8 Rule Generalization and Correction

One of the most important steps during the induction of extraction rules is their generalization. The linguistic patterns for expression of information are obtained abstracting from the concrete training examples. Such an abstraction can be achieved excerpting different common features from the initial patterns or replacing the specific elements of an extraction pattern by more general elements. The corresponding generalizing heuristics of merging, abstraction and substitution of extraction patterns are presented in this chapter.

Rule generalization is performed gradually so that a generalized rule can again participate in the generalization process. The validation of extraction rules and control of their induction are handled in chapter 9.

Generalization of extraction rules is connected with the tradeoff of increasing the covering degree of a rule at the cost of its precision. A general rule accounts for various expression possibilities of the extracted information and its context and becomes therefore prone to erroneous extractions localizing irrelevant information. Such errors do not just negatively affect the overall precision, but compromise the extraction rule itself preventing it from extraction of certain attributes or leading to a complete discarding of the rule as we will see in the next chapter. Rule correction is an important instrument that helps to reduce the error rate and increase the precision of the rule without loss of its generality. We analyze the types of errors extraction rules can commit and present an algorithm for correction of rules based on the negation pattern.

8.1 Rule Merging

Especially at the beginning of the induction process the extraction rules can hardly extract more information than the training examples they originate from. The rule patterns at the first stage of induction contain very specific context and cannot therefore be matched with any other text parts containing relevant information. The purpose of generalization is to improve the covering degree of extraction rules without considerable loss of precision relaxing tight context constraints and making the encoding of extractions less detailed.

Based on our observation that similar information is expressed in a similar way in the natural language we can derive a single linguistic pattern for several similar variants of information expression. The idea behind rule merging is to find and generalize commonalities like similar expressions in the rule patterns. If similar lexical expression, similar linguistic property or any other feature occur in the context or in the immediate encoding of extracted information (e.g. values of a certain attribute) several times we can assume that there is a correlation between this feature and the extracted information. The more similar two rules are, the more combinations of similar features of extracted information can be identified and their correlation with the extracted information is more confident.

To incorporate two similar patterns in one more general pattern the generalizing capabilities of the pattern language can be optimally exploited. In fact, the pattern language offers constructs for expression of variations occurring in the natural language in a single pattern. Consider the following examples from the *seminar announcement corpus*:

- ▷ Often the same information can be expressed with more or less details that are not part of the information but may be its important indicators in the context. Such an optional occurrence of details is covered by the option pattern. Consider the sample sentences: *The lecture begins at 2:15 p.m.* and *The lecture begins as usual at 2:15 p.m. after the lunch break*. Using the option pattern we can incorporate both expression variants in a single pattern *The lecture begins (as usual)? at 2:15 p.m. (after the lunch break)?*[1].

- ▷ Many details mentioned with the information of interest do not contribute to its identification. The wildcard pattern can be used to ignore irrelevant text fragments. If, for instance, we are interested in the starting time of a lecture, the irrelevant sentence parts of the following examples *Since the lecture begins at 2:15 p.m., the dinner will be postponed*, *In spite of the postponed dinner the lecture begins at 2:15 p.m.* can be blinded out replacing them by wildcards: ** the lecture begins at 2:15 p.m. **.

- ▷ Sometimes sentences contain repetitions of morphological or syntactic structures with the variable length. For example, the values of the attribute SPEAKER usually contain a title, a variable number of first names and a last name: *Prof. Steven Ewing*, *Prof. Joshua Erwin Griffith*. Repetitive structures can adequately be covered by the Kleene star pattern: *Prof. (NE)**.

- ▷ The natural language grammars often allow different order of words and syntactic constituents in a sentence (e.g. in German the order of the main verb and its object depends on whether they are in the main or subordinate clause). To express various possible positions of the prepositional phrase in the sentences *After the lunch break the lecture will begin at 2:15 p.m.* and *The lecture begins at 2:15 p.m. after the lunch break* the permutation pattern can be applied:*((After the lunch break) (the lecture will? PF:"begin" at 2:15 p.m.)%*

- ▷ Even in very similar contexts the natural language still offers different expression possibilities:*The lecture begins at 2:15 p.m.*, *The lecture begins a*

[1] For the seek of brevity we use sequences of strings as sample patterns in this and the following examples

quarter past two in the afternoon. Alternative expression forms can be captured by the union pattern: *The lecture begins (at 2:15 p.m. | a quarter past two in the afternoon)*.

