Direct Parsing of Schema–TAGs*

K. Harbusch & J. Woch
University of Koblenz–Landau, Computer Science Department
E–mail: {harbusch|woch}@uni-koblenz.de

Schema–Tree Adjoining Grammars (S–TAGs) are an extension of *Tree Adjoining Grammars (TAGs)* (for an introduction see, e.g., [5]), initially introduced by [7]. In a *schematic elementary tree*, a *regular expression (RX)*[1], is annotated at each inner node of an elementary tree. This means, that the elementary schemata enumerate a possibly infinite set of elementary trees. For instance, see the tree t_1 of Fig. 1, which performs NP–coordination as, e.g., the zero–coordination *Peter* (|2|) or *Bob, Bill, Mary, Sue, and the dog* (|2|^4.|3|.|1|.|2|). Obviously, the schemata of S–TAGs provide a condensed

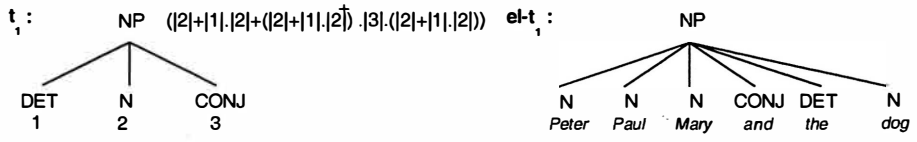

Figure 1: NP-Coordination

grammar representation. Concerning the structures, TAGs and S–TAGs are identical and hence all advantages of TAGs are conserved such as the extended domain of locality and the mild context–sensitivity. Thus S–TAGs are well suited for linguistic specifications.

In order to analyse S–TAGs, an ordinary TAG parser, e.g., the Earley–based parser by Yves Schabes ([6]), can only be applied if the length of the input string is considered in order to compute a finite subset of applicable elementary trees. This is disadvantegeous because the computation must be performed at runtime and the property of condensed representations is lost. Therefore we introduce a *direct parser of S–TAGs* which works directly on the schemata. Accordingly, common prefixes (i.e. substructures) of the same scheme are represented only once instead of enumerating all explicit orderings in the individual elementary trees. This leads to a better average runtime. However, the worst case time complexity remains the same as for TAGs ($O(n^6)$). The direct parser adapts the basic idea of an Earley parser [2] to the exploration of regular expressions. This means, that a new dot position ($\odot$) is defined which indicates whether a prefix of an alternative in a regular expression is analysed. In order to move the $\odot$, two operations SHIFT and NEXT are defined. $SHIFT(\phi)$ moves $\odot$ by one token to the right, i.e. $SHIFT(\alpha \odot \beta) = \{\gamma | \gamma = \alpha NEXT(\beta) \odot (\beta \setminus NEXT(\beta))\}$ where $NEXT(\phi)$ returns a set of all alternatives for the next symbol (Gorn address). $NEXT(\alpha) = \{\odot\alpha\}$ iff α is a Gorn address; $NEXT(\alpha) = \{\odot\beta_1, ..., \odot\beta_n\}$, iff $\alpha = (\beta_1 + ... + \beta_n)$, $n \geq 2$ and $NEXT$ is applied recursively as long as β_i starts with an opening bracket, finally a Gorn address is reached[2]; $\{\odot\beta^+.\gamma,$

*This work is partially funded by the DFG — German Research Foundation — under grant HA 2716/1–1 and 1–2.

[1]RXs are inductively defined. Let α, β and $\beta_1, ..., \beta_n$ ($n \geq 2$) be Gorn addresses uniquely referring to daughters or arbirarily complex RXs, then $\alpha.\beta$ (concatenation of branches), $(\beta_1 + ... + \beta_n)$ (enumeration of alternatives), α^* (Kleene Star) and α^+ (α^* without the empty repetition) are RXs. Finally, "–" allows to cut off a subtree at the end of a path. As an abbreviation we write $\alpha^{(n|m)}$ which enumerates $\sum_{i=0}^{m} \alpha^{n+i}$ ($n, m \geq 0$). Notice, "." binds stronger than "+".

