2. Das Basismodell der Diskreten-Ereignis-Simulation

In der Simulation hat sich seit langer Zeit in der Forschung das Konzept der Diskreten-Ereignis-Simulation (eigentlich: diskrete ereignisorientierte Simulation) etabliert, bei der sich der Zustand beim Auftreten von Ereignissen sprunghaft ändert, während er im Zeitraum zwischen zwei Ereignissen konstant bleibt. Ändert man in der Simulation den Zustand gemäß des Auftretens eines Ereignisses, so spricht man auch von der *Ausführung* des Ereignisses. Man betrachtet eine Ereignismenge, in der sich alle noch nicht ausgeführten Ereignisse befinden. In einem Ausführungsschritt wird das Ereignis aus der Ereignismenge mit dem frühesten Zeitpunkt ausgewählt und ausgeführt, wodurch sich nicht nur der Zustand ändern kann sondern außerdem neue Ereignisse in die Ereignismenge eingefügt werden. Bei solchen Ereignissen spricht man von *Folgeereignissen* des ausgeführten Ereignisses. Das Ereignis, dessen Folgeereignisse die neuen Ereignisse sind, wird auch als das *erzeugende Ereignis* der Folgeereignisse bezeichnet. Die Diskrete-Ereignis-Simulation zeichnet sich durch ihre klare verständliche Struktur aus, die es auch ermöglicht, eine einfache mathematische Formalisierung zu finden.

In Abschnitt 2.1 wird das in der Praxis sehr heterogene Konzept der Diskreten-Ereignis-Simulation präzisiert und mathematisch formalisiert. Abschnitt 2.2 enthält ein Beispiel für ein mit Hilfe der Diskreten-Ereignis-Simulation spezifiziertes Beispiel: eine Warteschlange in einem Supermarkt. In Abschnitt 2.3 gibt es ein weiteres Beispiel, bei dem sich zwei Fahrstühle in einem Aufzugsschacht befinden.

2.1 Formalisierung

In der Diskreten-Ereignis-Simulation werden die Ereignisse erfasst, bei denen sich der Zustand[6] des simulierten Systems ändert. Zwischen zwei Ereignissen wird demnach der Zustand als konstant angenommen, während ein Ereignis ein System von einem Zustand in einen anderen überführt. Ausgeführt wird immer das Ereignis, das zeitlich als nächstes folgt.

Der Zustand Z eines Simulationssystems[7] in der Diskreten-Ereignis-Simulation besteht aus:

- dem Zustand S des simulierten Systems
- der aktuellen Simulationszeit c
- der Ereignismenge E, wobei nur alle zukünftigen, noch nicht ausgeführten Ereignisse enthalten sind.

Z lässt sich somit als Tupel (S, c, E) auffassen. Ein Simulationssystem hat einen Startzustand Z_0 = (S_0, c_0, E_0) oder eine Initialisierungsfunktion, die einen vom Zufall abhängigen Startzustand bestimmt.

Ein Ereignis e besteht aus:

- einem Ausführungszeitpunkt t
- einem Ereignistyp τ
- einer Menge P = { p_1, p_2, .. , p_n } von Parametern, wobei Anzahl und Typ der Parameter von τ abhängig sind.

Jedes Ereignis e ließe sich somit als Tupel (t, τ, P) auffassen. Einfacher lesbar ist aber die Darstellung τ(p_1, p_2, .. , p_n)@t, die deshalb im Folgenden verwendet wird. Hat ein Nachrichtentyp keine Parameter, also ist n = 0, so sind die die Parameter umschließenden Klammern optional.

---

[6] Den Zustand eines Systems kann man z.B. als Menge von diskreten und stetigen Größen charakterisieren. Eine andere Möglichkeit ist eine Menge von Zustandsaussagen, das sind Prädikate mit Zustandsattributen.

[7] Man muss hier den Zustand des Simulationssystems vom Zustand des simulierten Systems unterscheiden. Der Zustand des simulierten Systems ist Bestandteil des Zustandes des Simulationssystems.

Außerdem gibt es eine Transitionsfunktion f, die der Ausführung eines Ereignisses entspricht. Sie ist abhängig vom aktuellen Systemzustand S und vom Ereignis e, verändert den Systemzustand und erzeugt Folgeereignisse. Dabei ist f (S, e) = (S', E'), wobei S' den neuen Zustand des Systems und E' die Menge der erzeugten Folgeereignisse darstellt. Denkbar ist auch, dass f eine randomisierte Funktion ist.

Die Transitionsfunktion f hat somit den Typ f: (State × Event) $\rightarrow$ (State × 2^{Event}), wobei *State* die Menge aller möglichen Zustände und *Event* die Menge aller möglichen Ereignisse ist.

Es ist sehr schwierig, die komplette Transitionsfunktion f explizit in geschlossener Form anzugeben. Deshalb besteht sie fast immer aus mehreren Funktionsdefinitionen, um eine Fallunterscheidung vorzunehmen. Eine einzelne Funktionsdefinition der Form f (S, e) = (S', E'), bei der S und S' konkrete Zustände, e ein konkretes Ereignis und E' eine konkrete Ereignismenge ist, wird auch Regel[8] genannt und kann der Übersichtlichkeit halber auch in folgender tabellarischer Form dargestellt werden:

Ereignis	**e**
alter Zustand	**S**
neuer Zustand	**S'**
Folgeereignisse	**E'**

Bei der Darstellung der Ereignisausführungen gilt immer die am weitesten oben stehende Regel, wenn mehrere Regeln passen.[9] Dies setzt voraus, dass eine totale Ordnung auf allen zu einem Ereignistyp definierten Regeln existiert. Regeln so zu formulieren, dass mehrere Regeln für die Ausführung des aktuellen Ereignisses in Frage kommen, kann Vor- und Nachteile haben. Diese werden in Abschnitt 3.3.2 näher erläutert.

Es ergibt sich eine Zustandsüberführungsfunktion δ, die das Simulationssystem von einem Zustand Z in einen Folgezustand Z' überführt. Das Simulationssystem ist somit ein Transitionssystem.

Dabei ist δ (S, c, E) = (S', t, (E \ { e_{min} }) $\cup$ E'), wobei $e_{min} = \tau(p_1, p_2, .., p_n)@t \in E$ ein Ereignis mit minimalem Ausführungszeitpunkt[10] t ist und (S', E') = f (S, e_{min}) gilt. Das heißt in Worten, dass ein nächstes Ereignis e_{min} gewählt wird und auf dieses Ereignis und den alten Systemzustand die Transitionsfunktion f angewendet wird. Der Systemzustand wird auf den durch den Funktionsaufruf ermittelten Wert gesetzt, e_{min} wird aus der Ereignismenge entfernt und die ermittelten Folgeereignisse werden hinzugefügt. Hierbei fällt auf, dass die aktuelle Simulationszeit c nicht in die Zustandsüberführungsfunktion δ eingeht. Trotzdem ist es sinnvoll, jeweils die aktuelle Simulationszeit zu ermitteln und in den Zustand des Simulationssystems aufzunehmen, da es Teil des Ergebnisses einer Simulationsstudie ist, über welche Zeitdauer sich das simuliere System in welchem Zustand befunden hat.

Beendet ist die Simulation, wenn die Ereignismenge E leer ist. Oftmals soll ein Simulationssystem nach einer gewissen Simulationszeit t_{max} beendet werden. Dies lässt sich, wenn man keinen zusätzlichen Formalismus für diese Art der Beendigung der Simulation einführen will, dadurch erreichen, dass man die Transitionsfunktion f so definiert, dass alle Ereignisse mit einem

---

[8] Man kann hier auch das Wort *Reaktionsregel* verwenden, da das Simulationssystem auf ein Ereignis reagiert. Als Abgrenzung zu den Reaktionsregeln in der Objektbasierten und Agentenbasierten Simulation, die in den Kapiteln 3 und 4 eingeführt werden, wird hier jedoch das Wort *Regel* verwendet.

[9] Dass mehrere Regeln in Frage kommen, ist möglich, wenn die Beschreibung des alten Zustands mehrere mögliche konkrete alte Zustände zulässt.

[10] Haben mehrere Ereignisse den gleichen Ausführungszeitpunkt, so wird eines von ihnen gewählt. Nach welchen Kriterien die Auswahl geschieht, bleibt hier bewusst unspezifiziert. Jedoch sollte in konkreten Systemen eine Auswahlregel festgelegt sein, damit Simulationsergebnisse reproduzierbar sind.

Ereigniszeitpunkt nach t_{max} den Zustand des Systems unverändert lassen und keine Folgeereignisse definieren.

2.2 Beispiel: Warteschlange im Supermarkt

Eine Warteschlange in einem Supermarkt ist ein häufig betrachtetes Beispiel für die Diskrete-Ereignis-Simulation.

In Abschnitt 2.2.1 wird zunächst das Problem beschrieben und in Abschnitt 2.2.2 werden Ereignistypen und Systemzustand vorgestellt. In den Abschnitten 2.2.3 bis 2.2.5 werden die Ereignisausführungen der einzelnen Ereignistypen beschrieben und Abschnitt 2.2.6 enthält Abschlussbemerkungen zum Supermarktbeispiel.

