TAG in XDG with Copy and Paste

Ralph Debusmann

June 23, 2004

1 Introduction

This article encodes of Tree Adjoining Grammar (TAG) (Joshi 1987) into a two-dimensional instance of Extensible Dependency Grammar (XDG). We restrict our attention to TAG grammars where each elementary tree has precisely one anchor.

By encoding TAG into XDG, we can make use of the constraint-based parsing techniques developed for XDG also for TAG (Duchier 1999), (Duchier 2001). Among the highlights of XDG parsing is an efficient treatment of lexical ambiguity, and the possibility to postpone enumeration of analyses. XDG parsing also allows for the integration of preferences, which can be utilized to find the best solution without having to generate all solutions in advance. This avenue of research has been started in (Dienes, Koller & Kuhlmann 2003), and is in the focus of active research.

The encoding also allows us to make use of large-scale TAG grammars for English (XTAG, (Group 2001)) and French (FTAG, (Abeillé & Candito 2000)) in the context of XDG. Another line of our research is the development of a syntax-semantics interface for XDG. Because this syntax-semantics interface relies on dependency trees (ID trees) and is independent of word order, we could also use plug it in to obtain a syntax-semantics interface for these large TAG grammars.

Finally, the encoding can give us insights into the comparison of the grammar formalism of Topological Dependency Grammar (TDG) (Duchier & Debusmann 2001), from which XDG originates, and TAG.

2 Example

As an example, consider the TAG grammar G for the language $L = \{ww \mid w \in \{a, b\}^+\}$. We show the elementary trees of the grammar in Figure 1. Here, a superscripted $\emptyset$ indicates a null adjunction constraint. $\downarrow$ to the right of a node label indicates a substitution site. A star $*$ indicates the foot node of an auxiliary tree. The numbers to the right of the node labels indicate the Gorn address of the node. The symbols to the left of the elementary trees are the names of the elementary trees. α_1, α_2, α_3 and α_4 are initial trees, whereas β_1 and β_2 are auxiliary trees.

Let us consider the example derivation of the string *abbabb*, displayed in Figure 2 and Figure 3. In each derived tree in the derivation, we draw the tree into which another is substituted or adjoined in black, and the tree which is

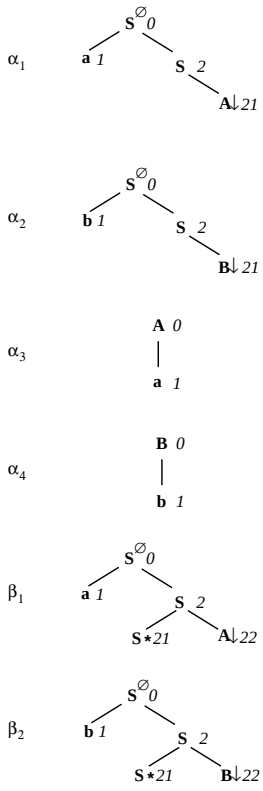


Figure 1: Elementary trees of G

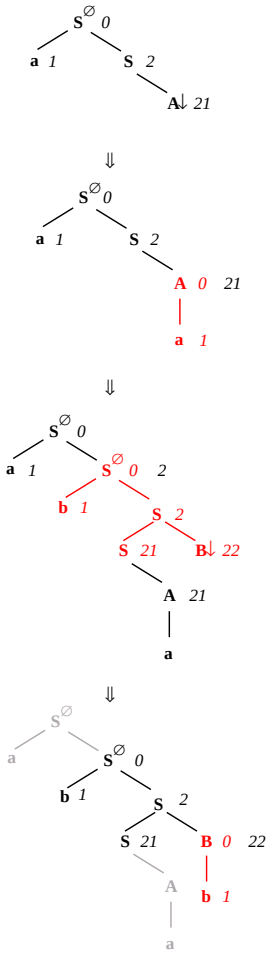


Figure 2: Part 1 of the derivation of *abbabb*

substituted or adjoined in red. Tree parts taken over from previous derivation steps in are gray.

In Figure 4, we show the derivation tree. Here, solid lines indicate adjunction, and dotted lines substitution. The edge labels indicate the Gorn address where either adjunction or substitution has taken place.

