A Formal Model for Plausible Dependencies in Lexicalized Tree Adjoining Grammar

Laura Kallmeyer
Heinrich-Heine-Universität Düsseldorf
Universitätsstr. 1
40225 Düsseldorf, Germany
kallmeyer@phil.uni-duesseldorf.de

Marco Kuhlmann
Uppsala University
Box 635
751 26 Uppsala, Sweden
marco.kuhlmann@lingfil.uu.se

Abstract

Several authors have pointed out that the correspondence between LTAG derivation trees and dependency structures is not as direct as it may seem at first glance, and various proposals have been made to overcome this divergence. In this paper we propose to view the correspondence between derivation trees and dependency structures as a tree transformation during which the direction of some of the original edges is reversed. We show that, under this transformation, LTAG is able to induce both ill-nested dependency trees and dependency trees with gap-degree greater than 1, which is not possible under the direct reading of derivation trees as dependency trees.

1 Introduction

In lexicalized tree adjoining grammar (LTAG), the operations of substitution and adjunction establish an asymmetric relation between two lexical items: The elementary tree for one item is substituted or adjoined into the elementary tree for another one. In many cases, this relation can be interpreted as a relation of syntactic dependency (complementation or adjunction), and it is natural then to try to interpret LTAG derivations as dependency trees. However, as several authors have pointed out, the correspondence between derivation trees and dependency structures is not as direct as it may seem at first glance (Rambow et al., 1995; Candito and Kahane, 1998; Frank and van Genabith, 2001). Examples of mismatches are in particular those where predicates adjoin into their arguments, an analysis that is chosen whenever an argument allows for long extractions. In these cases, the edge in the derivation tree and the syntactic dependency have opposite directions.

Different strategies have been adopted to obtain linguistically plausible dependency structures using LTAG. Some proposals adopt variants of the formalism with different derivation operations (Rambow et al., 2001; Chen-Main and Joshi, 2012); others retrieve the missing dependencies from the derivation tree and the derived tree (Kallmeyer, 2002; Gardent and Kallmeyer, 2003). In this paper we follow the second line of work in that we take a two-step approach: To get a dependency tree, we first construct a derivation tree, and then obtain the dependencies in a postprocessing step. However, in contrast to previous work we retain the property that dependencies should form a tree structure: We do not regard the missing dependencies as additions to the derivation tree, but view the correspondence between derivation trees and dependency structures as a tree-to-tree mapping. The crucial feature of this mapping is that it can reverse some of the edges in the derivation tree. In particular it can 'correct' the directions of predicate–argument adjunctions.

The paper is structured as follows. We start by reviewing the divergence between derivation trees and dependency trees in Section 2. In Section 3 we present the basic ideas behind our transformation. A formalization of this transformation is given in Section 4. In Section 5 we examine the structural properties of the dependency trees that can be induced using our transformation. We propose in particular analyses for some examples of ill-nested dependencies and of dependencies of gap degree > 1. Under the direct interpretation, LTAG induces the class of well-nested dependency trees with gap-degree at most 1 (Bodirsky et

Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+11), pages 108–116,
Paris, September 2012.

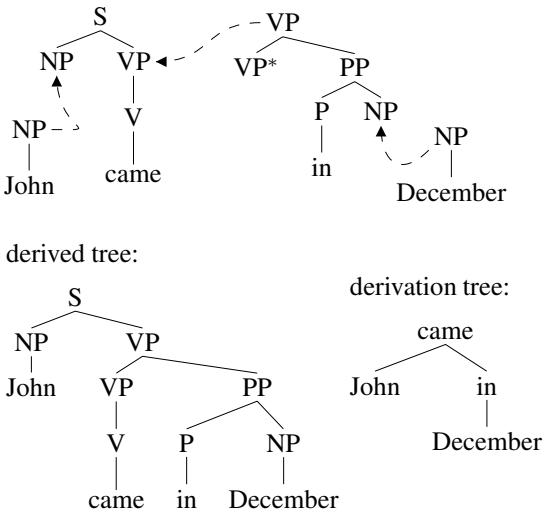

Figure 1: A sample derivation

al., 2005). With our transformation, both ill-nested dependency trees and dependency trees with gap-degree > 1 can be induced. Section 6 discusses related work and Section 7 concludes.

2 Derivations and Dependencies

A lexicalized tree adjoining grammar (LTAG, Joshi and Schabes (1997)) consists of a finite set of *elementary trees*, each of which has at least one leaf with a lexical item. Starting from the elementary trees, larger trees are derived by the operations of *substitution* (which replaces a leaf with a new tree) and *adjunction* (which replaces an internal node with a new tree). In case of an adjunction, the tree being adjoined has exactly one leaf that is marked as the *foot node*. Such a tree is called an *auxiliary* tree. To license its adjunction to a node n, the root and foot node must have the same label as n. When adjoining an auxiliary tree at a node n, in the resulting tree, the subtree with root n from the old tree is attached to the foot node of the auxiliary tree. Non-auxiliary elementary trees are called *initial* trees. A complete derivation starts with an initial tree and produces a derived tree in which all leaves have terminal labels. The history of a derivation is captured in its *derivation tree*. The nodes of this tree correspond to the elementary trees that have been used in the derivation; the edges correspond to the operations performed. Fig. 1 shows a sample derivation.

