An Alternative Conception of Tree-Adjoining Derivation

Yves Schabes*
Mitsubishi Electric
Research Laboratory

Stuart M. Shieber†
Harvard University

The precise formulation of derivation for tree-adjoining grammars has important ramifications for a wide variety of uses of the formalism, from syntactic analysis to semantic interpretation and statistical language modeling. We argue that the definition of tree-adjoining derivation must be reformulated in order to manifest the proper linguistic dependencies in derivations. The particular proposal is both precisely characterizable through a definition of TAG derivations as equivalence classes of ordered derivation trees, and computationally operational, by virtue of a compilation to linear indexed grammars together with an efficient algorithm for recognition and parsing according to the compiled grammar.

1. Introduction

In a context-free grammar, the derivation of a string in the rewriting sense can be captured in a single canonical tree structure that abstracts all possible derivation orders. As it turns out, this *derivation tree* also corresponds exactly to the hierarchical structure that the derivation imposes on the string, the *derived tree* structure of the string. The formalism of tree-adjoining grammars (TAG), on the other hand, decouples these two notions of derivation tree and derived tree. Intuitively, the derivation tree is a more finely grained structure than the derived tree, and as such can serve as a substrate on which to pursue further analysis of the string. This intuitive possibility is made manifest in several ways. Fine-grained syntactic analysis can be pursued by imposing on the derivation tree further combinatorial constraints, for instance, selective adjoining constraints or equational constraints over feature structures. Statistical analysis can be explored through the specification of derivational probabilities as formalized in stochastic tree-adjoining grammars. Semantic analysis can be overlaid through the synchronous derivations of two TAGs.

All of these methods rely on the derivation tree as the source of the important primitive relationships among trees. The decoupling of derivation trees from derived trees thus makes possible a more flexible ability to pursue these types of analyses. At the same time, the exact definition of derivation becomes of paramount importance. In this paper, we argue that previous definitions of tree-adjoining derivation have not taken full advantage of this decoupling, and are not as appropriate as they might be for the kind of further analysis that tree-adjoining analyses could make possible. In particular, the standard definition of derivation, due to Vijay-Shanker (1987), requires that auxiliary trees be adjoined at distinct nodes in elementary trees. However, in certain cases, especially cases characterized as linguistic modification, it is more appropriate to

* Cambridge, MA 02139
† Division of Applied Sciences, Cambridge, MA 02138

allow multiple adjunctions at a single node.

In this paper, we propose a redefinition of TAG derivation along these lines, whereby multiple auxiliary trees of modification can be adjoined at a single node, whereas only a single auxiliary tree of predication can. The redefinition constitutes a new definition of derivation for TAG that we will refer to as *extended derivation*. In order for such a redefinition to be serviceable, however, it is necessary that it be both precise and operational. In service of the former, we provide a formal definition of extended derivation using a new approach to representing derivations as equivalence classes of ordered derivation trees. With respect to the latter, we provide a method of compilation of TAGs into corresponding linear indexed grammars (LIG), which makes the derivation structure explicit, and show how the generated LIG can drive a parsing algorithm that recovers, either implicitly or explicitly, the extended derivations of the string.

The paper is organized as follows. First, we review Vijay-Shanker's standard definition of TAG derivation, and introduce the motivation for extended derivations. Then, we present the extended notion of derivation and its formal definition. The original compilation of TAGs to LIGs provided by Vijay-Shanker and Weir and our variant for extended derivations are both described. Finally, we discuss a parsing algorithm for TAG that operates by a variant of Earley parsing on the corresponding LIG. The set of extended derivations can subsequently be recovered from the set of Earley items generated by the algorithm. The resultant algorithm is further modified so as to build an explicit derivation tree incrementally as parsing proceeds; this modification, which is a novel result in its own right, allows the parsing algorithm to be used by systems that require incremental processing with respect to tree-adjoining grammars.

2. The Standard Definition of Derivation

To exemplify the distinction between standard and extended derivations, we exhibit the TAG of Figure 1.[1] This grammar derives some simple noun phrases such as "roasted red pepper" and "baked red potato". The former, for instance, is associated with the derived tree in Figure 2(a). The tree can be viewed as being derived in two ways[2]

Dependent: The auxiliary tree β_{ro} is adjoined at the root node (address ϵ)[3] of β_{re}. The resultant tree is adjoined at the N node (address 1) of initial tree α_{pe}. This derivation is depicted as the derivation tree in Figure 3(a).

