "→E" by fast
      then AOT_obtain a where a_prop: ‹[F]a & [R]ay›
        using "∃E"[rotated] by blast
      moreover AOT_have ‹∀z ([F]z & [R]zy → z = a)›
      proof (rule GEN; rule "→I")
        fix z
        AOT_assume ‹[F]z & [R]zy›
        AOT_thus ‹z = a›
          using D[THEN "∀E"(2)[where β=z], THEN "∀E"(2)[where β=a],
                  THEN "∀E"(2)[where β=y], THEN "→E", THEN "→E"]
                a_prop 0 "&E" "&I" by metis
      qed
      ultimately AOT_have ‹∃x ([F]x & [R]xy & ∀z ([F]z & [R]zy → z = x))›
        using "&I" "∃I"(2) by fast
      AOT_thus ‹∃!x ([F]x & [R]xy)›
        using "uniqueness:1"[THEN "≡dfI"] by fast
    qed
  qed
qed

text‹We have already introduced the restricted type of Ordinary objects in the
     Extended Relation Comprehension theory. However, make sure all variable names
     are defined as expected (avoiding conflicts with situations
     of possible world theory).›
AOT_register_variable_names
  Ordinary: u v r t s
```

```
AOT_theorem "equi:1": ‹∃!u φ{u} ≡ ∃u (φ{u} & ∀v (φ{v} → v =E u))›          (729.1)
proof(rule "≡I"; rule "→I")
  AOT_assume ‹∃!u φ{u}›
  AOT_hence ‹∃!x (O!x & φ{x})›.
  AOT_hence ‹∃x (O!x & φ{x} & ∀β (O!β & φ{β} → β = x))›
    using "uniqueness:1"[THEN "≡dfE"] by blast
  then AOT_obtain x where x_prop: ‹O!x & φ{x} & ∀β (O!β & φ{β} → β = x)›
    using "∃E"[rotated] by blast
  {
    fix β
    AOT_assume beta_ord: ‹O!β›
    moreover AOT_assume ‹φ{β}›
    ultimately AOT_have ‹β = x›
      using x_prop[THEN "&E"(2), THEN "∀E"(2)[where β=β]] "&I" "→E" by blast
    AOT_hence ‹β =E x›
      using "ord-=E=:1"[THEN "→E", OF "∨I"(1)[OF beta_ord],
                        THEN "qml:2"[axiom_inst, THEN "→E"],
                        THEN "≡E"(1)]
      by blast
  }
  AOT_hence ‹(O!β → (φ{β} → β =E x))› for β
    using "→I" by blast
  AOT_hence ‹∀β(O!β → (φ{β} → β =E x))›
    by (rule GEN)
  AOT_hence ‹O!x & φ{x} & ∀y (O!y → (φ{y} → y =E x))›
    using x_prop[THEN "&E"(1)] "&I" by blast
  AOT_hence ‹O!x & (φ{x} & ∀y (O!y → (φ{y} → y =E x)))›
    using "&E" "&I" by meson
  AOT_thus ‹∃u (φ{u} & ∀v (φ{v} → v =E u))›
    using "∃I" by fast
next
  AOT_assume ‹∃u (φ{u} & ∀v (φ{v} → v =E u))›
  AOT_hence ‹∃x (O!x & (φ{x} & ∀y (O!y → (φ{y} → y =E x))))›
    by blast
  then AOT_obtain x where x_prop: ‹O!x & (φ{x} & ∀y (O!y → (φ{y} → y =E x)))›
    using "∃E"[rotated] by blast
  AOT_have ‹∀y ([O!]y & φ{y} → y = x)›
  proof(rule GEN; rule "→I")
    fix y
    AOT_assume ‹O!y & φ{y}›
    AOT_hence ‹y =E x›
      using x_prop[THEN "&E"(2), THEN "&E"(2), THEN "∀E"(2)[where β=y]]
            "→E" "&E" by blast
    AOT_thus ‹y = x›
      using "ord-=E=:1"[THEN "→E", OF "∨I"(2)[OF x_prop[THEN "&E"(1)]],
                        THEN "qml:2"[axiom_inst, THEN "→E"], THEN "≡E"(2)] by blast
  qed
  AOT_hence ‹[O!]x & φ{x} & ∀y ([O!]y & φ{y} → y = x)›
    using x_prop "&E" "&I" by meson
  AOT_hence ‹∃x ([O!]x & φ{x} & ∀y ([O!]y & φ{y} → y = x))›
    by (rule "∃I")
  AOT_hence ‹∃!x (O!x & φ{x})›
    by (rule "uniqueness:1"[THEN "≡dfI"])
  AOT_thus ‹∃!u φ{u}›.
qed

AOT_define CorrelatesEOneToOne :: ‹τ ⇒ τ ⇒ τ ⇒ φ› (‹_ |: _ 1-1⟷E _›)
  "equi:2": ‹R |: F 1-1⟷E G ≡df R↓ & F↓ & G↓ &          (729.2)
                              ∀u ([F]u → ∃!v([G]v & [R]uv)) &
                              ∀v ([G]v → ∃!u([F]u & [R]uv))›

AOT_define EquinumerousE :: ‹τ ⇒ τ ⇒ φ› (infixl "≈E" 50)
  "equi:3": ‹F ≈E G ≡df ∃R (R |: F 1-1⟷E G)›          (729.3)
```

```
text‹Note: not explicitly in PLM.›
AOT_theorem eq_den_1: ‹Π↓› if ‹Π ≈E Π'›
proof -
  AOT_have ‹∃R (R |: Π 1-1⟷E Π')›
    using "equi:3"[THEN "≡dfE"] that by blast
  then AOT_obtain R where ‹R |: Π 1-1⟷E Π'›
    using "∃E"[rotated] by blast
  AOT_thus ‹Π↓›
    using "equi:2"[THEN "≡dfE"] "&E" by blast
qed

text‹Note: not explicitly in PLM.›
AOT_theorem eq_den_2: ‹Π'↓› if ‹Π ≈E Π'›
proof -
  AOT_have ‹∃R (R |: Π 1-1⟷E Π')›
    using "equi:3"[THEN "≡dfE"] that by blast
  then AOT_obtain R where ‹R |: Π 1-1⟷E Π'›
    using "∃E"[rotated] by blast
  AOT_thus ‹Π'↓›
    using "equi:2"[THEN "≡dfE"] "&E" by blast+
qed

AOT_theorem "eq-part:1": ‹F ≈E F›                                        (730.1)
proof (safe intro!: "&I" GEN "→I" "cqt:2[const_var]"[axiom_inst]
                    "≡dfI"[OF "equi:3"] "≡dfI"[OF "equi:2"] "∃I"(1))
  fix x
  AOT_assume 1: ‹O!x›
  AOT_assume 2: ‹[F]x›
  AOT_show ‹∃!v ([F]v & x =E v)›
  proof(rule "equi:1"[THEN "≡E"(2)];
        rule "∃I"(2)[where β=x];
        safe dest!: "&E"(2)
             intro!:  "&I" "→I" 1 2 Ordinary.GEN "ord=Eequiv:1"[THEN "→E", OF 1])
    AOT_show ‹v =E x› if ‹x =E v› for v
      by (metis that "ord=Eequiv:2"[THEN "→E"])
  qed
next
  fix y
  AOT_assume 1: ‹O!y›
  AOT_assume 2: ‹[F]y›
  AOT_show ‹∃!u ([F]u & u =E y)›
    by(safe dest!: "&E"(2)
            intro!: "equi:1"[THEN "≡E"(2)] "∃I"(2)[where β=y]
                    "&I" "→I" 1 2 GEN "ord=Eequiv:1"[THEN "→E", OF 1])
qed(auto simp: "=E[denotes]")


AOT_theorem "eq-part:2": ‹F ≈E G → G ≈E F›                              (730.2)
proof (rule "→I")
  AOT_assume ‹F ≈E G›
  AOT_hence ‹∃R R |: F 1-1⟷E G›
    using "equi:3"[THEN "≡dfE"] by blast
  then AOT_obtain R where ‹R |: F 1-1⟷E G›
    using "∃E"[rotated] by blast
  AOT_hence 0: ‹R↓ & F↓ & G↓ & ∀u ([F]u → ∃!v([G]v & [R]uv)) &
                         ∀v ([G]v → ∃!u([F]u & [R]uv))›
    using "equi:2"[THEN "≡dfE"] by blast

  AOT_have ‹[λxy [R]yx]↓ & G↓ & F↓ & ∀u ([G]u → ∃!v([F]v & [λxy [R]yx]uv)) &
                         ∀v ([F]v → ∃!u([G]u & [λxy [R]yx]uv))›
  proof (AOT_subst ‹[λxy [R]yx]yx› ‹[R]xy› for: x y;
         (safe intro!: "&I" "cqt:2[const_var]"[axiom_inst] 0[THEN "&E"(2)]
```

```
                      O[THEN "&E"(1), THEN "&E"(2)]; "cqt:2[lambda]")?)
    AOT_modally_strict {
      AOT_have ‹[λxy [R]yx]xy› if ‹[R]yx› for y x
        by (auto intro!: "β←C"(1) "cqt:2"
                 simp: "&I" "ex:1:a" prod_denotesI "rule-ui:3" that)
      moreover AOT_have ‹[R]yx› if ‹[λxy [R]yx]xy› for y x
        using "β→C"(1)[where φ="λ(x,y). _ (x,y)" and κ1κn="(_,_)",
                       simplified, OF that, simplified].
      ultimately AOT_show ‹[λxy [R]yx]αβ ≡ [R]βα› for α β
        by (metis "deduction-theorem" "≡I")
    }
  qed
  AOT_hence ‹[λxy [R]yx] |: G 1-1⟷E F›
    using "equi:2"[THEN "≡dfI"] by blast
  AOT_hence ‹∃R R |: G 1-1⟷E F›
    by (rule "∃I"(1)) "cqt:2[lambda]"
  AOT_thus ‹G ≈E F›
    using "equi:3"[THEN "≡dfI"] by blast
qed

text‹Note: not explicitly in PLM.›
AOT_theorem "eq-part:2[terms]": ‹Π ≈E Π' → Π' ≈E Π›                    (730.2)
  using "eq-part:2"[unvarify F G] eq_den_1 eq_den_2 "→I" by meson
declare "eq-part:2[terms]"[THEN "→E", sym]

AOT_theorem "eq-part:3": ‹(F ≈E G & G ≈E H) → F ≈E H›                 (730.3)
proof (rule "→I")
  AOT_assume ‹F ≈E G & G ≈E H›
  then AOT_obtain R1 and R2 where
       ‹R1 |: F 1-1⟷E G›
   and ‹R2 |: G 1-1⟷E H›
    using "equi:3"[THEN "≡dfE"] "&E" "∃E"[rotated] by metis
  AOT_hence ϑ: ‹∀u ([F]u → ∃!v([G]v & [R1]uv)) & ∀v ([G]v → ∃!u([F]u & [R1]uv))›
        and ξ: ‹∀u ([G]u → ∃!v([H]v & [R2]uv)) & ∀v ([H]v → ∃!u([G]u & [R2]uv))›
    using "equi:2"[THEN "≡dfE", THEN "&E"(2)]
          "equi:2"[THEN "≡dfE", THEN "&E"(1), THEN "&E"(2)]
          "&I" by blast+
  AOT_have ‹∃R R = [λxy O!x & O!y & ∃v ([G]v & [R1]xv & [R2]vy)]›
    by (rule "free-thms:3[lambda]") cqt_2_lambda_inst_prover
  then AOT_obtain R where R_def: ‹R = [λxy O!x & O!y & ∃v ([G]v & [R1]xv & [R2]vy)]›
    using "∃E"[rotated] by blast
  AOT_have 1: ‹∃!v (([H]v & [R]uv))› if a: ‹[O!]u› and b: ‹[F]u› for u
  proof (rule "≡E"(2)[OF "equi:1"])
    AOT_obtain b where
      b_prop: ‹[O!]b & ([G]b & [R1]ub & ∀v ([G]v & [R1]uv → v =E b))›
      using ϑ[THEN "&E"(1), THEN "∀E"(2), THEN "→E", THEN "→E",
                OF a b, THEN "≡E"(1)[OF "equi:1"]]
              "∃E"[rotated] by blast
    AOT_obtain c where
      c_prop: "[O!]c & ([H]c & [R2]bc & ∀v ([H]v & [R2]bv → v =E c))"
      using ξ[THEN "&E"(1), THEN "∀E"(2)[where β=b], THEN "→E",
                OF b_prop[THEN "&E"(1)], THEN "→E",
                OF b_prop[THEN "&E"(2), THEN "&E"(1), THEN "&E"(1)],
                THEN "≡E"(1)[OF "equi:1"]]
    "∃E"[rotated] by blast
    AOT_show ‹∃v ([H]v & [R]uv & ∀v' ([H]v' & [R]uv' → v' =E v))›
    proof (safe intro!: "&I" GEN "→I" "∃I"(2)[where β=c])
      AOT_show ‹O!c› using c_prop "&E" by blast
    next
      AOT_show ‹[H]c› using c_prop "&E" by blast
    next
      AOT_have 0: ‹[O!]u & [O!]c & ∃v ([G]v & [R1]uv & [R2]vc)›
        by (safe intro!: "&I" a c_prop[THEN "&E"(1)] "∃I"(2)[where β=b]
```

```
                        b_prop[THEN "&E"(1)] b_prop[THEN "&E"(2), THEN "&E"(1)]
                        c_prop[THEN "&E"(2), THEN "&E"(1), THEN "&E"(2)])
      AOT_show ‹[R]uc›
        by (auto intro: "rule=E"[rotated, OF R_def[symmetric]]
                 intro!: "β←C"(1) "cqt:2"
                 simp: "&I" "ex:1:a" prod_denotesI "rule-ui:3" 0)
    next
      fix x
      AOT_assume ordx: ‹O!x›
      AOT_assume ‹[H]x & [R]ux›
      AOT_hence hx: ‹[H]x› and ‹[R]ux› using "&E" by blast+
      AOT_hence ‹[λxy O!x & O!y & ∃v ([G]v & [R₁]xv & [R₂]vy)]ux›
        using "rule=E"[rotated, OF R_def] by fast
      AOT_hence ‹O!u & O!x & ∃v ([G]v & [R₁]uv & [R₂]vx)›
        by (rule "β→C"(1)[where φ="λ(κ,κ'). _ κ κ'" and κ₁κₙ="(_,_)", simplified])
      then AOT_obtain z where z_prop: ‹O!z & ([G]z & [R₁]uz & [R₂]zx)›
        using "&E" "∃E"[rotated] by blast
      AOT_hence ‹z =E b›
        using b_prop[THEN "&E"(2), THEN "&E"(2), THEN "∀E"(2)[where β=z]]
        using "&E" "→E" by metis
      AOT_hence ‹z = b›
        by (metis "=E-simple:2"[THEN "→E"])
      AOT_hence ‹[R₂]bx›
        using z_prop[THEN "&E"(2), THEN "&E"(2)] "rule=E" by fast
      AOT_thus ‹x =E c›
        using c_prop[THEN "&E"(2), THEN "&E"(2), THEN "∀E"(2)[where β=x],
                    THEN "→E", THEN "→E", OF ordx]
              hx "&I" by blast
    qed
  qed
  AOT_have 2: ‹∃!u (([F]u & [R]uv))› if a: ‹[O!]v› and b: ‹[H]v› for v
  proof (rule "≡E"(2)[OF "equi:1"])
    AOT_obtain b where
      b_prop: ‹[O!]b & ([G]b & [R₂]bv & ∀u ([G]u & [R₂]uv → u =E b))›
      using ξ[THEN "&E"(2), THEN "∀E"(2), THEN "→E", THEN "→E",
              OF a b, THEN "≡E"(1)[OF "equi:1"]]
            "∃E"[rotated] by blast
    AOT_obtain c where
      c_prop: "[O!]c & ([F]c & [R₁]cb & ∀v ([F]v & [R₁]vb → v =E c))"
      using ϑ[THEN "&E"(2), THEN "∀E"(2)[where β=b], THEN "→E",
              OF b_prop[THEN "&E"(1)], THEN "→E",
              OF b_prop[THEN "&E"(2), THEN "&E"(1), THEN "&E"(1)],
              THEN "≡E"(1)[OF "equi:1"]]
    "∃E"[rotated] by blast
    AOT_show ‹∃u ([F]u & [R]uv & ∀v' ([F]v' & [R]v'v → v' =E u))›
    proof (safe intro!: "&I" GEN "→I" "∃I"(2)[where β=c])
      AOT_show ‹O!c› using c_prop "&E" by blast
    next
      AOT_show ‹[F]c› using c_prop "&E" by blast
    next
      AOT_have ‹[O!]c & [O!]v & ∃u ([G]u & [R₁]cu & [R₂]uv)›
        by (safe intro!: "&I" a "∃I"(2)[where β=b]
                    c_prop[THEN "&E"(1)] b_prop[THEN "&E"(1)]
                    b_prop[THEN "&E"(2), THEN "&E"(1), THEN "&E"(1)]
                    b_prop[THEN "&E"(2), THEN "&E"(1), THEN "&E"(2)]
                    c_prop[THEN "&E"(2), THEN "&E"(1), THEN "&E"(2)])
      AOT_thus ‹[R]cv›
        by (auto intro: "rule=E"[rotated, OF R_def[symmetric]]
                 intro!: "β←C"(1) "cqt:2"
                 simp: "&I" "ex:1:a" prod_denotesI "rule-ui:3")
    next
      fix x
      AOT_assume ordx: ‹O!x›
```

```
        AOT_assume ‹[F]x & [R]xv›
        AOT_hence hx: ‹[F]x› and ‹[R]xv› using "&E" by blast+
        AOT_hence ‹[λxy O!x & O!y & ∃v ([G]v & [R1]xv & [R2]vy)]xv›
          using "rule=E"[rotated, OF R_def] by fast
        AOT_hence ‹O!x & O!v & ∃u ([G]u & [R1]xu & [R2]uv)›
          by (rule "β→C"(1)[where φ="λ(κ,κ'). _ κ κ'" and κ1κn="(_,_)", simplified])
        then AOT_obtain z where z_prop: ‹O!z & ([G]z & [R1]xz & [R2]zv)›
          using "&E" "∃E"[rotated] by blast
        AOT_hence ‹z =E b›
          using b_prop[THEN "&E"(2), THEN "&E"(2), THEN "∀E"(2)[where β=z]]
          using "&E" "→E" "&I" by metis
        AOT_hence ‹z = b›
          by (metis "=E-simple:2"[THEN "→E"])
        AOT_hence ‹[R1]xb›
          using z_prop[THEN "&E"(2), THEN "&E"(1), THEN "&E"(2)] "rule=E" by fast
        AOT_thus ‹x =E c›
          using c_prop[THEN "&E"(2), THEN "&E"(2), THEN "∀E"(2)[where β=x],
                       THEN "→E", THEN "→E", OF ordx]
                hx "&I" by blast
      qed
    qed
    AOT_show ‹F ≈E H›
      apply (rule "equi:3"[THEN "≡dfI"])
      apply (rule "∃I"(2)[where β=R])
      by (auto intro!: 1 2 "equi:2"[THEN "≡dfI"] "&I" "cqt:2[const_var]"[axiom_inst]
                       Ordinary.GEN "→I" Ordinary.ψ)
qed

text‹Note: not explicitly in PLM.›
AOT_theorem "eq-part:3[terms]": ‹Π ≈E Π''› if ‹Π ≈E Π'› and ‹Π' ≈E Π''›             (730.3)
  using "eq-part:3"[unvarify F G H, THEN "→E"] eq_den_1 eq_den_2 "→I" "&I"
  by (metis that(1) that(2))
declare "eq-part:3[terms]"[trans]

AOT_theorem "eq-part:4": ‹F ≈E G ≡ ∀H (H ≈E F ≡ H ≈E G)›                           (730.4)
proof(rule "≡I"; rule "→I")
  AOT_assume 0: ‹F ≈E G›
  AOT_hence 1: ‹G ≈E F› using "eq-part:2"[THEN "→E"] by blast
  AOT_show ‹∀H (H ≈E F ≡ H ≈E G)›
  proof (rule GEN; rule "≡I"; rule "→I")
    AOT_show ‹H ≈E G› if ‹H ≈E F› for H using 0
      by (meson "&I" "eq-part:3" that "vdash-properties:6")
  next
    AOT_show ‹H ≈E F› if ‹H ≈E G› for H using 1
      by (metis "&I" "eq-part:3" that "vdash-properties:6")
  qed
next
  AOT_assume ‹∀H (H ≈E F ≡ H ≈E G)›
  AOT_hence ‹F ≈E F ≡ F ≈E G› using "∀E" by blast
  AOT_thus ‹F ≈E G› using "eq-part:1" "≡E" by blast
qed

AOT_define MapsE :: ‹τ ⇒ τ ⇒ τ ⇒ φ› ("_ |: _ ⟶E _")
  "equi-rem:1":                                                                      (731.1)
  ‹R |: F ⟶E G ≡df R↓ & F↓ & G↓ & ∀u ([F]u → ∃!v ([G]v & [R]uv))›

AOT_define MapsEOneToOne :: ‹τ ⇒ τ ⇒ τ ⇒ φ› ("_ |: _ 1-1⟶E _")
  "equi-rem:2":                                                                      (731.2)
  ‹R |: F 1-1⟶E G ≡df
      R |: F ⟶E G & ∀t∀u∀v (([F]t & [F]u & [G]v) → ([R]tv & [R]uv → t =E u))›

AOT_define MapsEOnto :: ‹τ ⇒ τ ⇒ τ ⇒ φ› ("_ |: _ ⟶ontoE _")
  "equi-rem:3":                                                                      (731.3)
```

```
  ‹R |: F ⟶ontoE G ≡df R |: F ⟶E G & ∀v ([G]v → ∃u ([F]u & [R]uv))›

AOT_define MapsEOneToOneOnto :: ‹τ ⇒ τ ⇒ τ ⇒ φ› ("_ |: _ 1-1⟶ontoE _")
  "equi-rem:4":                                                    (731.4)
  ‹R |: F 1-1⟶ontoE G ≡df R |: F 1-1⟶E G & R |: F ⟶ontoE G›

AOT_theorem "equi-rem-thm":                                        (732)
  ‹R |: F 1-1⟷E G ≡ R |: F 1-1⟶ontoE G›
proof -
  AOT_have ‹R |: F 1-1⟷E G ≡ R |: [λx O!x & [F]x] 1-1⟷ [λx O!x & [G]x]›
  proof(safe intro!: "≡I" "→I" "&I")
    AOT_assume ‹R |: F 1-1⟷E G›
    AOT_hence ‹∀u ([F]u → ∃!v ([G]v & [R]uv))›
          and ‹∀v ([G]v → ∃!u ([F]u & [R]uv))›
      using "equi:2"[THEN "≡dfE"] "&E" by blast+
    AOT_hence a: ‹([F]u → ∃!v ([G]v & [R]uv))›
          and b: ‹([G]v → ∃!u ([F]u & [R]uv))› for u v
      using "Ordinary.∀E" by fast+
    AOT_have ‹([λx [O!]x & [F]x]x → ∃!y ([λx [O!]x & [G]x]y & [R]xy))› for x
      apply (AOT_subst ‹[λx [O!]x & [F]x]x› ‹[O!]x & [F]x›)
       apply (rule "beta-C-meta"[THEN "→E"])
       apply "cqt:2[lambda]"
      apply (AOT_subst ‹[λx [O!]x & [G]x]x› ‹[O!]x & [G]x› for: x)
       apply (rule "beta-C-meta"[THEN "→E"])
       apply "cqt:2[lambda]"
      apply (AOT_subst ‹O!y & [G]y & [R]xy› ‹O!y & ([G]y & [R]xy)› for: y)
       apply (meson "≡E"(6) "Associativity of &" "oth-class-taut:3:a")
      apply (rule "→I") apply (frule "&E"(1)) apply (drule "&E"(2))
      by (fact a[unconstrain u, THEN "→E", THEN "→E", of x])
    AOT_hence A: ‹∀x ([λx [O!]x & [F]x]x → ∃!y ([λx [O!]x & [G]x]y & [R]xy))›
      by (rule GEN)
    AOT_have ‹([λx [O!]x & [G]x]y → ∃!x ([λx [O!]x & [F]x]x & [R]xy))› for y
      apply (AOT_subst ‹[λx [O!]x & [G]x]y› ‹[O!]y & [G]y›)
       apply (rule "beta-C-meta"[THEN "→E"])
       apply "cqt:2[lambda]"
      apply (AOT_subst ‹[λx [O!]x & [F]x]x› ‹[O!]x & [F]x› for: x)
       apply (rule "beta-C-meta"[THEN "→E"])
       apply "cqt:2[lambda]"
      apply (AOT_subst ‹O!x & [F]x & [R]xy› ‹O!x & ([F]x & [R]xy)› for: x)
       apply (meson "≡E"(6) "Associativity of &" "oth-class-taut:3:a")
      apply (rule "→I") apply (frule "&E"(1)) apply (drule "&E"(2))
      by (fact b[unconstrain v, THEN "→E", THEN "→E", of y])
    AOT_hence B: ‹∀y ([λx [O!]x & [G]x]y → ∃!x ([λx [O!]x & [F]x]x & [R]xy))›
      by (rule GEN)
    AOT_show ‹R |: [λx [O!]x & [F]x] 1-1⟷ [λx [O!]x & [G]x]›
      by (safe intro!: "1-1-cor"[THEN "≡dfI"] "&I"
                       "cqt:2[const_var]"[axiom_inst] A B)
         "cqt:2[lambda]"+
  next
    AOT_assume ‹R |: [λx [O!]x & [F]x] 1-1⟷ [λx [O!]x & [G]x]›
    AOT_hence a: ‹([λx [O!]x & [F]x]x → ∃!y ([λx [O!]x & [G]x]y & [R]xy))› and
              b: ‹([λx [O!]x & [G]x]y → ∃!x ([λx [O!]x & [F]x]x & [R]xy))› for x y
      using "1-1-cor"[THEN "≡dfE"] "&E" "∀E"(2) by blast+
    AOT_have ‹[F]u → ∃!v ([G]v & [R]uv)› for u
    proof (safe intro!: "→I")
      AOT_assume fu: ‹[F]u›
      AOT_have 0: ‹[λx [O!]x & [F]x]u›
        by (auto intro!: "β←C"(1) "cqt:2" "cqt:2[const_var]"[axiom_inst]
                         Ordinary.ψ fu "&I")
      AOT_show ‹∃!v ([G]v & [R]uv)›
        apply (AOT_subst ‹[O!]x & ([G]x & [R]ux)›
                         ‹([O!]x & [G]x) & [R]ux› for: x)
         apply (simp add: "Associativity of &")
```

```
        apply (AOT_subst (reverse) ‹[O!]x & [G]x›
                                   ‹[λx [O!]x & [G]x]x› for: x)
         apply (rule "beta-C-meta"[THEN "→E"])
         apply "cqt:2[lambda]"
        using a[THEN "→E", OF 0] by blast
    qed
    AOT_hence A: ‹∀u ([F]u → ∃!v ([G]v & [R]uv))›
      by (rule Ordinary.GEN)
    AOT_have ‹[G]v → ∃!u ([F]u & [R]uv)› for v
    proof (safe intro!: "→I")
      AOT_assume gu: ‹[G]v›
      AOT_have 0: ‹[λx [O!]x & [G]x]v›
        by (auto intro!: "β←C"(1) "cqt:2" "cqt:2[const_var]"[axiom_inst]
                         Ordinary.ψ gu "&I")
      AOT_show ‹∃!u ([F]u & [R]uv)›
        apply (AOT_subst ‹[O!]x & ([F]x & [R]xv)› ‹([O!]x & [F]x) & [R]xv› for: x)
         apply (simp add: "Associativity of &")
        apply (AOT_subst (reverse) ‹[O!]x & [F]x›‹[λx [O!]x & [F]x]x›  for: x)
         apply (rule "beta-C-meta"[THEN "→E"])
         apply "cqt:2[lambda]"
        using b[THEN "→E", OF 0] by blast
    qed
    AOT_hence B: ‹∀v ([G]v → ∃!u ([F]u & [R]uv))› by (rule Ordinary.GEN)
    AOT_show ‹R |: F 1-1⟷E G›
      by (safe intro!: "equi:2"[THEN "≡dfI"] "&I" A B "cqt:2[const_var]"[axiom_inst])
  qed
  also AOT_have ‹... ≡ R |: F 1-1⟶ontoE G›
  proof(safe intro!: "≡I" "→I" "&I")
    AOT_assume ‹R |: [λx [O!]x & [F]x] 1-1⟷ [λx [O!]x & [G]x]›
    AOT_hence a: ‹([λx [O!]x & [F]x]x → ∃!y ([λx [O!]x & [G]x]y & [R]xy))› and
              b: ‹([λx [O!]x & [G]x]y → ∃!x ([λx [O!]x & [F]x]x & [R]xy))› for x y
      using "1-1-cor"[THEN "≡dfE"] "&E" "∀E"(2) by blast+
    AOT_show ‹R |: F 1-1⟶ontoE G›
    proof (safe intro!: "equi-rem:4"[THEN "≡dfI"] "&I" "equi-rem:3"[THEN "≡dfI"]
                        "equi-rem:2"[THEN "≡dfI"] "equi-rem:1"[THEN "≡dfI"]
                        "cqt:2[const_var]"[axiom_inst] Ordinary.GEN "→I")
      fix u
      AOT_assume fu: ‹[F]u›
      AOT_have 0: ‹[λx [O!]x & [F]x]u›
        by (auto intro!: "β←C"(1) "cqt:2" "cqt:2[const_var]"[axiom_inst]
                         Ordinary.ψ fu "&I")
      AOT_hence 1: ‹∃!y ([λx [O!]x & [G]x]y & [R]uy)›
        using a[THEN "→E"] by blast
      AOT_show ‹∃!v ([G]v & [R]uv)›
        apply (AOT_subst ‹[O!]x & ([G]x & [R]ux)› ‹([O!]x & [G]x) & [R]ux› for: x)
         apply (simp add: "Associativity of &")
        apply (AOT_subst (reverse) ‹[O!]x & [G]x› ‹[λx [O!]x & [G]x]x› for: x)
         apply (rule "beta-C-meta"[THEN "→E"])
         apply "cqt:2[lambda]"
        by (fact 1)
    next
      fix t u v
      AOT_assume ‹[F]t & [F]u & [G]v› and rtv_tuv: ‹[R]tv & [R]uv›
      AOT_hence oft: ‹[λx O!x & [F]x]t› and
                ofu: ‹[λx O!x & [F]x]u› and
                ogv: ‹[λx O!x & [G]x]v›
        by (auto intro!: "β←C"(1) "cqt:2" "&I"
                 simp: Ordinary.ψ dest: "&E")
      AOT_hence ‹∃!x ([λx [O!]x & [F]x]x & [R]xv)›
        using b[THEN "→E"] by blast
      then AOT_obtain a where
          a_prop: ‹[λx [O!]x & [F]x]a & [R]av &
                   ∀x (([λx [O!]x & [F]x]x & [R]xv) → x = a)›
```

```
        using "uniqueness:1"[THEN "≡dfE"] "∃E"[rotated] by blast
      AOT_hence ua: ‹u = a›
        using ofu rtv_tuv[THEN "&E"(2)] "∀E"(2) "→E" "&I" "&E"(2) by blast
      moreover AOT_have ta: ‹t = a›
        using a_prop oft rtv_tuv[THEN "&E"(1)] "∀E"(2) "→E" "&I" "&E"(2) by blast
      ultimately AOT_have ‹t = u› by (metis "rule=E" id_sym)
      AOT_thus ‹t =E u›
        using "rule=E" id_sym "ord=Eequiv:1" Ordinary.ψ ta ua "→E" by fast
    next
      fix u
      AOT_assume ‹[F]u›
      AOT_hence ‹[λx O!x & [F]x]u›
        by (auto intro!: "β←C"(1) "cqt:2" "&I"
                 simp: "cqt:2[const_var]"[axiom_inst]  Ordinary.ψ)
      AOT_hence ‹∃!y ([λx [O!]x & [G]x]y & [R]uy)›
        using a[THEN "→E"] by blast
      then AOT_obtain a where
        a_prop: ‹[λx [O!]x & [G]x]a & [R]ua &
                 ∀x (([λx [O!]x & [G]x]x & [R]ux) → x = a)›
        using "uniqueness:1"[THEN "≡dfE"] "∃E"[rotated] by blast
      AOT_have ‹O!a & [G]a›
        by (rule "β→C"(1)) (auto simp: a_prop[THEN "&E"(1), THEN "&E"(1)])
      AOT_hence ‹O!a› and ‹[G]a› using "&E" by blast+
      moreover AOT_have ‹∀v ([G]v & [R]uv → v =E a)›
      proof(safe intro!: Ordinary.GEN "→I"; frule "&E"(1); drule "&E"(2))
        fix v
        AOT_assume ‹[G]v› and ruv: ‹[R]uv›
        AOT_hence ‹[λx [O!]x & [G]x]v›
          by (auto intro!: "β←C"(1) "cqt:2" "&I" simp: Ordinary.ψ)
        AOT_hence ‹v = a›
          using a_prop[THEN "&E"(2), THEN "∀E"(2), THEN "→E", OF "&I"] ruv by blast
        AOT_thus ‹v =E a›
          using "rule=E" "ord=Eequiv:1" Ordinary.ψ "→E" by fast
      qed
      ultimately AOT_have ‹O!a & ([G]a & [R]ua & ∀v' ([G]v' & [R]uv' → v' =E a))›
        using "∃I" "&I" a_prop[THEN "&E"(1), THEN "&E"(2)] by simp
      AOT_hence ‹∃v ([G]v & [R]uv & ∀v' ([G]v' & [R]uv' → v' =E v))›
        by (rule "∃I")
      AOT_thus ‹∃!v ([G]v & [R]uv)›
        by (rule "equi:1"[THEN "≡E"(2)])
    next
      fix v
      AOT_assume ‹[G]v›
      AOT_hence ‹[λx O!x & [G]x]v›
        by (auto intro!: "β←C"(1) "cqt:2" "&I" Ordinary.ψ)
      AOT_hence ‹∃!x ([λx [O!]x & [F]x]x & [R]xv)›
        using b[THEN "→E"] by blast
      then AOT_obtain a where
        a_prop: ‹[λx [O!]x & [F]x]a & [R]av &
                 ∀y ([λx [O!]x & [F]x]y & [R]yv → y = a)›
        using "uniqueness:1"[THEN "≡dfE", THEN "∃E"[rotated]] by blast
      AOT_have ‹O!a & [F]a›
        by (rule "β→C"(1)) (auto simp: a_prop[THEN "&E"(1), THEN "&E"(1)])
      AOT_hence ‹O!a & ([F]a & [R]av)›
        using a_prop[THEN "&E"(1), THEN "&E"(2)] "&E" "&I" by metis
      AOT_thus ‹∃u ([F]u & [R]uv)›
        by (rule "∃I")
    qed
  next
    AOT_assume ‹R |: F 1-1⟶ontoE G›
    AOT_hence 1: ‹R |: F 1-1⟶E G›
        and 2: ‹R |: F ⟶ontoE G›
      using "equi-rem:4"[THEN "≡dfE"] "&E" by blast+
```

```
AOT_hence 3: ‹R |: F ⟶E G›
      and A: ‹∀t ∀u ∀v ([F]t & [F]u & [G]v → ([R]tv & [R]uv → t =E u))›
  using "equi-rem:2"[THEN "≡dfE", OF 1] "&E" by blast+
AOT_hence B: ‹∀u ([F]u → ∃!v ([G]v & [R]uv))›
  using "equi-rem:1"[THEN "≡dfE"] "&E" by blast
AOT_have C: ‹∀v ([G]v → ∃u ([F]u & [R]uv))›
  using "equi-rem:3"[THEN "≡dfE", OF 2] "&E" by blast
AOT_show ‹R |: [λx [O!]x & [F]x] 1-1⟷ [λx [O!]x & [G]x]›
proof (rule "1-1-cor"[THEN "≡dfI"];
       safe intro!: "&I" "cqt:2" GEN "→I")
  fix x
  AOT_assume 1: ‹[λx [O!]x & [F]x]x›
  AOT_have ‹O!x & [F]x›
    by (rule "β→C"(1)) (auto simp: 1)
  AOT_hence ‹∃!v ([G]v & [R]xv)›
    using B[THEN "∀E"(2), THEN "→E", THEN "→E"] "&E" by blast
  then AOT_obtain y where
    y_prop: ‹O!y & ([G]y & [R]xy & ∀u ([G]u & [R]xu → u =E y))›
    using "equi:1"[THEN "≡E"(1)] "∃E"[rotated] by fastforce
  AOT_hence ‹[λx O!x & [G]x]y›
    by (auto intro!: "β←C"(1) "cqt:2" "&I" dest: "&E")
  moreover AOT_have ‹∀z ([λx O!x & [G]x]z & [R]xz → z = y)›
  proof(safe intro!: GEN "→I"; frule "&E"(1); drule "&E"(2))
    fix z
    AOT_assume 1: ‹[λx [O!]x & [G]x]z›
    AOT_have 2: ‹O!z & [G]z›
      by (rule "β→C"(1)) (auto simp: 1)
    moreover AOT_assume ‹[R]xz›
    ultimately AOT_have ‹z =E y›
      using y_prop[THEN "&E"(2), THEN "&E"(2), THEN "∀E"(2),
                   THEN "→E", THEN "→E", rotated, OF "&I"] "&E"
      by blast
    AOT_thus ‹z = y›
      using 2[THEN "&E"(1)] by (metis "=E-simple:2" "→E")
  qed
  ultimately AOT_have ‹[λx O!x & [G]x]y & [R]xy &
                       ∀z ([λx O!x & [G]x]z & [R]xz → z = y)›
    using y_prop[THEN "&E"(2), THEN "&E"(1), THEN "&E"(2)] "&I" by auto
  AOT_hence ‹∃y ([λx O!x & [G]x]y & [R]xy &
                 ∀z ([λx O!x & [G]x]z & [R]xz → z = y))›
    by (rule "∃I")
  AOT_thus ‹∃!y ([λx [O!]x & [G]x]y & [R]xy)›
    using "uniqueness:1"[THEN "≡dfI"] by fast
next
  fix y
  AOT_assume 1: ‹[λx [O!]x & [G]x]y›
  AOT_have oy_gy: ‹O!y & [G]y›
    by (rule "β→C"(1)) (auto simp: 1)
  AOT_hence ‹∃u ([F]u & [R]uy)›
    using C[THEN "∀E"(2), THEN "→E", THEN "→E"] "&E" by blast
  then AOT_obtain x where x_prop: ‹O!x & ([F]x & [R]xy)›
    using "∃E"[rotated] by blast
  AOT_hence ofx: ‹[λx O!x & [F]x]x›
    by (auto intro!: "β←C"(1) "cqt:2" "&I" dest: "&E")
  AOT_have ‹∃α ([λx [O!]x & [F]x]α & [R]αy &
                ∀β ([λx [O!]x & [F]x]β & [R]βy → β = α))›
  proof (safe intro!: "∃I"(2)[where β=x] "&I" GEN "→I")
    AOT_show ‹[λx O!x & [F]x]x› using ofx.
  next
    AOT_show ‹[R]xy› using x_prop[THEN "&E"(2), THEN "&E"(2)].
  next
    fix z
    AOT_assume 1: ‹[λx [O!]x & [F]x]z & [R]zy›
```

```
        AOT_have oz_fz: ‹O!z & [F]z›
          by (rule "β→C"(1)) (auto simp: 1[THEN "&E"(1)])
        AOT_have ‹z =E x›
          using A[THEN "∀E"(2)[where β=z], THEN "→E", THEN "∀E"(2)[where β=x],
                  THEN "→E", THEN "∀E"(2)[where β=y], THEN "→E",
                  THEN "→E", THEN "→E", OF oz_fz[THEN "&E"(1)],
                  OF x_prop[THEN "&E"(1)], OF oy_gy[THEN "&E"(1)], OF "&I", OF "&I",
                  OF oz_fz[THEN "&E"(2)], OF x_prop[THEN "&E"(2), THEN "&E"(1)],
                  OF oy_gy[THEN "&E"(2)], OF "&I", OF 1[THEN "&E"(2)],
                  OF x_prop[THEN "&E"(2), THEN "&E"(2)]].
        AOT_thus ‹z = x›
          by (metis "=E-simple:2" "vdash-properties:10")
      qed
      AOT_thus ‹∃!x ([λx [O!]x & [F]x]x & [R]xy)›
        by (rule "uniqueness:1"[THEN "≡dfI"])
    qed
  qed
  finally show ?thesis.
qed

AOT_theorem "empty-approx:1": ‹(¬∃u [F]u & ¬∃v [H]v) → F ≈E H›                    (733.1)
proof(rule "→I"; frule "&E"(1); drule "&E"(2))
  AOT_assume 0: ‹¬∃u [F]u› and 1: ‹¬∃v [H]v›
  AOT_have ‹∀u ([F]u → ∃!v ([H]v & [R]uv))› for R
  proof(rule Ordinary.GEN; rule "→I"; rule "raa-cor:1")
    fix u
    AOT_assume ‹[F]u›
    AOT_hence ‹∃u [F]u› using "Ordinary.∃I" "&I" by fast
    AOT_thus ‹∃u [F]u & ¬∃u [F]u› using "&I" 0 by blast
  qed
  moreover AOT_have ‹∀v ([H]v → ∃!u ([F]u & [R]uv))› for R
  proof(rule Ordinary.GEN; rule "→I"; rule "raa-cor:1")
    fix v
    AOT_assume ‹[H]v›
    AOT_hence ‹∃v [H]v› using "Ordinary.∃I" "&I" by fast
    AOT_thus ‹∃v [H]v & ¬∃v [H]v› using 1 "&I" by blast
  qed
  ultimately AOT_have ‹R |: F 1-1⟷E H› for R
    apply (safe intro!: "equi:2"[THEN "≡dfI"] "&I" GEN "cqt:2[const_var]"[axiom_inst])
    using "∀E" by blast+
  AOT_hence ‹∃R R |: F 1-1⟷E H› by (rule "∃I")
  AOT_thus ‹F ≈E H›
    by (rule "equi:3"[THEN "≡dfI"])
qed

AOT_theorem "empty-approx:2": ‹(∃u [F]u & ¬∃v [H]v) → ¬(F ≈E H)›                  (733.2)
proof(rule "→I"; frule "&E"(1); drule "&E"(2); rule "raa-cor:2")
  AOT_assume 1: ‹∃u [F]u› and 2: ‹¬∃v [H]v›
  AOT_obtain b where b_prop: ‹O!b & [F]b›
    using 1 "∃E"[rotated] by blast
  AOT_assume ‹F ≈E H›
  AOT_hence ‹∃R R |: F 1-1⟷E H›
    by (rule "equi:3"[THEN "≡dfE"])
  then AOT_obtain R where ‹R |: F 1-1⟷E H›
    using "∃E"[rotated] by blast
  AOT_hence ϑ: ‹∀u ([F]u → ∃!v ([H]v & [R]uv))›
    using "equi:2"[THEN "≡dfE"] "&E" by blast+
  AOT_have ‹∃!v ([H]v & [R]bv)› for u
    using ϑ[THEN "∀E"(2)[where β=b], THEN "→E", THEN "→E",
               OF b_prop[THEN "&E"(1)], OF b_prop[THEN "&E"(2)]].
  AOT_hence ‹∃v ([H]v & [R]bv & ∀u ([H]u & [R]bu → u =E v))›
    by (rule "equi:1"[THEN "≡E"(1)])
  then AOT_obtain x where ‹O!x & ([H]x & [R]bx & ∀u ([H]u & [R]bu → u =E x))›
```

```
    using "∃E"[rotated] by blast
  AOT_hence ‹O!x & [H]x› using "&E" "&I" by blast
  AOT_hence ‹∃v [H]v› by (rule "∃I")
  AOT_thus ‹∃v [H]v & ¬∃v [H]v› using 2 "&I" by blast
qed


AOT_define FminusU :: ‹Π ⇒ τ ⇒ Π› ("_^-_")
  "F-u": ‹[F]^-x =df [λz [F]z & z ≠E x]›                                    (734)

text‹Note: not explicitly in PLM.›
AOT_theorem "F-u[den]": ‹[F]^-x↓›                                          (734)
  by (rule "=dfI"(1)[OF "F-u", where τ1τn="(_,_)", simplified]; "cqt:2[lambda]")
AOT_theorem "F-u[equiv]": ‹[[F]^-x]y ≡ ([F]y & y ≠E x)›                    (734)
  by (auto intro: "F-u"[THEN "=dfI"(1), where τ1τn="(_,_)", simplified]
           intro!: "cqt:2" "beta-C-cor:2"[THEN "→E", THEN "∀E"(2)])

AOT_theorem eqP': ‹F ≈E G & [F]u & [G]v → [F]^-u ≈E [G]^-v›                (735)
proof (rule "→I"; frule "&E"(2); drule "&E"(1); frule "&E"(2); drule "&E"(1))
  AOT_assume ‹F ≈E G›
  AOT_hence ‹∃R R |: F 1-1⟷E G›
    using "equi:3"[THEN "≡dfE"] by blast
  then AOT_obtain R where R_prop: ‹R |: F 1-1⟷E G›
    using "∃E"[rotated] by blast
  AOT_hence A: ‹∀u ([F]u → ∃!v ([G]v & [R]uv))›
        and B: ‹∀v ([G]v → ∃!u ([F]u & [R]uv))›
    using "equi:2"[THEN "≡dfE"] "&E" by blast+
  AOT_have ‹R |: F 1-1⟶ontoE G›
    using "equi-rem-thm"[THEN "≡E"(1), OF R_prop].
  AOT_hence ‹R |: F 1-1⟶E G & R |: F ⟶ontoE G›
    using "equi-rem:4"[THEN "≡dfE"] by blast
  AOT_hence C: ‹∀t∀u∀v (([F]t & [F]u & [G]v) → ([R]tv & [R]uv → t =E u))›
    using "equi-rem:2"[THEN "≡dfE"] "&E" by blast
  AOT_assume fu: ‹[F]u›
  AOT_assume gv: ‹[G]v›
  AOT_have ‹[λz [Π]z & z ≠E κ]↓› for Π κ
    by "cqt:2[lambda]"
  note Π_minus_κI = "rule-id-df:2:b[2]"[
      where τ=‹(λ(Π, κ). «[Π]^-κ»)›, simplified, OF "F-u", simplified, OF this]
   and Π_minus_κE = "rule-id-df:2:a[2]"[
      where τ=‹(λ(Π, κ). «[Π]^-κ»)›, simplified, OF "F-u", simplified, OF this]
  AOT_have Π_minus_κ_den: ‹[Π]^-κ↓› for Π κ
    by (rule Π_minus_κI) "cqt:2[lambda]"+
  {
    fix R
    AOT_assume R_prop: ‹R |: F 1-1⟷E G›
    AOT_hence A: ‹∀u ([F]u → ∃!v ([G]v & [R]uv))›
          and B: ‹∀v ([G]v → ∃!u ([F]u & [R]uv))›
      using "equi:2"[THEN "≡dfE"] "&E" by blast+
    AOT_have ‹R |: F 1-1⟶ontoE G›
      using "equi-rem-thm"[THEN "≡E"(1), OF R_prop].
    AOT_hence ‹R |: F 1-1⟶E G & R |: F ⟶ontoE G›
      using "equi-rem:4"[THEN "≡dfE"] by blast
    AOT_hence C: ‹∀t∀u∀v (([F]t & [F]u & [G]v) → ([R]tv & [R]uv → t =E u))›
      using "equi-rem:2"[THEN "≡dfE"] "&E" by blast

    AOT_assume Ruv: ‹[R]uv›
    AOT_have ‹R |: [F]^-u 1-1⟷E [G]^-v›
    proof(safe intro!: "equi:2"[THEN "≡dfI"] "&I" "cqt:2[const_var]"[axiom_inst]
                       Π_minus_κ_den Ordinary.GEN "→I")
      fix u'
      AOT_assume ‹[[F]^-u]u'›
      AOT_hence 0: ‹[λz [F]z & z ≠E u]u'›
```

```
      using Π_minus_κE by fast
    AOT_have 0: ‹[F]u' & u' ≠E u›
      by (rule "β→C"(1)[where κ1κn="AOT_term_of_var (Ordinary.Rep u')"]) (fact 0)
    AOT_have ‹∃!v ([G]v & [R]u'v)›
      using A[THEN "Ordinary.∀E"[where α=u'], THEN "→E", OF 0[THEN "&E"(1)]].
    then AOT_obtain v' where
      v'_prop: ‹[G]v' & [R]u'v' & ∀ t ([G]t & [R]u't → t =E v')›
      using "equi:1"[THEN "≡E"(1)] "Ordinary.∃E"[rotated] by fastforce

    AOT_show ‹∃!v' ([[G]^-v]v' & [R]u'v')›
    proof (safe intro!: "equi:1"[THEN "≡E"(2)] "Ordinary.∃I"[where β=v']
                        "&I" Ordinary.GEN "→I")
      AOT_show ‹[[G]^-v]v'›
      proof (rule Π_minus_κI;
                 safe intro!: "β←C"(1) "cqt:2" "&I" "thm-neg=E"[THEN "≡E"(2)])
        AOT_show ‹[G]v'› using v'_prop "&E" by blast
      next
        AOT_show ‹¬v' =E v›
        proof (rule "raa-cor:2")
          AOT_assume ‹v' =E v›
          AOT_hence ‹v' = v› by (metis "=E-simple:2" "→E")
          AOT_hence Ruv': ‹[R]uv'› using "rule=E" Ruv id_sym by fast
          AOT_have ‹u' =E u›
            by (rule C[THEN "Ordinary.∀E", THEN "Ordinary.∀E",
                       THEN "Ordinary.∀E"[where α=v'], THEN "→E", THEN "→E"])
               (safe intro!: "&I" 0[THEN "&E"(1)] fu
                              v'_prop[THEN "&E"(1), THEN "&E"(1)]
                              Ruv' v'_prop[THEN "&E"(1), THEN "&E"(2)])
          moreover AOT_have ‹¬(u' =E u)›
            using "0" "&E"(2) "≡E"(1) "thm-neg=E" by blast
          ultimately AOT_show ‹u' =E u & ¬u' =E u› using "&I" by blast
        qed
      qed
    next
      AOT_show ‹[R]u'v'› using v'_prop "&E" by blast
    next
      fix t
      AOT_assume t_prop: ‹[[G]^-v]t & [R]u't›
      AOT_have gt_t_noteq_v: ‹[G]t & t ≠E v›
        apply (rule "β→C"(1)[where κ1κn="AOT_term_of_var (Ordinary.Rep t)"])
        apply (rule Π_minus_κE)
        by (fact t_prop[THEN "&E"(1)])
      AOT_show ‹t =E v'›
        using v'_prop[THEN "&E"(2), THEN "Ordinary.∀E", THEN "→E",
                      OF "&I", OF gt_t_noteq_v[THEN "&E"(1)],
                      OF t_prop[THEN "&E"(2)]].
    qed
  next
    fix v'
    AOT_assume G_minus_v_v': ‹[[G]^-v]v'›
    AOT_have gt_t_noteq_v: ‹[G]v' & v' ≠E v›
      apply (rule "β→C"(1)[where κ1κn="AOT_term_of_var (Ordinary.Rep v')"])
      apply (rule Π_minus_κE)
      by (fact G_minus_v_v')
    AOT_have ‹∃!u([F]u & [R]uv')›
      using B[THEN "Ordinary.∀E", THEN "→E", OF gt_t_noteq_v[THEN "&E"(1)]].
    then AOT_obtain u' where
      u'_prop: ‹[F]u' & [R]u'v' & ∀t ([F]t & [R]tv' → t =E u')›
      using "equi:1"[THEN "≡E"(1)] "Ordinary.∃E"[rotated] by fastforce
    AOT_show ‹∃!u' ([[F]^-u]u' & [R]u'v')›
    proof (safe intro!: "equi:1"[THEN "≡E"(2)] "Ordinary.∃I"[where β=u'] "&I"
                        u'_prop[THEN "&E"(1), THEN "&E"(2)] Ordinary.GEN "→I")
      AOT_show ‹[[F]^-u]u'›
```