2.2.1 Problembeschreibung

In einem Supermarkt mit n Kassen stellen sich ankommende[11] Kunden bei einer der Kassen an und warten, bis sie dort bedient werden. Die Zwischenankunftszeit, das ist die Zeit zwischen der Ankunft zweier Kunden sei exponentialverteilt mit Mittelwert μ. Die Bedienungszeit eines Kunden an einer Kasse sei gleichverteilt im Intervall [a , b]. Jede Kasse hat eine eigene von den anderen Kassen getrennte Schlange. Ein neu ankommender Kunde geht zu einer freien Kasse, wenn eine solche existiert. Sind mehrere Kassen frei, so geht er zu der freien Kasse mit der kleinsten Nummer. Ist keine Kasse frei, so stellt er sich bei der kürzesten Schlange an. Bei gleicher Länge wird die Schlange der Kasse mit der kleinsten Nummer gewählt.[12] Bei Ermittlung der Länge einer Schlange zählt der Kunde, der gerade bedient wird, nicht zur Schlange. Nach t_{max} Zeiteinheiten ($t_{max} > 0$) wird der Eingang des Supermarkts geschlossen. Es stellen sich dann keine weiteren Kunden mehr an, aber alle Kunden, die sich schon angestellt haben, werden noch bedient.

2.2.2 Ereignistypen und Systemzustand

Zu unterscheiden sind hier die drei Ereignistypen *Kunde_kommt_an*, *Bedienung_beendet* und *Eingang_wird_geschlossen*. Ein Ereignis vom Typ *Kunde_kommt_an* stellt dabei die Ankunft eines neuen Kunden, ein Ereignis vom Typ *Bedienung_beendet* die Beendigung der Bedienung eines Kunden an einer Kasse und ein Ereignis vom Typ *Eingang_wird_geschlossen* die Schließung des Eingangs dar.

Den Zustand des Systems kann man durch die Länge[13] l_i der Warteschlange i (i = 1 .. n), den Belegungszustand s_i der Kasse i (kann die Werte *frei* oder *belegt* annehmen) sowie den Wert *Eingang* (kann die Werte *offen* oder *geschlossen* annehmen), der besagt ob der Eingang geschlossen wurde, beschreiben. Länge und Belegungszustand einer Kasse kann man durch das Prädikat[14] *Kasse (i,* l_i, s_i *)* beschreiben, den Eingang durch das Prädikat *Eingang (s)*.

Für den Zustand

S = { Kasse (1, l_1 , s_1), .. , Kasse (n, l_n , s_n), Eingang (s) }

---

[11] Aus Gründen der Einfachheit wird hier nicht zwischen Ankunft im Supermarkt und Anstellen an der Kasse unterschieden, d.h. ankommende Kunden halten sich nicht in Verkaufsräumen auf, sondern stellen sich sofort an einer Kasse an. Strenggenommen entspricht das somit eher der Simulation eines Tante-Emma-Ladens, einer Theaterkasse oder einer Postfiliale.

[12] Auf Gründen der Einfachheit wird hier davon ausgegangen, dass der Kunde die Schlange, an die er sich einmal angestellt hat, nicht wechselt, auch wenn zwischendurch die Schlange einer anderen Kasse kürzer wird, als die in der er sich befindet. Selbst, wenn eine andere Kasse frei wird, wechselt er die einmal gewählte Schlange nicht.

[13] Wenn man nur die Längen der Warteschlangen betrachtet und nicht, welche Kunden sich in ihr befinden, so werden bestimmte Auswertungen schwierig, wie etwa die Antwort auf die Frage "Wie lang war die längste Wartezeit eines Kunden?". Aus Gründen der Einfachheit wird trotzdem hier so verfahren.

[14] Hier werden Prädikate anstelle von Zustandsvariablen verwendet, da sie eine strukturiertere, prägnantere und damit besser lesbare Darstellung erlauben.

ergibt sich in natürlicher Weise der Startzustand

S_0 = { Kasse (1, 0, frei), .. , Kasse (n, 0, frei), Eingang (offen) },

denn am Anfang sind alle Warteschlangen leer, die Kassen frei und der Eingang ist noch nicht geschlossen. Gestartet wird zum Zeitpunkt $c_0 = 0$. Die initiale Ereignismenge E_0 besteht aus zwei Ereignissen, der Ankunft des ersten Kunden und dem Schließen des Eingangs, also

E_0 = { Kunde_kommt_an ( ) @ (c_0 + Expo_ZV(μ)), Eingang_wird_geschlossen ( ) @ t_{max} },

wobei Expo_ZV(μ) eine exponentialverteilte Zufallsvariable[15] mit Mittelwert μ erzeugt.

2.2.3 Ereignisausführung Kunde_kommt_an

Das Ereignis *Kunde_kommt_an* stellt die Ankunft eines neuen Kunden dar. *Kunde_kommt_an* kommt ohne Parameter aus, hat aber, wie jedes Ereignis, einen Ereigniszeitpunkt.

Kommt ein neuer Kunde an und ist der Eingang noch nicht geschlossen, so geht er zu der freien Kasse mit der kleinsten Nummer, falls eine solche existiert. Es gilt folgende Regel:

Ereignis	**Kunde_kommt_an ( ) @ t**
alter Zustand	**{ .. , Kasse (i, 0, *frei*), .. , Eingang (*offen*)},** **wobei i = min { j | Kasse (j, 0, *frei*)}**
neuer Zustand	**{ .. , Kasse (i, 0, *belegt*), .. , Eingang (*offen*)}**
Folgeereignisse	**{ Kunde_kommt_an ( ) @ (t + Expo_ZV(μ)),** **Bedienung_beendet (i) @ (t + Gleich_ZV(a, b)) }**

Regel 1: Kunde_kommt_an.1

Gleich_ZV(a, b) erzeugt eine gleichverteilte Zufallsvariable[16] im Intervall [a , b]. Die Variable i stellt dabei den Index der kleinsten freien Kasse dar. Der Zustand ändert sich nur bei s_i, das auf *belegt* gesetzt wird. Erzeugt werden zwei Folgeereignisse, die Ankunft des nächsten Kunden[17] und das Ereignis, dass die Bedienung an Kasse i, die der Kunde jetzt belegt, später beendet sein wird.

Zur Verdeutlichung:

Die tabellarische Darstellung entspricht dabei der nachfolgenden Regel.

f ({ .., Kasse (i, 0, *frei*), .. , Eingang (*offen*)}, Kunde_kommt_an ( ) @ t) =
({ .., Kasse (i, 0, *belegt*), .. , Eingang (*offen*)},
{ Kunde_kommt_an ( ) @ (t + Expo_ZV(μ)),
Bedienung_beendet (i) @ (t + Gleich_ZV(a, b)) }),
wobei i = min { j \| Kasse (j, 0, *frei*)}

[15] Eine exponentialverteilte Zufallsvariable mit Mittelwert μ lässt sich mit der Formel $-\mu$ *ln (Gleich_ZV())* erzeugen, wobei *Gleich_ZV()* eine im Intervall [0 , 1] gleichverteilte Zufallsvariable ist.

[16] Eine im Intervall [a , b] gleichverteilte Zufallsvariable lässt sich mit der Formel *a + (b – a) (Gleich_ZV())* erzeugen, wobei *Gleich_ZV()* eine im Intervall [0 , 1] gleichverteilte Zufallsvariable ist.

[17] Eigentlich stellt die Ankunft des nächsten Kunden logisch kein Folgeereignis der Ankunft eines Kunden dar. Vielmehr handelt es sich um ein (zufalls-)periodisch auftretendes Ereignis, bei dem mehrere Auftreten des Ereignisses nichts miteinander zu tun haben. In der Diskreten-Ereignis-Simulation modelliert man bei periodischen Ereignissen oft das nächste Auftreten eines Ereignisses als Folgeereignis des aktuellen Ereignisses, da die Periodizität so technisch leicht handhabbar ist.

Kommt ein neuer Kunde an, ist der Eingang noch nicht geschlossen und ist keine Kasse frei, stellt sich der neue Kunde bei der kürzesten Schlange an, wobei bei gleicher Länge die Schlange der Kasse mit der kleinsten Nummer gewählt wird.[18] Dabei stellt i den Index der Kasse mit der kürzesten Schlange dar. Der Zustand ändert sich nur bei l_i, das um 1 erhöht wird. Erzeugt wird ein Folgeereignis, die Ankunft des nächsten Kunden.

Ereignis	**Kunde_kommt_an ( ) @ t**
alter Zustand	**{ Kasse (1,** l_1 **,** ***belegt*** **), .. , Kasse (i,** l_i **,** ***belegt*** **), .. ,** **Kasse (n,** l_n **,** ***belegt*** **), Eingang (** ***offen*** **) },** **wobei i = min { j |** $\forall$ **k (1≤k≤n):** $l_j \leq l_k$ **)}**
neuer Zustand	**{ Kasse (1,** l_1 **,** ***belegt*** **), .. , Kasse (i,** l_i **+1,** ***belegt*** **), .. ,** **Kasse (n,** l_n **,** ***belegt*** **), Eingang (** ***offen*** **) }**
Folgeereignisse	**{ Kunde_kommt_an ( ) @ (t + Expo_ZV(**μ**)) }**

Regel 2: Kunde_kommt_an.2

Ist der Eingang bereits geschlossen, so wird der neu ankommende Kunde nicht mehr bedient. Der Systemzustand bleibt gleich, es werden keine Folgeereignisse erzeugt.

Ereignis	**Kunde_kommt_an ( ) @ t**
alter Zustand	**{ Kasse (1,** l_1 **,** s_1 **), .. , Kasse (n,** l_n **,** s_n **),** **Eingang (** ***geschlossen*** **) }**
neuer Zustand	**{ Kasse (1,** l_1 **,** s_1 **), .. , Kasse (n,** l_n **,** s_n **),** **Eingang (** ***geschlossen*** **) }**
Folgeereignisse	**Ø**

Regel 3: Kunde_kommt_an.3

2.2.4 Ereignisausführung Bedienung_beendet

Das Ereignis *Bedienung_beendet* stellt die Beendigung der Bedienung eines Kunden an einer Kasse dar. Als Parameter hat *Bedienung_beendet* die Nummer der Kasse.

