7 Induction of Extraction Rules

In the previous chapter we presented a different view on our approach to information extraction regarding patterns of extraction rules as XML queries and the pattern unification as the evaluation of XML queries. This view does not change the essence of our approach, which is learning the extraction rules from a number of training examples by collecting relevant context and lexical, structural and linguistic features of extracted information in extraction patterns. The learning is accomplished by the formal induction of extraction rules specified in the pattern language presented in the previous chapter. The rule induction has the goal to derive general and at the same time reliable rules that are capable of extracting information from any text in the regarded domain. This chapter illustrates the beginning of the induction cycle (cf. fig. 4.2) dealing with the generation of initial rules as the induction basis and assessment of rule similarity while generalization, correction and validation steps are presented in the next chapters.

7.1 Generation of Initial Rules

The rule induction begins with the set of initial rules that are derived from the training examples provided by human annotator. Basically, initial rules are able to extract only the training example they are derived from. They specify the exact context and comprise all linguistic and structural features of the extracted example. Certainly, these rules do not have any abstracting potential, but capturing a multiplicity of features they provide a very good source for generalization of rules.

7.1.1 Localization of Extracted Fragments in the Training Documents

After the linguistic preprocessing extraction annotations are inserted in the preprocessed XML document. The lowest level of linguistic processing consists of POS elements that represent single tokens. If the textual content of a POS element is extracted, the extraction is annotated by inserting the element EXTRACTED as the child element of the POS element. Inserting annotations as children of the lowest linguistic nodes in the XML hierarchy allows to maintain the original XML structure of the preprocessed document at this stage. Figure 7.1 demonstrates a fragment of the preprocessed document from the figure 5.1.

```
...
    <CONST TYPE="NC">
        <POS TYPE="CD" NORMAL="seven">SEVEN
            <EXTRACTED TABLE="T-ACT" ATTRIBUTE="ANIMATE_VICTIMS"/>
        </POS>
        <POS TYPE="NNS" NORMAL="soldier">SOLDIERS
            <EXTRACTED CONTINUED="true" TABLE="T-ACT"
            ATTRIBUTE="ANIMATE_VICTIMS"/>
        </POS>
    </CONST>
...
```

Figure 7.1: A fragment of the preprocessed text in fig. 5.1 with annotated extractions

The EXTRACTED elements specify the relation and the attribute the extracted token is assigned to by their attributes TABLE and ATTRIBUTE. To distinguish the beginning and continuation of extracted fragment the attribute CONTINUED is used indicating that the token is a continuation of an extraction if its value is true. Sometimes the tokenization performed by TreeTagger is not fine enough (e.g. fragments containing punctuation signs are regarded as one token like *3:00-5:00*). If only a fragment of the token is extracted (e.g. *3:00*), START and END attributes will mark the beginning and ending character.

The preprocessed documents enriched with the annotations of extracted content include the complete information that is available to the system for training and are therefore used for the generation of initial rules. To locate all extracted items in the documents a simple XML query selecting all EXTRACTED elements is issued. After an extracted fragment has been localized in the document, an initial extraction rule for this fragment is constructed.

7.1.2 Choosing the Appropriate Context

An extraction rule is not only supposed to reflect the features of the extracted item but to capture its relevant context. A fix context window model used by the most IE approaches does not adequately reflect the relevant context often disrupting connected syntactic and semantic structures. As we have already discussed, we regard the sentence as the fundamental semantic unit of the natural language. The sentence is the smallest linguistic structure that can express complete thoughts, descriptions, events. That is why we consider the sentence as the best general context model for information extraction. The fact that the sentences in the most cases comprise extractions of several attributes of a relation provides additional evidence for its goodness. Furthermore, this fact is one of the major factors making the extraction rules more reliable. If an extraction rule finds evidence of several extracted attribute values, the likelihood that their extractions will be correct is generally higher than in case that the evidence of a single relevant fragment is found.

An initial rule for extraction of a certain fragment is built by encoding the sentence, in which the fragment occurs, in the pattern language. Sentences are recognized at the preprocessing stage and designated by the SENT element. The corresponding sentence can therefore be found looking upward in the ancestor branch of the EXTRACTED element.

The main focus of GROPUS is fully grammatical free texts in that the information is contained in sentences. However, GROPUS is intended to handle any

kinds of text including ungrammatical, informal, telegraphic in style documents. In such texts there is usually no sentential structure and information can be expressed in headlines or table- or form-like passages. Therefore if the extracted information is not included in a sentence, alternative context has to be identified. For this purpose the structural information identified by *txt2html* (see sec. 5.1) can be leveraged. In the absence of obvious semantic connection the structural connection between the extracted item and its context can be utilized. If structures like lists, tables, preformatted text are recognized, it is possible that the structure itself and the content within the structure help to identify the extracted fragment and occur in other documents. Therefore if no SENT ancestor of extracted item can be found, common structural HTML elements (DL, TR, PRE, H_i etc.) are looked for.

The sentence or structural element comprising an extraction fulfil a dual role defining the context and at the same time the sphere of action of extraction rules. If an extracted item is nor comprised by a sentence neither subsumed by a structural element, a parent element of the extracted token (which is normally the syntactic constituent) is chosen as the context node.

