How to prevent adjoining in TAGs and its impact on the Average Case Complexity

Jens Woch
University of Koblenz, Germany

The introduction of the adjoining operation to context-free grammars comes at high costs: The worst case time complexity of (Earley, 1968) is $O(n^3)$, whereas Tree Adjoining Grammars have $O(n^6)$ ((Schabes, 1990)). Thus, avoiding adjoining as far as possible seems to be a good idea for reducing costs (e.g. (Kempen and Harbusch, 1998)). Tree Insertion Grammars (TIGs, (Schabes and Waters, 1995)) address this problem more radically by restricting the adjoining operation of TAGs such that it is no context-sensitive operation anymore. The result is $O(n^3)$ worst case parseability which stems from TIG's context-freeness. However, to preserve TAG's mildly context-sensitiveness the adjoining operation must not be restricted in any way. Another solution would be simply to call the adjoining operation less frequently: The production of items directly depends on the fashion of the underlying grammar and often adjoining is used to make the grammar more comprehensible or more adequate to the linguistic phenomenon even if there would be simpler representations as, for instance, left- or right recursion.

This abstract (1st) sketches a way of reducing item generation through grammar transformation by using Schema-TAGs (S-TAGs, as introduced by (Weir, 1987), where tree sets are enumerated by regular expressions) which in contrast to TIGs keeps weak equivalence and performs better than factoring as proposed by (Harbusch, Widmann and Woch, 1998), and (2nd) provides a proof of the average case time complexity of S-TAGs based on the proposed transformation.

In the following, adressing of nodes occurs in the fashion of (Gorn, 1967), i.e. each node of a tree gets a unique number – beginning with zero – which preserves the structure of the tree. For example, 1.2 points to the second daughter of the first daughter of a root node, and in grammar G_1 of Fig. 2, $(A_2,2)$ identifies the foot node of A_2. The regular expressions of S-TAGs are defined as "schema descriptors": Let g be a Gorn number, then

- $|g|$ is a schema descriptor.
- If μ and ν are schema descriptors, then $\mu+\nu$, describing the alternative between μ and ν, is a schema descriptor.
- If μ and ν are schema descriptors, then $\mu.\nu$, describing the concatenation of μ and ν, is a schema descriptor.
- If μ is a schema descriptor, so are (μ) (bracketing), μ^* (arbitrary iteration) and $\mu^{(0|n)}, n \in \mathbb{N}$ (n times iteration) schema descriptors.
- If $|g|$ is a schema descriptor, so is $|n-n.g|, n \in \mathbb{N}$ a schema descriptor (the via g addressed subtree is cut out from the via n addressed (sub)tree).

Reduction of item production by factoring

A first idea of circumventing adjoining is to avoid it by reducing generation of items, on which *Predict* can be applied. Thus, the idea is to aggregate common substructures appropriately, i.e. to condense the grammar in order to get rid of redundancies (Harbusch, Widmann and Woch, 1998).

In general, factoring is applied to the schema descriptors (the regular expressions) of the S-TAG and can be done by applying, e.g., the following rules:

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

where $\alpha, \beta, \gamma, \delta_1, ..., \delta_m$ are arbitrary complex schema descriptors. Rule (1) saves nothing, since the amount of predictions remains the same, as well as the amount of alternatives $k-l$. However, applying it reversely, in combination with rule (4) in case of alternatives it is probably possible to factor γ up to l times, such that $\gamma^{(0|k-l)}$ remains as infix, which is not reduce-able any more. The factoring of rule (2) not only saves $m-1$ predictions of γ, but also of α and β. Thus, rule (2) saves $3(m-1)$ predictions. Finally, rule (3) reduces the number of predictions of α and β down to 1, while rule (4) divides the number of predictions caused by γ by 2.

Generally, the above rules save $\sum_{i=1}^{|N|} c|ADJ(\eta_i)|$ predictions and $\sum_{i=1}^{|N|} m_i c|ADJ(\eta_i)|$ completions with m_i being the number of occurrences of node $\eta_i \in N$, which can be factored out c times. Obviously, each circumvented

 Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6), pp. 102–107. Universitá di Venezia.

prediction saves the whole cycle of left and right prediction, resp. completion. Additionally, and much more important, it reduces the amount of hypothesis checking needed within the operations, as expressed by the factor m_i of the above sum.