As can be seen from Fig. 1, in many cases the LTAG derivation tree corresponds to a dependency structure. However, the correspondence between derivation trees and dependency structures is not always a direct one. Examples of mismatches are in particular those where a) an adjunction of a predicate into one of its arguments occurs and b) furthermore, a higher predicate adjoins into the same argument (Rambow et al., 1995). Consider the following example:

(1) John Bill claims Mary seems to love.

A derivation of this sentence, together with the corresponding derivation tree and a dependency tree, is shown in Fig. 2. Here, we have a long-distance topicalization of one of the arguments of *love*. In order to account for this in a satisfying way, in particular without violating any of the linguistic assumptions underlying the form of LTAG elementary trees, one has to realize the substitution node for *John* in the elementary tree for *to_love* while making sure that *claim* can end up in between. Therefore, standardly, *claim* adjoins to an S node in the *to_love* tree. Concerning raising verbs such as *seems*, the standard analysis is to adjoin them to the VP node of the sentence to which they contribute finiteness and tense. In this example, *seems* adjoins to the VP node of *to_love*.

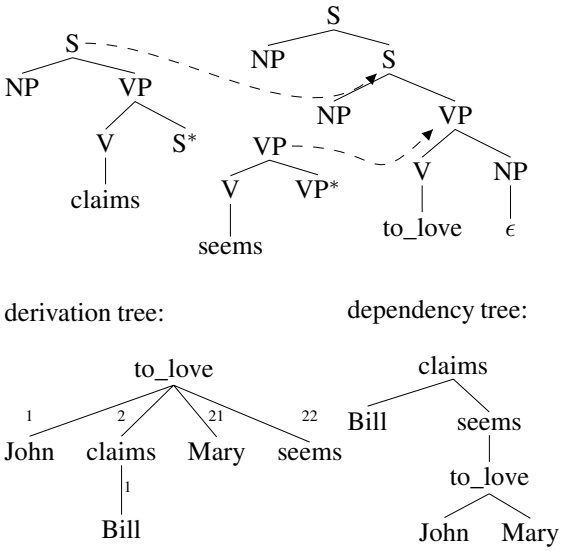

Figure 2: Derivation for (1), together with the corresponding derivation tree and a dependency tree

3 From Derivation to Dependency Trees

Let us inspect the derivation and dependency trees in Fig. 2 more closely in order to understand what the difference is. Concerning substitution nodes, the two trees show the same predicate–argument

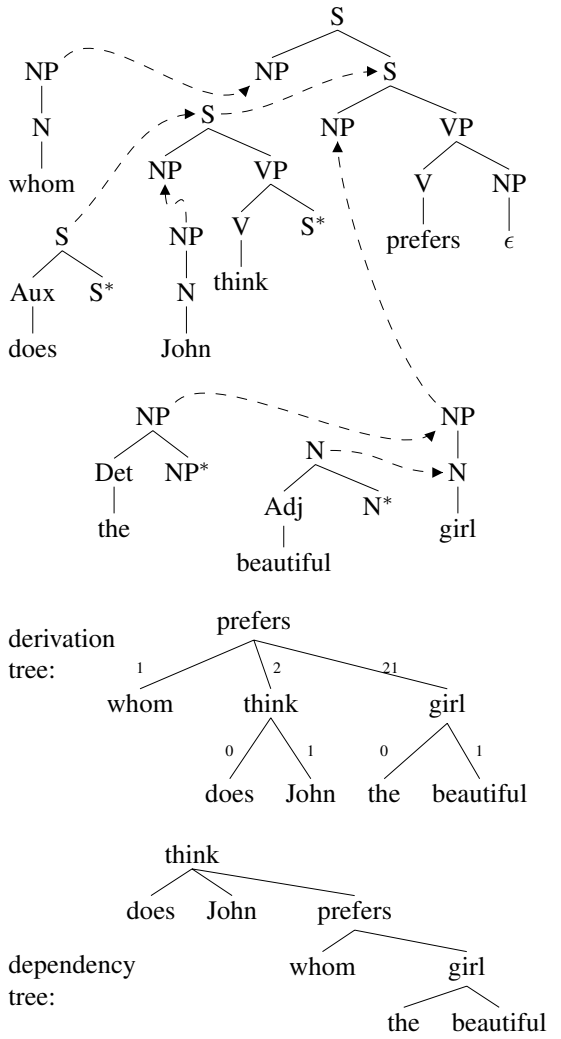

Figure 3: Derivation for (2)

dependencies: *John* and *Mary* are daughters of *to_love*, and *Bill* is a daughter of *claims*. The edge between *seems* and *to_love* has the opposite direction in the derivation tree. This is due to *seems* adjoining into the complement clause that it embeds, which is separated into two discontinuous parts. So, as a first attempt to characterize the relation between LTAG derivation trees and dependency trees, we can say that we should reverse any derivation tree edges originating from complement-taking adjunctions.[1] More precisely, we should replace every complement-adjoining edge $\gamma \rightarrow \gamma'$ with the reverse edge, and redirect the previous incoming edge of γ (if it existed) into γ'.

