Towards a Workbench for Schema–TAGs*

Karin Harbusch and Friedbert Widmann and Jens Woch
University of Koblenz–Landau, Computer Science Department
E–mail: {harbusch|widi|woch}@uni-koblenz.de

Abstract

In the following the components of a workbench for the grammar formalism of Schema–Tree Adjoining Grammars (S–TAGs) *are outlined. This workbench can also serve as a workbench for pure TAGs because it provides a component which transforms an arbitrary TAG into an S–TAG in a non–trivial manner. Another interesting property of the workbench is that it provides a parser, which is realized as a reversible component to generate as well.*

Introduction

The formalism of augmenting *Tree Adjoining Grammars* with schemata was introduced in [Weir 87] in order to compress syntactic descriptions. For that purpose, a TAG (see, e.g., [Joshi 86]) is extended in order to provide the facility to specify a regular expression (RE). A RE is of type a.b, a+b, a^+, a^* and $a^{(0|n)}$, where a, b can uniquely refer to child nodes (via Gorn numbers) or a tree–modifying reference of the form g_1–g_2, where g_1, g_2 are Gorn numbers and g_2 denotes a subtree of g_1. This expression means that the subtree g_2 in g_1 is ignored and replaced with ϵ. Finally, a,b can be regular expressions themselves. Regular expressions are annotated at each inner node of an elementary tree. The resulting tree is called a *schematic elementary tree.* Such a tree denotes an elementary tree set just as a regular expression denotes some regular set. Thus, an individual scheme corresponds to a — possibly infinite — set of elementary trees, but itself is not the structural element to build derivation trees of.

In order to stress the power of compressing a grammar let us reconsider the coordination construction proposed in [Weir 87]. In Fig. 1, the root node NP of the substitution tree t_1 (which is element in the set of initial trees I) is annotated with a regular expression. In this regular expression, the Gorn number |n| refers to the n-th daughter of the node. For an illustration of this reference in the figure the numbers are explicitely annotated to the individual nodes. For instance, the regular

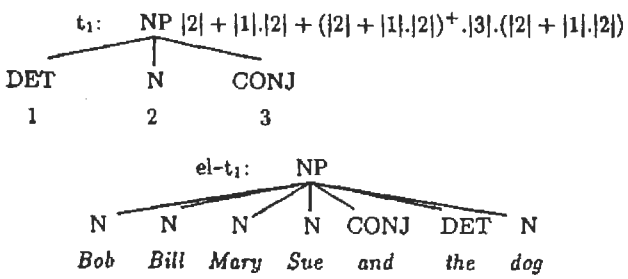

Figure 1: Coordination of NPs

expression |2| at the node NP in t_1 represents the tree with the root NP and the unique daughter N — e.g., producing *"John"*. The operation "." concatenates siblings in the same currently evaluated elementary tree. Accordingly, |1|.|2| produces an elementary tree where DET and N are the two daughters of NP (*"a man"*). The operation "+" enumerates alternative elementary trees. For instance, the regular expression |2| + |1|.|2| enumerates the two trees mentioned above. The exponents "+" and "*" produce infinite sets of elementary trees where the construction marked with such an exponent can be repeated arbitrarily often ("+" represents the infinite repetition exclusing zero occurrences and "*" inclusing zero). For instance, t_1 can produce *"Bob Bill Mary Sue and the dog"* (see tree el–t_1 in Fig. 1) but not *"and the dog"* because $(|2| + |1|.|2|)^+$ prevents the zero repetition so that at least N occurs. Furthermore a single *"and"* cannot be produced because no alternative in the regular expression at the root node starts with |3|. A finite number of repetitions can be written with the exponent $|x|^{(l|k)}$, where the component with the Gorn number x occurs at least l and up to k times.

Note, that the example is not lexicalized because Weir's dissertation proposal was earlier published than the definition of lexicalization (cf. [Schabes 90]). The coordination with Schema–TAGs works similarly with

*This work is partially funded by the DFG — German Research Foundation — under grant HA 2716/1–1.

lexicalization. Accordingly, the root node has two children (Simple_NP↓ and CONJ) and the RE is "|1| + (|1|+.|2|.|1|)". The substitution tree Simple_NP has two children (DET and N) and its root node is annotated with "|1| + |1|.|2|".

Description of the S–TAG Workbench

In the following, the components of an *S–TAG workbench (STAGWB)* are outlined. In the first subsection a facility to transform arbitrary TAG grammars (in our case the UPENN tree bench [Doran et al. 94]) into schematic trees. Then the reversible component for parsing and generation is outlined (for details s. [Woch et al. 98]).

