Mapping Dependency Structures to Phrase Structures and the Automatic Acquisition of Mapping Rules

Bernd Bohnet* **& Halyna Seniv****

*Institute for Intelligent Systems – University of Stuttgart
Universitätsstr. 38, 70569 Stuttgart, Germany
bohnet@iis.uni-stuttgart.de

**Institute of Natural Language Processing – University of Stuttgart
Azenbergstraße 12, 70174 Stuttgart, Germany
senivha@ims.uni-stuttgart.de

Abstract

This paper describes a simple graph grammar based formalism that is capable to translate dependency structures into phrase structures. A procedure is introduced for the automatic acquisition of mapping rules from corpora which are annotated with both phrase structures and dependency structures. The acquired rules are evaluated by applying them to a corpus annotated with dependency structures.

1. Introduction

Recently, it has been reported that German corpora annotated with phrase structures have been converted into corpora annotated with dependency structures, cf. (Bohnet, 2003), (Forst, 2003). Lin (1995) has published such a method for English phrase structure corpora. Collins et al. (1999) as well as Xia and Palmer (2001) have proposed an inverse method for converting dependency structures to phrase structures. In the case of German, no such method has been adapted so far, due to the fact that the word order can not be easily determined from a dependency tree. But in the case of corpora annotated with dependency structures, the information about the word order is already available in the word sequence of the sentences. However, when the word order is not available, it is possible to retrieve unordered phrase structures. This unordered phrase structures are useful structures, e.g. they can be used in the linearization process of free word order languages. They can serve as so called word order domains or precedence units, cf. (Bröker, 1998), (Gerdes and Kahane, 2001), (Bohnet, 2004).

In order to translate dependency structures into phrase structures, we use a graph grammar, cf. (Rozenberg, 1997). Graph grammars consists of declarative rules which describe how a source graph is mapped onto a target graph. We reused the graph grammar compiler described in Bohnet and Wanner (2001) to translate the phrase structure annotation of the NEGRA corpus into dependency structures, cf. (Bohnet, 2003). For the learning procedure of the mapping rules we need a corpus annotated with phrase structures and dependency structures. Therefore, we also reused results of that phrase structure to dependency structure translation. Additionally, the procedure was applied to the recently available TIGER corpus. The method was easily extendable, since the annotation of the TIGER corpus is mostly similar to that of the NEGRA corpus.

In this paper the adopted dependency sturctures and phrase structures are described. Then, the basic notion of the graph grammar formalism is given. After that we have a look at an sample grammar and explain the automatic acquistion of the mapping rules and finally, the results are evaluated.

2. Dependency and Phrase Structures

The nodes of the dependency structures are labelled with the basic word form and the edges with syntactic relations (e.g. *subjective, dobjective, iobjective,* etc.), cf. (Mel'čuk, 1988). Morphosyntactic features (e.g. *number, tense,* etc.) are represented as attributes, which are attached to the nodes. Unlike phrase structures, dependency structures do not store word order information. The dependency structures annotation is retrieved by structure translation from the phrase structure annotation which is described in detail by Bohnet (2003). An example of a dependency structure is shown in Figure 1.

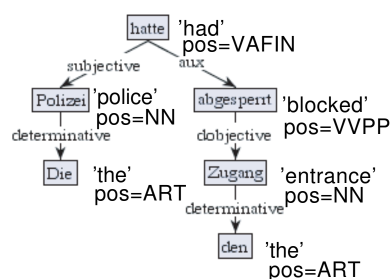

Figure 1: A dependency structure. The nodes are labelled for better readability with already inflected forms.

For the rule acquisition, we use the phrase structure annotation of the German NEGRA and TIGER Corpus which contain about 20000 and 40000 sentences of German newspapers, respectively. The phrase structures used in the corpora are supposed to be theory independent and hence, they are rather flat. The words of the corpora are annotated with syntactic categories (e.g. NN 'normal noun', KON 'coordinated conjunction', etc.), and the grammatical functions (e.g. SB 'subject' , OA 'accusative object', etc.) which are represented as attributes. An example of a phrase structure is shown in Figure 2.