Factoring, however, is not able to effectively prevent (unnecessary) adjoinings, it simply reduces redundancies. This strategy only pays back if direct processing mechanisms are applied, who do not explicitely iterate alternatives of schemata as instances of trees (cf. (Harbusch and Woch, 2000)).

Preventing adjoining (at least, partially)

A more promising approach to circumvent adjoining is to prevent it. The use of null- and selective constraints, which restrict adjoining at the node, at which they are attached, perform this very good, but the application of constraints always should be linguistically motivated. However, adjoining is often used for modelling simple left, resp. right recursion of the context-free form $S \rightarrow \alpha S$, resp. $S \rightarrow S\alpha$. Auxiliary trees, whose terminal nodes are unexceptional either to the left or to the right of the root-to-foot spine are modelling exactly this behaviour. Transforming those trees do not compromise the underlying linguistic concept. Those trees can be merged with the help of schema descriptors into the trees, in which they should adjoin and after merging can completely be removed from the auxiliary tree set. Hence their prediction is effectively prevented.

Definition 1 (Simplified S-TAG (S^2-TAG)) *Let the 6-tuple $G = (N, T, S, I, A, \Sigma)$ be a Schema-TAG.*

- *The tree schema $\beta \in A$ is called right recursive, if and only if for each tree $t_{|\beta}$ holds, that the foot node of t is the rightmost node of the frontier of t.*
- *The tree schema $\beta \in A$ is called left recursive, if and only if for each tree $t_{|\beta}$ holds, that the foot node of t is the leftmost node of the frontier of t.*

G is called Simplified S-TAG, or S^2-TAG, if for each $\beta \in A$ holds, that β is neither left nor right recursive.

Theorem 1 (L(S^2-TAG) = L(S-TAG)) *Let G be an S-TAG and G′ be an S^2-TAG. Then $L(G) = L(G')$.*

This is shown by transforming auxiliary left and right recursive tree schemata into equivalent schema descriptor representations and *vice versa*. Def. 1 states, that an auxiliary schema β is right/left recursive, if and only if all $t_{|\beta}$ are right/left recursive. With other words, for the decision whether a schema is right/left recursive, its schema descriptor has to be investigated. To make things easier, this aspect is not mentioned in the following discussion, but should be kept in mind.

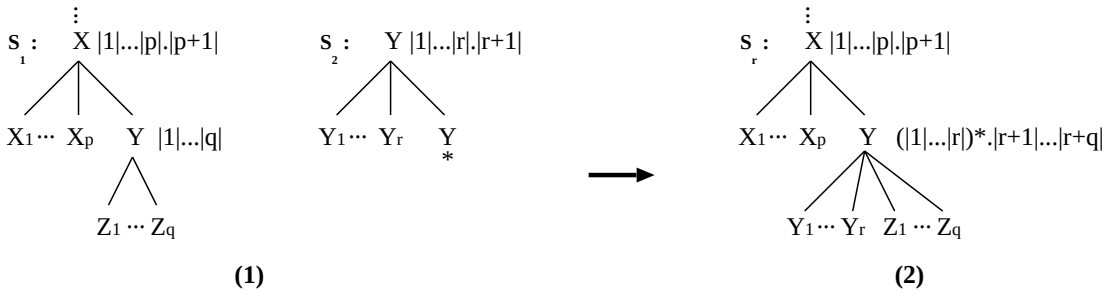

Figure 1: Avoiding (right-recursive) adjoining by iteration

L(S-TAG) $\subseteq$ L(S^2-TAG):
Case 1: Right recursion
To be applied to each $\beta \in ADJ(\eta), \eta \in N$, where β is right recursive. Fig. 1 shows the elimination of an auxiliary schema, which effectively performs right recursion at node $(S_1, Y)^1$. Without loss of generality, the descriptors

1. It is not possible to give correct absolute Gorn addresses for nodes of S_1 in that example, because it is not known, whether X is the root node or not. Hence, for the time being, the Gorn number is replaced with the node label, e.g. (S_1, Y), which also should be non-ambiguous in Fig. 1.

shown in Fig. 1 are simple enumerations of the respective children nodes, in order to make the relocation arithmetics clear, but they could be arbitrary complex, as long as the position of the foot node does not change.