```
      proof (rule Π_minus_κI;
             safe intro!: "β←C"(1) "cqt:2" "&I" "thm-neg=E"[THEN "≡E"(2)]
             u'_prop[THEN "&E"(1), THEN "&E"(1)]; rule "raa-cor:2")
        AOT_assume u'_eq_u: ‹u' =E u›
        AOT_hence ‹u' = u›
          using "=E-simple:2" "vdash-properties:10" by blast
        AOT_hence Ru'v: ‹[R]u'v› using "rule=E" Ruv id_sym by fast
        AOT_have ‹v' ≠E v›
          using "&E"(2) gt_t_noteq_v by blast
        AOT_hence v'_noteq_v: ‹¬(v' =E v)› by (metis "≡E"(1) "thm-neg=E")
        AOT_have ‹∃u ([G]u & [R]u'u & ∀v ([G]v & [R]u'v → v =E u))›
          using A[THEN "Ordinary.∀E", THEN "→E",
                  OF u'_prop[THEN "&E"(1), THEN "&E"(1)],
                  THEN "equi:1"[THEN "≡E"(1)]].
        then AOT_obtain t where
          t_prop: ‹[G]t & [R]u't & ∀v ([G]v & [R]u'v → v =E t)›
          using "Ordinary.∃E"[rotated] by meson
        AOT_have ‹v =E t› if ‹[G]v› and ‹[R]u'v› for v
          using t_prop[THEN "&E"(2), THEN "Ordinary.∀E", THEN "→E",
                       OF "&I", OF that].
        AOT_hence ‹v' =E t› and ‹v =E t›
          by (auto simp: gt_t_noteq_v[THEN "&E"(1)] Ru'v gv
                         u'_prop[THEN "&E"(1), THEN "&E"(2)])
        AOT_hence ‹v' =E v›
          using "rule=E" "=E-simple:2" id_sym "→E" by fast
        AOT_thus ‹v' =E v & ¬v' =E v›
          using v'_noteq_v "&I" by blast
      qed
    next
      fix t
      AOT_assume 0: ‹[[F]-u]t & [R]tv'›
      moreover AOT_have ‹[F]t & t ≠E u›
        apply (rule "β→C"(1)[where κ1κn="AOT_term_of_var (Ordinary.Rep t)"])
        apply (rule Π_minus_κE)
        by (fact 0[THEN "&E"(1)])
      ultimately AOT_show ‹t =E u'›
        using u'_prop[THEN "&E"(2), THEN "Ordinary.∀E", THEN "→E", OF "&I"]
              "&E" by blast
    qed
  qed
  AOT_hence ‹∃R R |: [F]-u 1-1⟷E [G]-v›
    by (rule "∃I")
} note 1 = this
moreover {
  AOT_assume not_Ruv: ‹¬[R]uv›
  AOT_have ‹∃!v ([G]v & [R]uv)›
    using A[THEN "Ordinary.∀E", THEN "→E", OF fu].
  then AOT_obtain b where
    b_prop: ‹O!b & ([G]b & [R]ub & ∀t([G]t & [R]ut → t =E b))›
    using "equi:1"[THEN "≡E"(1)] "∃E"[rotated] by fastforce
  AOT_hence ob: ‹O!b› and gb: ‹[G]b› and Rub: ‹[R]ub›
    using "&E" by blast+
  AOT_have ‹O!t → ([G]t & [R]ut → t =E b)› for t
    using b_prop "&E"(2) "∀E"(2) by blast
  AOT_hence b_unique: ‹t =E b› if ‹O!t› and ‹[G]t› and ‹[R]ut› for t
    by (metis Adjunction "modus-tollens:1" "reductio-aa:1" that)
  AOT_have not_v_eq_b: ‹¬(v =E b)›
  proof(rule "raa-cor:2")
    AOT_assume ‹v =E b›
    AOT_hence 0: ‹v = b›
      by (metis "=E-simple:2" "→E")
    AOT_have ‹[R]uv›
      using b_prop[THEN "&E"(2), THEN "&E"(1), THEN "&E"(2)]
```

```
            "rule=E"[rotated, OF 0[symmetric]] by fast
    AOT_thus ‹[R]uv & ¬[R]uv›
      using not_Ruv "&I" by blast
  qed
  AOT_have not_b_eq_v: ‹¬(b =E v)›
    using "modus-tollens:1" not_v_eq_b "ord=Eequiv:2" by blast
  AOT_have ‹∃!u ([F]u & [R]uv)›
    using B[THEN "Ordinary.∀E", THEN "→E", OF gv].
  then AOT_obtain a where
    a_prop: ‹O!a & ([F]a & [R]av & ∀t([F]t & [R]tv → t =E a))›
    using "equi:1"[THEN "≡E"(1)] "∃E"[rotated] by fastforce
  AOT_hence Oa: ‹O!a› and fa: ‹[F]a› and Rav: ‹[R]av›
    using "&E" by blast+
  AOT_have ‹O!t → ([F]t & [R]tv → t =E a)› for t
    using a_prop "&E" "∀E"(2) by blast
  AOT_hence a_unique: ‹t =E a› if ‹O!t› and ‹[F]t› and ‹[R]tv› for t
    by (metis Adjunction "modus-tollens:1" "reductio-aa:1" that)
  AOT_have not_u_eq_a: ‹¬(u =E a)›
  proof(rule "raa-cor:2")
    AOT_assume ‹u =E a›
    AOT_hence 0: ‹u = a›
      by (metis "=E-simple:2" "→E")
    AOT_have ‹[R]uv›
      using a_prop[THEN "&E"(2), THEN "&E"(1), THEN "&E"(2)]
            "rule=E"[rotated, OF 0[symmetric]] by fast
    AOT_thus ‹[R]uv & ¬[R]uv›
      using not_Ruv "&I" by blast
  qed
  AOT_have not_a_eq_u: ‹¬(a =E u)›
    using "modus-tollens:1" not_u_eq_a "ord=Eequiv:2" by blast
  let ?R = ‹«[λu'v' (u' ≠E u & v' ≠E v & [R]u'v') ∨
                    (u' =E a & v' =E b) ∨
                    (u' =E u & v' =E v)]»›
  AOT_have ‹[«?R»]↓› by "cqt:2[lambda]"
  AOT_hence ‹∃ β β = [«?R»]›
    using "free-thms:1" "≡E"(1) by fast
  then AOT_obtain R1 where R1_def: ‹R1 = [«?R»]›
    using "∃E"[rotated] by blast
  AOT_have Rxy1: ‹[R]xy› if ‹[R1]xy› and ‹x ≠E u› and ‹x ≠E a› for x y
  proof -
    AOT_have 0: ‹[«?R»]xy›
      by (rule "rule=E"[rotated, OF R1_def]) (fact that(1))
    AOT_have ‹(x ≠E u & y ≠E v & [R]xy) ∨ (x =E a & y =E b) ∨ (x =E u & y =E v)›
      using "β→C"(1)[OF 0] by simp
    AOT_hence ‹x ≠E u & y ≠E v & [R]xy› using that(2,3)
      by (metis "∨E"(3) "Conjunction Simplification"(1) "≡E"(1)
                "modus-tollens:1" "thm-neg=E")
    AOT_thus ‹[R]xy› using "&E" by blast+
  qed
  AOT_have Rxy2: ‹[R]xy›  if ‹[R1]xy› and ‹y ≠E v› and ‹y ≠E b› for x y
  proof -
    AOT_have 0: ‹[«?R»]xy›
      by (rule "rule=E"[rotated, OF R1_def]) (fact that(1))
    AOT_have ‹(x ≠E u & y ≠E v & [R]xy) ∨ (x =E a & y =E b) ∨ (x =E u & y =E v)›
      using "β→C"(1)[OF 0] by simp
    AOT_hence ‹x ≠E u & y ≠E v & [R]xy›
      using that(2,3)
      by (metis "∨E"(3) "Conjunction Simplification"(2) "≡E"(1)
                "modus-tollens:1" "thm-neg=E")
    AOT_thus ‹[R]xy› using "&E" by blast+
  qed
  AOT_have R1xy: ‹[R1]xy› if ‹[R]xy› and ‹x ≠E u› and ‹y ≠E v› for x y
    by (rule "rule=E"[rotated, OF R1_def[symmetric]])
```

```
      (auto intro!: "β←C"(1) "cqt:2"
              simp: "&I" "ex:1:a" prod_denotesI "rule-ui:3" that "∨I"(1))
  AOT_have R1ab: ‹[R1]ab›
    apply (rule "rule=E"[rotated, OF R1_def[symmetric]])
    apply (safe intro!: "β←C"(1) "cqt:2" prod_denotesI "&I")
    by (meson a_prop b_prop "&I" "&E"(1) "∨I"(1) "∨I"(2) "ord=Eequiv:1" "→E")
  AOT_have R1uv: ‹[R1]uv›
    apply (rule "rule=E"[rotated, OF R1_def[symmetric]])
    apply (safe intro!: "β←C"(1) "cqt:2" prod_denotesI "&I")
    by (meson "&I" "∨I"(2) "ord=Eequiv:1" Ordinary.ψ "→E")
  moreover AOT_have ‹R1 |: F 1-1⟷E G›
  proof (safe intro!: "equi:2"[THEN "≡dfI"] "&I" "cqt:2" Ordinary.GEN "→I")
    fix u'
    AOT_assume fu': ‹[F]u'›
    {
      AOT_assume not_u'_eq_u: ‹¬(u' =E u)› and not_u'_eq_a: ‹¬(u' =E a)›
      AOT_hence u'_noteq_u: ‹u' ≠E u› and u'_noteq_a: ‹u' ≠E a›
        by (metis "≡E"(2) "thm-neg=E")+
      AOT_have ‹∃!v ([G]v & [R]u'v)›
        using A[THEN "Ordinary.∀E", THEN "→E", OF fu'].
      AOT_hence ‹∃v ([G]v & [R]u'v & ∀t ([G]t & [R]u't → t =E v))›
        using "equi:1"[THEN "≡E"(1)] by simp
      then AOT_obtain v' where
        v'_prop: ‹[G]v' & [R]u'v' & ∀t ([G]t & [R]u't → t =E v')›
        using "Ordinary.∃E"[rotated] by meson
      AOT_hence gv': ‹[G]v'› and Ru'v': ‹[R]u'v'›
        using "&E" by blast+
      AOT_have not_v'_eq_v: ‹¬v' =E v›
      proof (rule "raa-cor:2")
        AOT_assume ‹v' =E v›
        AOT_hence ‹v' = v›
          by (metis "=E-simple:2" "→E")
        AOT_hence Ru'v: ‹[R]u'v›
          using "rule=E" Ru'v' by fast
        AOT_have ‹u' =E a›
          using a_unique[OF Ordinary.ψ, OF fu', OF Ru'v].
        AOT_thus ‹u' =E a & ¬u' =E a›
          using not_u'_eq_a "&I" by blast
      qed
      AOT_hence v'_noteq_v: ‹v' ≠E v›
        using "≡E"(2) "thm-neg=E" by blast
      AOT_have ‹∀t ([G]t & [R]u't → t =E v')›
        using v'_prop "&E" by blast
      AOT_hence ‹[G]t & [R]u't → t =E v'› for t
        using "Ordinary.∀E" by meson
      AOT_hence v'_unique: ‹t =E v'› if ‹[G]t› and ‹[R]u't› for t
        by (metis "&I" that "→E")

      AOT_have ‹[G]v' & [R1]u'v' & ∀t ([G]t & [R1]u't → t =E v')›
      proof (safe intro!: "&I" gv' R1xy Ru'v' u'_noteq_u u'_noteq_a "→I"
                          Ordinary.GEN "thm-neg=E"[THEN "≡E"(2)] not_v'_eq_v)
        fix t
        AOT_assume 1: ‹[G]t & [R1]u't›
        AOT_have ‹[R]u't›
          using Rxy1[OF 1[THEN "&E"(2)], OF u'_noteq_u, OF u'_noteq_a].
        AOT_thus ‹t =E v'›
          using v'_unique 1[THEN "&E"(1)] by blast
      qed
      AOT_hence ‹∃v ([G]v & [R1]u'v & ∀t ([G]t & [R1]u't → t =E v))›
        by (rule "Ordinary.∃I")
      AOT_hence ‹∃!v ([G]v & [R1]u'v)›
        by (rule "equi:1"[THEN "≡E"(2)])
    }
```

```
      moreover {
        AOT_assume 0: ‹u' =E u›
        AOT_hence u'_eq_u: ‹u' = u›
          using "=E-simple:2" "→E" by blast
        AOT_have ‹∃!v ([G]v & [R1]u'v)›
        proof (safe intro!: "equi:1"[THEN "≡E"(2)] "Ordinary.∃I"[where β=v]
                             "&I" Ordinary.GEN "→I" gv)
          AOT_show ‹[R1]u'v›
            apply (rule "rule=E"[rotated, OF R1_def[symmetric]])
            apply (safe intro!: "β←C"(1) "cqt:2" "&I" prod_denotesI)
            by (safe intro!: "∨I"(2) "&I" 0 "ord=Eequiv:1"[THEN "→E", OF Ordinary.ψ])
        next
          fix v'
          AOT_assume ‹[G]v' & [R1]u'v'›
          AOT_hence 0: ‹[R1]uv'›
            using "rule=E"[rotated, OF u'_eq_u] "&E"(2) by fast
          AOT_have 1: ‹[«?R»]uv'›
            by (rule "rule=E"[rotated, OF R1_def]) (fact 0)
          AOT_have 2: ‹(u ≠E u & v' ≠E v & [R]uv') ∨
                       (u =E a & v' =E b) ∨
                       (u =E u & v' =E v)›
            using "β→C"(1)[OF 1] by simp
          AOT_have ‹¬u ≠E u›
            using "≡E"(4) "modus-tollens:1" "ord=Eequiv:1" Ordinary.ψ
                  "reductio-aa:2" "thm-neg=E" by blast
          AOT_hence ‹¬((u ≠E u & v' ≠E v & [R]uv') ∨ (u =E a & v' =E b))›
            using not_u_eq_a
            by (metis "∨E"(2) "Conjunction Simplification"(1)
                      "modus-tollens:1" "reductio-aa:1")
          AOT_hence ‹(u =E u & v' =E v)›
            using 2 by (metis "∨E"(2))
          AOT_thus ‹v' =E v›
            using "&E" by blast
        qed
      }
      moreover {
        AOT_assume 0: ‹u' =E a›
        AOT_hence u'_eq_a: ‹u' = a›
          using "=E-simple:2" "→E" by blast
        AOT_have ‹∃!v ([G]v & [R1]u'v)›
        proof (safe intro!: "equi:1"[THEN "≡E"(2)] "∃I"(2)[where β=b] "&I"
                            Ordinary.GEN "→I" b_prop[THEN "&E"(1)]
                            b_prop[THEN "&E"(2), THEN "&E"(1), THEN "&E"(1)])
          AOT_show ‹[R1]u'b›
            apply (rule "rule=E"[rotated, OF R1_def[symmetric]])
            apply (safe intro!: "β←C"(1) "cqt:2" "&I" prod_denotesI)
            apply (rule "∨I"(1); rule "∨I"(2); rule "&I")
             apply (fact 0)
            using b_prop "&E"(1) "ord=Eequiv:1" "→E" by blast
        next
          fix v'
          AOT_assume gv'_R1u'v': ‹[G]v' & [R1]u'v'›
          AOT_hence 0: ‹[R1]av'›
            using u'_eq_a by (meson "rule=E" "&E"(2))
          AOT_have 1: ‹[«?R»]av'›
            by (rule "rule=E"[rotated, OF R1_def]) (fact 0)
          AOT_have ‹(a ≠E u & v' ≠E v & [R]av') ∨
                    (a =E a & v' =E b) ∨
                    (a =E u & v' =E v)›
            using "β→C"(1)[OF 1] by simp
          moreover {
            AOT_assume 0: ‹a ≠E u & v' ≠E v & [R]av'›
            AOT_have ‹∃!v ([G]v & [R]u'v)›
```

```
          using A[THEN "Ordinary.∀E", THEN "→E", OF fu'].
        AOT_hence <∃!v ([G]v & [R]av)>
          using u'_eq_a "rule=E" by fast
        AOT_hence <∃v ([G]v & [R]av & ∀t ([G]t & [R]at → t =E v))>
          using "equi:1"[THEN "≡E"(1)] by fast
        then AOT_obtain s where
          s_prop: <[G]s & [R]as & ∀t ([G]t & [R]at → t =E s)>
          using "Ordinary.∃E"[rotated] by meson
        AOT_have <v' =E s>
          using s_prop[THEN "&E"(2), THEN "Ordinary.∀E"]
                gv'_R1u'v'[THEN "&E"(1)] O[THEN "&E"(2)]
          by (metis "&I" "vdash-properties:10")
        moreover AOT_have <v =E s>
          using s_prop[THEN "&E"(2), THEN "Ordinary.∀E"] gv Rav
          by (metis "&I" "→E")
        ultimately AOT_have <v' =E v>
          by (metis "&I" "ord=Eequiv:2" "ord=Eequiv:3" "→E")
        moreover AOT_have <¬(v' =E v)>
          using O[THEN "&E"(1), THEN "&E"(2)]
          by (metis "≡E"(1) "thm-neg=E")
        ultimately AOT_have <v' =E b>
          by (metis "raa-cor:3")
      }
      moreover {
        AOT_assume <a =E u & v' =E v>
        AOT_hence <v' =E b>
          by (metis "&E"(1) not_a_eq_u "reductio-aa:1")
      }
      ultimately AOT_show <v' =E b>
        by (metis "&E"(2) "∨E"(3) "reductio-aa:1")
    qed
  }
  ultimately AOT_show <∃!v ([G]v & [R1]u'v)>
    by (metis "raa-cor:1")
next
  fix v'
  AOT_assume gv': <[G]v'>
  {
    AOT_assume not_v'_eq_v: <¬(v' =E v)>
           and not_v'_eq_b: <¬(v' =E b)>
    AOT_hence v'_noteq_v: <v' ≠E v>
          and v'_noteq_b: <v' ≠E b>
      by (metis "≡E"(2) "thm-neg=E")+
    AOT_have <∃!u ([F]u & [R]uv')>
      using B[THEN "Ordinary.∀E", THEN "→E", OF gv'].
    AOT_hence <∃u ([F]u & [R]uv' & ∀t ([F]t & [R]tv' → t =E u))>
      using "equi:1"[THEN "≡E"(1)] by simp
    then AOT_obtain u' where
      u'_prop: <[F]u' & [R]u'v' & ∀t ([F]t & [R]tv' → t =E u')>
      using "Ordinary.∃E"[rotated] by meson
    AOT_hence fu': <[F]u'> and Ru'v': <[R]u'v'>
      using "&E" by blast+
    AOT_have not_u'_eq_u: <¬u' =E u>
    proof (rule "raa-cor:2")
      AOT_assume <u' =E u>
      AOT_hence <u' = u>
        by (metis "=E-simple:2" "→E")
      AOT_hence Ruv': <[R]uv'>
        using "rule=E" Ru'v' by fast
      AOT_have <v' =E b>
        using b_unique[OF Ordinary.ψ, OF gv', OF Ruv'].
      AOT_thus <v' =E b & ¬v' =E b>
        using not_v'_eq_b "&I" by blast
```

```
        qed
        AOT_hence u'_noteq_u: ‹u' ≠E u›
          using "≡E"(2) "thm-neg=E" by blast
        AOT_have ‹∀t ([F]t & [R]tv' → t =E u')›
          using u'_prop "&E" by blast
        AOT_hence ‹[F]t & [R]tv' → t =E u'› for t
          using "Ordinary.∀E" by meson
        AOT_hence u'_unique: ‹t =E u'› if ‹[F]t› and ‹[R]tv'› for t
          by (metis "&I" that "→E")

        AOT_have ‹[F]u' & [R1]u'v' & ∀t ([F]t & [R1]tv' → t =E u')›
        proof (safe intro!: "&I" gv' R1xy Ru'v' u'_noteq_u Ordinary.GEN "→I"
                            "thm-neg=E"[THEN "≡E"(2)] not_v'_eq_v fu')
          fix t
          AOT_assume 1: ‹[F]t & [R1]tv'›
          AOT_have ‹[R]tv'›
            using Rxy2[OF 1[THEN "&E"(2)], OF v'_noteq_v, OF v'_noteq_b].
          AOT_thus ‹t =E u'›
            using u'_unique 1[THEN "&E"(1)] by blast
        qed
        AOT_hence ‹∃u ([F]u & [R1]uv' & ∀t ([F]t & [R1]tv' → t =E u))›
          by (rule "Ordinary.∃I")
        AOT_hence ‹∃!u ([F]u & [R1]uv')›
          by (rule "equi:1"[THEN "≡E"(2)])
      }
      moreover {
        AOT_assume 0: ‹v' =E v›
        AOT_hence u'_eq_u: ‹v' = v›
          using "=E-simple:2" "→E" by blast
        AOT_have ‹∃!u ([F]u & [R1]uv')›
        proof (safe intro!: "equi:1"[THEN "≡E"(2)] "Ordinary.∃I"[where β=u]
                            "&I" Ordinary.GEN "→I" fu)
          AOT_show ‹[R1]uv'›
            by (rule "rule=E"[rotated, OF R1_def[symmetric]])
               (safe intro!: "β←C"(1) "cqt:2" "&I" prod_denotesI Ordinary.ψ
                             "∨I"(2) 0 "ord=Eequiv:1"[THEN "→E"])
        next
          fix u'
          AOT_assume ‹[F]u' & [R1]u'v'›
          AOT_hence 0: ‹[R1]u'v›
            using "rule=E"[rotated, OF u'_eq_u] "&E"(2) by fast
          AOT_have 1: ‹[«?R»]u'v›
            by (rule "rule=E"[rotated, OF R1_def]) (fact 0)
          AOT_have 2: ‹(u' ≠E u & v ≠E v & [R]u'v) ∨
                       (u' =E a & v =E b) ∨
                       (u' =E u & v =E v)›
            using "β→C"(1)[OF 1, simplified] by simp
          AOT_have ‹¬v ≠E v›
            using "≡E"(4) "modus-tollens:1" "ord=Eequiv:1" Ordinary.ψ
                  "reductio-aa:2" "thm-neg=E" by blast
          AOT_hence ‹¬((u' ≠E u & v ≠E v & [R]u'v) ∨ (u' =E a & v =E b))›
            by (metis "&E"(1) "&E"(2) "∨E"(3) not_v_eq_b "raa-cor:3")
          AOT_hence ‹(u' =E u & v =E v)›
            using 2 by (metis "∨E"(2))
          AOT_thus ‹u' =E u›
            using "&E" by blast
        qed
      }
      moreover {
        AOT_assume 0: ‹v' =E b›
        AOT_hence v'_eq_b: ‹v' = b›
          using "=E-simple:2" "→E" by blast
        AOT_have ‹∃!u ([F]u & [R1]uv')›
```

```
      proof (safe intro!: "equi:1"[THEN "≡E"(2)] "∃I"(2)[where β=a] "&I"
                          Ordinary.GEN "→I" b_prop[THEN "&E"(1)] Oa fa
                          b_prop[THEN "&E"(2), THEN "&E"(1), THEN "&E"(1)])
        AOT_show ‹[R1]av'›
          apply (rule "rule=E"[rotated, OF R1_def[symmetric]])
          apply (safe intro!: "β←C"(1) "cqt:2" "&I" prod_denotesI)
          apply (rule "∨I"(1); rule "∨I"(2); rule "&I")
          using Oa "ord=Eequiv:1" "→E" apply blast
          using "0" by blast
      next
        fix u'
        AOT_assume fu'_R1u'v': ‹[F]u' & [R1]u'v'›
        AOT_hence 0: ‹[R1]u'b›
          using v'_eq_b by (meson "rule=E" "&E"(2))
        AOT_have 1: ‹[«?R»]u'b›
          by (rule "rule=E"[rotated, OF R1_def]) (fact 0)
        AOT_have ‹(u' ≠E u & b ≠E v & [R]u'b) ∨
                  (u' =E a & b =E b) ∨
                  (u' =E u & b =E v)›
          using "β→C"(1)[OF 1, simplified] by simp
        moreover {
          AOT_assume 0: ‹u' ≠E u & b ≠E v & [R]u'b›
          AOT_have ‹∃!u ([F]u & [R]uv')›
            using B[THEN "Ordinary.∀E", THEN "→E", OF gv'].
          AOT_hence ‹∃!u ([F]u & [R]ub)›
            using v'_eq_b "rule=E" by fast
          AOT_hence ‹∃u ([F]u & [R]ub & ∀t ([F]t & [R]tb → t =E u))›
            using "equi:1"[THEN "≡E"(1)] by fast
          then AOT_obtain s where
            s_prop: ‹[F]s & [R]sb & ∀t ([F]t & [R]tb → t =E s)›
            using "Ordinary.∃E"[rotated] by meson
          AOT_have ‹u' =E s›
            using s_prop[THEN "&E"(2), THEN "Ordinary.∀E"]
                  fu'_R1u'v'[THEN "&E"(1)] 0[THEN "&E"(2)]
            by (metis "&I" "→E")
          moreover AOT_have ‹u =E s›
            using s_prop[THEN "&E"(2), THEN "Ordinary.∀E"] fu Rub
            by (metis "&I" "→E")
          ultimately AOT_have ‹u' =E u›
            by (metis "&I" "ord=Eequiv:2" "ord=Eequiv:3" "→E")
          moreover AOT_have ‹¬(u' =E u)›
            using 0[THEN "&E"(1), THEN "&E"(1)] by (metis "≡E"(1) "thm-neg=E")
          ultimately AOT_have ‹u' =E a›
            by (metis "raa-cor:3")
        }
        moreover {
          AOT_assume ‹u' =E u & b =E v›
          AOT_hence ‹u' =E a›
            by (metis "&E"(2) not_b_eq_v "reductio-aa:1")
        }
        ultimately AOT_show ‹u' =E a›
          by (metis "&E"(1) "∨E"(3) "reductio-aa:1")
      qed
    }
    ultimately AOT_show ‹∃!u ([F]u & [R1]uv')›
      by (metis "raa-cor:1")
  qed
  ultimately AOT_have ‹∃R R |: [F]^-u 1-1⟷E [G]^-v›
    using 1 by blast
}
ultimately AOT_have ‹∃R R |: [F]^-u 1-1⟷E [G]^-v›
  using R_prop by (metis "reductio-aa:2")
AOT_thus ‹[F]^-u ≈E [G]^-v›
```

```
    by (rule "equi:3"[THEN "≡dfI"])
qed


AOT_theorem "P'-eq": ‹[F]^-u ≈E [G]^-v & [F]u & [G]v → F ≈E G›                    (736)
proof(safe intro!: "→I"; frule "&E"(1); drule "&E"(2);
      frule "&E"(1); drule "&E"(2))
  AOT_have ‹[λz [Π]z & z ≠E κ]↓› for Π κ by "cqt:2[lambda]"
  note Π_minus_κI = "rule-id-df:2:b[2]"[
      where τ=‹(λ(Π, κ). «[Π]^-κ»)›, simplified, OF "F-u", simplified, OF this]
   and Π_minus_κE = "rule-id-df:2:a[2]"[
   where τ=‹(λ(Π, κ). «[Π]^-κ»)›, simplified, OF "F-u", simplified, OF this]
  AOT_have Π_minus_κ_den: ‹[Π]^-κ↓› for Π κ
    by (rule Π_minus_κI) "cqt:2[lambda]"+

  AOT_have Π_minus_κE1: ‹[Π]κ'›
       and Π_minus_κE2: ‹κ' ≠E κ› if ‹[[Π]^-κ]κ'› for Π κ κ'
  proof -
    AOT_have ‹[λz [Π]z & z ≠E κ]κ'›
      using Π_minus_κE that by fast
    AOT_hence ‹[Π]κ' & κ' ≠E κ›
      by (rule "β→C"(1))
    AOT_thus ‹[Π]κ'› and ‹κ' ≠E κ›
      using "&E" by blast+
  qed
  AOT_have Π_minus_κI': ‹[[Π]^-κ]κ'› if ‹[Π]κ'› and ‹κ' ≠E κ› for Π κ κ'
  proof -
    AOT_have κ'_den: ‹κ'↓›
      by (metis "russell-axiom[exe,1].ψ_denotes_asm" that(1))
    AOT_have ‹[λz [Π]z & z ≠E κ]κ'›
      by (safe intro!: "β←C"(1) "cqt:2" κ'_den "&I" that)
    AOT_thus ‹[[Π]^-κ]κ'›
      using Π_minus_κI by fast
  qed

  AOT_assume Gv: ‹[G]v›
  AOT_assume Fu: ‹[F]u›
  AOT_assume ‹[F]^-u ≈E [G]^-v›
  AOT_hence ‹∃R R |: [F]^-u 1-1⟷E [G]^-v›
    using "equi:3"[THEN "≡dfE"] by blast
  then AOT_obtain R where R_prop: ‹R |: [F]^-u 1-1⟷E [G]^-v›
    using "∃E"[rotated] by blast
  AOT_hence Fact1: ‹∀r([[F]^-u]r → ∃!s ([[G]^-v]s & [R]rs))›
        and Fact1': ‹∀s([[G]^-v]s → ∃!r ([[F]^-u]r & [R]rs))›
    using "equi:2"[THEN "≡dfE"] "&E" by blast+
  AOT_have ‹R |: [F]^-u 1-1⟶ontoE [G]^-v›
    using "equi-rem-thm"[unvarify F G, OF Π_minus_κ_den, OF Π_minus_κ_den,
                         THEN "≡E"(1), OF R_prop].
  AOT_hence ‹R |: [F]^-u 1-1⟶E [G]^-v & R |: [F]^-u ⟶ontoE [G]^-v›
    using "equi-rem:4"[THEN "≡dfE"] by blast
  AOT_hence Fact2:
    ‹∀r∀s∀t(([[F]^-u]r & [[F]^-u]s & [[G]^-v]t) → ([R]rt & [R]st → r =E s))›
    using "equi-rem:2"[THEN "≡dfE"] "&E" by blast

  let ?R = ‹«[λxy ([[F]^-u]x & [[G]^-v]y & [R]xy) ∨ (x =E u & y =E v)]»›
  AOT_have R_den: ‹«?R»↓› by "cqt:2[lambda]"

  AOT_show ‹F ≈E G›
  proof(safe intro!: "equi:3"[THEN "≡dfI"] "∃I"(1)[where τ="?R"] R_den
                     "equi:2"[THEN "≡dfI"] "&I" "cqt:2" Ordinary.GEN "→I")
    fix r
    AOT_assume Fr: ‹[F]r›
    {
```

```
      AOT_assume not_r_eq_u: ‹¬(r =E u)›
      AOT_hence r_noteq_u: ‹r ≠E u›
        using "≡E"(2) "thm-neg=E" by blast
      AOT_have ‹[[F]^-u]r›
        by(rule Π_minus_κI; safe intro!: "β←C"(1) "cqt:2" "&I" Fr r_noteq_u)
      AOT_hence ‹∃!s ([[G]^-v]s & [R]rs)›
        using Fact1[THEN "∀E"(2)] "→E" Ordinary.ψ by blast
      AOT_hence ‹∃s ([[G]^-v]s & [R]rs & ∀t ([[G]^-v]t & [R]rt → t =E s))›
        using "equi:1"[THEN "≡E"(1)] by simp
      then AOT_obtain s where s_prop: ‹[[G]^-v]s & [R]rs & ∀t ([[G]^-v]t & [R]rt → t =E s)›
        using "Ordinary.∃E"[rotated] by meson
      AOT_hence G_minus_v_s: ‹[[G]^-v]s› and Rrs: ‹[R]rs›
        using "&E" by blast+
      AOT_have s_unique: ‹t =E s› if ‹[[G]^-v]t› and ‹[R]rt› for t
        using s_prop[THEN "&E"(2), THEN "Ordinary.∀E", THEN "→E", OF "&I", OF that].
      AOT_have Gs: ‹[G]s›
        using Π_minus_κE1[OF G_minus_v_s].
      AOT_have s_noteq_v: ‹s ≠E v›
        using Π_minus_κE2[OF G_minus_v_s].
      AOT_have ‹∃s ([G]s & [«?R»]rs & (∀t ([G]t & [«?R»]rt → t =E s)))›
      proof(safe intro!: "Ordinary.∃I"[where β=s] "&I" Gs Ordinary.GEN "→I")
        AOT_show ‹[«?R»]rs›
          by (auto intro!: "β←C"(1) "cqt:2" "&I" "∨I"(1) Π_minus_κI' Fr Gs
                           s_noteq_v Rrs r_noteq_u
                   simp: "&I" "ex:1:a" prod_denotesI "rule-ui:3")
      next
        fix t
        AOT_assume 0: ‹[G]t & [«?R»]rt›
        AOT_hence ‹([[F]^-u]r & [[G]^-v]t & [R]rt) ∨ (r =E u & t =E v)›
          using "β→C"(1)[OF 0[THEN "&E"(2)], simplified] by blast
        AOT_hence 1: ‹[[F]^-u]r & [[G]^-v]t & [R]rt›
          using not_r_eq_u by (metis "&E"(1) "∨E"(3) "reductio-aa:1")
        AOT_show ‹t =E s› using s_unique 1 "&E" by blast
      qed
    }
    moreover {
      AOT_assume r_eq_u: ‹r =E u›
      AOT_have ‹∃s ([G]s & [«?R»]rs & (∀t ([G]t & [«?R»]rt → t =E s)))›
      proof(safe intro!: "Ordinary.∃I"[where β=v] "&I" Gv Ordinary.GEN "→I")
        AOT_show ‹[«?R»]rv›
          by (auto intro!: "β←C"(1) "cqt:2" "&I" "∨I"(2) Π_minus_κI' Fr r_eq_u
                           "ord=Eequiv:1"[THEN "→E"] Ordinary.ψ
                   simp: "&I" "ex:1:a" prod_denotesI "rule-ui:3")
      next
        fix t
        AOT_assume 0: ‹[G]t & [«?R»]rt›
        AOT_hence ‹([[F]^-u]r & [[G]^-v]t & [R]rt) ∨ (r =E u & t =E v)›
          using "β→C"(1)[OF 0[THEN "&E"(2)], simplified] by blast
        AOT_hence ‹r =E u & t =E v›
          using r_eq_u Π_minus_κE2
          by (metis "&E"(1) "∨E"(2) "≡E"(1) "reductio-aa:1" "thm-neg=E")
        AOT_thus ‹t =E v› using "&E" by blast
      qed
    }
    ultimately AOT_show ‹∃!s ([G]s & [«?R»]rs)›
      using "reductio-aa:2" "equi:1"[THEN "≡E"(2)] by fast
  next
    fix s
    AOT_assume Gs: ‹[G]s›

    {
      AOT_assume not_s_eq_v: ‹¬(s =E v)›
      AOT_hence s_noteq_v: ‹s ≠E v›
```

```
        using "≡E"(2) "thm-neg=E" by blast
      AOT_have ‹[[G]^-v]s›
        by (rule Π_minus_κI; auto intro!: "β←C"(1) "cqt:2" "&I" Gs s_noteq_v)
      AOT_hence ‹∃!r ([[F]^-u]r & [R]rs)›
        using Fact1'[THEN "Ordinary.∀E"] "→E" by blast
      AOT_hence ‹∃r ([[F]^-u]r & [R]rs & ∀t ([[F]^-u]t & [R]ts → t =E r))›
        using "equi:1"[THEN "≡E"(1)] by simp
      then AOT_obtain r where
        r_prop: ‹[[F]^-u]r & [R]rs & ∀t ([[F]^-u]t & [R]ts → t =E r)›
        using "Ordinary.∃E"[rotated] by meson
      AOT_hence F_minus_u_r: ‹[[F]^-u]r› and Rrs: ‹[R]rs›
        using "&E" by blast+
      AOT_have r_unique: ‹t =E r› if ‹[[F]^-u]t› and ‹[R]ts› for t
        using r_prop[THEN "&E"(2), THEN "Ordinary.∀E",
                     THEN "→E", OF "&I", OF that].
      AOT_have Fr: ‹[F]r›
        using Π_minus_κE1[OF F_minus_u_r].
      AOT_have r_noteq_u: ‹r ≠E u›
        using Π_minus_κE2[OF F_minus_u_r].
      AOT_have ‹∃r ([F]r & [«?R»]rs & (∀t ([F]t & [«?R»]ts → t =E r)))›
      proof(safe intro!: "Ordinary.∃I"[where β=r] "&I" Fr Ordinary.GEN "→I")
        AOT_show ‹[«?R»]rs›
          by (auto intro!: "β←C"(1) "cqt:2" "&I" "∨I"(1) Π_minus_κI' Fr
                           Gs s_noteq_v Rrs r_noteq_u
                     simp: "&I" "ex:1:a" prod_denotesI "rule-ui:3")
      next
        fix t
        AOT_assume 0: ‹[F]t & [«?R»]ts›
        AOT_hence ‹([[F]^-u]t & [[G]^-v]s & [R]ts) ∨ (t =E u & s =E v)›
          using "β→C"(1)[OF 0[THEN "&E"(2)], simplified] by blast
        AOT_hence 1: ‹[[F]^-u]t & [[G]^-v]s & [R]ts›
          using not_s_eq_v by (metis "&E"(2) "∨E"(3) "reductio-aa:1")
        AOT_show ‹t =E r› using r_unique 1 "&E" by blast
      qed
    }
    moreover {
      AOT_assume s_eq_v: ‹s =E v›
      AOT_have ‹∃r ([F]r & [«?R»]rs & (∀t ([F]t & [«?R»]ts → t =E r)))›
      proof(safe intro!: "Ordinary.∃I"[where β=u] "&I" Fu Ordinary.GEN "→I")
        AOT_show ‹[«?R»]us›
          by (auto intro!: "β←C"(1) "cqt:2" "&I" prod_denotesI "∨I"(2)
                            Π_minus_κI' Gs s_eq_v Ordinary.ψ
                            "ord=Eequiv:1"[THEN "→E"])
      next
        fix t
        AOT_assume 0: ‹[F]t & [«?R»]ts›
        AOT_hence 1: ‹([[F]^-u]t & [[G]^-v]s & [R]ts) ∨ (t =E u & s =E v)›
          using "β→C"(1)[OF 0[THEN "&E"(2)], simplified] by blast
        moreover AOT_have ‹¬([[F]^-u]t & [[G]^-v]s & [R]ts)›
        proof (rule "raa-cor:2")
          AOT_assume ‹([[F]^-u]t & [[G]^-v]s & [R]ts)›
          AOT_hence ‹[[G]^-v]s› using "&E" by blast
          AOT_thus ‹s =E v & ¬(s =E v)›
            by (metis Π_minus_κE2 "≡E"(4) "reductio-aa:1" s_eq_v "thm-neg=E")
        qed
        ultimately AOT_have ‹t =E u & s =E v›
          by (metis "∨E"(2))
        AOT_thus ‹t =E u› using "&E" by blast
      qed
    }
    ultimately AOT_show ‹∃!r ([F]r & [«?R»]rs)›
      using "≡E"(2) "equi:1" "reductio-aa:2" by fast
  qed
```

```
qed


AOT_theorem "approx-cont:1": ‹∃F∃G ◇(F ≈_E G & ◇¬F ≈_E G)›                (737.1)
proof -
  let ?P = ‹«[λx E!x & ¬𝒜E!x]»›
  AOT_have ‹◇q_0 & ◇¬q_0› by (metis q_0_prop)
  AOT_hence 1: ‹◇∃x(E!x & ¬𝒜E!x) & ◇¬∃x(E!x & ¬𝒜E!x)›
    by (rule q_0_def[THEN "=_dfE"(2), rotated])
       (simp add: "log-prop-prop:2")
  AOT_have ϑ: ‹◇∃x [«?P»]x & ◇¬∃x [«?P»]x›
    apply (AOT_subst ‹[«?P»]x› ‹E!x & ¬𝒜E!x› for: x)
     apply (rule "beta-C-meta"[THEN "→E"]; "cqt:2[lambda]")
    by (fact 1)
  show ?thesis
  proof (rule "∃I"(1))+
    AOT_have ‹◇[L]^- ≈_E [«?P»] & ◇¬[L]^- ≈_E [«?P»]›
    proof (rule "&I"; rule "RM◇"[THEN "→E"]; (rule "→I")?)
      AOT_modally_strict {
        AOT_assume A: ‹¬∃x [«?P»]x›
        AOT_show ‹[L]^- ≈_E [«?P»]›
        proof (safe intro!: "empty-approx:1"[unvarify F H, THEN "→E"]
                            "rel-neg-T:3" "&I")
          AOT_show ‹[«?P»]↓› by "cqt:2[lambda]"
        next
          AOT_show ‹¬∃u [L^-]u›
          proof (rule "raa-cor:2")
            AOT_assume ‹∃u [L^-]u›
            then AOT_obtain u where ‹[L^-]u›
              using "Ordinary.∃E"[rotated] by blast
            moreover AOT_have ‹¬[L^-]u›
              using "thm-noncont-e-e:2"[THEN "contingent-properties:2"[THEN "≡_dfE"],
                                        THEN "&E"(2)]
              by (metis "qml:2"[axiom_inst] "rule-ui:3" "→E")
            ultimately AOT_show ‹p & ¬p› for p
              by (metis  "raa-cor:3")
          qed
        next
          AOT_show ‹¬∃v [«?P»]v›
          proof (rule "raa-cor:2")
            AOT_assume ‹∃v [«?P»]v›
            then AOT_obtain u where ‹[«?P»]u›
              using "Ordinary.∃E"[rotated] by blast
            AOT_hence ‹[«?P»]u›
              using "&E" by blast
            AOT_hence ‹∃x [«?P»]x›
              by (rule "∃I")
            AOT_thus ‹∃x [«?P»]x & ¬∃x [«?P»]x›
              using A "&I" by blast
          qed
        qed
      }
    next
      AOT_show ‹◇¬∃x [«?P»]x›
        using ϑ "&E" by blast
    next
      AOT_modally_strict {
        AOT_assume A: ‹∃x [«?P»]x›
        AOT_have B: ‹¬[«?P»] ≈_E [L]^-›
        proof (safe intro!: "empty-approx:2"[unvarify F H, THEN "→E"]
                            "rel-neg-T:3" "&I")
          AOT_show ‹[«?P»]↓›
            by "cqt:2[lambda]"
```

```
        next
          AOT_obtain x where Px: ‹[«?P»]x›
            using A "∃E" by blast
          AOT_hence ‹E!x & ¬𝒜E!x›
            by (rule "β→C"(1))
          AOT_hence 1: ‹◇E!x›
            by (metis "T◇" "&E"(1) "vdash-properties:10")
          AOT_have ‹[λx ◇E!x]x›
            by (auto intro!: "β←C"(1) "cqt:2" 1)
          AOT_hence ‹O!x›
            by (rule AOT_ordinary[THEN "=dfI"(2), rotated]) "cqt:2[lambda]"
          AOT_hence ‹O!x & [«?P»]x›
            using Px "&I" by blast
          AOT_thus ‹∃u [«?P»]u›
            by (rule "∃I")
        next
          AOT_show ‹¬∃u [L⁻]u›
          proof (rule "raa-cor:2")
            AOT_assume ‹∃u [L⁻]u›
            then AOT_obtain u where ‹[L⁻]u›
              using "Ordinary.∃E"[rotated] by blast
            moreover AOT_have ‹¬[L⁻]u›
              using "thm-noncont-e-e:2"[THEN "contingent-properties:2"[THEN "≡dfE"]]
              by (metis "qml:2"[axiom_inst] "rule-ui:3" "→E" "&E"(2))
            ultimately AOT_show ‹p & ¬p› for p
              by (metis "raa-cor:3")
          qed
        qed
        AOT_show ‹¬[L]⁻ ≈E [«?P»]›
        proof (rule "raa-cor:2")
          AOT_assume ‹[L]⁻ ≈E [«?P»]›
          AOT_hence ‹[«?P»] ≈E [L]⁻›
            apply (rule "eq-part:2"[unvarify F G, THEN "→E", rotated 2])
             apply "cqt:2[lambda]"
            by (simp add: "rel-neg-T:3")
          AOT_thus ‹[«?P»] ≈E [L]⁻ & ¬[«?P»] ≈E [L]⁻›
            using B "&I" by blast
        qed
      }
    next
      AOT_show ‹◇∃x [«?P»]x›
        using ϑ "&E" by blast
    qed
    AOT_thus ‹◇([L]⁻ ≈E [«?P»] & ◇¬[L]⁻ ≈E [«?P»])›
      using "S5Basic:11" "≡E"(2) by blast
  next
    AOT_show ‹[λx [E!]x & ¬𝒜[E!]x]↓›
      by "cqt:2"
  next
    AOT_show ‹[L]⁻↓›
      by (simp add: "rel-neg-T:3")
  qed
qed


AOT_theorem "approx-cont:2":                                                    (737.2)
  ‹∃F∃G ◇([λz 𝒜[F]z] ≈E G & ◇¬[λz 𝒜[F]z] ≈E G)›
proof -
  let ?P = ‹«[λx E!x & ¬𝒜E!x]»›
  AOT_have ‹◇q0 & ◇¬q0› by (metis q0_prop)
  AOT_hence 1: ‹◇∃x(E!x & ¬𝒜E!x) & ◇¬∃x(E!x & ¬𝒜E!x)›
    by (rule q0_def[THEN "=dfE"(2), rotated])
       (simp add: "log-prop-prop:2")
```

```
AOT_have ϑ: ‹◇∃x [«?P»]x & ◇¬∃x [«?P»]x›
  apply (AOT_subst ‹[«?P»]x› ‹E!x & ¬𝓐E!x› for: x)
   apply (rule "beta-C-meta"[THEN "→E"]; "cqt:2")
  by (fact 1)
show ?thesis
proof (rule "∃I"(1))+
  AOT_have ‹◇[λz 𝓐[L¯]z] ≈E [«?P»] & ◇¬[λz 𝓐[L¯]z] ≈E [«?P»]›
  proof (rule "&I"; rule "RM◇"[THEN "→E"]; (rule "→I")?)
    AOT_modally_strict {
      AOT_assume A: ‹¬∃x [«?P»]x›
      AOT_show ‹[λz 𝓐[L¯]z] ≈E [«?P»]›
      proof (safe intro!: "empty-approx:1"[unvarify F H, THEN "→E"]
                          "rel-neg-T:3" "&I")
        AOT_show ‹[«?P»]↓› by "cqt:2"
      next
        AOT_show ‹¬∃u [λz 𝓐[L¯]z]u›
        proof (rule "raa-cor:2")
          AOT_assume ‹∃u [λz 𝓐[L¯]z]u›
          then AOT_obtain u where ‹[λz 𝓐[L¯]z]u›
            using "Ordinary.∃E"[rotated] by blast
          AOT_hence ‹𝓐[L¯]u›
            using "β→C"(1) "&E" by blast
          moreover AOT_have ‹□¬[L¯]u›
            using "thm-noncont-e-e:2"[THEN "contingent-properties:2"[THEN "≡dfE"]]
            by (metis RN "qml:2"[axiom_inst] "rule-ui:3" "→E" "&E"(2))
          ultimately AOT_show ‹p & ¬p› for p
            by (metis "Act-Sub:3" "KBasic2:1" "≡E"(1) "raa-cor:3" "→E")
        qed
      next
        AOT_show ‹¬∃v [«?P»]v›
        proof (rule "raa-cor:2")
          AOT_assume ‹∃v [«?P»]v›
          then AOT_obtain u where ‹[«?P»]u›
            using "Ordinary.∃E"[rotated] by blast
          AOT_hence ‹[«?P»]u›
            using "&E" by blast
          AOT_hence ‹∃x [«?P»]x›
            by (rule "∃I")
          AOT_thus ‹∃x [«?P»]x & ¬∃x [«?P»]x›
            using A "&I" by blast
        qed
      next
        AOT_show ‹[λz 𝓐[L¯]z]↓› by "cqt:2"
      qed
    }
  next
    AOT_show ‹◇¬∃x [«?P»]x› using ϑ "&E" by blast
  next
    AOT_modally_strict {
      AOT_assume A: ‹∃x [«?P»]x›
      AOT_have B: ‹¬[«?P»] ≈E [λz 𝓐[L¯]z]›
      proof (safe intro!: "empty-approx:2"[unvarify F H, THEN "→E"]
                          "rel-neg-T:3" "&I")
        AOT_show ‹[«?P»]↓› by "cqt:2"
      next
        AOT_obtain x where Px: ‹[«?P»]x›
          using A "∃E" by blast
        AOT_hence ‹E!x & ¬𝓐E!x›
          by (rule "β→C"(1))
        AOT_hence ‹◇E!x›
          by (metis "T◇" "&E"(1) "→E")
        AOT_hence ‹[λx ◇E!x]x›
          by (auto intro!: "β←C"(1) "cqt:2")
```