The pattern language covers the most common of possible expression variations occurring in natural language and allows to generalize the linguistic patterns finding an appropriate general backtracking pattern that subsumes different expression forms. The number of subsumed variations can be drastically increased by recursive usage of backtracking patterns. Before considering this issue (cf. sec. 8.1.4) we introduce the merging operation formally.

The merging operation is defined as a function taking two patterns as their arguments and returning the merged pattern:
$merge :: Pattern \rightarrow Pattern \rightarrow Pattern$
From a theoretical point of view rule merging is a heuristic derivation (functional mapping) of a new context-free expression from two context-free expressions representing extraction patterns. The level of context-free languages is not left because the resulting pattern is also specified in the pattern specification language, which is guaranteed by the *merge* function.

Since the patterns significantly vary in their type and properties (refer to appendix A), merging of specific pattern types requires a differentiated approach. To account for the specific properties of primitive (strings, XML and POS) patterns and sequences we first define the merging function for these special cases before proceeding with the general cases.

8.1.1 Merging Lexical Strings, POS and XML Patterns

The simplest pattern that carries the most concise information is the string pattern. When two strings are not identical, we still can generalize them, if they originate from the same word root, i.e. their principal form is identical. Let the function $PForm$ denote the principal form (e.g. infinitive of a verb) of a string:

$merge(A, B) = PF : PForm(A)$ where A and B are lexical strings and $PForm(A) = PFrom(B)$[2]
If the word roots of the strings differ, they still can be semantically related having a similar word meaning. If the strings belong to the same synonym set (see sec. 5.2.5), their generalization is the SynSet pattern matching any synonym from the set:
$merge(A, B) = SynSet : A$ where A, B are strings and $B \in SynSet(A)$

POS patterns embed lexical, morphological and semantic information, and each of these components can be exploited for merging. Unless all components of two merged patterns are identical, a component has to be found the merging will be based on. The most stringent component of a POS pattern is the lexical element so that it is first tried to merge lexical elements in order to maintain the maximum common information comprised by patterns. Similarly as in the case of strings, if no morphological connection between the lexical elements can be found, their semantic relatedness is checked. If the lexical elements are synonyms, the POS patterns can be merged in a synonym pattern. Otherwise the weakest component – the part of speech – is compared. If parts of speech of both POS patterns are identical, the merged POS pattern will contain only this part of speech without the lexical element:

[2] all cases not regarded here (e.g. when the strings are identical or with different principal forms) will be handled in the section 8.1.3.

$merge(POS : pos_1\ lex\_el_1, POS : pos_2\ lex\_el_2) = PF : PForm(lex\_el_1)$ if $PForm(lex\_el_1) = PForm(lex\_el_2)$
$merge(POS : pos_1\ lex\_el_1, POS : pos_2\ lex\_el_2) = SynSet : lex\_el_1$ if $lex\_el_2 \in SynSet(lex_el_1)$
$merge(POS : pos_1\ lex\_el_1, POS : pos_2\ lex\_el_2) = POS : pos_1$ if $pos_1 = pos_2$

XML patterns can be merged based on their tags and their inner patterns. XML elements may be maintained during merging if their tags correspond. If the tags are not identical, the hierarchical structure build by the XML patterns is abandoned, but the inner patterns of both XML patterns can still be merged.

$merge(\backslash A[A_1 \dots A_n], \backslash B[B_1 \dots B_n]) = \backslash A[merge(A_1 \dots A_n, B_1 \dots B_n)]$ where $A = B$
$merge(\backslash A[A_1 \dots A_n], \backslash B[B_1 \dots B_n]) = merge(A_1 \dots A_n, B_1 \dots B_n)$ otherwise

Since the inner patterns of XML patterns are usually sequences, the actual challenge while merging XML patterns is the merging of sequences, which will be regarded in the following section.