7.1.3 Translation of Extractions and their Context in Pattern Language

After the appropriate context has been identified, the rule pattern can be generated. The initial rules follow the syntactic and lexical structure of the sentences[1] they are derived from. No information should be lost at this stage that might be important for the refinement of the rules. Therefore the rules are very specific since they capture the contexts on a rather concise level and will usually be able to extract only facts from the encoded sentence. The initial rule patterns reflect every feature identified on the preprocessing stage, in particular:

- ▷ Exact syntactic structure of the sentence. The order and nestings of all syntactic constituents and part of speech tags are encoded using syntactic category pattern and pos pattern (cf. appendix A).
- ▷ Lexical specification. The words occurring in the sentence and their principal forms are captured as the inner patterns of the POS pattern
- ▷ Specification of extractions. All extracted fragments are distinguished by assignment patterns that mark their borders and specify the attribute of the target structure represented by the fragments.
- ▷ Text layout and structure. Since the layout and structure are expressed by XML elements, the initial rule patterns reproduce the same hierarchical structure using XML patterns.
- ▷ HTML/XML structure. Analogously to layout any XML or HTML elements contained in the sentence are incorporated in the rule pattern. Their interleaving with the syntactic and layout elements is reflected by the nesting structure of the rule pattern.

Our sample sentence from the sec. 4.1 depicted in fig. 7.2 will be encoded as

[1] In the following we will refer to the context nodes as sentences without loss of generality

```
<SENT>
    <CONST TYPE="NC">
        <POS TYPE="NE" NORMAL="General">General
            <EXTRACTED TABLE="T-ACT" ATTRIBUTE="VICTIM_TARGET"/>
        </POS>
        <POS TYPE="NE" NORMAL="Bustillo">Bustillo
            <EXTRACTED CONTINUED="true" TABLE="T-ACT"
            ATTRIBUTE="VICTIM_TARGET"/>
        </POS>
    </CONST>
    <CONST TYPE="VC">
        <POS TYPE="VBD" NORMAL="be">was</POS>
        <POS TYPE="VVN" NORMAL="kill">killed
            <EXTRACTED TABLE="T-ACT" ATTRIBUTE="ACTION"/>
        </POS>
    </CONST>
    <CONST TYPE="PC">
        <POS TYPE="PP" NORMAL="by">by</POS>
        <CONST TYPE="NC">
            <POS TYPE="DT" NORMAL="a">a</POS>
            <POS TYPE="NN" NORMAL="bomb">bomb
                <EXTRACTED TABLE="T-ACT" ATTRIBUTE="WEAPON"/>
            </POS>
            <POS TYPE="NN" NORMAL="explosion">explosion</POS>
        </CONST>
    </CONST>
    <POS TYPE="SENT" NORMAL=".">.</POS>
</SENT>
```

Figure 7.2: Preprocessed sentence used for generation of an initial rule

[NC: NP:"General" NP:"Bustillo"]:=VICTIM [VC: VBD:"be" (VVN:"kill"):=ACTION] [PC: PP:"by" [NC: DT (NN:"bomb"):=WEAPON NN:"explosion"]] →
INSERT INTO TERRORISTACT VALUES (VICTIM_TARGET „, ACTION„ INSTRUMENT„)

The extracting action (the right hand side of the rule) specifies what attributes the extraction rule is supposed to extract. This information is used in a later generalization step for assessing the rule similarity.

7.1.4 Encoding of Extractions

In our example in the figure 7.2 the extracted fragments correspond with the syntactic structure of the sentence, that is, all extractions lie within certain syntactic constituents. However, sometimes extracted fragments may cross the borders of syntactic elements and overlap, e.g. with two different syntactic chunks. This can happen due to a wrong syntactic analysis when, for example, the TreeTagger splits a syntactic constituent in two different units. Moreover, extracted fragments may also be in conflict with other XML structures or HTML layout elements beginning inside and ending outside their borders. Since the pattern structure resembles the XML structure and is strictly hierarchical, a pattern can only be subsumed by at most one pattern (acyclic property). If the tokens (POS elements) of extracted fragment belong to different parents and there is at least one sibling token that is not extracted, they cannot be subsumed by a single assignment pattern because the latter cannot be subsumed by different parent

patterns (see fig. 7.3). In this case the XML structure has to be adjusted to comply with extractions by removing the elements that disrupt the sequence of extracted tokens and attaching their children to their parents.