Ist die Bedienung eines Kunden an der Kasse abgeschlossen und ist die Warteschlange nicht leer, so kommt der nächste Kunde aus der Warteschlange an die Kasse. Die Länge der Warteschlange der Kasse verkürzt sich um 1. Erzeugt wird ein Folgeereignis: das Ereignis, dass die Bedienung an Kasse i, die der erste Kunde aus der Warteschlange jetzt belegt, später beendet sein wird.

Ereignis	**Bedienung_beendet (i) @ t**
alter Zustand	**{ .., Kasse (i, l+1 ,** ***belegt*** **), .. , Eingang (s)}**
neuer Zustand	**{ .., Kasse (i, l ,** ***belegt*** **), .. , Eingang (s)}**
Folgeereignisse	**{ Bedienung_beendet (i) @ (t + Gleich_ZV(a, b))}**

Regel 4: Bedienung_beendet.1

Ist die Warteschlange hingegen leer, so wird die Kasse jetzt frei. Es wird in diesem Fall kein Folgeereignis erzeugt.

[18] Im Folgenden wird aus Gründen der sprachlichen Einfachheit bei gleicher Schlangenlänge die Schlange der Kasse mit der kleinsten Nummer als *die kürzeste Schlange* bezeichnet.

Ereignis	**Bedienung_beendet (i) @ t**
alter Zustand	**{ .., Kasse (i, 0 , *belegt*), .. , Eingang (s)}**
neuer Zustand	**{ .., Kasse (i, 0 , *frei*), .. , Eingang (s)}**
Folgeereignisse	**Ø**

Regel 5: Bedienung_beendet.2

2.2.5 Ereignisausführung Eingang_wird_geschlossen

Das Ereignis *Eingang_wird_geschlossen* stellt die Schließung des Eingangs dar. Von diesem Zeitpunkt an können sich keine weiteren Kunden mehr an einer Kasse anstellen. *Eingang_wird_geschlossen* hat keine Parameter.

Beim Ereignis *Eingang_wird_geschlossen* wird der Wert des Prädikats *Eingang* auf *geschlossen* gesetzt, ansonsten bleibt der Zustand unverändert. Es wird kein Folgeereignis erzeugt.

Ereignis	**Eingang_wird_geschlossen ( ) @ t**
alter Zustand	**{ Kasse (1,** l_1 **,** s_1 **), .. , Kasse (n,** l_n **,** s_n **),** **Eingang (*offen*) }**
neuer Zustand	**{ Kasse (1,** l_1 **,** s_1 **), .. , Kasse (n,** l_n **,** s_n **),** **Eingang (*geschlossen*)}**
Folgeereignisse	**Ø**

Regel 6: Eingang_wird_geschlossen.1

2.2.6 Abschlussbemerkungen

In diesem Abschnitt werden einige abschließende Bemerkungen für das Supermarkt-Beispiel und seiner Simulation vorgenommen. Es wird gezeigt, dass das Simulationssystem terminiert, es wird die Problematik gleichzeitiger Ereignisse behandelt ein Beweis erbracht, dass bei einer freien Kasse die Länge der Warteschlange immer 0 ist.

Termination des Systems

Das System terminiert, da nach Abarbeitung des Ereignisses *Eingang_wird_geschlossen* keine neuen Ereignisse vom Typ *Kunde_kommt_an* oder *Eingang_wird_geschlossen* erzeugt werden. Es werden zwar noch Ereignisse vom Typ *Bedienung_beendet* gemäß Regel *Bedienung_beendet.1* (Regel 4) erzeugt, aber nur, solange ein $l_i > 0$ ist. Da bei jeder Erzeugung eines Ereignisses vom Typ *Bedienung_beendet* gemäß Regel *Bedienung_beendet.1* (Regel 4) ein l_i dekrementiert wird, kann es höchstens $\sum_{i=1}^{n} l_i$ solcher Ereignisse nach Abarbeitung des Ereignisses *Eingang_wird_geschlossen* geben.

Es bleibt noch zu zeigen, dass das Ereignis *Eingang_wird_geschlossen* überhaupt erreicht wird, also dass es nur endlich viele Ereignisse vor diesem Ereignis gibt. Da bei jeder Erezeugung eines Ereignisses vom Typ *Bedienung_beendet* gemäß Regel *Bedienung_beendet.1* (Regel 4) ein l_i um 1 dekrementiert wird, alle l_i anfangs den Wert 0 haben, kein l_i unter 0 sinken kann und in keiner Regel ein l_i um mehr als 1 inkrementiert wird, kann es nur so viele Ereignisse vom Typ *Bedienung_beendet*, die gemäß Regel *Bedienung_beendet.1* (Regel 4) erzeugt wurden, geben, wie es Ereignisse gibt, bei denen ein l_i um 1 inkrementiert wird. Das sind aber nur genau die Ereignisse vom Typ *Kunde_kommt_an*, bei denen Regel *Kunde_kommt_an.2* (Regel 2) angewendet wird. Jedes Ereignis vom Typ *Bedienung_beendet*, das gemäß Regel *Bedienung_beendet.1* (Regel 4) erzeugt wird, lässt sich also einem Ereignis vom Typ *Kunde_kommt_an*, bei dem Regel *Kunde_kommt_an.2* (Regel 2) angewendet wird zuordnen. Jedes Ereignis vom Typ *Bedienung_beendet*, das gemäß Regel

Kunde_kommt_an.1 (Regel 1) erzeugt wird, lässt sich einem Ereignis vom Typ *Kunde_kommt_an*, bei dem Regel *Kunde_kommt_an.1* (Regel 1) angewendet wird, zuordnen, nämlich genau ihrem erzeugenden Ereignis. Somit lässt sich jedes Ereignis vom Typ *Bedienung_beendet* einem Ereignis vom Typ *Kunde_kommt_an* zuordnen.[19] Somit müsste nur noch gezeigt werden, dass es nur endlich viele Ereignisse vom Typ *Kunde_kommt_an* geben kann. Dies lässt sich leider nicht beweisen, da man aufgrund der Exponentialverteilung keine untere Schranke für die Zeit zwischen der Ankunft zweier Kunden bzw. keine obere Schranke für die Anzahl der Kunden in einem Zeitintervall finden kann. Allerdings ist der Erwartungswert für die Anzahl der Kunden im Zeitintervall [0 , t_{max}] endlich, nämlich t_{max} / μ und die Wahrscheinlichkeit, dass die Zahl der Kunden unendlich ist, ist Null[20]. In der praktischen Anwendung kann man die Termination des Simulationssystems somit als gesichert annehmen.

Gleichzeitige Ereignisse

Auch wenn es aufgrund der Erzeugung von kontinuierlichen Zufallsvariablen unwahrscheinlich ist, kann es trotzdem zwei oder mehr[21] Ereignisse geben, die zum gleichen Zeitpunkt stattfinden. Welches Ereignis in diesem Fall zuerst ausgeführt werden soll, ist nicht spezifiziert. Der Zustand des Systems nach der Ausführung beider Ereignisse kann von der Reihenfolge der Ausführung abhängig sein. Handelt es sich um Ereignisse vom Typ *Kunde_kommt_an* und *Bedienung_beendet*, so kann es sein, dass abhängig davon, ob das Ereignis *Bedienung_beendet* zuerst ausgeführt wird oder nicht, der Kunde sich an diese oder an eine andere Kasse anstellt, nämlich dann, wenn die Schlange der Kasse, die um 1 kürzer wird, dadurch zur kürzesten wird. Wird allerdings vom Kunden im konkreten Fall dieselbe Kasse gewählt, egal ob sich die Schlange einer Kasse schon um 1 verkürzt hat oder nicht[22], ist also die Wahl der Kasse von der Ausführungsreihenfolge unabhängig, so besteht in der Simulation ein sich je nach Ausführungsreihenfolge unterscheidender Zwischenzustand für 0 Zeiteinheiten, hat demnach keine Relevanz. Handelt es sich um die Ereignisse *Eingang_wird_geschlossen* und *Kunde_kommt_an*, so ist die Ausführungsreihenfolge auf jeden Fall entscheidend, denn von ihr ist abhängig, ob der neue Kunde noch bedient wird oder nicht. In allen anderen Fällen von gleichzeitigen Ereignissen (der Fall zweier Ereignisse vom Typ *Bedienung_beendet* und der Fall von Ereignissen *Bedienung_beendet* und *Eingang_wird_geschlossen*) besteht in der Simulation ein sich je nach Ausführungsreihenfolge unterscheidender Zwischenzustand nur für 0 Zeiteinheiten und hat damit keine Relevanz. In allen Fällen verhält sich das System unabhängig von der Wahl der Ausführungsreihenfolge wohldefiniert und sinnvoll.

Länge 0 bei freien Kassen

Es kann nicht vorkommen, dass es eine freie Kasse gibt, bei der die Warteschlange nicht die Länge 0 hat, somit gilt also:

Kasse (i, l_i , frei) $\Rightarrow$ $l_i = 0$.

Beweis:

Im Anfangszustand ist die Implikation wahr, da dort alle l_i den Wert 0 haben. Zu zeigen ist, dass die Implikation immer wahr bleibt, solange sie einmal wahr ist. Denkbar wären zwei Möglichkeiten, dass die Implikation zum ersten Mal verletzt werden könnte:

---

[19] Intuitiv bedeutet das nichts weiter als, dass jeder Kunde, dessen Bedienung an der Kasse beendet ist, irgendwann vorher im Supermarkt angekommen sein muss.