```
          AOT_hence ‹O!x›
            by (rule AOT_ordinary[THEN "=dfI"(2), rotated]) "cqt:2"
          AOT_hence ‹O!x & [«?P»]x›
            using Px "&I" by blast
          AOT_thus ‹∃u [«?P»]u›
            by (rule "∃I")
        next
          AOT_show ‹¬∃u [λz 𝒜[L⁻]z]u›
          proof (rule "raa-cor:2")
            AOT_assume ‹∃u [λz 𝒜[L⁻]z]u›
            then AOT_obtain u where ‹[λz 𝒜[L⁻]z]u›
              using "Ordinary.∃E"[rotated] by blast
            AOT_hence ‹𝒜[L⁻]u›
              using "β→C"(1) "&E" by blast
            moreover AOT_have ‹□¬[L⁻]u›
              using "thm-noncont-e-e:2"[THEN "contingent-properties:2"[THEN "≡dfE"]]
              by (metis RN "qml:2"[axiom_inst] "rule-ui:3" "→E" "&E"(2))
            ultimately AOT_show ‹p & ¬p› for p
              by (metis "Act-Sub:3" "KBasic2:1" "≡E"(1) "raa-cor:3" "→E")
          qed
        next
          AOT_show ‹[λz 𝒜[L⁻]z]↓› by "cqt:2"
        qed
        AOT_show ‹¬[λz 𝒜[L⁻]z] ≈E [«?P»]›
        proof (rule "raa-cor:2")
          AOT_assume ‹[λz 𝒜[L⁻]z] ≈E [«?P»]›
          AOT_hence ‹[«?P»] ≈E [λz 𝒜[L⁻]z]›
            by (rule "eq-part:2"[unvarify F G, THEN "→E", rotated 2])
               "cqt:2"+
          AOT_thus ‹[«?P»] ≈E [λz 𝒜[L⁻]z] & ¬[«?P»] ≈E [λz 𝒜[L⁻]z]›
            using B "&I" by blast
        qed
      }
    next
      AOT_show ‹◇∃x [«?P»]x›
        using ϑ "&E" by blast
    qed
    AOT_thus ‹◇([λz 𝒜[L⁻]z] ≈E [«?P»] & ◇¬[λz 𝒜[L⁻]z] ≈E [«?P»])›
      using "S5Basic:11" "≡E"(2) by blast
  next
    AOT_show ‹[λx [E!]x & ¬𝒜[E!]x]↓› by "cqt:2"
  next
    AOT_show ‹[L]⁻↓›
      by (simp add: "rel-neg-T:3")
  qed
qed

notepad
begin
  text‹We already have defined being equivalent on the ordinary objects in the
     Extended Relation Comprehension theory.›
  AOT_have ‹F ≡E G ≡df F↓ & G↓ & ∀u ([F]u ≡ [G]u)› for F G
    using eqE by blast
end

AOT_theorem "apE-eqE:1": ‹F ≡E G → F ≈E G›                              (739.1)
proof(rule "→I")
  AOT_assume 0: ‹F ≡E G›
  AOT_have ‹∃R R |: F 1-1⟷E G›
  proof (safe intro!: "∃I"(1)[where τ="«(=E)»"] "equi:2"[THEN "≡dfI"] "&I"
                      "=E[denotes]" "cqt:2[const_var]"[axiom_inst] Ordinary.GEN
                      "→I" "equi:1"[THEN "≡E"(2)])
    fix u
```

```
    AOT_assume Fu: ‹[F]u›
    AOT_hence Gu: ‹[G]u›
      using "≡dfE"[OF eqE, OF 0, THEN "&E"(2),
               THEN "Ordinary.∀E"[where α=u], THEN "≡E"(1)]
            Ordinary.ψ Fu by blast
    AOT_show ‹∃v ([G]v & u =E v & ∀v' ([G]v' & u =E v' → v' =E v))›
      by (safe intro!: "Ordinary.∃I"[where β=u] "&I" GEN "→I" Ordinary.ψ Gu
                       "ord=Eequiv:1"[THEN "→E", OF Ordinary.ψ]
                       "ord=Eequiv:2"[THEN "→E"] dest!: "&E"(2))
  next
    fix v
    AOT_assume Gv: ‹[G]v›
    AOT_hence Fv: ‹[F]v›
      using "≡dfE"[OF eqE, OF 0, THEN "&E"(2),
               THEN "Ordinary.∀E"[where α=v], THEN "≡E"(2)]
            Ordinary.ψ Gv by blast
    AOT_show ‹∃u ([F]u & u =E v & ∀v' ([F]v' & v' =E v → v' =E u))›
      by (safe intro!: "Ordinary.∃I"[where β=v] "&I" GEN "→I" Ordinary.ψ Fv
                       "ord=Eequiv:1"[THEN "→E", OF Ordinary.ψ]
                       "ord=Eequiv:2"[THEN "→E"] dest!: "&E"(2))
  qed
  AOT_thus ‹F ≈E G›
    by (rule "equi:3"[THEN "≡dfI"])
qed

AOT_theorem "apE-eqE:2": ‹(F ≈E G & G ≡E H) → F ≈E H›                    (739.2)
proof(rule "→I")
  AOT_assume ‹F ≈E G & G ≡E H›
  AOT_hence ‹F ≈E G› and ‹G ≈E H›
    using "apE-eqE:1"[THEN "→E"] "&E" by blast+
  AOT_thus ‹F ≈E H›
    by (metis Adjunction "eq-part:3" "vdash-properties:10")
qed


AOT_act_theorem "eq-part-act:1": ‹[λz 𝒜[F]z] ≡E F›                       (740.1)
proof (safe intro!: eqE[THEN "≡dfI"] "&I" "cqt:2" Ordinary.GEN "→I")
  fix u
  AOT_have ‹[λz 𝒜[F]z]u ≡ 𝒜[F]u›
    by (rule "beta-C-meta"[THEN "→E"]) "cqt:2[lambda]"
  also AOT_have ‹... ≡ [F]u›
    using "act-conj-act:4" "logic-actual"[act_axiom_inst, THEN "→E"] by blast
  finally AOT_show ‹[λz 𝒜[F]z]u ≡ [F]u›.
qed

AOT_act_theorem "eq-part-act:2": ‹[λz 𝒜[F]z] ≈E F›                       (740.2)
  by (safe intro!: "apE-eqE:1"[unvarify F, THEN "→E"] "eq-part-act:1") "cqt:2"


AOT_theorem "actuallyF:1": ‹𝒜(F ≈E [λz 𝒜[F]z])›                          (741.1)
proof -
  AOT_have 1: ‹𝒜([F]x ≡ 𝒜[F]x)› for x
    by (meson "Act-Basic:5" "act-conj-act:4" "≡E"(2) "Commutativity of ≡")
  AOT_have ‹𝒜([F]x ≡ [λz 𝒜[F]z]x)› for x
    apply (AOT_subst ‹[λz 𝒜[F]z]x› ‹𝒜[F]x›)
     apply (rule "beta-C-meta"[THEN "→E"])
     apply "cqt:2[lambda]"
    by (fact 1)
  AOT_hence ‹O!x → 𝒜([F]x ≡ [λz 𝒜[F]z]x)› for x
    by (metis "→I")
  AOT_hence ‹∀u 𝒜([F]u ≡ [λz 𝒜[F]z]u)›
    using "∀I" by fast
  AOT_hence 1: ‹𝒜∀u ([F]u ≡ [λz 𝒜[F]z]u)›
```

```
    by (metis "Ordinary.res-var-bound-reas[2]" "→E")
  AOT_modally_strict {
    AOT_have ‹[λz 𝒜[F]z]↓› by "cqt:2"
  } note 2 = this
  AOT_have ‹𝒜(F ≡E [λz 𝒜[F]z])›
    apply (AOT_subst ‹F ≡E [λz 𝒜[F]z]› ‹∀u ([F]u ≡ [λz 𝒜[F]z]u)›)
    using eqE[THEN "≡Df", THEN "≡S"(1), OF "&I",
              OF "cqt:2[const_var]"[axiom_inst], OF 2]
    by (auto simp: 1)
  moreover AOT_have ‹𝒜(F ≡E [λz 𝒜[F]z] → F ≈E [λz 𝒜[F]z])›
    using "apE-eqE:1"[unvarify G, THEN "RA[2]", OF 2] by metis
  ultimately AOT_show ‹𝒜F ≈E [λz 𝒜[F]z]›
    by (metis "act-cond" "→E")
qed

AOT_theorem "actuallyF:2": ‹Rigid([λz 𝒜[F]z])›                              (741.2)
proof(safe intro!: GEN "→I" "df-rigid-rel:1"[THEN "≡dfI"] "&I")
  AOT_show ‹[λz 𝒜[F]z]↓› by "cqt:2"
next
  AOT_show ‹□∀x ([λz 𝒜[F]z]x → □[λz 𝒜[F]z]x)›
  proof(rule RN; rule GEN; rule "→I")
    AOT_modally_strict {
      fix x
      AOT_assume ‹[λz 𝒜[F]z]x›
      AOT_hence ‹𝒜[F]x›
        by (rule "β→C"(1))
      AOT_hence 1: ‹□𝒜[F]x› by (metis "Act-Basic:6" "≡E"(1))
      AOT_show ‹□[λz 𝒜[F]z]x›
        apply (AOT_subst ‹[λz 𝒜[F]z]x› ‹𝒜[F]x›)
         apply (rule "beta-C-meta"[THEN "→E"])
         apply "cqt:2[lambda]"
        by (fact 1)
    }
  qed
qed

AOT_theorem "approx-nec:1": ‹Rigid(F) → F ≈E [λz 𝒜[F]z]›                   (742.1)
proof(rule "→I")
  AOT_assume ‹Rigid([F])›
  AOT_hence A: ‹□∀x ([F]x → □[F]x)›
    using "df-rigid-rel:1"[THEN "≡dfE", THEN "&E"(2)] by blast
  AOT_hence 0: ‹∀x □([F]x → □[F]x)›
    using CBF[THEN "→E"] by blast
  AOT_hence 1: ‹∀x ([F]x → □[F]x)›
    using A "qml:2"[axiom_inst, THEN "→E"] by blast
  AOT_have act_F_den: ‹[λz 𝒜[F]z]↓›
    by "cqt:2"
  AOT_show ‹F ≈E [λz 𝒜[F]z]›
  proof (safe intro!: "apE-eqE:1"[unvarify G, THEN "→E"] eqE[THEN "≡dfI"] "&I"
                      "cqt:2" act_F_den Ordinary.GEN "→I" "≡I")
    fix u
    AOT_assume ‹[F]u›
    AOT_hence ‹□[F]u›
      using 1[THEN "∀E"(2), THEN "→E"] by blast
    AOT_hence act_F_u: ‹𝒜[F]u›
      by (metis "nec-imp-act" "→E")
    AOT_show ‹[λz 𝒜[F]z]u›
      by (auto intro!: "β←C"(1) "cqt:2" act_F_u)
  next
    fix u
    AOT_assume ‹[λz 𝒜[F]z]u›
    AOT_hence ‹𝒜[F]u›
      by (rule "β→C"(1))
```

```
    AOT_thus ‹[F]u›
      using 0[THEN "∀E"(2)]
      by (metis "≡E"(1) "sc-eq-fur:2" "→E")
  qed
qed


AOT_theorem "approx-nec:2":                                          (742.2)
  ‹F ≈E G ≡ ∀H ([λz 𝒜[H]z] ≈E F ≡ [λz 𝒜[H]z] ≈E G)›
proof(rule "≡I"; rule "→I")
  AOT_assume 0: ‹F ≈E G›
  AOT_assume 0: ‹F ≈E G›
  AOT_hence ‹∀H (H ≈E F ≡ H ≈E G)›
    using "eq-part:4"[THEN "≡E"(1), OF 0] by blast
  AOT_have ‹[λz 𝒜[H]z] ≈E F ≡ [λz 𝒜[H]z] ≈E G› for H
    by (rule "∀E"(1)[OF "eq-part:4"[THEN "≡E"(1), OF 0]]) "cqt:2"
  AOT_thus ‹∀H ([λz 𝒜[H]z] ≈E F ≡ [λz 𝒜[H]z] ≈E G)›
    by (rule GEN)
next
  AOT_assume 0: ‹∀H ([λz 𝒜[H]z] ≈E F ≡ [λz 𝒜[H]z] ≈E G)›
  AOT_obtain H where ‹Rigidifies(H,F)›
    using "rigid-der:3" "∃E" by metis
  AOT_hence H: ‹Rigid(H) & ∀x ([H]x ≡ [F]x)›
    using "df-rigid-rel:2"[THEN "≡dfE"] by blast
  AOT_have H_rigid: ‹□∀x ([H]x → □[H]x)›
    using H[THEN "&E"(1), THEN "df-rigid-rel:1"[THEN "≡dfE"], THEN "&E"(2)].
  AOT_hence ‹∀x □([H]x → □[H]x)›
    using "CBF" "vdash-properties:10" by blast
  AOT_hence ‹□([H]x → □[H]x)› for x using "∀E"(2) by blast
  AOT_hence rigid: ‹[H]x ≡ 𝒜[H]x› for x
     by (metis "≡E"(6) "oth-class-taut:3:a" "sc-eq-fur:2" "→E")
  AOT_have ‹H ≡E F›
  proof (safe intro!: eqE[THEN "≡dfI"] "&I" "cqt:2" Ordinary.GEN "→I")
    AOT_show ‹[H]u ≡ [F]u› for u using H[THEN "&E"(2)] "∀E"(2) by fast
  qed
  AOT_hence ‹H ≈E F›
    by (rule "apE-eqE:2"[THEN "→E", OF "&I", rotated])
       (simp add: "eq-part:1")
  AOT_hence F_approx_H: ‹F ≈E H›
    by (metis "eq-part:2" "→E")
  moreover AOT_have H_eq_act_H: ‹H ≡E [λz 𝒜[H]z]›
  proof (safe intro!: eqE[THEN "≡dfI"] "&I" "cqt:2" Ordinary.GEN "→I")
    AOT_show ‹[H]u ≡ [λz 𝒜[H]z]u› for u
      apply (AOT_subst ‹[λz 𝒜[H]z]u› ‹𝒜[H]u›)
       apply (rule "beta-C-meta"[THEN "→E"])
       apply "cqt:2[lambda]"
      using rigid by blast
  qed
  AOT_have a: ‹F ≈E [λz 𝒜[H]z]›
    apply (rule "apE-eqE:2"[unvarify H, THEN "→E"])
     apply "cqt:2[lambda]"
    using F_approx_H H_eq_act_H "&I" by blast
  AOT_hence ‹[λz 𝒜[H]z] ≈E F›
    apply (rule "eq-part:2"[unvarify G, THEN "→E", rotated])
    by "cqt:2[lambda]"
  AOT_hence b: ‹[λz 𝒜[H]z] ≈E G›
    by (rule 0[THEN "∀E"(1), THEN "≡E"(1), rotated]) "cqt:2"
  AOT_show ‹F ≈E G›
    by (rule "eq-part:3"[unvarify G, THEN "→E", rotated, OF "&I", OF a, OF b])
       "cqt:2"
qed

AOT_theorem "approx-nec:3":                                          (742.3)
```

```
  ‹(Rigid(F) & Rigid(G)) → □(F ≈E G → □F ≈E G)›
proof (rule "→I")
  AOT_assume ‹Rigid(F) & Rigid(G)›
  AOT_hence ‹□∀x([F]x → □[F]x)› and ‹□∀x([G]x → □[G]x)›
    using "df-rigid-rel:1"[THEN "≡dfE", THEN "&E"(2)] "&E" by blast+
  AOT_hence ‹□(□∀x([F]x → □[F]x) & □∀x([G]x → □[G]x))›
    using "KBasic:3" "4" "&I" "≡E"(2) "vdash-properties:10" by meson
  moreover AOT_have ‹□(□∀x([F]x → □[F]x) & □∀x([G]x → □[G]x)) →
                     □(F ≈E G → □F ≈E G)›
  proof(rule RM; rule "→I"; rule "→I")
    AOT_modally_strict {
      AOT_assume ‹□∀x([F]x → □[F]x) & □∀x([G]x → □[G]x)›
      AOT_hence ‹□∀x([F]x → □[F]x)› and ‹□∀x([G]x → □[G]x)›
        using "&E" by blast+
      AOT_hence ‹∀x□([F]x → □[F]x)› and ‹∀x□([G]x → □[G]x)›
        using CBF[THEN "→E"] by blast+
      AOT_hence F_nec: ‹□([F]x → □[F]x)›
            and G_nec: ‹□([G]x → □[G]x)› for x
        using "∀E"(2) by blast+
      AOT_assume ‹F ≈E G›
      AOT_hence ‹∃R R |: F 1-1⟷E G›
        by (metis "≡dfE" "equi:3")
      then AOT_obtain R where ‹R |: F 1-1⟷E G›
        using "∃E"[rotated] by blast
      AOT_hence C1: ‹∀u ([F]u → ∃!v ([G]v & [R]uv))›
            and C2: ‹∀v ([G]v → ∃!u ([F]u & [R]uv))›
        using "equi:2"[THEN "≡dfE"] "&E" by blast+
      AOT_obtain R' where ‹Rigidifies(R', R)›
        using "rigid-der:3" "∃E"[rotated] by blast
      AOT_hence 1: ‹Rigid(R') & ∀x₁...∀xₙ ([R']x₁...xₙ ≡ [R]x₁...xₙ)›
        using "df-rigid-rel:2"[THEN "≡dfE"] by blast
      AOT_hence ‹□∀x₁...∀xₙ ([R']x₁...xₙ → □[R']x₁...xₙ)›
        using "df-rigid-rel:1"[THEN "≡dfE"] "&E" by blast
      AOT_hence ‹∀x₁...∀xₙ (◇[R']x₁...xₙ → □[R']x₁...xₙ)›
        using "≡E"(1) "rigid-rel-thms:1" by blast
      AOT_hence D: ‹∀x₁∀x₂ (◇[R']x₁x₂ → □[R']x₁x₂)›
        using tuple_forall[THEN "≡dfE"] by blast
      AOT_have E: ‹∀x₁∀x₂ ([R']x₁x₂ ≡ [R]x₁x₂)›
        using tuple_forall[THEN "≡dfE", OF 1[THEN "&E"(2)]] by blast
      AOT_have ‹∀u □([F]u → ∃!v ([G]v & [R']uv))›
           and ‹∀v □([G]v → ∃!u ([F]u & [R']uv))›
      proof (safe intro!: Ordinary.GEN "→I")
        fix u
        AOT_show ‹□([F]u → ∃!v ([G]v & [R']uv))›
        proof (rule "raa-cor:1")
          AOT_assume ‹¬□([F]u → ∃!v ([G]v & [R']uv))›
          AOT_hence 1: ‹◇¬([F]u → ∃!v ([G]v & [R']uv))›
            using "KBasic:11" "≡E"(1) by blast
          AOT_have ‹◇([F]u & ¬∃!v ([G]v & [R']uv))›
            apply (AOT_subst ‹[F]u & ¬∃!v ([G]v & [R']uv)›
                             ‹¬([F]u → ∃!v ([G]v & [R']uv))›)
             apply (meson "≡E"(6) "oth-class-taut:1:b" "oth-class-taut:3:a")
            by (fact 1)
          AOT_hence A: ‹◇[F]u & ◇¬∃!v ([G]v & [R']uv)›
            using "KBasic2:3" "→E" by blast
          AOT_hence ‹□[F]u›
            using F_nec "&E"(1) "≡E"(1) "sc-eq-box-box:1" "→E" by blast
          AOT_hence ‹[F]u›
            by (metis "qml:2"[axiom_inst] "→E")
          AOT_hence ‹∃!v ([G]v & [R]uv)›
            using C1[THEN "Ordinary.∀E", THEN "→E"] by blast
          AOT_hence ‹∃v ([G]v & [R]uv & ∀v' ([G]v' & [R]uv' → v' =E v))›
            using "equi:1"[THEN "≡E"(1)] by auto
```

```
      then AOT_obtain a where
        a_prop: ‹O!a & ([G]a & [R]ua & ∀v' ([G]v' & [R]uv' → v' =E a))›
        using "∃E"[rotated] by blast
      AOT_have ‹∃v □([G]v & [R']uv & ∀v' ([G]v' & [R']uv' → v' =E v))›
      proof(safe intro!: "∃I"(2)[where β=a] "&I" a_prop[THEN "&E"(1)]
                         "KBasic:3"[THEN "≡E"(2)])
        AOT_show ‹□[G]a›
          using a_prop[THEN "&E"(2), THEN "&E"(1), THEN "&E"(1)]
          by (metis G_nec "qml:2"[axiom_inst] "→E")
      next
        AOT_show ‹□[R']ua›
          using D[THEN "∀E"(2), THEN "∀E"(2), THEN "→E"]
             E[THEN "∀E"(2), THEN "∀E"(2), THEN "≡E"(2),
              OF a_prop[THEN "&E"(2), THEN "&E"(1), THEN "&E"(2)]]
          by (metis "T◇" "→E")
      next
        AOT_have ‹∀v' □([G]v' & [R']uv' → v' =E a)›
        proof (rule Ordinary.GEN; rule "raa-cor:1")
          fix v'
          AOT_assume ‹¬□([G]v' & [R']uv' → v' =E a)›
          AOT_hence ‹◇¬([G]v' & [R']uv' → v' =E a)›
            by (metis "KBasic:11" "≡E"(1))
          AOT_hence ‹◇([G]v' & [R']uv' & ¬v' =E a)›
            by (AOT_subst ‹[G]v' & [R']uv' & ¬v' =E a›
                       ‹¬([G]v' & [R']uv' → v' =E a)›)
                 (meson "≡E"(6) "oth-class-taut:1:b" "oth-class-taut:3:a")
          AOT_hence 1: ‹◇[G]v'› and 2: ‹◇[R']uv'› and 3: ‹◇¬v' =E a›
            using "KBasic2:3"[THEN "→E", THEN "&E"(1)]
                "KBasic2:3"[THEN "→E", THEN "&E"(2)] by blast+
          AOT_have Gv': ‹[G]v'› using G_nec 1
            by (meson "B◇" "KBasic:13" "→E")
          AOT_have ‹□[R']uv'›
            using 2 D[THEN "∀E"(2), THEN "∀E"(2), THEN "→E"] by blast
          AOT_hence R'uv': ‹[R']uv'›
            by (metis "B◇" "T◇" "→E")
          AOT_hence ‹[R]uv'›
            using E[THEN "∀E"(2), THEN "∀E"(2), THEN "≡E"(1)] by blast
          AOT_hence ‹v' =E a›
            using a_prop[THEN "&E"(2), THEN "&E"(2), THEN "Ordinary.∀E",
                      THEN "→E", OF "&I", OF Gv'] by blast
          AOT_hence ‹□(v' =E a)›
            by (metis "id-nec3:1" "≡E"(4) "raa-cor:3")
          moreover AOT_have ‹¬□(v' =E a)›
            using 3 "KBasic:11" "≡E"(2) by blast
          ultimately AOT_show ‹□(v' =E a) & ¬□(v' =E a)›
            using "&I" by blast
        qed
        AOT_thus ‹□∀v'([G]v' & [R']uv' → v' =E a)›
          using "Ordinary.res-var-bound-reas[BF]" "→E" by fast
      qed
      AOT_hence ‹□∃v ([G]v & [R']uv & ∀v' ([G]v' & [R']uv' → v' =E v))›
        using "Ordinary.res-var-bound-reas[Buridan]" "→E" by fast
      AOT_hence ‹□∃!v ([G]v & [R']uv)›
        by (AOT_subst_thm "equi:1")
      moreover AOT_have ‹¬□∃!v ([G]v & [R']uv)›
        using A[THEN "&E"(2)] "KBasic:11"[THEN "≡E"(2)] by blast
      ultimately AOT_show ‹□∃!v ([G]v & [R']uv) & ¬□∃!v ([G]v & [R']uv)›
        by (rule "&I")
     qed
    next
     fix v
     AOT_show ‹□([G]v → ∃!u ([F]u & [R']uv))›
     proof (rule "raa-cor:1")
```

```
        AOT_assume ‹¬□([G]v → ∃!u ([F]u & [R']uv))›
        AOT_hence 1: ‹◇¬([G]v → ∃!u ([F]u & [R']uv))›
          using "KBasic:11" "≡E"(1) by blast
        AOT_hence ‹◇([G]v & ¬∃!u ([F]u & [R']uv))›
          by (AOT_subst ‹[G]v & ¬∃!u ([F]u & [R']uv)›
                        ‹¬([G]v → ∃!u ([F]u & [R']uv))›)
             (meson "≡E"(6) "oth-class-taut:1:b" "oth-class-taut:3:a")
        AOT_hence A: ‹◇[G]v & ◇¬∃!u ([F]u & [R']uv)›
          using "KBasic2:3" "→E" by blast
        AOT_hence ‹□[G]v›
          using G_nec "&E"(1) "≡E"(1) "sc-eq-box-box:1" "→E" by blast
        AOT_hence ‹[G]v› by (metis "qml:2"[axiom_inst] "→E")
        AOT_hence ‹∃!u ([F]u & [R]uv)›
          using C2[THEN "Ordinary.∀E", THEN "→E"] by blast
        AOT_hence ‹∃u ([F]u & [R]uv & ∀u' ([F]u' & [R]u'v → u' =E u))›
          using "equi:1"[THEN "≡E"(1)] by auto
        then AOT_obtain a where
            a_prop: ‹O!a & ([F]a & [R]av & ∀u' ([F]u' & [R]u'v → u' =E a))›
          using "∃E"[rotated] by blast
        AOT_have ‹∃u □([F]u & [R']uv & ∀u' ([F]u' & [R']u'v → u' =E u))›
        proof(safe intro!: "∃I"(2)[where β=a] "&I" a_prop[THEN "&E"(1)]
                           "KBasic:3"[THEN "≡E"(2)])
          AOT_show ‹□[F]a›
            using a_prop[THEN "&E"(2), THEN "&E"(1), THEN "&E"(1)]
            by (metis F_nec "qml:2"[axiom_inst] "→E")
        next
          AOT_show ‹□[R']av›
            using D[THEN "∀E"(2), THEN "∀E"(2), THEN "→E"]
                  E[THEN "∀E"(2), THEN "∀E"(2), THEN "≡E"(2),
                    OF a_prop[THEN "&E"(2), THEN "&E"(1), THEN "&E"(2)]]
            by (metis "T◇" "→E")
        next
          AOT_have ‹∀u' □([F]u' & [R']u'v → u' =E a)›
          proof (rule Ordinary.GEN; rule "raa-cor:1")
            fix u'
            AOT_assume ‹¬□([F]u' & [R']u'v → u' =E a)›
            AOT_hence ‹◇¬([F]u' & [R']u'v → u' =E a)›
              by (metis "KBasic:11" "≡E"(1))
            AOT_hence ‹◇([F]u' & [R']u'v & ¬u' =E a)›
              by (AOT_subst ‹[F]u' & [R']u'v & ¬u' =E a›
                            ‹¬([F]u' & [R']u'v → u' =E a)›)
                 (meson "≡E"(6) "oth-class-taut:1:b" "oth-class-taut:3:a")
            AOT_hence 1: ‹◇[F]u'› and 2: ‹◇[R']u'v› and 3: ‹◇¬u' =E a›
              using "KBasic2:3"[THEN "→E", THEN "&E"(1)]
                    "KBasic2:3"[THEN "→E", THEN "&E"(2)] by blast+
            AOT_have Fu': ‹[F]u'› using F_nec 1
              by (meson "B◇" "KBasic:13" "→E")
            AOT_have ‹□[R']u'v›
              using 2 D[THEN "∀E"(2), THEN "∀E"(2), THEN "→E"] by blast
            AOT_hence R'u'v: ‹[R']u'v›
              by (metis "B◇" "T◇" "→E")
            AOT_hence ‹[R]u'v›
              using E[THEN "∀E"(2), THEN "∀E"(2), THEN "≡E"(1)] by blast
            AOT_hence ‹u' =E a›
              using a_prop[THEN "&E"(2), THEN "&E"(2), THEN "Ordinary.∀E",
                           THEN "→E", OF "&I", OF Fu'] by blast
            AOT_hence ‹□(u' =E a)›
              by (metis "id-nec3:1" "≡E"(4) "raa-cor:3")
            moreover AOT_have ‹¬□(u' =E a)›
              using 3 "KBasic:11" "≡E"(2) by blast
            ultimately AOT_show ‹□(u' =E a) & ¬□(u' =E a)›
              using "&I" by blast
          qed
```

```
            AOT_thus ‹□∀u'([F]u' & [R']u'v → u' =E a)›
              using "Ordinary.res-var-bound-reas[BF]" "→E" by fast
          qed
          AOT_hence 1: ‹□∃u ([F]u & [R']uv & ∀u' ([F]u' & [R']u'v → u' =E u))›
            using "Ordinary.res-var-bound-reas[Buridan]" "→E" by fast
          AOT_hence ‹□∃!u ([F]u & [R']uv)›
            by (AOT_subst_thm "equi:1")
          moreover AOT_have ‹¬□∃!u ([F]u & [R']uv)›
            using A[THEN "&E"(2)] "KBasic:11"[THEN "≡E"(2)] by blast
          ultimately AOT_show ‹□∃!u ([F]u & [R']uv) & ¬□∃!u ([F]u & [R']uv)›
            by (rule "&I")
        qed
      qed
      AOT_hence ‹□∀u ([F]u → ∃!v ([G]v & [R']uv))›
            and ‹□∀v ([G]v → ∃!u ([F]u & [R']uv))›
        using "Ordinary.res-var-bound-reas[BF]"[THEN "→E"] by auto
      moreover AOT_have ‹□[R']↓› and ‹□[F]↓› and ‹□[G]↓›
        by (simp_all add: "ex:2:a")
      ultimately AOT_have ‹□([R']↓ & [F]↓ & [G]↓ & ∀u ([F]u → ∃!v ([G]v & [R']uv)) &
                                                ∀v ([G]v → ∃!u ([F]u & [R']uv)))›
        using "KBasic:3" "&I" "≡E"(2) by meson
      AOT_hence ‹□R' |: F 1-1⟷E G›
        by (AOT_subst_def "equi:2")
      AOT_hence ‹∃R □R |: F 1-1⟷E G›
        by (rule "∃I"(2))
      AOT_hence ‹□∃R R |: F 1-1⟷E G›
        by (metis Buridan "→E")
      AOT_thus ‹□F ≈E G›
        by (AOT_subst_def "equi:3")
    }
  qed
  ultimately AOT_show ‹□(F ≈E G → □F ≈E G)›
    using "→E" by blast
qed


AOT_define numbers :: ‹τ ⇒ τ ⇒ φ› (‹Numbers'(_,_')›)                                    (744)
  ‹Numbers(x,G) ≡df A!x & G↓ & ∀F(x[F] ≡ [λz 𝒜[F]z] ≈E G)›

AOT_theorem "numbers[den]":                                                              (744)
  ‹Π↓ → (Numbers(κ, Π) ≡ A!κ & ∀F(κ[F] ≡ [λz 𝒜[F]z] ≈E Π))›
  apply (safe intro!: numbers[THEN "≡dfI"] "&I" "≡I" "→I" "cqt:2"
              dest!: numbers[THEN "≡dfE"])
  using "&E" by blast+

AOT_theorem "num-tran:1":                                                                (745.1)
  ‹G ≈E H → (Numbers(x, G) ≡ Numbers(x, H))›
proof (safe intro!: "→I" "≡I")
  AOT_assume 0: ‹G ≈E H›
  AOT_assume ‹Numbers(x, G)›
  AOT_hence Ax: ‹A!x› and ϑ: ‹∀F (x[F] ≡ [λz 𝒜[F]z] ≈E G)›
    using numbers[THEN "≡dfE"] "&E" by blast+
  AOT_show ‹Numbers(x, H)›
  proof(safe intro!: numbers[THEN "≡dfI"] "&I" Ax "cqt:2" GEN)
    fix F
    AOT_have ‹x[F] ≡ [λz 𝒜[F]z] ≈E G›
      using ϑ[THEN "∀E"(2)].
    also AOT_have ‹... ≡ [λz 𝒜[F]z] ≈E H›
      using 0 "approx-nec:2"[THEN "≡E"(1), THEN "∀E"(2)] by metis
    finally AOT_show ‹x[F] ≡ [λz 𝒜[F]z] ≈E H›.
  qed
next
  AOT_assume ‹G ≈E H›
```

```
  AOT_hence 0: ‹H ≈E G›
    by (metis "eq-part:2" "→E")
  AOT_assume ‹Numbers(x, H)›
  AOT_hence Ax: ‹A!x› and ϑ: ‹∀F (x[F] ≡ [λz 𝒜[F]z] ≈E H)›
    using numbers[THEN "≡dfE"] "&E" by blast+
  AOT_show ‹Numbers(x, G)›
  proof(safe intro!: numbers[THEN "≡dfI"] "&I" Ax "cqt:2"  GEN)
    fix F
    AOT_have ‹x[F] ≡ [λz 𝒜[F]z] ≈E H›
      using ϑ[THEN "∀E"(2)].
    also AOT_have ‹... ≡ [λz 𝒜[F]z] ≈E G›
      using 0 "approx-nec:2"[THEN "≡E"(1), THEN "∀E"(2)] by metis
    finally AOT_show ‹x[F] ≡ [λz 𝒜[F]z] ≈E G›.
  qed
qed

AOT_theorem "num-tran:2":                                                  (745.2)
  ‹(Numbers(x, G) & Numbers(x,H)) → G ≈E H›
proof (rule "→I"; frule "&E"(1); drule "&E"(2))
  AOT_assume ‹Numbers(x,G)›
  AOT_hence ‹∀F (x[F] ≡ [λz 𝒜[F]z] ≈E G)›
    using numbers[THEN "≡dfE"] "&E" by blast
  AOT_hence 1: ‹x[F] ≡ [λz 𝒜[F]z] ≈E G› for F
    using "∀E"(2) by blast
  AOT_assume ‹Numbers(x,H)›
  AOT_hence ‹∀F (x[F] ≡ [λz 𝒜[F]z] ≈E H)›
    using numbers[THEN "≡dfE"] "&E" by blast
  AOT_hence ‹x[F] ≡ [λz 𝒜[F]z] ≈E H› for F
    using "∀E"(2) by blast
  AOT_hence ‹[λz 𝒜[F]z] ≈E G ≡ [λz 𝒜[F]z] ≈E H› for F
    by (metis "1" "≡E"(6))
  AOT_thus ‹G ≈E H›
    using "approx-nec:2"[THEN "≡E"(2), OF GEN] by blast
qed

AOT_theorem "num-tran:3":                                                  (745.3)
  ‹G ≡E H → (Numbers(x, G) ≡ Numbers(x, H))›
  using "apE-eqE:1" "Hypothetical Syllogism" "num-tran:1" by blast

AOT_theorem "pre-Hume":                                                    (746)
  ‹(Numbers(x,G) & Numbers(y,H)) → (x = y ≡ G ≈E H)›
proof(safe intro!: "→I" "≡I"; frule "&E"(1); drule "&E"(2))
  AOT_assume ‹Numbers(x, G)›
  moreover AOT_assume ‹x = y›
  ultimately AOT_have ‹Numbers(y, G)› by (rule "rule=E")
  moreover AOT_assume ‹Numbers(y, H)›
  ultimately AOT_show ‹G ≈E H› using "num-tran:2" "→E" "&I" by blast
next
  AOT_assume ‹Numbers(x, G)›
  AOT_hence Ax: ‹A!x› and xF: ‹∀F (x[F] ≡ [λz 𝒜[F]z] ≈E G)›
    using numbers[THEN "≡dfE"] "&E" by blast+
  AOT_assume ‹Numbers(y, H)›
  AOT_hence Ay: ‹A!y› and yF: ‹∀F (y[F] ≡ [λz 𝒜[F]z] ≈E H)›
    using numbers[THEN "≡dfE"] "&E" by blast+
  AOT_assume G_approx_H: ‹G ≈E H›
  AOT_show ‹x = y›
  proof(rule "ab-obey:1"[THEN "→E", THEN "→E", OF "&I", OF Ax, OF Ay]; rule GEN)
    fix F
    AOT_have ‹x[F] ≡ [λz 𝒜[F]z] ≈E G›
      using xF[THEN "∀E"(2)].
    also AOT_have ‹... ≡ [λz 𝒜[F]z] ≈E H›
      using "approx-nec:2"[THEN "≡E"(1), OF G_approx_H, THEN "∀E"(2)].
    also AOT_have ‹... ≡ y[F]›
```

```
      using yF[THEN "∀E"(2), symmetric].
    finally AOT_show ‹x[F] ≡ y[F]›.
  qed
qed

AOT_theorem "two-num-not":                                                  (748)
  ‹∃u∃v(u ≠ v) → ∃x∃G∃H(Numbers(x,G) & Numbers(x, H) & ¬G ≡E H)›
proof (rule "→I")
  AOT_have eqE_den: ‹[λx x =E y]↓› for y by "cqt:2"
  AOT_assume ‹∃u∃v(u ≠ v)›
  then AOT_obtain c where Oc: ‹O!c› and ‹∃v (c ≠ v)›
    using "&E" "∃E"[rotated] by blast
  then AOT_obtain d where Od: ‹O!d› and c_noteq_d: ‹c ≠ d›
    using "&E" "∃E"[rotated] by blast
  AOT_hence c_noteqE_d: ‹c ≠E d›
    using "=E-simple:2"[THEN "→E"] "=E-simple:2" "≡E"(2) "modus-tollens:1"
          "=-infix" "≡dfE" "thm-neg=E" by fast
  AOT_hence not_c_eqE_d: ‹¬c =E d›
    using "≡E"(1) "thm-neg=E" by blast
  AOT_have ‹∃x (A!x & ∀F (x[F] ≡ [λz 𝒜[F]z] ≈E [λx x =E c]))›
    by (simp add: "A-objects"[axiom_inst])
  then AOT_obtain a where a_prop: ‹A!a & ∀F (a[F] ≡ [λz 𝒜[F]z] ≈E [λx x =E c])›
    using "∃E"[rotated] by blast
  AOT_have ‹∃x (A!x & ∀F (x[F] ≡ [λz 𝒜[F]z] ≈E [λx x =E d]))›
    by (simp add: "A-objects" "vdash-properties:1[2]")
  then AOT_obtain b where b_prop: ‹A!b & ∀F (b[F] ≡ [λz 𝒜[F]z] ≈E [λx x =E d])›
    using "∃E"[rotated] by blast
  AOT_have num_a_eq_c: ‹Numbers(a, [λx x =E c])›
    by (safe intro!: numbers[THEN "≡dfI"] "&I" a_prop[THEN "&E"(1)]
                     a_prop[THEN "&E"(2)]) "cqt:2"
  moreover AOT_have num_b_eq_d: ‹Numbers(b, [λx x =E d])›
    by (safe intro!: numbers[THEN "≡dfI"] "&I" b_prop[THEN "&E"(1)]
                     b_prop[THEN "&E"(2)]) "cqt:2"
  moreover AOT_have ‹[λx x =E c] ≈E [λx x =E d]›
  proof (rule "equi:3"[THEN "≡dfI"])
    let ?R = ‹«[λxy (x =E c & y =E d)]»›
    AOT_have Rcd: ‹[«?R»]cd›
      by (auto intro!: "β←C"(1) "cqt:2" "&I" prod_denotesI
                       "ord=Eequiv:1"[THEN "→E"] Od Oc)
    AOT_show ‹∃R R |: [λx x =E c] 1-1⟷E [λx x =E d]›
    proof (safe intro!: "∃I"(1)[where τ=‹?R›] "equi:2"[THEN "≡dfI"] "&I"
                        eqE_den Ordinary.GEN "→I")
      AOT_show ‹«?R»↓› by "cqt:2"
    next
      fix u
      AOT_assume ‹[λx x =E c]u›
      AOT_hence ‹u =E c›
        by (metis "β→C"(1))
      AOT_hence u_is_c: ‹u = c›
        by (metis "=E-simple:2" "→E")
      AOT_show ‹∃!v ([λx x =E d]v & [«?R»]uv)›
      proof (safe intro!: "equi:1"[THEN "≡E"(2)] "∃I"(2)[where β=d] "&I"
                          Od Ordinary.GEN "→I")
        AOT_show ‹[λx x =E d]d›
          by (auto intro!: "β←C"(1) "cqt:2" "ord=Eequiv:1"[THEN "→E", OF Od])
      next
        AOT_show ‹[«?R»]ud›
          using u_is_c[symmetric] Rcd "rule=E" by fast
      next
        fix v
        AOT_assume ‹[λx x =E d]v & [«?R»]uv›
        AOT_thus ‹v =E d›
          by (metis "β→C"(1) "&E"(1))
```

```
      qed
    next
      fix v
      AOT_assume ‹[λx x =E d]v›
      AOT_hence ‹v =E d›
        by (metis "β→C"(1))
      AOT_hence v_is_d: ‹v = d›
        by (metis "=E-simple:2" "→E")
      AOT_show ‹∃!u ([λx x =E c]u & [«?R»]uv)›
      proof (safe intro!: "equi:1"[THEN "≡E"(2)] "∃I"(2)[where β=c] "&I"
                          Oc Ordinary.GEN "→I")
        AOT_show ‹[λx x =E c]c›
          by (auto intro!: "β←C"(1) "cqt:2" "ord=Eequiv:1"[THEN "→E", OF Oc])
      next
        AOT_show ‹[«?R»]cv›
          using v_is_d[symmetric] Rcd "rule=E" by fast
      next
        fix u
        AOT_assume ‹[λx x =E c]u & [«?R»]uv›
        AOT_thus ‹u =E c›
          by (metis "β→C"(1) "&E"(1))
      qed
    next
      AOT_show ‹«?R»↓›
        by "cqt:2"
    qed
  qed
  ultimately AOT_have ‹a = b›
    using "pre-Hume"[unvarify G H, OF eqE_den, OF eqE_den, THEN "→E",
                     OF "&I", THEN "≡E"(2)] by blast
  AOT_hence num_a_eq_d: ‹Numbers(a, [λx x =E d])›
    using num_b_eq_d "rule=E" id_sym by fast
  AOT_have not_equiv: ‹¬[λx x =E c] ≡E [λx x =E d]›
  proof (rule "raa-cor:2")
    AOT_assume ‹[λx x =E c] ≡E [λx x =E d]›
    AOT_hence ‹[λx x =E c]c ≡ [λx x =E d]c›
      using eqE[THEN "≡dfE", THEN "&E"(2), THEN "∀E"(2), THEN "→E"] Oc by blast
    moreover AOT_have ‹[λx x =E c]c›
      by (auto intro!: "β←C"(1) "cqt:2" "ord=Eequiv:1"[THEN "→E", OF Oc])
    ultimately AOT_have ‹[λx x =E d]c›
      using "≡E"(1) by blast
    AOT_hence ‹c =E d›
      by (rule "β→C"(1))
    AOT_thus ‹c =E d & ¬c =E d›
      using not_c_eqE_d "&I" by blast
  qed
  AOT_show ‹∃x ∃G ∃H (Numbers(x,G) & Numbers(x,H) & ¬G ≡E H)›
    apply (rule "∃I"(2)[where β=a])
    apply (rule "∃I"(1)[where τ=‹«[λx x =E c]»›])
     apply (rule "∃I"(1)[where τ=‹«[λx x =E d]»›])
    by (safe intro!: eqE_den "&I" num_a_eq_c num_a_eq_d not_equiv)
qed

AOT_theorem "num:1": ‹∃x Numbers(x,G)›                                    (749.1)
  by (AOT_subst ‹Numbers(x,G)› ‹[A!]x & ∀F (x[F] ≡ [λz 𝒜[F]z] ≈E G)› for: x)
     (auto simp: "numbers[den]"[THEN "→E", OF "cqt:2[const_var]"[axiom_inst]]
                 "A-objects"[axiom_inst])

AOT_theorem "num:2": ‹∃!x Numbers(x,G)›                                   (749.2)
  by (AOT_subst ‹Numbers(x,G)› ‹[A!]x & ∀F (x[F] ≡ [λz 𝒜[F]z] ≈E G)› for: x)
     (auto simp: "numbers[den]"[THEN "→E", OF "cqt:2[const_var]"[axiom_inst]]
                 "A-objects!")
```

```
AOT_theorem "num-cont:1":                                          (750.1)
  ‹∃x∃G(Numbers(x, G) & ¬□Numbers(x, G))›
proof -
  AOT_have ‹∃F∃G ◇([λz 𝒜[F]z] ≈E G & ◇¬[λz 𝒜[F]z] ≈E G)›
    using "approx-cont:2".
  then AOT_obtain F where ‹∃G ◇([λz 𝒜[F]z] ≈E G & ◇¬[λz 𝒜[F]z] ≈E G)›
    using "∃E"[rotated] by blast
  then AOT_obtain G where ‹◇([λz 𝒜[F]z] ≈E G & ◇¬[λz 𝒜[F]z] ≈E G)›
    using "∃E"[rotated] by blast
  AOT_hence ϑ: ‹◇[λz 𝒜[F]z] ≈E G› and ζ: ‹◇¬[λz 𝒜[F]z] ≈E G›
    using "KBasic2:3"[THEN "→E"] "&E" "4◇"[THEN "→E"] by blast+
  AOT_obtain a where ‹Numbers(a, G)›
    using "num:1" "∃E"[rotated] by blast
  moreover AOT_have ‹¬□Numbers(a, G)›
  proof (rule "raa-cor:2")
    AOT_assume ‹□Numbers(a, G)›
    AOT_hence ‹□([A!]a & G↓ & ∀F (a[F] ≡ [λz 𝒜[F]z] ≈E G))›
      by (AOT_subst_def (reverse) numbers)
    AOT_hence ‹□A!a› and ‹□∀F (a[F] ≡ [λz 𝒜[F]z] ≈E G)›
      using "KBasic:3"[THEN "≡E"(1)] "&E" by blast+
    AOT_hence ‹∀F □(a[F] ≡ [λz 𝒜[F]z] ≈E G)›
      using CBF[THEN "→E"] by blast
    AOT_hence ‹□(a[F] ≡ [λz 𝒜[F]z] ≈E G)›
      using "∀E"(2) by blast
    AOT_hence A: ‹□(a[F] → [λz 𝒜[F]z] ≈E G)›
          and B: ‹□([λz 𝒜[F]z] ≈E G → a[F])›
      using "KBasic:4"[THEN "≡E"(1)] "&E" by blast+
    AOT_have ‹□(¬[λz 𝒜[F]z] ≈E G → ¬a[F])›
      apply (AOT_subst ‹¬[λz 𝒜[F]z] ≈E G → ¬a[F]› ‹a[F] → [λz 𝒜[F]z] ≈E G›)
       using "≡I" "useful-tautologies:4" "useful-tautologies:5" apply presburger
       by (fact A)
     AOT_hence ‹◇¬a[F]›
       by (metis "KBasic:13" ζ "→E")
    AOT_hence ‹¬a[F]›
      by (metis "KBasic:11" "en-eq:2[1]" "≡E"(2) "≡E"(4))
    AOT_hence ‹¬◇a[F]›
      by (metis "en-eq:3[1]" "≡E"(4))
    moreover AOT_have ‹◇a[F]›
      by (meson B ϑ "KBasic:13" "→E")
    ultimately AOT_show ‹◇a[F] & ¬◇a[F]›
      using "&I" by blast
  qed

  ultimately AOT_have ‹Numbers(a, G) & ¬□Numbers(a, G)›
    using "&I" by blast
  AOT_hence ‹∃G (Numbers(a, G) & ¬□Numbers(a, G))›
    by (rule "∃I")
  AOT_thus ‹∃x∃G (Numbers(x, G) & ¬□Numbers(x, G))›
    by (rule "∃I")
qed

AOT_theorem "num-cont:2":                                          (750.2)
  ‹Rigid(G) → □∀x(Numbers(x,G) → □Numbers(x,G))›
proof(rule "→I")
  AOT_assume ‹Rigid(G)›
  AOT_hence ‹□∀z([G]z → □[G]z)›
    using "df-rigid-rel:1"[THEN "≡dfE", THEN "&E"(2)] by blast
  AOT_hence ‹□□∀z([G]z → □[G]z)› by (metis "S5Basic:6" "≡E"(1))
  moreover AOT_have ‹□□∀z([G]z → □[G]z) → □∀x(Numbers(x,G) → □Numbers(x,G))›
  proof(rule RM; safe intro!: "→I" GEN)
    AOT_modally_strict {
      AOT_have act_den: ‹[λz 𝒜[F]z]↓› for F by "cqt:2[lambda]"
      fix x
```

```
      AOT_assume G_nec: ‹□∀z([G]z → □[G]z)›
      AOT_hence G_rigid: ‹Rigid(G)›
        using "df-rigid-rel:1"[THEN "≡dfI", OF "&I"] "cqt:2"
        by blast
      AOT_assume ‹Numbers(x, G)›
      AOT_hence ‹[A!]x & G↓ & ∀F (x[F] ≡ [λz 𝒜[F]z] ≈E G)›
        using numbers[THEN "≡dfE"] by blast
      AOT_hence Ax: ‹[A!]x› and ‹∀F (x[F] ≡ [λz 𝒜[F]z] ≈E G)›
        using "&E" by blast+
      AOT_hence ‹x[F] ≡ [λz 𝒜[F]z] ≈E G› for F
        using "∀E"(2) by blast
      moreover AOT_have ‹□([λz 𝒜[F]z] ≈E G → □[λz 𝒜[F]z] ≈E G)› for F
        using "approx-nec:3"[unvarify F, OF act_den, THEN "→E", OF "&I",
                             OF "actuallyF:2", OF G_rigid].
      moreover AOT_have ‹□(x[F] → □x[F])› for F
        by (simp add: RN "pre-en-eq:1[1]")
      ultimately AOT_have ‹□(x[F] ≡ [λz 𝒜[F]z] ≈E G)› for F
        using "sc-eq-box-box:5" "→E" "qml:2"[axiom_inst] "&I" by meson
      AOT_hence ‹∀F □(x[F] ≡ [λz 𝒜[F]z] ≈E G)›
        by (rule "∀I")
      AOT_hence 1: ‹□∀F (x[F] ≡ [λz 𝒜[F]z] ≈E G)›
        using BF[THEN "→E"] by fast
      AOT_have ‹□G↓›
        by (simp add: "ex:2:a")
      moreover AOT_have ‹□[A!]x›
        using Ax "oa-facts:2" "→E" by blast
      ultimately AOT_have ‹□(A!x & G↓)›
        by (metis "KBasic:3" "&I" "≡E"(2))
      AOT_hence ‹□(A!x & G↓ & ∀F (x[F] ≡ [λz 𝒜[F]z] ≈E G))›
        using 1 "KBasic:3" "&I" "≡E"(2) by fast
      AOT_thus ‹□Numbers(x, G)›
        by (AOT_subst_def numbers)
    }
  qed
  ultimately AOT_show ‹□∀x(Numbers(x,G) → □Numbers(x,G))›
    using "→E" by blast
qed
```

```
AOT_theorem "num-cont:3":
  ‹□∀x(Numbers(x, [λz 𝒜[G]z]) → □Numbers(x, [λz 𝒜[G]z]))›
  by (rule "num-cont:2"[unvarify G, THEN "→E"];
      ("cqt:2[lambda]" | rule "actuallyF:2"))
```
(750.3)

```
AOT_theorem "num-uniq": ‹ιx Numbers(x, G)↓›
  using "≡E"(2) "A-Exists:2" "RA[2]" "num:2" by blast
```
(751)

```
AOT_define num :: ‹τ ⇒ κs› (‹#_› [100] 100)
  "num-def:1": ‹#G =df ιx Numbers(x, G)›
```
(752.1)