8.1.2 Merging Sequences

Sequence is a basic pattern in extraction rules because the sentence structure is sequential and syntactic constituents consist of word sequences. Merging sequence patterns is not straightforward as there are numerous possibilities of grouping and merging different similar sequence elements. Since we are interested in maintaining characteristic common features in the patterns during the merging process, as many as possible similar sequence elements should be merged. Thus we can leverage the $MaxSimScore$ algorithm (refer to fig. 7.5) to determine the optimal alignment of two sequences. The aligned sequence elements and the subsequences between the aligned elements are merged. The resulting sequence contains the merged aligned elements and the merged subsequences between the aligned elements in the order they occur in the alignment.

Let $A_1 \dots A_n$ and $B_1 \dots B_n$ be the sequences that have to be merged and $L_{max} = \{\dots (A_i, B_j), (A_k, B_l) \dots\}$ – the optimal alignment determined by $MaxSimScore$ algorithm so that $\sum_{i=1}^{|L_{max}|} Score(l_i)$ is maximum where $l_i \in L$.

$merge(A_1 \dots A_n, B_1 \dots B_n) = \dots merge(A_i, B_j)\ merge(A_{i+1} \dots A_{k-1}, B_{j+1} \dots B_{l-1})$ $merge(A_k, B_l) \dots$ where $(A_i, B_j) \in L_{max}$ and $(A_k, B_l) \in L_{max}$.

The $MaxSimScore$ algorithm plays a crucial role in the rule generalization process. On the one hand, it establishes the rule similarity measure allowing an adequate selection of rule pairs for merging. On the other hand, it solves the problem of sequence merging determining the optimal set of merged pairs of sequence elements that allows to maximize the amount of common information hidden in original merged patterns and comprised by the resulting pattern.

8.1.3 Generalizing Differences by Backtracking Patterns

The pattern specification language comprises a rich set of different patterns to adequately cover the multifariousness of the natural language. In the previous sections we dealt with the important special cases of merging two patterns, i.e. we focused on special instances in the domain of the *merge* function. Since similar rules are merged, it is very likely that a POS pattern will be merged with a POS pattern and not, for example, with a XML pattern. However, since the rule patterns originate from the different text parts, the extraction patterns differ inter alia because of the diversity of natural language. Many of these

differences have certain regularities that we presented at the beginning of this chapter (s. p. 90). The regularities can be exploited to derive a more general pattern including the original patterns and matching other potential expressions of similar information. Backtracking patterns have the capability to concisely capture these common differences and can therefore be leveraged for merging of different patterns.

$merge(A, A) = A$
$merge(*, A) = *$

$merge(A, \oslash) = (A)?$
$merge(A, (A)?) = (A)?$

$merge(A, A\ A) = A\ A^*$
$merge(A, A^*) = A^*$

$merge((A)\%\ B, B\ (A)\%) = (A\ B)\%$
$merge(A\ B, (A\ B)\%) = (A\ B)\%$

$merge(A, B) = (A|B)$
$merge(A, (A|B)) = (A|B)$

The "semi-idempotent" property of the *merge* function is especially important for the generalization: $merge(A, B) = B$, where B is the more general pattern subsuming A (e.g. consider the patterns C and $C \mid D$ with the second pattern subsuming the first one). This semi-idempotence allows to integrate different examples of a general pattern in this pattern without extending the syntax and semantics of the general pattern. The definition of the *merge* function includes a semi-idempotent clause for every backtracking pattern.

8.1.4 Recursive Usage of Backtracking Patterns

While backtracking patterns allow to capture common variations in the natural language, the expressive power of patterns is significantly enhanced when the backtracking patterns are used recursively. When backtracking patterns function as inner pattern of another backtracking patterns, the number of expressible linguistic phrases of the inner pattern is multiplied by the number of expressible instances of the outer pattern (e.g. while the pattern *(A)** can express only one sequences of *A*s with the length n, the patten *(A|B)** expresses 2^n sequences with this length).