[20] Das bedeutet nicht, dass es sich um ein unmögliches Ereignis handeln muss. Zwar hat jedes unmögliche Ereignis die Wahrscheinlichkeit 0, aber nicht jedes Ereignis mit Wahrscheinlichkeit 0 ist unmöglich.

[21] Da der Fall von mehr als zwei gleichzeitigen Ereignissen noch viel unwahrscheinlicher ist, wird hier auf eine genaue Betrachtung verzichtet. Allerdings lassen sich die Überlegungen für zwei gleichzeitige Ereignisse leicht auf mehr als zwei gleichzeitige Ereignisse erweitern.

[22] Hier ist sowohl der Fall möglich, dass sich der Kunde an die Kasse stellt, deren Schlange sich um 1 verkürzt als auch der Fall, dass der Kunde sich an eine andere Schlange stellt. Im ersten Fall ist die Schlange auch ohne die Verkürzung um 1 die kürzeste, im zweiten Fall ist eine andere Schlange trotz der Verkürzung um 1 immer noch kürzer.

a) Die Kasse erhöht ihre Länge auf einen Wert größer 0 und bleibt im Zustand *frei*.

b) Die Kasse ändert ihren Zustand von *belegt* auf *frei*, ohne dass die Länge danach 0 ist.

Die einzige Stelle, an der eine Länge erhöht wird, ist ein Übergang von *Kasse (i,* l_i*, belegt)* zu *Kasse (i,* l_i*+1, belegt)* gemäß Regel *Kunde_kommt_an.2* (Regel 2). Damit ist Fall a) nicht möglich, da dort der (alte und) neue Zustand *belegt* ist.

Die einzige Stelle, an der der Zustand von *belegt* auf *frei* wechselt, ist ein Übergang von *Kasse (i, 0 , belegt)* zu *Kasse (i, 0 , frei)* gemäß Regel *Bedienung_beendet.2* (Regel 5). Damit ist auch Fall b) nicht möglich, da dort im (alten und) neuen Zustand die Länge 0 ist. Somit ist gezeigt, dass bei einer freien Kasse die Warteschlange immer die Länge 0 haben muss. ■

2.3 Beispiel: Fahrstühle

Als weiteres Beispiel werden hier zwei Fahrstühle, die sich einen Schacht teilen, vorgestellt. Das Beispiel ist angelehnt an Wagner [Wag97a]. Es gibt weiterführende Forschung im Bereich der Planung von komplexen Fahrstuhlsystemen (z.B. [KS00]). In dieser Arbeit soll das Fahrstuhl-Beispiel jedoch lediglich ein einfaches Szenario zur Illustration der Basisbegriffe der vorgestellten Simulationsparadigmen sein. Deshalb wird hier auf diese weiterführenden Forschungsergebnisse nicht eingegangen.

In Abschnitt 2.3.1 wird zunächst das Problem beschrieben, dann in Abschnitt 2.3.2 eine algorithmische Lösung für das Problem angegeben und in Abschnitt 2.3.3 werden Ereignistypen und Systemzustand für das Fahrstuhlsystem vorgestellt. In den Abschnitten 2.3.4 bis 2.3.9 werden die Ereignisausführungen der einzelnen Ereignistypen beschrieben und Abschnitt 2.3.10 enthält Abschlussbemerkungen zum Fahrstuhlbeispiel.

2.3.1 Problembeschreibung

Zwei (Personen-)Fahrstühle bewegen sich im selben Schacht und müssen eine Kollision verhindern. Der Schacht erstreckt sich über drei Etagen, wobei Fahrstuhl 1 die Etagen 1 und 2 und Fahrstuhl 2 die Etagen 2 und 3 bedient.[23] Abbildung 5 verdeutlicht die Situation grafisch.

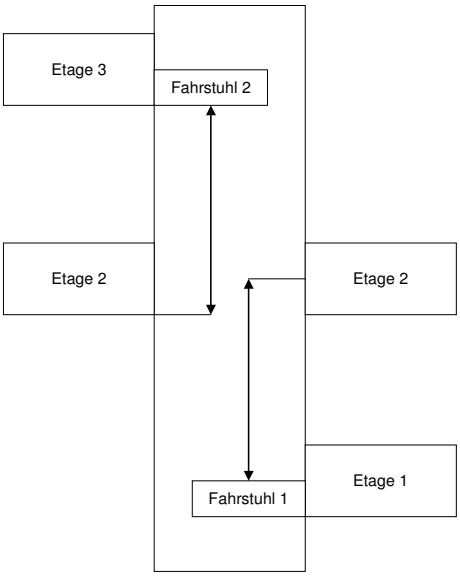

Abbildung 5: Zwei Fahrstühle in einem Schacht

Etage 1 wird als Heimatetage von Fahrstuhl 1 und Etage 3 als Heimatetage von Fahrstuhl 2 bezeichnet. Eine Kollision kann nur in Etage 2 passieren, was die Fahrstühle durch geeignete Kommunikation verhindern müssen. Dabei wird von einer nachrichtenbasierten Kommunikation ausgegangen. Bei einem Kommunikationsvorgang dauert es Δ Zeiteinheiten, ehe die Nachricht vom

[23] Eine solche Topologie ist sinnvoll, wenn kein Fahrgast von Etage 1 nach Etage 3 fahren will, z.B., wenn sich in Etage 1 und Etage 3 Parkflächen befinden und in Etage 2 ein Einkaufszentrum.

Absender zum Empfänger gelangt ist. Keine Nachricht geht verloren und die Nachrichten kommen beim Empfänger in der selben Reihenfolge an, in der der Sender sie abgeschickt hat.

Personen, die mit einem Fahrstuhl befördert werden möchten, können ihn von außen anfordern, worauf der Fahrstuhl irgendwann[24] zu der Etage fährt, von der aus die Person ihn angefordert hat, die Person aufnimmt und sie zur anderen Etage fährt. Die Zeit zwischen zwei Anforderungen von Etage 1 nach Etage 2 sei exponentialverteilt mit Mittelwert $\mu_{1,1}$[25]. Analog sei die Zeit zwischen zwei Anforderungen von Etage 2 nach Etage 1 exponentialverteilt mit Mittelwert $\mu_{1,2}$, von Etage 2 nach Etage 3 mit Mittelwert $\mu_{2,2}$ und von Etage 3 nach Etage 2 mit Mittelwert $\mu_{2,3}$. Die Aufnahmekapazität der Fahrstühle sei hier aus Gründen der Einfachheit als nicht beschränkt angenommen, so dass alle aufgelaufenen Anforderungen auf einmal bedient, d.h. alle wartenden Personen auf einmal befördert werden können, wenn der Fahrstuhl die entsprechende Etage erreicht. Eine Anforderung wird bedient, indem der Fahrstuhl zur Ausgangsetage der Anforderung fährt. Dort dauert es maximal c_1 Zeiteinheiten, bis die Person, die den Fahrstuhl angefordert hat, eingestiegen ist. Auch wenn mehrere Personen einsteigen und auch wenn vorher noch Personen aussteigen, dauert es nur maximal c_1 Zeiteinheiten. Nach Ablauf dieser Zeit fährt der Fahrstuhl in die Zieletage. Eine Anforderung von Etage i nach Etage j setzt sich demnach aus Sicht des Fahrstuhls aus einer Anforderung von außen zu Etage i und einer Anforderung von innen zu Etage j zusammen[26]. Die Fahrzeit eines Fahrstuhls von einer Etage zur nächsten sei für beide Fahrstühle in beiden Richtungen beladen und unbeladen gleichverteilt im Intervall [a , b].

Die Fahrstühle können sich auf- und abwärts bewegen und erfahren dann (über Sensoren), wenn sie an ihrem Ziel angekommen sind. Außerdem werden sie über eingehende Anforderungen sowie über eingehende Nachrichten (vom anderen Fahrstuhl) informiert.

2.3.2 Algorithmische Lösung

Um zu verhindern, dass beide Fahrstühle sich gleichzeitig in Etage 2 befinden und es so zu einer Kollision kommen kann, muss ein Fahrstuhl, der in Etage 2 fahren möchte, den anderen um Erlaubnis fragen. Erhält er die Erlaubnis, so kann er ohne Gefahr in Etage 2 fahren.

Um Konflikte zu lösen, in denen beide Fahrstühle in Etage 2 fahren wollen, ist zu jedem Zeitpunkt einer der Fahrstühle der Bevorrechtigte. Er darf in Etage 2 fahren, wenn beide Fahrstühle dorthin fahren möchten, d.h. der andere Fahrstuhl muss ihm auch dann eine Erlaubnis erteilen, wenn er selbst gerade in Etage 2 fahren möchte. Allerdings wird danach der andere Fahrstuhl der Bevorrechtigte. Es muss dabei sichergestellt sein, dass sich beide Fahrstühle einig sind, wer gerade der Bevorrechtigte ist. Halten beide den anderen für bevorrechtigt, so erteilen sich beide gegenseitig die Erlaubnis, in Etage 2 zu fahren und es kann zu einer Kollision kommen. Halten beide sich selbst für bevorrechtigt, dann erteilt keiner dem anderen die Erlaubnis und es kommt zu einer Verklemmung, denn kein Fahrstuhl kann jemals wieder in Etage 2 fahren.

Möchte ein Fahrstuhl in Etage 2 fahren, fragt er den anderen Fahrstuhl um Erlaubnis. Wenn dieser sich bereits in Etage 2 oder auf dem Weg zu Etage 2 befindet, so verweigert er dem anfragenden Fahrstuhl

---

[24] Irgendwann heißt, dass sichergestellt ist, dass es immer einen späteren Zeitpunkt geben muss, an dem die genannte Bedingung wahr wird, aber keine Garantie besteht, wie lange das dauert. Mit anderen Worten muss die *Fairness* des Fahrstuhlsystems sichergestellt sein (siehe [Mau99]).