3 Encoding

We model a TAG derivation using a two-dimensional instance of XDG, with the two dimensions ID (Immediate Dominance) and LP (Linear Precedence). The models of both dimensions share the same set of nodes which correspond 1:1 to TAG elementary trees.

The ID dimension models the TAG derivation tree. The models of the ID dimension are unordered trees. An edge in the ID tree corresponds to either a substitution or an adjunction. The edge labels in the ID tree correspond to the label of the node of the substitution or adjunction.

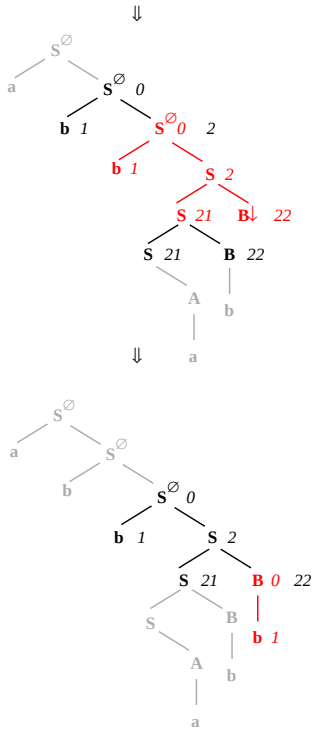


Figure 3: Part 2 of the derivation of *abbabb*

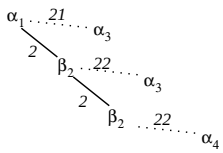


Figure 4: Derivation tree of *abbabb*

The LP dimension extends the ID dimension with the required information to recover the correct order of the leaves in the corresponding TAG derived tree. The models of the LP dimension are ordered directed acyclic graphs (dags). An edge in the LP dag corresponds to either 1) a substitution or an adjunction, or 2) a *copy and paste* operation. The edge labels in the LP dag correspond to Gorn addresses, either 1) of the node of the substitution or adjunction, or 2) of the foot node of an auxiliary tree (in case of the copy and paste operation).

In the following sections, we illustrate the idea of our encoding of TAG into XDG. In particular, we illustrate how we encode elementary trees (initial and auxiliary trees), and how we model substitution and adjunction.

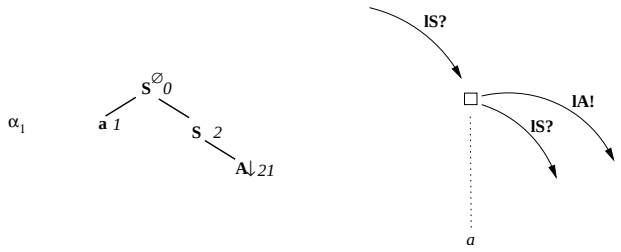
3.1 ID dimension

3.1.1 Initial trees

Initial trees can be substituted at substitution nodes whose label equals their root node label, or they can be the starting tree of a derivation. Hence, in the encoding, each initial tree licenses zero (for the case if it is the starting tree of a derivation) or one (if it is substituted) incoming edges labeled with its root node label.

Initial trees can have substitution and adjunction nodes. Substitution nodes correspond to requiring precisely one outgoing edge labeled with the label of the substitution node (since substitution is obligatory). Adjunction nodes correspond to requiring zero or one outgoing edges labeled with the label of the adjunction node (since adjunction is optional). Null adjunction constraints correspond to requiring that there are no outgoing edges labeled with the label of the node with the null adjunction constraint.

For instance, consider the initial tree α_1 of our example grammar:



(1)

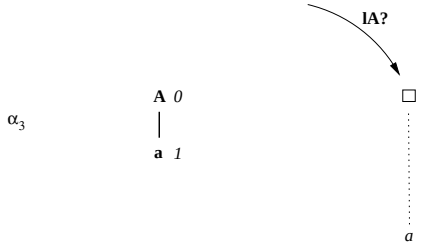
To its right, we show its ID dimension encoding. Here, the word a is connected to the corresponding to its corresponding node in the dependency graph by a dotted projection edge. The edge pointing to this node indicates the XDG constraints on incoming edges. The edges emanating from this node indicate the XDG constraints on outgoing edges. For each edge label l, $l!$ stands for requiring precisely one edge labeled l, $l?$ stands for requiring zero or one edges labeled l, and $l*$ stands for requiring zero or more edges labeled l. We prefix the node labels of the TAG trees with **l** for *label*.

