

Covering the Mapping from Semantic up to Morphology by Transducers

Bernd Bohnet

Innovative Sprachtechnologie – TTI GmbH
and
Visualization and Interactive Systems Group
Universitätstraße 38, D-70569 Stuttgart
bernd.bohnet@informatik.uni-stuttgart.de

Abstract

The Meaning-Text Theory (MTT) describes a rich set of linguistic representations and their mapping. Therefore, the MTT consists of a large range of structure types like strings, trees and graphs. For the mapping of these structure types and linguistic representations, transducers seems to be the appropriate tool. However, there is no tool which covers all the levels of the MTT. Building mapping tools on common principles and covering in a few formalism all possible transitions between the levels has a lot of advantages. This paper investigates such common principles and algorithms which can be the basis for mapping tools.

Keywords

Meaning-Text Theory, text generation, parsing, morphology, FST, tree and graph transducer

1 Introduction

For the mapping between all of these structure types and linguistic representations of the Meaning-Text Theory, transducers seems to be the appropriate tool, cf. (Mel'cuk, 1988), (Kahane, 2000). However, there is no tool which covers all of the needed mappings in one formalism. For instance, the systems AlethGen (Coch, 1996), RealPro (Lavoie and Rambow, 1997), MATE (Bohnet and Wanner, 2001; Bohnet, 2006) and DepLin (Gerdes, 2002), Polarized Unification Grammar (Kahane and Lareau, 2005), cover the mapping from trees/graphs to strings mainly in the generation direction. They do not cover morphology or/and they are not intended for parsing.

Transducers are standard tools in computational linguistic. Finite state transducers (FST) have become as two level morphology (Koskenniemi, 1983) the most favored tools for word form recognition and word form production because of their advantageous properties (Antworth, 1990), (Beesley and Karttunen, 2003). This properties distinguishes them from the generative rewriting approach. Generative rewriting rules work with only one tape from which they read

and onto that they write symbols. While generative rules overwrite or transform symbols, transductive rules work with two tapes, one for reading and one for writing. This is the reason, why they are called two level rules. Generative rules are applied sequentially and because of that, they have to be ordered. This means that each rule creates its intermediate level which serves as input to the next rule. Whereas two level rules can be applied in parallel and need only two levels of representation. Since the input of transductive rules stay stable the rules create a static relationship between the symbols of the two levels. Transductive rules have access to both levels and their application is determined by specifying their context and not by ordering. Finally, transductive rules are bidirectional.

In order to process more complex structured linguistic representations like syntactic structures, tree transducers are used. We count to the group of tree transducers string to tree, tree to string and tree to tree transducers. Top down tree transducers have been independently introduced by Rounds (1970) and Thatcher (1970) as extensions of finite state transducers. Tree transducers traverse the input trees from the root to the leafs. Therefore, they are often called **Root-to-frontier** transducer (short **R**-transducer). Rules are applied in parallel to the branches and they rewrite the tree top down. There are many extensions and types of tree transducers like **R**-transducers with finite look ahead (context) or regular-look ahead, **Frontier-to-root** transducers which process a tree bottom up, etc. A more complete typology of tree transducers is in Knight and Graehl (2005). Tree transducers do not share most of the advantages of the above mentioned two level rules. They do not use two tapes and rewrite the tree, therefore, they create intermediate levels of representation. The rules have to be ordered and are not bidirectional.

Graph transducers can be used, for instance to process semantic representations and conceptual graphs. To the group of graph transducer, we count tree to graph, graph to tree and graph to graph transducer. However, graph transducers are seldom used which has several reasons. Semantic representations and conceptual graphs are not yet so often used. The problem of graph matching, known as sub graph isomorphism problem, is in the class of NP-complete problems. This is the reason, why it seems computational not attractive.