Writing Grammar and Lexicon Rules

With respect to lexicalized TAGs [Schabes 90]) where each tree in the set of initial and auxiliary trees has at least one lexical leaf (called anchor) no lexicon component is required (cf. XTAG [Doran et al. 94]). But since the workbench should not determine the grammar formalism it is possible to specify a non–lexicalized TAG as well.

A main emphasis lies on the facility to transform an arbitrary TAG into an STAG. Obviously, an arbitrary TAG G can trivially be transformed into an S–TAG G' by annotating the concatenation of all daughters from left to right at each inner node of each elementary tree. Obviously, this transformation involves no compression. Therefore, the transformation component of the STAGWB produces an S–TAG which guarantees that each label at the root node occurs only once in the set of initial and auxiliary trees.

The component performs the following steps. Firstly, in all elementary trees all subtrees which do not contain the foot node are rewritten by substitution in order to find shared structures[1]. Since new non-terminals must be introduced to prevent the grammar from overgeneration, the adjoinable auxiliary trees are duplicated and root and foot nodes are renamed by the new non-terminals. Now, all alternatives for the same root node are collected. For each elementary tree where the root node is labelled with X (b_1, ..., b_n), a new schematic tree s_X is introduced to the S–TAG G' where its root node is labelled with X and the children result from enumerating all occurring children in all elementary trees b_1, ..., b_n without repeating the same label. In the

[1]Here, one can decide whether the structures are collapsed, although their features may differ. In the first case the disjunction of both feature descriptions is stored together with the history where they originally belonged to. Accordingly, more condensed structures are produced but the interpretation of the feature structures becomes more complicated.

```
((("NP" . "")))  ((("DetP" . "")) :substp T))
                 ((("N" . "")) :headp T)))
((("NP" . "")))  ((("N" . "")) :headp T)))
((("NP" . "r"))) ((("N" . "")) :headp T))
                 (((("S" . "")) :substp T)))
((("NP" . "")))  ((("DetP" . "")) :substp T))
                 ((("N" . "r")):constraints "NA"
                                :constraint-type :NA)
                          ((("N" . "")) :headp T))
                          ((("S" . "")) :substp T))))
((("NP" . "")))  ((("G" . "")) :headp T)))
((("NP" . "g"))) ((("NP" . "")) :substp T))
                 ((("G" . "")) :headp T)))
  ...
                                ⇓
((("NP" . "")) |1|.|2| + |2| + |2|.|3| + |1|.|4| + |5| + |6|.|5|)
                 ((("DetP" . "")) :substp T))
                 ((("N" . "")) :headp T))
                 ((("S" . "")) :substp T))
                 ((("N0" . "r")) :substp T))
                 ((("G" . "")) :headp T))
                 ((("NP" . "")) :substp T)))
((("N0" . "r")) |1|.|2|
                 :constraints "NA" :constraint-type :NA)
                 ((("N" . "")) :headp T))
                 ((("S" . "")) :substp T)))
```

Figure 2: Grammar transformation

next step the annotation of the root node of s_X is constructed by summing up all alternatives according to b_1, ..., b_n where all labels are rewritten as numerical references pointing at the respective child.

An instance of a grammar transformation is shown in Fig. 2[2]. Note, that here the first step of introducing substitutions does not have to do much, because most lexicalized TAGs already use substitution. The only new substitution node is N^0.

The resulting REs can be reformulated applying the following transformation rules:

1. $\alpha\gamma^{(i|k)}.\gamma\beta = \alpha\gamma^{(i|k+1)}\beta$,
2. $\alpha(\gamma.\delta_1)\beta + ... + \alpha(\gamma.\delta_m)\beta = \alpha\gamma.(\delta_1 + ... + \delta_m)\beta$
3. $\alpha\gamma\beta + \alpha\beta = \alpha\gamma^{(0|1)}\beta$

where $\alpha, \beta, \gamma, \delta_1, ..., \delta_m$ are arbitrary complex REs.

Note, that different compressing strategies result in different REs. For analysis grammars the rule of factoring out common prefixes is convenient, whereas the factorization according to common heads is more adequate in generation. E.g. in the example in Fig. 2 for analysis the two alternatives |1|.|2| and |1|.|4| result in |1|.(|2| + |4|). For generation the alternatives |1|.|2|+|2|+|2|.|3| result in $|1|^{(0|1)}$.|2|+|2|.|3|. Additionally, this example illustrates that an LD/LP–Schema–

[2]This transformation does not show the unification structures (c.f. footnote 1).

TAG can be advantageous especially for generation because there the alternative |2|.|3| can easily be incorporated in the compact expression.