To perform merging operations on recursive backtracking patterns it is useful to extend the definition of the *merge* function supplying an algebra for backtracking patterns that allows to simplify complex pattern expressions. Regarding backtracking patterns as an expression with operands and operators we can define some transformations that are backed by the semantics of the patterns (refer for the complete overview to the appendix C), e.g.

$$\begin{aligned} (A?)^* &= A^* \\ (A? \mid A) &= A? \\ (A \mid (B \mid C)) &= ((A \mid B) \mid C) \end{aligned}$$

These transformations are used whenever two patterns are merged, i.e. if the resulting pattern satisfies the left hand side of a transformation, this transformation will be applied to simplify the pattern (cf. the derivation below).

The following example considers the generalization of the attribute BEWEGUNGSRICHTUNG (direction of movement) and its context that is extracted in

the Bosnian corpus (refer to 10.1). The sample derivation below shows the stepwise generalization by merging. During merging the definition of the *merge* function is applied recursively on different nesting levels of extraction patterns (starting with merging the top-level sequence and proceeding with the merging of sequence elements) leading to recursive backtracking patterns.

Starting with two examples of instances of the attribute BEWEGUNGSRICHTUNG from the text a more general pattern is derived and further abstracted by incorporating features of new examples from the text.

in südliche Richtung
$\longrightarrow$ *in ADJ Richtung*
in westliche Richtung

in ADJ Richtung
$\longrightarrow$ *in? ADJ? Richtung NE?*
Richtung Hannover/Elbe

in? ADJ? Richtung NE?
$\longrightarrow$ *in? ADJ? Richtung (NE | nach NN)?*
Entgegengesetzte Richtung nach Norden

This generalized extraction pattern can already match potential attribute values that have a different structure than those used for its derivation, e.g. *in Richtung Sachsen*, *in westliche Richtung nach Frankreich*. Let the assignment pattern *(in? ADJ? Richtung (NE | nach NN)?)=:direction1* denote the generalized extraction of the attribute BEWEGUNGSRICHTUNG. Using the definition of the *merge* function and algebraic transformations of recursive backtracking patterns we can achieve further generalization of the context of the attribute:

(in? ADJ? Richtung (NE | nach NN)?)=:direction1
bewegt sich in Richtung Berlin
↪ *bewegen $direction1$*
seitdem er sich in südliche Richtung bewegt
↪ *(bewegen $direction1$)%*
bewegt sich nach mehreren Stunden in westliche Richtung
↪ *(bewegen PP? $direction1$)%*
bewegt sich langsam Richtung Hannover
↪ *(bewegen (PP? | ADV) $direction1$)%*
bewegt sich am nächsten Tag unerwartet in östliche Richtung
↪ *(bewegen PP? ADV? $direction1$)%*

The already generalized extraction pattern is subsequently merged with the examples of occurrences of the attribute value BEWEGUNGSRICHTUNG in the training texts. The examples correspond with the initial rules, the natural language representation is chosen for better readability. While merging the context of the attribute value referenced by *direction*1 is continuously relaxed and accounting for an increasing number of lexical and syntactic variations of expression.

8.1.5 Merging of Complete Extraction Rules

In the previous sections we introduced the merging function for different kinds of patterns on a formal level. Taking into account that the main purpose of extraction rules is to identify and extract information, certain additional constraints hold for merging of complete extraction patterns. Extraction patterns usually consist of sequences of syntactic constituents and potentially other XML elements. The extracted fragments are distinguished by the assignment patterns that declare what attribute value is matched by the inner pattern of an

assignment pattern. When merging two extraction patterns, the extractions of the same attribute values should be merged and the context around the merged attribute values should be merged correspondingly. Therefore we cannot treat extraction patterns as ordinary sequences and apply the *merge* function defined above for sequence patterns. Instead the merging of extraction patterns can be performed algorithmically.

Rule patterns often contain several extractions of different attribute values, and the order of their appearance plays a major role in the identification of relevant information. Since we are interested in maintaining as much similar information as possible during the generalization, the sequences of extracted attributes in both patterns should be aligned so that the maximum number of attribute matches is achieved. To determine the optimal alignment we again can utilize the $MaxSimScore$ algorithm (setting the value of the $Score$ function to 1 if two attributes are identical and 0 otherwise). After the best alignment is identified, we can generalize the context of extracted attribute values. The generalization is achieved merging context fragments that surround the same attribute values and are at the same position in the respective rule pattern, i.e. the left and right context between the aligned extractions are merged respectively.