In order to identify common principles and algorithms, we investigate the following questions. (1) How can tree and graph transducers use two Tapes like finite state transducers? (2) What is the meaning of the two level operators for tree and graph transducers? (3) How to implement efficient graph matching and how to implement strings, trees and graph transducers within one matching procedure? (4) FST have regular expressions as context. How can this be integrated in a common matching algorithm.

2 Two Tapes for String, Tree and Graph Transducer

One of the most important principle of finite state transducers is that they use two tapes. FST read from one tape and write onto the other one while deletion and rewriting (overwriting) on both tapes is forbidden. This principle is the source of most of the advantages: context is possible on both tapes, transducers are bidirectional, no ordering of rules is needed and a static relationship between symbols of the tapes is introduced. How to realize this principle for tree and graph transducers? To answer this question, we review first the structure of string transducers and then the structure of tree / graph transducers.

The two level morphology formalism defines a lexical representation (LR) and the corresponding surface representation (SR).

LR: g e h e n
 SR: g e h t 0

A rule consists of three parts; the correspondence, the rule operator and the environment or context. The correspondence specifies a lexical symbol that corresponds to a surface symbol. The operator specifies the relationship between the correspondence and the context. The context specifies the environment in which the correspondence is found. In the following rule, the lexical e corresponds to the surface t only when it follows a consonant (C) and any symbol (@) on the surface.

e:t => C:@ __

A standard **tree transducer** rule as shown in Figure 1 consists of a left-hand side and a right-hand side. The left-hand side is searched in the source tree. The search starts at a state symbol (cf. Figure 1: q, r). If this is successful then the found part is replaced by the right-hand side. The left-hand side of the rule `det_v` matches the left branch of the tree in Figure 2. If the rule is applied then the matched part is replaced by the right hand side and the intermediate graph in the middle is created. In the next step, the left-hand side of the second rule `mod_v` is matched and the branch on the right is replaced.

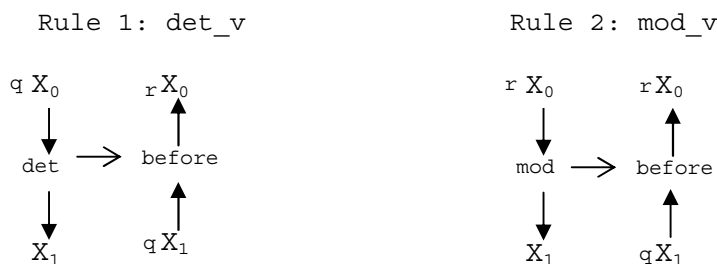


Figure 1: Two tree transducer rules

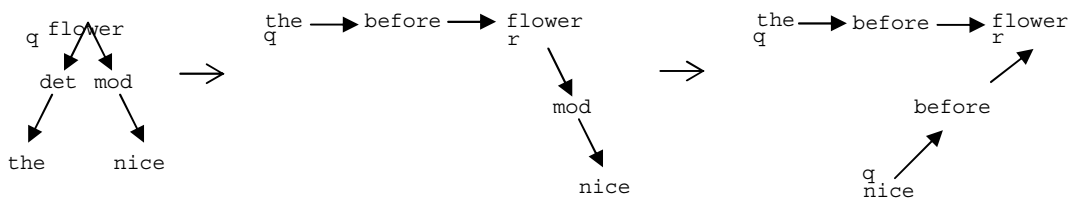


Figure 2: The application of the rules of Figure 1

It is obvious that FST and tree transducers are different in several aspects. The corresponding symbols of string transducers are before the operator ($=>$) and the corresponding symbols of the tree transducers are separated between the left-hand side and right-hand side. In the case of string transducers, the operator defines the relationship between the correspondence and the context and in the case of tree transducers, it represents just an implication. FST have symbols as their characters and tree and graph transducer consists of arrows and symbols which are frequently strings.