The word a in the dependency graph corresponds 1:1 to the anchor of the elementary tree. Hence, because we restrict ourselves to TAGs where each elementary tree has precisely one anchor, the node connected to the word in the dependency graph corresponds 1:1 to the elementary tree, and, transitively, the word also corresponds 1:1 to the elementary tree.

The encoding of α_1 licenses zero or one incoming edges labeled with the node label of its root, viz. **S** (**lS?**). That is, it can be the starting tree of a derivation (no incoming edge) or it can be substituted at a substitution node whose label is also **S**.

The encoding of α_1 licenses zero or one outgoing edges labeled **lS** (**lS?**), corresponding to the adjunction node at address 2 (with node label **S**). The encoding requires precisely one outgoing edge labeled **lA** (**lA!**), corresponding to the substitution node at address 21 (with node label **A**).

As another example, consider the initial tree α_3 of our example grammar:



(2)

The encoding of α_3 licenses zero or one incoming edges labeled with the node label of its root, viz. **A** (**lA?**).

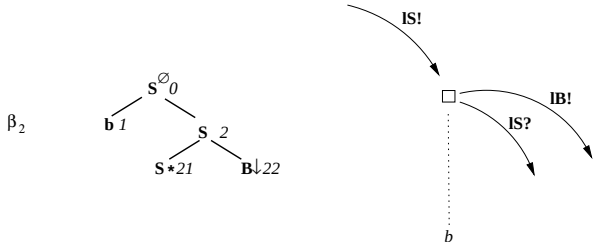
The encoding of α_3 licenses no outgoing edges; it has neither substitution nor adjunction nodes.

3.1.2 Auxiliary trees

Auxiliary trees can be adjoined into adjunction nodes whose label equals their root node label. Hence, in the encoding, each auxiliary tree licenses precisely one incoming edge labeled with its root node label. They cannot be the starting tree of a derivation, which is why they *must* have an incoming edge.

The outgoing edges of auxiliary trees are modeled in the same way as for initial trees.

As an example, consider the auxiliary tree β_2 of our example grammar:



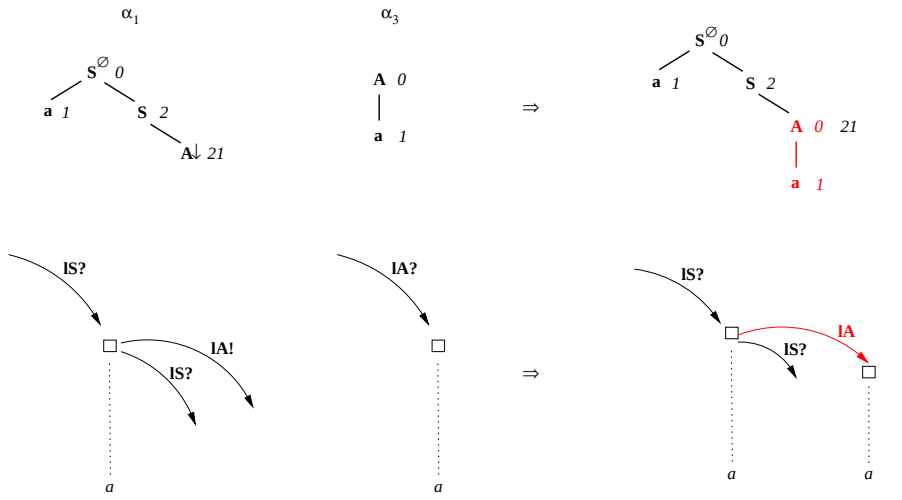
(3)

The encoding of β_2 licenses precisely one incoming edge labeled with the node label of its root, viz. **S** (**lS!**). Notice that contrary to the encoding of the initial tree α_1, there *must* be an edge labeled **lS** since auxiliary trees cannot be starting trees (and therefore cannot be the root of a derivation tree).