This transformation works in many cases. As an example consider the following:

(2) Whom does John think the beautiful girl prefers?

The derivation of (2) is given in Fig. 3. There are different types of adjunctions: The adjunction of *think* into *prefers* is a complement-taking adjunction, whose edge gets reversed. All the other adjunctions (of *does*, *the*, and *beautiful*) are not complement-taking, and therefore their edges remain unchanged. As a result, we obtain the desired dependency tree after the tree transformation.

Now let us go back to Fig. 2. Here we have the additional complication that there are *two* complement-taking adjunctions that target the *to_love* tree, and that we get different dependency trees depending on which of the edges we reverse first. A look at the desired tree tells us that *claim* should dominate *seems*. Our hypothesis is that this is related to the fact that *claim* adjoins higher on the head projection line in the *to_love* tree than *seems*.[2] The address of the verbal head (i.e., the anchor) is 221, so the verbal projection line is $\varepsilon, 2, 22, 221$. The edge reversals proceed from higher complement-taking adjunctions to lower ones, i.e., in Fig. 4, first the adjunction of *claim* is reversed and then the adjunction of *seems*. More generally, for our transformation to be deterministic, we need to assume a total order on complement-taking adjunctions on the head projection line. Figure 4 shows the two steps of the transformation of the derivation tree for (1) that yields the desired dependency tree.

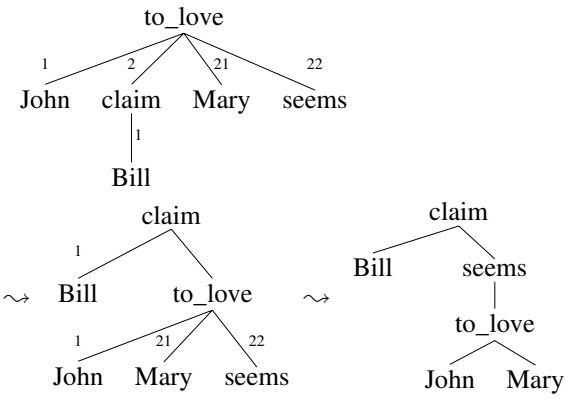

Figure 4: Transformation of the derivation tree for (1)

[1]The observation that complement-taking adjunction edges need to be reversed in order to retrieve the correct dependency goes back to Rambow and Joshi (1994).

[2]Kallmeyer and Romero (2008) use specific features on the head projection line of verbs (the *verbal spine* in their terminology) in order to retrieve the missing dependency links.

4 Formal Version of the Transformation

In this section we give a formal account of the transformation sketched above.

4.1 Derivation Trees as Terms

We represent derivation trees as *terms*, formal expressions over a signature of operation symbols (Vijay-Shanker et al., 1987). For example, we represent the derivation in Fig. 2 by the term t_0:

$$t_0 = \text{to\_love}(\text{John}, \text{claims}(\text{Bill}), \text{Mary}, \text{seems})$$

Here 'to_love' is an operation symbol with rank 4; this corresponds to the fact that during the derivation, four trees are substituted/adjoined into the elementary tree for *to_love*. We will use the term t_0 as a running example in this section.

Note that in order to capture all information contained in a derivation tree, each operation symbol needs to encapsulate not only an elementary tree γ but also the specific addresses at which trees were substituted/adjoined into γ. Since adjunctions may be optional, there will in general be several (albeit a bounded number) of operation symbols per elementary tree, and in particular several different versions of the symbol 'to_love'. To simplify our notation, we ignore this aspect of the term representation in this paper.

4.2 Yield Functions

Each derivation tree t encodes the derivation of a derived tree γ. This can be formalized by interpreting the operation symbols in t as operations on derived trees: The symbol to_love for example can be interpreted as a function that takes four derived trees as arguments and returns the tree obtained by substituting/adjoining these trees at the specified nodes of the elementary tree for *to_love*. To compute the derived tree corresponding to t_0, for example, one applies the operation corresponding to to_love to the derived trees corresponding to the subderivations John, claims(Bill), Mary and seems, respectively.