However, if we rotate a string transducer rule by 90° , we obtain the symbols of the lexical representation on left-hand side and the symbols of the surface representation on the right-hand side. The following Figure shows on the left, a rotated rule and on the right, the rule in the style of tree transducers. Additionally, we kept the correspondence links which are drawn

by dashed lines and we marked the context by brackets. The operator of the two level morphology rules can not be translated directly to the formalism of tree transducers.



In the following, we consider tree/graph transducers which work on two tapes. If the right-hand side of each of the rules would be created without replacing the part in the source graph and creating the parts in the result tree then the rules behave as they read from one tape and write to a second tap. However, the result parts of the applied rules are not yet connected. In order to connect the graph parts, the correspondence links, like in the case of FST, can be introduced. Using this technique they can either (1) grasp in the target graph part and attach to this part a new part or (2) they indicate which parts should be glued¹ together.

The first approach is useful when rules are applied in a sequence. In this way, already created parts, can be accessed. This is the case with FST which are applied left to right and with tree transducers which are applied top down. The sequence comes from the processing technique as used nowadays and is not determined by linguistic necessities. We will come back to this point later and show that the rules can be applied simultaneously on the string and tree transducer level having a sequence only justified by linguistic properties. The second approach is useful when the rules are applied simultaneously as then the right hand sides are created at same time and the parts can already be glued together in the same step.

(1) Figure 3 illustrates the access to the right-hand side by a two tape transducer. The signs of the word ‘gehen’ are represented as a tree with a tree width of one. A part of the tree is shown in the following Figure. The upper part has been already processed and is therefore not shown. The h of the source tree is connected by a dashed line with an h of the result which has so far been build by other rules. The rule above is shown again in the middle of the figure. The arrows out-going from the rule to the source and to the result tree show which part is matched by the rule. The arrows to the right show which part is build. The h in the result (left) is matched as context by the @ symbol. When the rule-interpreter applies the rule then it builds from the context (h), an out-going arrow ending at new created part (t).

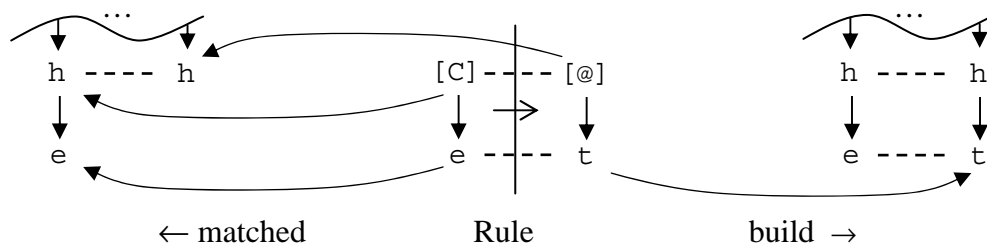


Figure 3: Tree Transducers with two tapes

(2) Figure 5 illustrates the gluing approach of a two tape transducer. On the left, a source tree is shown. The rules of Figure 4 are applied to the source tree. During the application step, the parts specified on the right-hand side of the rules are independently created. Independently means that the rules do not need results computed by other rules. Therefore, the left-hand side

¹ The term glue comes from graph grammar terminology and the process as similar or equal to unification.

of these rules can be matched and the right-hand side can be build in parallel. After this, the nodes which have correspondence links (drawn as dashed lines) to the same node in the source are glued together. The result of this procedure is the graph on the right.