```
AOT_theorem "num-def:2": ‹#G↓›
  using "num-def:1"[THEN "=dfI"(1)] "num-uniq" by simp
```
(752.2)

```
AOT_theorem "num-can:1":
  ‹#G = ιx(A!x & ∀F (x[F] ≡ [λz 𝒜[F]z] ≈E G))›
proof -
  AOT_have ‹□∀x(Numbers(x,G) ≡ [A!]x & ∀F (x[F] ≡ [λz 𝒜[F]z] ≈E G))›
    by (safe intro!: RN GEN "numbers[den]"[THEN "→E"] "cqt:2")
  AOT_hence ‹ιx Numbers(x, G) = ιx([A!]x & ∀F (x[F] ≡ [λz 𝒜[F]z] ≈E G))›
    using "num-uniq" "equiv-desc-eq:3"[THEN "→E", OF "&I"] by auto
  thus ?thesis
    by (rule "=dfI"(1)[OF "num-def:1", OF "num-uniq"])
qed
```
(753.1)

```
AOT_theorem "num-can:2": ‹#G = ιx(A!x & ∀F (x[F] ≡ F ≈E G))›                                          (753.2)
proof (rule id_trans[OF "num-can:1"]; rule "equiv-desc-eq:2"[THEN "→E"];
       safe intro!: "&I" "A-descriptions" GEN "Act-Basic:5"[THEN "≡E"(2)]
                    "logic-actual-nec:3"[axiom_inst, THEN "≡E"(2)])
  AOT_have act_den: ‹⊢□ [λz 𝒜[F]z]↓› for F
    by "cqt:2"
  AOT_have "eq-part:3[terms]": ‹⊢□ F ≈E G & F ≈E H → G ≈E H› for F G H                               (730.3)
    by (metis "&I" "eq-part:2" "eq-part:3" "→I" "&E" "→E")
  fix x
  {
    fix F
    AOT_have ‹𝒜(F ≈E [λz 𝒜[F]z])›
      by (simp add: "actuallyF:1")
    moreover AOT_have ‹𝒜((F ≈E [λz 𝒜[F]z]) → ([λz 𝒜[F]z] ≈E G ≡ F ≈E G))›
      by (auto intro!: "RA[2]" "→I" "≡I"
               simp: "eq-part:3"[unvarify G, OF act_den, THEN "→E", OF "&I"]
                     "eq-part:3[terms]"[unvarify G, OF act_den, THEN "→E", OF "&I"])
    ultimately AOT_have ‹𝒜([λz 𝒜[F]z] ≈E G ≡ F ≈E G)›
      using "logic-actual-nec:2"[axiom_inst, THEN "≡E"(1), THEN "→E"] by blast

    AOT_hence ‹𝒜[λz 𝒜[F]z] ≈E G ≡ 𝒜F ≈E G›
      by (metis "Act-Basic:5" "≡E"(1))
    AOT_hence 0: ‹(𝒜x[F] ≡ 𝒜[λz 𝒜[F]z] ≈E G) ≡ (𝒜x[F] ≡ 𝒜F ≈E G)›
      by (auto intro!: "≡I" "→I" elim: "≡E")
    AOT_have ‹𝒜(x[F] ≡ [λz 𝒜[F]z] ≈E G) ≡ (𝒜x[F] ≡ 𝒜[λz 𝒜[F]z] ≈E G)›
      by (simp add: "Act-Basic:5")
    also AOT_have ‹... ≡ (𝒜x[F] ≡ 𝒜F ≈E G)› using 0.
    also AOT_have ‹... ≡ 𝒜((x[F] ≡ F ≈E G))›
      by (meson "Act-Basic:5" "≡E"(6) "oth-class-taut:3:a")
    finally AOT_have 0: ‹𝒜(x[F] ≡ [λz 𝒜[F]z] ≈E G) ≡ 𝒜((x[F] ≡ F ≈E G))›.
  } note 0 = this
  AOT_have ‹𝒜∀F (x[F] ≡ [λz 𝒜[F]z] ≈E G) ≡ ∀F 𝒜(x[F] ≡ [λz 𝒜[F]z] ≈E G)›
    using "logic-actual-nec:3" "vdash-properties:1[2]" by blast
  also AOT_have ‹... ≡  ∀F 𝒜((x[F] ≡ F ≈E G))›
    apply (safe intro!: "≡I" "→I" GEN)
    using 0 "≡E"(1) "≡E"(2) "rule-ui:3" by blast+
  also AOT_have ‹... ≡ 𝒜(∀F (x[F] ≡ F ≈E G))›
    using "≡E"(6) "logic-actual-nec:3"[axiom_inst] "oth-class-taut:3:a" by fast
  finally AOT_have 0: ‹𝒜∀F (x[F] ≡ [λz 𝒜[F]z] ≈E G) ≡ 𝒜(∀F (x[F] ≡ F ≈E G))›.
  AOT_have ‹𝒜([A!]x & ∀F (x[F] ≡ [λz 𝒜[F]z] ≈E G)) ≡
            (𝒜A!x & 𝒜∀F (x[F] ≡ [λz 𝒜[F]z] ≈E G))›
    by (simp add: "Act-Basic:2")
  also AOT_have ‹... ≡ 𝒜[A!]x & 𝒜(∀F (x[F] ≡ F ≈E G))›
    using 0 "oth-class-taut:4:f" "→E" by blast
  also AOT_have ‹... ≡ 𝒜(A!x & ∀F (x[F] ≡ F ≈E G))›
    using "Act-Basic:2" "≡E"(6) "oth-class-taut:3:a" by blast
  finally AOT_show ‹𝒜([A!]x & ∀F (x[F] ≡ [λz 𝒜[F]z] ≈E G)) ≡
                    𝒜([A!]x & ∀F (x[F] ≡ F ≈E G))›.
qed

AOT_define NaturalCardinal :: ‹τ ⇒ φ› (‹NaturalCardinal'(_')›)
  card: ‹NaturalCardinal(x) ≡df ∃G(x = #G)›                                                              (755)

AOT_theorem "natcard-nec": ‹NaturalCardinal(x) → □NaturalCardinal(x)›                                    (756)
proof(rule "→I")
  AOT_assume ‹NaturalCardinal(x)›
  AOT_hence ‹∃G(x = #G)› using card[THEN "≡dfE"] by blast
  then AOT_obtain G where ‹x = #G› using "∃E"[rotated] by blast
  AOT_hence ‹□x = #G› by (metis "id-nec:2" "→E")
  AOT_hence ‹∃G □x = #G› by (rule "∃I")
  AOT_hence ‹□∃G x = #G› by (metis Buridan "→E")
  AOT_thus ‹□NaturalCardinal(x)›
    by (AOT_subst_def card)
```

```
qed

AOT_act_theorem "hume:1": ‹Numbers(#G, G)›                                    (757.1)
  apply (rule "=dfI"(1)[OF "num-def:1"])
  apply (simp add: "num-uniq")
  using "num-uniq" "vdash-properties:10" "y-in:3" by blast

AOT_act_theorem "hume:2": ‹#F = #G ≡ F ≈E G›                                  (757.2)
  by (safe intro!: "pre-Hume"[unvarify x y, OF "num-def:2",
                              OF "num-def:2", THEN "→E"] "&I" "hume:1")

AOT_act_theorem "hume:3": ‹#F = #G ≡ ∃R (R |: F 1-1⟶ontoE G)›                (757.3)
  using "equi-rem-thm"
  apply (AOT_subst (reverse) ‹R |: F 1-1⟶ontoE G›
                             ‹R |: F 1-1⟷E G› for: R :: ‹<κ×κ>›)
  using "equi:3" "hume:2" "≡E"(5) "≡Df" by blast

AOT_act_theorem "hume:4": ‹F ≡E G → #F = #G›                                  (757.4)
  by (metis "apE-eqE:1" "deduction-theorem" "hume:2" "≡E"(2) "→E")

AOT_theorem "hume-strict:1":                                                  (758.1)
  ‹∃x (Numbers(x, F) & Numbers(x, G)) ≡ F ≈E G›
proof(safe intro!: "≡I" "→I")
  AOT_assume ‹∃x (Numbers(x, F) & Numbers(x, G))›
  then AOT_obtain a where ‹Numbers(a, F) & Numbers(a, G)›
    using "∃E"[rotated] by blast
  AOT_thus ‹F ≈E G›
    using "num-tran:2" "→E" by blast
next
  AOT_assume 0: ‹F ≈E G›
  moreover AOT_obtain b where num_b_F: ‹Numbers(b, F)›
    by (metis "instantiation" "num:1")
  moreover AOT_have num_b_G: ‹Numbers(b, G)›
    using calculation "num-tran:1"[THEN "→E", THEN "≡E"(1)] by blast
  ultimately AOT_have ‹Numbers(b, F) & Numbers(b, G)›
    by (safe intro!: "&I")
  AOT_thus ‹∃x (Numbers(x, F) & Numbers(x, G))›
    by (rule "∃I")
qed

AOT_theorem "hume-strict:2":                                                  (758.2)
  ‹∃x∃y (Numbers(x, F) &
         ∀z(Numbers(z,F) → z = x) &
         Numbers(y, G) &
         ∀z (Numbers(z, G) → z = y) &
         x = y) ≡
   F ≈E G›
proof(safe intro!: "≡I" "→I")
  AOT_assume ‹∃x∃y (Numbers(x, F) & ∀z(Numbers(z,F) → z = x) &
                   Numbers(y, G) & ∀z (Numbers(z, G) → z = y) & x = y)›
  then AOT_obtain x where
    ‹∃y (Numbers(x, F) & ∀z(Numbers(z,F) → z = x) & Numbers(y, G) &
         ∀z (Numbers(z, G) → z = y) & x = y)›
    using "∃E"[rotated] by blast
  then AOT_obtain y where
    ‹Numbers(x, F) & ∀z(Numbers(z,F) → z = x) & Numbers(y, G) &
     ∀z (Numbers(z, G) → z = y) & x = y›
    using "∃E"[rotated] by blast
  AOT_hence ‹Numbers(x, F)› and ‹Numbers(y,G)› and ‹x = y›
    using "&E" by blast+
  AOT_hence ‹Numbers(y, F) & Numbers(y, G)›
    using "&I" "rule=E" by fast
  AOT_hence ‹∃y (Numbers(y, F) & Numbers(y, G))›
```

```
    by (rule "∃I")
  AOT_thus ‹F ≈E G›
    using "hume-strict:1"[THEN "≡E"(1)] by blast
next
  AOT_assume ‹F ≈E G›
  AOT_hence ‹∃x (Numbers(x, F) & Numbers(x, G))›
    using "hume-strict:1"[THEN "≡E"(2)] by blast
  then AOT_obtain x where ‹Numbers(x, F) & Numbers(x, G)›
    using "∃E"[rotated] by blast
  moreover AOT_have ‹∀z (Numbers(z, F) → z = x)›
              and ‹∀z (Numbers(z, G) → z = x)›
    using calculation
    by (auto intro!: GEN "→I" "pre-Hume"[THEN "→E", OF "&I", THEN "≡E"(2),
                                    rotated 2, OF "eq-part:1"] dest: "&E")
  ultimately AOT_have ‹Numbers(x, F) & ∀z(Numbers(z,F) → z = x) &
                      Numbers(x, G) & ∀z (Numbers(z, G) → z = x) & x = x›
    by (auto intro!: "&I" "id-eq:1" dest: "&E")
  AOT_thus ‹∃x∃y (Numbers(x, F) & ∀z(Numbers(z,F) → z = x) & Numbers(y, G) &
                 ∀z (Numbers(z, G) → z = y) & x = y)›
    by (auto intro!: "∃I")
qed

AOT_theorem unotEu: ‹¬∃y[λx O!x & x ≠E x]y›                                  (759)
proof(rule "raa-cor:2")
  AOT_assume ‹∃y[λx O!x & x ≠E x]y›
  then AOT_obtain y where ‹[λx O!x & x ≠E x]y›
    using "∃E"[rotated] by blast
  AOT_hence 0: ‹O!y & y ≠E y›
    by (rule "β→C"(1))
  AOT_hence ‹¬(y =E y)›
    using "&E"(2) "≡E"(1) "thm-neg=E" by blast
  moreover AOT_have ‹y =E y›
    by (metis 0[THEN "&E"(1)] "ord=Eequiv:1" "→E")
  ultimately AOT_show ‹p & ¬p› for p
    by (metis "raa-cor:3")
qed

AOT_define zero :: ‹κs› (‹0›)
  "zero:1": ‹0 =df #[λx O!x & x ≠E x]›                                      (760.1)

AOT_theorem "zero:2": ‹0↓›                                                  (760.2)
  by (rule "=dfI"(2)[OF "zero:1"]; rule "num-def:2"[unvarify G]; "cqt:2")

AOT_theorem "zero-card": ‹NaturalCardinal(0)›                               (761)
  apply (rule "=dfI"(2)[OF "zero:1"])
   apply (rule "num-def:2"[unvarify G]; "cqt:2")
  apply (rule card[THEN "≡dfI"])
  apply (rule "∃I"(1)[where τ=‹«[λx [O!]x & x ≠E x]»›])
   apply (rule "rule=I:1"; rule "num-def:2"[unvarify G]; "cqt:2")
  by "cqt:2"

AOT_theorem "eq-num:1":                                                     (762.1)
  ‹𝓐Numbers(x, G) ≡ Numbers(x,[λz 𝓐[G]z])›
proof -
  AOT_have act_den: ‹⊢□ [λz 𝓐[F]z]↓› for F by "cqt:2"
  AOT_have ‹□(∃x(Numbers(x, G) & Numbers(x,[λz 𝓐[G]z])) ≡ G ≈E [λz 𝓐[G]z])›
    using "hume-strict:1"[unvarify G, OF act_den, THEN RN].
  AOT_hence ‹𝓐(∃x(Numbers(x, G) & Numbers(x,[λz 𝓐[G]z])) ≡ G ≈E [λz 𝓐[G]z])›
    using "nec-imp-act"[THEN "→E"] by fast
  AOT_hence ‹𝓐(∃x(Numbers(x, G) & Numbers(x,[λz 𝓐[G]z])))›
    using "actuallyF:1" "Act-Basic:5" "≡E"(1) "≡E"(2) by fast
  AOT_hence ‹∃x 𝓐((Numbers(x, G) & Numbers(x,[λz 𝓐[G]z])))›
    by (metis "Act-Basic:10" "intro-elim:3:a")
```

```
  then AOT_obtain a where ‹𝒜(Numbers(a, G) & Numbers(a,[λz 𝒜[G]z]))›
    using "∃E"[rotated] by blast
  AOT_hence act_a_num_G: ‹𝒜Numbers(a, G)›
     and act_a_num_actG: ‹𝒜Numbers(a,[λz 𝒜[G]z])›
    using "Act-Basic:2" "&E" "≡E"(1) by blast+
  AOT_hence num_a_act_g: ‹Numbers(a, [λz 𝒜[G]z])›
    using "num-cont:2"[unvarify G, OF act_den, THEN "→E", OF "actuallyF:2",
                       THEN CBF[THEN "→E"], THEN "∀E"(2)]
    by (metis "≡E"(1) "sc-eq-fur:2" "vdash-properties:6")
  AOT_have 0: ‹⊢□ Numbers(x, G) & Numbers(y, G) → x = y› for y
    using "pre-Hume"[THEN "→E", THEN "≡E"(2), rotated, OF "eq-part:1"]
          "→I" by blast
  show ?thesis
  proof(safe intro!: "≡I" "→I")
    AOT_assume ‹𝒜Numbers(x, G)›
    AOT_hence ‹𝒜x = a›
      using 0[THEN "RA[2]", THEN "act-cond"[THEN "→E"], THEN "→E",
                OF "Act-Basic:2"[THEN "≡E"(2)], OF "&I"]
            act_a_num_G by blast
    AOT_hence ‹x = a› by (metis "id-act:1" "≡E"(2))
    AOT_hence ‹a = x› using id_sym by auto
    AOT_thus ‹Numbers(x, [λz 𝒜[G]z])›
      using "rule=E" num_a_act_g by fast
  next
    AOT_assume ‹Numbers(x, [λz 𝒜[G]z])›
    AOT_hence ‹a = x›
      using "pre-Hume"[unvarify G H, THEN "→E", OF act_den, OF act_den, OF "&I",
                       OF num_a_act_g, THEN "≡E"(2)]
            "eq-part:1"[unvarify F, OF act_den] by blast
    AOT_thus ‹𝒜Numbers(x, G)›
      using act_a_num_G "rule=E" by fast
  qed
qed

AOT_theorem "eq-num:2": ‹Numbers(x,[λz 𝒜[G]z]) ≡ x = #G›                    (762.2)
proof -
  AOT_have 0: ‹⊢□ x = ιx Numbers(x, G) ≡ ∀y (Numbers(y, [λz 𝒜[G]z]) ≡ y = x)› for x
    by (AOT_subst (reverse) ‹Numbers(x, [λz 𝒜[G]z])› ‹𝒜Numbers(x, G)› for: x)
       (auto simp: "eq-num:1" descriptions[axiom_inst])
  AOT_have ‹#G = ιx Numbers(x, G) ≡ ∀y (Numbers(y, [λz 𝒜[G]z]) ≡ y = #G)›
    using 0[unvarify x, OF "num-def:2"].
  moreover AOT_have ‹#G = ιx Numbers(x, G)›
    using "num-def:1" "num-uniq" "rule-id-df:1" by blast
  ultimately AOT_have ‹∀y (Numbers(y, [λz 𝒜[G]z]) ≡ y = #G)›
    using "≡E" by blast
  thus ?thesis using "∀E"(2) by blast
qed

AOT_theorem "eq-num:3": ‹Numbers(#G, [λy 𝒜[G]y])›                            (762.3)
proof -
  AOT_have ‹#G = #G›
    by (simp add: "rule=I:1" "num-def:2")
  thus ?thesis
    using "eq-num:2"[unvarify x, OF "num-def:2", THEN "≡E"(2)] by blast
qed

AOT_theorem "eq-num:4":                                                      (762.4)
  ‹A!#G & ∀F (#G[F] ≡ [λz 𝒜[F]z] ≈E [λz 𝒜[G]z])›
  by (auto intro!: "&I" "eq-num:3"[THEN numbers[THEN "≡dfE"],
                                   THEN "&E"(1), THEN "&E"(1)]
                   "eq-num:3"[THEN numbers[THEN "≡dfE"], THEN "&E"(2)])

AOT_theorem "eq-num:5": ‹#G[G]›                                              (762.5)
```

```
  by (auto intro!: "eq-num:4"[THEN "&E"(2), THEN "∀E"(2), THEN "≡E"(2)]
                   "eq-part:1"[unvarify F] simp: "cqt:2")

AOT_theorem "eq-num:6": ‹Numbers(x, G) → NaturalCardinal(x)›          (762.6)
proof(rule "→I")
  AOT_have act_den: ‹⊢□ [λz 𝒜[F]z]↓› for F
    by "cqt:2"
  AOT_obtain F where ‹Rigidifies(F, G)›
    by (metis "instantiation" "rigid-der:3")
  AOT_hence ϑ: ‹Rigid(F)› and ‹∀x([F]x ≡ [G]x)›
    using "df-rigid-rel:2"[THEN "≡dfE", THEN "&E"(2)]
          "df-rigid-rel:2"[THEN "≡dfE", THEN "&E"(1)]
    by blast+
  AOT_hence ‹F ≡E G›
    by (auto intro!: eqE[THEN "≡dfI"] "&I" "cqt:2" GEN "→I" elim: "∀E"(2))
  moreover AOT_assume ‹Numbers(x, G)›
  ultimately AOT_have ‹Numbers(x, F)›
    using "num-tran:3"[THEN "→E", THEN "≡E"(2)] by blast
  moreover AOT_have ‹F ≈E [λz 𝒜[F]z]›
    using ϑ "approx-nec:1" "→E" by blast
  ultimately AOT_have ‹Numbers(x, [λz 𝒜[F]z])›
    using "num-tran:1"[unvarify H, OF act_den, THEN "→E", THEN "≡E"(1)] by blast
  AOT_hence ‹x = #F›
    using "eq-num:2"[THEN "≡E"(1)] by blast
  AOT_hence ‹∃F x = #F›
    by (rule "∃I")
  AOT_thus ‹NaturalCardinal(x)›
    using card[THEN "≡dfI"] by blast
qed

AOT_theorem "eq-df-num": ‹∃G (x = #G) ≡ ∃G (Numbers(x,G))›          (763)
proof(safe intro!: "≡I" "→I")
  AOT_assume ‹∃G (x = #G)›
  then AOT_obtain P where ‹x = #P›
    using "∃E"[rotated] by blast
  AOT_hence ‹Numbers(x,[λz 𝒜[P]z])›
    using "eq-num:2"[THEN "≡E"(2)] by blast
  moreover AOT_have ‹[λz 𝒜[P]z]↓› by "cqt:2"
  ultimately AOT_show ‹∃G(Numbers(x,G))› by (rule "∃I")
next
  AOT_assume ‹∃G (Numbers(x,G))›
  then AOT_obtain Q where ‹Numbers(x,Q)›
    using "∃E"[rotated] by blast
  AOT_hence ‹NaturalCardinal(x)›
    using "eq-num:6"[THEN "→E"] by blast
  AOT_thus ‹∃G (x = #G)›
    using card[THEN "≡dfE"] by blast
qed

AOT_theorem "card-en": ‹NaturalCardinal(x) → ∀F(x[F] ≡ x = #F)›          (764)
proof(rule "→I"; rule GEN)
  AOT_have act_den: ‹⊢□ [λz 𝒜[F]z]↓› for F by "cqt:2"
  fix F
  AOT_assume ‹NaturalCardinal(x)›
  AOT_hence ‹∃F x = #F›
    using card[THEN "≡dfE"] by blast
  then AOT_obtain P where x_def: ‹x = #P›
    using "∃E"[rotated] by blast
  AOT_hence num_x_act_P: ‹Numbers(x,[λz 𝒜[P]z])›
    using "eq-num:2"[THEN "≡E"(2)] by blast
  AOT_have ‹#P[F] ≡ [λz 𝒜[F]z] ≈E [λz 𝒜[P]z]›
    using "eq-num:4"[THEN "&E"(2), THEN "∀E"(2)] by blast
  AOT_hence ‹x[F] ≡ [λz 𝒜[F]z] ≈E [λz 𝒜[P]z]›
```

```
    using x_def[symmetric] "rule=E" by fast
  also AOT_have ‹... ≡ Numbers(x, [λz 𝒜[F]z])›
    using "num-tran:1"[unvarify G H, OF act_den, OF act_den]
    using "num-tran:2"[unvarify G H, OF act_den, OF act_den]
    by (metis "&I" "deduction-theorem" "≡I" "≡E"(2) num_x_act_P)
  also AOT_have ‹... ≡ x = #F›
    using "eq-num:2" by blast
  finally AOT_show ‹x[F] ≡ x = #F›.
qed

AOT_theorem "OF:1": ‹¬∃u [F]u ≡ Numbers(0, F)›                    (765.1)
proof -
  AOT_have unotEu_act_ord: ‹¬∃v[λx O!x & 𝒜x ≠E x]v›
  proof(rule "raa-cor:2")
    AOT_assume ‹∃v[λx O!x & 𝒜x ≠E x]v›
    then AOT_obtain y where ‹[λx O!x & 𝒜x ≠E x]y›
      using "∃E"[rotated] "&E" by blast
    AOT_hence 0: ‹O!y & 𝒜y ≠E y›
      by (rule "β→C"(1))
    AOT_have ‹𝒜¬(y =E y)›
      apply (AOT_subst  ‹¬(y =E y)› ‹y ≠E y›)
       apply (meson "≡E"(2) "Commutativity of ≡" "thm-neg=E")
      by (fact 0[THEN "&E"(2)])
    AOT_hence ‹¬(y =E y)›
      by (metis "¬¬I" "Act-Sub:1" "id-act2:1" "≡E"(4))
    moreover AOT_have ‹y =E y›
      by (metis 0[THEN "&E"(1)] "ord=Eequiv:1" "→E")
    ultimately AOT_show ‹p & ¬p› for p
      by (metis "raa-cor:3")
  qed
  AOT_have ‹Numbers(0, [λy 𝒜[λx O!x & x ≠E x]y])›
    apply (rule "=dfI"(2)[OF "zero:1"])
     apply (rule "num-def:2"[unvarify G]; "cqt:2")
    apply (rule "eq-num:3"[unvarify G])
    by "cqt:2[lambda]"
  AOT_hence numbers0: ‹Numbers(0, [λx [O!]x & 𝒜x ≠E x])›
  proof (rule "num-tran:3"[unvarify x G H, THEN "→E", THEN "≡E"(1), rotated 4])
    AOT_show ‹[λy 𝒜[λx O!x & x ≠E x]y] ≡E [λx [O!]x & 𝒜x ≠E x]›
    proof (safe intro!: eqE[THEN "≡dfI"] "&I" Ordinary.GEN "→I" "cqt:2")
      fix u
      AOT_have ‹[λy 𝒜[λx O!x & x ≠E x]y]u ≡ 𝒜[λx O!x & x ≠E x]u›
        by (rule "beta-C-meta"[THEN "→E"]; "cqt:2[lambda]")
      also AOT_have ‹... ≡ 𝒜(O!u & u ≠E u)›
        apply (AOT_subst ‹[λx O!x & x ≠E x]u› ‹O!u & u ≠E u›)
         apply (rule "beta-C-meta"[THEN "→E"]; "cqt:2[lambda]")
        by (simp add: "oth-class-taut:3:a")
      also AOT_have ‹... ≡ (𝒜O!u & 𝒜u ≠E u)›
        by (simp add: "Act-Basic:2")
      also AOT_have ‹... ≡ (O!u & 𝒜u ≠E u)›
        by (metis Ordinary.ψ "&I" "&E"(2) "→I" "≡I" "≡E"(1) "oa-facts:7")
      also AOT_have ‹... ≡ [λx [O!]x & 𝒜x ≠E x]u›
        by (rule "beta-C-meta"[THEN "→E", symmetric]; "cqt:2[lambda]")
      finally AOT_show ‹[λy 𝒜[λx O!x & x ≠E x]y]u ≡ [λx [O!]x & 𝒜x ≠E x]u›.
    qed
  qed(fact "zero:2" | "cqt:2")+
  show ?thesis
  proof(safe intro!: "≡I" "→I")
    AOT_assume ‹¬∃u [F]u›
    moreover AOT_have ‹¬∃v [λx [O!]x & 𝒜x ≠E x]v›
      using unotEu_act_ord.
    ultimately AOT_have 0: ‹F ≈E [λx [O!]x & 𝒜x ≠E x]›
      by (rule "empty-approx:1"[unvarify H, THEN "→E", rotated, OF "&I"]) "cqt:2"
    AOT_thus ‹Numbers(0, F)›
```

```
      by (rule "num-tran:1"[unvarify x H, THEN "→E",
                            THEN "≡E"(2), rotated, rotated])
         (fact "zero:2" numbers0 | "cqt:2[lambda]")+
  next
    AOT_assume ‹Numbers(0, F)›
    AOT_hence 1: ‹F ≈E [λx [O!]x & 𝒜x ≠E x]›
      by (rule "num-tran:2"[unvarify x H, THEN "→E", rotated 2, OF "&I"])
         (fact numbers0 "zero:2" | "cqt:2[lambda]")+
    AOT_show ‹¬∃u [F]u›
    proof(rule "raa-cor:2")
      AOT_have 0: ‹[λx [O!]x & 𝒜x ≠E x]↓› by "cqt:2[lambda]"
      AOT_assume ‹∃u [F]u›
      AOT_hence ‹¬(F ≈E [λx [O!]x & 𝒜x ≠E x])›
        by (rule "empty-approx:2"[unvarify H, OF 0, THEN "→E", OF "&I"])
           (rule unotEu_act_ord)
      AOT_thus ‹F ≈E [λx [O!]x & 𝒜x ≠E x] & ¬(F ≈E [λx [O!]x & 𝒜x ≠E x])›
        using 1 "&I" by blast
    qed
  qed
qed

AOT_theorem "OF:2": ‹¬∃u 𝒜[F]u ≡ #F = 0›                              (765.2)
proof(rule "≡I"; rule "→I")
  AOT_assume 0: ‹¬∃u 𝒜[F]u›
  AOT_have ‹¬∃u [λz 𝒜[F]z]u›
  proof(rule "raa-cor:2")
    AOT_assume ‹∃u [λz 𝒜[F]z]u›
    then AOT_obtain u where ‹[λz 𝒜[F]z]u›
      using "Ordinary.∃E"[rotated] by blast
    AOT_hence ‹𝒜[F]u›
      by (metis "betaC:1:a")
    AOT_hence ‹∃u 𝒜[F]u›
      by (rule "Ordinary.∃I")
    AOT_thus ‹∃u 𝒜[F]u & ¬∃u 𝒜[F]u›
      using 0 "&I" by blast
  qed
  AOT_hence ‹Numbers(0,[λz 𝒜[F]z])›
    by (safe intro!: "OF:1"[unvarify F, THEN "≡E"(1)]) "cqt:2"
  AOT_hence ‹0 = #F›
    by (rule "eq-num:2"[unvarify x, OF "zero:2", THEN "≡E"(1)])
  AOT_thus ‹#F = 0› using id_sym by blast
next
  AOT_assume ‹#F = 0›
  AOT_hence ‹0 = #F› using id_sym by blast
  AOT_hence ‹Numbers(0,[λz 𝒜[F]z])›
    by (rule "eq-num:2"[unvarify x, OF "zero:2", THEN "≡E"(2)])
  AOT_hence 0: ‹¬∃u [λz 𝒜[F]z]u›
    by (safe intro!: "OF:1"[unvarify F, THEN "≡E"(2)]) "cqt:2"
  AOT_show ‹¬∃u 𝒜[F]u›
  proof(rule "raa-cor:2")
    AOT_assume ‹∃u 𝒜[F]u›
    then AOT_obtain u where ‹𝒜[F]u›
      using "Ordinary.∃E"[rotated] by meson
    AOT_hence ‹[λz 𝒜[F]z]u›
      by (auto intro!: "β←C" "cqt:2")
    AOT_hence ‹∃u [λz 𝒜[F]z]u›
      using "Ordinary.∃I" by blast
    AOT_thus ‹∃u [λz 𝒜[F]z]u & ¬∃u [λz 𝒜[F]z]u›
      using "&I" 0 by blast
  qed
qed

AOT_theorem "OF:3": ‹□¬∃u [F]u → #F = 0›                               (765.3)
```

```
proof(rule "→I")
  AOT_assume ‹□¬∃u [F]u›
  AOT_hence 0: ‹¬◇∃u [F]u›
    using "KBasic2:1" "≡E"(1) by blast
  AOT_have ‹¬∃u [λz 𝒜[F]z]u›
  proof(rule "raa-cor:2")
    AOT_assume ‹∃u [λz 𝒜[F]z]u›
    then AOT_obtain u where ‹[λz 𝒜[F]z]u›
      using "Ordinary.∃E"[rotated] by blast
    AOT_hence ‹𝒜[F]u›
      by (metis "betaC:1:a")
    AOT_hence ‹◇[F]u›
      by (metis "Act-Sub:3" "→E")
    AOT_hence ‹∃u ◇[F]u›
      by (rule "Ordinary.∃I")
    AOT_hence ‹◇∃u [F]u›
      using "Ordinary.res-var-bound-reas[CBF◇]"[THEN "→E"] by blast
    AOT_thus ‹◇∃u [F]u & ¬◇∃u [F]u›
      using 0 "&I" by blast
  qed
  AOT_hence ‹Numbers(0,[λz 𝒜[F]z])›
    by (safe intro!: "OF:1"[unvarify F, THEN "≡E"(1)]) "cqt:2"
  AOT_hence ‹0 = #F›
    by (rule "eq-num:2"[unvarify x, OF "zero:2", THEN "≡E"(1)])
  AOT_thus ‹#F = 0› using id_sym by blast
qed

AOT_theorem "OF:4": ‹w ⊨ ¬∃u [F]u ≡ #[F]_w = 0›                    (765.4)
proof (rule "rule-id-df:2:b"[OF "w-index", where τ_1τ_n="(_,_)", simplified])
  AOT_show ‹[λx_1...x_n w ⊨ [F]x_1...x_n]↓›
    by (simp add: "w-rel:3")
next
  AOT_show ‹w ⊨ ¬∃u [F]u ≡ #[λx w ⊨ [F]x] = 0›
  proof (rule "≡I"; rule "→I")
    AOT_assume ‹w ⊨ ¬∃u [F]u›
    AOT_hence 0: ‹¬w ⊨ ∃u [F]u›
      using "coherent:1"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(1)] by blast
    AOT_have ‹¬∃u 𝒜[λx w ⊨ [F]x]u›
    proof(rule "raa-cor:2")
      AOT_assume ‹∃u 𝒜[λx w ⊨ [F]x]u›
      then AOT_obtain u where ‹𝒜[λx w ⊨ [F]x]u›
        using "Ordinary.∃E"[rotated] by meson
      AOT_hence ‹𝒜w ⊨ [F]u›
        by (AOT_subst (reverse) ‹w ⊨ [F]u› ‹[λx w ⊨ [F]x]u›;
            safe intro!: "beta-C-meta"[THEN "→E"] "w-rel:1"[THEN "→E"])
           "cqt:2"
      AOT_hence 1: ‹w ⊨ [F]u›
        using "rigid-truth-at:4"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(1)]
        by blast
      AOT_have ‹□([F]u → ∃u [F]u)›
        using "Ordinary.∃I" "→I" RN by simp
      AOT_hence ‹w ⊨ ([F]u → ∃u [F]u)›
        using "fund:2"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(1)]
              "PossibleWorld.∀E" by fast
      AOT_hence ‹w ⊨ ∃u [F]u›
        using 1 "conj-dist-w:2"[unvarify p q, OF "log-prop-prop:2",
                                OF "log-prop-prop:2", THEN "≡E"(1),
                                THEN "→E"] by blast
      AOT_thus ‹w ⊨ ∃u [F]u & ¬w ⊨ ∃u [F]u›
        using 0 "&I" by blast
    qed
    AOT_thus ‹#[λx w ⊨ [F]x] = 0›
      by (safe intro!: "OF:2"[unvarify F, THEN "≡E"(1)] "w-rel:1"[THEN "→E"])
```

```
        "cqt:2"
  next
    AOT_assume ‹#[λx w ⊨ [F]x] = 0›
    AOT_hence 0: ‹¬∃u 𝒜[λx w ⊨ [F]x]u›
      by (safe intro!: "0F:2"[unvarify F, THEN "≡E"(2)] "w-rel:1"[THEN "→E"])
        "cqt:2"
    AOT_have ‹¬w ⊨ ∃u [F]u›
    proof (rule "raa-cor:2")
      AOT_assume ‹w ⊨ ∃u [F]u›
      AOT_hence ‹∃x w ⊨ (O!x & [F]x)›
        using "conj-dist-w:6"[THEN "≡E"(1)] by fast
      then AOT_obtain x where ‹w ⊨ (O!x & [F]x)›
        using "∃E"[rotated] by blast
      AOT_hence ‹w ⊨ O!x› and Fx_in_w: ‹w ⊨ [F]x›
        using "conj-dist-w:1"[unvarify p q] "≡E"(1) "log-prop-prop:2"
              "&E" by blast+
      AOT_hence ‹◊O!x›
        using "fund:1"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(2)]
              "PossibleWorld.∃I" by simp
      AOT_hence ord_x: ‹O!x›
        using "oa-facts:3"[THEN "→E"] by blast
      AOT_have ‹𝒜w ⊨ [F]x›
        using "rigid-truth-at:4"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(2)]
              Fx_in_w by blast
      AOT_hence ‹𝒜[λx w ⊨ [F]x]x›
        by (AOT_subst ‹[λx w ⊨ [F]x]x› ‹w ⊨ [F]x›;
            safe intro!: "beta-C-meta"[THEN "→E"] "w-rel:1"[THEN "→E"]) "cqt:2"
      AOT_hence ‹O!x & 𝒜[λx w ⊨ [F]x]x›
        using ord_x "&I" by blast
      AOT_hence ‹∃x (O!x & 𝒜[λx w ⊨ [F]x]x)›
        using "∃I" by fast
      AOT_thus ‹∃u (𝒜[λx w ⊨ [F]x]u) & ¬∃u 𝒜[λx w ⊨ [F]x]u›
        using 0 "&I" by blast
    qed
    AOT_thus ‹w ⊨ ¬∃u[F]u›
      using "coherent:1"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(2)] by blast
  qed
qed

AOT_act_theorem "zero=:1":                                                    (766.1)
  ‹NaturalCardinal(x) → ∀F (x[F] ≡ Numbers(x, F))›
proof(safe intro!: "→I" GEN)
  fix F
  AOT_assume ‹NaturalCardinal(x)›
  AOT_hence ‹∀F (x[F] ≡ x = #F)›
    by (metis "card-en" "→E")
  AOT_hence 1: ‹x[F] ≡ x = #F›
    using "∀E"(2) by blast
  AOT_have 2: ‹x[F] ≡ x = ιy(Numbers(y, F))›
    by (rule "num-def:1"[THEN "=dfE"(1)])
       (auto simp: 1 "num-uniq")
  AOT_have ‹x = ιy(Numbers(y, F)) → Numbers(x, F)›
    using "y-in:1" by blast
  moreover AOT_have ‹Numbers(x, F) → x = ιy(Numbers(y, F))›
  proof(rule "→I")
    AOT_assume 1: ‹Numbers(x, F)›
    moreover AOT_obtain z where z_prop: ‹∀y (Numbers(y, F) → y = z)›
      using "num:2"[THEN "uniqueness:1"[THEN "≡dfE"]] "∃E"[rotated] "&E" by blast
    ultimately AOT_have ‹x = z›
      using "∀E"(2) "→E" by blast
    AOT_hence ‹∀y (Numbers(y, F) → y = x)›
      using z_prop "rule=E" id_sym by fast
    AOT_thus ‹x = ιy(Numbers(y,F))›
```

```
      by (rule hintikka[THEN "≡E"(2), OF "&I", rotated])
         (fact 1)
  qed
  ultimately AOT_have ‹x = ιy(Numbers(y, F)) ≡ Numbers(x, F)›
    by (metis "≡I")
  AOT_thus ‹x[F] ≡ Numbers(x, F)›
    using 2 by (metis "≡E"(5))
qed

AOT_act_theorem "zero=:2": ‹0[F] ≡ ¬∃u[F]u›                          (766.2)
proof -
  AOT_have ‹0[F] ≡ Numbers(0, F)›
    using "zero=:1"[unvarify x, OF "zero:2", THEN "→E",
                    OF "zero-card", THEN "∀E"(2)].
  also AOT_have ‹... ≡ ¬∃u[F]u›
    using "OF:1"[symmetric].
  finally show ?thesis.
qed

AOT_act_theorem "zero=:3": ‹¬∃u[F]u ≡ #F = 0›                        (766.3)
proof -
  AOT_have ‹¬∃u[F]u ≡ 0[F]› using "zero=:2"[symmetric].
  also AOT_have ‹... ≡ 0 = #F›
    using "card-en"[unvarify x, OF "zero:2", THEN "→E",
                    OF "zero-card", THEN "∀E"(2)].
  also AOT_have ‹... ≡ #F = 0›
    by (simp add: "deduction-theorem" id_sym "≡I")
  finally show ?thesis.
qed

AOT_define Hereditary :: ‹τ ⇒ τ ⇒ φ› (‹Hereditary'(_,_')›)
  "hered:1":                                                          (767.1)
  ‹Hereditary(F, R) ≡df R↓ & F↓ & ∀x∀y([R]xy → ([F]x → [F]y))›

AOT_theorem "hered:2":                                                (767.2)
  ‹[λxy ∀F((∀z([R]xz → [F]z) & Hereditary(F,R)) → [F]y)]↓›
  by "cqt:2[lambda]"

AOT_define StrongAncestral :: ‹τ ⇒ Π› (‹_*›)
  "ances-df":                                                         (768)
  ‹R* =df [λxy ∀F((∀z([R]xz → [F]z) & Hereditary(F,R)) → [F]y)]›

AOT_theorem "ances":                                                  (769)
  ‹[R*]xy ≡ ∀F((∀z([R]xz → [F]z) & Hereditary(F,R)) → [F]y)›
  apply (rule "=dfI"(1)[OF "ances-df"])
   apply "cqt:2[lambda]"
  apply (rule "beta-C-meta"[THEN "→E", OF "hered:2", unvarify ν1νn,
                            where τ=‹(_,_)›, simplified])
  by (simp add: "&I" "ex:1:a" prod_denotesI "rule-ui:3")

AOT_theorem "anc-her:1":                                              (770.1)
  ‹[R]xy → [R*]xy›
proof (safe intro!: "→I" ances[THEN "≡E"(2)] GEN)
  fix F
  AOT_assume ‹∀z ([R]xz → [F]z) & Hereditary(F, R)›
  AOT_hence ‹[R]xy → [F]y›
    using "∀E"(2) "&E" by blast
  moreover AOT_assume ‹[R]xy›
  ultimately AOT_show ‹[F]y›
    using "→E" by blast
qed

AOT_theorem "anc-her:2":                                              (770.2)
```

```
  ‹([R*]xy & ∀z([R]xz → [F]z) & Hereditary(F,R)) → [F]y›
proof(rule "→I"; (frule "&E"(1); drule "&E"(2))+)
  AOT_assume ‹[R*]xy›
  AOT_hence ‹(∀z([R]xz → [F]z) & Hereditary(F,R)) → [F]y›
    using ances[THEN "≡E"(1)] "∀E"(2) by blast
  moreover AOT_assume ‹∀z([R]xz → [F]z)›
  moreover AOT_assume ‹Hereditary(F,R)›
  ultimately AOT_show ‹[F]y›
    using "→E" "&I" by blast
qed

AOT_theorem "anc-her:3":                                        (770.3)
  ‹([F]x & [R*]xy & Hereditary(F, R)) → [F]y›
proof(rule "→I"; (frule "&E"(1); drule "&E"(2))+)
  AOT_assume 1: ‹[F]x›
  AOT_assume 2: ‹Hereditary(F, R)›
  AOT_hence 3: ‹∀x ∀y ([R]xy → ([F]x → [F]y))›
    using "hered:1"[THEN "≡dfE"] "&E" by blast
  AOT_have ‹∀z ([R]xz → [F]z)›
  proof (rule GEN; rule "→I")
    fix z
    AOT_assume ‹[R]xz›
    moreover AOT_have ‹[R]xz → ([F]x → [F]z)›
      using 3 "∀E"(2) by blast
    ultimately AOT_show ‹[F]z›
      using 1 "→E" by blast
  qed
  moreover AOT_assume ‹[R*]xy›
  ultimately AOT_show ‹[F]y›
    by (auto intro!: 2 "anc-her:2"[THEN "→E"] "&I")
qed

AOT_theorem "anc-her:4": ‹([R]xy & [R*]yz) → [R*]xz›            (770.4)
proof(rule "→I"; frule "&E"(1); drule "&E"(2))
  AOT_assume 0: ‹[R*]yz› and 1: ‹[R]xy›
  AOT_show ‹[R*]xz›
  proof(safe intro!: ances[THEN "≡E"(2)] GEN "&I" "→I";
                     frule "&E"(1); drule "&E"(2))
    fix F
    AOT_assume ‹∀z ([R]xz → [F]z)›
    AOT_hence 1: ‹[F]y›
      using 1 "∀E"(2) "→E" by blast
    AOT_assume 2: ‹Hereditary(F,R)›
    AOT_show ‹[F]z›
      by (rule "anc-her:3"[THEN "→E"]; auto intro!: "&I" 1 2 0)
  qed
qed

AOT_theorem "anc-her:5": ‹[R*]xy → ∃z [R]zy›                    (770.5)
proof (rule "→I")
  AOT_have 0: ‹[λy ∃x [R]xy]↓› by "cqt:2"
  AOT_assume 1: ‹[R*]xy›
  AOT_have ‹[λy∃x [R]xy]y›
  proof(rule "anc-her:2"[unvarify F, OF 0, THEN "→E"];
        safe intro!: "&I" GEN "→I" "hered:1"[THEN "≡dfI"] "cqt:2" 0)
    AOT_show ‹[R*]xy› using 1.
  next
    fix z
    AOT_assume ‹[R]xz›
    AOT_hence ‹∃x [R]xz› by (rule "∃I")
    AOT_thus ‹[λy∃x [R]xy]z›
      by (auto intro!: "β←C"(1) "cqt:2")
  next
```

```
    fix x y
    AOT_assume ‹[R]xy›
    AOT_hence ‹∃x [R]xy› by (rule "∃I")
    AOT_thus ‹[λy ∃x [R]xy]y›
      by (auto intro!: "β←C"(1) "cqt:2")
  qed
  AOT_thus ‹∃z [R]zy›
    by (rule "β→C"(1))
qed

AOT_theorem "anc-her:6": ‹([R*]xy & [R*]yz) → [R*]xz›     (770.6)
proof (rule "→I"; frule "&E"(1); drule "&E"(2))
  AOT_assume ‹[R*]xy›
  AOT_hence ϑ: ‹∀z ([R]xz → [F]z) & Hereditary(F,R) → [F]y› for F
    using "∀E"(2)  ances[THEN "≡E"(1)] by blast
  AOT_assume ‹[R*]yz›
  AOT_hence ξ: ‹∀z ([R]yz → [F]z) & Hereditary(F,R) → [F]z› for F
    using "∀E"(2) ances[THEN "≡E"(1)] by blast
  AOT_show ‹[R*]xz›
  proof (rule ances[THEN "≡E"(2)]; safe intro!: GEN "→I")
    fix F
    AOT_assume ζ: ‹∀z ([R]xz → [F]z) & Hereditary(F,R)›
    AOT_show ‹[F]z›
    proof (rule ξ[THEN "→E", OF "&I"])
      AOT_show ‹Hereditary(F,R)›
        using ζ[THEN "&E"(2)].
    next
      AOT_show ‹∀z ([R]yz → [F]z)›
      proof(rule GEN; rule "→I")
        fix z
        AOT_assume ‹[R]yz›
        moreover AOT_have ‹[F]y›
          using ϑ[THEN "→E", OF ζ].
        ultimately AOT_show ‹[F]z›
          using ζ[THEN "&E"(2), THEN "hered:1"[THEN "≡dfE"],
                  THEN "&E"(2), THEN "∀E"(2), THEN "∀E"(2),
                  THEN "→E", THEN "→E"]
          by blast
      qed
    qed
  qed
qed

AOT_define OneToOne :: ‹τ ⇒ φ› (‹1-1'(_')›)
  "df-1-1:1": ‹1-1(R) ≡df R↓ & ∀x∀y∀z([R]xz & [R]yz → x = y)›     (772.1)

AOT_define RigidOneToOne :: ‹τ ⇒ φ› (‹Rigid1-1'(_')›)
  "df-1-1:2": ‹Rigid1-1(R) ≡df 1-1(R) & Rigid(R)›     (772.2)

AOT_theorem "df-1-1:3": ‹Rigid1-1(R) → □1-1(R)›     (772.3)
proof(rule "→I")
  AOT_assume ‹Rigid1-1(R)›
  AOT_hence ‹1-1(R)› and RigidR: ‹Rigid(R)›
    using "df-1-1:2"[THEN "≡dfE"] "&E" by blast+
  AOT_hence 1: ‹[R]xz & [R]yz → x = y› for x y z
    using "df-1-1:1"[THEN "≡dfE"] "&E"(2) "∀E"(2) by blast
  AOT_have 1: ‹[R]xz & [R]yz → □x = y› for x y z
    by (AOT_subst (reverse) ‹□x = y›  ‹x = y›)
       (auto simp: 1 "id-nec:2" "≡I" "qml:2"[axiom_inst])
  AOT_have ‹□∀x1...∀xn ([R]x1...xn → □[R]x1...xn)›
    using "df-rigid-rel:1"[THEN "≡dfE", OF RigidR] "&E" by blast
  AOT_hence ‹∀x1...∀xn □([R]x1...xn → □[R]x1...xn)›
    using "CBF"[THEN "→E"] by fast
```

```
  AOT_hence ‹∀x₁∀x₂ □([R]x₁x₂ → □[R]x₁x₂)›
    using tuple_forall[THEN "≡dfE"] by blast
  AOT_hence ‹□([R]xy → □[R]xy)› for x y
    using "∀E"(2) by blast
  AOT_hence ‹□(([R]xz → □[R]xz) & ([R]yz → □[R]yz))› for x y z
    by (metis "KBasic:3" "&I" "≡E"(3) "raa-cor:3")
  moreover AOT_have ‹□(([R]xz → □[R]xz) & ([R]yz → □[R]yz)) →
                      □(([R]xz & [R]yz) → □([R]xz & [R]yz))› for x y z
    by (rule RM) (metis "→I" "KBasic:3" "&I" "&E"(1) "&E"(2) "≡E"(2) "→E")
  ultimately AOT_have 2: ‹□(([R]xz & [R]yz) → □([R]xz & [R]yz))› for x y z
    using "→E" by blast
  AOT_hence 3: ‹□([R]xz & [R]yz → x = y)› for x y z
    using "sc-eq-box-box:6"[THEN "→E", THEN "→E", OF 2, OF 1] by blast
  AOT_hence 4: ‹□∀x∀y∀z([R]xz & [R]yz → x = y)›
    by (safe intro!: GEN BF[THEN "→E"] 3)
  AOT_thus ‹□1-1(R)›
    by (AOT_subst_thm "df-1-1:1"[THEN "≡Df", THEN "≡S"(1),
                              OF "cqt:2[const_var]"[axiom_inst]])
qed

AOT_theorem "df-1-1:4": ‹∀R(Rigid₁₋₁(R) → □Rigid₁₋₁(R))›                (772.4)
proof(rule GEN;rule "→I")
AOT_modally_strict {
  fix R
      AOT_assume 0: ‹Rigid₁₋₁(R)›
      AOT_hence 1: ‹R↓›
        by (meson "≡dfE" "&E"(1) "df-1-1:1" "df-1-1:2")
      AOT_hence 2: ‹□R↓›
        using "exist-nec" "→E" by blast
      AOT_have 4: ‹□1-1(R)›
        using "df-1-1:3"[unvarify R, OF 1, THEN "→E", OF 0].
      AOT_have ‹Rigid(R)›
        using 0 "≡dfE"[OF "df-1-1:2"] "&E" by blast
      AOT_hence ‹□∀x₁...∀xₙ ([R]x₁...xₙ → □[R]x₁...xₙ)›
        using  "df-rigid-rel:1"[THEN "≡dfE"] "&E" by blast
      AOT_hence ‹□□∀x₁...∀xₙ ([R]x₁...xₙ → □[R]x₁...xₙ)›
        by (metis "S5Basic:6" "≡E"(1))
      AOT_hence ‹□Rigid(R)›
        apply (AOT_subst_def "df-rigid-rel:1")
        using 2 "KBasic:3" "≡S"(2) "≡E"(2) by blast
      AOT_thus ‹□Rigid₁₋₁(R)›
        apply (AOT_subst_def "df-1-1:2")
        using 4 "KBasic:3" "≡S"(2) "≡E"(2) by blast
}
qed

