Two-step TAG Parsing Revisited

Peter Poller, Tilman Becker
DFKI GmbH, Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
{poller,becker}@dfki.de

Introduction

Based on the work in (Poller, 1994) and a minor assumption about a normal form for TAGs, we present a highly simplified version of the two-step parsing approach for TAGs which allows for a much easier analysis of run-time and space complexity. It also snggests how restrictions on the grammars might result in improvements in run-time complexity.

The main advantage of a two-step parsing system shows in practical applications like Verbmobil (Bub et al., 1997) where the parser must look at multiple hypotheses supplied by a speech recognizer (encoded in a *word hypotheses lattice*) and filter out illicit hypotheses as early as possible. The first (context-free) step of our parser filters out some illicit hypotheses fast ($O(n^3)$); the constructed parsing matrix is then reused for the second step, the complete ($O(n^6)$) TAG parse.

Simplifying Root and Foot Nodes

The normal form that we assume in the following is only a very minor modification and allows for a trivial retrieval of parses from the results of the normal form-based parser.

We call a TAG *clean* if the root node of every elementary tree and the foot node of every auxiliary tree is labeled with the null-adjoining constraint. Obviously, every TAG can be transformed into a clean TAG by simply adding to every elementary tree an additional node, immediately dominating the root node, with the same label as the root node and the null-adjoining constraint and also adding an additional node, immediately dominated by the foot node, with the the same label as the foot node and the null-adjoining constraint (see figure 1). While this transformation adds new nodes to the derived trees, no adjunctions can take place at these additional nodes and they can easily be eliminated again from a derived tree, resulting in the derived tree of the original grammar. Thus, every TAG can be transformed into an *"almost"* strongly equivalent clean TAG.

In a clean TAG, no adjunction can take place at the root or foot node. This allows us to drop numerous special data structures and steps from the algorithm in (Poller, 1994), resulting in a much cleaner presentation. We also omit the treatment of linear precedence rules, which can easily be added.

A Simplified Two-Step TAG parser

An initial offline step is the extraction of the *context-free kernel* from the TAG G, a context-free grammar G_K which overgenerates, i.e., $L(G) \subset L(G_K)$.

The first step of the parser is a standard parse with the Earley-algorithm (Earley, 1970). The second step is the repeated *elimination* of adjoined trees from the parser's matrix. Thus a TAG derivation is constructed *inside-out*[1].

First, we describe the additional data structure which is added to the items of the Earley parser. An item is a tuple $(i, j, S \rightarrow \alpha \bullet \beta)$, representing a derivation of $a_{i+1} \ldots a_j$ from α. In addition, every non-terminal node in the item carries a list with *node numbers*, taken from the TAG grammar G, uniquely identifying a node in an elementary tree of G which contributed the rule $S \rightarrow \alpha\beta$. Furthermore, every node number in an item can store a list of pointers, called *foot node pointers* (see below). Figure 2 shows two elementary trees and an example item with node numbers.

[1] Or rather bottom-up in terms of the derivation tree of the TAG.

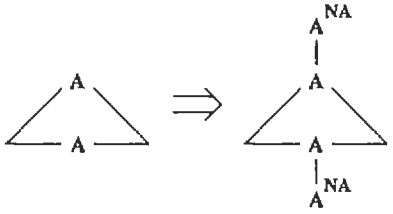

Figure 1: Transforming an auxiliary tree into a clean tree.

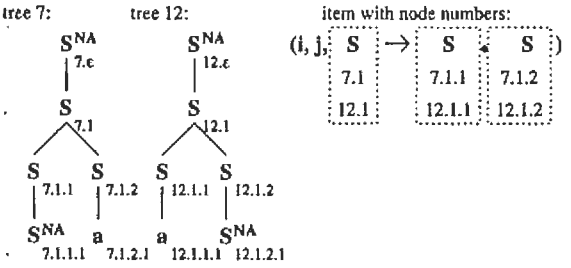

Figure 2: Two example trees and an item from the context-free kernel with node numbers.

Our stepwise approach to TAG-parsing is open to different degrees of precision for the context-free parsing step depending on how much information about the elementary TAG-trees is integrated into the context-free parsing step. We expect that the following alternatives have different influences on the realtime behaviour of a practical system mainly depending on the grammar's characteristics (size, ambiguitiy, ...).

(1) Solely the node labels are used to generate the context-free kernel. In this case the node numbers attached to the terminals and nonterminals have no influence at all on the Earley operators. In this paper we describe this alternative.

(2) An other possibility is to integrate the node addresses from the elementary TAG trees into the rules of the context-free kernel[2]. This requires extensions of the Earley operators because they are now controlled by the characteristics of a specific node of a TAG tree instead of just a label. In particular, the prediction of a nonterminal node now only produces items for context-free rules that are valid according to the TAG grammar and also don't violate any adjunction constraint of the predicted node. This allows for the integration of the TAG constraint check into the context-free parsing step. Similarly, the completor also works only with valid derivation steps according to the TAG grammar. On the other hand, we cannot share node number alternatives in one item anymore. But this increases the overall number of items only by a constant factor.