[2]Notice sums are always bracketed even if no binding conflict occurs as on the top level (cf. footnote 1).

$\beta^*.\odot\gamma\}$, iff $\alpha = \beta^*.\gamma$; $\{\odot\beta.\beta^*.\gamma\}$, iff $\alpha = \beta^+.\gamma$. Here also $NEXT$ is recursively computed as long as β (and γ for $\beta^*.\odot\gamma$) starts with an opening bracket in order to reach the Gorn addresses to be finally returned. Traversing the regular expressions replaces the analysis of branches in the six procedures of the TAG parser by Schabes in a straight–forward manner. For more details of the extension of the individual procedures of the Schabes parser see [3] and [8].

With the close relation to ordinary TAGs and the claim that direct parsing is advantageous, the question arises how can existing TAGs made available to the direct parser? 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. This transformation involves no compression and accordingly, no benefit of a direct S–TAG parser arises. Therefore, we have developed a transformation component which produces a S–TAG where each label at the root node occurs only once in the set of initial and auxiliary trees. The following steps lead to this goal: Firstly, in all elementary trees all subtrees which do not contain the foot node are rewritten by substitution in order to find shared structures. Some reformulations become necessary to prevent the grammar from overgeneration because existing substitution trees with the same label would become applicable unintentionally. Accordingly, the corresponding trees are renamed. Now, all alternatives with the same root node are collected and described by a regular expression denoting the set of all these trees. Finally, the resulting RXs are condensed by applying the distributivity law ($\alpha\gamma^{(l|k)}\gamma\beta = \alpha\gamma^{(l|k+1)}\beta$, $\alpha(\gamma\delta_1)\beta + ... + \alpha(\gamma\delta_m)\beta = \alpha\gamma(\delta_1 + ... + \delta_m)\beta$ and $\alpha\gamma\beta + \alpha\beta = \alpha\gamma^{(0|1)}\beta$ where $\alpha, \beta, \gamma, \delta_1, ..., \delta_m$ are arbitrary complex RXs). This procedure has been applied to the XTAG system [1] and the transformed grammar serves a syntax grammar for analysis and generation as we run our direct parser as a *reversible uniform generation modul* (cf. [3] and [4]). Currently we transform existing semantic and pragmatic knowledge sources into the S–TAG formalism.

References

[1] C. Doran, D. Egedi, B. Hockey, B. Srinivas & M. Zaidel. XTAG System — A Wide Coverage Grammar for English. In M. Nagao, ed., *Proceedings of the 15th International Conference on Computational Linguistics*, vol. 2, pp. 922–928, Kyoto, Japan, 1994.

[2] J. Earley. An Efficient Context–Free Parsing Algorithm. *Communications of the Association for Computing Machinery*, 13(2):94–102, 1970.

[3] K. Harbusch, F. Widmann & J. Woch. Towards a Workbench for Schema-TAGs. In A. Abeillé, T. Becker, O. Rambow, G. Satta & K. Vijay-Shanker, eds., *Procs. of the 4th International TAG+ Workshop*, pp. 56 – 61, Philadelphia, PA, USA, 1998. IRCS-Report 98–12.

[4] K. Harbusch, F. Widmann & J. Woch. Ein reversibles Analyse–/Generierungsmodul für Schema-Tree Adjoining Grammars. In C. Habel & T. Pechmann, eds., *Sprachproduktion*. Westdeutscher Verlag, Wiesbaden, Germany, 2000.

[5] A. K. Joshi & Y. Schabes. Tree Adjoinung Grammars. In G. Rozenberg, A. Salomaa, eds., Handbook of Formal Languages, Vol. 3, pp. 69–214, 1997.

[6] Y. Schabes. *Mathematical and Computational Aspects of Lexicalized Grammars*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA, USA, 1990.

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

[8] J. Woch & F. Widmann. Implementation of a Schema-TAG-Parser. Fachberichte Informatik 8–99, Universität Koblenz, Koblenz, Germany, 1999.