AOT_define InDomainOf :: ‹τ ⇒ τ ⇒ φ› (‹InDomainOf'(_,_')›)
  "df-1-1:5": ‹InDomainOf(x, R) ≡df ∃y [R]xy›                              (772.5)

AOT_register_rigid_restricted_type
  RigidOneToOneRelation: ‹Rigid₁₋₁(Π)›
proof
  AOT_modally_strict {
    AOT_show ‹∃α Rigid₁₋₁(α)›
    proof (rule "∃I"(1)[where τ=‹«(=E)»›])
      AOT_show ‹Rigid₁₋₁((=E))›
      proof (safe intro!: "df-1-1:2"[THEN "≡dfI"] "&I" "df-1-1:1"[THEN "≡dfI"]
                          GEN "→I" "df-rigid-rel:1"[THEN "≡dfI"] "=E[denotes]")
        fix x y z
        AOT_assume ‹x =E z & y =E z›
        AOT_thus ‹x = y›
          by (metis "rule=E" "&E"(1) "Conjunction Simplification"(2)
                    "=E-simple:2" id_sym "→E")
```

```
      next
        AOT_have ‹∀x∀y □(x =E y → □x =E y)›
        proof(rule GEN; rule GEN)
          AOT_show ‹□(x =E y → □x =E y)› for x y
            by (meson RN "deduction-theorem" "id-nec3:1" "≡E"(1))
        qed
        AOT_hence ‹∀x1...∀xn □([(=E)]x1...xn → □[(=E)]x1...xn)›
          by (rule tuple_forall[THEN "≡dfI"])
        AOT_thus ‹□∀x1...∀xn ([(=E)]x1...xn → □[(=E)]x1...xn)›
          using BF[THEN "→E"] by fast
      qed
    qed(fact "=E[denotes]")
  }
next
  AOT_modally_strict {
    AOT_show ‹Rigid1-1(Π) → Π↓› for Π
    proof(rule "→I")
      AOT_assume ‹Rigid1-1(Π)›
      AOT_hence ‹1-1(Π)›
        using "df-1-1:2"[THEN "≡dfE"] "&E" by blast
      AOT_thus ‹Π↓›
        using "df-1-1:1"[THEN "≡dfE"] "&E" by blast
    qed
  }
next
  AOT_modally_strict {
    AOT_show ‹∀F(Rigid1-1(F) → □Rigid1-1(F))›
      by (safe intro!: GEN "df-1-1:4"[THEN "∀E"(2)])
  }
qed
AOT_register_variable_names
  RigidOneToOneRelation: ℛ 𝒮

AOT_define IdentityRestrictedToDomain :: ‹τ ⇒ Π› (‹'(=_')›)
  "id-d-R": ‹(=ℛ) =df [λxy ∃z ([ℛ]xz & [ℛ]yz)]›                      (773)

syntax "_AOT_id_d_R_infix" :: ‹τ ⇒ τ ⇒ τ ⇒ φ› ("(_ =_/ _)" [50, 51, 51] 50)
translations
  "_AOT_id_d_R_infix κ Π κ'" ==
  "CONST AOT_exe (CONST IdentityRestrictedToDomain Π) (κ,κ')"

AOT_theorem "id-R-thm:1": ‹x =ℛ y ≡ ∃z ([ℛ]xz & [ℛ]yz)›               (774.1)
proof -
  AOT_have 0: ‹[λxy ∃z ([ℛ]xz & [ℛ]yz)]↓› by "cqt:2"
  show ?thesis
    apply (rule "=dfI"(1)[OF "id-d-R"])
    apply (fact 0)
    apply (rule "beta-C-meta"[THEN "→E", OF 0, unvarify ν1νn,
                              where τ=‹(_,_)›, simplified])
    by (simp add: "&I" "ex:1:a" prod_denotesI "rule-ui:3")
qed

AOT_theorem "id-R-thm:2":                                             (774.2)
  ‹x =ℛ y → (InDomainOf(x, ℛ) & InDomainOf(y, ℛ))›
proof(rule "→I")
  AOT_assume ‹x =ℛ y›
  AOT_hence ‹∃z ([ℛ]xz & [ℛ]yz)›
    using "id-R-thm:1"[THEN "≡E"(1)] by simp
  then AOT_obtain z where z_prop: ‹[ℛ]xz & [ℛ]yz›
    using "∃E"[rotated] by blast
  AOT_show ‹InDomainOf(x, ℛ) & InDomainOf(y, ℛ)›
  proof (safe intro!: "&I" "df-1-1:5"[THEN "≡dfI"])
    AOT_show ‹∃y [ℛ]xy›
```

```
      using z_prop[THEN "&E"(1)] "∃I" by fast
  next
    AOT_show ‹∃z [ℛ]yz›
      using z_prop[THEN "&E"(2)] "∃I" by fast
  qed
qed

AOT_theorem "id-R-thm:3": ‹x =ℛ y → x = y›                                   (774.3)
proof(rule "→I")
  AOT_assume ‹x =ℛ y›
  AOT_hence ‹∃z ([ℛ]xz & [ℛ]yz)›
    using "id-R-thm:1"[THEN "≡E"(1)] by simp
  then AOT_obtain z where z_prop: ‹[ℛ]xz & [ℛ]yz›
    using "∃E"[rotated] by blast
  AOT_thus ‹x = y›
    using "df-1-1:3"[THEN "→E", OF RigidOneToOneRelation.ψ,
                     THEN "qml:2"[axiom_inst, THEN "→E"],
                     THEN "≡dfE"[OF "df-1-1:1"], THEN "&E"(2),
                     THEN "∀E"(2), THEN "∀E"(2),
                     THEN "∀E"(2), THEN "→E"]
     by blast
qed

AOT_theorem "id-R-thm:4":                                                    (774.4)
  ‹(InDomainOf(x, ℛ) ∨ InDomainOf(y, ℛ)) → (x =ℛ y ≡ x = y)›
proof (rule "→I")
  AOT_assume ‹InDomainOf(x, ℛ) ∨ InDomainOf(y, ℛ)›
  moreover {
    AOT_assume ‹InDomainOf(x, ℛ)›
    AOT_hence ‹∃z [ℛ]xz›
      by (metis "≡dfE" "df-1-1:5")
    then AOT_obtain z where z_prop: ‹[ℛ]xz›
      using "∃E"[rotated] by blast
    AOT_have ‹x =ℛ y ≡ x = y›
    proof(safe intro!: "≡I" "→I" "id-R-thm:3"[THEN "→E"])
      AOT_assume ‹x = y›
      AOT_hence ‹[ℛ]yz›
        using z_prop "rule=E" by fast
      AOT_hence ‹[ℛ]xz & [ℛ]yz›
        using z_prop "&I" by blast
      AOT_hence ‹∃z ([ℛ]xz & [ℛ]yz)›
        by (rule "∃I")
      AOT_thus ‹x =ℛ y›
        using "id-R-thm:1" "≡E"(2) by blast
    qed
  }
  moreover {
    AOT_assume ‹InDomainOf(y, ℛ)›
    AOT_hence ‹∃z [ℛ]yz›
      by (metis "≡dfE" "df-1-1:5")
    then AOT_obtain z where z_prop: ‹[ℛ]yz›
      using "∃E"[rotated] by blast
    AOT_have ‹x =ℛ y ≡ x = y›
    proof(safe intro!: "≡I" "→I" "id-R-thm:3"[THEN "→E"])
      AOT_assume ‹x = y›
      AOT_hence ‹[ℛ]xz›
        using z_prop "rule=E" id_sym by fast
      AOT_hence ‹[ℛ]xz & [ℛ]yz›
        using z_prop "&I" by blast
      AOT_hence ‹∃z ([ℛ]xz & [ℛ]yz)›
        by (rule "∃I")
      AOT_thus ‹x =ℛ y›
        using "id-R-thm:1" "≡E"(2) by blast
```

```
    qed
  }
  ultimately AOT_show ‹x =ℛ y ≡ x = y›
    by (metis "∨E"(2) "raa-cor:1")
qed

AOT_theorem "id-R-thm:5": ‹InDomainOf(x, ℛ) → x =ℛ x›                    (774.5)
proof (rule "→I")
  AOT_assume ‹InDomainOf(x, ℛ)›
  AOT_hence ‹∃z [ℛ]xz›
    by (metis "≡dfE" "df-1-1:5")
  then AOT_obtain z where z_prop: ‹[ℛ]xz›
    using "∃E"[rotated] by blast
  AOT_hence ‹[ℛ]xz & [ℛ]xz›
    using "&I" by blast
  AOT_hence ‹∃z ([ℛ]xz & [ℛ]xz)›
    using "∃I" by fast
  AOT_thus ‹x =ℛ x›
    using "id-R-thm:1" "≡E"(2) by blast
qed

AOT_theorem "id-R-thm:6": ‹x =ℛ y → y =ℛ x›                               (774.6)
proof(rule "→I")
  AOT_assume 0: ‹x =ℛ y›
  AOT_hence 1: ‹InDomainOf(x,ℛ) & InDomainOf(y,ℛ)›
    using "id-R-thm:2"[THEN "→E"] by blast
  AOT_hence ‹x =ℛ y ≡ x = y›
    using "id-R-thm:4"[THEN "→E", OF "∨I"(1)] "&E" by blast
  AOT_hence ‹x = y›
    using 0 by (metis "≡E"(1))
  AOT_hence ‹y = x›
    using id_sym by blast
  moreover AOT_have ‹y =ℛ x ≡ y = x›
    using "id-R-thm:4"[THEN "→E", OF "∨I"(2)] 1 "&E" by blast
  ultimately AOT_show ‹y =ℛ x›
    by (metis "≡E"(2))
qed

AOT_theorem "id-R-thm:7": ‹x =ℛ y & y =ℛ z → x =ℛ z›                      (774.7)
proof (rule "→I"; frule "&E"(1); drule "&E"(2))
  AOT_assume 0: ‹x =ℛ y›
  AOT_hence 1: ‹InDomainOf(x,ℛ) & InDomainOf(y,ℛ)›
    using "id-R-thm:2"[THEN "→E"] by blast
  AOT_hence ‹x =ℛ y ≡ x = y›
    using "id-R-thm:4"[THEN "→E", OF "∨I"(1)] "&E" by blast
  AOT_hence x_eq_y: ‹x = y›
    using 0 by (metis "≡E"(1))
  AOT_assume 2: ‹y =ℛ z›
  AOT_hence 3: ‹InDomainOf(y,ℛ) & InDomainOf(z,ℛ)›
    using "id-R-thm:2"[THEN "→E"] by blast
  AOT_hence ‹y =ℛ z ≡ y = z›
    using "id-R-thm:4"[THEN "→E", OF "∨I"(1)] "&E" by blast
  AOT_hence ‹y = z›
    using 2 by (metis "≡E"(1))
  AOT_hence x_eq_z: ‹x = z›
    using x_eq_y id_trans by blast
  AOT_have ‹InDomainOf(x,ℛ) & InDomainOf(z,ℛ)›
    using 1 3 "&I" "&E" by meson
  AOT_hence ‹x =ℛ z ≡ x = z›
    using "id-R-thm:4"[THEN "→E", OF "∨I"(1)] "&E" by blast
  AOT_thus ‹x =ℛ z›
    using x_eq_z "≡E"(2) by blast
qed
```

```
AOT_define WeakAncestral :: ‹Π ⇒ Π› (‹_⁺›)
  "w-ances-df": ‹[ℛ]⁺ =df [λxy [ℛ]*xy ∨ x =ℛ y]›                    (775)

AOT_theorem "w-ances-df[den1]": ‹[λxy [Π]*xy ∨ x =Π y]↓›            (775)
  by "cqt:2"
AOT_theorem "w-ances-df[den2]": ‹[Π]⁺↓›                              (775)
  using "w-ances-df[den1]" "=dfI"(1)[OF "w-ances-df"] by blast

AOT_theorem "w-ances": ‹[ℛ]⁺xy ≡ ([ℛ]*xy ∨ x =ℛ y)›                  (776)
proof -
  AOT_have 0: ‹[λxy [ℛ*]xy ∨ x =ℛ y]↓›
    by "cqt:2"
  AOT_have 1: ‹«(AOT_term_of_var x,AOT_term_of_var y)»↓›
    by (simp add: "&I" "ex:1:a" prod_denotesI "rule-ui:3")
  have 2: ‹«[λμ1...μn [ℛ*]μ1...μn ∨ [(=ℛ)]μ1...μn]xy» =
           «[λxy [ℛ*]xy ∨ [(=ℛ)]xy]xy»›
    by (simp add: cond_case_prod_eta)
  show ?thesis
    apply (rule "=dfI"(1)[OF "w-ances-df"])
     apply (fact "w-ances-df[den1]")
    using "beta-C-meta"[THEN "→E", OF 0, unvarify ν1νn,
                        where τ=‹(_,_)›, simplified, OF 1] 2 by simp
qed

AOT_theorem "w-ances-her:1": ‹[ℛ]xy → [ℛ]⁺xy›
proof(rule "→I")
  AOT_assume ‹[ℛ]xy›
  AOT_hence ‹[ℛ]*xy›
    using "anc-her:1"[THEN "→E"] by blast
  AOT_thus ‹[ℛ]⁺xy›
    using "w-ances"[THEN "≡E"(2)] "∨I" by blast
qed

AOT_theorem "w-ances-her:2":
  ‹[F]x & [ℛ]⁺xy & Hereditary(F, ℛ) → [F]y›
proof(rule "→I"; (frule "&E"(1); drule "&E"(2))+)
  AOT_assume 0: ‹[F]x›
  AOT_assume 1: ‹Hereditary(F, ℛ)›
  AOT_assume ‹[ℛ]⁺xy›
  AOT_hence ‹[ℛ]*xy ∨ x =ℛ y›
    using "w-ances"[THEN "≡E"(1)] by simp
  moreover {
    AOT_assume ‹[ℛ]*xy›
    AOT_hence ‹[F]y›
      using "anc-her:3"[THEN "→E", OF "&I", OF "&I"] 0 1 by blast
  }
  moreover {
    AOT_assume ‹x =ℛ y›
    AOT_hence ‹x = y›
      using "id-R-thm:3"[THEN "→E"] by blast
    AOT_hence ‹[F]y›
      using 0 "rule=E" by blast
  }
  ultimately AOT_show ‹[F]y›
    by (metis "∨E"(3) "raa-cor:1")
qed

AOT_theorem "w-ances-her:3": ‹([ℛ]⁺xy & [ℛ]yz) → [ℛ]*xz›
proof(rule "→I"; frule "&E"(1); drule "&E"(2))
  AOT_assume ‹[ℛ]⁺xy›
  moreover AOT_assume Ryz: ‹[ℛ]yz›
  ultimately AOT_have ‹[ℛ]*xy ∨ x =ℛ y›
```

```
    using "w-ances"[THEN "≡E"(1)] by metis
  moreover {
    AOT_assume R_star_xy: ‹[ℛ]*xy›
    AOT_have ‹[ℛ]*xz›
    proof (safe intro!: ances[THEN "≡E"(2)] "→I" GEN)
      fix F
      AOT_assume 0: ‹∀z ([ℛ]xz → [F]z) & Hereditary(F,ℛ)›
      AOT_hence ‹[F]y›
        using R_star_xy ances[THEN "≡E"(1), OF R_star_xy,
                              THEN "∀E"(2), THEN "→E"] by blast
      AOT_thus ‹[F]z›
        using "hered:1"[THEN "≡dfE", OF 0[THEN "&E"(2)], THEN "&E"(2)]
              "∀E"(2) "→E" Ryz by blast
    qed
  }
  moreover {
    AOT_assume ‹x =ℛ y›
    AOT_hence ‹x = y›
      using "id-R-thm:3"[THEN "→E"] by blast
    AOT_hence ‹[ℛ]xz›
      using Ryz "rule=E" id_sym by fast
    AOT_hence ‹[ℛ]*xz›
      by (metis "anc-her:1"[THEN "→E"])
  }
  ultimately AOT_show ‹[ℛ]*xz›
    by (metis "∨E"(3) "raa-cor:1")
qed

AOT_theorem "w-ances-her:4": ‹([ℛ]*xy & [ℛ]yz) → [ℛ]+xz›
proof(rule "→I"; frule "&E"(1); drule "&E"(2))
  AOT_assume ‹[ℛ]*xy›
  AOT_hence ‹[ℛ]*xy ∨ x =ℛ y›
    using "∨I" by blast
  AOT_hence ‹[ℛ]+xy›
    using "w-ances"[THEN "≡E"(2)] by blast
  moreover AOT_assume ‹[ℛ]yz›
  ultimately AOT_have ‹[ℛ]*xz›
    using "w-ances-her:3"[THEN "→E", OF "&I"] by simp
  AOT_hence ‹[ℛ]*xz ∨ x =ℛ z›
    using "∨I" by blast
  AOT_thus ‹[ℛ]+xz›
    using "w-ances"[THEN "≡E"(2)] by blast
qed

AOT_theorem "w-ances-her:5": ‹([ℛ]xy & [ℛ]+yz) → [ℛ]*xz›
proof(rule "→I"; frule "&E"(1); drule "&E"(2))
  AOT_assume 0: ‹[ℛ]xy›
  AOT_assume ‹[ℛ]+yz›
  AOT_hence ‹[ℛ]*yz ∨ y =ℛ z›
    by (metis "≡E"(1) "w-ances")
  moreover {
    AOT_assume ‹[ℛ]*yz›
    AOT_hence ‹[ℛ]*xz›
      using 0 by (metis "anc-her:4" Adjunction "→E")
  }
  moreover {
    AOT_assume ‹y =ℛ z›
    AOT_hence ‹y = z›
      by (metis "id-R-thm:3" "→E")
    AOT_hence ‹[ℛ]xz›
      using 0 "rule=E" by fast
    AOT_hence ‹[ℛ]*xz›
      by (metis "anc-her:1" "→E")
```

```
  }
  ultimately AOT_show ‹[ℛ]*xz› by (metis "∨E"(2) "reductio-aa:1")
qed

AOT_theorem "w-ances-her:6": ‹([ℛ]⁺xy & [ℛ]⁺yz) → [ℛ]⁺xz›
proof(rule "→I"; frule "&E"(1); drule "&E"(2))
  AOT_assume 0: ‹[ℛ]⁺xy›
  AOT_hence 1: ‹[ℛ]*xy ∨ x =ℛ y›
    by (metis "≡E"(1) "w-ances")
  AOT_assume 2: ‹[ℛ]⁺yz›
  {
    AOT_assume ‹x =ℛ y›
    AOT_hence ‹x = y›
      by (metis "id-R-thm:3" "→E")
    AOT_hence ‹[ℛ]⁺xz›
      using 2 "rule=E" id_sym by fast
  }
  moreover {
    AOT_assume ‹¬(x =ℛ y)›
    AOT_hence 3: ‹[ℛ]*xy›
      using 1 by (metis "∨E"(3))
    AOT_have ‹[ℛ]*yz ∨ y =ℛ z›
      using 2 by (metis "≡E"(1) "w-ances")
    moreover {
      AOT_assume ‹[ℛ]*yz›
      AOT_hence ‹[ℛ]*xz›
        using 3 by (metis "anc-her:6" Adjunction "→E")
      AOT_hence ‹[ℛ]⁺xz›
        by (metis "∨I"(1) "≡E"(2) "w-ances")
    }
    moreover {
      AOT_assume ‹y =ℛ z›
      AOT_hence ‹y = z›
        by (metis "id-R-thm:3" "→E")
      AOT_hence ‹[ℛ]⁺xz›
        using 0 "rule=E" id_sym by fast
    }
    ultimately AOT_have ‹[ℛ]⁺xz›
      by (metis "∨E"(3) "reductio-aa:1")
  }
  ultimately AOT_show ‹[ℛ]⁺xz›
    by (metis "reductio-aa:1")
qed

AOT_theorem "w-ances-her:7": ‹[ℛ]*xy → ∃z([ℛ]⁺xz & [ℛ]zy)›
proof(rule "→I")
  AOT_assume 0: ‹[ℛ]*xy›
  AOT_have 1: ‹∀z ([ℛ]xz → [Π]z) & Hereditary(Π,ℛ) → [Π]y› if ‹Π↓› for Π
    using ances[THEN "≡E"(1), THEN "∀E"(1), OF 0] that by blast
  AOT_have ‹[λy ∃z([ℛ]⁺xz & [ℛ]zy)]y›
  proof (rule 1[THEN "→E"]; "cqt:2[lambda]"?;
         safe intro!: "&I" GEN "→I" "hered:1"[THEN "≡dfI"] "cqt:2")
    fix z
    AOT_assume 0: ‹[ℛ]xz›
    AOT_hence ‹∃z [ℛ]xz› by (rule "∃I")
    AOT_hence ‹InDomainOf(x, ℛ)› by (metis "≡dfI" "df-1-1:5")
    AOT_hence ‹x =ℛ x› by (metis "id-R-thm:5" "→E")
    AOT_hence ‹[ℛ]⁺xx› by (metis "∨I"(2) "≡E"(2) "w-ances")
    AOT_hence ‹[ℛ]⁺xx & [ℛ]xz› using 0 "&I" by blast
    AOT_hence ‹∃y ([ℛ]⁺xy & [ℛ]yz)› by (rule "∃I")
    AOT_thus ‹[λy ∃z ([ℛ]⁺xz & [ℛ]zy)]z›
      by (auto intro!: "β←C"(1) "cqt:2")
  next
```

```
    fix x' y
    AOT_assume Rx'y: ‹[ℛ]x'y›
    AOT_assume ‹[λy ∃z ([ℛ]+xz & [ℛ]zy)]x'›
    AOT_hence ‹∃z ([ℛ]+xz & [ℛ]zx')›
      using "β→C"(1) by blast
    then AOT_obtain c where c_prop: ‹[ℛ]+xc & [ℛ]cx'›
      using "∃E"[rotated] by blast
    AOT_hence ‹[ℛ]+xx'›
      by (meson Rx'y "anc-her:1" "anc-her:6" Adjunction "→E" "w-ances-her:3")
    AOT_hence ‹[ℛ]+xx' ∨ x =ℛ x'› by (rule "∨I")
    AOT_hence ‹[ℛ]*xx'› by (metis "≡E"(2) "w-ances")
    AOT_hence ‹[ℛ]*xx' & [ℛ]x'y› using Rx'y by (metis "&I")
    AOT_hence ‹∃z ([ℛ]*xz & [ℛ]zy)› by (rule "∃I")
    AOT_thus ‹[λy ∃z ([ℛ]*xz & [ℛ]zy)]y›
      by (auto intro!: "β←C"(1) "cqt:2")
  qed
  AOT_thus ‹∃z([ℛ]+xz & [ℛ]zy)›
    using "β→C"(1) by fast
qed

AOT_theorem "1-1-R:1": ‹([ℛ]xy & [ℛ]*zy) → [ℛ]+zx›                              (778.1)
proof(rule "→I"; frule "&E"(1); drule "&E"(2))
  AOT_assume ‹[ℛ]*zy›
  AOT_hence ‹∃x ([ℛ]+zx & [ℛ]xy)›
    using "w-ances-her:7"[THEN "→E"] by simp
  then AOT_obtain a where a_prop: ‹[ℛ]+za & [ℛ]ay›
    using "∃E"[rotated] by blast
  moreover AOT_assume ‹[ℛ]xy›
  ultimately AOT_have ‹x = a›
    using "df-1-1:2"[THEN "≡dfE", OF RigidOneToOneRelation.ψ, THEN "&E"(1),
                     THEN "≡dfE"[OF "df-1-1:1"], THEN "&E"(2), THEN "∀E"(2),
                     THEN "∀E"(2), THEN "∀E"(2), THEN "→E", OF "&I"]
    "&E" by blast
  AOT_thus ‹[ℛ]+zx›
    using a_prop[THEN "&E"(1)] "rule=E" id_sym by fast
qed

AOT_theorem "1-1-R:2": ‹[ℛ]xy → (¬[ℛ]*xx → ¬[ℛ]*yy)›                           (778.2)
proof(rule "→I"; rule "useful-tautologies:5"[THEN "→E"]; rule "→I")
  AOT_assume 0: ‹[ℛ]xy›
  moreover AOT_assume ‹[ℛ]*yy›
  ultimately AOT_have ‹[ℛ]+yx›
    using "1-1-R:1"[THEN "→E", OF "&I"] by blast
  AOT_thus ‹[ℛ]*xx›
    using 0 by (metis "&I" "→E" "w-ances-her:5")
qed

AOT_theorem "1-1-R:3": ‹¬[ℛ]*xx → ([ℛ]+xy → ¬[ℛ]*yy)›                           (778.3)
proof(safe intro!: "→I")
  AOT_have 0: ‹[λz ¬[ℛ]*zz]↓› by "cqt:2"
  AOT_assume 1: ‹¬[ℛ]*xx›
  AOT_assume 2: ‹[ℛ]+xy›
  AOT_have ‹[λz ¬[ℛ]*zz]y›
  proof(rule "w-ances-her:2"[unvarify F, OF 0, THEN "→E"];
        safe intro!: "&I" "hered:1"[THEN "≡dfI"] "cqt:2" GEN "→I")
    AOT_show  ‹[λz ¬[ℛ]*zz]x›
      by (auto intro!: "β←C"(1) "cqt:2" simp: 1)
  next
    AOT_show ‹[ℛ]+xy› by (fact 2)
  next
    fix x y
    AOT_assume ‹[λz ¬[ℛ*]zz]x›
    AOT_hence ‹¬[ℛ]*xx› by (rule "β→C"(1))
```

```
    moreover AOT_assume ‹[ℛ]xy›
    ultimately AOT_have ‹¬[ℛ]*yy›
      using "1-1-R:2"[THEN "→E", THEN "→E"] by blast
    AOT_thus ‹[λz ¬[ℛ*]zz]y›
      by (auto intro!: "β←C"(1) "cqt:2")
  qed
  AOT_thus ‹¬[ℛ]*yy›
    using "β→C"(1) by blast
qed

AOT_theorem "1-1-R:4": ‹[ℛ]*xy → InDomainOf(x,ℛ)›                     (778.4)
proof(rule "→I"; rule "df-1-1:5"[THEN "≡dfI"])
  AOT_assume 1: ‹[ℛ]*xy›
  AOT_have ‹[λz [ℛ*]xz → ∃y [ℛ]xy]y›
  proof (safe intro!: "anc-her:2"[unvarify F, THEN "→E"];
         safe intro!: "cqt:2" "&I" GEN "→I" "hered:1"[THEN "≡dfI"])
    AOT_show ‹[ℛ]*xy› by (fact 1)
  next
    fix z
    AOT_assume ‹[ℛ]xz›
    AOT_thus ‹[λz [ℛ*]xz → ∃y [ℛ]xy]z›
      by (safe intro!: "β←C"(1) "cqt:2")
         (meson "→I" "existential:2[const_var]")
  next
    fix x' y
    AOT_assume Rx'y: ‹[ℛ]x'y›
    AOT_assume ‹[λz [ℛ*]xz → ∃y [ℛ]xy]x'›
    AOT_hence 0: ‹[ℛ*]xx' → ∃y [ℛ]xy› by (rule "β→C"(1))
    AOT_have 1: ‹[ℛ*]xy → ∃y [ℛ]xy›
    proof(rule "→I")
      AOT_assume ‹[ℛ]*xy›
      AOT_hence ‹[ℛ]+xx'› by (metis Rx'y "&I" "1-1-R:1" "→E")
      AOT_hence ‹[ℛ]*xx' ∨ x =ℛ x'› by (metis "≡E"(1) "w-ances")
      moreover {
        AOT_assume ‹[ℛ]*xx'›
        AOT_hence ‹∃y [ℛ]xy› using 0 by (metis "→E")
      }
      moreover {
        AOT_assume ‹x =ℛ x'›
        AOT_hence ‹x = x'› by (metis "id-R-thm:3" "→E")
        AOT_hence ‹[ℛ]xy› using Rx'y "rule=E" id_sym by fast
        AOT_hence ‹∃y [ℛ]xy› by (rule "∃I")
      }
      ultimately AOT_show ‹∃y [ℛ]xy›
        by (metis "∨E"(3) "reductio-aa:1")
    qed
    AOT_show ‹[λz [ℛ*]xz → ∃y [ℛ]xy]y›
      by (auto intro!: "β←C"(1) "cqt:2" 1)
  qed
  AOT_hence ‹[ℛ*]xy → ∃y [ℛ]xy› by (rule "β→C"(1))
  AOT_thus ‹∃y [ℛ]xy› using 1 "→E" by blast
qed

AOT_theorem "1-1-R:5": ‹[ℛ]+xy → InDomainOf(x,ℛ)›                     (778.5)
proof (rule "→I")
  AOT_assume ‹[ℛ]+xy›
  AOT_hence ‹[ℛ]*xy ∨ x =ℛ y›
    by (metis "≡E"(1) "w-ances")
  moreover {
    AOT_assume ‹[ℛ]*xy›
    AOT_hence ‹InDomainOf(x,ℛ)›
      using "1-1-R:4" "→E" by blast
  }
```

```
  moreover {
    AOT_assume ‹x =ℛ y›
    AOT_hence ‹InDomainOf(x,ℛ)›
      by (metis "Conjunction Simplification"(1) "id-R-thm:2" "→E")
  }
  ultimately AOT_show ‹InDomainOf(x,ℛ)›
    by (metis "∨E"(3) "reductio-aa:1")
qed

AOT_theorem "pre-ind":                                                      (779)
  ‹([F]z & ∀x∀y(([ℛ]⁺zx & [ℛ]⁺zy) → ([ℛ]xy → ([F]x → [F]y)))) →
   ∀x ([ℛ]⁺zx → [F]x)›
proof(safe intro!: "→I" GEN)
  AOT_have den: ‹[λy [F]y & [ℛ]⁺zy]↓› by "cqt:2"
  fix x
  AOT_assume ϑ: ‹[F]z & ∀x∀y(([ℛ]⁺zx & [ℛ]⁺zy) → ([ℛ]xy → ([F]x → [F]y)))›
  AOT_assume 0: ‹[ℛ]⁺zx›

  AOT_have ‹[λy [F]y & [ℛ]⁺zy]x›
  proof (rule "w-ances-her:2"[unvarify F, OF den, THEN "→E"]; safe intro!: "&I")
    AOT_show ‹[λy [F]y & [ℛ]⁺zy]z›
    proof (safe intro!: "β←C"(1) "cqt:2" "&I")
      AOT_show ‹[F]z› using ϑ "&E" by blast
    next
      AOT_show ‹[ℛ]⁺zz›
        by (rule "w-ances"[THEN "≡E"(2), OF "∨I"(2)])
           (meson "0" "id-R-thm:5" "1-1-R:5" "→E")
    qed
  next
    AOT_show ‹[ℛ]⁺zx› by (fact 0)
  next
    AOT_show ‹Hereditary([λy [F]y & [ℛ]⁺zy],ℛ)›
    proof (safe intro!: "hered:1"[THEN "≡dfI"] "&I" "cqt:2" GEN "→I")
      fix x' y
      AOT_assume 1: ‹[ℛ]x'y›
      AOT_assume ‹[λy [F]y & [ℛ]⁺zy]x'›
      AOT_hence 2: ‹[F]x' & [ℛ]⁺zx'› by (rule "β→C"(1))
      AOT_have ‹[ℛ]*zy› using 1 2[THEN "&E"(2)]
        by (metis Adjunction "modus-tollens:1" "reductio-aa:1" "w-ances-her:3")
      AOT_hence 3: ‹[ℛ]⁺zy› by (metis "∨I"(1) "≡E"(2) "w-ances")
      AOT_show ‹[λy [F]y & [ℛ]⁺zy]y›
      proof (safe intro!: "β←C"(1) "cqt:2" "&I" 3)
        AOT_show ‹[F]y›
        proof (rule ϑ[THEN "&E"(2), THEN "∀E"(2), THEN "∀E"(2),
                      THEN "→E", THEN "→E", THEN "→E"])
          AOT_show ‹[ℛ]⁺zx' & [ℛ]⁺zy›
            using 2 3 "&E" "&I" by blast
        next
          AOT_show ‹[ℛ]x'y› by (fact 1)
        next
          AOT_show ‹[F]x'› using 2 "&E" by blast
        qed
      qed
    qed
  qed
  AOT_thus ‹[F]x› using "β→C"(1) "&E"(1) by fast
qed

text‹The following is not part of PLM, but a theorem of AOT.
     It states that the predecessor relation coexists with numbering a property.
     We will use this fact to derive the predecessor axiom, which asserts that the
     predecessor relation denotes, from the fact that our models validate that
     numbering a property denotes.›
```

```
AOT_theorem pred_coex:
  ‹[λxy ∃F∃u ([F]u & Numbers(y,F) & Numbers(x,[F]⁻ᵘ))]↓ ≡ ∀F ([λx Numbers(x,F)]↓)›
proof(safe intro!: "≡I" "→I" GEN)
  fix F
  let ?P = ‹«[λxy ∃F∃u ([F]u & Numbers(y,F) & Numbers(x,[F]⁻ᵘ))]»›
  AOT_assume ‹[«?P»]↓›
  AOT_hence ‹□[«?P»]↓›
    using "exist-nec" "→E" by blast
  moreover AOT_have
    ‹□[«?P»]↓ → □(∀x∀y(∀F([F]x ≡ [F]y) → (Numbers(x,F) ≡ Numbers(y,F))))›
  proof(rule RM; safe intro!: "→I" GEN)
    AOT_modally_strict {
      fix x y
      AOT_assume pred_den: ‹[«?P»]↓›
      AOT_hence pred_equiv:
        ‹[«?P»]xy ≡ ∃F∃u ([F]u & Numbers(y,F) & Numbers(x,[F]⁻ᵘ))› for x y
        by (safe intro!: "beta-C-meta"[unvarify ν₁νₙ, where τ=‹(_,_)›, THEN "→E",
                                      rotated, OF pred_den, simplified]
                          tuple_denotes[THEN "≡dfI"] "&I" "cqt:2")
      text‹We show as a subproof that any natural cardinal that is not zero
           has a predecessor.›
      AOT_have CardinalPredecessor:
        ‹∃y [«?P»]yx› if card_x: ‹NaturalCardinal(x)› and x_nonzero: ‹x ≠ 0› for x
      proof -
        AOT_have ‹∃G x = #G›
          using card[THEN "≡dfE", OF card_x].
        AOT_hence ‹∃G Numbers(x,G)›
          using "eq-df-num"[THEN "≡E"(1)] by blast
        then AOT_obtain G' where numxG': ‹Numbers(x,G')›
          using "∃E"[rotated] by blast
        AOT_obtain G where ‹Rigidifies(G,G')›
          using "rigid-der:3" "∃E"[rotated] by blast

        AOT_hence H: ‹Rigid(G) & ∀x ([G]x ≡ [G']x)›
          using "df-rigid-rel:2"[THEN "≡dfE"] by blast
        AOT_have H_rigid: ‹□∀x ([G]x → □[G]x)›
          using H[THEN "&E"(1), THEN "df-rigid-rel:1"[THEN "≡dfE"], THEN "&E"(2)].
        AOT_hence ‹∀x □([G]x → □[G]x)›
          using "CBF" "→E" by blast
        AOT_hence R: ‹□([G]x → □[G]x)› for x using "∀E"(2) by blast
        AOT_hence rigid: ‹[G]x ≡ 𝒜[G]x› for x
           by (metis "≡E"(6) "oth-class-taut:3:a" "sc-eq-fur:2" "→E")
        AOT_have ‹G ≡E G'›
        proof (safe intro!: eqE[THEN "≡dfI"] "&I" "cqt:2" GEN "→I")
          AOT_show ‹[G]x ≡ [G']x› for x using H[THEN "&E"(2)] "∀E"(2) by fast
        qed
        AOT_hence ‹G ≈E G'›
          by (rule "apE-eqE:2"[THEN "→E", OF "&I", rotated])
             (simp add: "eq-part:1")
        AOT_hence numxG: ‹Numbers(x,G)›
          using "num-tran:1"[THEN "→E", THEN "≡E"(2)] numxG' by blast

        {
          AOT_assume ‹¬∃y(y ≠ x & [«?P»]yx)›
          AOT_hence ‹∀y ¬(y ≠ x & [«?P»]yx)›
            using "cqt-further:4" "→E" by blast
          AOT_hence ‹¬(y ≠ x & [«?P»]yx)› for y
            using "∀E"(2) by blast
          AOT_hence 0: ‹¬y ≠ x ∨ ¬[«?P»]yx› for y
            using "¬¬E" "intro-elim:3:c" "oth-class-taut:5:a" by blast
          {
            fix y
            AOT_assume ‹[«?P»]yx›
```

```
        AOT_hence ‹¬y ≠ x›
          using 0 "¬¬I" "con-dis-i-e:4:c" by blast
        AOT_hence ‹y = x›
          using "=-infix" "≡dfI" "raa-cor:4" by blast
      } note Pxy_imp_eq = this
      AOT_have ‹[«?P»]xx›
      proof(rule "raa-cor:1")
        AOT_assume notPxx: ‹¬[«?P»]xx›
        AOT_hence ‹¬∃F∃u([F]u & Numbers(x,F) & Numbers(x,[F]^-u))›
          using pred_equiv "intro-elim:3:c" by blast
        AOT_hence ‹∀F ¬∃u([F]u & Numbers(x,F) & Numbers(x,[F]^-u))›
          using "cqt-further:4"[THEN "→E"] by blast
        AOT_hence ‹¬∃u([F]u & Numbers(x,F) & Numbers(x,[F]^-u))› for F
          using "∀E"(2) by blast
        AOT_hence ‹∀y ¬(O!y & ([F]y & Numbers(x,F) & Numbers(x,[F]^-y)))› for F
          using "cqt-further:4"[THEN "→E"] by blast
        AOT_hence 0: ‹¬(O!u & ([F]u & Numbers(x,F) & Numbers(x,[F]^-u)))› for F u
          using "∀E"(2) by blast
        AOT_have ‹□¬∃u [G]u›
        proof(rule "raa-cor:1")
          AOT_assume ‹¬□¬∃u [G]u›
          AOT_hence ‹◇∃u [G]u›
            using "≡dfI" "conventions:5" by blast
          AOT_hence ‹∃u ◇[G]u›
            by (metis "Ordinary.res-var-bound-reas[BF◇]"[THEN "→E"])
          then AOT_obtain u where posGu: ‹◇[G]u›
            using "Ordinary.∃E"[rotated] by meson
          AOT_hence Gu: ‹[G]u›
            by (meson "B◇" "K◇" "→E" R)
          AOT_have ‹¬([G]u & Numbers(x,G) & Numbers(x,[G]^-u))›
            using 0 Ordinary.ψ
            by (metis "con-dis-i-e:1" "raa-cor:1")
          AOT_hence notnumx: ‹¬Numbers(x,[G]^-u)›
            using Gu numxG "con-dis-i-e:1" "raa-cor:5" by metis
          AOT_obtain y where numy: ‹Numbers(y,[G]^-u)›
            using "num:1"[unvarify G, OF "F-u[den]"] "∃E"[rotated] by blast
          AOT_hence ‹[G]u & Numbers(x,G) & Numbers(y,[G]^-u)›
            using Gu numxG "&I" by blast
          AOT_hence ‹∃u ([G]u & Numbers(x,G) & Numbers(y,[G]^-u))›
            by (rule "Ordinary.∃I")
          AOT_hence ‹∃G∃u ([G]u & Numbers(x,G) & Numbers(y,[G]^-u))›
            by (rule "∃I")
          AOT_hence ‹[«?P»]yx›
            using pred_equiv[THEN "≡E"(2)] by blast
          AOT_hence ‹y = x› using Pxy_imp_eq by blast
          AOT_hence ‹Numbers(x,[G]^-u)›
            using numy "rule=E" by fast
          AOT_thus ‹p & ¬p› for p using notnumx "reductio-aa:1" by blast
        qed
        AOT_hence ‹¬∃u [G]u›
          using "qml:2"[axiom_inst, THEN "→E"] by blast
        AOT_hence num0G: ‹Numbers(0, G)›
          using "0F:1"[THEN "≡E"(1)] by blast
        AOT_hence ‹x = 0›
          using "pre-Hume"[unvarify x, THEN "→E", OF "zero:2", OF "&I",
                           THEN "≡E"(2), OF num0G, OF numxG, OF "eq-part:1"]
            id_sym by blast
        moreover AOT_have ‹¬x = 0›
          using x_nonzero
          using "=-infix" "≡dfE" by blast
        ultimately AOT_show ‹p & ¬p› for p using "reductio-aa:1" by blast
      qed
    }
```

```
        AOT_hence ‹[«?P»]xx ∨ ∃y (y ≠ x & [«?P»]yx)›
          using "con-dis-i-e:3:a" "con-dis-i-e:3:b" "raa-cor:1" by blast
        moreover {
          AOT_assume ‹[«?P»]xx›
          AOT_hence ‹∃y [«?P»]yx›
            by (rule "∃I")
        }
        moreover {
          AOT_assume ‹∃y (y ≠ x & [«?P»]yx)›
          then AOT_obtain y where ‹y ≠ x & [«?P»]yx›
            using "∃E"[rotated] by blast
          AOT_hence ‹[«?P»]yx›
            using "&E" by blast
          AOT_hence ‹∃y [«?P»]yx›
            by (rule "∃I")
        }
        ultimately AOT_show ‹∃y [«?P»]yx›
          using "∨E"(1) "→I" by blast
      qed

      text‹Given above lemma, we can show that if one of two indistinguishable objects
           numbers a property, the other one numbers this property as well.›
      AOT_assume indist: ‹∀F([F]x ≡ [F]y)›
      AOT_assume numxF: ‹Numbers(x,F)›
      AOT_hence 0: ‹NaturalCardinal(x)›
        by (metis "eq-num:6" "vdash-properties:10")
      text‹We show by case distinction that x equals y.
           As first case we consider x to be non-zero.›
      {
        AOT_assume ‹¬(x = 0)›
        AOT_hence ‹x ≠ 0›
          by (metis "=-infix" "≡dfI")
        AOT_hence ‹∃y [«?P»]yx›
          using CardinalPredecessor 0 by blast
        then AOT_obtain z where Pxz: ‹[«?P»]zx›
          using "∃E"[rotated] by blast
        AOT_hence ‹[λy [«?P»]zy]x›
          by (safe intro!: "β←C" "cqt:2")
        AOT_hence ‹[λy [«?P»]zy]y›
          by (safe intro!: indist[THEN "∀E"(1), THEN "≡E"(1)] "cqt:2")
        AOT_hence Pyz: ‹[«?P»]zy›
          using "β→C"(1) by blast
        AOT_hence ‹∃F∃u ([F]u & Numbers(y,F) & Numbers(z,[F]⁻ᵘ))›
          using Pyz pred_equiv[THEN "≡E"(1)] by blast
        then AOT_obtain F₁ where ‹∃u ([F₁]u & Numbers(y,F₁) & Numbers(z,[F₁]⁻ᵘ))›
          using "∃E"[rotated] by blast
        then AOT_obtain u where u_prop: ‹[F₁]u & Numbers(y,F₁) & Numbers(z,[F₁]⁻ᵘ)›
          using "Ordinary.∃E"[rotated] by meson
        AOT_have ‹∃F∃u ([F]u & Numbers(x,F) & Numbers(z,[F]⁻ᵘ))›
          using Pxz pred_equiv[THEN "≡E"(1)] by blast
        then AOT_obtain F₂ where ‹∃u ([F₂]u & Numbers(x,F₂) & Numbers(z,[F₂]⁻ᵘ))›
          using "∃E"[rotated] by blast
        then AOT_obtain v where v_prop: ‹[F₂]v & Numbers(x,F₂) & Numbers(z,[F₂]⁻ᵛ)›
          using "Ordinary.∃E"[rotated] by meson
        AOT_have ‹[F₂]⁻ᵛ ≈E [F₁]⁻ᵘ›
          using "hume-strict:1"[unvarify F G, THEN "≡E"(1), OF "F-u[den]",
                                OF "F-u[den]", OF "∃I"(2)[where β=z], OF "&I"]
                v_prop u_prop "&E" by blast
        AOT_hence ‹F₂ ≈E F₁›
          using "P'-eq"[THEN "→E", OF "&I", OF "&I"]
               u_prop v_prop "&E" by meson
        AOT_hence ‹x = y›
          using "pre-Hume"[THEN "→E", THEN "≡E"(2), OF "&I"]
```

```
              v_prop u_prop "&E" by blast
    }
    text‹The second case handles x being equal to zero.›
    moreover {
      fix u
      AOT_assume x_is_zero: ‹x = 0›
      moreover AOT_have ‹Numbers(0,[λz z =E u]^-u)›
      proof (safe intro!: "OF:1"[unvarify F, THEN "≡E"(1)] "cqt:2" "raa-cor:2"
                          "F-u[den]"[unvarify F])
        AOT_assume ‹∃v [[λz z =E u]^-u]v›
        then AOT_obtain v where ‹[[λz z =E u]^-u]v›
          using "Ordinary.∃E"[rotated] by meson
        AOT_hence ‹[λz z =E u]v & v ≠E u›
          by (auto intro: "F-u"[THEN "=dfE"(1), where τ1τn="(_,_)", simplified]
                   intro!: "cqt:2" "F-u[equiv]"[unvarify F, THEN "≡E"(1)]
                           "F-u[den]"[unvarify F])
        AOT_thus ‹p & ¬p› for p
          using "β→C" "thm-neg=E"[THEN "≡E"(1)] "&E" "&I"
                "raa-cor:3" by fast
      qed
      ultimately AOT_have 0: ‹Numbers(x,[λz z =E u]^-u)›
        using "rule=E" id_sym by fast
      AOT_have ‹∃y Numbers(y,[λz z =E u])›
        by (safe intro!: "num:1"[unvarify G] "cqt:2")
      then AOT_obtain z where ‹Numbers(z,[λz z =E u])›
        using "∃E" by metis
      moreover AOT_have ‹[λz z=E u]u›
        by (safe intro!: "β←C" "cqt:2" "ord=Eequiv:1"[THEN "→E"] Ordinary.ψ)
      ultimately AOT_have
        1: ‹[λz z=E u]u & Numbers(z,[λz z=E u]) & Numbers(x,[λz z=E u]^-u)›
        using 0 "&I" by auto
      AOT_hence ‹∃v([λz z=E u]v & Numbers(z,[λz z =E u]) & Numbers(x,[λz z=E u]^-v))›
        by (rule "Ordinary.∃I")
      AOT_hence ‹∃F∃u([F]u & Numbers(z,[F]) & Numbers(x,[F]^-u))›
        by (rule "∃I"; "cqt:2")
      AOT_hence Px1: ‹[«?P»]xz›
        using "beta-C-cor:2"[THEN "→E", OF pred_den,
                THEN tuple_forall[THEN "≡dfE"], THEN "∀E"(2),
                THEN "∀E"(2), THEN "≡E"(2)] by simp
      AOT_hence ‹[λy [«?P»]yz]x›
        by (safe intro!: "β←C" "cqt:2")
      AOT_hence ‹[λy [«?P»]yz]y›
        by (safe intro!: indist[THEN "∀E"(1), THEN "≡E"(1)] "cqt:2")
      AOT_hence Py1: ‹[«?P»]yz›
        using "β→C" by blast
      AOT_hence ‹∃F∃u([F]u & Numbers(z,[F]) & Numbers(y,[F]^-u))›
        using "β→C" by fast
      then AOT_obtain G where ‹∃u([G]u & Numbers(z,[G]) & Numbers(y,[G]^-u))›
        using "∃E"[rotated] by blast
      then AOT_obtain v where 2: ‹[G]v & Numbers(z,[G]) & Numbers(y,[G]^-v)›
        using "Ordinary.∃E"[rotated] by meson
      with 1 2 AOT_have ‹[λz z =E u] ≈E G›
        by (auto intro!: "hume-strict:1"[unvarify F, THEN "≡E"(1), rotated,
                            OF "∃I"(2)[where β=z], OF "&I"] "cqt:2"
                 dest: "&E")
      AOT_hence 3: ‹[λz z =E u]^-u ≈E [G]^-v›
        using 1 2
        by (safe_step intro!: "eqP'"[unvarify F, THEN "→E"])
           (auto dest: "&E" intro!: "cqt:2" "&I")
      with 1 2 AOT_have ‹x = y›
        by (auto intro!: "pre-Hume"[unvarify G H, THEN "→E",
                               THEN "≡E"(2), rotated 3, OF 3]
                    "F-u[den]"[unvarify F] "cqt:2" "&I"
```

```
                 dest: "&E")
      }
      ultimately AOT_have ‹x = y›
        using "∨E"(1) "→I" "reductio-aa:1" by blast
      text‹Now since x numbers F, so does y.›
      AOT_hence ‹Numbers(y,F)›
        using numxF "rule=E" by fast
    } note 0 = this
    text‹The only thing left is to generalize this result to a biconditional.›
    AOT_modally_strict {
      fix x y
      AOT_assume ‹[«?P»]↓›
      moreover AOT_assume ‹∀F([F]x ≡ [F]y)›
      moreover AOT_have ‹∀F([F]y ≡ [F]x)›
        by (metis "cqt-basic:11" "intro-elim:3:a" calculation(2))
      ultimately AOT_show ‹Numbers(x,F) ≡ Numbers(y,F)›
        using 0 "≡I" "→I" by auto
    }
  qed
  ultimately AOT_show ‹[λx Numbers(x,F)]↓›
    using "kirchner-thm:1"[THEN "≡E"(2)] "→E" by fast
next
  text‹The converse can be shown by coexistence.›
  AOT_assume ‹∀F [λx Numbers(x,F)]↓›
  AOT_hence ‹[λx Numbers(x,F)]↓› for F
    using "∀E"(2) by blast
  AOT_hence ‹□[λx Numbers(x,F)]↓› for F
    using "exist-nec"[THEN "→E"] by blast
  AOT_hence ‹∀F □[λx Numbers(x,F)]↓›
    by (rule GEN)
  AOT_hence ‹□∀F [λx Numbers(x,F)]↓›
    using BF[THEN "→E"] by fast
  moreover AOT_have
    ‹□∀F [λx Numbers(x,F)]↓ →
     □∀x ∀y (∃F ∃u ([F]u & [λz Numbers(z,F)]y & [λz Numbers(z,[F]-u)]x) ≡
             ∃F ∃u ([F]u & Numbers(y,F) & Numbers(x,[F]-u)))›
  proof(rule RM; safe intro!: "→I" GEN)
    AOT_modally_strict {
      fix x y
      AOT_assume 0: ‹∀F [λx Numbers(x,F)]↓›
      AOT_show ‹∃F ∃u ([F]u & [λz Numbers(z,F)]y & [λz Numbers(z,[F]-u)]x) ≡
             ∃F ∃u ([F]u & Numbers(y,F) & Numbers(x,[F]-u))›
      proof(safe intro!: "≡I" "→I")
        AOT_assume ‹∃F ∃u ([F]u & [λz Numbers(z,F)]y & [λz Numbers(z,[F]-u)]x)›
        then AOT_obtain F where
          ‹∃u ([F]u & [λz Numbers(z,F)]y & [λz Numbers(z,[F]-u)]x)›
          using "∃E"[rotated] by blast
        then AOT_obtain u where ‹[F]u & [λz Numbers(z,F)]y & [λz Numbers(z,[F]-u)]x›
          using "Ordinary.∃E"[rotated] by meson
        AOT_hence ‹[F]u & Numbers(y,F) & Numbers(x,[F]-u)›
          by (auto intro!: "&I" dest: "&E" "β→C")
        AOT_thus ‹∃F ∃u ([F]u & Numbers(y,F) & Numbers(x,[F]-u))›
          using "∃I" "Ordinary.∃I" by fast
      next
        AOT_assume ‹∃F ∃u ([F]u & Numbers(y,F) & Numbers(x,[F]-u))›
        then AOT_obtain F where ‹∃u ([F]u & Numbers(y,F) & Numbers(x,[F]-u))›
          using "∃E"[rotated] by blast
        then AOT_obtain u where ‹[F]u & Numbers(y,F) & Numbers(x,[F]-u)›
          using "Ordinary.∃E"[rotated] by meson
        AOT_hence ‹[F]u & [λz Numbers(z,F)]y & [λz Numbers(z,[F]-u)]x›
          by (auto intro!: "&I" "β←C" 0[THEN "∀E"(1)] "F-u[den]"
                   dest: "&E" intro: "cqt:2")
        AOT_hence ‹∃u([F]u & [λz Numbers(z,F)]y & [λz Numbers(z,[F]-u)]x)›
```