While the first alternative filters out only invalid context-free derivation steps with respect to node labels, the second one is a stronger filter because it only produces items which represent locally valid derivation steps with respect to the TAG grammar but reduced to the context-free domain of locality. Furthermore it requires one item for each occurence of a context-free rule in different TAG trees. This is a trade off between the number of items to be produced in the first parsing step and the precision of its filtering effect.

It is interesting to note that it is also possible to derive the node number specific items of the second alternative from the parsing matrix of the first one. If the node number check is organized top-down starting with successful context-free derivations (similar to the initialization of the TAG parsing step below) we get a 3-step parser functioning as a cascade of filters.

Independent of these alternatives there is a special parsing strategy for lexicalized TAGs (Schabes et al., 1988). As each terminal is associated with a set of elementary trees we can immediately restrict the relevant TAG trees for the parser to those that are associated with the terminals of the input string. This strategy can still be applied since the rules of the context-free kernel can be computed in advance for each elementary tree separately. Once the relevant elementary trees are determined for an input string, the context-free kernel is simply the union of the associated context-free rules.

For all variants of the context-free parsing step, the second step (the actual TAG parsing step) remains basically the same.

Within the second step an initialization procedure filters out irrelevant items by a top-down traversal starting from roots of successful context-free derivations through the parsing matrix. This sets the ground for an iterated elimination of complete, adjoined trees. This initialization is not strictly necessary (and takes $O(n^3)$ time), but it provides an important speed-up because now only valid context-

[2]Thanks to the anonymous reviewer who suggested this procedure.

free derivations are considered. Invalid context-free derivation steps are filtered out which might become relevant in practical systems with large grammars.

Initially, all leaf nodes (including foot nodes) are marked, i.e., the corresponding node numbers in all items, are labeled *ok*, then these initial *ok's* are propagated "bottom-up" along the context-free derivation steps if they took place inside the same elementary tree which can easily be checked by comparing the unique node numbers. This *ok*-propagation also propagates relevant information about foot node positions.

While recovering elementary trees in the parsing matrix, we need to keep track of possible foot node positions. Each node number in an item is associated with a set of corresponding *foot node pointers*. A foot node pointer points to a particular node number in some item. Thus, when an *ok* is eventually propagated to a node number that represents the root node of an elementary tree, all possible positions of its foot node have been collected in the *foot node pointer* list. Note that there can be $O(n^2)$ foot node pointers for each node number, since there are $O(n^2)$ items.

The relevant computational steps during the iteration are: *elimination, upwards propagation,* and *horizontal propagation. Elimination* of an adjoined tree in the Earley matrix is realized by propagating all *ok's* from immediately "below" all possible foot nodes to all immediate supertrees of the root node[3]. *Upwards propagation* is the propagation of an *ok* from a complete[4] item to its ancestor. *Horizontal propagation* is the propagation of an *ok* to an item where the dot has moved one position to the right.

In the following, all complexity statements are based on the limited number of items that are produced by the Earley algorithm, in particular the number of items in a so called itemlist[5]. Each itemlist I_k contains at most $O(k)$ items so that the number of all items produced by the Earley algorithm is bound by: $\sum_{k=0}^{n} O(k) = O(n^2)$. Another important point for our complexity statements is that each individual item is stored exactly once by the Earley algorithm (even though it might be derived by more than one operation), which means that there are no two identical items.

We can now present a sketch of the algorithm:

```
for j from 0 to n
  for i from j downto 0
    foreach item (i, j, A → α • β)
```

There are only three cases for a node number N of A labeled *ok* of an item $(i, j, A \rightarrow \alpha \bullet \beta)$:

1. $\beta = \epsilon$:
   - 1.1 N is the root of an auxiliary tree: perform an *elimination* of all encodings of this tree,
     This can be done in $O(n^4)$ time.
   - 1.2 N is an inner node of an elementary tree: perform an *upwards propagation.*
     This can be done in $O(n^3)$ time.
2. $\beta \neq \epsilon$:
   perform a *horizontal propagation.*
   This can require $O(n^3)$ time.

```
end foreach; end for j; end for i;
```

The most expensive step is *elimination* (step 1.1). For each root node of an adjoined tree to be eliminated there can be $O(n^2)$ foot node pointers because there are at most $O(n^2)$ items to which they can point to. They result in $O(n^2)$ positions from which this tree can be eliminated, i.e., *ok's* at these positions and their foot node pointer lists must be propagated. Collecting all these foot node pointers lists (of size $O(n^2)$ each) from each of the $O(n^2)$ positions results in $O(n^4)$ time complexity (see figure 3). It is important to note that this computational step cannot produce more than $O(n^2)$ new foot node pointers at the current root node although their computation costs $O(n^4)$. Therefore the fact that each node has at most $O(n^2)$ foot node pointers is an invariant of the iterative elimination.