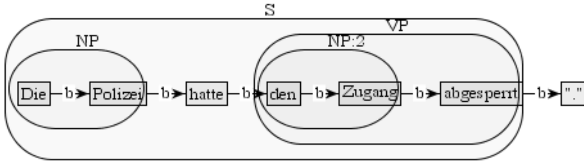

Figure 2: Phrase structure as hierarchical graph.

3. Graph Grammar Formalism

In the mapping procedure the compiler applies grammar rules to the input graphs. A graph rule consists of a left-hand side graph (LS), a right-hand side graph (RS), a set of conditions and a set of correspondences which specifies the gluing of the result, i.e. the right-hand side graph fragments. If a fragment of the source graph matches the left-hand side of a rule, and the condition specified, the fragment is mapped onto the subgraph specified at the right-hand side. Additionally, rules can contain context information in both sides. Left-hand side context means that a rule can share an input fragment with other rules; right-hand side context is a target fragment that must have already been created by other rules. In order to represent dependency trees and phrase structures in one graph formalism, we use hierarchical graphs, cf. (Busatto, 2002). They allow to define a hierarchy of packages on top of a directed labelled attributed graph. The packages can contain other packages and nodes.

4. Sample Grammar

We identified two main rule groups. *Build rules* building phrases and *adjoin rules* inserting phrases into phrases. The rule groups are applied in two phases. In the first step the build rules are applied and in further steps the adjoin rules, because they need the result of the build rules or other adjoin rules.

Two typical build rules are shown in Figure 3 and in Figure 5. The left-hand side of each rule consist of two nodes labelled with the variable names ?Xs and ?Ys which are connected by an edge. The edges of the rules are labelled with aux, subjective and determinative. Part-of-Speech tags (pos) are located beneath the nodes which are conditions. On the right-hand side of each rule is a package that is created, if the rule is applicable. Each package contains two nodes and it is labelled with a name or possible names separated by vertical strokes representing or. Each package has an attribute func that encodes the possible grammatical functions. The exact functions are selected by the adjoin rules. Both node names and attribute values can represent a set of possible values. They are interpreted as constrains which restrict the applicable adjoin rules and the unification of intersecting packages.

In the first step, only the build rules are applicable to the source graph. When they are applied to the dependency tree of Figure 1 the intermediate graph shown in Figure 4 is created. Two of the applicable rules are shown in Figure 3 and one on the left side in Figure 5. The rule on the left side of Figure 3 matches as following: the node ?Xs to hatte 'had', and the node ?Ys to abgesperrt 'blocked'. The rule on the right side matches as following: the node ?Xs to hatte, and the node ?Ys to Polizei 'police'.

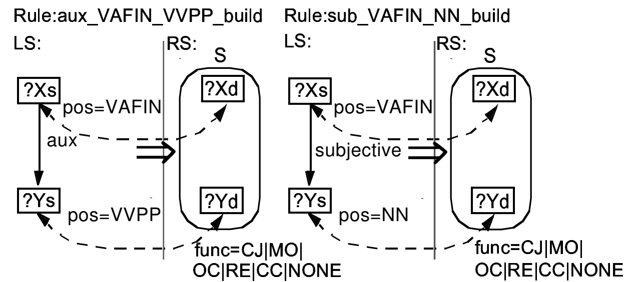

Figure 3: Two build rules.

They build together the phrase that is labelled with S (cf. Figure 4), i.e. each of the rules builds a package. The first one is S[hatte abgesperrt] and the second S[hatte Polizei]. The two packages are unified to one package S[hatte Polizei abgesperrt] because the packages intersect, the labels (S) are equal and the attribute(s) (func) are compatible.