Independent: The auxiliary trees β_{ro} and β_{re} are adjoined at the N node (address 1) of the initial tree α_{pe}. This derivation is depicted as the derivation tree in Figure 3(b).

In the *independent* derivation, two trees are separately adjoined at one and the same node in the initial tree. In the *dependent* derivation, on the other hand, one auxiliary tree is adjoined to the other, the latter only being adjoined to the initial tree. We will use this

1 Here and elsewhere, we conventionally use the Greek letter α and its subscripted and primed variants for initial trees, β and its variants for auxiliary trees, and γ and its variants for elementary trees in general. The foot node of an auxiliary tree is marked with an asterisk ('*').

2 We ignore here the possibility of another dependent derivation wherein adjunction occurs at the foot node of an auxiliary tree. Because this introduces yet another systematic ambiguity, it is typically disallowed by stipulation in the literature on linguistic analyses using TAGs.

3 The address of a node in a tree is taken to be its Gorn number, that sequence of integers specifying which branches to traverse in order starting from the root of the tree to reach the node. The address of the root of the tree is therefore the empty sequence, notated ϵ. See the appendix for a more complete discussion of notation.

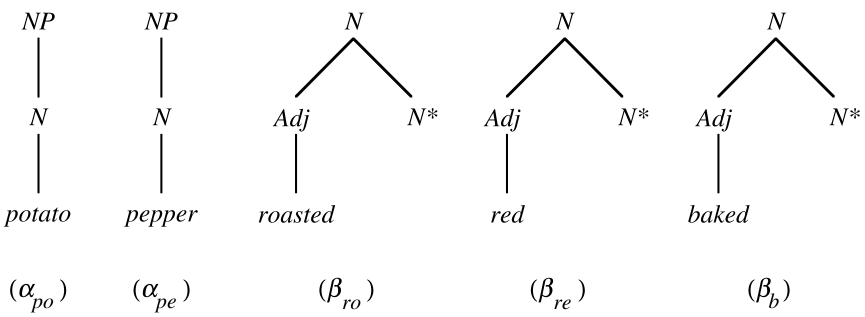

Figure 1
A sample tree-adjoining grammar

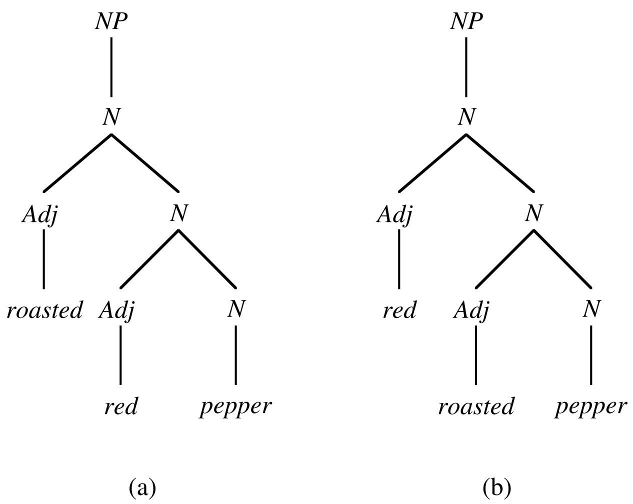

Figure 2
Two trees derived by the grammar of Figure 1

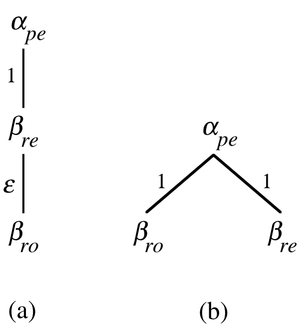

Figure 3
Derivation trees for the derived tree of Figure 2(a) according to the grammar of Figure 1

informal terminology uniformly in the sequel to distinguish the two general topologies of derivation trees.

The *standard* definition of derivation, as codified by Vijay-Shanker, restricts derivations so that *two adjunctions cannot occur at the same node in the same elementary tree*. The dependent notion of derivation (Figure 3(a)) is therefore the only sanctioned derivation for the desired tree in Figure 2(a); the independent derivation (Figure 3(b)) is disallowed. Vijay-Shanker's definition is appropriate because for any independent derivation, there is a dependent derivation of the same derived tree. This can be easily seen in that any adjunction of β_2 at a node at which an adjunction of β_1 occurs could instead be replaced by an adjunction of β_2 at the root of β_1.