The complexity of step 1.2 (*upwards propagation*) is also based on the limited number of foot node pointers. Since the *ok* of a node is propagated to its "context-free" ancestors and all possible ancestors are contained in the same itemlist, the complexity is limited by the $O(n^2)$ foot node pointers and the $O(n)$ items ("*supertree*" in figure 4) to which they have to be propagated to, which results in $O(n^3)$ time complexity.

Finally, step 2 (*horizontal propagation*) can also be done in $O(n^3)$ time. Again, $O(n^2)$ foot node pointers of an *ok* have to be propagated to all items where the dot has moved one position

[3]Implementations of the concepts "below" and "supertree" are already provided by the Earley parser.

[4]A complete item has the dot at the rightmost position.

[5]An itemlist I_k is defined as the set of all items $(i, j, S \rightarrow \alpha \bullet \beta)$ where $k = j$.

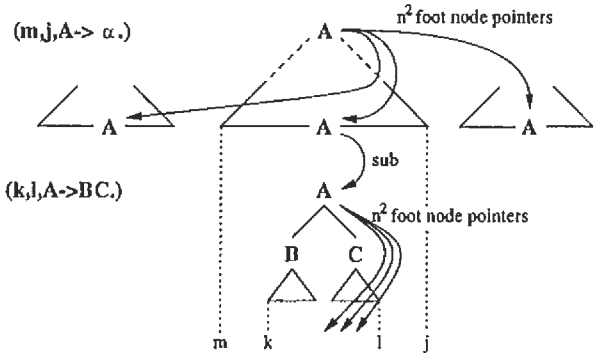

Figure 3: Complex elimination of a completely recognized auxiliary tree.

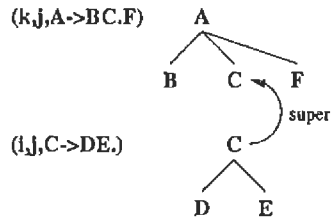

Figure 4: Simple propagation of *ok* to the ancestor node, i.e., the corresponding items.

to the right. Since there are no identical items there can be at most $O(n)$ items (one item from each itemlist; see figure 5)[6].

$$(i, j, S \rightarrow \alpha \bullet X\beta) \overset{next}{\rightarrow} (i, j+k, S \rightarrow \alpha X \bullet \beta)$$

Figure 5: Horizontal propagation of *ok* to the next item.

The overall time complexity of the parsing algorithm is $O(n^6)$ since there are $O(n^2)$ items for which *elimination* ($O(n^4)$) must be performed.

Obviously, the two-step parsing algorithm does not have the correct prefix property (Nederhof, 1997) as it requires the entire sentence to be analyzed by the Earley parser before the second (TAG parsing) step begins. However, the Earley step itself has the correct prefix property wrt. the context-free kernel and also the discussion in (Poller, 1994) of a completely incremental setup also applies to the simplified two-step TAG parsing algorithm presented here.

[6]The concepts "next" and "previous" explicitly represent links between items coming from dot movements by the Earley parser.

Current Work

Although our parser does not have the correct-prefix property it can run incrementally as described in (Poller, 1994) namely by running the TAG parsing step in parallel to the construction of the context-free parsing matrix. Although this may require additional computational steps on unsuccessful context-free derivation steps, the effects on the realtime behavior of a practical system again depend on grammar characteristics. So it would be very helpful to find some kind of "grammar classification" with respect to their "parser suitability" in practical implementations answering the question "Which TAG parser is best suitable for my current task?".

The analysis of the *elimination* step shows clearly that the time complexity of our TAG parser stems from the number of possible foot node positions. We are currently investigating whether certain restrictions on TAG grammars can lower this number. E.g., this is obviously the case for unambiguous grammars.

References

Thomas Bub, Wolfgang Wahlster, and Alex Waibel. 1997. Verbmobil: The combination of deep and shallow processing for spontaneous speech translation. In *Proceedings of ICASSP-97*, pages 71–74, Munich.

J. Earley. 1970. An efficient context–free parsing algorithm. *Communications of the ACM*, 13(2):94–102.

M. J. Nederhof. 1997. Solving the correct-prefix property for TAGs. In T. Becker and H.-U. Krieger, editors, *Proceedings of the Fifth Meeting on Mathematics of Language (MOL5)*, number D-97-02, pages 124–130, Saarbrücken, August. DFKI GmbH.

Peter Poller. 1994. Incremental parsing with LD/TLP-TAGs. *Computational Intelligence*, 10(4):549–562, November.

Yves Schabes, Anne Abeillé, and Aravind K. Joshi. 1988. Parsing strategies with 'lexicalized' grammars: Application to Tree Adjoining Grammars. pages 578–583, Budapest, August.