[25] Der Name $\mu_{i,j}$ ergibt sich daraus, dass Fahrstuhl i von Etage j aus angefordert wurde.

[26] Die Begriffe von außen und von innen beziehen sich auf außerhalb und innerhalb des Fahrstuhls. Eine Anforderung von außen wird von einer Person gestellt, damit der Fahrstuhl sich zu ihr hinbewegt. An sie schließt sich eine Anforderung von innen an, mit der die Person den Fahrstuhl, in dem sie sich dann auch selbst befindet, zu einer anderen Etage schickt. Da hier jedoch im Gegensatz zu Fahrstühlen, die über mehr als zwei Etagen fahren, die Zieletage feststeht, wird hier nicht auf ein explizites Losschicken des Fahrstuhls durch Drücken eines Knopfes gewartet, sondern der Fahrstuhl fährt nach einer gewissen Zeit einfach an.

die Erlaubnis, merkt sich aber vor, dass er ihm später die Erlaubnis in dem Moment gibt, wenn er dann Etage 2 wieder verlässt.[27]

Befindet sich der Empfänger der Erlaubnisanfrage sich gerade auf dem Weg von Etage 2 weg, so erteilt er dem anderen Fahrstuhl die Erlaubnis.[28]

Befindet sich der Empfänger der Erlaubnisanfrage in seiner Heimatetage, so gibt er dem anfragenden Fahrstuhl die Erlaubnis, wenn er selbst nicht gerade ebenfalls in Etage 2 fahren möchte, somit bereits eine Erlaubnisanfrage gestellt hat und auf eine Erlaubnis wartet, in Etage 2 fahren zu dürfen.

Wartet er jedoch selbst auf eine Erlaubnis, so ist entscheidend, ob er der bevorrechtigte Fahrstuhl ist. Ist dies der Fall, so verweigert er die Erlaubnis, merkt sich aber vor, dass er später die Erlaubnis erteilt, wenn er dann später irgendwann Etage 2 wieder verlässt. Außerdem wird dadurch der andere Fahrstuhl der Bevorrechtigte. Ist dies nicht der Fall, so erteilt er die Erlaubnis und wird dadurch selbst zum Bevorrechtigten.

Welcher Fahrstuhl der Bevorrechtigte ist, ist also nur genau dann relevant, wenn sich die Erlaubnisanfragen der beiden Fahrstühle kreuzen, d.h. beide ihre Erlaubnisanfrage abgeschickt haben, bevor sie die Erlaubnisanfrage vom anderen Fahrstuhl erhalten haben. Und auch nur genau dann findet ein Wechsel der Bevorrechtigung statt.

Immer wenn ein Fahrstuhl vom anderen die Erlaubnis erhalten hat, zu Etage 2 zu fahren, fährt er los. Von Etage 2 weg kann ein Fahrstuhl auch ohne Erlaubnis des anderen fahren. Hat er sich außerdem gemerkt, dass der andere Fahrstuhl die Erlaubnis haben möchte, so erteilt er sie ihm in diesem Moment.

Zur Vermeidung einer langen Wartezeit fährt ein Fahrstuhl von Etage 2 auch weg, wenn er nicht zur anderen Etage will, aber der andere Fahrstuhl zu Etage 2 will. Das tut er c_2 Zeiteinheiten, nachdem er in Etage 2 stehend eine Erlaubnisanfrage vom anderen Fahrstuhl erhalten hat. Hat er schon vor dem Erreichen von Etage 2 eine Erlaubnisanfrage vom anderen Fahrstuhl erhalten, so fährt er c_2 Zeiteinheiten nach seiner Ankunft in Etage 2 wieder ab. Da einsteigende Personen c_1 Zeiteinheiten zum Einsteigen benötigen, sollte man $c_2 \geq c_1$ wählen.

Ein Fahrstuhl, der in einer Etage steht und eine Anforderung zur anderen Etage erhalten hat, fährt nach c_1 Zeiteinheiten los bzw. stellt nach c_1 Zeiteinheiten seine Erlaubnisanfrage, wenn er eine Anforderung zu Etage 2 erhalten hat.

2.3.3 Ereignistypen und Systemzustand

Folgende Ereignistypen werden betrachtet: *Kommt_an*, *Wird_angefordert*, *Will_fahren*, *Fährt_an*, *Erhält_Erlaubnisanfrage*, *Erhält_Erlaubnis*.

Ein Ereignis vom Typ *Kommt_an* stellt die Ankunft eines Fahrstuhls in einer Etage und ein Ereignis vom Typ *Wird_angefordert* die Anforderung eines Fahrstuhls (von außen) dar. Ein Ereignis vom Typ *Will_fahren* stellt das Entstehen eines Anfahrwunsches eines Fahrstuhls dar, während ein Ereignis vom Typ *Fährt_an* das Anfahren eines Fahrstuhls darstellt. Ein Ereignis vom Typ *Erhält_Erlaubnisanfrage* bzw. vom Typ *Erhält_Erlaubnis* stellt dar, dass ein Fahrstuhl vom anderen Fahrstuhl eine Erlaubnisanfrage bzw. eine Erlaubnis für das Fahren in Etage 2 erhält.

Der Zustand des Systems besteht aus den Zuständen der beiden Fahrstühle. Der Zustand eines jeden Fahrstuhls beinhaltet, ob er der Bevorrechtigte ist, ob er sich vorgemerkt hat, dass der andere Fahrstuhl auf eine Erlaubnis von ihm wartet, ob er selbst auf eine Erlaubnis des anderen Fahrstuhls wartet, welche Anforderungen er erhalten hat, ob diese Anforderungen von außen kamen, ob er steht oder

---

[27] Diese Lösung setzt voraus, dass aufgrund der Entfernung der Fahrstühle voneinander sichergestellt ist, dass sie nicht kollidieren, wenn ein Fahrstuhl auf dem Weg zu Etage 2 losfährt, kurz nachdem der andere sich von Etage 2 weg in Bewegung gesetzt hat. Wäre das nicht sichergestellt, dann dürfte die Erlaubnis erst erteilt werden, wenn der Fahrstuhl, der Etage 2 verlässt, bei Etage 1 bzw. Etage 3 angekommen ist.

[28] Auch hier setzt diese Lösung voraus, dass aufgrund der Entfernung der Fahrstühle voneinander sichergestellt ist, dass sie nicht kollidieren, wenn ein Fahrstuhl auf dem Weg zu Etage 2 losfährt, kurz nachdem der andere sich von Etage 2 weg in Bewegung gesetzt hat.

fährt und an welchem Ort er sich befindet (wenn er steht) bzw. zuletzt befand (wenn er fährt). Aus Gründen der Übersicht wird der Zustand eines Fahrstuhls in mehrere Prädikate aufgeteilt.

- *Bevorrechtigt (i,* b_i *)* drückt aus, ob Fahrstuhl *i* sich selbst als bevorrechtigt ansieht[29], wobei b_i die Werte *ja* und *nein* annehmen kann.

- *Vorgemerkt (i,* v_i *)* drückt aus, ob Fahrstuhl *i* sich vorgemerkt hat, dass der andere Fahrstuhl auf eine Erlaubnis von ihm wartet, wobei v_i die Werte *ja* und *nein* annehmen kann.

- *Wartend (i,* w_i *)* drückt aus, ob Fahrstuhl *i* darauf wartet, dass der andere Fahrstuhl ihm eine Erlaubnis erteilt, wobei w_i die Werte *ja* und *nein* annehmen kann.

- *Anforderung (i, e,* a_{ie}, $außen_{ie}$ *)* drückt aus, ob Fahrstuhl *i* eine Anforderung zu Etage *e* erhalten hat, wobei a_{ie} und $außen_{ie}$ die Werte *ja* und *nein* annehmen können. Dabei drückt a_{ie} aus, ob es überhaupt eine Anforderung gegeben hat und $außen_{ie}$, ob die Anforderung von außen gekommen ist. Wenn immer a_{ie} den Wert *nein* hat, so hat auch $außen_{ie}$ den Wert *nein*[30]. Mehrere Anforderungen eines Fahrstuhls zu einer Etage werden grundsätzlich zu einer zusammengefasst, wobei eine zusammengesetzte Anforderung genau dann als von außen erfolgt angesehen wird, wenn mindestens eine ihrer Teilanforderungen von außen erfolgte.

- *Ort (i,* e_i, s_i *)* besagt, dass Fahrstuhl *i* den aktuellen Bewegungszustand s_i hat, wobei s_i die Werte *fahrend* und *stehend* annehmen kann. Ist der Wert von s_i *fahrend*, so besagt e_i, dass sich Fahrstuhl *i* auf dem Weg von Etage e_i weg befindet. Ist hingegen der Wert von s_i *stehend*, so besagt e_i, dass sich Fahrstuhl *i* in Etage e_i befindet.

Für den Startzustand ergibt sich

S_0 = { Bevorrechtigt (1, *ja*), Bevorrechtigt (2, *nein*),
Vorgemerkt (1, *nein*), Vorgemerkt (2, *nein*),
Wartend (1, *nein*), Wartend (2, *nein*),
Anforderung (1, 1, *nein*, *nein*), Anforderung (1, 2, *nein*, *nein*),
Anforderung (2, 2, *nein*, *nein*), Anforderung (2, 3, *nein*, *nein*),
Ort (1, 1, *stehend*), Ort (2, 3, *stehend*) }.