The encoding of β_2 licenses zero or one outgoing edges labeled **lS** (**lS?**), corresponding to the adjunction node at address 2 (with node label **S**). It requires precisely one outgoing edge labeled **lB** (**lB!**), corresponding to the substitution node at address 22 (with node label **B**).

3.1.3 Substitution

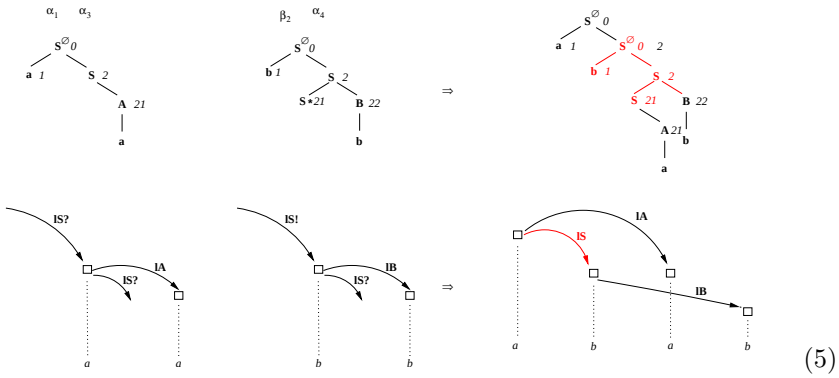
On the ID dimension, each substitution is reflected by an edge from the tree which is being substituted in into the substituted tree, labeled with the TAG node label of the substitution site (which equals the root of the substituted tree). Here is an example, where the encoding of the initial tree α_3 is substituted into the encoding of the initial tree α_1. We draw the new edge mirroring the substitution in red:



(4)

3.1.4 Adjunction

On the ID dimension, each adjunction is reflected by an edge from the tree which is being adjoined in into the adjoined tree, labeled with the TAG node label of the adjunction site (which equals the root of the adjoined tree). Here is an example, where the encoding of the auxiliary tree β_2 is adjoined into the encoding of the initial tree α_1 (into which the initial tree α_3 has already been substituted). We draw the new edge mirroring the adjunction in red:



(5)

3.2 LP dimension

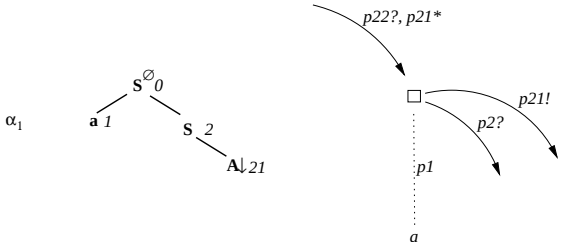
3.2.1 Initial trees

Initial trees can be substituted at any substitution site. Hence, they license zero or one incoming edges labeled with the address of any of the substitution sites in the grammar. In addition, initial trees can be *copied and pasted* at the foot node of an arbitrary number of adjoined auxiliary trees. Hence, they license zero or more incoming edges labeled with the address of any of the foot nodes in the grammar.

Initial trees can have substitution and adjunction nodes. Substitution nodes correspond to requiring precisely one outgoing edge labeled with the address of the substitution node (since substitution is obligatory). Adjunction nodes correspond to requiring zero or one outgoing edges labeled with the address of the adjunction node (since adjunction is optional). Null adjunction constraints correspond to requiring that there are no outgoing edges labeled with the address of the node with the null adjunction constraint.

We position the anchor with respect to its daughters by the node label[1] corresponding to the Gorn address of the anchor.

As an example, consider the initial tree α_1 of our example grammar:



(6)

We prefix the node labels of the TAG trees with p for *position*.