Let the *yield* of a derived tree γ be defined as follows. If γ is a derived initial tree, then its yield is the one-component tuple $\langle w \rangle$, where w is the string consisting of the symbols at the frontier of γ. If γ is a derived auxiliary tree, then its yield is the pair $\langle w_1, w_2 \rangle$, where w_1 and w_2 are the strings corresponding to the parts of the frontier of γ to the left and to the right of the foot node, respectively. The yield of the derived tree corresponding to a derivation tree t can be obtained by associating with each operation symbol in t a *yield function* (Weir, 1988). To illustrate the idea, suppose that the yields of the four subterms of t_0 are given as $\langle \textit{John} \rangle$, $\langle \textit{Bill claims}, \varepsilon \rangle$, $\langle \textit{Mary} \rangle$, and $\langle \textit{seems}, \varepsilon \rangle$, respectively. Then the full yield (1) is obtained by assigning the following yield function to 'to_love':

$$\begin{aligned}\text{to\_love}(\langle x_{11} \rangle, \langle x_{21}, x_{22} \rangle, \langle x_{31} \rangle, \langle x_{41}, x_{42} \rangle) = \\ \langle x_{11}\, x_{21}\, x_{31}\, x_{41}\, \textit{to\_love}\, x_{42}\, x_{22} \rangle\end{aligned}$$

In the context of our transformation, yields may consist of more than two strings, so yield functions will be operations on arbitrary (finite) tuples of strings. We only require that they are non-copying and non-deleting, as in LCFRSs (Weir, 1988). This means that each yield function f can be defined by an equation of the form

$$f(\vec{x}_1, \ldots, \vec{x}_m) = \langle \alpha_1, \ldots, \alpha_k \rangle$$

where the $\vec{x}_i$ are vectors of variables and $\alpha_1 \cdots \alpha_k$ is a string over these variables and symbols from the yield alphabet in which each variable occurs exactly once. We adopt the convention that the variables in the vector $\vec{x}_i$ should be named $x_{i1}, \ldots, x_{ik_i}$. In this case, the defining equation of a yield function is uniquely determined by the tuple on its right-hand side. We call this tuple the *template* of f, and use it as a unique name for f. This means that we can talk about e.g. the standard binary concatenation function as 'the yield function $\langle x_{11}\, x_{21} \rangle$'.

The yield function associated with an operation symbol in a derivation tree can be extracted in a systematic way; see Boullier (1999) for a procedure that performs this extraction in the formalism of range concatenation grammars.

4.3 Direct Interpretation

The direct interpretation of a derivation tree as a dependency tree can be formalized by interpreting symbols as operations on dependency trees (Kuhlmann, 2013). Continuing our running example, suppose that for each subtree of t_0 we are not only given a string or a pair of strings as before, but also a corresponding dependency tree. Then the symbol to_love has a straightforward reading as constructing a new dependency tree for the full sentence (1): Preserve all the old dependencies,

and add new edges from *to_love* to the roots of the dependency trees associated with the subterms.

4.4 Transformation

We illustrate the formal transformation by means of our running example. In order to convey the intuitive idea, we first present the transformation as an iterative procedure, in much the same way as in Section 3. Then, in a second step, we show how the transformation can be carried out in a single top–down traversal of the initial derivation tree.

Starting from the tree t_0, we will reverse some edges and change some operation symbols to obtain the modified tree

$$t_1 = \text{claims}'(\text{seems}'(\text{to\_love}'(\text{John}, \text{Mary})), \text{Bill})$$

When interpreting this transformed tree as outlined in Section 4.3, we will obtain the plausible dependency tree shown in Fig. 2.

In a first step, we want to reverse the direction of the edge from *to_love* to *claims*, so the transformed derivation tree should have the form

$$t_0' = \text{claims}'(\text{to\_love}''(\text{John}, \text{Mary}, \text{seems}), \text{Bill})$$

However, the yield of t_0 and t_0' should be the same. To achieve this, we change the yield functions of to_love and claims from

$$\begin{aligned} \text{to\_love} &\rightsquigarrow \langle x_{11}\, x_{21}\, x_{31}\, x_{41}\, \textit{to\_love}\, x_{42}\, x_{22} \rangle \\ \text{claims} &\rightsquigarrow \langle x_{11}\, \textit{claims}, \varepsilon \rangle \end{aligned}$$

in the source derivation tree t_0 to

$$\begin{aligned} \text{claims}' &\rightsquigarrow \langle x_{11}\, x_{21}\, \textit{claims}\, x_{12}\, x_{13} \rangle \\ \text{to\_love}'' &\rightsquigarrow \langle x_{11}, x_{21}\, x_{31}\, \textit{to\_love}\, x_{32}, \varepsilon \rangle \end{aligned}$$

in the target derivation tree t_0'.