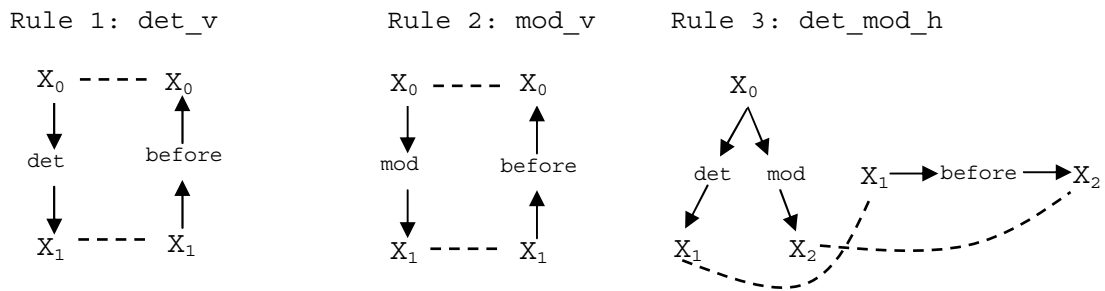


Figure 4: Tree transducer rules with correspondence links

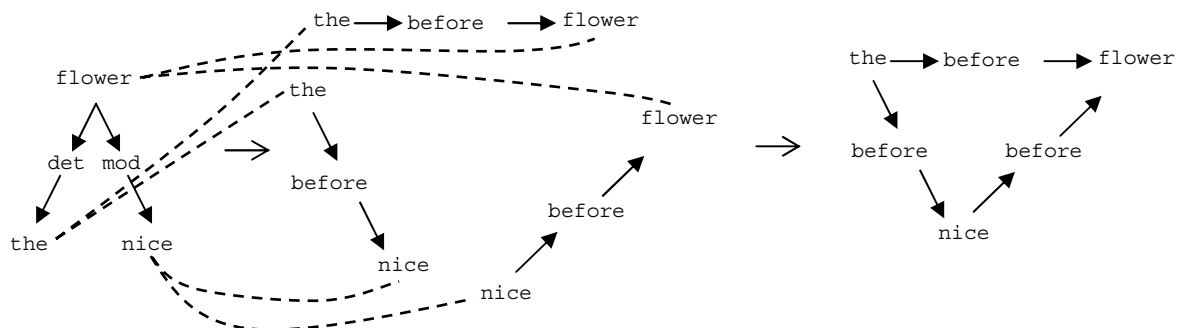


Figure 5: Tree/Graph transducer rules using correspondence links

Additional rules are needed to order the complete dependency tree. The rules are bidirectional and could also be applied from right to left. This means matching the right hand side and creating the left-hand side. For the rules of the examples, this is from a linguistic point of view not reasonable since each of the rules would be applied to many times. The rules would need at least information about the part of speech on the right-hand side. A description about, how such a formalism can be used to determine word order and how word order rules can be learned from a corpus is described by Bohnet (2004).

This section described the mapping of the correspondence and context parts of two level morphology rules to the formalism of tree and graph transducers and we investigated, how the advantageous properties of string transducers can be gained for tree and graph transducers by introducing correspondence links and two tapes. The meaning of two the level operators for tree and graph transducers is still open. This topic is addressed by the next section.

3 The Meaning of Two Level Operators for Tree and Graph Transducers

The two level morphology rules have four operators: \Rightarrow , \Leftarrow , \Leftrightarrow , and \nrightarrow . The operator of a rule specifies the relationship between the correspondence and the context in which the correspondence occurs: (1) the left arrow \Rightarrow means the correspondence only occurs in the context (but there might be other correspondences with this context), (2) the right arrow \Leftarrow means the correspondence always occurs in the context (but there might be other contexts where this correspondences occurs), (3) the correspondence symbol \Leftrightarrow means the correspondence always and only occurs in the environment, and (4) the 'not' right arrow \nrightarrow

means the correspondence never occurs in the context. The operator \Rightarrow allows concurrent rules. The application of concurrent rules lead to alternative results.

$$R2 \quad C:C \Rightarrow [C:C _ \mid _ _] \qquad R3 \quad t:c \Rightarrow C:C _$$

The rule R2 maps each consonant which has left of it a consonant or each consonant which is at the beginning of a word to the same consonant. R3 maps each t to a c after a consonant. For instance, the rules map gtc to gtc and alternatively to gcc . Tree and graph transducers can obtain the same effect by ‘locking’. A rule can lock each symbol or arc in the left-hand side and right-hand side. If parts which are locked overlap, then the rules are concurrent and alternative results are created.