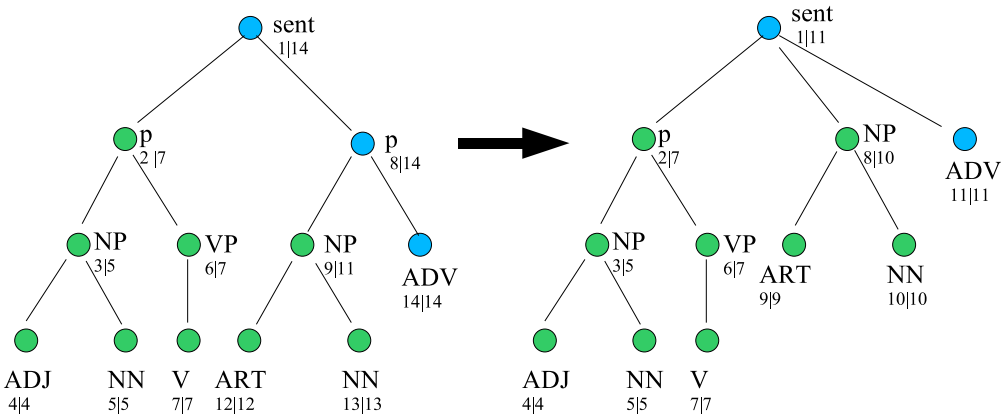

Figure 7.3: Determination of extracted and inconsistent elements and establishing consistent state

The extracted fragments are represented by a sequence of POS elements, which are at the lowest level of XML hierarchy. Their parents, the syntactic constituents may be consistent with the extractions, while the overlap may occur at any level of hierarchy (cf. in the fig. 7.3 the syntactic constituent NP_9[2] comprises two extracted tokens and is therefore itself extracted, while its parent p_8 subsumes extracted and not extracted fragments and is therefore inconsistent). Thus the complete XML sentence structure has to be examined whether it complies with the extractions.

Let $S = e_a \dots e_n$ be the sequence of tokens (leaf nodes of the XML tree) extracted as the value of attribute A:

$$\forall e_i \in S \; extraction(e_i) = A$$
$$\forall e_i \notin S \; extraction(e_i) = \oslash$$

We can mark the borders of each element by the attribute name of the leftmost and rightmost tokens subsumed by this element:

$$\begin{array}{ll} if\ n_i\ is\ a\ token, & LeftBorder(n_i) = extraction(n_i) \\ i.e.\ \nexists m\ pred(n_i, m) & RightBorder(n_i) = extraction(n_i) \\ otherwise, i.e. \exists m\ pred(n_i, m) & LeftBorder(n_i) = LeftBorder(n_{i+1}) \\ & RightBorder(n_i) = RightBorder(n_{RightBound(n_i)}) \end{array}$$

Since the adjacent sibling nodes play an important role when deciding whether a node is conform with extractions, we define the siblings as the surrounding children of the parent node:

$$\begin{array}{ll} LeftSibling(n_i) = n_f \Leftrightarrow & i = RightBound(n_f) + 1 \wedge \neg pred(i, i-1) \\ RightSibling(n_i) = n_k \Leftrightarrow & k = RightBound(n_i) + 1 \wedge \\ & \wedge\ \nexists n_p\ (pred(p, i) \wedge RightBound(n_i) = RightBound(n_p)) \end{array}$$

In the definition of siblings we can exploit the fact that the difference between the OID of the right sibling of a node and its maximum descendant OID is 1 according to preorder numbering. Besides, the leftmost child has no left and the rightmost - no right sibling so that we have to exclude these cases imposing additional constraints (in case of the leftmost child the difference between its OID

[2] In the following we will use the abbreviated index form for noting the OID of the node introduced in the previous chapter: $n_i \Leftrightarrow OID(n) = i$

and the OID of its parent is 1; the rightmost child has at least one predecessor with the same right bound - its parent).

To algorithmically determine whether a node is extracted and should be included in the assignment pattern or whether it does not comply with extracted sequence and should be removed, we can use a top-down recursive procedure that decides about the status of single nodes based on the following criteria[3]:

$$
\begin{aligned}
&n_i \; is \; extracted \Leftrightarrow LeftBorder(n_i) = RightBorder(n_i) \\
&n_i \; is \; inconsistent \Leftrightarrow LeftBorder(n_i) \neq RightBorder(n_i) \wedge \\
&\wedge \; (LeftBorder(n_i) = RightBorder(LeftSibling(n_i)) \neq \oslash \vee \\
&\vee \; RightBorder(n-i) = LeftBorder(RightSibling(n_i)) \neq \oslash
\end{aligned}
$$

According to the definition p_2 in the fig. 7.3 is extracted because its borders are identical (i.e. the leftmost ADJ_4 and the rightmost V_7 subsumed tokens belong to the extraction of the same attribute). On the contrary, the element p_8 is inconsistent because its left border is identical to the right border of its left sibling (p_2) and is therefore a continuation of an extraction while its right border is equal to $\oslash$ (i.e. the rightmost token is not extracted). Comprising a part of an extraction and a not extracted fragment the element p_8 contradicts the structure of the extraction and has to be removed. The consistent state after the removal of p_8 is represented at the right side of the fig. 7.3.

Before encoding the initial pattern the borders of every XML element are determined and inconsistent elements are removed. Inconsistency can always be resolved at least at the token level because the extraction can always be represented as a sequence of extracted tokens abandoning any linguistic or other XML structure. However, the definition of extracted elements implies that during encoding of extractions the XML elements should be captured at the most general level. If, for instance, all tokens of a syntactic constituent are extracted, the assignment pattern will include this element and not only the sequence of extracted tokens as a part of the extraction structure. In our example the extraction of the attribute A will be encoded as `(\p[[NP: ADJ NN][VP:V]][NP: ART NN])=:A`.