The advantage of this standard definition of derivation is that a derivation tree in this normal form unambiguously specifies a derived tree. The independent derivation tree on the other hand is ambiguous as to the derived tree it specifies in that a notion of precedence of the adjunctions at the same node is unspecified, but crucial to the derived tree specified. This follows from the fact that the independent derivation tree is symmetric with respect to the roles of the two auxiliary trees (by inspection), whereas the derived tree is not. By symmetry, therefore, it must be the case that the same independent derivation tree specifies the alternative derived tree in Figure 2(b).

3. Motivation for an Extended Definition of Derivation

In the absence of some further interpretation of the derivation tree nothing hinges on the choice of derivation definition, so that the standard definition disallowing independent derivations is as reasonable as any other. However, tree-adjoining grammars are almost universally extended with augmentations that make the issue apposite. We discuss three such variations here, all of which argue for the use of independent derivations under certain circumstances.[4]

3.1 Adding Adjoining Constraints

Already in very early work on tree-adjoining grammars (Joshi, Levy, and Takahashi, 1975) constraints were allowed to be specified as to whether a particular auxiliary tree may or may not be adjoined at a particular node in a particular tree. The idea is formulated in its modern variant as *selective-adjoining constraints* (Vijay-Shanker and Joshi, 1985). As an application of this capability, we consider the traditional grammatical view that directional adjuncts can be used only with certain verbs.[5] This would account for the felicity distinctions between the following sentences:

(1) a. Brockway walked his Labrador towards the yacht club.
b. # Brockway resembled his Labrador towards the yacht club.

4 The formulation of derivation for tree-adjoining grammars is also of significance for other grammatical formalisms based on weaker forms of adjunction such as lexicalized context-free grammar (Schabes and Waters, 1993a) and its stochastic extension (Schabes and Waters, 1993b), though we do not discuss these arguments here.

5 For instance, Quirk et al. (1985, page 517) remark that "direction adjuncts of both goal and source can normally be used only with verbs of motion". Although the restriction is undoubtedly a semantic one, we will examine the modeling of it in a TAG deriving syntactic trees for two reasons. First, the problematic nature of independent derivation is more easily seen in this way. Second, much of the intuition behind TAG analyses is based on a tight relationship between syntactic and semantic structure. Thus, whatever scheme for semantics is to be used with TAGs will require appropriate derivations to model these data. For example, an analysis of this phenomenon by adjoining constraints on the semantic half of a synchronous TAG would be subject to the identical argument. See Section 3.3.

This could be modeled by disallowing through selective adjoining constraints the adjunction of the elementary tree corresponding to a *towards* adverbial at the VP node of the elementary tree corresponding to the verb *resembles*.[6] However, the restriction applies even with intervening (and otherwise acceptable) adverbials.

(2) a. Brockway walked his Labrador yesterday.
b. Brockway walked his Labrador yesterday towards the yacht club.
(3) a. Brockway resembled his Labrador yesterday.
b. # Brockway resembled his Labrador yesterday towards the yacht club.

Under the standard definition of derivation, there is no direct adjunction in the latter sentence of the *towards* tree into the *resembles* tree. Rather, it is dependently adjoined at the root of the elementary tree that heads the adverbial *yesterday*, the latter directly adjoining into the main verb tree. To restrict both of the ill-formed sentences, then, a restriction must be placed not only on adjoining the goal adverbial in a *resembles* context, but also in the *yesterday* adverbial context. But this constraint is too strong, as it disallows sentence (2b) above as well.

The problem is that the standard derivation does not correctly reflect the syntactic relation between the adverbial modifier and the phrase it modifies when there are multiple modifications in a single clause. In such a case, each of the adverbials independently modifies the verb, and this should be reflected in their independent adjunction at the same point. But this is specifically disallowed in a standard derivation.

Another example along the same lines follows from the requirement that tense as manifested in a verb group be consistent with temporal adjuncts. For instance, consider the following examples:

(4) a. Brockway walked his Labrador yesterday.
b. # Brockway will walk his Labrador yesterday.
(5) a. # Brockway walked his Labrador tomorrow.
b. Brockway will walk his Labrador tomorrow.