Given that S_2 is adjoinable at (S_1, Y), the substructures of $(S_2, 0)$ (without foot node), namely Y_1 ... Y_r, are copied over as children of (S_1, Y) to the left of its own children Z_1 ... Z_q, giving the sibling's sequence $Y_1 ... Y_r Z_1 ... Z_q$. Then the descriptor $(|1| \dots |r|)^*$, which licenses the arbitrary repetition of S_2's children Y_1 ... Y_r, is added as prefix to the descriptor of (S_1, Y). After that, the Gorn numbers $|1| \dots |q|$, which referred to the now right-shifted nodes Z_1 ... Z_q, are updated to $|r| \dots |r+q|$. Apply this to any node Y of grammar G, except for foot nodes, because they would be destroyed by attaching children to it[2]. See Fig. 2 for an example how the right-recursive auxiliary tree A_1 is eliminated through the transformation process.

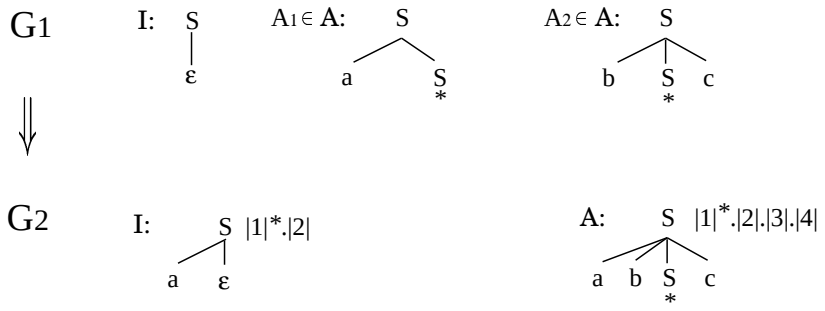

Figure 2: $L(G_1) = \{w | w \in \{a,b\}^* c^n, |b| = |c|\} = L(G_2) = \{w | w \in (a^*b)^n a^* c^n\}$

However, things are getting more complicated, if $|ADJ(\eta)| > 1$, i.e. there exist more than one auxiliary schema β with root node η: The prefixes, which iterate the substructures of the auxiliary schemata have to be combined as alternatives: Having the descriptors $\sigma_i, \dots, \sigma_j$ from the root node of auxiliary schemata $\beta_i, \dots, \beta_j$ would result in descriptor $\sigma = (\sigma'_i + \dots + \sigma'_j)^*.\sigma'_r$. The respective Gorn numbers $\{|1|, \dots, |\beta_{im_i}|\} \in \sigma_i, \dots, \{|1|, \dots, |\beta_{jm_j}|\} \in \sigma_j$, would be relocated to

$$\begin{aligned}
\{|1|, \dots, |\beta_{im_i}|\} &\in \sigma'_i \\
&\vdots \\
\{|\beta_{im_i} + \dots + \beta_{j-1m_{j-1}} + 1|, \dots, |\beta_{im_i} + \dots + \beta_{jm_j}|\} &\in \sigma'_j \\
\{|\beta_{im_i} + \dots + \beta_{jm_j} + 1|, \dots, |\beta_{im_i} + \dots + \beta_{jm_j} + q|\} &\in \sigma'_r
\end{aligned}$$

This relocation reveals an important characteristics of the transformation process: Either the transformation is done simultaneously, or later transformations lead to unhealthy grammars: If, e.g., after transforming $\beta_i, \dots, \beta_j$ the auxiliary tree β_{j+1} has to be transformed, then the resulting descriptor σ falsely would be $(\sigma_{j+1})^*.(\sigma_i + \dots + \sigma_j)^*.\sigma_r$ instead of $(\sigma_i + \dots + \sigma_j + \sigma_{j+1})^*.\sigma_r$.