Mapping the training examples to the linguistic patterns the initial rules eventually move away from the natural language towards formal representation. Capturing all relevant features and context of extraction they provide the basis for induction of new more general and reliable rules.

7.2 Rule Similarity

The spectrum of initial rules may to a large degree depend on the kind of training texts and be very heterogeneous. If a text contains a semistructured and a free text part (as we will see later in the *seminar announcement corpus*), the rules extracting information from the form-like parts have a totally different structure in comparison to rules that are encoded on the sentence basis. Generally, rules that extract different attribute values are not supposed to have much in common because different attribute values imply different features and context.

To achieve effective generalization of extraction rules, one has to find common properties of extractions and their context that distinguish and uniquely char-

[3] This definition neglects that non-syntactic, e.g. HTML elements that do not contain textual data and therefore do not subsume any pos elements may be as well at the borders as between the pos elements. The borders of non-syntactic elements are adjusted according to the surrounding elements subsuming textual data.

acterize relevant information. Therefore the rule similarity is one of the most important factors for rule generalization. If rules are similar, the confidence is enhanced that the features and the context captured by the rules are relevant and reliable because they occur more than once and possibly in different texts. The characteristic properties of extracted information can be distilled merging similar rules while the specific properties of concrete training examples that do not contribute to identification of relevant content in general can be filtered.

In the following the challenges of determination of rule similarity and solutions applied in our approach including an algorithm for sequence comparison are described. Based on these solutions a rule similarity measure is defined at the end of the chapter.

7.2.1 Mapping Hierarchies to Sequences for Comparison

One of the major difficulties comparing the extraction rules is that they simultaneously incorporate hierarchical XML structures and sequential properties of natural language. Similar sequences of elements may therefore occur at the different levels of hierarchies. To find these similarities we have to compare extraction patterns in the same fuzzy manner as in the pattern unification (see previous chapter) omitting levels of hierarchy. However, the comparison is complicated by the fact that we have to compute the similarity score for every possible configuration of both patterns to determine their most similar configurations and the maximum similarity score. For example, the most similar configurations of patterns displayed in fig. 7.4 are achieved when the level of syntactic categories (elements *NP* and *VP*) in the left and the elements *p* and *b* in the right pattern marked red are removed.

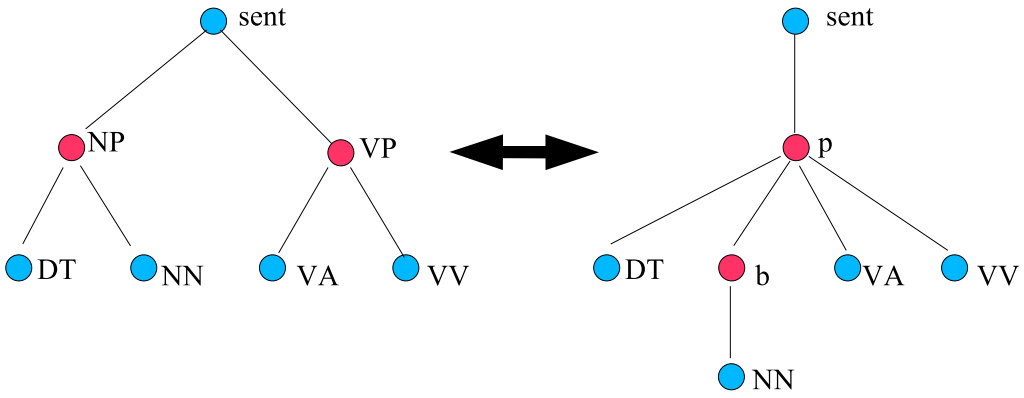

Figure 7.4: Comparison of two extraction patterns in hierarchical representation

To account for all different configurations any subset of inner patterns can be removed resulting in $\sum_{i=0}^{n} \binom{n}{i} = 2^n$ configurations for a single extraction pattern and $2^n * 2^m = 2^{n+m}$ combinations of different configurations where n and m are the number of inner patterns in both extraction patterns respectively. The number of inner patterns can be estimated by $O(t)$ where t is the number of tokens (leaf elements in the hierarchy), if an inner pattern includes at least two child patterns in average. Therefore the runtime required for determination of similarity of two patterns naively considering all possible hierarchical structures would lie in $O(2^{t_1+t_2} * (t_1 + t_2))$ (second factor denotes the time needed for comparison of hierarchies). Such a computation is practically not feasible because of exponential asymptotic runtime.

Regarding patterns as labels we can reduce the comparison of pattern hierarchies to the *tree edit problem* [Bil05]. Using the algorithm proposed by Tai [Tai79] we

could reduce the time complexity to $O(N * M * t_1 * t_2)$ where $N = n + t_1$ and $M = m + t_2$ denote the total number of nodes in both hierarchies. Zhang and Shasha proposed an algorithm [Zha89] improving the bounds to $O(N * M * min(t_1, D_1) * min(t_2, D_2))$ where D_i is the respective height of the hierarchy. The worst case bound has been decreased by Klein [Kle98] to $O(N^2 * M * logN)$. Chen presented an algorithm using $O(N * M + t_1^2 * M + t_1^{2.5} * t_2)$ [Che01], which in our case corresponds to the polynomial of fourth degree since we refer to the number of tokens as the problem size. Robinson-Foulds distance [Rob81] provides a tree distance measure that is linear in the sum of node numbers of both trees – $O(N + M)$. This metric is, however, applicable only to trees that comprise the same set of labeled nodes. Thus reducing patterns to labels we can avoid exponential runtime of comparison requiring though the runtime with the upper bound equal to fourth-degree-polynomial of the number of tokens, which still imposes a severe computational burden.