Let $Extraction\_Alignment = \{(E_{i_1}, E_{j_1}), \ldots, (E_{i_n}, E_{j_n})\}$ be the optimal alignment of extraction sequences of rule patterns p_i and p_j. We can write the pattern p_i and p_j as $p_i = C_{i_1}\ E_{i_1}\ C_{i_2} \ldots C_{i_n}\ E_{i_n}\ C_{i_{n+1}}$ and $p_j = C_{j_1}\ E_{j_1}\ C_{j_2} \ldots C_{j_n}\ E_{j_n}\ C_{i_{n+1}}$ consisting of aligned extractions and context patterns around them. In this representation the not aligned extractions build the context of aligned extractions and are therefore part of context patterns. Merging p_i and p_j all aligned extractions and the context patterns surrounding the respective extractions are merged:

$$\begin{aligned} merge(p_i, p_j) \quad = \quad & merge(C_{i_1}, C_{j_1})\ merge(E_{i_1}, E_{i_2})\ merge(C_{i_2}, C_{j_2}) \ldots \\ & merge(E_{i_n}, E_{j_n})\ merge(C_{i_{n+1}}, C_{j_{n+1}}) \end{aligned}$$

Thus merging complete rule patterns the biggest possible generalization of extracted attribute values is achieved establishing the optimal alignment of extractions (reaching the biggest number of matching attributes) and obtaining the generalization of context by bringing the context fragments related to identical attributes and situated at the same position in the respective rule patterns together.

8.2 Rule Abstraction

A big advantage of rule merging is that the resulting generalized rules incorporate shared features of many similar less general rules relying on a significant evidence that these features are relevant. Sometimes, however, given a rule, there are no similar rules that can be merged with this rule because it incorporates a not very common expression or unusual structure. Besides, there may be just a few instances of certain attributes in the training texts so that it is difficult to find sufficiently similar rules for effective merging. In such cases the rules can be generalized by rule abstraction.

8.2.1 Relaxation of Context

The rule abstraction is based on the observation that quite often some parts of encoded sentences do not characterize extracted information or contain any relevant context. For example, subordinate sentences, relative clauses usually provide some additional information related to one aspect of the main sentence. If they do not contribute to the identification of relevant fragments, they can be ignored in the linguistic pattern reducing it to the relevant features and making it more general.

A single rule can be abstracted relaxing the specification of context of the extracted item in the rule pattern. Elements of context that do not contribute to identification of a fact are either replaced by a more general element or removed being subsumed by a wildcard pattern. Since both possibilities may have a positive effect, several candidate rules may be generated to be verified in the next step. Generalization by abstraction of single rules is especially effective at the beginning of the learning process.

8.2.2 Abstracting Function

Analogously to the *merge* function the formal structure of the pattern language can be utilized to specify the abstracting heuristics as mathematical function:

$abstract :: Pattern \rightarrow Pattern$
$abstract(A) = PF : \ PForm(A)$ where A is a string (abstraction of a word is its principal form)
$abstract(PF : \ A) = POS : part\_of\_speech(A) \ A$ (abstraction of a principal form of a word is its POS-tag). The function *part_of_speech* returns the POS tag to a given string.
$abstract(POS : pos \ lex\_el) = parent_const(POS : pos \ lex\_el)$ (abstraction of a POS-tag is the syntactic constituent that contains the POS-tag). The function *parent_const* is calculated looking for the first parent that is a syntactic constituent in the ancestor branch of the pos element.
$abstract([A]) = *$ (abstraction of a syntactic constituent is anything)
...
$abstract((A \ A)) = (A)^*$ (abstraction of a sequence of two identical elements is the Kleene closure of these elements)

Repeated application of abstracting function leads gradually to higher degree of generalization up to the subsumption of syntactic units by the wildcard pattern, which is equivalent to their removal from the linguistic pattern. There is no general criterion how many times the abstracting function should be applied to an extraction rule, since the optimal abstraction degree depends on the rule structure, the attributes the rule extracts etc. Therefore several candidates are generated applying *abstract* different number of times, and the optimal abstracted rule is determined empirically evaluating the extraction results on the training corpus (refer to the next chapter).