Case 2: Left recursion
Analogue.

Case 3: Both recursions
To be applied to each $\beta \in ADJ(\eta), \eta \in N$ holds, that, where β is either right or left recursive. This case is performed like left and right recursion, but simultaneously: Let $\sigma_i, \dots, \sigma_j$ be schema descriptors from the root node of left recursive auxiliary schemata $\beta_i, \dots, \beta_j$, and $\sigma_k, \dots, \sigma_l$ schema descriptors from the root node of right recursive auxiliary schemata $\beta_k, \dots, \beta_l$, each without the Gorn number for their foot nodes. Let σ_r be the original descriptor of the node, in which $\beta_i, \dots, \beta_l$ may adjoin. Then the resulting schema descriptor is $(\sigma'_i + \dots + \sigma'_j)^*.\sigma'_r.(\sigma'_k + \dots + \sigma'_l)^*$ with the Gorn number relocations are performed as described above.

2. This suffices, because the children are attached as well to the node, in which adjoining occurs.

After applying case 1,2 or both for each node $\eta \in N, ADJ(\eta)$ is either empty, or $\exists\beta \in ADJ(\eta)$ with β is neither left nor right recursive. See Fig. 3 for an example how the right-recursive auxiliary tree A_1 and the left-recursive auxiliary tree A_2 or eliminated simultaneously through the transformation process.

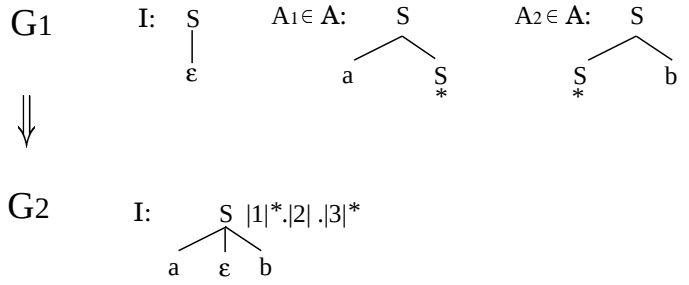

Figure 3: G_2 is the result of transforming G_1

L(S^2-TAG) $\subseteq$ L(S-TAG):
Trivial, since each S^2-TAG is an S-TAG per definition. ■

Gentle application of rules (1)–(4) of page 102 to σ might be advisable. As a side note, transforming the grammar $G = \{S \Rightarrow \gamma,\ S \Rightarrow S\beta,\ S \Rightarrow \alpha S\}$ simultaneously is not the same as transforming $G' = \{S \Rightarrow \gamma,\ S \Rightarrow \alpha S\beta\}$ (which is not allowed, anyway). The former grammar G expresses $\alpha^n\gamma\beta^m$, whereas G' expresses $\alpha^n\gamma\beta^n$, but fatally, after transformation the resulting descriptors would be identical for G and G': Something according to $\alpha^*\gamma\beta^*$.

Whilst the method of transforming whenever possible auxiliary schemata greatly reduces the costs as shown below, there are two aspects to consider:

- Rewriting of auxiliary tree schemata raises the redundancies of the grammar, such that maintainability suffers.
- Furthermore, as side effect of the last point: The principle of locality of TAGs may get lost, where (groups of) auxiliary schemata are completely replaced with that method. Thus, the intended linguistic concept of that (group of) auxiliary schemata is scattered throughout the grammar.