The key idea that enables a proper and efficient comparison of similarity of extraction rules in our approach is to flatten hierarchies of extraction patterns to sequences and compare the similarity of the latter. The sequential representation of hierarchies, which maintains the order of sibling sequences and represents the hierarchical relation between parent and child nodes by positions in the sequence, is obtained writing down the elements during the pre-order traversal of the hierarchy. Even though the original hierarchy cannot be reconstructed from its sequential representation (the functional mapping is not injective), a reliable estimation of similarity can be achieved by sophisticated comparison of sequences. The sequences are aligned by shifting and dispersing both sequences so that the number of identical elements is maximum. Consider the comparison of sequential representations of the extraction patterns from the fig. 7.4:

sent	NP	DT		NN	VP	VA	VV
sent	p	DT	b	NN		VA	VV

Dispersing and shifting the upper sequence between *DT* and *NN* we achieve the correspondence on the token level implicitly omitting the higher hierarchical level (element *b*). Generally, the non-identical elements (marked red) are not considered because of shifting while identical elements and sibling sequences are aligned. The more accordances exist, the more similar the extraction patterns are. Accordances in subsequences may also indicate the same hierarchical structures. In spite of variety of nesting structures caused by merged HTML and linguistic markup (refer to 6.1) there are certain fix hierarchical dependencies, for instance, between linguistic elements (e.g. prepositional phrase usually subsume nominal phrase but is never subsumed by nominal phrase). If both pattern sequences contain a subsequence *PC PP NC*, it is very likely that both comprise a prepositional phrase subsuming a nominal phrase and not likely that PP and NC are on the same level of hierarchy. Therefore the similarity of sequential representations provides a strong evidence of hierarchical similarity.

The main advantage of mapping hierarchies to their sequential representations is that sequences can be efficiently compared. We developed an algorithm that compares sequences in a minimum time and represents also the core element of rule merging (described in the next chapter).

7.2.2 Algorithm for Comparison of Sequence Similarity

Among many possibilities of measuring sequence similarity we are interested in a measure that reflects all common features of sequences. For this purpose we align the sequences by shifting and dispersing them to achieve the maximum number of accordances between the sequence elements. Given the sequences $A = A_1 \dots A_m$ and $B = B_1 \dots B_n$ an *alignment* L of A and B is defined as an ordered set of pairs of aligned sequence elements: $L = \{\dots (A_i, B_j), (A_k, B_l) \dots\}$ where $i < k$ and $j < l$. The similarity measure can then be defined as a function $MaxSimScore(A, B)$ that identifies the alignment with the maximum number of common elements and calculates its similarity value.

Similarity of Non-Sequential Elements

However, before analyzing similarity of sequence patterns we have to consider how the similarity of sequence elements can be assessed. The view suggested in the previous section that elements may be either identical or not is somewhat simplistic, though it serves well for explanation of sequence alignment. Indeed, the degree of similarity of some patterns is not binary, but also has to be calculated. For instance, two POS patterns may encode the same part of speech, but different or identical lexical content. Analogously, two XML element patterns may encode the same element, but the similarity of their children encoded as sequence patterns varies.

To assess the similarity of non-sequential patterns we define the function $Score$, which assigns to two patterns a similarity score (a relative numeric value)[4]:

$$
\begin{aligned}
&Score(String_1, String_2) = (String_1 \equiv String_2)?5 : 0 \\
&\dots \qquad \text{where } \equiv \text{ denotes lexicographical equivalence} \\
&Score(POS : String_1[PF : String_2], POS : String_3[PF : String_4]) = \\
&= (String_1 \equiv String_3)?((String_2 \equiv String_4)?5 : 2) : 0 \\
&Score(\backslash Tag_1, \backslash Tag_2) = (Tag_1 \equiv Tag_2)?5 : 0 \\
&\dots \\
&Score((A_1 \dots A_m)?, (B_1 \dots B_n)?) = MaxSimScore(A_1 \dots A_m, B_1 \dots B_n) \\
&Score((A_1 \dots A_m)?, (B_1 \dots B_n)^*) = 0
\end{aligned}
$$

The inner patterns of two backtracking patterns are compared only if their kind is identical (i.e. two option patterns). Comparing the POS pattern the lexical content provides a much more precise description than just the part of speech. That is why the correspondence of lexical content is weighted stronger than the correspondence of parts of speech (ratio 5:1). Assessing the similarity of two XML element patterns the correspondence of the tags is rewarded by 5 similarity points. Recall that the XML element patterns cannot contain any children sequences because the hierarchical structure has been flattened by the sequential representation.