```
          by (rule "Ordinary.∃I")
        AOT_thus ‹∃F∃u([F]u & [λz Numbers(z,F)]y & [λz Numbers(z,[F]⁻ᵘ)]x)›
          by (rule "∃I")
      qed
    }
  qed
  ultimately AOT_have
    ‹□∀x ∀y (∃F ∃u ([F]u & [λz Numbers(z,F)]y & [λz Numbers(z,[F]⁻ᵘ)]x) ≡
              ∃F ∃u ([F]u & Numbers(y,F) & Numbers(x,[F]⁻ᵘ)))›
    using "→E" by blast
  AOT_thus ‹[λxy ∃F ∃u ([F]u & Numbers(y,F) & Numbers(x,[F]⁻ᵘ))]↓›
    by (rule "safe-ext[2]"[axiom_inst, THEN "→E", OF "&I", rotated]) "cqt:2"
qed

text‹The following is not part of PLM, but a consequence of extended relation
     comprehension and can be used to @{emph ‹derive›} the predecessor axiom.›
AOT_theorem numbers_prop_den: ‹[λx Numbers(x,G)]↓›
proof (rule "safe-ext"[axiom_inst, THEN "→E", OF "&I"])
  AOT_show ‹[λx A!x & [λx ∀F (x[F] ≡ [λz 𝓐[F]z] ≈E G)]x]↓›
    by "cqt:2"
next
  AOT_have 0: ‹⊢□ [λx ∀F (x[F] ≡ [λz 𝓐[F]z] ≈E G)]↓›
  proof(safe intro!: Comprehension_3[THEN "→E"] "→I" RN GEN)
      AOT_modally_strict {
        fix F H
        AOT_assume ‹□H ≡E F›
        AOT_hence ‹□∀u ([H]u ≡ [F]u)›
          by (AOT_subst (reverse) ‹∀u ([H]u ≡ [F]u)› ‹H ≡E F›)
               (safe intro!: "eqE"[THEN "≡Df", THEN "≡S"(1), OF "&I"] "cqt:2")
        AOT_hence ‹∀u □([H]u ≡ [F]u)›
          by (metis "Ordinary.res-var-bound-reas[CBF]" "→E")
        AOT_hence ‹□([H]u ≡ [F]u)› for u
          using "Ordinary.∀E" by fast
        AOT_hence ‹𝓐([H]u ≡ [F]u)› for u
          by (metis "nec-imp-act" "→E")
        AOT_hence ‹𝓐([F]u ≡ [H]u)› for u
          by (metis "Act-Basic:5" "Commutativity of ≡" "intro-elim:3:b")
        AOT_hence ‹[λz 𝓐[F]z] ≡E [λz 𝓐[H]z]›
          by (safe intro!: "eqE"[THEN "≡dfI"] "&I" "cqt:2" Ordinary.GEN;
               AOT_subst ‹[λz 𝓐[F]z]u› ‹𝓐[F]u› for: u F)
              (auto intro!: "beta-C-meta"[THEN "→E"] "cqt:2"
                            "Act-Basic:5"[THEN "≡E"(1)])
        AOT_hence ‹[λz 𝓐[F]z] ≈E [λz 𝓐[H]z]›
          by (safe intro!: "apE-eqE:1"[unvarify F G, THEN "→E"] "cqt:2")
        AOT_thus ‹[λz 𝓐[F]z] ≈E G ≡ [λz 𝓐[H]z] ≈E G›
          using "≡I" "eq-part:2[terms]" "eq-part:3[terms]" "→E" "→I"
          by metis
      }
  qed
  AOT_show ‹□∀x (A!x & [λx ∀F (x[F] ≡ [λz 𝓐[F]z] ≈E G)]x ≡ Numbers(x,G))›
  proof (safe intro!: RN GEN)
    AOT_modally_strict {
      fix x
      AOT_show ‹A!x & [λx ∀F (x[F] ≡ [λz 𝓐[F]z] ≈E G)]x ≡ Numbers(x,G)›
        by (AOT_subst_def numbers; AOT_subst_thm "beta-C-meta"[THEN "→E", OF 0])
           (auto intro!: "beta-C-meta"[THEN "→E", OF 0] "≡I" "→I" "&I" "cqt:2"
                 dest: "&E")
    }
  qed
qed

text‹The two theorems above allow us to derive
     the predecessor axiom of PLM as theorem.›
```

```
AOT_theorem pred: ‹[λxy ∃F∃u ([F]u & Numbers(y,F) & Numbers(x,[F]^-u))]↓›   (782)
  using pred_coex numbers_prop_den["∀I" G] "≡E" by blast

AOT_define Predecessor :: ‹Π› (‹ℙ›)
  "pred-thm:1":                                                             (783.1)
  ‹ℙ =df [λxy ∃F∃u ([F]u & Numbers(y,F) & Numbers(x,[F]^-u))]›

AOT_theorem "pred-thm:2": ‹ℙ↓›                                               (783.2)
  using pred "pred-thm:1" "rule-id-df:2:b[zero]" by blast

AOT_theorem "pred-thm:3":                                                    (783.3)
  ‹[ℙ]xy ≡ ∃F∃u ([F]u & Numbers(y,F) & Numbers(x,[F]^-u))›
   by (auto intro!: "beta-C-meta"[unvarify ν1νn, where τ=‹(_,_)›, THEN "→E",
                                rotated, OF pred, simplified]
                     tuple_denotes[THEN "≡dfI"] "&I" "cqt:2" pred
              intro: "=dfI"(2)[OF "pred-thm:1"])

AOT_theorem "pred-1-1:1": ‹[ℙ]xy → □[ℙ]xy›                                   (784.1)
proof(rule "→I")
  AOT_assume ‹[ℙ]xy›
  AOT_hence ‹∃F∃u ([F]u & Numbers(y,F) & Numbers(x,[F]^-u))›
    using "≡E"(1) "pred-thm:3" by fast
  then AOT_obtain F where ‹∃u ([F]u & Numbers(y,F) & Numbers(x,[F]^-u))›
    using "∃E"[rotated] by blast
  then AOT_obtain u where props: ‹[F]u & Numbers(y,F) & Numbers(x,[F]^-u)›
    using "Ordinary.∃E"[rotated] by meson
  AOT_obtain G where Ridigifies_G_F: ‹Rigidifies(G, F)›
    by (metis "instantiation" "rigid-der:3")
  AOT_hence ξ: ‹□∀x([G]x → □[G]x)› and ζ: ‹∀x([G]x ≡ [F]x)›
    using "df-rigid-rel:2"[THEN "≡dfE", THEN "&E"(1),
                           THEN "≡dfE"[OF "df-rigid-rel:1"], THEN "&E"(2)]
          "df-rigid-rel:2"[THEN "≡dfE", THEN "&E"(2)] by blast+

  AOT_have rigid_num_nec: ‹Numbers(x,F) & Rigidifies(G,F) → □Numbers(x,G)›
    for x G F
  proof(rule "→I"; frule "&E"(1); drule "&E"(2))
    fix G F x
    AOT_assume Numbers_xF: ‹Numbers(x,F)›
    AOT_assume ‹Rigidifies(G,F)›
    AOT_hence ξ: ‹Rigid(G)› and ζ: ‹∀x([G]x ≡ [F]x)›
      using "df-rigid-rel:2"[THEN "≡dfE"] "&E" by blast+
    AOT_thus ‹□Numbers(x,G)›
    proof (safe intro!:
          "num-cont:2"[THEN "→E", OF ξ, THEN "qml:2"[axiom_inst, THEN "→E"],
                       THEN "∀E"(2), THEN "→E"]
          "num-tran:3"[THEN "→E", THEN "≡E"(1), rotated, OF Numbers_xF]
          eqE[THEN "≡dfI"]
            "&I" "cqt:2[const_var]"[axiom_inst] Ordinary.GEN "→I")
      AOT_show ‹[F]u ≡ [G]u› for u
        using ζ[THEN "∀E"(2)] by (metis "≡E"(6) "oth-class-taut:3:a")
    qed
  qed
  AOT_have ‹□Numbers(y,G)›
    using rigid_num_nec[THEN "→E", OF "&I", OF props[THEN "&E"(1), THEN "&E"(2)],
                        OF Ridigifies_G_F].
  moreover {
    AOT_have ‹Rigidifies([G]^-u, [F]^-u)›
    proof (safe intro!: "df-rigid-rel:1"[THEN "≡dfI"] "df-rigid-rel:2"[THEN "≡dfI"]
                        "&I" "F-u[den]" GEN "≡I" "→I")
      AOT_have ‹□∀x([G]x → □[G]x) → □∀x([[G]^-u]x → □[[G]^-u]x)›
      proof (rule RM; safe intro!: "→I" GEN)
        AOT_modally_strict {
```

```
        fix x
        AOT_assume 0: ‹∀x([G]x → □[G]x)›
        AOT_assume 1: ‹[[G]⁻ᵘ]x›
        AOT_have ‹[λx [G]x & x ≠E u]x›
          apply (rule "F-u"[THEN "=dfE"(1), where τ₁τₙ="(_,_)", simplified])
           apply "cqt:2[lambda]"
          by (fact 1)
        AOT_hence ‹[G]x & x ≠E u›
          by (rule "β→C"(1))
        AOT_hence 2: ‹□[G]x› and 3: ‹◇x ≠E u›
          using "&E" 0[THEN "∀E"(2), THEN "→E"] "id-nec4:1" "≡E"(1) by blast+
        AOT_show ‹□[[G]⁻ᵘ]x›
          apply (AOT_subst ‹[[G]⁻ᵘ]x› ‹[G]x & x ≠E u›)
           apply (rule "F-u"[THEN "=dfI"(1), where τ₁τₙ="(_,_)", simplified])
            apply "cqt:2[lambda]"
           apply (rule "beta-C-meta"[THEN "→E"])
          apply "cqt:2[lambda]"
          using 2 3 "KBasic:3" "≡S"(2) "≡E"(2) by blast
      }
    qed
    AOT_thus ‹□∀x([[G]⁻ᵘ]x → □[[G]⁻ᵘ]x)› using ξ "→E" by blast
  next
    fix x
    AOT_assume ‹[[G]⁻ᵘ]x›
    AOT_hence ‹[λx [G]x & x ≠E u]x›
      by (auto intro: "F-u"[THEN "=dfE"(1), where τ₁τₙ="(_,_)", simplified]
               intro!: "cqt:2")
    AOT_hence ‹[G]x & x ≠E u›
      by (rule "β→C"(1))
    AOT_hence ‹[F]x & x ≠E u›
      using ζ "&I" "&E"(1) "&E"(2) "≡E"(1) "rule-ui:3" by blast
    AOT_hence ‹[λx [F]x & x ≠E u]x›
      by (auto intro!: "β←C"(1) "cqt:2")
    AOT_thus ‹[[F]⁻ᵘ]x›
      by (auto intro: "F-u"[THEN "=dfI"(1), where τ₁τₙ="(_,_)", simplified]
               intro!: "cqt:2")
  next
    fix x
    AOT_assume ‹[[F]⁻ᵘ]x›
    AOT_hence ‹[λx [F]x & x ≠E u]x›
      by (auto intro: "F-u"[THEN "=dfE"(1), where τ₁τₙ="(_,_)", simplified]
               intro!: "cqt:2")
    AOT_hence ‹[F]x & x ≠E u›
      by (rule "β→C"(1))
    AOT_hence ‹[G]x & x ≠E u›
      using ζ "&I" "&E"(1) "&E"(2) "≡E"(2) "rule-ui:3" by blast
    AOT_hence ‹[λx [G]x & x ≠E u]x›
      by (auto intro!: "β←C"(1) "cqt:2")
    AOT_thus ‹[[G]⁻ᵘ]x›
      by (auto intro: "F-u"[THEN "=dfI"(1), where τ₁τₙ="(_,_)", simplified]
               intro!: "cqt:2")
  qed
  AOT_hence ‹□Numbers(x,[G]⁻ᵘ)›
    using rigid_num_nec[unvarify F G, OF "F-u[den]", OF "F-u[den]", THEN "→E",
                        OF "&I", OF props[THEN "&E"(2)]] by blast
}
moreover AOT_have ‹□[G]u›
  using props[THEN "&E"(1), THEN "&E"(1), THEN ζ[THEN "∀E"(2), THEN "≡E"(2)]]
        ξ[THEN "qml:2"[axiom_inst, THEN "→E"], THEN "∀E"(2), THEN "→E"]
  by blast
ultimately AOT_have ‹□([G]u & Numbers(y,G) & Numbers(x,[G]⁻ᵘ))›
  by (metis "KBasic:3" "&I" "≡E"(2))
AOT_hence ‹∃u (□([G]u & Numbers(y,G) & Numbers(x,[G]⁻ᵘ)))›
```

```
    by (rule "Ordinary.∃I")
  AOT_hence ‹□∃u ([G]u & Numbers(y,G) & Numbers(x,[G]^-u))›
    using "Ordinary.res-var-bound-reas[Buridan]" "→E" by fast
  AOT_hence ‹∃F □∃u ([F]u & Numbers(y,F) & Numbers(x,[F]^-u))›
    by (rule "∃I")
  AOT_hence 0: ‹□∃F∃u ([F]u & Numbers(y,F) & Numbers(x,[F]^-u))›
    using Buridan "vdash-properties:10" by fast
  AOT_show ‹□[ℙ]xy›
    by (AOT_subst ‹[ℙ]xy› ‹∃F∃u ([F]u & Numbers(y,F) & Numbers(x,[F]^-u))›;
        simp add: "pred-thm:3" 0)
qed

AOT_theorem "pred-1-1:2": ‹Rigid(ℙ)›                                    (784.2)
  by (safe intro!: "df-rigid-rel:1"[THEN "≡dfI"] "pred-thm:2" "&I"
                   RN tuple_forall[THEN "≡dfI"];
      safe intro!: GEN "pred-1-1:1")

AOT_theorem "pred-1-1:3": ‹1-1(ℙ)›                                      (784.3)
proof (safe intro!: "df-1-1:1"[THEN "≡dfI"] "pred-thm:2" "&I" GEN "→I";
       frule "&E"(1); drule "&E"(2))
  fix x y z
  AOT_assume ‹[ℙ]xz›
  AOT_hence ‹∃F∃u ([F]u & Numbers(z,F) & Numbers(x,[F]^-u))›
    using "pred-thm:3"[THEN "≡E"(1)] by blast
  then AOT_obtain F where ‹∃u ([F]u & Numbers(z,F) & Numbers(x,[F]^-u))›
    using "∃E"[rotated] by blast
  then AOT_obtain u where u_prop: ‹[F]u & Numbers(z,F) & Numbers(x,[F]^-u)›
    using "Ordinary.∃E"[rotated] by meson
  AOT_assume ‹[ℙ]yz›
  AOT_hence ‹∃F∃u ([F]u & Numbers(z,F) & Numbers(y,[F]^-u))›
    using "pred-thm:3"[THEN "≡E"(1)] by blast
  then AOT_obtain G where ‹∃u ([G]u & Numbers(z,G) & Numbers(y,[G]^-u))›
    using "∃E"[rotated] by blast
  then AOT_obtain v where v_prop: ‹[G]v & Numbers(z,G) & Numbers(y,[G]^-v)›
    using "Ordinary.∃E"[rotated] by meson
  AOT_show ‹x = y›
  proof (rule "pre-Hume"[unvarify G H, OF "F-u[den]", OF "F-u[den]",
                         THEN "→E", OF "&I", THEN "≡E"(2)])
    AOT_show ‹Numbers(x, [F]^-u)›
      using u_prop "&E" by blast
  next
    AOT_show ‹Numbers(y, [G]^-v)›
      using v_prop "&E" by blast
  next
    AOT_have ‹F ≈E G›
      using u_prop[THEN "&E"(1), THEN "&E"(2)]
      using v_prop[THEN "&E"(1), THEN "&E"(2)]
      using "num-tran:2"[THEN "→E", OF "&I"] by blast
    AOT_thus ‹[F]^-u ≈E [G]^-v›
      using u_prop[THEN "&E"(1), THEN "&E"(1)]
      using v_prop[THEN "&E"(1), THEN "&E"(1)]
      using eqP'[THEN "→E", OF "&I", OF "&I"]
      by blast
  qed
qed

AOT_theorem "pred-1-1:4": ‹Rigid1-1(ℙ)›                                  (784.4)
  by (meson "≡dfI" "&I" "df-1-1:2" "pred-1-1:2" "pred-1-1:3")

AOT_theorem "assume-anc:1":                                              (785.1)
  ‹[ℙ]* = [λxy ∀F((∀z([ℙ]xz → [F]z) & Hereditary(F,ℙ)) → [F]y)]›
  apply (rule "=dfI"(1)[OF "ances-df"])
   apply "cqt:2[lambda]"
```

```
  apply (rule "=I"(1))
  by "cqt:2[lambda]"

AOT_theorem "assume-anc:2": ‹ℙ*↓›                                          (785.2)
  using "t=t-proper:1" "assume-anc:1" "vdash-properties:10" by blast

AOT_theorem "assume-anc:3":                                                (785.3)
  ‹[ℙ*]xy ≡ ∀F((∀z([ℙ]xz → [F]z) & ∀x'∀y'([ℙ]x'y' → ([F]x' → [F]y'))) → [F]y)›
proof -
  AOT_have prod_den: ‹⊢□ «(AOT_term_of_var x₁,AOT_term_of_var x₂)»↓›
    for x₁ x₂ :: ‹κ AOT_var›
    by (simp add: "&I" "ex:1:a" prod_denotesI "rule-ui:3")
  AOT_have den: ‹[λxy ∀F((∀z([ℙ]xz → [F]z) & Hereditary(F,ℙ)) → [F]y)]↓›
    by "cqt:2[lambda]"
  AOT_have 1: ‹[ℙ*]xy ≡ ∀F((∀z([ℙ]xz → [F]z) & Hereditary(F,ℙ)) → [F]y)›
    apply (rule "rule=E"[rotated, OF "assume-anc:1"[symmetric]])
    by (rule "beta-C-meta"[unvarify ν₁νₙ, OF prod_den, THEN "→E",
                           simplified, OF den, simplified])
  show ?thesis
    apply (AOT_subst (reverse) ‹∀x'∀y' ([ℙ]x'y' → ([F]x' → [F]y'))›
                               ‹Hereditary(F,ℙ)› for: F :: ‹<κ>›)
    using "hered:1"[THEN "≡Df", THEN "≡S"(1), OF "&I", OF "pred-thm:2",
                    OF "cqt:2[const_var]"[axiom_inst]] apply blast
    by (fact 1)
qed

AOT_theorem "no-pred-0:1": ‹¬∃x [ℙ]x 0›                                   (786.1)
proof(rule "raa-cor:2")
  AOT_assume ‹∃x [ℙ]x 0›
  then AOT_obtain a where ‹[ℙ]a 0›
    using "∃E"[rotated] by blast
  AOT_hence ‹∃F∃u ([F]u & Numbers(0, F) & Numbers(a, [F]⁻ᵘ))›
    using "pred-thm:3"[unvarify y, OF "zero:2", THEN "≡E"(1)] by blast
  then AOT_obtain F where ‹∃u ([F]u & Numbers(0, F) & Numbers(a, [F]⁻ᵘ))›
    using "∃E"[rotated] by blast
  then AOT_obtain u where ‹[F]u & Numbers(0, F) & Numbers(a, [F]⁻ᵘ)›
    using "Ordinary.∃E"[rotated] by meson
  AOT_hence ‹[F]u› and num0_F: ‹Numbers(0, F)›
    using "&E" "&I" by blast+
  AOT_hence ‹∃u [F]u›
    using "Ordinary.∃I" by fast
  moreover AOT_have ‹¬∃u [F]u›
    using num0_F  "≡E"(2) "OF:1" by blast
  ultimately AOT_show ‹p & ¬p› for p
    by (metis "raa-cor:3")
qed

AOT_theorem "no-pred-0:2": ‹¬∃x [ℙ*]x 0›                                  (786.2)
proof(rule "raa-cor:2")
  AOT_assume ‹∃x [ℙ*]x 0›
  then AOT_obtain a where ‹[ℙ*]a 0›
    using "∃E"[rotated] by blast
  AOT_hence ‹∃z [ℙ]z 0›
    using "anc-her:5"[unvarify R y, OF "zero:2",
                      OF "pred-thm:2", THEN "→E"] by auto
  AOT_thus ‹∃z [ℙ]z 0 & ¬∃z [ℙ]z 0›
    by (metis "no-pred-0:1" "raa-cor:3")
qed

AOT_theorem "no-pred-0:3": ‹¬[ℙ*]0 0›                                     (786.3)
  by (metis "existential:1" "no-pred-0:2" "reductio-aa:1" "zero:2")

AOT_theorem "assume1:1": ‹(=ℙ) = [λxy ∃z ([ℙ]xz & [ℙ]yz)]›                (787.1)
```

```
  apply (rule "=_dfI"(1)[OF "id-d-R"])
   apply "cqt:2[lambda]"
  apply (rule "=I"(1))
  by "cqt:2[lambda]"

AOT_theorem "assume1:2": ‹x =_ℙ y ≡ ∃z ([ℙ]xz & [ℙ]yz)›                    (787.2)
proof (rule "rule=E"[rotated, OF "assume1:1"[symmetric]])
  AOT_have prod_den: ‹⊢_□ «(AOT_term_of_var x_1,AOT_term_of_var x_2)»↓›
    for x_1 x_2 :: ‹κ AOT_var›
    by (simp add: "&I" "ex:1:a" prod_denotesI "rule-ui:3")
  AOT_have 1: ‹[λxy ∃z ([ℙ]xz & [ℙ]yz)]↓›
    by "cqt:2"
  AOT_show ‹[λxy ∃z ([ℙ]xz & [ℙ]yz)]xy ≡ ∃z ([ℙ]xz & [ℙ]yz)›
    using "beta-C-meta"[THEN "→E", OF 1, unvarify ν_1ν_n,
                        OF prod_den, simplified] by blast
qed

AOT_theorem "assume1:3": ‹[ℙ]^+ = [λxy [ℙ]^*xy ∨ x =_ℙ y]›                  (787.3)
  apply (rule "=_dfI"(1)[OF "w-ances-df"])
   apply (simp add: "w-ances-df[den1]")
  apply (rule "rule=E"[rotated, OF "assume1:1"[symmetric]])
  apply (rule "=_dfI"(1)[OF "id-d-R"])
   apply "cqt:2[lambda]"
  apply (rule "=I"(1))
  by "cqt:2[lambda]"

AOT_theorem "assume1:4": ‹[ℙ]^+↓›                                          (787.4)
  using "w-ances-df[den2]".

AOT_theorem "assume1:5": ‹[ℙ]^+xy ≡ [ℙ]^*xy ∨ x =_ℙ y›                     (787.5)
proof -
  AOT_have 0: ‹[λxy [ℙ]^*xy ∨ x =_ℙ y]↓› by "cqt:2"
  AOT_have prod_den: ‹⊢_□ «(AOT_term_of_var x_1, AOT_term_of_var x_2)»↓›
    for x_1 x_2 :: ‹κ AOT_var›
    by (simp add: "&I" "ex:1:a" prod_denotesI "rule-ui:3")
  show ?thesis
    apply (rule "rule=E"[rotated, OF "assume1:3"[symmetric]])
    using "beta-C-meta"[THEN "→E", OF 0, unvarify ν_1ν_n, OF prod_den, simplified]
    by (simp add: cond_case_prod_eta)
qed

AOT_define NaturalNumber :: ‹τ› (‹ℕ›)
  "nnumber:1": ‹ℕ =_df [λx [ℙ]^+0x]›                                        (788.1)

AOT_theorem "nnumber:2": ‹ℕ↓›                                              (788.2)
  by (rule "=_dfI"(2)[OF "nnumber:1"]; "cqt:2[lambda]")

AOT_theorem "nnumber:3": ‹[ℕ]x ≡ [ℙ]^+0x›                                   (788.3)
  apply (rule "=_dfI"(2)[OF "nnumber:1"])
   apply "cqt:2[lambda]"
  apply (rule "beta-C-meta"[THEN "→E"])
  by "cqt:2[lambda]"

AOT_theorem "0-n": ‹[ℕ]0›                                                  (789)
proof (safe intro!: "nnumber:3"[unvarify x, OF "zero:2", THEN "≡E"(2)]
    "assume1:5"[unvarify x y, OF "zero:2", OF "zero:2", THEN "≡E"(2)]
    "∨I"(2) "assume1:2"[unvarify x y, OF "zero:2", OF "zero:2", THEN "≡E"(2)])
  fix u
  AOT_have den: ‹[λx O!x & x =_E u]↓› by "cqt:2[lambda]"
  AOT_obtain a where a_prop: ‹Numbers(a, [λx O!x & x =_E u])›
    using "num:1"[unvarify G, OF den] "∃E"[rotated] by blast
  AOT_have ‹[ℙ]0a›
  proof (safe intro!: "pred-thm:3"[unvarify x, OF "zero:2", THEN "≡E"(2)]
```

```
                    "∃I"(1)[where τ=‹«[λx O!x & x =E u]»›]
                    "Ordinary.∃I"[where β=u] "&I" den
                    "OF:1"[unvarify F, OF "F-u[den]", unvarify F,
                           OF den, THEN "≡E"(1)])
    AOT_show ‹[λx [O!]x & x =E u]u›
      by (auto intro!: "β←C"(1) "cqt:2" "&I" "ord=Eequiv:1"[THEN "→E"]
                       Ordinary.ψ)
  next
    AOT_show ‹Numbers(a,[λx [O!]x & x =E u])›
      using a_prop.
  next
    AOT_show ‹¬∃v [[λx [O!]x & x =E u]^-u]v›
    proof(rule "raa-cor:2")
      AOT_assume ‹∃v [[λx [O!]x & x =E u]^-u]v›
      then AOT_obtain v where ‹[[λx [O!]x & x =E u]^-u]v›
        using "Ordinary.∃E"[rotated] "&E" by blast
      AOT_hence ‹[λz [λx [O!]x & x =E u]z & z ≠E u]v›
        apply (rule "F-u"[THEN "=dfE"(1), where τ1τn="(_,_)", simplified, rotated])
        by "cqt:2[lambda]"
      AOT_hence ‹[λx [O!]x & x =E u]v & v ≠E u›
        by (rule "β→C"(1))
      AOT_hence ‹v =E u› and ‹v ≠E u›
        using "β→C"(1) "&E" by blast+
      AOT_hence ‹v =E u & ¬(v =E u)›
        by (metis "≡E"(4) "reductio-aa:1" "thm-neg=E")
      AOT_thus ‹p & ¬p› for p
        by (metis "raa-cor:1")
    qed
  qed
  AOT_thus ‹∃z ([ℙ]0z & [ℙ]0z)›
    by (safe intro!: "&I" "∃I"(2)[where β=a])
qed

AOT_theorem "mod-col-num:1": ‹[ℕ]x → □[ℕ]x›                                    (790.1)
proof(rule "→I")
  AOT_have necON: ‹[λx □[ℕ]x]0›
    by (auto intro!: "β←C"(1) "cqt:2" simp: "zero:2" RN "0-n")
  AOT_have 1: ‹[λx □[ℕ]x]0 &
    ∀x∀y ([[ℙ]^+]0x & [[ℙ]^+]0y → ([ℙ]xy → ([λx □[ℕ]x]x → [λx □[ℕ]x]y))) →
    ∀x ([[ℙ]^+]0x → [λx □[ℕ]x]x)›
    by (auto intro!: "cqt:2"
             intro: "pre-ind"[unconstrain ℛ, unvarify β, OF "pred-thm:2",
                              THEN "→E", OF "pred-1-1:4", unvarify z, OF "zero:2",
                              unvarify F])
  AOT_have ‹∀x ([[ℙ]^+]0x → [λx □[ℕ]x]x)›
  proof (rule 1[THEN "→E"]; safe intro!: "&I" GEN "→I" necON;
         frule "&E"(1); drule "&E"(2))
    fix x y
    AOT_assume ‹[ℙ]xy›
    AOT_hence 0: ‹□[ℙ]xy›
      by (metis "pred-1-1:1" "→E")
    AOT_assume ‹[λx □[ℕ]x]x›
    AOT_hence ‹□[ℕ]x›
      by (rule "β→C"(1))
    AOT_hence ‹□([ℙ]xy & [ℕ]x)›
      by (metis "0" "KBasic:3" Adjunction "≡E"(2) "→E")
    moreover AOT_have ‹□([ℙ]xy & [ℕ]x) → □[ℕ]y›
    proof (rule RM; rule "→I"; frule "&E"(1); drule "&E"(2))
      AOT_modally_strict {
        AOT_assume 0: ‹[ℙ]xy›
        AOT_assume ‹[ℕ]x›
        AOT_hence 1: ‹[[ℙ]^+]0x›
          by (metis "≡E"(1) "nnumber:3")
```

```
        AOT_show ‹[ℕ]y›
          apply (rule "nnumber:3"[THEN "≡E"(2)])
          apply (rule "assume1:5"[unvarify x, OF "zero:2", THEN "≡E"(2)])
          apply (rule "∨I"(1))
          apply (rule "w-ances-her:3"[unconstrain ℛ, unvarify β, OF "pred-thm:2",
                                      THEN "→E", OF "pred-1-1:4", unvarify x,
                                      OF "zero:2", THEN "→E"])
          apply (rule "&I")
           apply (fact 1)
          by (fact 0)
      }
    qed
    ultimately AOT_have ‹□[ℕ]y›
      by (metis "→E")
    AOT_thus ‹[λx □[ℕ]x]y›
      by (auto intro!: "β←C"(1) "cqt:2")
  qed
  AOT_hence 0: ‹[[ℙ]⁺]0x → [λx □[ℕ]x]x›
    using "∀E"(2) by blast
  AOT_assume ‹[ℕ]x›
  AOT_hence ‹[[ℙ]⁺]0x›
    by (metis "≡E"(1) "nnumber:3")
  AOT_hence ‹[λx □[ℕ]x]x›
    using 0[THEN "→E"] by blast
  AOT_thus ‹□[ℕ]x›
    by (rule "β→C"(1))
qed

AOT_theorem "mod-col-num:2": ‹Rigid(ℕ)›                                    (790.2)
  by (safe intro!: "df-rigid-rel:1"[THEN "≡dfI"] "&I" RN GEN
                   "mod-col-num:1" "nnumber:2")

AOT_register_rigid_restricted_type
  Number: ‹[ℕ]κ›
proof
  AOT_modally_strict {
    AOT_show ‹∃x [ℕ]x›
      by (rule "∃I"(1)[where τ=‹«0»›]; simp add: "0-n" "zero:2")
  }
next
  AOT_modally_strict {
    AOT_show ‹[ℕ]κ → κ↓› for κ
      by (simp add: "→I" "cqt:5:a[1]"[axiom_inst, THEN "→E", THEN "&E"(2)])
  }
next
  AOT_modally_strict {
    AOT_show ‹∀x([ℕ]x → □[ℕ]x)›
      by (simp add: GEN "mod-col-num:1")
  }
qed
AOT_register_variable_names
  Number: m n k i j

AOT_theorem "0-pred": ‹¬∃n [ℙ]n 0›                                       (791)
proof (rule "raa-cor:2")
  AOT_assume ‹∃n [ℙ]n 0›
  then AOT_obtain n where ‹[ℙ]n 0›
    using "Number.∃E"[rotated] by meson
  AOT_hence ‹∃x [ℙ]x 0›
    using "&E" "∃I" by fast
  AOT_thus ‹∃x [ℙ]x 0 & ¬∃x [ℙ]x 0›
    using "no-pred-0:1" "&I" by auto
qed
```

```
AOT_theorem "no-same-succ":                                                    (792)
  ‹∀n∀m∀k([ℙ]nk & [ℙ]mk → n = m)›
proof(safe intro!: Number.GEN "→I")
  fix n m k
  AOT_assume ‹[ℙ]nk & [ℙ]mk›
  AOT_thus ‹n = m›
    by (safe intro!: "cqt:2[const_var]"[axiom_inst] "df-1-1:3"[
          unvarify R, OF "pred-thm:2",
          THEN "→E", OF "pred-1-1:4", THEN "qml:2"[axiom_inst, THEN "→E"],
          THEN "≡dfE"[OF "df-1-1:1"], THEN "&E"(2), THEN "∀E"(1), THEN "∀E"(1),
          THEN "∀E"(1)[where τ=‹AOT_term_of_var (Number.Rep k)›], THEN "→E"])
qed

AOT_theorem induction:                                                         (793)
  ‹∀F([F]0 & ∀n∀m([ℙ]nm → ([F]n → [F]m)) → ∀n[F]n)›
proof (safe intro!: GEN[where 'a=‹κ›] Number.GEN "&I" "→I";
       frule "&E"(1); drule "&E"(2))
  fix F n
  AOT_assume F0: ‹[F]0›
  AOT_assume 0: ‹∀n∀m([ℙ]nm → ([F]n → [F]m))›
  {
    fix x y
    AOT_assume ‹[[ℙ]⁺]0x & [[ℙ]⁺]0y›
    AOT_hence ‹[ℕ]x› and ‹[ℕ]y›
      using "&E" "≡E"(2) "nnumber:3" by blast+
    moreover AOT_assume ‹[ℙ]xy›
    moreover AOT_assume ‹[F]x›
    ultimately AOT_have ‹[F]y›
      using 0[THEN "∀E"(2), THEN "→E", THEN "∀E"(2), THEN "→E",
              THEN "→E", THEN "→E"] by blast
  } note 1 = this
  AOT_have 0: ‹[[ℙ]⁺]0n›
    by (metis "≡E"(1) "nnumber:3" Number.ψ)
  AOT_show ‹[F]n›
    apply (rule "pre-ind"[unconstrain ℛ, unvarify β, THEN "→E", OF "pred-thm:2",
                          OF "pred-1-1:4", unvarify z, OF "zero:2", THEN "→E",
                          THEN "∀E"(2), THEN "→E"];
           safe intro!: 0 "&I" GEN "→I" F0)
    using 1 by blast
qed

AOT_theorem "suc-num:1": ‹[ℙ]nx → [ℕ]x›                                      (794.1)
proof(rule "→I")
  AOT_have ‹[[ℙ]⁺]0 n›
    by (meson Number.ψ "≡E"(1) "nnumber:3")
  moreover AOT_assume ‹[ℙ]nx›
  ultimately AOT_have ‹[[ℙ]*]0 x›
    using "w-ances-her:3"[unconstrain ℛ, unvarify β, OF "pred-thm:2", THEN "→E",
                          OF "pred-1-1:4", unvarify x, OF "zero:2",
                          THEN "→E", OF "&I"]
    by blast
  AOT_hence ‹[[ℙ]⁺]0 x›
    using "assume1:5"[unvarify x, OF "zero:2", THEN "≡E"(2), OF "∨I"(1)]
    by blast
  AOT_thus ‹[ℕ]x›
    by (metis "≡E"(2) "nnumber:3")
qed

AOT_theorem "suc-num:2": ‹[[ℙ]*]nx → [ℕ]x›                                    (794.2)
proof(rule "→I")
  AOT_have ‹[[ℙ]⁺]0 n›
    using Number.ψ "≡E"(1) "nnumber:3" by blast
```

```
  AOT_assume ‹[[ℙ]*]n x›
  AOT_hence ‹∀F (∀z ([ℙ]nz → [F]z) & ∀x'∀y' ([ℙ]x'y' → ([F]x' → [F]y')) → [F]x)›
    using "assume-anc:3"[THEN "≡E"(1)] by blast
  AOT_hence ϑ: ‹∀z ([ℙ]nz → [ℕ]z) & ∀x'∀y' ([ℙ]x'y' → ([ℕ]x' → [ℕ]y')) → [ℕ]x›
    using "∀E"(1) "nnumber:2" by blast
  AOT_show ‹[ℕ]x›
  proof (safe intro!: ϑ[THEN "→E"] GEN "→I" "&I")
    AOT_show ‹[ℕ]z› if ‹[ℙ]nz› for z
      using Number.ψ "suc-num:1" that "→E" by blast
  next
    AOT_show ‹[ℕ]y› if ‹[ℙ]xy› and ‹[ℕ]x› for x y
      using "suc-num:1"[unconstrain n, THEN "→E"] that "→E" by blast
  qed
qed

AOT_theorem "suc-num:3": ‹[ℙ]+nx → [ℕ]x›                                    (794.3)
proof (rule "→I")
  AOT_assume ‹[ℙ]+nx›
  AOT_hence ‹[ℙ]*nx ∨ n =ℙ x›
    by (metis "assume1:5" "≡E"(1))
  moreover {
    AOT_assume ‹[ℙ]*nx›
    AOT_hence ‹[ℕ]x›
      by (metis "suc-num:2" "→E")
  }
  moreover {
    AOT_assume ‹n =ℙ x›
    AOT_hence ‹n = x›
      using "id-R-thm:3"[unconstrain ℛ, unvarify β, OF "pred-thm:2",
                         THEN "→E", OF "pred-1-1:4", THEN "→E"] by blast
    AOT_hence ‹[ℕ]x›
      by (metis "rule=E" Number.ψ)
  }
  ultimately AOT_show ‹[ℕ]x›
    by (metis "∨E"(3) "reductio-aa:1")
qed

AOT_theorem "pred-num": ‹[ℙ]xn → [ℕ]x›                                       (795)
proof (rule "→I")
  AOT_assume 0: ‹[ℙ]xn›
  AOT_have ‹[[ℙ]+]0 n›
    using Number.ψ "≡E"(1) "nnumber:3" by blast
  AOT_hence ‹[[ℙ]*]0 n ∨ 0 =ℙ n›
    using "assume1:5"[unvarify x, OF "zero:2"] by (metis "≡E"(1))
  moreover {
    AOT_assume ‹0 =ℙ n›
    AOT_hence ‹∃z ([ℙ]0z & [ℙ]nz)›
      using "assume1:2"[unvarify x, OF "zero:2", THEN "≡E"(1)] by blast
    then AOT_obtain a where ‹[ℙ]0a & [ℙ]na› using "∃E"[rotated] by blast
    AOT_hence ‹0 = n›
      using "pred-1-1:3"[THEN "df-1-1:1"[THEN "≡dfE"], THEN "&E"(2),
                         THEN "∀E"(1), OF "zero:2", THEN "∀E"(2),
                         THEN "∀E"(2), THEN "→E"] by blast
    AOT_hence ‹[ℙ]x 0›
      using 0 "rule=E" id_sym by fast
    AOT_hence ‹∃x [ℙ]x 0›
      by (rule "∃I")
    AOT_hence ‹∃x [ℙ]x 0 & ¬∃x [ℙ]x 0›
      by (metis "no-pred-0:1" "raa-cor:3")
  }
  ultimately AOT_have ‹[[ℙ]*]0n›
    by (metis "∨E"(3) "raa-cor:1")
  AOT_hence ‹∃z ([[ℙ]+]0z & [ℙ]zn)›
```

```
    using "w-ances-her:7"[unconstrain 𝓡, unvarify β, OF "pred-thm:2",
                          THEN "→E", OF "pred-1-1:4", unvarify x,
                          OF "zero:2", THEN "→E"] by blast
  then AOT_obtain b where b_prop: ‹[[ℙ]⁺]0b & [ℙ]bn›
    using "∃E"[rotated] by blast
  AOT_hence ‹[ℕ]b›
    by (metis "&E"(1) "≡E"(2) "nnumber:3")
  moreover AOT_have ‹x = b›
    using "pred-1-1:3"[THEN "df-1-1:1"[THEN "≡dfE"], THEN "&E"(2),
                       THEN "∀E"(2), THEN "∀E"(2), THEN "∀E"(2), THEN "→E",
                       OF "&I", OF 0, OF b_prop[THEN "&E"(2)]].
  ultimately AOT_show ‹[ℕ]x›
    using "rule=E" id_sym by fast
qed

AOT_theorem "nat-card": ‹[ℕ]x → NaturalCardinal(x)›                    (796)
proof(rule "→I")
  AOT_assume ‹[ℕ]x›
  AOT_hence ‹[[ℙ]⁺]0x›
    by (metis "≡E"(1) "nnumber:3")
  AOT_hence ‹[[ℙ]*]0x ∨ 0 =ℙ x›
    using "assume1:5"[unvarify x, OF "zero:2", THEN "≡E"(1)] by blast
  moreover {
    AOT_assume ‹[[ℙ]*]0x›
    then AOT_obtain a where ‹[ℙ]ax›
      using "anc-her:5"[unvarify R x, OF "zero:2", OF "pred-thm:2", THEN "→E"]
            "∃E"[rotated] by blast
    AOT_hence ‹∃F∃u ([F]u & Numbers(x,F) & Numbers(a,[F]⁻ᵘ))›
      using "pred-thm:3"[THEN "≡E"(1)] by blast
    then AOT_obtain F where ‹∃u ([F]u & Numbers(x,F) & Numbers(a,[F]⁻ᵘ))›
      using "∃E"[rotated] by blast
    then AOT_obtain u where ‹[F]u & Numbers(x,F) & Numbers(a,[F]⁻ᵘ)›
      using "Ordinary.∃E"[rotated] by meson
    AOT_hence ‹NaturalCardinal(x)›
      using "eq-num:6"[THEN "→E"] "&E" by blast
  }
  moreover {
    AOT_assume ‹0 =ℙ x›
    AOT_hence ‹0 = x›
      using "id-R-thm:3"[unconstrain 𝓡, unvarify β, OF "pred-thm:2",
                         THEN "→E", OF "pred-1-1:4", unvarify x,
                         OF "zero:2", THEN "→E"] by blast
    AOT_hence ‹NaturalCardinal(x)›
      by (metis "rule=E" "zero-card")
  }
  ultimately AOT_show ‹NaturalCardinal(x)›
    by (metis "∨E"(2) "raa-cor:1")
qed

AOT_theorem "pred-func:1": ‹[ℙ]xy & [ℙ]xz → y = z›                       (797.1)
proof (rule "→I"; frule "&E"(1); drule "&E"(2))
  AOT_assume ‹[ℙ]xy›
  AOT_hence ‹∃F∃u ([F]u & Numbers(y,F) & Numbers(x,[F]⁻ᵘ))›
    using "pred-thm:3"[THEN "≡E"(1)] by blast
  then AOT_obtain F where ‹∃u ([F]u & Numbers(y,F) & Numbers(x,[F]⁻ᵘ))›
    using "∃E"[rotated] by blast
  then AOT_obtain a where
            Oa: ‹O!a›
    and a_prop: ‹[F]a & Numbers(y,F) & Numbers(x,[F]⁻ᵃ)›
    using "∃E"[rotated] "&E" by blast
  AOT_assume ‹[ℙ]xz›
  AOT_hence ‹∃F∃u ([F]u & Numbers(z,F) & Numbers(x,[F]⁻ᵘ))›
    using "pred-thm:3"[THEN "≡E"(1)] by blast
```

```
  then AOT_obtain G where ‹∃u ([G]u & Numbers(z,G) & Numbers(x,[G]⁻ᵘ))›
    using "∃E"[rotated] by blast
  then AOT_obtain b where Ob: ‹O!b›
                and b_prop: ‹[G]b & Numbers(z,G) & Numbers(x,[G]⁻ᵇ)›
    using "∃E"[rotated] "&E" by blast
  AOT_have ‹[F]⁻ᵃ ≈E [G]⁻ᵇ›
    using "num-tran:2"[unvarify G H, OF "F-u[den]", OF "F-u[den]",
                       THEN "→E", OF "&I", OF a_prop[THEN "&E"(2)],
                       OF b_prop[THEN "&E"(2)]].
  AOT_hence ‹F ≈E G›
    using "P'-eq"[unconstrain u, THEN "→E", OF Oa, unconstrain v, THEN "→E",
                  OF Ob, THEN "→E", OF "&I", OF "&I"]
          a_prop[THEN "&E"(1), THEN "&E"(1)]
          b_prop[THEN "&E"(1), THEN "&E"(1)] by blast
  AOT_thus ‹y = z›
    using "pre-Hume"[THEN "→E", THEN "≡E"(2), OF "&I",
                     OF a_prop[THEN "&E"(1), THEN "&E"(2)],
                     OF b_prop[THEN "&E"(1), THEN "&E"(2)]]
    by blast
qed

AOT_theorem "pred-func:2": ‹[ℙ]nm & [ℙ]nk → m = k›                          (797.2)
  using "pred-func:1".

AOT_theorem being_number_of_den: ‹[λx x = #G]↓›
proof (rule "safe-ext"[axiom_inst, THEN "→E"]; safe intro!: "&I" GEN RN)
  AOT_show ‹[λx Numbers(x,[λz 𝒜[G]z])]↓›
    by (rule numbers_prop_den[unvarify G]) "cqt:2[lambda]"
next
  AOT_modally_strict {
    AOT_show ‹Numbers(x,[λz 𝒜[G]z]) ≡ x = #G› for x
      using "eq-num:2".
  }
qed

axiomatization ω_nat :: ‹ω ⇒ nat› where ω_nat: ‹surj ω_nat›
text‹Unfortunately, since the axiom requires the type @{typ ω}
     to have an infinite domain, @{command nitpick} can only find a potential model
     and no genuine model.
     However, since we could trivially choose @{typ ω} as a copy of @{typ nat},
     we can still be assured that above axiom is consistent.›
lemma ‹True› nitpick[satisfy, user_axioms, card nat=1, expect = potential] ..