8.3 Substitution Heuristic

Merging and abstraction of extraction rules are universal generalizing heuristics that can be applied at any stage of the induction process. However, the set of correct rules can achieve a high degree of generality so that neither rule merging, nor rule abstraction can improve extraction results or even change the rule set.

In spite of general rules the recall of such rule set may still be low, i.e. many of expected extractions remain uncovered, which may be caused by several factors. If the text corpus is very heterogenous, the expressions of relevant information in different texts may considerably differ so that general linguistic patterns derived from texts of the training corpus are still not able to cover many instances of relevant information in other texts of the application domain. Another reason may be the small number of training examples for certain attributes so that it is not possible to derive a representative set of extraction rules from the instances of the training corpus.

If the rule set cannot be generalized by merging and abstraction of extraction rules and the recall of such rule set is still low, substitution heuristic can be used to obtain new linguistic patterns that do not occur in the training corpus and enlarge the covering degree and recall of the rule set. As already outlined the main two constituents of a linguistic pattern are the encodings of contexts and extracted parts. Since both constituents fulfil different roles (while context helps to identify the relevant content, encodings of extractions match the extracted fragment and assign it to an attribute of the target structure), they are separable within the linguistic pattern. Considering the fact that the language diversity manifests itself in various possible contexts and manifold expressions of extracted information, we can gain new diverse linguistic patterns replacing the pattern parts that encode extracted text fragments by encodings of other patterns.

Consider the example from the *seminar announcement corpus*. The pattern ** NN:"lecture" [PC: "by" [NC:"Prof." NE*]=:speaker] [VC: VA VV:"take" "place"] [PC: "in" [NC: DT? NE* NN:"Hall" CARD]:=location] ** matches amongst others the sentence *The lecture by Prof. Wolfgang J. Rutenbar will take place in the Wean Hall 5409*. Analogously the pattern ** [NC: "Mr." NE? NE]=:speaker [VC: VV:"present"] [NC: (DT | PRN) NN:"talk"] * "at" ([NC: DT NN:"center"] [PC:"for" NC[]])=:location *]* incorporates the sentence *Mr. Irfan Ali presents his talk on ATM products at the Center for Education, One Kingsway, Edmonton, Alberta, Canada*.
Replacing the encoding of extracted fragments that denote the values of the attributes SPEAKER and LOCATION we can generate four additional linguistic patterns (replacing the encoding of one or both attributes). For instance, the new pattern ** NN:"lecture" [PC: "by" [NC: "Mr." NE? NE]=:speaker] [VC: VA VV:"take" "place"] [PC: "in" ([NC: DT NN:"center"] [PC:"for" NC[])=:location] ** will match a sentence that comprises relevant location and speaker information, but could not be identified by the two original patterns: *Today the lecture by Mr. Kurtz takes place in the Center for Cultural Studies following the workshop on the language history*.

The substitution heuristic is justified by the observation that the context and extracted fragments are often mutually interchangeable. Since a rule usually comprises several attribute values, substituting them by encoding of other patterns leads to a big number of combinations of different encodings in a certain context. Of course, not all of them will reflect expressions that are used in the natural language and in the application domain, some of them may also be ungrammatical. However, these "malformed" linguistic patterns will not damage the extraction quality because they will not match any sentences or text parts (since they do not occur in the real language) causing no incorrect extractions. Thus rules that do not make sense will be filtered during the validation. On the other hand, those rules that represent the valid language and identify relevant information will be validated and added to the set of correct rules increasing the covering degree of the rule set.

8.4 Rule Correction

In spite of sophisticated generalization routines extraction rules will never be perfect. This implies that they will both extract wrong (irrelevant) information and miss expected items. One origin of errors is the diversity of natural language that makes it hardly possible to cover every conceivable formulation of relevant information by an adequate linguistic pattern. Another factor is the loss of precision of extraction rules during the generalization. Trying to capture an increasing number of expression possibilities in one pattern many structural, lexical and syntactic constraints of the extractions and their context are relaxed. Hence it becomes more likely that a general pattern can match a sentence that is similar to those incorporated by the pattern but does not contain any relevant information. The consequence are erroneous extractions that discredit extraction rules in the validation step.