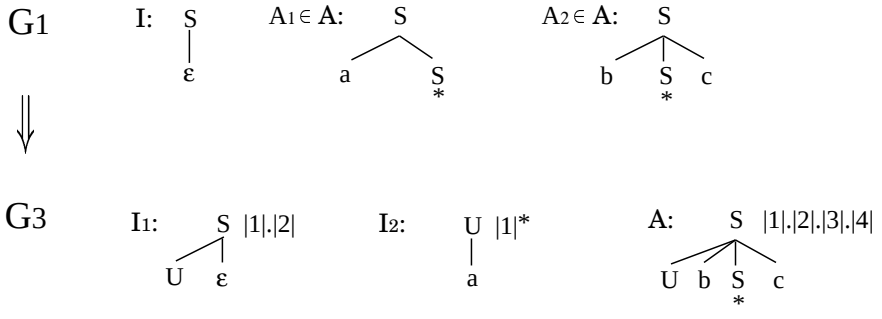

Figure 4: Transformation of G_1 of Fig. 2 with substitution

A solution to get rid of those drawbacks is to introduce unique substitution tree schemata for each transformed auxiliary schema as shown in Fig. 4. The modularisation effect of substitution remedies the redundancies similar to adjoining and preserves the locality of simple left or right auxiliary schemata.

Theorem 2 (Average Case of S^2-TAGs) *The average case for S^2-TAGs is $O(n^4)$.*

The assumption here is that in S^2-TAGs adjoining rarely happens, since the most common, i.e. pure left or right auxiliary schemata are not existent. Those non-transformable schemata, which are neither left nor right recursive, however, are very seldom even in large grammars. The XTAG grammar ((Doran *et al.*, 1994)), for example, consists of more than 1000 trees and only about three non-transformable ones. Therefore, the average case assumption is that there are barely any adjoining operations at all.

According to (Schabes, 1990), *Right Completion* and *Move Dot Up* are by far the most expensive operations: Their worst case complexity is $O(n^5)$ and this is the reason for the overall worst case complexity of $O(n^6)$. Assuming that there are no adjoinings, the *left/right prediction* and *completion* operations *(LP1, LC, RP1, RC)* as well as *move dot up (MU2.2)* (the dotted node subsumes a foot node) do not apply to any state. The latter ommission reduces the complexity of *MU* to the complexity of *MU2.1*(the dotted node does not subsume a foot node), which is $O(n^3)$. Following from the worst case behaviour of S-TAGs, *MU* is still the most expensive operation. Therefore, the algorithm takes at most $O(n^3)$ time to process a given state set S_i. Having at most n state sets, the overall complexity for S^2-TAGs under above assumption is $O(n^4)$. As a side note, Schabes' algorithm performs worse than Earley for complete context-free TALs.

Further investigations

Whilst $L(G_1) = L(G_2) = \{w|w = a^n b^m\}$ (cf. Fig. 3), attaching indices to the terminals in the order of their prediction makes clear that the number of hypotheses varies heavily between G_1 and G_2: *aabb* can be predicted by G_2 in the sequence $a_1 a_2 b_3 b_4$ only. G_1, however, is able to predict it as any sequence of $\{a_i a_j b_k b_l | ijkl \in PERM(1234)\}$. The consequence is the prediction of all possible sequences, which leads to heavily interconnected derivation graphs. Applying null constraints at $(A_1, 2)$ and $(A_2, 1)$ would help to reduce the number of predictions by the half, because the sequence of a and b would be determined to $a_j \ldots a_i$, $b_k \ldots b_l$ with $i < j$ and $k < l$, but nevertheless the prediction order of a's and b's would be mixed up. The transformed grammar G_2, however, only permits one derivation, and that is linear from left to right. Thus, the total number of items produced by G_2 is a fraction of that of grammar G_1, as depicted in Fig. 5 and 6, in which the relative growth of time (TR), number of items (IR) and number of operations performed (OR) are shown. In G_1, e.g. for $n = 20$ approx. 5500 times more operations are performed than for $n = 2$. The time behaviour of G_2 is linear, and that is not surprising, since the only production rule left in G_2 is regular. In G_2, e.g. for $n = 20$ only about 7 times more operations are performed than for $n = 2$. The open question is whether this is an artifact of the example's simplicity, or whether this behaviour can be expected and to what degree in larger grammars after transforming them into S^2-TAGs.