One of the most important properties of the two level morphology is that it is failure-driven. It defines what is forbidden. The operator \Rightarrow forbid the same correspondence in another context. The operator \Leftarrow forbids in a given context other correspondences. The operator $\Leftarrow\Rightarrow$ forbids in a given context other correspondences and for correspondences it forbids other contexts. Finally, the operator $/\Leftarrow$ explicit forbids a correspondence in a given context. In comparison to tree and graph transducers, it is unusual to forbid or check equal correspondences in relation to context.

Since the standard case is that a sign is mapped without any change, it is natural and very useful to define or forbid the exception from the standard. However, if the mapping is between formal languages which are more different then the mapping formalism might concentrate more on positive definition, like how to map parts between the levels as in the case of structure transducer rules. This rules have the possibility to restrict the application of single rules. The context of the rules can contain negated parts and regular graph expressions. The left-hand side or right-hand side of rules states for all parts (g_a) in the source (S) and for all context (e_a) in the source or in the target (S, T) and if there exists the parts (g_e) in the source and the context parts (e_e) in the source or target (S, T) then the right hand side or left hand side is build depending on the execution direction. The side which is matched looks formally like the following expression where g_a and g_e are graphs (correspondence graphs), e_a and e_e are context or environment graphs, S is the source graph and T is the target graph.

$$\forall g_a, e_a : (g_a \subseteq S \ \& \ e_a \subseteq S, T) \ \& \ \exists g_e, e_e : (g_e \subseteq S \ \& \ e_e \subseteq S, T)$$

Tree and graph transducers can describe mappings in equally powerful manner. In contrast to FST, the mapping is defined by explicit positive expressions and the grammar writer has to ensure that rules define correct mappings. Structure transducer rules can not restrict other rules and therefore, single rules have to be restricted so that they perform only correct mappings. The question remains also even when it is evident that tree and graph can describe mappings equal powerful, should structure transducers have the same operators? An answer to this question can give a closer look onto the way structure transducer work and if it is possible to integrate efficient operators like the restriction ($/\Leftarrow$). Another essential issue is that string transducers can actively reject input. Tree transducers might process an input tree not completely that could be interpreted as rejected and also graph transducers can have the same criterion for reject that the graph is not completely covered by the left hand side of applicable mapping rules.

4 The Sub-Graph Isomorphism Problem

Graph transducers can be defined in the same way as tree transducers except that the left-hand side and right-hand side are graphs. The use of graphs seems to be a disadvantage as the problem of sub graph isomorphism is known as NP-complete problem. However, the problem can be solved in polynomial time for many restricted classes like two-connected outer-planar graphs, bounded tree-width graphs, etc. On the other hand, semantic representations have properties from which a matching algorithm can profit very much. Semantic representations are connected graphs as their content has to be linked to make sense. The edges and nodes of semantic representations are labeled. The number of out-going edges is restricted as a predicate has usually not more than about six arguments.

We describe now a graph matching algorithm which is based on the property that semantic graphs are connected and which takes advantage of the labels. The used method is called dynamic programming. The solutions for a problem is build searching optimal sub-solutions. In the bottom-up version of this approach, all sub-problems that might be needed are solved, stored and then used to solve larger problems. Input to the algorithm is the source graph and the rule graph. The goal is to find all occurrences of the rule graph in the source graph. In the following sketch of the matching algorithm, we consider a simple graph type $(G=\langle N,E,L,\Sigma \rangle)$ which consists only of nodes (N), edges ($E \subseteq N \times N$) and node labels ($L \subseteq \Sigma \times N$).

1. The algorithm starts by a node of a rule and compares the label of the node with all of the node labels in the source graph.
2. For all matches, the algorithm continues matching with the in-going and out-going edges of the rule to the source graph. Step 2 is repeated recursively until the complete rule graph is found or the matching can not be continued.