The idea is illustrated in Fig. 5, which shows the schematic structure of the yield of a derived tree that results from the adjunction of an auxiliary tree β_2 (grey part) into an auxiliary tree β_1 (white parts). In the term representation, the yield functions associated with β_1 and β_2 take the forms

$$f_1 = \langle v_1\, x_1\, v_2, v_3\, x_2\, v_4 \rangle \quad \text{and} \quad f_2 = \langle w_1, w_2 \rangle,$$

respectively, where the variables x_1, x_2 in f_1 are placeholders for the two components of the tuple returned by f_2. When we now reverse the edge between β_1 and β_2, but want the resulting term to have the same yield as before, then we need to change the yield functions into

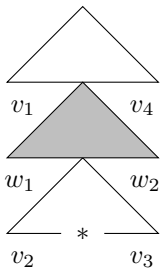

Figure 5: Schematic structure of the yields involved in an adjunction

$$f_2' = \langle x_1\, w_1\, x_2, x_3\, w_2\, x_4 \rangle \quad f_1' = \langle v_1, v_2, v_3, v_4 \rangle.$$

In the second step, we want to reverse the direction of the edge from *to_love* to *seems* and obtain

$$t_1 = \text{claims}(\text{seems}(\text{to\_love}(\text{John}, \text{Mary})), \text{Bill})$$

as above. Again, the yield of t_0' and t_1 should be the same. We therefore change the yield functions of to_love and seems from

$$\begin{aligned} \text{to\_love}'' &\rightsquigarrow \langle x_{11}, x_{21}\, x_{31}\, \textit{to\_love}\, x_{32}, \varepsilon \rangle \\ \text{seems} &\rightsquigarrow \langle \textit{seems}, \varepsilon \rangle \end{aligned}$$

in the source derivation tree t_1 to

$$\begin{aligned} \text{seems}' &\rightsquigarrow \langle x_{11}, x_{12}\, \textit{seems}\, x_{13}\, x_{14}, x_{15} \rangle \\ \text{to\_love}' &\rightsquigarrow \langle x_{11}, x_{21}, \textit{to\_love}, \varepsilon, \varepsilon \rangle \end{aligned}$$

in the target derivation tree t_1. Note that the yield of *to_love*$''$ no longer has the schematic structure of Fig. 5, but consists of *three* discontinuous segments (even though the third segment is empty). More generally, each step of our transformation may increase the *fan-out* of the yield functions that are involved.

4.5 Macro Tree Transducers

To implement our transformation in a single pass over the initial derivation tree, we use a *macro tree transducer* (Engelfriet and Vogler, 1985). Macro tree transducers extend standard top–down tree transducers by the ability to pass the output of the translation of a subtree to the translation of another subtree as an argument. We illustrate how we take advantage of this ability by means of the derivation tree t_0. To obtain the tree t_1, we apply the following rule to t_0:

$$\begin{aligned} &\langle q_0, \text{to\_love}(x_1, x_2, x_3, x_4) \rangle \rightarrow \\ &\langle q_2, x_2 \rangle (\langle q_4, x_4 \rangle (\text{to\_love}'(\langle q_1, x_1 \rangle, \langle q_3, x_3 \rangle))) \end{aligned}$$

The informal reading of this rule is: 'To translate an input tree of the form to_love(t_1, t_2, t_3, t_4): translate the subtrees t_1 and t_3 (corresponding to *John* and *Mary*); attach the outputs of these translations as arguments of the modified symbol to_love$'$; attach the resulting tree to the output of the translation of t_4 (*seems*); and attach the tree resulting from that to the output of the translation of t_2 (*claims*). The q_i are states that can be used to communicate a limited amount of contextual information to the translations of the subtrees. In our case, they are used to transport information about how to modify the yield functions. Each rule of the macro tree transducer encapsulates the full set of modifications to the yield functions that are necessary to simulate the reversals of the complement-taking adjunctions.

In the macro tree transducer for our running example, the rules for the translations of *seems* and *claims* have access to a special variable y that will be instantiated with the output of the translation of the subtrees for *to_love* and *seems*, respectively:

$$\langle q_4, \text{seems}\rangle(y) \rightarrow \text{seems}'(y)$$
$$\langle q_2, \text{claims}(x_1)\rangle(y) \rightarrow \text{claims}'(y, \langle q_{21}, x_1\rangle)$$

These rules produce the following output trees:

$$\textit{seems} \rightsquigarrow \text{seems}'(\text{to\_love}'(\text{john}, \text{mary}))$$
$$\textit{claims} \rightsquigarrow \text{claims}'(\text{seems}'(\cdots), \text{bill})$$

Concerning the complexity of parsing, our transformation is linear in the size of the derivation tree. Consequently, parsing (i.e., obtaining derived, derivation, and dependency tree for a given input) is still polynomial both in the size of the input string and in the size of the grammar, as in the case of standard TAG. This is a difference compared to tree-local MCTAG where, due to the fact that adjunctions at different nodes of an elementary tree are no longer completely independent from each other, the universal recognition problem is NP-complete (Søgaard et al., 2007).

5 Structural Properties

In this section, we reconsider some examples involving ill-nested dependency structures and dependency structures with gap-degree > 1 that have been argued to be problematic for TAG. If we assume that LTAG derivation trees are dependency trees, TAG is limited to well-nested dependency trees of gap-degree ≤ 1 (Bodirsky et al., 2005). We will see that if the transformation described above is applied to a TAG derivation tree, then this limit is no longer given.