Am Anfang ist Fahrstuhl 1 bevorrechtigt und steht in Etage 1, während Fahrstuhl 2 in Etage 3 steht, Anforderungen gibt es noch keine.

Gestartet wird zum Zeitpunkt $c_0 = 0$. Die initiale Ereignismenge E_0 besteht aus vier Ereignissen, den jeweils ersten Anforderungen an die Fahrstühle, also

E_0 = { Wird_angefordert (1, 1) @ (c_0 + Expo_ZV($\mu_{1,1}$)),

Wird_angefordert (1, 2) @ (c_0 + Expo_ZV($\mu_{1,2}$),

Wird_angefordert (2, 2) @ (c_0 + Expo_ZV($\mu_{2,2}$),

Wird_angefordert (2, 3) @ (c_0 + Expo_ZV($\mu_{2,3}$)}.

---

[29] Wenn die Spezifikation keinen Fehler enthält, dann sieht sich ein Fahrstuhl genau dann als bevorrechtigt an, wenn er auch bevorrechtigt ist.

[30] An sich ist der Wert von $außen_{ie}$ dann ohne Bedeutung, aber durch diese Festlegung lassen sich die Regeln zur Spezifikation des Fahrstuhlsystems leichter formulieren.

Ein explizites Simulationsende wird hier nicht erfasst, d.h. die Simulation läuft unendlich lange.

Bei der Darstellung der Regeln sind diejenigen Argumente von Prädikaten, deren Wert nicht verwendet wird, mit _ gekennzeichnet.

2.3.4 Ereignisausführung Kommt_an

Das Ereignis *Kommt_an* stellt die Ankunft eines Fahrstuhls in einer Etage dar. Parameter sind die Nummer des Fahrstuhls und die Nummer der Etage, in der der Fahrstuhl ankommt.

In allen Fällen gilt, dass, wenn der Fahrstuhl in einer Etage ankommt, alle Anforderungen an diese Etage erfüllt sind. Ferner muss in allen Fällen der Ort und der Bewegungszustand des Fahrstuhls aktualisiert werden.

Kommt der Fahrstuhl in einer Etage an, zu der er von außen angefordert wurde, so will der Fahrstuhl demnächst (in c_1 Zeiteinheiten) in die andere Etage fahren, um die eingestiegenen Personen dorthin zu befördern. Die Anforderung von innen zur anderen Etage wird ergänzt. Der Wert von $außen_{ik}$ (wobei *i* den Fahrstuhl und *k* die andere Etage darstellt) wird beibehalten, um keine evtl. bereits existierende Anforderung von außen zu löschen.

Ereignis	**Kommt_an (i, j) @ t**
alter Zustand	**{ .. , Anforderung (i, j, *ja*, *ja*), Anforderung (i, k, _,** $außen_{ik}$ **), .. , Ort (i, k, *fahrend*), .. }**
neuer Zustand	**{ .. , Anforderung (i, j, *nein*, *nein*), Anforderung (i, k, *ja*,** $außen_{ik}$ **), .. , Ort (i, j, *stehend*), .. }**
Folgeereignisse	**{ Will_fahren (i, j) @ (t +** c_1 **) }**

Regel 7: Kommt_an.1

Wurde der Fahrstuhl von innen zu dieser Etage angefordert, aber existiert schon eine Anforderung zur anderen Etage, so will der Fahrstuhl ebenfalls demnächst dorthin fahren.

Ereignis	**Kommt_an (i, j) @ t**
alter Zustand	**{ .. , Anforderung (i, j, _, _), Anforderung (i, k, *ja*,** $außen_{ik}$ **), .. , Ort (i, k, *fahrend*), .. }**
neuer Zustand	**{ .. , Anforderung (i, j, *nein*, *nein*), Anforderung (i, k, *ja*,** $außen_{ik}$ **), .. , Ort (i, j, *stehend*), .. }**
Folgeereignisse	**{ Will_fahren (i, j) @ (t +** c_1 **) }**

Regel 8: Kommt_an.2

Kommt der Fahrstuhl in Etage 2 an und hat sich vorgemerkt, dass der andere Fahrstuhl auf eine Erlaubnis wartet, in Etage 2 fahren zu dürfen, so will der Fahrstuhl (spätestens) nach c_2 Zeiteinheiten anfahren, um Etage 2 für den anderen Fahrstuhl freizumachen.

Ereignis	**Kommt_an (i, 2) @ t**
alter Zustand	**{ .. , Vorgemerkt (i, *ja*), .. , Anforderung (i, 2, _, _), .., Ort (i, k, *fahrend*), .. }**
neuer Zustand	**{ .. , Vorgemerkt (i, *ja*), .. , Anforderung (i, 2, *nein*, *nein*), .., Ort (i, 2, *stehend*), .. }**
Folgeereignisse	**{ Will_fahren (i, 2) @ (t +** c_2 **) }**

Regel 9: Kommt_an.3

Ansonsten werden nur die ggf. vorhandenen Anforderungen[31] als erfüllt markiert und der Ort und der Bewegungszustand werden aktualisiert.

Ereignis	**Kommt_an (i, j) @ t**
alter Zustand	**{ .. , Anforderung (i, j, _, *nein*), .. , Ort (i, k, *fahrend*), .. }**
neuer Zustand	**{ .. , Anforderung (i, j, *nein*, *nein*), .. , Ort (i, j, *stehend*), .. }**
Folgeereignisse	**Ø**

Regel 10: Kommt_an.4

2.3.5 Ereignisausführung Wird_angefordert

Das Ereignis *Wird_angefordert* stellt die Anforderung eines Fahrstuhls von außen dar. Parameter sind die Nummer des Fahrstuhls und die Nummer der Etage, von der aus der Fahrstuhl angefordert wird.

In allen Fällen wird als Folgeereignis die nächste Anforderung an diesen Fahrstuhl von außen zu dieser Etage generiert.

Steht der Fahrstuhl gerade in der angeforderten Etage (dann kann vorher noch keine Anforderung zu dieser Etage vorhanden gewesen sein), so ist der erste Teil der Anforderung sofort erfüllt. Deshalb will der Fahrstuhl demnächst (in c_1 Zeiteinheiten) in die andere Etage fahren, um die eingestiegenen Personen dorthin zu befördern. Die Anforderung von innen zur anderen Etage wird ergänzt.

Ereignis	**Wird_angefordert (i, j) @ t**
alter Zustand	**{ .. , Anforderung (i, j, *nein*, *nein*), Anforderung (i, k, _,** $außen_{ik}$ **), .., Ort (i, j, *stehend*), .. }**
neuer Zustand	**{ .. , Anforderung (i, j, *nein*, *nein*), Anforderung (i, k, *ja*,** $außen_{ik}$ **) .., Ort (i, j, *stehend*), .. }**
Folgeereignisse	**{ Will_fahren (i, j) @ (t +** c_1 **),** **Wird_angefordert (i, j) @ (t + Expo_ZV(** $\mu_{i,j}$ **) }**

Regel 11: Wird_angefordert.1

Sollte vorher noch keine Anforderung nach Etage j vorhanden sein und steht der Fahrstuhl gerade in der anderen Etage, so wird die Anforderung in den Zustand aufgenommen und der Fahrstuhl will außerdem demnächst (in c_1 Zeiteinheiten) losfahren.

[31] Dass ein Fahrstuhl in einer Etage ankommt, ohne dass eine Anforderung in diese Etage vorlag, ist möglich, wenn ein Fahrstuhl nur zurück in seine Heimatetage gefahren ist, um Etage 2 für den anderen Fahrstuhl frei zu machen.

Ereignis	**Wird_angefordert (i, j) @ t**
alter Zustand	**{ .. , Anforderung (i, j, *nein*, *nein*), Anforderung (i, k, *nein*, *nein*), .., Ort (i, k, *stehend*), .. }**
neuer Zustand	**{ .. , Anforderung (i, j, *ja*, *ja*), Anforderung (i, k, *nein*, *nein*) .., Ort (i, k, *stehend*), .. }**
Folgeereignisse	**{ Will_fahren (i, k) @ (t +** c_1 **),** **Wird_angefordert (i, j) @ (t + Expo_ZV(** $\mu_{i,j}$ **) }**

Regel 12: Wird_angefordert.2

Ansonsten wird nur die Anforderung in den Zustand aufgenommen (sofern nicht bereits vorhanden) sowie, wie in allen Fällen, das Folgeereignis der nächsten Anforderung generiert.

Ereignis	**Wird_angefordert (i, j) @ t**
alter Zustand	**{ .. , Anforderung (i, j, _, _), .. }**
neuer Zustand	**{ .. , Anforderung (i, j, *ja*, *ja*), .. }**
Folgeereignisse	**{ Wird_angefordert (i, j) @ (t + Expo_ZV(** $\mu_{i,j}$ **) }**

Regel 13: Wird_angefordert.3

2.3.6 Ereignisausführung Will_fahren

Das Ereignis *Will_fahren* stellt das Entstehen eines Anfahrwunsches eines Fahrstuhls dar. Parameter von *Will_fahren* sind die Nummer des Fahrstuhls und die Nummer der Etage, von der aus der Fahrstuhl anfahren will.

Das Ereignis ist ein Hilfsereignis, dass eingesetzt wird, um die Länge der Beschreibung zu reduzieren. Würde man das Simulationssystem ohne dieses Ereignis spezifizieren wollen, so müssten alle Regeln, die ein Ereignis vom Typ *Will_fahren* als Folgeereignis haben, entsprechend der Regeln von *Will_fahren* modifiziert werden. Dies hätte zur Folge, dass das selbe Verhalten des Fahrstuhls an vielen Stellen der Spezifikation beschrieben werden müsste, was eine gewisse Redundanz zur Folge hätte. Würde man die Spezifikation des Verhaltens des Fahrstuhls bei einem Anfahrwunsch ändern wollen, so müsste man die gleichen Änderungen an mehreren Stellen vornehmen, was eine potentielle Fehlerquelle wäre.