In Figure 6, we illustrate the matching procedure. The rule graph (G_R), which is shown on the left-hand side, is searched in the source graph (G_S). In the first step (1), the node labeled with an **a** is found. This is indicated by bold face. In the second step (2), the in-going edge from **R** to **a** is found, then (3) the edge from **a** to **1** and finally (4) the edge from **1** to **b**. It is easy to imagine and so to understand, how the matching algorithms works. It begins from a starting point, moves along the edges until the searched sub graph is found. In practice, the complexity of the algorithm is linearly related to the number of rules and input nodes, given as $O(n^2)$. The rules can be compiled in a rule hierarchy similar to the RETE-algorithm (Forgy, 1982) then the complexity is $O(n \log n)$.

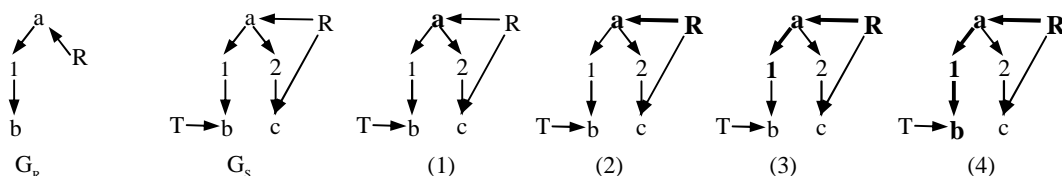


Figure 6: Illustration of the matching algorithm

Semantic representations of the MTT are more complex than the above graphs. An example is shown left in Figure 7. The semantic representation represents sentences like *The high ozone concentration cause the bad air quality*. Semantic representations have labeled edges, attributes like main which is represented as underlined word in Figure 7, nodes which contain other nodes like **Rheme** and **Theme**. But they can be easily compiled into simpler graphs

with typed nodes. The types are *relation* for edge labels, *attribute* for features, and *package* for nodes which contain other nodes. The graph on the right in Figure 7 shows the simplified graph which is a graph of the above defined graph type.

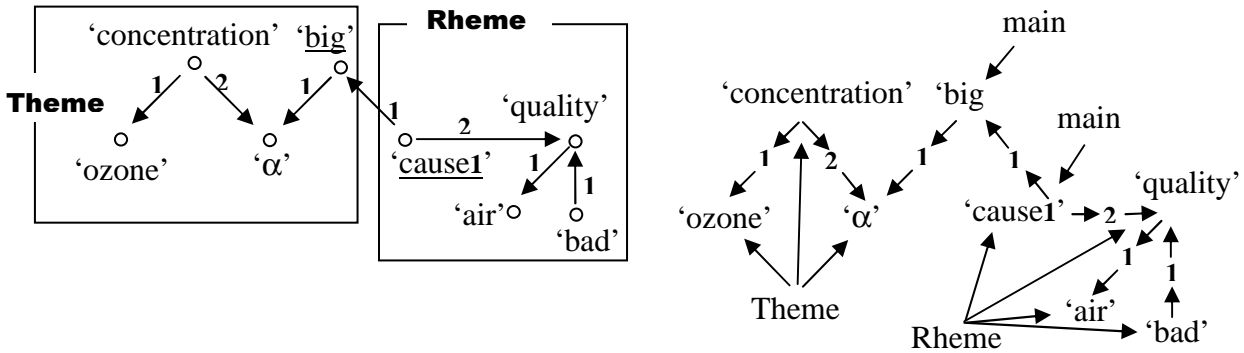


Figure 7: Semantic Representation

In contrast to FST and tree transducers, graph transducers can not have an explicit processing direction. However, rules can request with context on the target side that a distinct part has been already created by other rules. With the introduced matching approach, a chain of chars or a tree can be processed in the same manner by starting at the root or leaves and matching always the last created node by context of the target side. However, the application direction is not mandatory also for trees and chains (of chars). Applying the above matching technique for chains and trees is possible without change. More important, a processing direction is not forced by the used technique as it is the case by FST or tree transducers. But it is possible to have in access a fine grained manner to needed parts by accessing the result of other rules or if needed having a processing direction.