Determination of Sequence Similarity

After establishing a similarity measure for the elements of sequences the task of determination of similarity of two sequences can be defined as follows:
Given two sequences $A_1 \dots A_m$ and $B_1 \dots B_n$, find an alignment $L = \{\dots, (A_i, B_j), \dots\}$ so that $\sum_{i=1}^{|L|} Score(l_i)$ is maximum where $l_i \in L$.

[4] Instead of conventional "if (A) then B else C" notation we use the ternary operator (A)?B:C known from C and Java programming languages for brevity

```
for (i=0; i<m; ++i)
    for (j=0; j<n; ++j)
        {cand1=0, cand2=0, cand3=0;
         if (i>0) cand1=max_sum[i-1][j][0];
         if (j>0) cand2=max_sum[i][j-1][0];
         cand3=((i>0 && j>0)?max_sum[i-1][j-1][0]:0)+Score(A_i,B_j);
         if (cand3>cand2 && cand3>cand1)
            max_sum[i][j][0]=cand3; max_sum[i][j][1]=2;
            if (cand2>cand1 && cand2>cand3)
                max_sum[i][j][0]=cand2;max_sum[i][j][1]=0;
            else
                max_sum[i][j][0]=cand1; max_sum[i][j][1]=1;
        }
i=m-1; j=n-1;
do  {if (max_sum[i][j][1]==2) alignment.addFirst((i--,j--));
     else if (max_sum[i][j][1]==1) i--;
        else if (max_sum[i][j][1]==0) j--;
    }while (i>=0 && j>=0);
return alignment;
```

Figure 7.5: Algorithm for determination of maximum similarity score and corresponding alignment of two sequences

This problem can be viewed as an extension of the well-known *Longest Common Subsequence* problem (s. [Cor90]). The LCS problem is a special case of the sequence alignment problem, where the *Score* function returns 1 if its arguments are identical and 0 otherwise. In a general case, no restrictions are imposed on the *Score* function, except that its values must be positive. The related algorithms computing the *edit distance* of two strings proposed by Levenshtein and Damerau [Lev66], [Dam64] do not solve our problem. They calculate the distance between two sequences based on the minimum number of correcting operations, each of which "costs" 1 distance unit. Our problem is to compute the maximum similarity of two sequences based on the *Scores* of aligned elements, which are real numbers. The number of necessary dispersing and shifting operations (corresponding to deletion in Levenshtein's algorithm) is insignificant, because the alignment depends solely on the aggregate *Score* function.

However, regarding sequence elements as letters of a fix alphabet and interpreting the values of the *Score* function as weights of the scoring matrix the problem can be mapped to the determination of *alphabet-weight edit distance* [Gus97].

The algorithm presented in fig. 7.5 determines the maximum similarity score and the corresponding sequence alignment using dynamic programming. It is based on the invariant

$$MaxSimScore(A_1 ... A_m, B_1 ... B_n) =$$

$$max \begin{pmatrix} MaxSimScoreS(A_1 \dots A_m, B_1 \dots B_{n-1}) & \text{omitting} B_n \\ MaxSimScore(A_1 \dots A_{m-1}, B_1 \dots B_n) & \text{omitting} A_m \\ MaxSimScore(A_1 \dots A_{m-1}, B_1 \dots B_{n-1}) + Score(A_m, B_n) & \text{aligning} A_m \text{ and } B_n \end{pmatrix} \quad (7.1)$$

The algorithm maintains a table $max\_sum[m][n][k]$ in that $max\_sum[i][j][0]$ denotes the maximum similarity score of subsequences $A_1, \dots, A_i$ and $B_1, \dots, B_j$ and $max\_sum[i][j][1]$ – the numeric encoding of the direction of the predecessor that participated in building $max\_sum[i][j][0]$ (i.e. which of the three cases in 7.1 provides the maximum value). The direction of the predecessor is necessary to reconstruct the path from the total maximum similarity score to $max\_sum[0][0][0]$, i.e. to recall the pairs of aligned sequence elements that maximize the score.

For every combination of subsequences the $MaxSimScore$ is calculated using (7.1) and the predecessor recorded. Once the total maximum similarity score $max\_sum[m-1][n-1][0]$ is determined, the alignment leading to this score is added to the result list. Using the direction value the preceding alignments can be determined. The result list is complete when the first element of one of the sequences is reached. Since the path to the maximum score is reproduced starting from its end, alignments are added at the front of the result list to establish their correct order.

Correctness Proof

The presented algorithm calculates the value of $max\_sum[i][j]$ for any i and j according to the invariant (7.1). To prove the correctness of the algorithm we have to show that $max\_sum[m][n][0]$ is maximum for all possible alignments of A and B. The correctness of (7.1) implies that $MaxSimScore(A, B)$ is the maximum similarity score of A and B (since the function $MaxSimScore$ returns the maximum score per definition, see 7.2.2, p. 83). To prove the correctness of the algorithm it is therefore sufficient to prove the correctness of the invariant (7.1). We provide a proof of (7.1) by an induction *I1* over the length of the second sequence and an implication *I2* from the sequence pairs with the lengths of the length $(m-1, n), (m, n-1)$ and $(m-1, n-1)$ to the sequence pair with the length (m, n). Combining the induction and the implication we can show the validity of the invariant for any sequences.