Befindet sich der Fahrstuhl in Etage 2 und will von dort wegfahren, so kann er unmittelbar losfahren.

Ereignis	**Will_fahren (i, 2) @ t**
alter Zustand	**{ .., Ort (i, 2, *stehend*), .. }**
neuer Zustand	**{ .., Ort (i, 2, *stehend*), .. }**
Folgeereignisse	**{ Fährt_an (i, 2) @ t }**

Regel 14: Will_fahren.1

Will er dagegen in Etage 2 fahren, so muss er den anderen Fahrstuhl um Erlaubnis fragen. Somit erhält der andere Fahrstuhl[32] demnächst (nach Δ Zeiteinheiten, der Übertragungsdauer einer Nachricht) eine Erlaubnisanfrage. Der Fahrstuhl wartet nun auf die Erlaubnis.

[32] Der andere Fahrstuhl wird hier durch die Funktion *andererFahrstuhl: { 1, 2 }* $\rightarrow$ *{ 1, 2 }* bestimmt, wobei *andererFahrstuhl (1) = 2* und *andererFahrstuhl (2) = 1* gilt. Kürzer, aber unintuitiver ist die Darstellung *andererFahrstuhl (i) = 3 - i.*

Ereignis	`Will_fahren ( i, j ) @ t`
alter Zustand	`{ .. , Wartend ( i, nein ), .. , Ort ( i, j, stehend ), .. }`
neuer Zustand	`{ .. , Wartend ( i, ja ), .. , Ort ( i, j, stehend ), .. }`
Folgeereignisse	`{ Erhält_Erlaubnisanfrage ( andererFahrstuhl ( i ) @ ( t + Δ )}`

Regel 15: Will_fahren.2

Befindet sich der Fahrstuhl nicht mehr in der Etage, von der aus er losfahren will, so ist dieses Ereignis irrelevant. Der Zustand ändert sich nicht, es wird kein Folgeereignis erzeugt. Dieser Fall kann eintreten, wenn sowohl gemäß einer der Regeln *Kommt_an.1* (Regel 7), *Kommt_an.2* (Regel 8), *Wird_angefordert.1* (Regel 11) oder *Wird_angefordert.2* (Regel 12) ein Folgeereignis *Will_fahren* erzeugt wurde, um Personen in die andere Etage zu befördern als auch gemäß einer der Regeln *Kommt_an.3* (Regel 9) oder *Erhält_Erlaubnisanfrage.2* (Regel 20) ein Folgeereignis *Will_fahren*, um Etage 2 für den anderen Fahrstuhl freizumachen.

Ereignis	`Will_fahren ( i, j ) @ t`
alter Zustand	`{ .. }`
neuer Zustand	`{ .. }`
Folgeereignisse	`Ø`

Regel 16: Will_fahren.3

2.3.7 Ereignisausführung Fährt_an

Das Ereignis *Fährt_an* stellt das Anfahren eines Fahrstuhls dar. Parameter sind die Nummer des Fahrstuhls und die Nummer der Etage, von der aus der Fahrstuhl anfährt.

In allen Fällen wechselt der Fahrstuhl in den Bewegungszustand *fahrend* und es wird ein Folgeereignis erzeugt, die Ankunft des Fahrstuhls auf der anderen Etage[33].

Fährt der Fahrstuhl von Etage 2 weg und hat er sich vorgemerkt, dass der andere Fahrstuhl auf eine Erlaubnis für das Fahren in Etage 2 wartet, so erteilt er ihm diese in diesem Moment, so dass sie nach Δ Zeiteinheiten beim anderen Fahrstuhl ankommt. Er braucht sich nun nicht mehr vorzumerken, dass der andere Fahrstuhl auf eine Erlaubnis von ihm wartet.

Ereignis	`Fährt_an ( i, 2 ) @ t`
alter Zustand	`{ .. , Vorgemerkt ( i, ja ), .. , Ort ( i, 2, stehend ), .. }`
neuer Zustand	`{ .. , Vorgemerkt ( i, nein ), .. , Ort ( i, 2, fahrend ), .. }`
Folgeereignisse	`{ Kommt_an ( i, andereEtage ( i, 2 ) ) @ ( t + Gleich_ZV ( a, b ) ),` `Erhält_Erlaubnis ( andererFahrstuhl ( i ) @ ( t + Δ ) }`

Regel 17: Fährt_an.1

Ansonsten wird nur in den Bewegungszustand *fahrend* gewechselt und als Folgeereignis die Ankunft des Fahrstuhls erzeugt.

[33] Die andere Etage wird hier durch die (partielle) Funktion *andereEtage: { 1, 2 } × { 1, 2, 3 } → { 1, 2, 3 }* bestimmt, wobei *andereEtage (1, 1) = 2*, *andereEtage (1, 2) = 1*, *andereEtage (2, 2) = 3* und *andereEtage (2, 3) = 2* gilt. Undefiniert sind *andereEtage (1, 3)* und *andereEtage (2, 1)*. Kürzer, aber unintuitiver ist die Darstellung *andereEtage (i, j) = 1 + 2 i - j*, wobei hier vorausgesetzt werden muss, dass die Funktion nur für Werte ausgewertet werden soll, für die sie definiert ist.

Ereignis	**Fährt_an (i, j) @ t**
alter Zustand	**{ .. , Ort (i, j, *stehend*), .. }**
neuer Zustand	**{ .. , Ort (i, j, *fahrend*), .. }**
Folgeereignisse	**{ Kommt_an (i, andereEtage (i, j)) @ (t + Gleich_ZV (a, b))}**

Regel 18: Fährt_an.2

2.3.8 Ereignisausführung Erhält_Erlaubnisanfrage

Das Ereignis *Erhält_Erlaubnisanfrage* stellt dar, dass ein Fahrstuhl vom anderen Fahrstuhl eine Erlaubnisanfrage für das Fahren in Etage 2 erhält. Parameter ist die Nummer des Fahrstuhls, der die Anfrage erhält.

Befindet sich der Fahrstuhl, der die Anfrage erhält, auf dem Weg von Etage 2 weg, so kann er bedenkenlos eine Erlaubnis erteilen.

Ereignis	**Erhält_Erlaubnisanfrage (i) @ t**
alter Zustand	**{ .. , Ort (i, 2, *fahrend*), .. }**
neuer Zustand	**{ .. , Ort (i, 2, *fahrend*), .. }**
Folgeereignisse	**{ Erhält_Erlaubnis (andererFahrstuhl (i) @ (t + Δ) }**

Regel 19: Erhält_Erlaubnisanfrage.1

Befindet sich der Fahrstuhl, der die Anfrage erhält, in Etage 2, so kann er dem anderen Fahrstuhl im Moment keine Erlaubnis erteilen, merkt sich aber vor, dass er ihm eine Erlaubnis erteilt, sobald er Etage 2 verlässt. Außerdem wird er (spätestens) in c_2 Zeiteinheiten anfahren, um dem anderen Fahrstuhl dann eine Fahrt in Etage 2 zu ermöglichen.

Ereignis	**Erhält_Erlaubnisanfrage (i) @ t**
alter Zustand	**{ .. , Vorgemerkt (i, *nein*), .. , Ort (i, 2, *stehend*), .. }**
neuer Zustand	**{ .. , Vorgemerkt (i, *ja*), .. , Ort (i, 2, *stehend*), .. }**
Folgeereignisse	**{ Will_fahren (i, 2) @ (t + c_2)}**

Regel 20: Erhält_Erlaubnisanfrage.2

Befindet sich der Fahrstuhl, der die Anfrage erhält, gerade auf dem Weg zu Etage 2 hin, so kann er dem anderen Fahrstuhl im Moment keine Erlaubnis erteilen, merkt sich aber vor, dass er ihm eine Erlaubnis erteilt, sobald er Etage 2 wieder verlässt.

Ereignis	**Erhält_Erlaubnisanfrage (i) @ t**
alter Zustand	**{ .. , Vorgemerkt (i, *nein*), .. , Ort (i, j, *fahrend*), .. }**
neuer Zustand	**{ .. , Vorgemerkt (i, *ja*), .. , Ort (i, j, *fahrend*), .. }**
Folgeereignisse	**Ø**

Regel 21: Erhält_Erlaubnisanfrage.3

Befindet er sich in seiner Heimatetage und hat selbst noch keine Erlaubnisanfrage gestellt (wartet also nicht seinerseits auf eine Erlaubnis des anderen Fahrstuhls), so erteilt er dem anderen die Erlaubnis.

Ereignis	**Erhält_Erlaubnisanfrage (i) @ t**
alter Zustand	**{ .., Wartend (i, *nein*), .., Ort (i, j , *stehend*), .. }**
neuer Zustand	**{ .., Wartend (i, *nein*), .., Ort (i, j , *stehend*), .. }**
Folgeereignisse	**{ Erhält_Erlaubnis (andererFahrstuhl (i) @ (t + Δ) }**

Regel 22: Erhält_Erlaubnisanfrage.4

Befindet er sich in seiner Heimatetage und hat ebenfalls schon eine Erlaubnisanfrage gestellt, so haben sich also die Erlaubnisanfragen der beiden Fahrstühle gekreuzt. Demnach ist es entscheidend, welcher Fahrstuhl der Bevorrechtigte ist. Ist der Empfänger der Nachricht der Bevorrechtigte, so erteilt er dem anderen Fahrstuhl keine Erlaubnis, merkt sich aber vor, dass er die Erlaubnis erteilt, sobald er dann später irgendwann Etage 2 erreicht und auch wieder verlassen hat. Dadurch wird der andere Fahrstuhl der Bevorrechtigte.