Form a linguistic point of view, looking at syntactic transitions, information such as case or number is moved down the tree like case for instance in German, but also up the syntax tree as number from the subject to the verb. In a lot of languages a determiner of a noun is placed independent from the verb before the noun and a sub-tree which is governed by the noun might be placed depending on the verb. Between this rules no order is needed. As already mentioned in subsection 2, the application order of rules should be only driven by linguistic requirements and not forced by the used technique.

5 Sketch of the Implementation

In order to implement finite state transducers, finite state machines (FSM) are used. Figure 8 shows an example of such an FSM which has two states. If it recognizes on the lexical level an consonant C, it outputs an consonant and enters the final state 2. In this state it can still accept other consonants or alternatively a c and output a t. For graph and tree transducers a similar technique can be used. We separated the left-hand side of the finite state machine from the right-hand side because we include additionally an RETE like optimization. Figure 9 shows such a network which is build out of the rules of Figure 4.

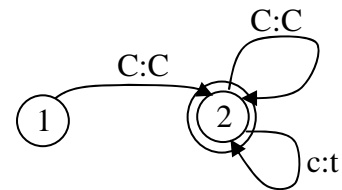


Figure 8: Finite state machine

The network uses the matching steps which are taken by the algorithm described in section 4 or the graph generation is inverted. The numbers in the circles represent the states. The edges are labeled

with the actions that are taken in the next step. Depending on the application direction, the matching starts with state 1 left and right, respectively.

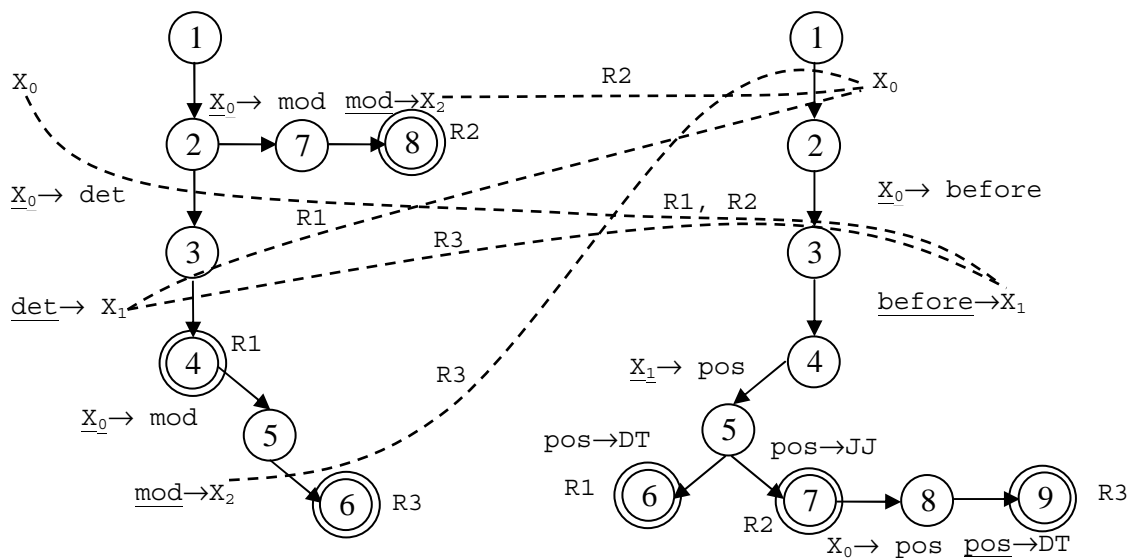


Figure 9: Graph matching and generation network (automata)