Now, the automatically introduced substitution trees can be replaced with their original substructures and furthermore all added auxiliary trees can be eliminated again if desired. So the grammar becomes as lexicalized as it was before. Finally, in order to introduce "–" to the annotations the following process is carried out. According to the annotation of each substitution node r, substitution trees s_1 and s_2 are identified which only differ in one leaf l in s_1. For these candidates the structure must match beside the path to l. If so, the substitution of tree s_1 is explicitely realized and r is modified to refer to s_1 – <path-to-l> instead of referring to s_2.

S–TAG Parser

To be able to deal with REs and substitutions the parser extends the Earley–based TAG–parser by [Schabes 90] as follows:

Instead of computing the set of trees described by schemata (which is impossible due to its infinity) explicitely, the REs are interpreted as follows (cf. [Harbusch 94]): To indicate a certain position, $\odot$ is used to point into the current RE, i.e. $\alpha \odot \beta$ indicates, that α already has been computed. Then, two functions are introduced, namely SHIFT(ϕ), which shifts $\odot$ to the right, and NEXT(ϕ), which returns a set of nodes to be computed next. SHIFT is performed in each parsing step, in which the computation of a certain node is completed (indicated by raising the dot position to "ra"): scanning of terminals (`scanner`), the prediction of the right part[3] of auxiliary trees (`right prediction`) in which no prediction took place, and the completion of a root node of an auxiliary tree (`right completion`).

The output of NEXT is responsible for the computation of all alternatives given in the currently considered RE. Thus, each alternative g in β of NEXT($\alpha \odot \beta$) has to be taken into account for the prediction of new items. This is done in `move dot down`. Whenever an elimination $|a - b|$ occurs, it is deferred until node b is actually computed. Instead of processing b an ϵ–scan is simulated. This usually is done in `scan` obviously, but also may take place in `left prediction`, if b is non-terminal.

In order to reflect substitutions, two new operations are introduced. The formerly forbidden case of non-terminal leafs now triggers the prediction of all possible substitution trees. On the other hand, the formerly end-test-only state of being at position "ra" for non-auxiliary roots now serves for the completion of predicted substitution trees.

S–TAG Generator

As modern workbenches (cf., e.g., the workbench PAGE for Head-driven Phrase Structure Grammar [Netter, Oepen 97]) usually provide a generator, our parser is parametrised to work for generation according to the idea of bidirectional processing (cf., e.g., [Neumann 94]).

As outlined by [Shieber et al. 90] a naïve structure–driven top–down generator may not terminate (e.g. for genitive phrases in English and German). Furthermore the approach is inefficient because the input does not guide the generation process. Instead of that, possible syntactic structures are realized and their corresponding logical forms are compared to the semantic input structure.

A more natural way of guiding the generation process is to make it driven by the semantic input structure (indexing on meaning instead of indexing on string position). Generally speaking such generator predicts semantic heads. Two different procedures continue searching for a connection to sub– and the super–deriviation tree.

In the terminology of [Shieber et al. 90] the generator predicts pivots. A pivot is defined as the lowest node in the tree such that it and all higher nodes up to the root node or a higher pivot node have the same semantics. According to the definition of a pivot node the set of grammar rules consists of two subsets. The set of chain rules consists of all rules in which the semantics of some right–hand side element is identical to the semantics of the left–hand side. The right–hand side element is called the semantic head. The set of non–chain rules contains all rules which do not satisfy this condition. The traversal will work top–down from the pivot node only using non–chain rules whereas the bottom–up steps which connect the pivot node with the root node only use chain rules.

Adapting this mechanism to the generation of lexicalized TAGs means that the chain rules are completely determined by the elementary tree under consideration[4]. Adjoining and substitution represent the application of non–chain rules. In order to illustrate this kind of processing let us assume that the input structure is (frequently(see(John,friends))). Furthermore, we assume that the grammar allows to pre-

[3]Due to the possibility of arbitrary mix–ups of precedences of children by REs, the expressions "left/right to" are to be understood in a more temporal than local manner, i.e. "left of the foot node" encloses all those items that have been computed before computing the foot.

[4]Since empty semantic heads can be associated with their syntactic realization they can be processed in the same manner.

dict the trees described in Fig. 3. Since here is not the space to outline the specification lists of the individual nodes, the semantics of the trees is informally annotated at the nodes where x and y are variables to be filled during the unification at that node.