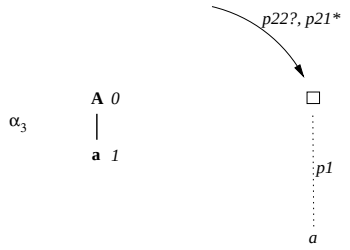
The encoding of α_1 licenses zero or one incoming edges labeled with the address of any of the substitution nodes in the grammar (here: *p22* and *p21*). In addition, it licenses zero or more incoming edges labeled with the address of any of the foot nodes in the grammar (here: *p21*). Because Gorn address 21 can be either a substitution node or a foot node, we license zero or more incoming edges labeled with *p21*.

The encoding of α_1 licenses zero or one outgoing edges labeled *p2* (*p2?*), corresponding to the adjunction node at address 2. The encoding requires precisely one outgoing edge labeled *p21* (*p21!*), corresponding to the substitution node at address 21.

The anchor of α_1 is at address 1; therefore the node label of the node in the dependency graph is *p1*.

[1] Here, we talk about node labels in the dependency graph, not in the TAG trees.

As another example, consider the initial tree α_3 of our example grammar:



(7)

Like α_1, α_3 licenses zero or one incoming edges labeled *p22*, and zero or more incoming edges labeled *p21*.

The encoding of α_3 licenses no outgoing edges; it has no substitution or adjunction nodes.

The anchor of α_3 is at address 1; therefore the node label of the node in the dependency graph is *p1*.

3.2.2 Auxiliary trees

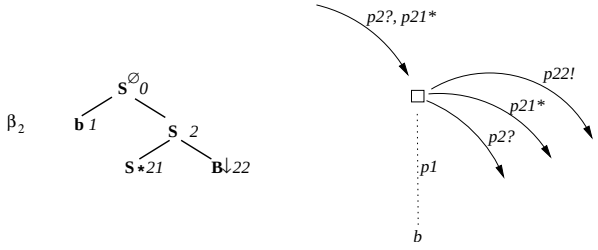
Initial trees can be adjoined at any adjunction site. Hence, they license zero or one incoming edges labeled with the address of any of the adjunction sites in the grammar. In addition, initial trees can be *copied and pasted* at the foot node of an arbitrary number of adjoined auxiliary trees. Hence, they license zero or more incoming edges labeled with the address of any of the foot nodes in the grammar.

Like initial trees, auxiliary trees can have substitution and adjunction nodes, and null adjunction constraints. All three are modeled in the same way as for initial trees.

In addition, auxiliary trees have a foot node where an arbitrary number of nodes, *copied* earlier in the derivation, can be *pasted*. Hence, auxiliary trees license zero or more outgoing edges labeled with the address of their foot node.

We position the anchor with respect to its daughters by the node label corresponding to the Gorn address of the anchor.

As an example, consider the initial tree β_2 of our example grammar:



(8)

The encoding of β_2 licenses zero or one incoming edges labeled with the address of any of the adjunction nodes in the grammar (here: *p2*). In addition, it licenses zero or more incoming edges labeled with the address of any of the foot nodes in the grammar (here: *p21*).

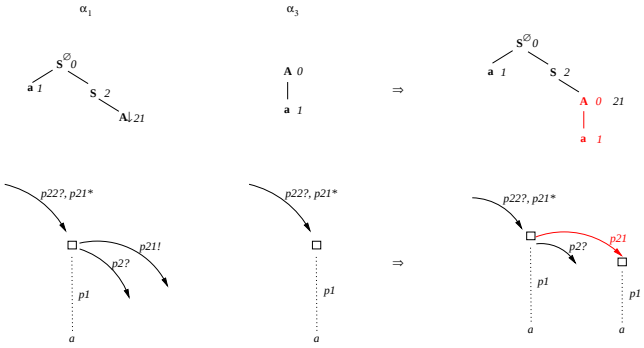
The encoding of β_2 licenses zero or one outgoing edges labeled *p2* (*p2?*), corresponding to the adjunction node at address 2. The encoding licenses zero or more outgoing edges labeled *p21* (*p21**), corresponding to the foot node

at address 21. The encoding requires precisely one outgoing edge labeled *p22* (*p22!*), corresponding to the substitution node at address 22.

The anchor of α_1 is at address 1; therefore the node label of the node in the dependency graph is *p1*.