In the following, we consider that the left side is matched. In the first step, the variable X_0 is matched. The transition from state 1 to 2 describes this. The variable X_0 matches potentially to all nodes of a source graph for instance to the nodes of the tree left in Figure 4. In the next step, the automata can go into state 3 or 7. In the first case, an edge going to a node labeled with `det` is matched. Then the automata can go to state 4 matching an edge to a node with any label. State 4 is the finale state indicated by the double circle. It belongs to rule 1 (`det_v`) and means that the rule 1 is applicable. The right-hand side of the rule can be build by going up from the final state of a rule to state 1. For instance, the rule 1 starting at its final state 6, which is shown right builds by going up to state 1; the target part for the rule. The correspondence links are drawn as dashed line and labeled with the rule name to which they belong.

In the above procedure it is easy to integrate the *Kleene Star* (*), *plus operator*, *not existence* and *or*. The operators are only allowed as context like in the case of FST. The star operator and plus operator are just cycles, the not operator leads away from an already found state and the or operator is realized by alternative paths in the network.

Conclusion and Outlook

This paper shows that it is possible to have the advantages of FST also for tree and graph transducers by using two tapes. Even more, it seems to be possible to cover all transitions efficiently in one formalism. FST are failure-driven while tree and graph transducers define the correspondence by positive statements. It is more a matter of style to forbid the exceptions by additional rules or to restrict single rules. The practical usage has to show, if the implementation of morphology in a common framework is convenient. Finally, we introduced a matching algorithm which allows a fine grained ordering of rules depending only on linguistic requirements that can match efficient linguistic representations having different

structure like chains, trees and graphs. We sketched the implementation of the bidirectional matching and graph building algorithm which is able to handle operators like the *star*-operator, the *or*-operator, and the *not existence* operator for graph parts. A simpler version of the matching algorithm is implemented in the MATE rule interpreter and the described matching algorithm is realized in a graph transducer compiler that is currently being implemented.

Bibliography

Antworth, E. L. 1990. *PC-KIMMO: A two-level processor for morphological analysis*. Occasional Publications in Academic Computing No. 16. Dallas.

Beesley, K. and Karttunen, L. 2003. *Finite State Morphology*. CSLI Publications.

Bohnet, B. and Wanner, L. 2001. *On Using a Parallel Graph Rewriting Grammar Formalism in Generation*. Proceedings of the 8th ENLG Workshop. Toulouse.

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

Bohnet, B. 2006. *Textgenerierung durch Transduktion linguistischer Strukturen*. DISKI 298. Akademische V. G., Berlin.

Coch, J. 1996. *Overview of AlethGen*. Proceedings of the INLG-96. Herstmonceux, England.

Forgy, C. 1982. *RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem*, Artificial Intelligence.

Gerdes, K. 2002. *Topologie et grammaires formelles de l'allemand*. Thèse de doctorat Université Paris 7

Kahane, S. 2000. *Des grammaires formelles pour définir une correspondance*. Conférence TALN. Lusanne.

Kahane, S. and Lareau F. *Meaning-Text Unification Grammar: modularity and polarization*. Meaning Text Conference, Moscow 2005.

Knight, K.; Graehl, J. 2005 *An Overview of Probabilistic Tree Transducers for Natural Language Processing*. In Proc. of the 6th International Conference on Intelligent Text Processing and Computational Linguistics, Lecture Notes in Computer Science.

Koskenniemi, K. 1983. *Two-level morphology: a general computational model for word-form recognition and production*. Publication No. 11. University of Helsinki.

Lavoie, B., and Rambow, O. 1997. *A Fast and Portable Realizer for Text Generation Systems*. In Proceedings of the 5th ANLP, Washington DC.

Mel'cuk, I. A. 1988. *Dependency Syntax*. State University of New York Press. Albany. NY.

Thatcher, J.W. 1970 *Generalized sequential machine maps*. J. Computer Systems.

Rounds, W. 1970 *Mappings and grammars on trees*. Mathematical Systems Theory.