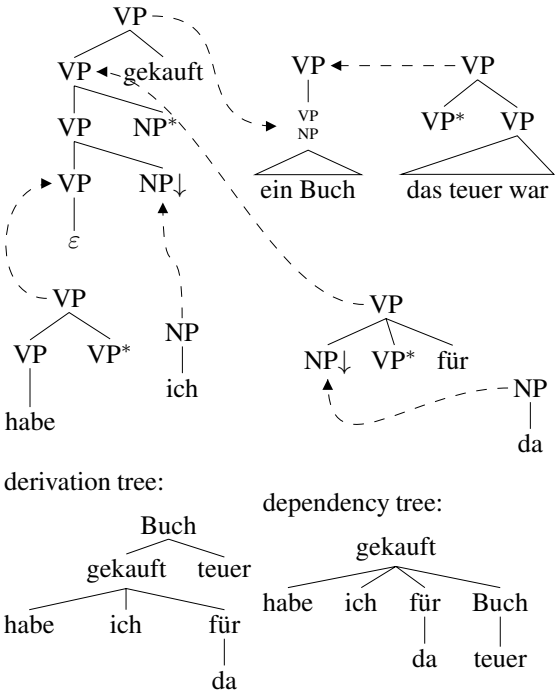

Figure 6: LTAG derivation for (3)

5.1 Ill-Nestedness

Consider the following ill-nested dependency structures in German, both taken from Chen-Main and Joshi (2012). (3) combines a *dafür*-split with an NP containing an extraposed relative clause. (4) combines a split quantifier with an NP having an extraposed relative clause.

(3) Da habe ich ein Buch für gekauft das
that have I a book for bought which
teuer war
expensive was
'for that I bought a book which was expensive'

(4) Bücher hat der Student drei gekauft
books has the student three bought
der am meisten Geld hatte
who the most money had
'the student with the most money bought three books'

Assuming that adjunctions to complements have to be reversed in order to obtain the correct dependencies, we can analyze (3) with the TAG derivation from Fig. 6. The adjunction of *gekauft* to *Buch* then has to be reversed. As a result, we

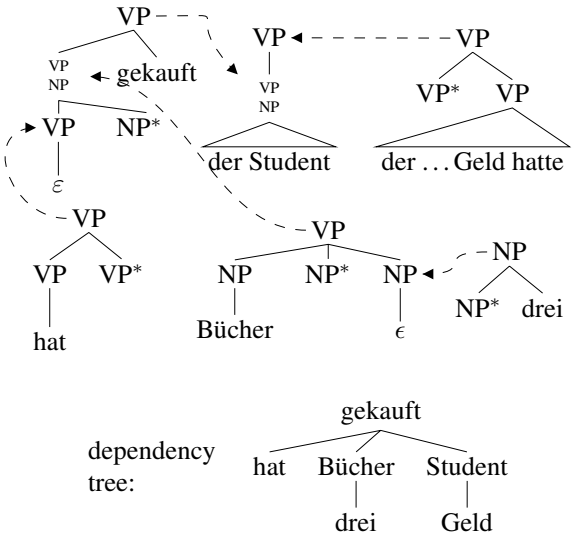

Figure 7: Derivation for (4)

obtain the dependency tree in the lower part of Fig. 6. Note that we assume a feature structure based TAG where the CAT (= category) feature can be different in the top and bottom structures of a node, requiring an adjunction. In Fig. 6 the *Buch* tree requires the adjunction of a VP tree that takes an NP-complement.[3]

The second example for ill-nested dependencies, (4), is slightly more complicated since both split constituents are arguments of the same verb. In order to deal with this, we need to adjoin one of the arguments. This is a major change to standard TAG analyses since there is no argument slot in the sense of substitution or foot node for this argument. The selection of this argument has to be done via the features, namely via the CAT feature of the node where the argument adjoins. The derivation is shown in Fig. 7. Here, the only complement taking adjunction is the adjunction of *gekauft* into *der Student*; the corresponding edge is reversed by our transformation. This yields the correct dependency tree.

5.2 Gap-Degree > 1

Now let us move to examples of gap-degree greater than 1, such as (5). The analysis of this sentence involves an NP that is split into three non-adjacent parts, *was für Bücher von Chomsky die spannend sind*. This is again an example from Chen-Main and Joshi (2012).

[3]With this feature-based modeling of adjunction constraints, the requirement that root and foot node have the same labels in auxiliary trees is no longer assumed.