3.2.3 Substitution

On the LP dimension, each substitution corresponds to an edge from the tree which is being substituted in into to the tree which is substituted, labeled with the address of the substitution site. Here is an example, where the encoding of the initial tree α_3 is substituted into the encoding of the initial tree α_1. We draw the new edge mirroring the substitution in red:

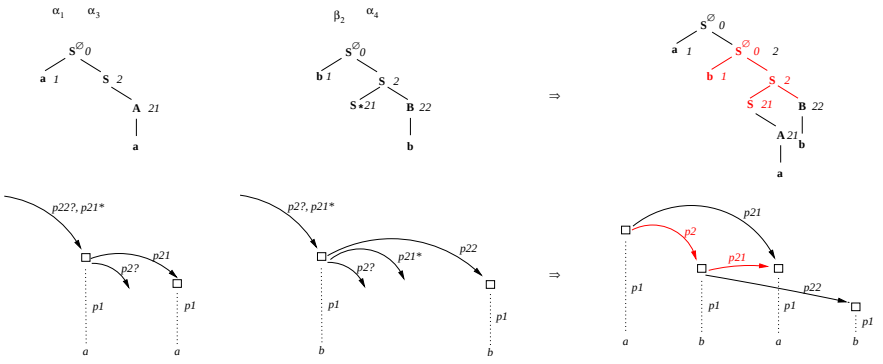


(9)

3.2.4 Adjunction

On the LP dimension, each adjunction corresponds to 1) an edge from the tree which is being adjoined in into the tree which is adjoined, labeled with the address of the adjunction site, and 2) a set of edges from the tree which is adjoined to the set of *pasted* nodes, which have been *copied* from the tree which is being adjoined in.

Here is an example, where the encoding of the auxiliary tree β_2 is adjoined into the encoding of the initial tree α_1 (into which the initial tree α_3 has already been substituted). We draw the new edges in red. The new edge labeled *p2* corresponds to the adjunction. The new edge labeled *p21* *pastes* the node *copied* from the tree which has been adjoined in at the foot node of the adjoined tree:



(10)

4 Principles

In this section, we list the XDG principles used for the encoding. We differentiate between one-dimensional principles (separately for the ID and LP dimensions) and two-dimensional principles (which stipulate relational constraints between the ID and LP dimensions).

4.1 ID dimension

4.1.1 In

The in principle restricts the number and label of the incoming edges of each node. We use it to state the restrictions on incoming edges explained in section 3.

4.1.2 Out

The out principle restricts the number and label of the outgoing edges of each node. We use it to state the restrictions on outgoing edges explained in section 3.

4.2 LP dimension

4.2.1 In

The in principle restricts the number and label of the incoming edges of each node. We use it to state the restrictions on incoming edges explained in section 3.

4.2.2 Out

The out principle restricts the number and label of the outgoing edges of each node. We use it to state the restrictions on outgoing edges explained in section 3.

4.2.3 Order

In the LP dag, the order of each mother and its daughters must be compatible with the total order $\prec$ on the set of node and edge labels on the LP dimension (i.e. the natural order on the Gorn addresses in the grammar). The daughters must be ordered with respect to the mother and its sisters according to their edge label. The mother must be ordered with respect to its daughters according to its node label.

For our example TAG grammar, we define $\prec$ as follows:

$$p0 \prec p1 \prec p2 \prec p21 \prec p22 \tag{11}$$

4.2.4 Projectivity

In the LP dag, we assign to each node an integer denoting its position in the input. We require that for each node, the set of nodes equal or below it must be contiguous.

4.2.5 Copy and Paste

We equip the lexical entries encoding TAG elementary trees with the function $copy : L \rightarrow 2^L$ mapping LP labels L (i.e. Gorn addresses) to sets of LP labels. The function maps each address to the set of addresses of the nodes below it in the corresponding TAG elementary tree. Since substitution nodes do not have any nodes below themselves, the *copy* function only yields non-empty sets for adjunction nodes.

In addition, we equip the lexical entries with the set $paste : 2^L$ of LP labels. This set is empty for initial trees, and is a singleton set containing the address of the foot node for auxiliary trees.