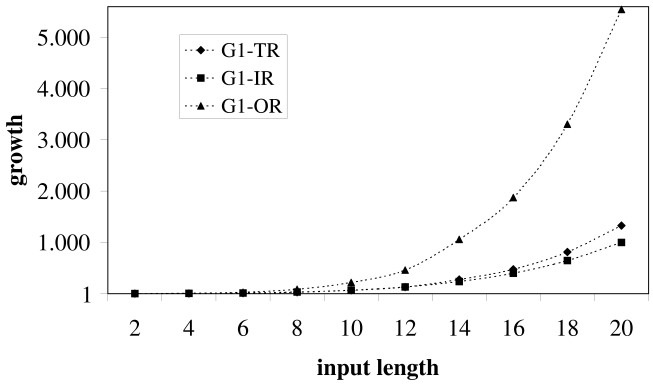

Figure 5: Relative growth of item numbers (IR), operations (OR) and time consumption (TR) for G_1

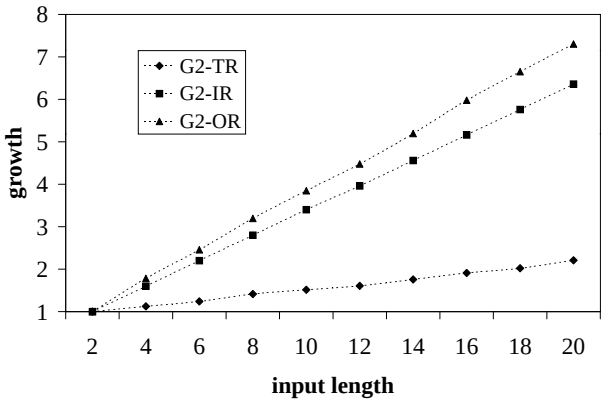

Figure 6: Relative growth of item numbers (IR), operations (OR) and time consumption (TR) for G_2

References

Doran, Christine, Dania Egedi, Beth Ann Hockey, Bangalore Srinivas and Martin Zaidel. 1994. XTAG System — A Wide Coverage Grammar for English. In Makoto Nagao, editor, *Procs. of the 15th International Conference on Computational Linguistics (COLING)*, volume 2, pages 922–928, Kyoto, Japan, August 23–28.

Earley, Jay. 1968. *An Efficient Context–Free Parsing Algorithm*. Ph.D. thesis, Carnegie-Mellon University, Pittburgh, PA, USA.

Gorn, Saul. 1967. Explicit definitions and linguistic dominoes. In John Hart and Satoru Takasu, editors, *Systems and Computer Science*. University of Toronto Press, Toronto, Canada, pages 77–115.

Harbusch, Karin, Friedbert Widmann and Jens Woch. 1998. Towards a Workbench for Schema-TAGs. In Anne Abeillé, Tilman Becker, Owen Rambow, Giorgio Satta and K. Vijay-Shanker, editors, *Procs. of the 4th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+4)*, pages 56–61, University of Pennsylvania, Philadelphia, PA, USA, August 1–2. Institute for Research in Cognitive Science. IRCS-Report 98–12.

Harbusch, Karin and Jens Woch. 2000. Direct Parsing of Schema–TAGs. In Harry C. Bunt, editor, *Procs. of the 6th International Workshop on Parsing Technologies (IWPT)*, pages 305–306, Trento, Italy, February 23–27. Institute for Scientific and Technological Research.

Kempen, Gerard and Karin Harbusch. 1998. Tree Adjoining Grammars without Adjoining? The Case of Scrambling in German. In Anne Abeillé, Tilman Becker, Owen Rambow, Giorgio Satta and K. Vijay-Shanker, editors, *Procs. of the 4th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+4)*, pages 80–83, University of Pennsylvania, Philadelphia, PA, USA, August 1–2. Institute for Research in Cognitive Science. IRCS-Report 98–12.

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

Schabes, Yves and Richard C. Waters. 1995. Tree Insertion Grammar: Cubic-Time, Parsable Formalism that Lexicalizes Context-Free Grammar without Changing the Trees Produced. *Computational Linguistics*, 21(4):479–513.

Weir, David. 1987. Characterising Mildly Context–Sensitive Grammar Formalisms. PhD. proposal, University of Pennsylvania, Philadelphia, USA, 1987.