Depending on their origin it can be possible to eliminate certain errors altering the extraction rules. Before discussing possible remedies for mistakes made by extraction rules it is useful to classify them and identify error types that can be corrected improving the deficiencies of the rules.

8.4.1 Types of Errors

In the area of IE the question what extraction should be regarded correct is far from being uniformly solved. We will discuss this issue analyzing different evaluation strategies in sec. 10.3.3. At this stage we can define an extraction to be incorrect if it differs from extractions expected by the human expert. Among these two major classes of errors can be distinguished:

- ▷ Missed extractions. They designate text fragments that are supposed to be extracted but are not extracted by the system. These errors are only partially caused by imperfect extraction rules, e.g. in case when an extraction rule correctly identifies the sentence containing relevant information but fails to localize the correct fragment. Often, however, there are no appropriate linguistic patterns matching the sentence that contains relevant information because no similar sentences occurred in the training corpus. Missed extractions diminish the recall value.
- ▷ Wrong extractions. Text fragments that do not contain any relevant information are mainly extracted because of overgeneralized rules. They have a direct negative effect on precision but may also affect recall if their extraction prevents the extraction of correct fragments.
- ▷ Partial extractions. They occur when the extracted text fragment and the expected extraction overlap, i.e. when a border of extracted fragment lies between the borders of expected extraction or vice-versa. The kinds of partial extractions can be further differentiated (e.g. subsumption by extracted or expected value, real overlap etc.). Partial extractions influence both precision and recall values.
- ▷ Confused attribute. When the borders of expected extraction and extracted fragment correspond the extraction can still be incorrect because the fragment can be extracted as the value of another attribute that is not expected by human expert. Especially in case of semantically close attributes the assignment of the attribute to the extracted fragment can be confused by the system also causing lower precision and recall values.

In contrast to missed extractions wrong and partial extractions are often caused by generalization of extraction rules. Rule merging as well as rule abstraction can involve that a feature distinguishing relevant information from the irrelevant content is replaced by a more general element that looses this capability (e.g. abstracting from certain lexical elements, see the example below). Confused attribute assignments happen usually in the underspecified context and may therefore also be the consequence of rule overgeneralization.

In summary, there are several types of errors that can be committed by extraction rules, and not all of them can be directly addressed by the rule correction. Some errors (especially missed extractions) are supposed to be minimized during the induction process of extraction rules. On the other hand, wrong and partial extractions are primarily caused by too general rules and can therefore be tackled trying to reverse the undesirable effects of generalization.

8.4.2 Rule Correction Algorithm

After having identified the generality of rules as the primary reason for wrong, partial extractions and confused attributes, the correction of extraction rules can consist in addition of constraints that better characterize relevant information. Thus important features that have been lost in the generalization step can be reintroduced during the rule correction making the extraction rule more specific. The rule correction should, however, not completely undo the generalization steps, but integrate only necessary specifications in the already generalized rule.

Similarly as the derivation of extraction rules their correction can be performed inductively too following the general principle: keep the features of positive extractions and exclude the features of negative extractions. In contrast to Ciravegna's rule correction by learning correction rules similarly as the extraction rules [Cir01a] we can exploit the formal nature of our extraction rules provided by the pattern language. The main idea in the correction algorithm is to use the incorrect extractions of a rule (negative examples) as initial instances (analogously to initial rules) for the induction of a general negative counterpart to the original rule. This general negative rule contains the features of the incorrect extractions, that is, of irrelevant information. Comparing the original rule and its negative counterpart the features of irrelevant information can be determined as the differences of both rule patterns. Since these features characterize the irrelevant information, they should not occur in the original extraction rule. Hence, they can be integrated in the original extraction rule in the negated form to explicitly exclude the possibility of matching irrelevant text fragments.

Consider a general extraction pattern derived from the MUC corpus:
** NC:=Victim_target [VC: VPP "kill"] [PC: (because of| by) [NC: (ART ADJ*)? "bomb" NN]] **
This rule pattern matches inter alia this sentence containing relevant information:
Several civilians were killed because of a bomb explosion in San Ramon on Thursday.
However, it also matches two sentences that do not contain relevant information because the human experts do not regard information communicated by reported speech as factual:

The leader of separatists Jose Ramirez claimed that two residents were killed by a recent bomb attack conducted by governmental troops.