Ereignis	**Erhält_Erlaubnisanfrage (i) @ t**
alter Zustand	**{ .., Bevorrechtigt (i, *ja*), .., Vorgemerkt (i, *nein*), .., Wartend (i, *ja*), .., Ort (i, j , *stehend*), .. }**
neuer Zustand	**{ .., Bevorrechtigt (i, *nein*), .., Vorgemerkt (i, *ja*), .., Wartend (i, *ja*), .., Ort (i, j , *stehend*), .. }**
Folgeereignisse	**Ø**

Regel 23: Erhält_Erlaubnisanfrage.5

Ist hingegen der Empfänger der Nachricht nicht der Bevorrechtigte, so erteilt er dem anderen Fahrstuhl die Erlaubnis. Dadurch wird er selbst der Bevorrechtigte.

Ereignis	**Erhält_Erlaubnisanfrage (i) @ t**
alter Zustand	**{ .., Bevorrechtigt (i, *nein*), .., (Wartend (i, *ja*), .., Ort (i, j , *stehend*), .. }**
neuer Zustand	**{ .., Bevorrechtigt (i, *ja*), .., (Wartend (i, *ja*), .., Ort (i, j , *stehend*), .. }**
Folgeereignisse	**{ Erhält_Erlaubnis (andererFahrstuhl (i) @ (t + Δ) }**

Regel 24: Erhält_Erlaubnisanfrage.6

2.3.9 Ereignisausführung Erhält_Erlaubnis

Das Ereignis *Erhält_Erlaubnis* stellt dar, dass ein Fahrstuhl vom anderen Fahrstuhl eine Erlaubnis für das Fahren in Etage 2 erhält. Parameter ist die Nummer des Fahrstuhls, der die Erlaubnis erhält.

Der Empfänger der Nachricht ändert seinen Zustand dahingehend, dass er jetzt nicht mehr auf die Erlaubnis wartet. Er fährt sofort an.

Ereignis	**Erhält_Erlaubnis (i) @ t**
alter Zustand	**{ .., Wartend (i, *ja*), .., Ort (i, j, *stehend*), .. }**
neuer Zustand	**{ .., Wartend (i, *nein*), .., Ort (i, j, *stehend*), .. }**
Folgeereignisse	**{ Fährt_an (i, j) @ t }**

Regel 25: Erhält_Erlaubnis.1

2.3.10 Abschlussbemerkungen

In diesem Abschnitt werden einige abschließende Bemerkungen für das Fahrstuhl-Beispiel und seiner Simulation vorgenommen. Dabei wird gezeigt, dass es keine Kollision zwischen den Fahrstühlen geben kann.

Um sicherzustellen, dass es zu keiner Kollision zwischen den Fahrstühlen kommt, dürfen sich nicht beide Fahrstühle gleichzeitig in Etage 2 oder auf dem Weg zu Etage 2 befinden.

Da beide Fahrstühle sich anfangs in ihrer Heimatetage befinden und nur in Etage 2 fahren, wenn sie vorher vom anderen Fahrstuhl die Erlaubnis dazu erhalten haben, kann eine solche Situation nur auftreten, wenn beide Fahrstühle sich gegenseitig die Erlaubnis gegeben haben.

Befindet sich ein Fahrstuhl in Etage 2 oder auf dem Weg dorthin, so erteilt er dem anderen Fahrstuhl gemäß Regeln *Erhält_Erlaubnisanfrage.2* (Regel 20) bzw. *Erhält_Erlaubnisanfrage.3* (Regel 21) keine Erlaubnis. Wenn ein Fahrstuhl von Etage 2 wegfährt, dann kann er dem anderen gemäß Regel *Erhält_Erlaubnisanfrage.1* (Regel 19) die Erlaubnis erteilen, was in diesem Fall aber auch unkritisch ist.

Kritisch ist der Fall, wenn sich beide Fahrstühle in ihrer Heimatetage befinden.

Überkreuzen sich die beiden Erlaubnisanfragen, also haben beide ihre Anfragen abgeschickt, bevor sie die des anderen erhalten haben, so erhält nur der Bevorrechtigte die Erlaubnis vom anderen. Zu diesem Zweck ist es wichtig, dass beide sich einig sind, wer gerade der Bevorrechtigte ist. Halten beide den anderen für bevorrechtigt, so erteilen sich beide nach Regel *Erhält_Erlaubnisanfrage.6* (Regel 24) gegenseitig die Erlaubnis, in Etage 2 zu fahren und es kann zu einer Kollision kommen. Halten beide sich selbst für bevorrechtigt, dann erteilt nach Regel *Erhält_Erlaubnisanfrage.5* (Regel 23) keiner dem anderen die Erlaubnis und es kommt zu einer Verklemmung, denn kein Fahrstuhl kann jemals wieder in Etage 2 fahren. Da aber beide Fahrstühle dann und nur dann, wenn sich zwei Erlaubnisanfragen überkreuzen, den aktuellen Bevorrechtigten wechseln, ist immer genau einer von ihnen der Bevorrechtigte. Dies gilt nur, sofern es im Anfangszustand auch schon so war, was aber auch der Fall ist. Es ist ferner sichergestellt, dass eine Überkreuzung der Nachrichten durch beide Fahrstühle als solche wahrgenommen wird, da sich Nachrichten nicht überholen können. Erhält z.B. Fahrstuhl 1 eine Erlaubnisanfrage, nachdem er selber eine abgeschickt hat und bevor er eine Erlaubnis erhalten hat, so weiß er, dass Fahrstuhl 2 die Erlaubnisanfrage losgeschickt hat, bevor er die von Fahrstuhl 1 erhalten hat. Denn sonst hätte Fahrstuhl 2 ihm nach Regel *Erhält_Erlaubnisanfrage.4* (Regel 22) sofort eine Erlaubnis geschickt und die wäre dann vor der Anfrage von Fahrstuhl 2 bei ihm angekommen.

Dass sich Nachrichten nicht überholen, ist in der Simulation allerdings nur für Nachrichten gewährleistet, die zu verschiedenen Simulationszeitpunkten abgeschickt worden sind, da nicht spezifiziert ist, welches Ereignis bei mehreren zeitgleichen Ereignissen aus der Ereignismenge ausgewählt wird. Zeitgleich abgeschickte Nachrichten würden allerdings in der richtigen Reihenfolge verarbeitet, wenn man aus der Ereignismenge bei gleichzeitigen Ereignissen das auswählt, das zuerst eingefügt wurde. Da die Zeitpunkte der Nachrichten aber kontinuierlich sind und vom Zufall abhängen, ist er sehr unwahrscheinlich, dass dieser Fall überhaupt eintritt.

Wenn es aber zu keiner Überkreuzung der Nachrichten kommt, also z.B. Fahrstuhl 1 vor Fahrstuhl 2 die Erlaubnisanfrage stellt, d.h. sie ankommt, bevor Fahrstuhl 2 seine abschickt, dann kommt die Erlaubnis von Fahrstuhl 2 auch vor der Erlaubnisanfrage von Fahrstuhl 2 bei Fahrstuhl 1 an, da Nachrichten sich nicht überholen können. Sobald aber die Erlaubnis von Fahrstuhl 2 bei Fahrstuhl 1 ankommt, wechselt Fahrstuhl 1 nach Regel *Erhält_Erlaubnis.1* (Regel 25) und dann nach Regel *Fährt_an.2* (Regel 18) sofort in den Bewegungszustand *fahrend*, wodurch aufgrund Regel *Erhält_Erlaubnisanfrage.3* (Regel 21) ausgeschlossen ist, dass er Fahrstuhl 2 eine Erlaubnis erteilt. Hierzu müsste allerdings sichergestellt sein, dass unverzögerte Folgeereignisse (Ereignisse, die den gleichen Zeitpunkt wie ihr erzeugendes Ereignis haben) "sofort" ausgeführt werden, also vor anderen Ereignissen, die zufällig ebenfalls den gleichen Ausführungszeitpunkt haben. Im konkreten Fall würde es zu einem Fehler führen, wenn erst das Ereignis *Erhält_Erlaubnis*, dann das (zufällig zeitgleiche) Ereignis *Erhält_Erlaubnisanfrage* und erst dann das nach Regel *Erhält_Erlaubnis.1* (Regel 25) erzeugte Folgeereignis *Fährt_an* ausgeführt werden würde. Dann würde bei *Erhält_Erlaubnisanfrage* zu Unrecht eine Erlaubnis nach Regel *Erhält_Erlaubnisanfrage.4* (Regel 22) erteilt werden, da der Zustand des Fahrstuhls, der die Anfrage erhält, in diesem Moment beinhalten würde, dass der

Fahrstuhl in seiner Heimatetage steht und nicht selbst auf eine Erlaubnis wartet. Da die Zeitpunkte der Nachrichten aber kontinuierlich sind und vom Zufall abhängen, ist es auch hier sehr unwahrscheinlich, dass dieser Fall überhaupt eintritt.

Damit ist gezeigt, dass es bei einem gemäß der Regeln spezifizierten Fahrstuhlsystem zu keinen Kollisionen der beiden Fahrstühle kommen kann. In der Simulation muss dabei allerdings das Simulationssystem mit gleichzeitigen Ereignissen so umgehen, dass sie prinzipiell nach Erzeugungsreihenfolge (FIFO, first-in, first-out), aber unverzögerte Folgeereignisse vor anderen gleichzeitigen Ereignissen ausgeführt werden.