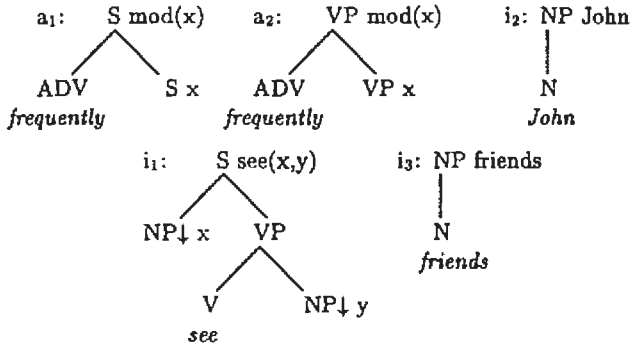

Figure 3: Predictible pivots

In a first step all predictible pivots according to the input structure can be written to the one and only item set during processing. This construction represents the unordered processing of the semantic structure. The bracketing structure of the logical form is achieved by evaluating the semantic expression associated with each elementary tree (e.g. for tree a_1 mod(x) where x is a value filled by the subtree of the foot node. The processing is successful only if a derivation tree can be constructed where all elements of the logical form occur only once[5].

Concerning the example two realizations for the input specification can be produced. The processing of the one with the sentential adverb (adjoing of a_1) is obvious whereas the adjoining of a_2 is not so clear. It also works because the variable x at the foot node is unified with the VP node of i_1 where according to the pivot definition the semantics on the spine from the root to the V node is identical. So, x contains the whole expression (see(John,friends)) and the check whether the bracketing structure is correct (i.e. the dependencies, specified in the logical form), is successful as well.

Final Remarks

All modules are implemented in JAVA [Gosling et al. 98]. Currently we run our transformation module to build a Schema–TAG equivalent to the English TAG by [Doran et al. 94]. Furthermore, we test how the average runtime varies for TAGs and Schema–TAGs. The differing size and depth of elementary trees is of special interest in incremental generation (cf. [Harbusch 94]) where the size of structures influence the time in which the processing can be finished and results can be handed over to other components.

Another topic of current considerations is how to define LD/LP–Schema–TAG which are especially interesting for generation. We assume that it suffices to rewrite the NEXT function to adapt our parser to run LD/LP–Schema–TAGs on the structural level. Our suggestion is that the separation of structural combination and linear ordering saves processing time, especially for generation.

[5]Since the bracketing structure is tested explicitely during the combination of elementary trees the accepting condition can be weaker so that the logical form equivalence problem (cf. [Shieber 93]) does not occur here.

References

[Doran et al. 94] C. Doran, D. Egedi, B.A. Hockey, B. Srinivas, M. Zaidel. XTAG System — A Wide Coverage Grammar for English. In the Proceedings of the *15th COLING*, Kyoto/Japan, 1994.

[Gosling et al. 98] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification* Addison-Wesley, Reading, 2nd. ed. 1998

[Harbusch 94] K. Harbusch. Incremental sentence processing with Schema–Tree Adjoining Grammars. In the procs. of the *3rd International TAG+ Workshop*, Paris, Frankreich, September 1994, TALANA–Report 94–01, pp. 41–44.

[Joshi 86] A.K. Joshi. The convergence of mildly-contextsensitive grammar formalisms. In T. Wasow, P. Sells, eds., *The Processing of Linguistic Structures.* MIT-Press, Cambridge, MA/USA, 1986.

[Netter, Oepen 97] K. Netter, S. Oepen. PAGE – Platform for Advanced Grammar Engineering. Slides (at http://cl-www.dfki.uni-sb.de/pagegifs/slides.html), Saarbrücken/Germany, 1997.

[Neumann 94] G. Neumann. *A Uniform Computational Model for Natural Language Parsing and Generation.* PhD thesis, Saarbrücken, Germany, 1994.

[Shieber et al. 90] S.M. Shieber, F.C.N. Pereira, G. van Noord, R.C. Moore. *Semantic–Head–Driven Generation. Computational Linguistics*, 16(1): 30–42, 1990.

[Shieber 93] S.M. Shieber. *The Problem of Logical–Form Equivalence. Computational Linguistics*, 19(1): 179–190, 1993.

[Schabes 90] Y. Schabes. *Mathematical and Computational Aspects of Lexicalized Grammars.* PhD thesis, Philadelphia, PA/USA, 1990.

[Weir 87] D. Weir. Characterising Mildly Context–Sensitive Grammar Formalisms. PhD. Proposal, University of Pennsylvania, Philadelphia/USA, 1987.

[Woch et al. 98] J. Woch, F. Widmann, K. Harbusch. *A Reversible Approach to Parsing and Generation of Schema-TAGs.* Technical Report, University of Koblenz, forthcoming.