General Bustillo said that four individuals were killed by bombs that have been planted by terrorists in the downtown of San Miguel.

Merging both negative examples we obtain the negative counterpart to the original rule pattern:
** NC [VC: SynSet("assert")] that NC:=Victim_target [VC: "be" "kill"] [PC: by [NC: (ART ADJ)? "bomb" NN]] **

Comparing both patterns we detect that the first sequence elements (basically representing reported speech) in the negative pattern are not comprised by the original pattern. Thus we can include them in the negated form in the original pattern obtaining new corrected pattern: ** !(NC [VC: SynSet("assert")] that) NC:=Victim_target [VC: VPP "kill"] [PC: (because of | by) [NC: (ART ADJ)? "bomb" NN]] **

Recall that the negation pattern states explicitly what should not occur and does not consume any tokens matching an empty fragment. The corrected pattern does not match two irrelevant sentences still matching the sentence with relevant information.

The merged negative pattern will be with a high likelihood more specific than the rule pattern, but certainly not more general, since the negative pattern incorporates exactly two initial rules while the rule pattern should subsume at least two initial rules incorporated during merging. Therefore for each element of negative pattern at least as general counterpart in the rule pattern can be found. We can define a *diff* function, which aligns the rule pattern and negative patterns so that every element in the sequence of negative pattern is matched by an element of the sequence of the rule pattern. It identifies subsequently the more specific contextual elements of the negative pattern. Context specified by these elements may be typical for irrelevant information. Therefore the more specific contextual elements are negated and inserted at the same position in the sequence of the rule pattern before their more general counterpart. Here the non-substantial character of negation is optimally exploited allowing to specify what should not occur without changing the original context pattern in the rule pattern sequence.

The algorithm for rule correction in fig. 8.1 builds on the idea of correction by excluding the features of negative extractions. The induction of negative examples and exclusion of potential differences is continued until the corrected rule achieves an acceptable precision value or until no new negative examples can be induced.

Let P be the pattern of extraction rule $R = P \rightarrow E$ that should be corrected and $N_1, \ldots, N_n$ - the sentences from that R made wrong extractions.

```
for (i=0; i<n; ++i)
    NP_i=encode_initial_pattern(N_i);
for (i=0; i<n; ++i) \\Construction of a heap storing similarity values
    for (j=i+1; j<n; ++j) \\for any two negative patterns
        pattern_sim_heap.put((i,j), RuleSim(NP_i, NP_j));
do  {(k,l)=pattern_sim_heap.removeTop();
     Gen_Neg_Pattern=merge(NP_k, NP_l);
     D_1 ... D_i =diff(P,Gen_Neg_Pattern);
     P =insert_negated(D_1 ... D_i, P);
     precision=apply_rule(P → E).get_precision();
    }while (precisison<prec_threshold && !pattern_sim_heap.isEmpty());
```

Figure 8.1: Algorithm for correction of extraction rules

8.4.3 Limitations of Rule Correction

The algorithm presented in fig. 8.1 does not cover the complete range of deficiencies of extraction rules. It is targeted at improvement of contextual specification excluding the typical context features of incorrect extractions. If however the characteristic features of negative examples lie in the specification of extracted fragment, the *diff* function is not able to detect this difference.

To determine structural difference in the extracted fragment positive extractions of the rule pattern have to be regarded too. The difference between the merged negative pattern and all initial patterns of positive extractions has to be calculated. But here the limits of rule correction can be recognized. In some cases the building of the difference is possible only on the lexical level (e.g. when the specification of negative and positive examples are identical on the syntactic and morphological level). The result would be an enumeration of lexical values that should not occur at the certain positions of extracted fragments. Such a correction may be useful in the training corpus because of adaptation to training texts, but lacking any abstraction it will hardly benefit the overall goodness of the rule set in a real application.

Even though the idea that every rule can be adequately corrected to achieve an acceptable extraction quality is quite illusionary, rule correction is a powerful mean to improve the quality of rules and to increase the coverage of the rule set allowing more rules to pass the precision threshold (s. experimental investigation in sec. 12.4.1).