The rule shown on the left side in Figure 5 is applicable twice, once to the node Polizei 'police', the edge determinative and the node Die 'the'. And once to the node Zugang 'entrance', the edge determinative and the node dem 'the'. The first instance of the rule creates the package shown in Figure 4 on the left side which is labelled with NP | PP and the second instance creates the package on the right side with same label. A rule which is not printed creates the package that is labelled with VP.

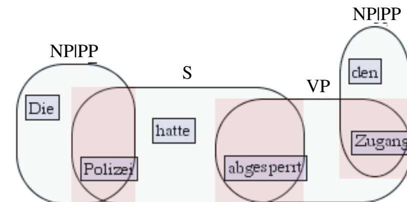

Figure 4: Result of the first step, intermediate graph.

In the next phase the adjoin rules are applied. On the right side in Figure 5 an adjoin rule is shown. It is applicable to the node hatte 'had', the edge subjective and the node Polizei 'police'. And further in the intermediate graph that is the resulting structure of the first step (cf. Figure 4) the dashed package (cf. Figure 5) labelled with NP | CNP matches to the package on the left that is labelled with NP | PP. The rule restricts the package further to NP, because the package is constrained to NP or PP and the rule is only applicable to NP's or CNP's, therefore the rule have to be an NP.

The instance of the rule creates a package labelled with S that contains the NP package and the node hatte. This package intersects with the package that is labelled with S and is shown in the intermediate graph. Therefore, the two packages are unified to one package S[NP[Die Polizei] hatte ...].

Other adjoin rules which are not printed pack the NP | PP on the right hand side in Figure 4 into the VP package and again the latter package into the S package. The

result is the unordered phrase structure that is shown in Figure 6.

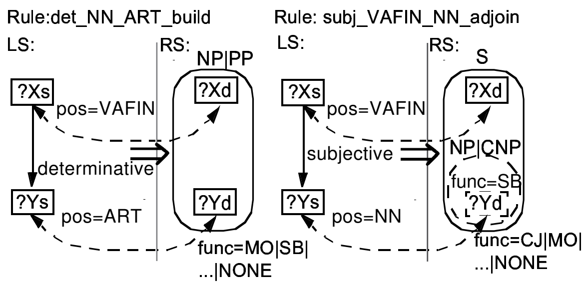

Figure 5: A build rule is shown on the left side and an adjoin rule on the right side.

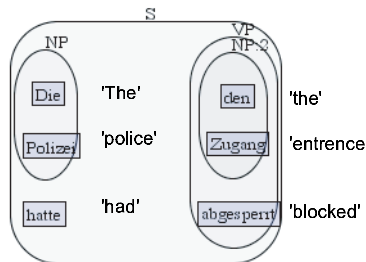

Figure 6: The result of the next steps, an unordered phrase structure.

5. Acquisition of the Graph Grammar Rules

The learning procedure of the rules is straight forward. We use a corpus annotated with dependency structures and phrase structures. Both are represented as hierarchical graphs. As input of the learning procedure we take each sentence and perform the following two steps: For the build rules we take each edge of the dependency tree and we will look up, if the start and end node of the edge are in the same phrase of the phrase structure. If they are in the same then we create a build rule. Otherwise we create an adjoin rule. After that similar rules are combined. The creation of build and adjoin rules is described below.

Creation of build rules. The left-hand side of a build rule consists of an edge which is similar to the one of the dependency tree. The node labels are replaced by variables, i.e. the label of the source node by `?Xs` and the label of the target node by `?Ys`. The attributes like `pos` are kept as conditions.

After that, we create the right-hand side of the rule. It consists of an package labelled with the name of the phrase and we keep the attributes. This package contains two variable nodes which have attached the attributes of the nodes from the phrase structure. Finally, the correspondences are added between the appropriate nodes of the left-hand side and right-hand side, cf. for instance Figure 3.

At this stage, we search in the list of already created build rules, and if there is a rule with the above left-hand side, then we merge both rules, i.e. the package label and attributes values of the right-hand side are associated with `or` ('|'). Otherwise we add the new rule to the list of rules.

Creation of adjoin rules. The creation of the left-hand side is equal to that of the build rules (see above).