At first we prove that the equation 7.1 is valid for a sequence of one element and any other sequence by the induction *I1*:

Induction Basis:
$A = A_1$, $B = B_1$: $MaxSimScore(A, B) = 0 + Score(A_1, B_1)$ – equation (7.1) is trivially fulfilled (only third case is possible).

Induction Hypothesis:
Equation 7.1 is valid for A and $B = B_1 \dots B_n$ implying that MaxSimScore(A,B) is the maximum similarity score for A and B

Induction Step:
Aligning A_1 with $B_1 \dots B_{n+1}$ two cases are possible:
(I) B_{n+1} participates in the alignment (is aligned with some element from A): in this case it can only be aligned with A_1 and the maximum similarity score corresponds to $Score(A_1, B_{n+1})$.
(II) B_{n+1} does not participate in the alignment: the maximum similarity score is then the maximum similarity score obtained by aligning A_1 and B, which is $MaxSimScore(A, B)$ according to our induction hypothesis.
The maximum similarity score of A_1 and $B_1 \dots B_{n+1}$ is therefore the maximum of both cases $MaxSimScore(A, B_1 \dots B_{n+1}) = max(MaxSimScore(A, B), 0 + Score(A_1, B_{n+1}))$, q.e.d.

The proof for $A_1 \dots A_m$ and B_1 is absolutely analogous and is therefore omitted.

In the next step we prove the implication *I2* that if the invariant is valid for three pairs of sequences $A_1 \dots A_{m-1}$ and $B_1, \dots, B_{n-1}$; $A = A_1 \dots A_m$ and $B_1, \dots, B_{n-1}$ and for $A_1 \dots A_{m-1}$ and $B = B_1, \dots, B_n$, it is valid for $A = A_1 \dots A_m$ and $B = B_1, \dots, B_n$.

Given:
The equation 7.1 is valid for $A' = A_1 \dots A_{m-1}$ and $B' = B_1, \dots, B_{n-1}$; $A = A_1 \dots A_m$ and B' and for A' and $B = B_1, \dots, B_n$ (implying that

$MaxSimScore(A', B')$, $MaxSimScore(A', B)$ and $MaxSimScore(A, B')$ are maximum similarity scores for these three pairs of sequences).

To be proved:
The equation 7.1 is valid for $A = A_1 \dots A_m$ and $B = B_1, \dots, B_n$

Proof:
Aligning A and B four cases are possible:
(I)A_n participates and B_n does not participate in the alignment: Maximum similarity score is equal to the maximum similarity score obtained aligning A and B', which is MaxSimScore(A,B') according to the induction hypothesis.
(II)A_n does not participate and B_n participates in the alignment: Analogously to the first case maximum similarity score is MaxSimScore(A',B).
(III)A_n and B_n participate in the alignment: since both are the last elements in their sequences, they can only be aligned with each other and the maximum similarity score is the sum of the maximum similarity score obtained by aligning A' and B' (which is MaxSimScore(A',B') according to the induction hypothesis and $Score(A_m, B_m)$, i.e. $MaxSimScore(A', B') + Score(A_m, B_m)$
(IV)A_n and B_n do not participate in the alignment: this case corresponds to the third case with $Score(A_m, B_n) = 0$.

The maximum similarity score of A and B is therefore the maximum of the three first cases:
$MaxSimScore(A, B) = max(MaxSimScore(A', B), MaxSimScore(A, B'),$
$MaxSimScore(A', B') + Score(A_m, B_n))$, q.e.d.

Using the proved propositions the validity of the invariant (7.1) can be shown for sequences A and B of any length. Starting from the induction basis of the induction *I1* we can show the validity of (7.1) incrementally increasing the size of sequences as shown in fig. 7.6. For example, to show the validity for sequences with lengths 2 and 2, according to the proposition *I2* we have to show the validity for sequences with lengths 1 and 1 (induction base), 1 and 2, 2 and 1 (both proved by induction *I1*). The validity for lengths 3 and 2 results from the validity of 2 and 1, 3 and 1 (both proved by *I1*) and 2 and 2 (proved above).

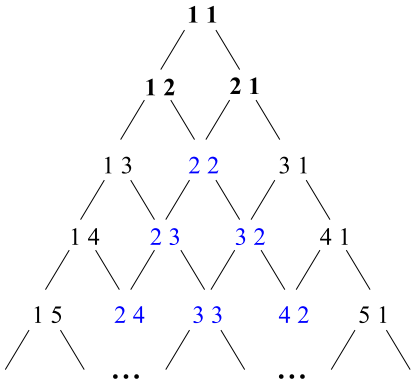

Figure 7.6: Proving the validity of the invariant 7.1 for any sequence lengths

Time Complexity of the Algorithm

Statement: To determine the maximum similarity score of two sequences the *Score* of all pairs of sequence elements has to be determined.