The intuition behind *copy* and *paste* is to view TAG adjunction such that the set of nodes below the adjunction node is first *copied* and then *pasted* at the foot node of the adjoining tree. In the LP dag, we model this by *copying* all edges in the adjoined tree which have edge labels corresponding to addresses below the adjunction node, and then *pasting* them as foot node daughters of the adjoining tree.

4.3 ID and LP dimensions

4.3.1 Climbing

We stipulate that the ID tree must be flatter than the LP dag.

4.3.2 Linking

For all LP daughters (except copy and paste edges), we stipulate a mapping from their node addresses (i.e. their incoming LP edge label) to the corresponding TAG node labels in the respective elementary tree (i.e. their incoming ID edge label).

4.3.3 Co-immediate dominance

For all LP edges (except copy and paste edges), we stipulate that the LP daughter equals the ID daughter.

5 Proof sketch

We present the proof sketch in three steps, depicted in Figure 5, Figure 6, and Figure 7 respectively. We want to prove that our encoding recovers the correct order of anchors after each 1) substitution and 2) adjunction step in the derivation.

In step 1, we substitute the two initial trees α_2 with anchor **A2** and α_3 with anchor **A3** into the initial tree α_1 with anchor **A1**. α_1 additionally has an adjunction node π which is not yet relevant at this point. After substitution, the correct order of anchors is $\mathbf{A1} < \mathbf{A2} < \mathbf{A3}$. In our encoding, we recover this order lexically. By the lexicon, we know that the anchor is in a position to the left of the two substitution nodes, and that α_2 is substituted into a substitution nodes preceding the node into which α_3 is substituted.

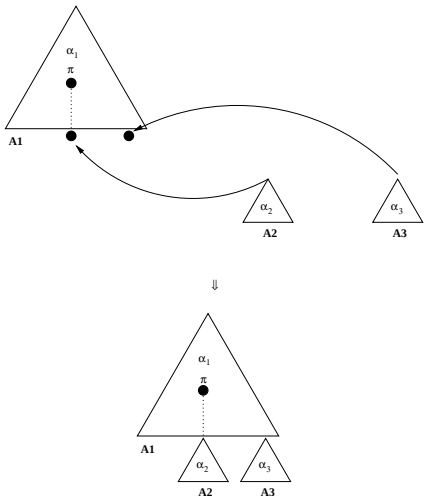


Figure 5: Proof sketch, step 1

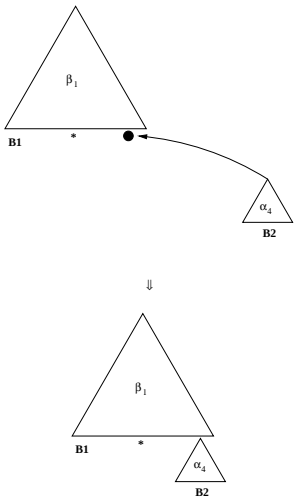


Figure 6: Proof sketch, step 2

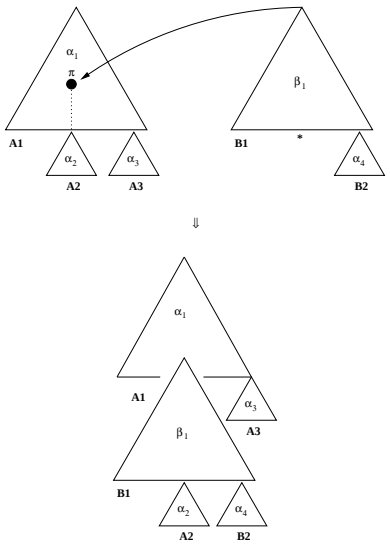


Figure 7: Proof sketch, step 3

In step 2, we substitute the initial tree α_4 with anchor **B2** into the auxiliary tree β_1 with anchor **B1**. Again, we recover the correct order of anchors lexically, here **B1** $<$ **B2**.