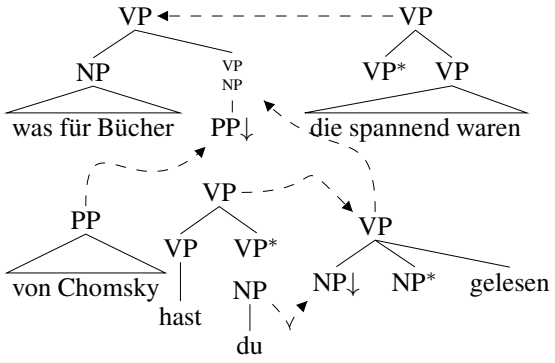

Figure 8: LTAG derivation for (5)

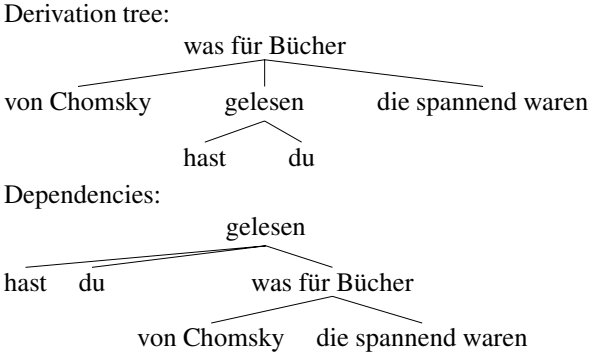

Figure 9: Derivation tree and dependencies for (5)

(5) Was für Bücher hast du von Chomsky
what type of books have you by Chomsky
gelesen die spannend sind
read which exciting are
'what type of books by Chomsky have you read that are exciting'

This example could be analyzed as in Fig. 8. Again, we assume that an adjunction into a complement (here *gelesen* adjoins into its object *Bücher*) is an adjunction that needs to be reversed. As a result, we obtain the derivation and dependency structures in Fig. 9.[4]

5.3 Generalization

From these examples we have already seen that, under the new interpretation of derivation trees as dependency structures, TAG can generate ill-nested dependencies and dependencies of gap degree > 1. A question is whether the gap-degree is limited at all. And here the answer is no. Or rather, there is a grammar-dependent limit on the maximal number of gaps a derivation can generate.

[4]Note that this example involves a complement-taking adjunction that does not take place above the lexical anchor. We leave the investigation of possible nodes for complement-taking adjunctions for future research.

As we have seen, gaps arise from complement-taking adjunctions where each of these adjunctions can introduce two new gaps (to the left and right of the spine of the auxiliary tree). Furthermore, the number of such adjunctions is limited by the maximal number of internal nodes per elementary tree. In addition to this, the tree itself can be a non-complement taking auxiliary tree that contains a gap below its foot node. Consequently, the following holds under our transformation:

Claim. Let G be an LTAG, and let k be the maximal number of internal nodes in a tree of G. Then the gap-degree of the dependency trees induced by G is upper-bounded by $2(k-2)+1$.

This is actually surprising, given that previous assumptions were that LTAG is too limited to generate the dependency structures needed for natural languages (Kuhlmann, 2010). As shown in this paper, instead of moving to a TAG variant different from standard LTAG, one can also use standard LTAG and assume the transformation described in this paper as the operation that induces the underlying dependency structure. In this sense, this paper opens a different perspective on LTAG, showing that the claim about the limitations of LTAG concerning dependency trees is not valid if the induction of dependencies is taken to be an operation on the derivation tree that is different from just the identity.

6 Related Work

The mismatch between TAG derivation trees and dependency structures has been known for a long time, and several solutions have been proposed. These fall, roughly, into two classes. On the one hand, several authors have proposed to use a variant of TAG that yields different derivation trees than standard TAG. On the other hand, some approaches keep standard TAG while exploring ways to obtain the missing links from the derivation tree. Our approach falls into the second class.

Concerning the first class, one of the earliest proposals was D-Tree Substitution Grammar (Rambow et al., 2001). The idea is to use tree descriptions instead of trees. These tree descriptions contain dominance links whenever different parts of an elementary tree can be split. Arguments are added by substitution but an 'extracted' part of an argument, linked to the lower part by a dominance link, can be separated from this lower part and end up much higher. This gives more flexibility concerning the modeling of discontinuities and non-projective dependencies.

Another LTAG variant that has been discussed a lot recently in the context of the 'missing link problem' is tree-local MCTAG with flexible composition (Joshi et al., 2007; Chen-Main and Joshi, 2012), formalized by the notion of delayed tree-locality (Chiang and Scheffler, 2008). The idea is roughly to perform the reversal of complement-taking adjunctions not on the derivation tree, but already during derivation. More precisely, instead of considering such an operation as a standard adjunction, it is considered as a wrapping operation directed from the adjunction site to the auxiliary tree. Consider for instance the derivation in Fig. 3. If we take the adjunction of the *think* tree into the *prefer* tree to be a wrapping of *prefer*, split at its internal S node, around the *think* tree. If this is reflected in the derivation tree by an edge from *prefer* to *think*, then one obtains the lower tree in Fig. 3 as a derivation tree. Chen-Main and Joshi (2012) provide analyses for the examples from Section 6 using tree-local MCTAG with flexible composition. In contrast to their approach, in our proposal flexible composition is replaced by a transformation on the derivation trees. As a result, for the construction of the derivation tree, we keep standard TAG and we can still use its parsing techniques. The choice to remain with standard TAG however requires some relaxation of the predicate argument cooccurrence principle since, as we have seen in with the analysis of (4), we sometimes have to adjoin arguments and express their selection via features of an internal node.