We prove the statement by reductio ad absurdum. Let us assume that there is an algorithm that can determine the $MaxSimScore$ without considering alignments of all elements. Furthermore let A_i and B_j be the pair of elements that the algorithm does not consider determining the $MaxSimScore$ of A and B. Since

the score of two sequence elements is in general randomly distributed, we can arbitrarily assume that $Score(A_i, B_j) > \sum_{k=1}^{m} \sum_{l=1}^{n} Score(A_k, B_l) - Score(A_i, B_j)$ (i.e. $Score(A_i, B_j)$ is greater than the total sum of scores of all other pairs). Thus an alignment including (A_i, B_j) yields a greater similarity score than the one found by the algorithm (since any alignment that does not contain (A_i, B_j) consists of a subset of other pairs), which contradicts our assumption.

Among all algorithms considering all pairs of sequence elements our algorithm requires the minimum number of operations because every pair is regarded only once and a constant number of operations c_1 is performed (cf. *for* loop of the algorithm). The determination of $MaxSimScore$ requires therefore $m * n * c_1$ calculation steps. The reconstruction of alignment (*do* loop) takes in worst case $(m + n) * c_2$ operations resulting in a total time complexity of the algorithm

$$T(m, n) = m * n * c_1 + (m + n) * c_2 \in O(m * n)$$

Since any algorithm calculating $MaxSimScore$ has to consider all pairs of sequence elements, the presented algorithm solves the task of sequence alignment with maximum similarity in the minimum possible time.

Approximating the number of tokens by the number of sequence elements the time required for the complete comparison of two hierarchies would lie in $O(2^{m+n} * (m + n))$ (refer to sec. 7.2.1). The huge difference between the asymptotic runtimes emphasizes the efficiency and importance of converting hierarchical structures to sequences for similarity assessment and generalization of extraction rules.

7.2.3 Rule Similarity Measure

Merging rules is reasonable if they extract at least one common attribute. For many attributes the structure and properties of their values play a crucial role for their identification. Besides, in many domains values of certain attributes tend to co-occur in a sentence due to the strong semantic connection (e.g. in a sentence describing a terrorist act the actual action performed by terrorists is usually mentioned with the perpetrator and the victims). Therefore the co-occurrence and the order of co-occurrence of attribute values is a very important factor for the identification of information.

Let $ExtSim$ be the similarity measure for extractions and their co-occurrence in a rule. Suppose the rule r_1 extracts the values of the attributes $A_1 \dots A_m$ and r_2 – $B_1 \dots B_n$ in the presented order. The attribute sequences are aligned so that the maximum number of corresponding attributes are found. Let L be the alignment of attributes, $E_{1_1} \dots E_{m_1}$ and $E_{1_2} \dots E_{n_2}$ – the patterns in r_1 and r_2 encoding the corresponding attribute value. The extraction similarity is determined as:

$$ExtSim(r_1, r_2) = \sum_{(A_i, B_j) \in L} Score(E_{i_1}, E_{j_2}) * 2^{|L|}$$

The similarity scores of aligned patterns that encode corresponding attribute values are summed and weighted by the power of two corresponding to the number of aligned patterns $|L|$. The co-occurrence and the order of attribute values are both reflected by the alignment L; since they have a significant influence on the similarity of rules, they are incorporated as a strong weighting factor.

On the other hand, many, especially complex, attributes cannot be identified without their context. The context similarity $ContextSim$ is calculated adding

up the similarity scores of context fragments around the extracted values. Suppose $r_1 = C_{1_1}E_{1_1}C_{2_1}E_{2_1}\ldots E_{m_1}C_{m+1_1}$, $r_2 = C_{1_2}E_{1_2}\ldots E_{n_2}C_{n+1_2}$ where C_{i_1} is the context fragment between the $i-1^{\text{th}}$and i^{th} extraction of the rule r_1.

$$ContextSim(r_1,r_2)=\sum_{(E_{i_1},E_{j_2})\in L}(MaxSimScore(C_{i_1},C_{j_2})+MaxSimScore(C_{i+1_1},C_{j+1_2}))-DuplicateSum$$

where $DuplicateSum$ is the sum of $MaxSimScore(C_{i_1}, C_{j_2})$ values added twice.

For the calculation of context similarity all hierarchical patterns are flattened to their sequential representations and the $MaxSimScore$ algorithm (refer to fig. 7.5) is applied.

The general rule similarity involves therefore the similarity of extracted fragments and their co-occurrence and the context similarity. To establish a single similarity measure these different similarity metrics have to be combined:
Let $R = r_1, \ldots, r_n$ be a set of rules. Both similarity measures $ExtSim$ and $ContextSim$ can be incorporated in a single metric in the similar manner as we did it for the *FirstCompM* metric in sec. 5.2.3 adding the contributions of single metrics relative to their maximum value:

$$RuleSim(r_i,r_j)=\frac{ExtSim(r_i,r_j)}{max(ExtSim(r_1,r_2),\ldots,ExtSim(r_{n-1},r_n)}+\frac{ContextSim(r_i,r_j)}{max(ContextSim(r_1,r_2),\ldots,ContextSim(r_{n-1},r_n)} \tag{7.2}$$

After determining the extraction and context similarity the overall rule similarity can be calculated according to 7.2.