Step 3 is the most interesting one. Here, we adjoin the auxiliary tree β_1 into the initial tree α_1, at adjunction site π. By the lexicon, we can already recover the order **A1** $<$ **B1** $<$ **A2** $<$ **A3** because we know that the adjunction site is between the anchor of α_1 and the substitution site into which α_2 has been substituted. But what is the correct mutual order between **A1** and **B2**, and between **B1**, **A2** and **B2**? At this point, the copy and paste principle enters the stage. By the lexicon, we know the set of nodes below the adjunction site, in this case the substitution site into which α_2 has been substituted. This node (and with it the substituted tree α_2) is *copied*, and then *pasted* at the foot node of the tree which has been adjoined, i.e. β_1. Because the foot node of β_1 follows the anchor **B1**, and precedes the substitution site into which α_4 has been substituted, we already know the mutual order between **B1**, **A2** and **B2**, viz. **B1** $<$ **A2** $<$ **B2**. Now because of projectivity, we know that the three anchors **B1**, **A2** and **B2**, by virtue of being below α_1, must be contiguous. We already know that **A1** precedes **B1**. Now if **A1** precedes **B1**, then it must also precede all elements of the contiguous set $\{\mathbf{B1}, \mathbf{A2}, \mathbf{B2}\}$, and thus we have completely recovered the correct order of anchors for this example, viz. **A1** $<$ **B1** $<$ **A2** $<$ **B2** $<$ **A3**.

6 Conclusion

We presented an encoding of TAG into XDG, using a two-dimensional instance of XDG consisting of an ID tree and an LP dag. The ID tree captures the linguistically relevant aspect of a TAG analysis, viz. the derivation tree. The LP dag is more of a vehicle to recover the correct order of anchors of a derivation. In this respect, it is similar to the TAG derived tree: it is required for parsing, but

it is linguistically irrelevant, especially in the light of an interface to semantics.

The encoding allows us to make use of the XDG parser for TAG parsing. The XDG parser has several features not present in current TAG parsers, e.g. an efficient treatment of lexical ambiguity, the postponing of the enumeration of solutions, and the possibility to employ preferences to guide search.

The encoding can potentially also allow us to equip the existing encoded TAG grammars with the syntax-semantics interface currently developed for XDG.

On the theoretical side, the encoding also provides hints how the grammar formalism of Topological Dependency Grammar (TDG) can be compared to TAG. Both in TDG and in our encoding presented here we make use of two dimensions, the ID and the LP dimension. The most striking differences between TAG and XDG are 1) the models of the LP dimension are dags in the TAG encoding, whereas they are trees in TDG, and 2) the climbing principle of TDG is used in a reversed form in the TAG encoding: In TDG, the LP tree is flatter than the ID tree, whereas in the TAG encoding, the ID tree is flatter than the LP dag. This is interesting because both TDG and TAG can be regarded as extensions of context-free grammar. But whereas TDG goes beyond context-free grammar in that it allows to "relax" word order, TAG does the opposite: it makes word order more rigid.

References

Abeillé, A. & Candito, M.-H. (2000), Ftag: A lexicalized tree-adjoining grammar for French, *in* A. Abeillé & O. Rambow, eds, 'Tree Adjoining Grammars', CSLI, Stanford/USA, pp. 305–330.

Dienes, P., Koller, A. & Kuhlmann, M. (2003), Statistical A* Dependency Parsing, *in* 'Prospects and Advances in the Syntax/Semantics Interface', Nancy/FRA.

Duchier, D. (1999), Axiomatizing dependency parsing using set constraints, *in* '6th Meeting on the Mathematics of Language (MOL 6)', Orlando/USA.

Duchier, D. (2001), Lexicalized syntax and topology for non-projective dependency grammar, *in* 'MOL 8 Proceedings', Helsinki/FIN.

Duchier, D. & Debusmann, R. (2001), Topological dependency trees: A constraint-based account of linear precedence, *in* 'ACL 2001 Proceedings', Toulouse/FRA.

Group, X. R. (2001), A lexicalized tree adjoining grammar for english, Technical Report IRCS-01-03, IRCS, University of Pennsylvania.

Joshi, A. K. (1987), An Introduction to Tree-Adjoining Grammars, *in* A. Manaster-Ramer, ed., 'Mathematics of Language', John Benjamins, Amsterdam/NL, pp. 87–115.