AOT_axiom "modal-axiom":                                                      (798)
  ‹∃x([ℕ]x & x = #G) → ◇∃y([E!]y & ∀u (𝒜[G]u → u ≠E y))›
proof(rule AOT_model_axiomI) AOT_modally_strict {
  text‹The actual extension on the ordinary objects of a property is the
       set of ordinary urelements that exemplifies the property in the
       designated actual world.›
  define act_ωext :: ‹<κ> ⇒ ω set› where
    ‹act_ωext ≡ λ Π . {x :: ω . [w₀ ⊨ [Π]«ωκ x»]}›
  text‹Encoding a property with infinite actual extension on the ordinary objects
       denotes a property by extended relation comprehension.›
  AOT_have enc_finite_act_ωext_den:
    ‹⊢□ [λx ∃F(¬«ε₀ w. finite (act_ωext F)» & x[F])]↓›
  proof(safe intro!: Comprehension_1[THEN "→E"] RN GEN "→I")
    AOT_modally_strict {
      fix F G
      AOT_assume ‹□G ≡E F›
      AOT_hence ‹𝒜G ≡E F›
        using "nec-imp-act"[THEN "→E"] by blast
      AOT_hence ‹𝒜(G↓ & F↓ & ∀u([G]u ≡ [F]u))›
        by (AOT_subst_def (reverse) eqE)
```

```
      hence ‹[w₀ ⊨ [G]«ωκ x»] = [w₀ ⊨ [F]«ωκ x»]› for x
        by (auto dest!: "∀E"(1) "→E"
                 simp: AOT_model_denotes_κ_def AOT_sem_denotes AOT_sem_conj
                       AOT_model_ωκ_ordinary AOT_sem_act AOT_sem_equiv)
      AOT_thus ‹¬«ε₀ w. finite (act_ωext (AOT_term_of_var F))» ≡
                ¬«ε₀ w. finite (act_ωext (AOT_term_of_var G))»›
        by (simp add: AOT_sem_not AOT_sem_equiv act_ωext_def
                      AOT_model_proposition_choice_simp)
    }
  qed
  text‹By coexistence, encoding only properties with finite actual extension
       on the ordinary objects denotes.›
  AOT_have ‹[λx ∀F(x[F] → «ε₀ w. finite (act_ωext F)»)]↓›
  proof(rule "safe-ext"[axiom_inst, THEN "→E"]; safe intro!: "&I" RN GEN)
    AOT_show ‹[λx ¬[λx ∃F(¬«ε₀ w. finite (act_ωext F)» & x[F])]x]↓›
      by "cqt:2"
  next
    AOT_modally_strict {
      fix x
      AOT_show ‹¬[λx ∃F (¬«ε₀ w. finite (act_ωext F)» & x[F])]x ≡
                ∀F(x[F] → «ε₀ w. finite (act_ωext F)»)›
        by (AOT_subst ‹[λx ∃F (¬«ε₀ w. finite (act_ωext F)» & x[F])]x›
                      ‹∃F (¬«ε₀ w. finite (act_ωext F)» & x[F])›;
            (rule "beta-C-meta"[THEN "→E"])?)
           (auto simp: enc_finite_act_ωext_den AOT_sem_equiv AOT_sem_not
                       AOT_sem_forall AOT_sem_imp AOT_sem_conj AOT_sem_exists)
    }
  qed
  text‹We show by induction that any property encoded by a natural number
       has a finite actual extension on the ordinary objects.›
  AOT_hence ‹[λx ∀F(x[F] → «ε₀ w. finite (act_ωext F)»)]n› for n
  proof(rule induction[THEN "∀E"(1), THEN "→E", THEN "Number.∀E"];
        safe intro!: "&I" "Number.GEN" "β←C" "zero:2" "→I" "cqt:2"
             dest!: "β→C")
    AOT_show ‹∀F(0[F] → «ε₀ w. finite (act_ωext F)»)›
    proof(safe intro!: GEN "→I")
      fix F
      AOT_assume ‹0[F]›
      AOT_actually {
        AOT_hence ‹¬∃u [F]u›
          using "zero=:2" "intro-elim:3:a" AOT_sem_enc_nec by blast
        AOT_hence ‹∀x ¬(O!x & [F]x)›
          using "cqt-further:4" "vdash-properties:10" by blast
        hence ‹¬([w₀ ⊨ [F]«ωκ x»])› for x
          by (auto dest!: "∀E"(1)[where τ=‹ωκ x›]
                   simp: AOT_sem_not AOT_sem_conj AOT_model_ωκ_ordinary
                         "russell-axiom[exe,1].ψ_denotes_asm")
      }
      AOT_thus ‹«ε₀ w. finite (act_ωext (AOT_term_of_var F))»›
        by (auto simp: AOT_model_proposition_choice_simp act_ωext_def)
    qed
  next
    fix n m
    AOT_assume ‹[ℙ]nm›
    AOT_hence ‹∃F∃u ([F]u & Numbers(m,F) & Numbers(n,[F]⁻ᵘ))›
      using "pred-thm:3"[THEN "≡E"(1)] by blast
    then AOT_obtain G where ‹∃u ([G]u & Numbers(m,G) & Numbers(n,[G]⁻ᵘ))›
      using "∃E"[rotated] by blast
    then AOT_obtain u where 0: ‹[G]u & Numbers(m,G) & Numbers(n,[G]⁻ᵘ)›
      using "Ordinary.∃E"[rotated] by meson

    AOT_assume n_prop: ‹∀F(n[F] → «ε₀ w. finite (act_ωext F)»)›
    AOT_show ‹∀F(m[F] → «ε₀ w. finite (act_ωext F)»)›
```

```
    proof(safe intro!: GEN "→I")
      fix F
      AOT_assume ‹m[F]›
      AOT_hence 1: ‹[λx 𝒜[F]x] ≈E G›
        using 0[THEN "&E"(1), THEN "&E"(2), THEN numbers[THEN "≡dfE"],
                THEN "&E"(2), THEN "∀E"(2), THEN "≡E"(1)] by auto
      AOT_show ‹«ε_o w. finite (act_ωext (AOT_term_of_var F))»›
      proof(rule "raa-cor:1")
        AOT_assume ‹¬«ε_o w. finite (act_ωext (AOT_term_of_var F))»›
        hence inf: ‹infinite (act_ωext (AOT_term_of_var F))›
          by (auto simp: AOT_sem_not AOT_model_proposition_choice_simp)
        then AOT_obtain v where act_F_v: ‹𝒜[F]v›
          unfolding AOT_sem_act act_ωext_def
          by (metis AOT_term_of_var_cases AOT_model_ωκ_ordinary
                    AOT_model_denotes_κ_def Ordinary.Rep_cases κ.disc(7)
                    mem_Collect_eq not_finite_existsD)
        AOT_hence ‹[λx 𝒜[F]x]v›
          by (safe intro!: "β←C" "cqt:2")
        AOT_hence ‹[λx 𝒜[F]x]^-v ≈E [G]^-u›
          by (safe intro!: eqP'[unvarify F, THEN "→E"] "&I" "cqt:2" 1
                  0[THEN "&E"(1), THEN "&E"(1)])
        moreover AOT_have ‹[λx 𝒜[F]x]^-v ≈E [λx 𝒜[λy [F]y & y ≠E v]x]›
        proof(safe intro!: "apE-eqE:1"[unvarify F G, THEN "→E"] "cqt:2"
                           "F-u[den]"[unvarify F] eqE[THEN "≡dfI"] "&I"
                           Ordinary.GEN)
          fix u
          AOT_have ‹[λx [λx 𝒜[F]x]x & x ≠E v]u ≡ [λx 𝒜[F]x]u & u ≠E v›
            by (safe intro!: "beta-C-meta"[THEN "→E"] "cqt:2")
          also AOT_have ‹[λx 𝒜[F]x]u & u ≠E v ≡ 𝒜[F]u & u ≠E v›
            by (AOT_subst ‹[λx 𝒜[F]x]u› ‹𝒜[F]u›)
               (safe intro!: "beta-C-meta"[THEN "→E"] "cqt:2"
                             "oth-class-taut:3:a")
          also AOT_have ‹𝒜[F]u & u ≠E v ≡ 𝒜([F]u & u ≠E v)›
            using "id-act2:2" AOT_sem_conj AOT_sem_equiv AOT_sem_act by auto
          also AOT_have ‹𝒜([F]u & u ≠E v) ≡ 𝒜[λy [F]y & y ≠E v]u›
            by (AOT_subst ‹[λy [F]y & y ≠E v]u› ‹[F]u & u ≠E v›)
               (safe intro!: "beta-C-meta"[THEN "→E"] "cqt:2"
                             "oth-class-taut:3:a")
          also AOT_have ‹𝒜[λy [F]y & y ≠E v]u ≡ [λx 𝒜[λy [F]y & y ≠E v]x]u›
            by (safe intro!: "beta-C-meta"[THEN "→E", symmetric] "cqt:2")
          finally AOT_show ‹[[λx 𝒜[F]x]^-v]u ≡ [λx 𝒜[λy [F]y & y ≠E v]x]u›
            by (auto intro!: "cqt:2"
                     intro: "rule-id-df:2:b"[OF "F-u", where τ1τn=‹(_,_)›, simplified])
        qed
        ultimately AOT_have ‹[λx 𝒜[λy [F]y & y ≠E v]x] ≈E [G]^-u›
          using "eq-part:2[terms]" "eq-part:3[terms]" "→E" by blast
        AOT_hence ‹n[λy [F]y & y ≠E v]›
          by (safe intro!: 0[THEN "&E"(2), THEN numbers[THEN "≡dfE"],
                THEN "&E"(2), THEN "∀E"(1), THEN "≡E"(2)] "cqt:2")
        hence finite: ‹finite (act_ωext «[λy [F]y & y ≠E v]»)›
          by (safe intro!: n_prop[THEN "∀E"(1), THEN "→E",
                                  simplified AOT_model_proposition_choice_simp]
                           "cqt:2")
        obtain y where y_def: ‹ωκ y = AOT_term_of_var (Ordinary.Rep v)›
          by (metis AOT_model_ordinary_ωκ Ordinary.restricted_var_condition)
        AOT_actually {
          fix x
          AOT_assume ‹[λy [F]y & y ≠E v]«ωκ x»›
          AOT_hence ‹[F]«ωκ x»›
            by (auto dest!: "β→C" "&E"(1))
        }
        moreover AOT_actually {
          AOT_have ‹[F]«ωκ y»›
```

```
          unfolding y_def using act_F_v AOT_sem_act by blast
      }
      moreover AOT_actually {
        fix x
        assume noteq: ‹x ≠ y›
        AOT_assume ‹[F]«ωκ x»›
        moreover AOT_have ωκ_x_den: ‹«ωκ x»↓›
          using AOT_sem_exe calculation by blast
        moreover {
          AOT_have ‹¬(«ωκ x» =E v)›
          proof(rule "raa-cor:2")
            AOT_assume ‹«ωκ x» =E v›
            AOT_hence ‹«ωκ x» = v›
              using "=E-simple:2"[unvarify x, THEN "→E", OF ωκ_x_den]
              by blast
            hence ‹ωκ x = ωκ y›
              unfolding y_def AOT_sem_eq
              by meson
            hence ‹x = y›
              by blast
            AOT_thus ‹p & ¬p› for p using noteq by blast
          qed
          AOT_hence ‹«ωκ x» ≠E v›
            by (safe intro!: "thm-neg=E"[unvarify x, THEN "≡E"(2)] ωκ_x_den)
        }
        ultimately AOT_have ‹[λy [F]y & y ≠E v]«ωκ x»›
          by (auto intro!: "β←C" "cqt:2" "&I")
      }
      ultimately have ‹(insert y (act_ωext «[λy [F]y & y ≠E v]»)) =
                       (act_ωext (AOT_term_of_var F))›
        unfolding act_ωext_def
        by auto
      hence ‹finite (act_ωext (AOT_term_of_var F))›
        using finite finite.insertI by metis
      AOT_thus ‹p & ¬p› for p
        using inf by blast
    qed
  qed
qed
AOT_hence nat_enc_finite: ‹∀F(n[F] → «εo w. finite (act_ωext F)»)› for n
  using "β→C"(1) by blast

text‹The main proof can now generate a witness, since we required
     the domain of ordinary objects to be infinite.›
AOT_show ‹∃x ([ℕ]x & x = #G) → ◇∃y (E!y & ∀u (𝒜[G]u → u ≠E y))›
proof(safe intro!: "→I")
  AOT_assume ‹∃x ([ℕ]x & x = #G)›
  then AOT_obtain n where ‹n = #G›
    using "Number.∃E"[rotated] by meson
  AOT_hence ‹Numbers(n,[λx 𝒜[G]x])›
    using "eq-num:3" "rule=E" id_sym by fast
  AOT_hence ‹n[G]›
    by (auto intro!: numbers[THEN "≡dfE", THEN "&E"(2),
                             THEN "∀E"(2), THEN "≡E"(2)]
                     "eq-part:1"[unvarify F] "cqt:2")
  AOT_hence ‹«εo w. finite (act_ωext (AOT_term_of_var G))»›
    using nat_enc_finite[THEN "∀E"(2), THEN "→E"] by blast
  hence finite: ‹finite (act_ωext (AOT_term_of_var G))›
    by (auto simp: AOT_model_proposition_choice_simp)
  AOT_have ‹∃u ¬𝒜[G]u›
  proof(rule "raa-cor:1")
    AOT_assume ‹¬∃u ¬𝒜[G]u›
    AOT_hence ‹∀x ¬(O!x & ¬𝒜[G]x)›
```

```
        by (metis "cqt-further:4" "→E")
      AOT_hence ‹𝒜[G]x› if ‹O!x› for x
        using "∀E"(2) AOT_sem_conj AOT_sem_not that by blast
      hence ‹[w0 ⊨ [G]«ωκ x»]› for x
        by (metis AOT_term_of_var_cases AOT_model_ωκ_ordinary
             AOT_model_denotes_κ_def AOT_sem_act κ.disc(7))
      hence ‹(act_ωext (AOT_term_of_var G)) = UNIV›
        unfolding act_ωext_def by auto
      moreover have ‹infinite (UNIV::ω set)›
        by (metis ω_nat finite_imageI infinite_UNIV_char_0)
      ultimately have ‹infinite (act_ωext (AOT_term_of_var G))›
        by simp
      AOT_thus ‹p & ¬p› for p using finite by blast
    qed
    then AOT_obtain x where x_prop: ‹O!x & ¬𝒜[G]x›
      using "∃E"[rotated] by blast
    AOT_hence ‹◇E!x›
      by (metis "betaC:1:a" "con-dis-i-e:2:a" AOT_sem_ordinary)
    moreover AOT_have ‹□∀u (𝒜[G]u → u ≠E x)›
    proof(safe intro!: RN GEN "→I")
      AOT_modally_strict {
        fix y
        AOT_assume ‹O!y›
        AOT_assume 0: ‹𝒜[G]y›
        AOT_show ‹y ≠E x›
        proof (safe intro!: "thm-neg=E"[THEN "≡E"(2)] "raa-cor:2")
          AOT_assume ‹y =E x›
          AOT_hence ‹y = x›
            by (metis "=E-simple:2" "vdash-properties:10")
          hence ‹y = x›
            by (simp add: AOT_sem_eq AOT_term_of_var_inject)
          AOT_hence ‹¬𝒜[G]y›
            using x_prop "&E" AOT_sem_not AOT_sem_act by metis
          AOT_thus ‹𝒜[G]y & ¬𝒜[G]y›
            using 0 "&I" by blast
        qed
      }
    qed
    ultimately AOT_have ‹◇(∀u (𝒜[G]u → u ≠E x) & E!x)›
      using "KBasic:16"[THEN "→E", OF "&I"] by blast
    AOT_hence ‹◇(E!x & ∀u (𝒜[G]u → u ≠E x))›
      by (AOT_subst ‹E!x & ∀u (𝒜[G]u → u ≠E x)› ‹∀u (𝒜[G]u → u ≠E x) & E!x›)
         (auto simp: "oth-class-taut:2:a")
    AOT_hence ‹∃y ◇(E!y & ∀u (𝒜[G]u → u ≠E y))›
      using "∃I" by fast
    AOT_thus ‹◇∃y (E!y & ∀u (𝒜[G]u → u ≠E y))›
      using "CBF◇"[THEN "→E"] by fast
  qed
} qed

AOT_theorem "modal-lemma":                                            (800)
  ‹◇∀u(𝒜[G]u → u ≠E v) → ∀u(𝒜[G]u → u ≠E v)›
proof(safe intro!: "→I" Ordinary.GEN)
  AOT_modally_strict {
    fix u
    AOT_assume act_Gu: ‹𝒜[G]u›
    AOT_have ‹∀u (𝒜[G]u → u ≠E v) → u ≠E v›
    proof(rule "→I")
      AOT_assume ‹∀u (𝒜[G]u → u ≠E v)›
      AOT_hence ‹𝒜[G]u → u ≠E v›
        using "Ordinary.∀E" by fast
      AOT_thus ‹u ≠E v›
        using act_Gu "→E" by blast
```

```
    qed
  } note 0 = this
  AOT_have ϑ: ‹□(∀u (𝓐[G]u → u ≠E v) → u ≠E v)› if ‹□𝓐[G]u› for u
  proof -
    AOT_have ‹□𝓐[G]u → □(∀u (𝓐[G]u → u ≠E v) → u ≠E v)›
      apply (rule RM) using 0 "&E" "→I" by blast
    thus ?thesis using that "→E" by blast
  qed
  fix u
  AOT_assume 1: ‹◇∀u(𝓐[G]u → u ≠E v)›
  AOT_assume ‹𝓐[G]u›
  AOT_hence ‹□𝓐[G]u›
    by (metis "Act-Basic:6" "≡E"(1))
  AOT_hence ‹□(∀u (𝓐[G]u → u ≠E v) → u ≠E v)›
    using Ordinary.ψ ϑ by blast
  AOT_hence ‹◇u ≠E v›
    using 1 "K◇"[THEN "→E", THEN "→E"] by blast
  AOT_thus ‹u ≠E v›
    by (metis "id-nec4:2" "≡E"(1))
qed

AOT_theorem "th-succ": ‹∀n∃!m [ℙ]nm›                                     (801)
proof(safe intro!: Number.GEN "→I" "uniqueness:1"[THEN "≡dfI"])
  fix n
  AOT_have ‹NaturalCardinal(n)›
    by (metis "nat-card" Number.ψ "→E")
  AOT_hence ‹∃G(n = #G)›
    by (metis "≡dfE" card)
  then AOT_obtain G where n_num_G: ‹n = #G›
    using "∃E"[rotated] by blast
  AOT_hence ‹∃n (n = #G)›
    by (rule "Number.∃I")
  AOT_hence ‹◇∃y ([E!]y & ∀u(𝓐[G]u → u ≠E y))›
    using "modal-axiom"[axiom_inst, THEN "→E"] by blast
  AOT_hence ‹∃y ◇([E!]y & ∀u(𝓐[G]u → u ≠E y))›
    using "BF◇"[THEN "→E"] by auto
  then AOT_obtain y where ‹◇([E!]y & ∀u(𝓐[G]u → u ≠E y))›
    using "∃E"[rotated] by blast
  AOT_hence ‹◇E!y› and 2: ‹◇∀u(𝓐[G]u → u ≠E y)›
    using "KBasic2:3" "&E" "→E" by blast+
  AOT_hence Oy: ‹O!y›
    by (auto intro!: "β←C"(1) "cqt:2" intro: AOT_ordinary[THEN "=dfI"(2)])
  AOT_have 0: ‹∀u(𝓐[G]u → u ≠E y)›
    using 2 "modal-lemma"[unconstrain v, THEN "→E", OF Oy, THEN "→E"] by simp
  AOT_have 1: ‹[λx 𝓐[G]x ∨ x =E y]↓›
    by "cqt:2"
  AOT_obtain b where b_prop: ‹Numbers(b, [λx 𝓐[G]x ∨ x =E y])›
    using "num:1"[unvarify G, OF 1] "∃E"[rotated] by blast
  AOT_have Pnb: ‹[ℙ]nb›
  proof(safe intro!: "pred-thm:3"[THEN "≡E"(2)]
                     "∃I"(1)[where τ=‹«[λx 𝓐[G]x ∨ x =E y]»›]
                     1 "∃I"(2)[where β=y] "&I" Oy b_prop)
    AOT_show ‹[λx 𝓐[G]x ∨ x =E y]y›
      by (auto intro!: "β←C"(1) "cqt:2" "∨I"(2)
                       "ord=Eequiv:1"[THEN "→E", OF Oy])
  next
    AOT_have equinum: ‹[λx 𝓐[G]x ∨ x =E y]⁻ʸ ≈E [λx 𝓐[G]x]›
    proof(rule "apE-eqE:1"[unvarify F G, THEN "→E"];
          ("cqt:2[lambda]" | rule "F-u[den]"[unvarify F]; "cqt:2[lambda]")?)
      AOT_show ‹[λx 𝓐[G]x ∨ x =E y]⁻ʸ ≡E [λx 𝓐[G]x]›
      proof (safe intro!: eqE[THEN "≡dfI"] "&I" "F-u[den]"[unvarify F]
                          Ordinary.GEN "→I"; "cqt:2"?)
        fix u
```

```
        AOT_have ‹[[λx 𝒜[G]x ∨ [(=E)]xy]^-y]u ≡ ([λx 𝒜[G]x ∨ x =E y]u) & u ≠E y›
          apply (rule "F-u"[THEN "=dfI"(1)[where τ1τn=‹(_,_)›], simplified]; "cqt:2"?)
          by (rule "beta-C-cor:2"[THEN "→E", THEN "∀E"(2)]; "cqt:2")
        also AOT_have ‹... ≡  (𝒜[G]u ∨ u =E y) & u ≠E y›
          apply (AOT_subst ‹[λx 𝒜[G]x ∨ [(=E)]xy]u› ‹𝒜[G]u ∨ u =E y›)
           apply (rule "beta-C-cor:2"[THEN "→E", THEN "∀E"(2)]; "cqt:2")
          using "oth-class-taut:3:a" by blast
        also AOT_have ‹... ≡ 𝒜[G]u›
        proof(safe intro!: "≡I" "→I")
          AOT_assume ‹(𝒜[G]u ∨ u =E y) & u ≠E y›
          AOT_thus ‹𝒜[G]u›
            by (metis "&E"(1) "&E"(2) "∨E"(3) "≡E"(1) "thm-neg=E")
        next
          AOT_assume ‹𝒜[G]u›
          AOT_hence ‹u ≠E y› and ‹𝒜[G]u ∨ u =E y›
            using 0[THEN "∀E"(2), THEN "→E", OF Ordinary.ψ, THEN "→E"]
                  "∨I" by blast+
          AOT_thus ‹(𝒜[G]u ∨ u =E y) & u ≠E y›
            using "&I" by simp
        qed
        also AOT_have ‹... ≡ [λx 𝒜[G]x]u›
          by (rule "beta-C-cor:2"[THEN "→E", THEN "∀E"(2), symmetric]; "cqt:2")
        finally AOT_show ‹[[λx 𝒜[G]x ∨ [(=E)]xy]^-y]u ≡ [λx 𝒜[G]x]u›.
      qed
    qed
    AOT_have 2: ‹[λx 𝒜[G]x]↓› by "cqt:2[lambda]"
    AOT_show ‹Numbers(n,[λx 𝒜[G]x ∨ x =E y]^-y)›
      using "num-tran:1"[unvarify G H, OF 2, OF "F-u[den]"[unvarify F, OF 1],
                         THEN "→E", OF equinum, THEN "≡E"(2),
                         OF "eq-num:2"[THEN "≡E"(2), OF n_num_G]].
  qed
  AOT_show ‹∃α ([ℕ]α & [ℙ]nα & ∀β ([ℕ]β & [ℙ]nβ → β = α))›
  proof(safe intro!: "∃I"(2)[where β=b] "&I" Pnb "→I" GEN)
    AOT_show ‹[ℕ]b› using "suc-num:1"[THEN "→E", OF Pnb].
  next
    fix y
    AOT_assume 0: ‹[ℕ]y & [ℙ]ny›
    AOT_show ‹y = b›
      apply (rule "pred-func:1"[THEN "→E"])
      using 0[THEN "&E"(2)] Pnb "&I" by blast
  qed
qed

(* Note the use of a bold ’. *)
AOT_define Successor :: ‹τ ⇒ κs› (‹_’› [100] 100)
  "def-suc": ‹n’ =df ιm([ℙ]nm)›                                              (804)

text‹Note: not explicitly in PLM›
AOT_theorem "def-suc[den1]": ‹ιm([ℙ]nm)↓›                                   (804)
  using "A-Exists:2" "RA[2]" "≡E"(2) "th-succ"[THEN "Number.∀E"] by blast
text‹Note: not explicitly in PLM›
AOT_theorem "def-suc[den2]": shows ‹n’↓›                                    (804)
  by (rule "def-suc"[THEN "=dfI"(1)])
     (auto simp: "def-suc[den1]")

(* TODO: not in PLM *)
AOT_theorem suc_eq_desc: ‹n’ = ιm([ℙ]nm)›
  by (rule "def-suc"[THEN "=dfI"(1)])
     (auto simp: "def-suc[den1]" "rule=I:1")

AOT_theorem "suc-fact": ‹n = m → n’ = m’›                                    (805)
proof (rule "→I")
  AOT_assume 0: ‹n = m›
```

```
  AOT_show ‹n' = m'›
    apply (rule "rule=E"[rotated, OF 0])
    by (rule "=I"(1)[OF "def-suc[den2]"])
qed

AOT_theorem "ind-gnd": ‹m = 0 ∨ ∃n(m = n')›                                    (806)
proof -
  AOT_have ‹[[ℙ]+]0m›
    using Number.ψ "≡E"(1) "nnumber:3" by blast
  AOT_hence ‹[[ℙ]*]0m ∨ 0 =ℙ m›
    using "assume1:5"[unvarify x, OF "zero:2", THEN "≡E"(1)] by blast
  moreover {
    AOT_assume ‹[[ℙ]*]0m›
    AOT_hence ‹∃z ([[ℙ]+]0z & [ℙ]zm)›
      using "w-ances-her:7"[unconstrain ℛ, unvarify β x, OF "zero:2",
                            OF "pred-thm:2", THEN "→E", OF "pred-1-1:4",
                            THEN "→E"]
      by blast
    then AOT_obtain z where ϑ: ‹[[ℙ]+]0z› and ξ: ‹[ℙ]zm›
      using "&E" "∃E"[rotated] by blast
    AOT_have Nz: ‹[ℕ]z›
      using ϑ "≡E"(2) "nnumber:3" by blast
    moreover AOT_have ‹m = z'›
    proof (rule "def-suc"[THEN "=dfI"(1)];
           safe intro!: "def-suc[den1]"[unconstrain n, THEN "→E", OF Nz]
                        "nec-hintikka-scheme"[THEN "≡E"(2)] "&I"
                        GEN "→I" "Act-Basic:2"[THEN "≡E"(2)])
      AOT_show ‹𝓐[ℕ]m› using Number.ψ
        by (meson "mod-col-num:1" "nec-imp-act" "→E")
    next
      AOT_show ‹𝓐[ℙ]zm› using ξ
        by (meson "nec-imp-act" "pred-1-1:1" "→E")
    next
      fix y
      AOT_assume ‹𝓐([ℕ]y & [ℙ]zy)›
      AOT_hence ‹𝓐[ℕ]y› and ‹𝓐[ℙ]zy›
        using "Act-Basic:2" "&E" "≡E"(1) by blast+
      AOT_hence 0: ‹[ℙ]zy›
        by (metis RN "≡E"(1) "pred-1-1:1" "sc-eq-fur:2" "→E")
      AOT_thus ‹y = m›
        using "pred-func:1"[THEN "→E", OF "&I"] ξ by metis
    qed
    ultimately AOT_have ‹[ℕ]z & m = z'›
      by (rule "&I")
    AOT_hence ‹∃n m = n'›
      by (rule "∃I")
    hence ?thesis
      by (rule "∨I")
  }
  moreover {
    AOT_assume ‹0 =ℙ m›
    AOT_hence ‹0 = m›
      using "id-R-thm:3"[unconstrain ℛ, unvarify β x, OF "zero:2", OF "pred-thm:2",
                         THEN "→E", OF "pred-1-1:4", THEN "→E"]
      by auto
    hence ?thesis using id_sym "∨I" by blast
  }
  ultimately show ?thesis
    by (metis "∨E"(2) "raa-cor:1")
qed

AOT_theorem "suc-thm": ‹[ℙ]n n'›                                               (807)
proof -
```

```
  AOT_obtain x where m_is_n: ‹x = n'›
    using "free-thms:1"[THEN "≡E"(1), OF "def-suc[den2]"]
    using "∃E" by metis
  AOT_have ‹𝒜([ℕ]n' & [ℙ]n n')›
    apply (rule "rule=E"[rotated, OF suc_eq_desc[symmetric]])
    apply (rule "actual-desc:4"[THEN "→E"])
    by (simp add:  "def-suc[den1]")
  AOT_hence ‹𝒜[ℕ]n'› and ‹𝒜[ℙ]n n'›
    using "Act-Basic:2" "≡E"(1) "&E" by blast+
  AOT_hence ‹𝒜[ℙ]nx›
    using m_is_n[symmetric] "rule=E" by fast+
  AOT_hence ‹[ℙ]nx›
    by (metis RN "≡E"(1) "pred-1-1:1" "sc-eq-fur:2" "→E")
  thus ?thesis
    using m_is_n "rule=E" by fast
qed

AOT_define Numeral1 :: ‹κs› ("1")
  "numerals:1": ‹1 =df 0'›                                            (808.1)

AOT_theorem "prec-facts:1": ‹[ℙ]0 1›                                  (809.1)
  by (auto intro: "numerals:1"[THEN "rule-id-df:2:b[zero]",
                               OF "def-suc[den2]"[unconstrain n, unvarify β,
                                                  OF "zero:2", THEN "→E", OF "0-n"]]
                  "suc-thm"[unconstrain n, unvarify β, OF "zero:2",
                            THEN "→E", OF "0-n"])

(* TODO: more theorems *)

(* Note: we forgo restricted variables for natural cardinals. *)
AOT_define Finite :: ‹τ ⇒ φ› (‹Finite'(_')›)
  "inf-card:1": ‹Finite(x) ≡df NaturalCardinal(x) & [ℕ]x›             (901.1)
AOT_define Infinite :: ‹τ ⇒ φ› (‹Infinite'(_')›)
  "inf-card:2": ‹Infinite(x) ≡df NaturalCardinal(x) & ¬Finite(x)›     (901.2)

AOT_theorem "inf-card-exist:1": ‹NaturalCardinal(#O!)›               (902.1)
  by (safe intro!: card[THEN "≡dfI"] "∃I"(1)[where τ=‹«O!»›] "=I"
                   "num-def:2"[unvarify G] "oa-exist:1")

AOT_theorem "inf-card-exist:2": ‹Infinite(#O!)›                      (902.2)
proof (safe intro!: "inf-card:2"[THEN "≡dfI"] "&I" "inf-card-exist:1")
  AOT_show ‹¬Finite(#O!)›
  proof(rule "raa-cor:2")
    AOT_assume ‹Finite(#O!)›
    AOT_hence 0: ‹[ℕ]#O!›
      using "inf-card:1"[THEN "≡dfE"] "&E"(2) by blast
    AOT_have ‹Numbers(#O!, [λz 𝒜O!z])›
      using "eq-num:3"[unvarify G, OF "oa-exist:1"].
    AOT_hence ‹#O! = #O!›
      using "eq-num:2"[unvarify x G, THEN "≡E"(1), OF "oa-exist:1",
                       OF "num-def:2"[unvarify G], OF "oa-exist:1"]
      by blast
    AOT_hence ‹[ℕ]#O! & #O! = #O!›
      using 0 "&I" by blast
    AOT_hence ‹∃x ([ℕ]x & x = #O!)›
      using "num-def:2"[unvarify G, OF "oa-exist:1"] "∃I"(1) by fast
    AOT_hence ‹◇∃y ([E!]y & ∀u (𝒜[O!]u → u ≠E y))›
      using "modal-axiom"[axiom_inst, unvarify G, THEN "→E", OF "oa-exist:1"] by blast
    AOT_hence ‹∃y ◇([E!]y & ∀u (𝒜[O!]u → u ≠E y))›
      using "BF◇"[THEN "→E"] by blast
    then AOT_obtain b where ‹◇([E!]b & ∀u (𝒜[O!]u → u ≠E b))›
      using "∃E"[rotated] by blast
    AOT_hence ‹◇[E!]b› and 2: ‹◇∀u (𝒜[O!]u → u ≠E b)›
```

```
      using "KBasic2:3"[THEN "→E"] "&E" by blast+
    AOT_hence ‹[λx ◇[E!]x]b›
      by (auto intro!: "β←C"(1) "cqt:2")
    moreover AOT_have ‹O! = [λx ◇[E!]x]›
      by (rule "rule-id-df:1[zero]"[OF "oa:1"]) "cqt:2"
    ultimately AOT_have b_ord: ‹O!b›
      using "rule=E" id_sym by fast
    AOT_hence ‹𝒜O!b›
      by (meson "≡E"(1) "oa-facts:7")
    moreover AOT_have 2: ‹∀u (𝒜[O!]u → u ≠E b)›
      using "modal-lemma"[unvarify G, unconstrain v, OF "oa-exist:1",
                          THEN "→E", OF b_ord, THEN "→E", OF 2].
    ultimately AOT_have ‹b ≠E b›
      using "Ordinary.∀E"[OF 2, unconstrain α, THEN "→E",
                          OF b_ord, THEN "→E"] by blast
    AOT_hence ‹¬(b =E b)›
      by (metis "≡E"(1) "thm-neg=E")
    moreover AOT_have ‹b =E b›
      using "ord=Eequiv:1"[THEN "→E", OF b_ord].
    ultimately AOT_show ‹p & ¬p› for p
      by (metis "raa-cor:3")
  qed
qed



(*<*)
end
(*>*)
```

A.13. Additional Theorems

```
theory AOT_misc
  imports AOT_NaturalNumbers
begin

AOT_theorem PossiblyNumbersEmptyPropertyImpliesZero:
  ‹◇Numbers(x,[λz O!z & z ≠E z]) → x = 0›
proof(rule "→I")
  AOT_have ‹Rigid([λz O!z & z ≠E z])›
  proof (safe intro!: "df-rigid-rel:1"[THEN "≡dfI"] "&I" "cqt:2";
          rule RN; safe intro!: GEN "→I")
    AOT_modally_strict {
      fix x
      AOT_assume ‹[λz O!z & z ≠E z]x›
      AOT_hence ‹O!x & x ≠E x› by (rule "β→C")
      moreover AOT_have ‹x =E x› using calculation[THEN "&E"(1)]
        by (metis "ord=Eequiv:1" "vdash-properties:10")
      ultimately AOT_have ‹x =E x & ¬x =E x›
        by (metis "con-dis-i-e:1" "con-dis-i-e:2:b" "intro-elim:3:a" "thm-neg=E")
      AOT_thus ‹□[λz O!z & z ≠E z]x› using "raa-cor:1" by blast
    }
  qed
  AOT_hence ‹□∀x (Numbers(x,[λz O!z & z ≠E z]) → □Numbers(x,[λz O!z & z ≠E z]))›
    by (safe intro!: "num-cont:2"[unvarify G, THEN "→E"] "cqt:2")
  AOT_hence ‹∀x □(Numbers(x,[λz O!z & z ≠E z]) → □Numbers(x,[λz O!z & z ≠E z]))›
    using "BFs:2"[THEN "→E"] by blast
  AOT_hence ‹□(Numbers(x,[λz O!z & z ≠E z]) → □Numbers(x,[λz O!z & z ≠E z]))›
    using "∀E"(2) by auto
  moreover AOT_assume ‹◇Numbers(x,[λz O!z & z ≠E z])›
  ultimately AOT_have ‹𝒜Numbers(x,[λz O!z & z ≠E z])›
    using "sc-eq-box-box:1"[THEN "≡E"(1), THEN "→E", THEN "nec-imp-act"[THEN "→E"]]
    by blast
  AOT_hence ‹Numbers(x,[λz 𝒜[λz O!z & z ≠E z]z])›
    by (safe intro!: "eq-num:1"[unvarify G, THEN "≡E"(1)] "cqt:2")
  AOT_hence ‹x = #[λz O!z & z ≠E z]›
    by (safe intro!: "eq-num:2"[unvarify G, THEN "≡E"(1)] "cqt:2")
  AOT_thus ‹x = 0›
    using "cqt:2"(1) "rule-id-df:2:b[zero]" "rule=E" "zero:1" by blast
qed

AOT_define Numbers' :: ‹τ ⇒ τ ⇒ φ› (‹Numbers''(_,_')›)
  ‹Numbers'(x, G) ≡df A!x & G↓ & ∀F (x[F] ≡ F ≈E G)›
AOT_theorem Numbers'equiv: ‹Numbers'(x,G) ≡ A!x & ∀F (x[F] ≡ F ≈E G)›
  by (AOT_subst_def Numbers')
     (auto intro!: "≡I" "→I" "&I" "cqt:2" dest: "&E")

AOT_theorem Numbers'DistinctZeroes:
  ‹∃x∃y (◇Numbers'(x,[λz O!z & z ≠E z]) & ◇Numbers'(y,[λz O!z & z ≠E z]) & x ≠ y)›
proof -
  AOT_obtain w₁ where ‹∃w w₁ ≠ w›
    using "two-worlds-exist:4" "PossibleWorld.∃E"[rotated] by fast
  then AOT_obtain w₂ where distinct_worlds: ‹w₁ ≠ w₂›
    using "PossibleWorld.∃E"[rotated] by blast
  AOT_obtain x where x_prop:
    ‹A!x & ∀F (x[F] ≡ w₁ ⊨ F ≈E [λz O!z & z ≠E z])›
    using "A-objects"[axiom_inst] "∃E"[rotated] by fast
  moreover AOT_obtain y where y_prop:
    ‹A!y & ∀F (y[F] ≡ w₂ ⊨ F ≈E [λz O!z & z ≠E z])›
    using "A-objects"[axiom_inst] "∃E"[rotated] by fast
  moreover {
    fix x w
    AOT_assume x_prop: ‹A!x & ∀F (x[F] ≡ w ⊨ F ≈E [λz O!z & z ≠E z])›
```

```
  AOT_have ‹∀F w ⊨ (x[F] ≡ F ≈E [λz O!z & z ≠E z])›
  proof(safe intro!: GEN "conj-dist-w:4"[unvarify p q, OF "log-prop-prop:2",
                            OF "log-prop-prop:2",THEN "≡E"(2)] "≡I" "→I")
    fix F
    AOT_assume ‹w ⊨ x[F]›
    AOT_hence ‹◇x[F]›
      using "fund:1"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(2),
                     OF "PossibleWorld.∃I"] by blast
    AOT_hence ‹x[F]›
      by (metis "en-eq:3[1]" "intro-elim:3:a")
    AOT_thus ‹w ⊨ (F ≈E [λz O!z & z ≠E z])›
      using x_prop[THEN "&E"(2), THEN "∀E"(2), THEN "≡E"(1)] by blast
  next
    fix F
    AOT_assume ‹w ⊨ (F ≈E [λz O!z & z ≠E z])›
    AOT_hence ‹x[F]›
      using x_prop[THEN "&E"(2), THEN "∀E"(2), THEN "≡E"(2)] by blast
    AOT_hence ‹□x[F]›
      using "pre-en-eq:1[1]"[THEN "→E"] by blast
    AOT_thus ‹w ⊨ x[F]›
      using "fund:2"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(1)]
            "PossibleWorld.∀E" by fast
  qed
  AOT_hence ‹w ⊨ ∀F (x[F] ≡ F ≈E [λz O!z & z ≠E z])›
    using "conj-dist-w:5"[THEN "≡E"(2)] by fast
  moreover {
    AOT_have ‹□[λz O!z & z ≠E z]↓›
      by (safe intro!: RN "cqt:2")
    AOT_hence ‹w ⊨ [λz O!z & z ≠E z]↓›
      using "fund:2"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(1),
                     THEN "PossibleWorld.∀E"] by blast
  }
  moreover {
    AOT_have ‹□A!x›
      using x_prop[THEN "&E"(1)] by (metis "oa-facts:2" "→E")
    AOT_hence ‹w ⊨ A!x›
      using "fund:2"[unvarify p, OF "log-prop-prop:2",
                     THEN "≡E"(1), THEN "PossibleWorld.∀E"] by blast
  }
  ultimately AOT_have ‹w ⊨ (A!x & [λz O!z & z ≠E z]↓ &
                             ∀F (x[F] ≡ F ≈E [λz O!z & z ≠E z]))›
    using "conj-dist-w:1"[unvarify p q, OF "log-prop-prop:2",
            OF "log-prop-prop:2", THEN "≡E"(2), OF "&I"] by auto
  AOT_hence ‹∃w w ⊨ (A!x & [λz O!z & z ≠E z]↓ &
                      ∀F (x[F] ≡ F ≈E [λz O!z & z ≠E z]))›
    using "PossibleWorld.∃I" by auto
  AOT_hence ‹◇(A!x & [λz O!z & z ≠E z]↓ & ∀F (x[F] ≡ F ≈E [λz O!z & z ≠E z]))›
    using "fund:1"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(2)] by blast
  AOT_hence ‹◇Numbers'(x,[λz O!z & z ≠E z])›
    by (AOT_subst_def Numbers')
}
ultimately AOT_have ‹◇Numbers'(x,[λz O!z & z ≠E z])›
                and ‹◇Numbers'(y,[λz O!z & z ≠E z])›
  by auto
moreover AOT_have ‹x ≠ y›
proof (rule "ab-obey:2"[THEN "→E"])
  AOT_have ‹□¬∃u [λz O!z & z ≠E z]u›
  proof (safe intro!: RN "raa-cor:2")
    AOT_modally_strict {
      AOT_assume ‹∃u [λz O!z & z ≠E z]u›
      then AOT_obtain u where ‹[λz O!z & z ≠E z]u›
        using "Ordinary.∃E"[rotated] by blast
      AOT_hence ‹O!u & u ≠E u›
```

```
        by (rule "β→C")
      AOT_hence ‹¬(u =E u)›
        by (metis "con-dis-taut:2" "intro-elim:3:d" "modus-tollens:1"
                  "raa-cor:3" "thm-neg=E")
      AOT_hence ‹u =E u & ¬u =E u›
        by (metis "modus-tollens:1" "ord=Eequiv:1" "raa-cor:3" Ordinary.ψ)
      AOT_thus ‹p & ¬p› for p
        by (metis "raa-cor:1")
    }
  qed
  AOT_hence nec_not_ex: ‹∀w w ⊨ ¬∃u [λz O!z & z ≠E z]u›
    using "fund:2"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(1)] by blast
  AOT_have ‹□([λy p]x ≡ p)› for x p
    by (safe intro!: RN "beta-C-meta"[THEN "→E"] "cqt:2")
  AOT_hence ‹∀w w ⊨ ([λy p]x ≡ p)› for x p
    using "fund:2"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(1)] by blast
  AOT_hence world_prop_beta: ‹∀w (w ⊨ [λy p]x ≡ w ⊨ p)› for x p
    using "conj-dist-w:4"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(1)]
          "PossibleWorld.∀E" "PossibleWorld.∀I" by meson

  AOT_have ‹∃p (w1 ⊨ p & ¬w2 ⊨ p)›
  proof(rule "raa-cor:1")
    AOT_assume 0: ‹¬∃p (w1 ⊨ p & ¬w2 ⊨ p)›
    AOT_have 1: ‹w1 ⊨ p → w2 ⊨ p› for p
    proof(safe intro!: GEN "→I")
      AOT_assume ‹w1 ⊨ p›
      AOT_thus ‹w2 ⊨ p›
        using 0 "con-dis-i-e:1" "∃I"(2) "raa-cor:4" by fast
    qed
    moreover AOT_have ‹w2 ⊨ p → w1 ⊨ p› for p
    proof(safe intro!: GEN "→I")
      AOT_assume ‹w2 ⊨ p›
      AOT_hence ‹¬w2 ⊨ ¬p›
        using "coherent:2" "intro-elim:3:a" by blast
      AOT_hence ‹¬w1 ⊨ ¬p›
        using 1["∀I" p, THEN "∀E"(1), OF "log-prop-prop:2"]
        by (metis "modus-tollens:1")
      AOT_thus ‹w1 ⊨ p›
        using "coherent:1" "intro-elim:3:b" "reductio-aa:1" by blast
    qed
    ultimately AOT_have ‹w1 ⊨ p ≡ w2 ⊨ p› for p
      by (metis "intro-elim:2")
    AOT_hence ‹w1 = w2›
      using "sit-identity"[unconstrain s, THEN "→E",
          OF PossibleWorld.ψ[THEN "world:1"[THEN "≡dfE"], THEN "&E"(1)],
          unconstrain s', THEN "→E",
          OF PossibleWorld.ψ[THEN "world:1"[THEN "≡dfE"], THEN "&E"(1)],
          THEN "≡E"(2)] GEN by fast
    AOT_thus ‹w1 = w2 & ¬w1 = w2›
      using  "=-infix" "≡dfE" "con-dis-i-e:1" distinct_worlds by blast
  qed
  then AOT_obtain p where 0: ‹w1 ⊨ p & ¬w2 ⊨ p›
    using "∃E"[rotated] by blast
  AOT_have ‹y[λy p]›
  proof (safe intro!: y_prop[THEN "&E"(2), THEN "∀E"(1), THEN "≡E"(2)] "cqt:2")
    AOT_show ‹w2 ⊨ [λy p] ≈E [λz O!z & z ≠E z]›
    proof (safe intro!:  "cqt:2" "empty-approx:1"[unvarify F H, THEN RN,
                THEN "fund:2"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(1)],
                THEN "PossibleWorld.∀E",
                THEN "conj-dist-w:2"[unvarify p q, OF "log-prop-prop:2",
                                     OF "log-prop-prop:2", THEN "≡E"(1)],
                                     THEN "→E"]
                "conj-dist-w:1"[unvarify p q, OF "log-prop-prop:2",
```

```
                              OF "log-prop-prop:2", THEN "≡E"(2)] "&I")
    AOT_have ‹¬w2 ⊨ ∃u [λy p]u›
    proof (rule "raa-cor:2")
      AOT_assume ‹w2 ⊨ ∃u [λy p]u›
      AOT_hence ‹∃x w2 ⊨ (O!x & [λy p]x)›
        by (metis "conj-dist-w:6" "intro-elim:3:a")
      then AOT_obtain x where ‹w2 ⊨ (O!x & [λy p]x)›
        using "∃E"[rotated] by blast
      AOT_hence ‹w2 ⊨ [λy p]x›
        using "conj-dist-w:1"[unvarify p q, OF "log-prop-prop:2",
                OF "log-prop-prop:2", THEN "≡E"(1), THEN "&E"(2)] by blast
      AOT_hence ‹w2 ⊨ p›
        using world_prop_beta[THEN "PossibleWorld.∀E", THEN "≡E"(1)] by blast
      AOT_thus ‹w2 ⊨ p & ¬w2 ⊨ p›
        using 0[THEN "&E"(2)] "&I" by blast
    qed
    AOT_thus ‹w2 ⊨ ¬∃u [λy p]u›
      by (safe intro!: "coherent:1"[unvarify p, OF "log-prop-prop:2",
                                    THEN "≡E"(2)])
  next
    AOT_show ‹w2 ⊨ ¬∃v [λz O!z & z ≠E z]v›
      using nec_not_ex[THEN "PossibleWorld.∀E"] by blast
  qed
qed
moreover AOT_have ‹¬x[λy p]›
proof(rule "raa-cor:2")
  AOT_assume ‹x[λy p]›
  AOT_hence "w1 ⊨ [λy p] ≈E [λz O!z & z ≠E z]"
    using x_prop[THEN "&E"(2), THEN "∀E"(1), THEN "≡E"(1)]
          "prop-prop2:2" by blast
  AOT_hence "¬w1 ⊨ ¬[λy p] ≈E [λz O!z & z ≠E z]"
    using "coherent:2"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(1)] by blast
  moreover AOT_have "w1 ⊨ ¬([λy p] ≈E [λz O!z & z ≠E z])"
  proof (safe intro!: "cqt:2" "empty-approx:2"[unvarify F H, THEN RN,
                THEN "fund:2"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(1)],
                THEN "PossibleWorld.∀E",
                THEN "conj-dist-w:2"[unvarify p q, OF "log-prop-prop:2",
                    OF "log-prop-prop:2", THEN "≡E"(1)], THEN "→E"]
                "conj-dist-w:1"[unvarify p q, OF "log-prop-prop:2",
                                OF "log-prop-prop:2", THEN "≡E"(2)] "&I")
    fix u
    AOT_have ‹w1 ⊨ O!u›
      using Ordinary.ψ[THEN RN,
              THEN "fund:2"[unvarify p, OF "log-prop-prop:2", THEN "≡E"(1)],
              THEN "PossibleWorld.∀E"] by simp
    moreover AOT_have ‹w1 ⊨ [λy p]u›
      by (safe intro!: world_prop_beta[THEN "PossibleWorld.∀E", THEN "≡E"(2)]
                       0[THEN "&E"(1)])
    ultimately AOT_have ‹w1 ⊨ (O!u & [λy p]u)›
      using "conj-dist-w:1"[unvarify p q, OF "log-prop-prop:2",
                            OF "log-prop-prop:2", THEN "≡E"(2),
                            OF "&I"] by blast
    AOT_hence ‹∃x w1 ⊨ (O!x & [λy p]x)›
      by (rule "∃I")
    AOT_thus ‹w1 ⊨ ∃u [λy p]u›
      by (metis "conj-dist-w:6" "intro-elim:3:b")
  next
    AOT_show ‹w1 ⊨ ¬∃v [λz O!z & z ≠E z]v›
      using "PossibleWorld.∀E" nec_not_ex by fastforce
  qed
  ultimately AOT_show ‹p & ¬p› for p
    using "raa-cor:3" by blast
qed
```

```
    ultimately AOT_have ‹y[λy p] & ¬x[λy p]›
      using "&I" by blast
    AOT_hence ‹∃F (y[F] & ¬x[F])›
      by (metis "existential:1" "prop-prop2:2")
    AOT_thus ‹∃F (x[F] & ¬y[F]) ∨ ∃F (y[F] & ¬x[F])›
      by (rule "∨I")
  qed
  ultimately AOT_have ‹◊Numbers'(x,[λz O!z & z ≠E z]) &
                       ◊Numbers'(y,[λz O!z & z ≠E z]) & x ≠ y›
    using "&I" by blast
  AOT_thus ‹∃x∃y (◊Numbers'(x,[λz O!z & z ≠E z]) &
                  ◊Numbers'(y,[λz O!z & z ≠E z]) & x ≠ y)›
    using "∃I"(2)[where β=x] "∃I"(2)[where β=y] by auto
qed

AOT_theorem restricted_identity:
  ‹x =ℛ y ≡ (InDomainOf(x,ℛ) & InDomainOf(y,ℛ) & x = y)›
  by (auto intro!: "≡I" "→I" "&I"
              dest: "id-R-thm:2"[THEN "→E"] "&E"
                    "id-R-thm:3"[THEN "→E"]
                    "id-R-thm:4"[THEN "→E", OF "∨I"(1), THEN "≡E"(2)])

AOT_theorem induction': ‹∀F ([F]0 & ∀n([F]n → [F]n') → ∀n [F]n)›
proof(rule GEN; rule "→I")
  fix F n
  AOT_assume A: ‹[F]0 & ∀n([F]n → [F]n')›
  AOT_have ‹∀n∀m([ℙ]nm → ([F]n → [F]m))›
  proof(safe intro!: "Number.GEN" "→I")
    fix n m
    AOT_assume ‹[ℙ]nm›
    moreover AOT_have ‹[ℙ]n n'›
      using "suc-thm".
    ultimately AOT_have m_eq_suc_n: ‹m = n'›
      using "pred-func:1"[unvarify z, OF "def-suc[den2]", THEN "→E", OF "&I"]
      by blast
    AOT_assume ‹[F]n›
    AOT_hence ‹[F]n'›
      using A[THEN "&E"(2), THEN "Number.∀E", THEN "→E"] by blast
    AOT_thus ‹[F]m›
      using m_eq_suc_n[symmetric] "rule=E" by fast
  qed
  AOT_thus ‹∀n[F]n›
    using induction[THEN "∀E"(2), THEN "→E", OF "&I", OF A[THEN "&E"(1)]]
    by simp
qed

AOT_define ExtensionOf :: ‹τ ⇒ Π ⇒ φ› (‹ExtensionOf'(_,_')›)
  "exten-property:1": ‹ExtensionOf(x,[G]) ≡df A!x & G↓ & ∀F(x[F] ≡ ∀z([F]z ≡ [G]z))›      (307.1)

AOT_define OrdinaryExtensionOf :: ‹τ ⇒ Π ⇒ φ› (‹OrdinaryExtensionOf'(_,_')›)
   ‹OrdinaryExtensionOf(x,[G]) ≡df A!x & G↓ & ∀F(x[F] ≡ ∀z(O!z → ([F]z ≡ [G]z)))›

AOT_theorem BeingOrdinaryExtensionOfDenotes:
  ‹[λx OrdinaryExtensionOf(x,[G])]↓›
proof(rule "safe-ext"[axiom_inst, THEN "→E", OF "&I"])
  AOT_show ‹[λx A!x & G↓ & [λx ∀F(x[F] ≡ ∀z(O!z → ([F]z ≡ [G]z)))]x]↓›
    by "cqt:2"
next
  AOT_show ‹□∀x (A!x & G↓ & [λx ∀F (x[F] ≡ ∀z (O!z → ([F]z ≡ [G]z)))]x ≡
            OrdinaryExtensionOf(x,[G]))›
  proof(safe intro!: RN GEN)
    AOT_modally_strict {
      fix x
```