In the case of the right-hand side, we identified four different relations of the position from a governor and its dependent[1] in a phrase structure:

1. In the phrase of the governor is a subphrase containing the dependent, cf. Figure 7 (left). This is the most frequent case.
2. In the phrase of the dependent is a subphrase containing the govenor, cf. Figure 7 (right). This is the inverse case to 1.
3. Both the dependent and the governor are contained in different subphrases of one phrase, cf. Figure 8 (left).
4. The dependent is in a subsubphrase of a phrase in which the governor is contained, cf. Figure 8 (right).

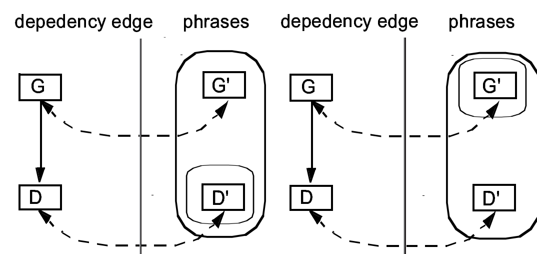

Figure 7: Relation of the position from a governor and its dependent in a phrase structure.

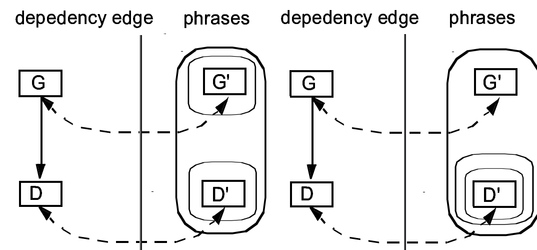

Figure 8: Relation of the position from a governor and its dependent in a phrase structure.

The creation of the right-hand side proceeds as follows: The appropriate case among the cases above is choosen. The inner package will be marked as right-hand side context which has to be already created and it is matched in the target structure during the application step. The packages are labelled with the names of the phrases and the attributes are kept. The package contains two variable nodes with the attached attributes and finally, the correspondences are added.

At this stage, we search in the list of already created adjoin rules, and if there is a rule with the above left-hand side, then we merge both rules. Otherwise we add the rule to the rule set.

Summarisation of rules. The user can choose among three options: do not summarise, summarise when the right-hand side context is equal (=), or summarise when the right-hand side context is a subset or equal ($\subseteq$). Of course, it is only possible to summarise rules of the same type (cf. Figure 7 and Figure 8).

[1]The governor is the mother node and the dependent is the child node of an edge from a dependency tree.

6. Evaluation

The evaluation of the learning procedure consists of four steps. At first, we execute the learning procedure to a set of training examples. After that, we summarise the rules. Then, we apply the learned rules to a set of dependency structures, and finally, we compare automatically the resulting phrase structure annotation to the original phrase structure annotation of the sentences.

We learned from a training sets of 2500, 5000, 7500 and 10k sentences, and as result, we received four rule sets. Then each of the sets are applied onto two sets of dependency structures. The first set is selected randomly among structures of the training set and the second set randomly from the remaining unkown structures. Afterwards, we repeated the steps with summarised rule sets.

The results of the summarisation for 10k annotated sentences are as following: After the learning step, we received 1860 rules, and when we summarise the rules with the equal option (=) the number of rules decreased to 1443 (reduced by 22%) and with the subset or equal option (⊆) to 771 (reduced by 58%). But also the number of correct results decreased.