Concerning the use of TAG with some additional means to retrieve the desired dependencies from the derivation tree, this has been pursued by Kallmeyer (2002), where the derivation tree is explicitly enriched with additional links, and by Gardent and Kallmeyer (2003) and Kallmeyer and Romero (2008), where these links are indirectly constructed via feature percolation. However, these approaches do not provide an explicit transformation from derivation trees to dependency structures.

7 Conclusion

In this paper we have addressed the relation between LTAG derivation trees and linguistically plausible dependency trees. We have formalized

the correspondence between the two as an edge-reversing tree transformation, and shown that, under this transformation, LTAG can induce dependency trees that cannot be induced under the direct interpretation.

We used our approach to analyze some of the problematic examples from the literature. It turned out that, given our transformation, these examples can be treated using only TAG, i.e., without multiple components. This is a nice result since it means that parsing (yielding derived, derivation and dependency tree) is polynomial both in input and grammar size. The latter is not the case for tree-local MCTAG.

In future work we hope to address the question whether there are linguistic phenomena that are not covered by our approach, and to carry out a more systematic and comprehensive comparison of the various proposals that have been put forward to address the 'missing link problem'.

References

Manuel Bodirsky, Marco Kuhlmann, and Mathias Möhl. 2005. Well-nested drawings as models of syntactic structure. In *Proceedings of the 10th Conference on Formal Grammar (FG) and Ninth Meeting on Mathematics of Language (MOL)*, pages 195–203, Edinburgh, UK.

Pierre Boullier. 1999. On TAG parsing. In *Traitement Automatique des Langues Naturelles (TALN)*, pages 75–84, Cargèse, France.

Marie-Hélène Candito and Sylvain Kahane. 1998. Can the TAG derivation tree represent a semantic graph? an answer in the light of Meaning-Text Theory. In *Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks, IRCS Report 98–12*, pages 25–28, University of Pennsylvania, Philadelphia.

Joan Chen-Main and Aravind Joshi. 2012. A dependency perspective on the adequacy of tree local multi-component tree adjoining grammar. *Journal of Logic and Computation Advance Access*, June.

David Chiang and Tatjana Scheffler. 2008. Flexible composition and delayed tree-locality. In *TAG+9 Proceedings of the Ninth International Workshop on Tree-Adjoining Grammar and Related Formalisms (TAG+9)*, pages 17–24, Tübingen, June.

Joost Engelfriet and Heiko Vogler. 1985. Macro tree transducers. *Journal of Computer and System Sciences*, 31(1):71–146.

Anette Frank and Josef van Genabith. 2001. GlueTag. Linear logic based semantics for LTAG – and what it teaches us about LFG and LTAG. In Miriam Butt and Tracy Holloway King, editors, *Proceedings of the LFG01 Conference*, Hong Kong.

Claire Gardent and Laura Kallmeyer. 2003. Semantic Construction in FTAG. In *Proceedings of EACL 2003*, pages 123–130, Budapest.

Aravind K. Joshi and Yves Schabes. 1997. Tree-Adjoining Grammars. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 69–123. Springer.

Aravind K. Joshi, Laura Kallmeyer, and Maribel Romero. 2007. Flexible Composition in LTAG: Quantifier Scope and Inverse Linking. In Reinhard Muskens and Harry Bunt, editors, *Computing Meaning Volume 3*, volume 83 of *Studies in Linguistics and Philosophy*, pages 233–256. Springer.

Laura Kallmeyer and Maribel Romero. 2008. Scope and situation binding in LTAG using semantic unification. *Research on Language and Computation*, 6(1):3–52.

Laura Kallmeyer. 2002. Using an Enriched TAG Derivation Structure as Basis for Semantics. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+6)*, pages 127–136, Venice, May.

Marco Kuhlmann. 2010. *Dependency Structures and Lexicalized Grammars*, volume 6270 of *LNCS*. Springer.

Marco Kuhlmann. 2013. Mildly non-projective dependency grammar. *Computational Linguistics*, 39(2). Just Accepted publication August 22, 2012.

Owen Rambow and Aravind K. Joshi. 1994. A processing model for free word order languages. In Charles J. Clifton, Lyn Frazier, and Keith Rayner, editors, *Perspectives on sentence processing*. L. Erlbaum Associates.

Owen Rambow, K. Vijay-Shanker, and David Weir. 1995. D-Tree Grammars. In *Proceedings of ACL*.

Owen Rambow, K. Vijay-Shanker, and David Weir. 2001. D-Tree Substitution Grammars. *Computational Linguistics*, 27(1):87–121.

Anders Søgaard, Timm Lichte, and Wolfgang Maier. 2007. The complexity of linguistically motivated extensions of tree-adjoining grammar. In *Recent Advances in Natural Language Processing 2007*, Borovets, Bulgaria.

K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 104–111, Stanford, CA, USA.

David J. Weir. 1988. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. Ph.D. thesis, University of Pennsylvania, Philadelphia, USA.