```
      AOT_modally_strict {
        AOT_have ‹[λx ∀F (x[F] ≡ ∀z (O!z → ([F]z ≡ [G]z)))]↓›
        proof (safe intro!: "Comprehension_3"[THEN "→E"] RN GEN
                          "→I" "≡I" Ordinary.GEN)
          AOT_modally_strict {
            fix F H u
            AOT_assume ‹□H ≡E F›
            AOT_hence ‹∀u([H]u ≡ [F]u)›
              using eqE[THEN "≡dfE", THEN "&E"(2)] "qml:2"[axiom_inst, THEN "→E"]
              by blast
            AOT_hence 0: ‹[H]u ≡ [F]u› using "Ordinary.∀E" by fast
            {
              AOT_assume ‹∀u([F]u ≡ [G]u)›
              AOT_hence 1: ‹[F]u ≡ [G]u› using "Ordinary.∀E" by fast
              AOT_show ‹[G]u› if ‹[H]u› using 0 1 "≡E"(1) that by blast
              AOT_show ‹[H]u› if ‹[G]u› using 0 1 "≡E"(2) that by blast
            }
            {
              AOT_assume ‹∀u([H]u ≡ [G]u)›
              AOT_hence 1: ‹[H]u ≡ [G]u› using "Ordinary.∀E" by fast
              AOT_show ‹[G]u› if ‹[F]u› using 0 1 "≡E"(1,2) that by blast
              AOT_show ‹[F]u› if ‹[G]u› using 0 1 "≡E"(1,2) that by blast
            }
          }
        qed
      }
      AOT_thus ‹(A!x & G↓ & [λx ∀F (x[F] ≡ ∀z (O!z → ([F]z ≡ [G]z)))]x) ≡
                OrdinaryExtensionOf(x,[G])›
        apply (AOT_subst_def OrdinaryExtensionOf)
        apply (AOT_subst ‹[λx ∀F (x[F] ≡ ∀z (O!z → ([F]z ≡ [G]z)))]x›
                        ‹∀F (x[F] ≡ ∀z (O!z → ([F]z ≡ [G]z)))›)
        by (auto intro!: "beta-C-meta"[THEN "→E"] simp: "oth-class-taut:3:a")
    }
  qed
qed

text‹Fragments of PLM's theory of Concepts.›

AOT_define FimpG :: ‹Π ⇒ Π ⇒ φ› (infixl ‹⇒› 50)
  "F-imp-G": ‹[G] ⇒ [F] ≡df F↓ & G↓ & □∀x ([G]x → [F]x)›              (432)

AOT_define concept :: ‹Π› (‹C!›)
  concepts: ‹C! =df A!›                                               (593)

AOT_register_rigid_restricted_type
  Concept: ‹C!κ›
proof
  AOT_modally_strict {
    AOT_have ‹∃x A!x›
      using "o-objects-exist:2" "qml:2"[axiom_inst] "→E" by blast
    AOT_thus ‹∃x C!x›
      using "rule-id-df:1[zero]"[OF concepts, OF "oa-exist:2"] "rule=E" id_sym
      by fast
  }
next
  AOT_modally_strict {
    AOT_show ‹C!κ → κ↓› for κ
      using "cqt:5:a"[axiom_inst, THEN "→E", THEN "&E"(2)] "→I"
      by blast
  }
next
  AOT_modally_strict {
    AOT_have ‹∀x(A!x → □A!x)›
```

```
      by (simp add: "oa-facts:2" GEN)
    AOT_thus ‹∀x(C!x → □C!x)›
      using "rule-id-df:1[zero]"[OF concepts, OF "oa-exist:2"] "rule=E" id_sym
      by fast
  }
qed

AOT_register_variable_names
  Concept: c d e

AOT_theorem "concept-comp:1": ‹∃x(C!x & ∀F(x[F] ≡ φ{F}))›                    (595.1)
    using concepts[THEN "rule-id-df:1[zero]", OF "oa-exist:2", symmetric]
          "A-objects"[axiom_inst]
          "rule=E" by fast

AOT_theorem "concept-comp:2": ‹∃!x(C!x & ∀F(x[F] ≡ φ{F}))›                   (595.2)
    using concepts[THEN "rule-id-df:1[zero]", OF "oa-exist:2", symmetric]
          "A-objects!"
          "rule=E" by fast

AOT_theorem "concept-comp:3": ‹ιx(C!x & ∀F(x[F] ≡ φ{F}))↓›                   (595.3)
  using "concept-comp:2" "A-Exists:2"[THEN "≡E"(2)] "RA[2]" by blast

AOT_theorem "concept-comp:4":                                                 (595.4)
  ‹ιx(C!x & ∀F(x[F] ≡ φ{F})) = ιx(A!x & ∀F(x[F] ≡ φ{F}))›
    using "=I"(1)[OF "concept-comp:3"]
          "rule=E"[rotated]
          concepts[THEN "rule-id-df:1[zero]", OF "oa-exist:2"]
          by fast

AOT_define conceptInclusion :: ‹τ ⇒ τ ⇒ φ› (infixl ‹≼› 100)
  "con:1": ‹c ≼ d ≡df ∀F(c[F] → d[F])›                                        (605.1)


AOT_define conceptOf :: ‹τ ⇒ τ ⇒ φ› (‹ConceptOf'(_,_')›)
  "concept-of-G": ‹ConceptOf(c,G) ≡df G↓ & ∀F (c[F] ≡ [G] ⇒ [F])›             (651)

AOT_theorem ConceptOfOrdinaryProperty: ‹([H] ⇒ O!) → [λx ConceptOf(x,H)]↓›
proof(rule "→I")
  AOT_assume ‹[H] ⇒ O!›
  AOT_hence ‹□∀x([H]x → O!x)›
    using "F-imp-G"[THEN "≡dfE"] "&E" by blast
  AOT_hence ‹□□∀x([H]x → O!x)›
    using "S5Basic:6"[THEN "≡E"(1)] by blast
  moreover AOT_have ‹□□∀x([H]x → O!x) →
                    □∀F∀G(□(G ≡E F) → ([H] ⇒ [F] ≡ [H] ⇒ [G]))›
  proof(rule RM; safe intro!: "→I" GEN "≡I")
    AOT_modally_strict {
      fix F G
      AOT_assume 0: ‹□∀x([H]x → O!x)›
      AOT_assume ‹□G ≡E F›
      AOT_hence 1: ‹□∀u([G]u ≡ [F]u)›
        by (AOT_subst_thm eqE[THEN "≡Df", THEN "≡S"(1), OF "&I",
              OF "cqt:2[const_var]"[axiom_inst],
              OF "cqt:2[const_var]"[axiom_inst], symmetric])
      {
        AOT_assume ‹[H] ⇒ [F]›
        AOT_hence ‹□∀x([H]x → [F]x)›
          using "F-imp-G"[THEN "≡dfE"] "&E" by blast
        moreover AOT_modally_strict {
          AOT_assume ‹∀x([H]x → O!x)›
          moreover AOT_assume ‹∀u([G]u ≡ [F]u)›
          moreover AOT_assume ‹∀x([H]x → [F]x)›
```

```
          ultimately AOT_have ‹[H]x → [G]x› for x
            by (auto intro!: "→I" dest!: "∀E"(2) dest: "→E" "≡E")
          AOT_hence ‹∀x([H]x → [G]x)›
            by (rule GEN)
        }
        ultimately AOT_have ‹□∀x([H]x → [G]x)›
          using "RN[prem]"[where
            Γ="{«∀x([H]x → O!x)», «∀u([G]u ≡ [F]u)», «∀x([H]x → [F]x)»}"]
          using 0 1 by fast
        AOT_thus ‹[H] ⇒ [G]›
          by (AOT_subst_def "F-imp-G")
             (safe intro!: "cqt:2" "&I")
      }
      {
        AOT_assume ‹[H] ⇒ [G]›
        AOT_hence ‹□∀x([H]x → [G]x)›
          using "F-imp-G"[THEN "≡dfE"] "&E" by blast
        moreover AOT_modally_strict {
          AOT_assume ‹∀x([H]x → O!x)›
          moreover AOT_assume ‹∀u([G]u ≡ [F]u)›
          moreover AOT_assume ‹∀x([H]x → [G]x)›
          ultimately AOT_have ‹[H]x → [F]x› for x
            by (auto intro!: "→I" dest!: "∀E"(2) dest: "→E" "≡E")
          AOT_hence ‹∀x([H]x → [F]x)›
            by (rule GEN)
        }
        ultimately AOT_have ‹□∀x([H]x → [F]x)›
          using "RN[prem]"[where
            Γ="{«∀x([H]x → O!x)», «∀u([G]u ≡ [F]u)», «∀x([H]x → [G]x)»}"]
          using 0 1 by fast
        AOT_thus ‹[H] ⇒ [F]›
          by (AOT_subst_def "F-imp-G")
             (safe intro!: "cqt:2" "&I")
      }
    }
  qed
  ultimately AOT_have ‹□∀F∀G(□(G ≡E F) → ([H] ⇒ [F] ≡ [H] ⇒ [G]))›
    using "→E" by blast
  AOT_hence 0: ‹[λx ∀F(x[F] ≡ ([H] ⇒ [F]))]↓›
    using Comprehension_3[THEN "→E"] by blast
  AOT_show ‹[λx ConceptOf(x,H)]↓›
  proof (rule "safe-ext"[axiom_inst, THEN "→E", OF "&I"])
    AOT_show ‹[λx C!x & [λx ∀F(x[F] ≡ ([H] ⇒ [F]))]x]↓› by "cqt:2"
  next
    AOT_show ‹□∀x (C!x & [λx ∀F (x[F] ≡ [H] ⇒ [F])]x ≡ ConceptOf(x,H))›
    proof (rule "RN[prem]"[where Γ=‹{«[λx ∀F(x[F] ≡ ([H] ⇒ [F]))]↓»}›, simplified])
      AOT_modally_strict {
        AOT_assume 0: ‹[λx ∀F (x[F] ≡ [H] ⇒ [F])]↓›
        AOT_show ‹∀x (C!x & [λx ∀F (x[F] ≡ [H] ⇒ [F])]x ≡ ConceptOf(x,H))›
        proof(safe intro!: GEN "≡I" "→I" "&I")
          fix x
          AOT_assume ‹C!x & [λx ∀F (x[F] ≡ [H] ⇒ [F])]x›
          AOT_thus ‹ConceptOf(x,H)›
            by (AOT_subst_def "concept-of-G")
               (auto intro!: "&I" "cqt:2" dest: "&E" "β→C")
        next
          fix x
          AOT_assume ‹ConceptOf(x,H)›
          AOT_hence ‹C!x & (H↓ & ∀F(x[F] ≡ [H] ⇒ [F]))›
            by (AOT_subst_def (reverse) "concept-of-G")
          AOT_thus ‹C!x› and ‹[λx ∀F(x[F] ≡ [H] ⇒ [F])]x›
            by (auto intro!: "β←C" 0 "cqt:2" dest: "&E")
        qed
```

```
      }
    next
      AOT_show ‹□[λx ∀F(x[F] ≡ ([H] ⇒ [F]))]↓›
        using "exist-nec"[THEN "→E"] 0 by blast
    qed
  qed
qed

AOT_theorem "con-exists:1": ‹∃c ConceptOf(c,G)›                    (652.1)
proof -
  AOT_obtain c where ‹∀F (c[F] ≡ [G] ⇒ [F])›
    using "concept-comp:1" "Concept.∃E"[rotated] by meson
  AOT_hence ‹ConceptOf(c,G)›
    by (auto intro!: "concept-of-G"[THEN "≡df I"] "&I" "cqt:2" Concept.ψ)
  thus ?thesis by (rule "Concept.∃I")
qed

AOT_theorem "con-exists:2": ‹∃!c ConceptOf(c,G)›                   (652.2)
proof -
  AOT_have ‹∃!c ∀F (c[F] ≡ [G] ⇒ [F])›
    using "concept-comp:2" by simp
  moreover {
    AOT_modally_strict {
      fix x
      AOT_assume ‹∀F (x[F] ≡ [G] ⇒ [F])›
      moreover AOT_have ‹[G] ⇒ [G]›
        by (safe intro!: "F-imp-G"[THEN "≡df I"] "&I" "cqt:2" RN GEN "→I")
      ultimately AOT_have ‹x[G]›
        using "∀E"(2) "≡E" by blast
      AOT_hence ‹A!x›
        using "encoders-are-abstract"[THEN "→E", OF "∃I"(2)] by simp
      AOT_hence ‹C!x›
        using concepts[THEN "rule-id-df:1[zero]", OF "oa-exist:2", symmetric]
          "rule=E"[rotated]
        by fast
    }
  }
  ultimately show ?thesis
    by (AOT_subst ‹ConceptOf(c,G)› ‹∀F (c[F] ≡ [G] ⇒ [F])› for: c;
           AOT_subst_def "concept-of-G")
       (auto intro!: "≡I" "→I" "&I" "cqt:2" Concept.ψ dest: "&E")
qed

AOT_theorem "con-exists:3": ‹ιc ConceptOf(c,G)↓›                   (652.3)
  by (safe intro!: "A-Exists:2"[THEN "≡E"(2)] "con-exists:2"[THEN "RA[2]"])


AOT_define theConceptOfG :: ‹τ ⇒ κs› (‹c_›)
  "concept-G": ‹cG =df ιc ConceptOf(c, G)›                          (653)

AOT_theorem "concept-G[den]": ‹cG↓›                                 (653)
  by (auto intro!: "rule-id-df:1"[OF "concept-G"]
                   "t=t-proper:1"[THEN "→E"]
                   "con-exists:3")


AOT_theorem "concept-G[concept]": ‹C!cG›                            (653)
proof -
  AOT_have ‹𝒜(C!cG & ConceptOf(cG, G))›
    by (auto intro!: "actual-desc:2"[unvarify x, THEN "→E"]
                     "rule-id-df:1"[OF "concept-G"]
                     "concept-G[den]"
                     "con-exists:3")
```

```
  AOT_hence ‹𝒜C!c_G›
    by (metis "Act-Basic:2" "con-dis-i-e:2:a" "intro-elim:3:a")
  AOT_hence ‹𝒜A!c_G›
    using "rule-id-df:1[zero]"[OF concepts, OF "oa-exist:2"]
          "rule=E" by fast
  AOT_hence ‹A!c_G›
    using "oa-facts:8"[unvarify x, THEN "≡E"(2)] "concept-G[den]" by blast
  thus ?thesis
    using "rule-id-df:1[zero]"[OF concepts, OF "oa-exist:2", symmetric]
          "rule=E" by fast
qed

AOT_theorem "conG-strict": ‹c_G = ιc ∀F(c[F] ≡ [G] ⇒ [F])›                    (654)
proof (rule "id-eq:3"[unvarify α β γ, THEN "→E"])
  AOT_have ‹□∀x (C!x & ConceptOf(x,G) ≡ C!x & ∀F (x[F] ≡ [G] ⇒ [F]))›
    by (auto intro!: "concept-of-G"[THEN "≡dfI"] RN GEN "≡I" "→I" "&I" "cqt:2"
             dest: "&E";
        auto dest: "∀E"(2) "≡E"(1,2) dest!: "&E"(2) "concept-of-G"[THEN "≡dfE"])
  AOT_thus ‹c_G = ιc ConceptOf(c, G) & ιc ConceptOf(c, G) = ιc ∀F(c[F] ≡ [G] ⇒ [F])›
    by (auto intro!: "&I" "rule-id-df:1"[OF "concept-G"] "con-exists:3"
                    "equiv-desc-eq:3"[THEN "→E"])
qed(auto simp: "concept-G[den]" "con-exists:3" "concept-comp:3")


AOT_theorem "conG-lemma:1": ‹∀F(c_G[F] ≡ [G] ⇒ [F])›                        (655.1)
proof(safe intro!: GEN "≡I" "→I")
  fix F
  AOT_have ‹𝒜∀F(c_G[F] ≡ [G] ⇒ [F])›
    using "actual-desc:4"[THEN "→E", OF "concept-comp:3",
                          THEN "Act-Basic:2"[THEN "≡E"(1)],
                          THEN "&E"(2)]
          "conG-strict"[symmetric] "rule=E" by fast
  AOT_hence ‹𝒜(c_G[F] ≡ [G] ⇒ [F])›
    using "logic-actual-nec:3"[axiom_inst, THEN "≡E"(1)] "∀E"(2)
    by blast
  AOT_hence 0: ‹𝒜c_G[F] ≡ 𝒜[G] ⇒ [F]›
    using "Act-Basic:5"[THEN "≡E"(1)] by blast
  {
    AOT_assume ‹c_G[F]›
    AOT_hence ‹𝒜c_G[F]›
      by(safe intro!: "en-eq:10[1]"[unvarify x_1, THEN "≡E"(2)]
                      "concept-G[den]")
    AOT_hence ‹𝒜[G] ⇒ [F]›
      using 0[THEN "≡E"(1)] by blast
    AOT_hence ‹𝒜(F↓ & G↓ & □∀x([G]x → [F]x))›
      by (AOT_subst_def (reverse) "F-imp-G")
    AOT_hence ‹𝒜□∀x([G]x → [F]x)›
      using "Act-Basic:2"[THEN "≡E"(1)] "&E" by blast
    AOT_hence ‹□∀x([G]x → [F]x)›
      using "qml-act:2"[axiom_inst, THEN "≡E"(2)] by simp
    AOT_thus ‹[G] ⇒ [F]›
      by (AOT_subst_def "F-imp-G"; auto intro!: "&I" "cqt:2")
  }
  {
    AOT_assume ‹[G] ⇒ [F]›
    AOT_hence ‹□∀x([G]x → [F]x)›
      by (safe dest!: "F-imp-G"[THEN "≡dfE"] "&E"(2))
    AOT_hence ‹𝒜□∀x([G]x → [F]x)›
      using "qml-act:2"[axiom_inst, THEN "≡E"(1)] by simp
    AOT_hence ‹𝒜(F↓ & G↓ & □∀x([G]x → [F]x))›
      by (auto intro!: "Act-Basic:2"[THEN "≡E"(2)] "&I" "cqt:2"
             intro: "RA[2]")
    AOT_hence ‹𝒜([G] ⇒ [F])›
```

```
      by (AOT_subst_def "F-imp-G")
    AOT_hence ‹𝒜c_G[F]›
      using 0[THEN "≡E"(2)] by blast
    AOT_thus ‹c_G[F]›
      by(safe intro!: "en-eq:10[1]"[unvarify x_1, THEN "≡E"(1)]
                      "concept-G[den]")
  }
qed

AOT_theorem conH_enc_ord:
  ‹([H] ⇒ O!) → □∀F ∀G (□G ≡_E F → (c_H[F] ≡ c_H[G]))›
proof(rule "→I")
  AOT_assume 0: ‹[H] ⇒ O!›
  AOT_have 0: ‹□([H] ⇒ O!)›
    apply (AOT_subst_def "F-imp-G")
    using 0[THEN "≡_dfE"[OF "F-imp-G"]]
    by (auto intro!: "KBasic:3"[THEN "≡E"(2)] "&I" "exist-nec"[THEN "→E"]
             dest: "&E" 4[THEN "→E"])
  moreover AOT_have ‹□([H] ⇒ O!) → □∀F ∀G (□G ≡_E F → (c_H[F] ≡ c_H[G]))›
  proof(rule RM; safe intro!: "→I" GEN)
    AOT_modally_strict {
      fix F G
      AOT_assume ‹[H] ⇒ O!›
      AOT_hence 0: ‹□∀x ([H]x → O!x)›
        by (safe dest!: "F-imp-G"[THEN "≡_dfE"] "&E"(2))
      AOT_assume 1: ‹□G ≡_E F›
      AOT_assume ‹c_H[F]›
      AOT_hence ‹[H] ⇒ [F]›
        using "conG-lemma:1"[THEN "∀E"(2), THEN "≡E"(1)] by simp
      AOT_hence 2: ‹□∀x ([H]x → [F]x)›
        by (safe dest!: "F-imp-G"[THEN "≡_dfE"] "&E"(2))
      AOT_modally_strict {
        AOT_assume 0: ‹∀x ([H]x → O!x)›
        AOT_assume 1: ‹∀x ([H]x → [F]x)›
        AOT_assume 2: ‹G ≡_E F›
        AOT_have ‹∀x ([H]x → [G]x)›
        proof(safe intro!: GEN "→I")
          fix x
          AOT_assume ‹[H]x›
          AOT_hence ‹O!x› and ‹[F]x›
            using 0 1 "∀E"(2) "→E" by blast+
          AOT_thus ‹[G]x›
            using 2[THEN eqE[THEN "≡_dfE"], THEN "&E"(2)]
                  "∀E"(2) "→E" "≡E"(2) calculation by blast
        qed
      }
      AOT_hence ‹□∀x ([H]x → [G]x)›
        using "RN[prem]"[where Γ=‹{«∀x ([H]x → O!x)»,
                                  «∀x ([H]x → [F]x)»,
                                  «G ≡_E F»}›, simplified] 0 1 2 by fast
      AOT_hence ‹[H] ⇒ [G]›
        by (safe intro!: "F-imp-G"[THEN "≡_dfI"] "&I" "cqt:2")
      AOT_hence ‹c_H[G]›
        using "conG-lemma:1"[THEN "∀E"(2), THEN "≡E"(2)] by simp
    } note 0 = this
    AOT_modally_strict {
      fix F G
      AOT_assume ‹[H] ⇒ O!›
      moreover AOT_assume ‹□G ≡_E F›
      moreover AOT_have ‹□F ≡_E G›
        by (AOT_subst ‹F ≡_E G› ‹G ≡_E F›)
           (auto intro!: calculation(2)
                         eqE[THEN "≡_dfI"]
```

```
                          "≡I" "→I" "&I" "cqt:2" Ordinary.GEN
                  dest!: eqE[THEN "≡_dfE"] "&E"(2)
                  dest: "≡E"(1,2) "Ordinary.∀E")
      ultimately AOT_show ‹(c_H[F] ≡ c_H[G])›
        using 0 "≡I" "→I" by auto
    }
  qed
  ultimately AOT_show ‹□∀F ∀G (□G ≡_E F → (c_H[F] ≡ c_H[G]))›
    using "→E" by blast
qed

AOT_theorem concept_inclusion_denotes_1:
  ‹([H] ⇒ O!) → [λx c_H ≼ x]↓›
proof(rule "→I")
  AOT_assume 0: ‹[H] ⇒ O!›
  AOT_show ‹[λx c_H ≼ x]↓›
  proof(rule "safe-ext"[axiom_inst, THEN "→E", OF "&I"])
    AOT_show ‹[λx C!x & ∀F(c_H[F] → x[F])]↓›
      by (safe intro!: conjunction_denotes[THEN "→E", OF "&I"]
                       Comprehension_2'[THEN "→E"]
                       conH_enc_ord[THEN "→E", OF 0]) "cqt:2"
  next
    AOT_show ‹□∀x (C!x & ∀F (c_H[F] → x[F]) ≡ c_H ≼ x)›
      by (safe intro!: RN GEN; AOT_subst_def "con:1")
         (auto intro!: "≡I" "→I" "&I" "concept-G[concept]" dest: "&E")
  qed
qed

AOT_theorem concept_inclusion_denotes_2:
  ‹([H] ⇒ O!) → [λx x ≼ c_H]↓›
proof(rule "→I")
  AOT_assume 0: ‹[H] ⇒ O!›
  AOT_show ‹[λx x ≼ c_H]↓›
  proof(rule "safe-ext"[axiom_inst, THEN "→E", OF "&I"])
    AOT_show ‹[λx C!x & ∀F(x[F] → c_H[F])]↓›
      by (safe intro!: conjunction_denotes[THEN "→E", OF "&I"]
                       Comprehension_1'[THEN "→E"]
                       conH_enc_ord[THEN "→E", OF 0]) "cqt:2"
  next
    AOT_show ‹□∀x (C!x & ∀F (x[F] → c_H[F]) ≡ x ≼ c_H)›
      by (safe intro!: RN GEN; AOT_subst_def "con:1")
         (auto intro!: "≡I" "→I" "&I" "concept-G[concept]" dest: "&E")
  qed
qed

AOT_define ThickForm :: ‹τ ⇒ τ ⇒ φ› (‹FormOf'(_,_')›)
  "tform-of": ‹FormOf(x,G) ≡_df A!x & G↓ & ∀F(x[F] ≡ [G] ⇒ [F])›          (434)

AOT_theorem FormOfOrdinaryProperty: ‹([H] ⇒ O!) → [λx FormOf(x,H)]↓›
proof(rule "→I")
  AOT_assume 0: ‹[H] ⇒ [O!]›
  AOT_show ‹[λx FormOf(x,H)]↓›
  proof (rule "safe-ext"[axiom_inst, THEN "→E", OF "&I"])
    AOT_show ‹[λx ConceptOf(x,H)]↓›
      using 0 ConceptOfOrdinaryProperty[THEN "→E"] by blast
    AOT_show ‹□∀x (ConceptOf(x,H) ≡ FormOf(x,H))›
    proof(safe intro!: RN GEN)
      AOT_modally_strict {
        fix x
        AOT_modally_strict {
          AOT_have ‹A!x ≡ A!x›
            by (simp add: "oth-class-taut:3:a")
          AOT_hence ‹C!x ≡ A!x›
```

```
            using "rule-id-df:1[zero]"[OF concepts, OF "oa-exist:2"]
                  "rule=E" id_sym by fast
        }
        AOT_thus ‹ConceptOf(x,H) ≡ FormOf(x,H)›
          by (AOT_subst_def "tform-of";
              AOT_subst_def "concept-of-G";
              AOT_subst ‹C!x› ‹A!x›)
             (auto intro!: "≡I" "→I" "&I" dest: "&E")
      }
    qed
  qed
qed

AOT_theorem equal_E_rigid_one_to_one: ‹Rigid1-1((=E))›
proof (safe intro!: "df-1-1:2"[THEN "≡dfI"] "&I" "df-1-1:1"[THEN "≡dfI"]
                    GEN "→I" "df-rigid-rel:1"[THEN "≡dfI"] "=E[denotes]")
  fix x y z
  AOT_assume ‹x =E z & y =E z›
  AOT_thus ‹x = y›
    by (metis "rule=E" "&E"(1) "Conjunction Simplification"(2)
              "=E-simple:2" id_sym "→E")
next
  AOT_have ‹∀x∀y □(x =E y → □x =E y)›
  proof(rule GEN; rule GEN)
    AOT_show ‹□(x =E y → □x =E y)› for x y
      by (meson RN "deduction-theorem" "id-nec3:1" "≡E"(1))
  qed
  AOT_hence ‹∀x1...∀xn □([(=E)]x1...xn → □[(=E)]x1...xn)›
    by (rule tuple_forall[THEN "≡dfI"])
  AOT_thus ‹□∀x1...∀xn ([(=E)]x1...xn → □[(=E)]x1...xn)›
    using BF[THEN "→E"] by fast
qed

AOT_theorem equal_E_domain: ‹InDomainOf(x,(=E)) ≡ O!x›
proof(safe intro!: "≡I" "→I")
  AOT_assume ‹InDomainOf(x,(=E))›
  AOT_hence ‹∃y x =E y›
    by (metis "≡dfE" "df-1-1:5")
  then AOT_obtain y where ‹x =E y›
    using "∃E"[rotated] by blast
  AOT_thus ‹O!x›
    using "=E-simple:1"[THEN "≡E"(1)] "&E" by blast
next
  AOT_assume ‹O!x›
  AOT_hence ‹x =E x›
    by (metis "ord=Eequiv:1"[THEN "→E"])
  AOT_hence ‹∃y x =E y›
    using "∃I"(2) by fast
  AOT_thus ‹InDomainOf(x,(=E))›
    by (metis "≡dfI" "df-1-1:5")
qed

AOT_theorem shared_urelement_projection_identity:
  assumes ‹∀y [λx (y[λz [R]zx])]↓›
  shows ‹∀F([F]a ≡ [F]b) → [λz [R]za] = [λz [R]zb]›
proof(rule "→I")
  AOT_assume 0: ‹∀F([F]a ≡ [F]b)›
  {
    fix z
    AOT_have ‹[λx (z[λz [R]zx])]↓›
      using assms[THEN "∀E"(2)].
    AOT_hence 1: ‹∀x ∀y (∀F ([F]x ≡ [F]y) → □(z[λz [R]zx] ≡ z[λz [R]zy]))›
      using "kirchner-thm-cor:1"[THEN "→E"]
```

```
      by blast
    AOT_have ‹□(z[λz [R]za] ≡ z[λz [R]zb])›
      using 1[THEN "∀E"(2), THEN "∀E"(2), THEN "→E", OF 0] by blast
  }
  AOT_hence ‹∀z □(z[λz [R]za] ≡ z[λz [R]zb])›
    by (rule GEN)
  AOT_hence ‹□∀z(z[λz [R]za] ≡ z[λz [R]zb])›
    by (rule BF[THEN "→E"])
  AOT_thus ‹[λz [R]za] = [λz [R]zb]›
    by (AOT_subst_def "identity:2")
       (auto intro!: "&I" "cqt:2")
qed

AOT_theorem shared_urelement_exemplification_identity:
  assumes ‹∀y [λx (y[λz [G]x])]↓›
  shows ‹∀F([F]a ≡ [F]b) → ([G]a) = ([G]b)›
proof(rule "→I")
  AOT_assume 0: ‹∀F([F]a ≡ [F]b)›
  {
    fix z
    AOT_have ‹[λx (z[λz [G]x])]↓›
      using assms[THEN "∀E"(2)].
    AOT_hence 1: ‹∀x ∀y (∀F ([F]x ≡ [F]y) → □(z[λz [G]x] ≡ z[λz [G]y]))›
      using "kirchner-thm-cor:1"[THEN "→E"]
      by blast
    AOT_have ‹□(z[λz [G]a] ≡ z[λz [G]b])›
      using 1[THEN "∀E"(2), THEN "∀E"(2), THEN "→E", OF 0] by blast
  }
  AOT_hence ‹∀z □(z[λz [G]a] ≡ z[λz [G]b])›
    by (rule GEN)
  AOT_hence ‹□∀z(z[λz [G]a] ≡ z[λz [G]b])›
    by (rule BF[THEN "→E"])
  AOT_hence ‹[λz [G]a] = [λz [G]b]›
    by (AOT_subst_def "identity:2")
       (auto intro!: "&I" "cqt:2")
  AOT_thus ‹([G]a) = ([G]b)›
    by (safe intro!: "identity:4"[THEN "≡dfI"] "&I" "log-prop-prop:2")
qed

text‹The assumptions of the theorems above are derivable, if the additional
     introduction rules for the upcoming extension of @{thm "cqt:2[lambda]"}
     are explicitly allowed (while they are currently not part of the
     abstraction layer).›
notepad
begin
  AOT_modally_strict {
    AOT_have ‹∀R∀y [λx (y[λz [R]zx])]↓›
      by (safe intro!: GEN "cqt:2" AOT_instance_of_cqt_2_intro_next)
    AOT_have ‹∀G∀y [λx (y[λz [G]x])]↓›
      by (safe intro!: GEN "cqt:2" AOT_instance_of_cqt_2_intro_next)
  }
end

end
```

References

[1] Jesse Alama, Paul E. Oppenheimer, and Edward N. Zalta. “Automating Leibniz’s Theory of Concepts”. In: *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*. Ed. by Amy P. Felty and Aart Middeldorp. Vol. 9195. Lecture Notes in Computer Science. Springer, 2015, pp. 73–97. DOI: 10.1007/978-3-319-21401-6.

[2] Christoph Benzmüller. “Universal (Meta-)Logical Reasoning: Recent Successes”. In: *Science of Computer Programming* 172 (2019), pp. 48–62. DOI: 10.1016/j.scico.2018.10.008.

[3] Christoph Benzmüller, Maximilian Claus, and Nik Sultana. “Systematic Verification of the Modal Logic Cube in Isabelle/HOL”. In: *PxTP 2015*. Ed. by Cezary Kaliszyk and Andrei Paskevich. Vol. 186. Berlin, Germany: EPTCS, 2015, pp. 27–41. DOI: 10.4204/EPTCS.186.5.

[4] Christoph Benzmüller and David Fuenmayor. “Computer-supported Analysis of Positive Properties, Ultrafilters and Modal Collapse in Variants of Gödel’s Ontological Argument”. In: *Bulletin of the Section of Logic* 49.2 (2020), pp. 127–148. DOI: 10.18778/0138-0680.2020.08.

[5] Christoph Benzmüller, Xavier Parent, and Leendert van der Torre. “Designing normative theories for ethical and legal reasoning: LogiKEy framework, methodology, and tool support”. In: *Artificial Intelligence* 287 (2020), p. 103348. ISSN: 0004-3702. DOI: 10.1016/j.artint.2020.103348.

[6] Christoph Benzmüller and Lawrence Paulson. “Quantified Multimodal Logics in Simple Type Theory”. In: *Logica Universalis (Special Issue on Multimodal Logics)* 7.1 (2013), pp. 7–20. DOI: 10.1007/s11787-012-0052-y.

[7] Christoph Benzmüller and Sebastian Reiche. *Modeling and Automating Public Announcement Logic with Relativized Common Knowledge as a Fragment of HOL in LogiKEy*. 2021. arXiv: 2111.01654 `[cs.AI]`.

[8] Christoph Benzmüller and Bruno Woltzenlogel Paleo. “Automating Gödel’s Ontological Proof of God’s Existence with Higher-order Automated Theorem Provers”. In: *ECAI 2014*. Ed. by Torsten Schaub, Gerhard Friedrich, and Barry O’Sullivan. Vol. 263. Frontiers in Artificial Intelligence and Applications. IOS Press, 2014, pp. 93–98. DOI: 10.3233/978-1-61499-419-0-93.

[9] Christoph Benzmüller and Bruno Woltzenlogel Paleo. “Higher-Order Modal Logics: Automation and Applications”. In: July 2015. ISBN: 978-3-319-21767-3. DOI: 10.1007/978-3-319-21768-0_2.

[10] Christoph Benzmüller et al. "LogiKEy workbench: Deontic logics, logic combinations and expressive ethical and legal reasoning (Isabelle/HOL dataset)". In: *Data in Brief* 33 (2020), p. 106409. ISSN: 2352-3409. DOI: 10.1016/j.dib.2020.106409.

[11] Patrick Blackburn and Johan van Benthem. "Modal logic: a semantic perspective". In: *Handbook of Modal Logic*. Ed. by Patrick Blackburn, Johan Van Benthem, and Frank Wolter. Vol. 3. Studies in Logic and Practical Reasoning. Elsevier, 2007, pp. 1–84. DOI: 10.1016/S1570-2464(07)80004-8.

[12] Jasmin Christian Blanchette and Tobias Nipkow. "Nitpick: A Counterexample Generator for Higher-Order Logic Based on a Relational Model Finder". In: *Interactive Theorem Proving*. Ed. by Matt Kaufmann and Lawrence C. Paulson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 131–146. ISBN: 978-3-642-14052-5. DOI: 10.1007/978-3-642-14052-5_11.

[13] Sascha Böhme and Tjark Weber. "Fast LCF-Style Proof Reconstruction for Z3". In: *Interactive Theorem Proving*. Ed. by Matt Kaufmann and Lawrence C. Paulson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 179–194. ISBN: 978-3-642-14052-5. DOI: 10.1007/978-3-642-14052-5_14.

[14] George Boolos. "The Consistency of Frege's Foundations of Arithmetic". In: *On Being and Saying: Essays in Honor of Richard Cartwright*. Ed. by J. Thomson. Cambridge: MIT Press, 1987, pp. 3–20.

[15] Thomas Bouton et al. "veriT: An Open, Trustable and Efficient SMT-Solver". In: *Automated Deduction – CADE-22*. Ed. by Renate A. Schmidt. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 151–156. ISBN: 978-3-642-02959-2. DOI: 10.1007/978-3-642-02959-2_12.

[16] N.G de Bruijn. "Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem". In: *Indagationes Mathematicae (Proceedings)* 75.5 (1972), pp. 381–392. ISSN: 1385-7258. DOI: 10.1016/1385-7258(72)90034-0.

[17] Branden Fitelson and Edward N. Zalta. "Steps Toward a Computational Metaphysics". In: *Journal Philosophical Logic* 36.2 (2007), pp. 227–247. DOI: 10.1007/s10992-006-9038-7.

[18] Melvin Fitting. "A tableau system for propositional S5." In: *Notre Dame Journal of Formal Logic* 18.2 (1977), pp. 292–294. DOI: 10.1305/ndjfl/1093887933.

[19] Mathias Fleury and Hans-Jörg Schurr. "Reconstructing veriT Proofs in Isabelle/HOL". In: *Electronic Proceedings in Theoretical Computer Science* 301 (Aug. 2019), pp. 36–50. ISSN: 2075-2180. DOI: 10.4204/eptcs.301.6.

[20] David Fuenmayor and Christoph Benzmüller. "A Case Study On Computational Hermeneutics: E. J. Lowe's Modal Ontological Argument". In: *Beyond Faith and Rationality: Essays on Logic, Religion and Philosophy*. Ed. by Ricardo Silvestre et al. Vol. 34. Sophia Studies in Cross-cultural Philosophy of Traditions and Cultures. Springer Nature Switzerland AG, 2020. Chap. 12. DOI: 10.1007/978-3-030-43535-6_12.

[21] Daniel Gallin. *Intensional and Higher-Order Modal Logic: With Applications to Montague Semantics*. North-Holland Mathematics Studies: Volume 19, 1975.

[22] Jeremy Gibbons and Nicolas Wu. "Folding Domain-Specific Languages: Deep and Shallow Embeddings (Functional Pearl)". In: *SIGPLAN Not.* 49.9 (Aug. 2014), pp. 339–347. ISSN: 0362-1340. DOI: 10.1145/2692915.2628138.

[23] Patrick Grim and Daniel Singer. "Computational Philosophy". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Spring 2020. Metaphysics Research Lab, Stanford University, 2020. URL: https://plato.stanford.edu/archives/spr2020/entries/computational-philosophy/ (visited on 11/20/2021).

[24] John Harrison. "HOL Light: An Overview". In: *Theorem Proving in Higher Order Logics*. Ed. by Stefan Berghofer et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 60–66. ISBN: 978-3-642-03359-9. DOI: 10.1007/978-3-642-03359-9_4.

[25] Brian Huffman and Ondej Kunar. "Lifting and Transfer: A Modular Design for Quotients in Isabelle/HOL". In: *Certified Programs and Proofs*. Ed. by Georges Gonthier and Michael Norrish. Cham: Springer International Publishing, 2013, pp. 131–146. ISBN: 978-3-319-03545-1. DOI: 10.1007/978-3-319-03545-1_9.

[26] Maciej Janowicz et al. "Application of Automated Theorem-Proving to Philosophical Thought: Spinoza's Ethics". In: *Information Science and Applications 2017*. Ed. by Kuinam Kim and Nikolai Joukov. Singapore: Springer Singapore, 2017, pp. 512–518. ISBN: 978-981-10-4154-9. DOI: 10.1007/978-981-10-4154-9_59.

[27] Daniel Kirchner. *Embedding of Abstract Object Theory in Isabelle/HOL*. Full sources. URL: https://github.com/ekpyron/AOT/tree/dissertation (visited on 12/14/2021).

[28] Daniel Kirchner. *Embedding of Abstract Object Theory in Isabelle/HOL*. Variant counting equivalence classes of objects. URL: https://github.com/ekpyron/AOT/tree/classes (visited on 12/14/2021).

[29] Daniel Kirchner. "Representation and Partial Automation of the Principia Logico-Metaphysica in Isabelle/HOL". In: *Archive of Formal Proofs* (Sept. 2017). Formal proof development. ISSN: 2150-914x. URL: http://isa-afp.org/entries/PLM.html (visited on 11/20/2021).

[30] Daniel Kirchner, Christoph Benzmüller, and Edward N. Zalta. "Computer Science and Metaphysics: A Cross-Fertilization". In: *Open Philosophy* 2.1 (2019), pp. 230–251. DOI: 10.1515/opphil-2019-0015.

[31] Daniel Kirchner, Christoph Benzmüller, and Edward N. Zalta. "Mechanizing Principia Logico-Metaphysica in functional type-theory". In: *The Review of Symbolic Logic* 13.1 (2020), pp. 206–218. DOI: 10.1017/S1755020319000297.

[32] Saul A. Kripke. "Semantical Considerations on Modal Logic". In: *Acta Philosophica Fennica* 16 (1963), pp. 83–94.

[33] Hannes Leitgeb, Uri Nodelman, and Edward N. Zalta. *A Defense of Logicism*. URL: http://mally.stanford.edu/papers/logicism.pdf (visited on 12/14/2021). In preparation.

[34] W. McCune. "Prover9 and Mace4". 2005–2010. URL: http://www.cs.unm.edu/~mccune/prover9/ (visited on 11/20/2021).

[35] Elliott Mendelson. "The Propositional Calculus". In: *Introduction to Mathematical Logic*. Boston, MA: Springer US, 1987, pp. 10–40. ISBN: 978-1-4615-7288-6. DOI: 10.1007/978-1-4615-7288-6_2.

[36] Christopher Menzel. "The Proper Treatment of Predication in Fine-Grained Intensional Logic". In: *Philosophical Perspectives* 7 (1993), pp. 61–87. ISSN: 15208583, 17582245. DOI: 10.2307/2214116.

[37] Leonardo de Moura and Nikolaj Bjørner. "Z3: An Efficient SMT Solver". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by C. R. Ramakrishnan and Jakob Rehof. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340. ISBN: 978-3-540-78800-3. DOI: 10.1007/978-3-540-78800-3_24.

[38] Tobias Nipkow, Lawrence C. Paulson, and Makarius Wenzel. *A Proof Assistant for Higher-Order Logic*. URL: https://isabelle.in.tum.de/website-Isabelle2021-1/dist/Isabelle2021-1/doc/tutorial.pdf (visited on 12/14/2021).

[39] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. Vol. 2283. LNCS. Springer, 2002. DOI: 10.1007/3-540-45949-9.

[40] Steven Obua. "Partizan Games in Isabelle/HOLZF". In: *Theoretical Aspects of Computing - ICTAC 2006*. Ed. by Kamel Barkaoui, Ana Cavalcanti, and Antonio Cerone. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 272–286. ISBN: 978-3-540-48816-3. DOI: 10.1007/11921240_19.

[41] Paul E. Oppenheimer and Edward N. Zalta. "A Computationally-Discovered Simplification of the Ontological Argument". In: *Australasian Journal of Philosophy* 89.2 (2011), pp. 333–349. DOI: 10.1080/00048401003674482.

[42] Paul E. Oppenheimer and Edward N. Zalta. "On the Logic of the Ontological Argument". In: *Philosophical Perspectives* 5 (1991), pp. 509–529.

[43] Paul E. Oppenheimer and Edward N. Zalta. "Relations Versus Functions at the Foundations of Logic: Type-Theoretic Considerations". In: *Journal of Logic and Computation* 21 (2011), pp. 351–374. DOI: 10.1093/logcom/exq017.

[44] Lawrence Paulson. "Three Years of Experience with Sledgehammer, a Practical Link between Automatic and Interactive Theorem Provers". In: *PAAR-2010: Proceedings of the 2nd Workshop on Practical Aspects of Automated Reasoning*. Ed. by Renate A. Schmidt, Stephan Schulz, and Boris Konev. Vol. 9. EPiC Series in Computing. EasyChair, 2012, pp. 1–10. DOI: 10.29007/tnfd.

[45] Lawrence C. Paulson. *Isabelle's Logics*. URL: https://isabelle.in.tum.de/website-Isabelle2021-1/dist/Isabelle2021-1/doc/logics.pdf (visited on 12/14/2021).

[46] Francis J. Pelletier and Edward N. Zalta. “How to Say Goodbye to the Third Man”. In: *Noûs* 34.2 (2000), pp. 165–202. DOI: 10.1111/0029-4624.00207.

[47] William Joseph Rapaport. “Intentionality and the Structure of Existence”. PhD thesis. Indiana University, 1976. URL: https://cse.buffalo.edu/~rapaport/Papers/rapaport1976-PhD-Diss.pdf (visited on 11/20/2021).

[48] Stephan Schulz, Simon Cruanes, and Petar Vukmirovi. “Faster, Higher, Stronger: E 2.3”. In: *Proc. of the 27th CADE, Natal, Brasil.* Ed. by Pascal Fontaine. LNAI 11716. Springer, 2019, pp. 495–507. DOI: 10.1007/978-3-030-29436-6_29.

[49] Herbert A. Simon. *Models of My Life.* Cambridge: MIT Press, 1996. ISBN: 978-0-262-69185-7.

[50] The Coq Development Team. *The Coq Proof Assistant.* Version 8.14. Oct. 2021. DOI: 10.5281/zenodo.5704840.

[51] Christoph Weidenbach. “Combining Superposition, Sorts and Splitting”. In: *Handbook of Automated Reasoning.* Ed. by Alan Robinson and Andrei Voronkov. Handbook of Automated Reasoning. Amsterdam: North-Holland, 2001, pp. 1965–2013. ISBN: 978-0-444-50813-3. DOI: 10.1016/B978-044450813-3/50029-1.

[52] Makarius Wenzel. “Interaction with Formal Mathematical Documents in Isabelle/PIDE”. In: *Intelligent Computer Mathematics.* Ed. by Cezary Kaliszyk et al. Cham: Springer International Publishing, 2019, pp. 1–15. ISBN: 978-3-030-23250-4. DOI: 10.1007/978-3-030-23250-4_1.

[53] Makarius Wenzel. *The Isabelle System Manual.* URL: https://isabelle.in.tum.de/website-Isabelle2021-1/dist/Isabelle2021-1/doc/system.pdf (visited on 12/14/2021).

[54] Makarius Wenzel. *The Isabelle/Isar Implementation.* URL: https://isabelle.in.tum.de/website-Isabelle2021-1/dist/Isabelle2021-1/doc/implementation.pdf (visited on 12/14/2021).

[55] Makarius Wenzel. *The Isabelle/Isar Reference Manual.* URL: https://isabelle.in.tum.de/website-Isabelle2021-1/dist/Isabelle2021-1/doc/isar-ref.pdf (visited on 12/14/2021).

[56] Edward N. Zalta. *Abstract Objects: An Introduction to Axiomatic Metaphysics.* Synthese Library. Springer, 1983. ISBN: 9789027714749. DOI: 10.1007/978-94-009-6980-3.

[57] Edward N. Zalta. “Freges Theorem and Foundations for Arithmetic”. In: *The Stanford Encyclopedia of Philosophy.* Ed. by Edward N. Zalta. Fall 2021. Metaphysics Research Lab, Stanford University, 2021. URL: https://plato.stanford.edu/archives/fall2021/entries/frege-theorem/ (visited on 11/29/2021).

[58] Edward N. Zalta. *Further Explanation of the Objectives of the Theory.* URL: http://mally.stanford.edu/objectives.html (visited on 12/14/2021).

[59] Edward N. Zalta. *Intensional Logic and the Metaphysics of Intentionality.* A Bradford book. MIT Press, 1988. ISBN: 9780262240277.

[60] Edward N. Zalta. “Natural Numbers and Natural Cardinals as Abstract Objects: A Partial Reconstruction of Frege’s Grundgesetze in Object Theory”. In: *Journal of Philosophical Logic* 28.6 (1999), pp. 619–660. DOI: 10.1023/A:1004330128910.

[61] Edward N. Zalta. “Neo-Logicism? An Ontological Reduction of Mathematics to Metaphysics”. In: *Erkenntnis* 53.1 (Sept. 2000), pp. 219–265. ISSN: 1572-8420. DOI: 10.1023/A:1005614102033.

[62] Edward N. Zalta. *Principia Logico-Metaphysica.* [Draft/Excerpt; continuously updated current version]. URL: http://mally.stanford.edu/principia.pdf (visited on 11/20/2021).

[63] Edward N. Zalta. *Principia Logico-Metaphysica.* [Draft/Excerpt; dated October 13, 2021]. URL: http://mally.stanford.edu/principia-2021-10-13.pdf (visited on 11/20/2021).

[64] Edward N. Zalta. *Principia Logico-Metaphysica.* [Draft/Excerpt; dated October 28, 2016]. URL: https://mally.stanford.edu/principia-2016-10-28.pdf (visited on 11/20/2021).

[65] Edward N. Zalta. “Twenty-Five Basic Theorems in Situation and World Theory”. In: *Journal of Philosophical Logic* 22.4 (1993), pp. 385–428. DOI: 10.1007/BF01052533.

Zusammenfassung der Ergebnisse

Wir präsentieren die Implementierung einer metaphysischen Grundlagentheorie in einem automatischen Theorembeweiser mit Hilfe einer Erweiterung des Prinzips von *shallow semantic embeddings* (SSEs) in klassischer Logik höherer Stufe. Insbesondere kommen wir zu folgenden Ergebnissen:

- SSEs sind skalierbar und können nicht nur für die Analyse einzelner Argumente verwendet werden, sondern können auch auf komplette metaphysische Theorien angewendet werden, und deren Axiome und Deduktionssysteme präzise darstellen.
- Eine solche Implementierung ist kein rein technisches Unterfangen, sondern kann zu einem fruchtbaren Austausch führen, der in unserem Fall einerseits zu signifikanten Verbesserungen der analysierten Theorie geführt hat und andererseits neues Licht auf die technischen Möglichkeiten und Einschränkungen von SSEs werfen konnte.
- Es ist nicht nur möglich, die Logik eines komplexen Zielsystems technisch zu reproduzieren, sondern auch eine nahezu transparente Darstellung von Syntax und Beweisführung im Zielsystem zu erreichen, was einen effizienten und einfachen Austausch von Ergebnissen zwischen traditioneller Beweisführung von Hand und der computerbasierten Implementierung ermöglicht.
- Die Automatisierungsverfahren von Isabelle/HOL bleiben dabei erhalten und können dazu verwendet werden, Beweise im Zielsystem zu konstruieren, die allein den Deduktionsregeln des Zielsystems folgen. Dadurch erreichen wir effektiv einen dedizierten Theorembeweiser für unser Zielsystem auf der Grundlage einer verifizierbar konsistenten metalogischen Konstruktion.
- Unser Zielsystem *Abstract Object Theory* (AOT) selbst kann seinem Anspruch gerecht werden, eine philosophisch fundierte Konstruktion und Analyse der natürlichen Zahlen bieten zu können. Insbesondere können wir bestätigen, dass Frege's Konstruktion der natürlichen Zahlen in AOT konsistent reproduziert werden kann. Darüber hinaus konnten wir signifikant zur Weiterentwicklung der Konstruktion beitragen und können zusätzliche Erkenntnisse über die benötigten Axiome und über mögliche Varianten der Konstruktion beisteuern.

Interessanterweise stützen unsere Ergebnisse einerseits die Verwendung von klassischer Logik höherer Stufe als universale Metalogik, nachdem wir demonstrieren konnten, dass mit Hilfe der SSE Methode selbst herausfordernde logische Fundamentaltheorien präzise einbettbar sind, während wir andererseits die Position unseres Zielsystems AOT als metaphysische Fundamentaltheorie stärken, nachdem wir bestätigen können, dass es eine philosophisch fundierte Konstruktion mathematischer Objekte erlaubt. In diesem Zusammenhang bilden die Implementierung und Analyse des vollen typen-theoretischen Systems höherer Ordnung von AOT mit Hilfe der SSE Methode, sowie die Analyse der relativen Stärke dieses Systems im Vergleich zu HOL und ZF faszinierende Themen für zukünftige Forschungsarbeit.