The results of the evaluation are summarised in Table 1. The first column contains the corpus name, the second column the number of sentences of the training set, the third column the number of sentences of the test set, the fourth column specifies whether the test set is contained in the training set or not, the fivth whether the rules are generalized or not, and finally the last column gives the precentage of correct results.

corpus (name)	training set (# sent.)	test set (# sent.)	known (Yes/No)	summarized (NO/=/⊆)	correct (%)
NEGRA	2500	1000	Yes	No	95
NEGRA	5000	1000	Yes	No	95
NEGRA	7500	1000	Yes	No	95
NEGRA	10000	1000	Yes	No	95
NEGRA	2500	1000	No	No	79
NEGRA	5000	1000	No	No	85
NEGRA	7500	1000	No	No	87
NEGRA	10000	1000	No	No	**88**
NEGRA	10000	1000	Yes	=	89
NEGRA	2500	1000	No	=	76
NEGRA	5000	1000	No	=	80
NEGRA	7500	1000	No	=	84
NEGRA	10000	1000	No	=	86
NEGRA	7500	1000	No	⊆	68
NEGRA	10000	1000	No	⊆	70
TIGER	7500	1000	Yes	No	96
TIGER	7500	1000	No	No	91

Table 1: Summarisation of the results.

Translation errors have the following impact:

1. Phrases are not packed. This is the most frequent case.
2. Phrases are packed into wrong phrases.
3. Phrases are packed recursively.
4. Phrases are wrong unified, e.g. two NPs with the same grammatical function are encapsulated in each other. This is the most seldom case.

The reason for error No. 1 is that the rules are not learned since the case was not in the training set or it seems that they are not learned as we avoid the execution of contradictory rules. It is possible to solve that problem by packing the none packed phrases into already packed phrases intersecting with them. Thus, we would get the correct structure, but not always a unique phrase label. However, it would reduce the errors of the structure by about 75%. The reason for error No. 2 and No. 3 are annotation errors in the dependency structures or in the phrase structures.

7. Conclusion and Further Work

We presented a declarative formalism to map dependency trees to phrase structures, and a procedure to learn mapping rules from a corpus annotated with both dependency structures and phrase structures. We showed that it is possible to build automatically a grammar with a high coverage of 88% for unkown sentences. Hence, the formalism and grammar interpreter can be used to translate both dependency structures into phrase structures as shown in this paper and to translate phrase structures into dependency structures as shown in Bohnet (2003).

We plan to extend the procedure to corpora annotated with topological information, i.e. we want to apply it onto the German Topological Field Model. Furthermore, we intend a generalization mechanism in order to be able to learn from smaller training sets.

8. References

B. Bohnet and L. Wanner. 2001. On Using a Parallel Graph Rewriting Formalism in Generation. In *Eight European Workshop on Natural Language Generation*, Toulouse.

B. Bohnet. 2003. Mapping Phrase Structures to Dependency Structures in the Case of Free Word Order Languages. In *First International Conference on Meaning-Text Theory*, Paris.

B. Bohnet. 2004. A Graph Grammar Approach to Map between Dependency Trees and Topological Models. In *First International Joint Conference for Natural Language Processing*, Hainan.

N. Bröker. 1998. Separating Surface Order and Syntactic Relations in a Dependency Grammar. In *COLING-ACL 98*.

G. Busatto. 2002. *An Abstract Model of Hierarchical Graphs and Hierarchical Graph Transformation*. Ph.D. thesis, Universität Paderborn.

M. Collins, J. Hajič, L. Ramshaw, and C. Tillmann. 1999. A Statistical Parser for Czech. In *Proceedings of the ACL*.

M. Forst. 2003. Treebank Conversion - Establishing a test-suite for a broad-coverage LFG from the the TIGER treebank. In *Proceedings of the EACL Workshop on Linguistically Interpreted Corpora*, Budapest.

K. Gerdes and S. Kahane. 2001. Word order in german: A formal dependency grammar using a topological hierarchy. In *Proceedings of the ACL*.

D. Lin. 1995. A dependency-based method for evaluating broad-coverage parsers. In *IJCAI*, pages 1420–1427.

I.A. Mel'čuk. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press, Albany.

G. Rozenberg, editor. 1997. *Handbook of Graph Grammars and Computing by Graph Transformation*. World Scientific, Singapore, New Jersey, London, Hong Kong.

F. Xia and M. Palmer. 2001. Converting Dependency Structures to Phrase Structures. In *The Proc. of the Human Language Technology Conference*, San Diego.