Again, the relationship is independent of other intervening adjuncts.

(6) a. Brockway walked his Labrador towards the yacht club yesterday.
b. # Brockway will walk his Labrador towards the yacht club yesterday.
(7) a. # Brockway walked his Labrador towards the yacht club tomorrow.
b. Brockway will walk his Labrador towards the yacht club tomorrow.

It is important to note that these arguments apply specifically to auxiliary trees that correspond to a modification relationship. Auxiliary trees are used in TAG typically for predication relations as well,[7] as in the case of raising and sentential complement constructions.[8] Consider the following sentences. (The brackets mark the leaves of the

6 Whether the adjunction occurs at the VP node or the S node is immaterial to the argument.
7 We use the term 'predication' in its logical sense, that is, for auxiliary trees that serve as logical predicates over the trees into which they adjoin, in contrast to the term's linguistic sub-sense in which the argument of the predicate is a linguistic subject.
8 The distinction between predicative and modifier trees has been proposed previously for purely linguistic reasons by Kroch (1989), who refers to them as complement and athematic trees, respectively. The arguments presented here can be seen as providing further evidence for differentiating the two kinds of auxiliary trees. A precursor to this idea can perhaps be seen in the distinction between repeatable and nonrepeatable adjunction in the formalism of string adjunct grammars, a precursor of TAGs (Joshi, Kosaraju, and Yamada, 1972b, pages 253–254).

pertinent trees to be combined by adjunction in the assumed analysis.)

(8) a. Brockway assumed that Harrison wanted to walk his Labrador.
b. [Brockway assumed that] [Harrison wanted] [to walk his Labrador]
(9) a. Brockway wanted to try to walk his Labrador.
b. [Brockway wanted] [to try] [to walk his Labrador]
(10) a. * Harrison wanted Brockway tried to walk his Labrador.
b. * [Harrison wanted] [Brockway tried] [to walk his Labrador]
(11) a. Harrison wanted to assume that Brockway walked his Labrador.
b. [Harrison wanted] [to assume that] [Brockway walked his Labrador]

Assume (following, for instance, the analysis of Kroch and Joshi (1985)) that the trees associated with the various forms of the verbs *try*, *want*, and *assume* all take sentential complements, certain of which are tensed with overt subjects and others untensed with empty subjects. The auxiliary trees for these verbs specify by adjoining constraints which type of sentential complement they take: *assume* requires tensed complements, *want* and *try* untensed. Under this analysis the auxiliary trees must not be allowed to independently adjoin at the same node. For instance, if trees corresponding to "Harrison wanted" and "Brockway tried" (which both require untensed complements) were both adjoined at the root of the tree for "to walk his Labrador", the selective adjoining constraints would be satisfied, yet the generated sentence (10a) is ungrammatical. Conversely, under independent adjunction, the sentence (11a) would be deemed ungrammatical, although it is in fact grammatical. Thus, the case of predicative trees is entirely unlike that of modifier trees. Here, the standard notion of derivation is exactly what is needed as far as interpretation of adjoining constraints is concerned.

An alternative would be to modify the way in which adjoining constraints are updated upon adjunction. If after adjoining a modifier tree at a node, the adjoining constraints of the original node, rather than those of the root and foot of the modifier tree, are manifest in the corresponding nodes in the derived tree, the adjoining constraints would propagate appropriately to handle the examples above. This alternative leads, however, to a formalism for which derivation trees are no longer context-free, with concomitant difficulties in designing parsing algorithms. Instead, the extended definition of derivation effectively allows use of a Kleene-* in the "grammar" of derivation trees.

Adjoining constraints can also be implemented using feature structure equations (Vijay-Shanker and Joshi, 1988). It is possible that judicious use of such techniques might prevent the particular problems noted here. Such an encoding of a solution requires consideration of constraints that pass among many trees just to limit the cooccurrence of a pair of trees. However, it more closely follows the spirit of TAGs to state such intuitively local limitations locally.

In summary, the interpretation of adjoining constraints in TAG is sensitive to the particular notion of derivation that is used. Therefore, it can be used as a litmus test for an appropriate definition of derivation. As such, it argues for a nonstandard, independent, notion of derivation for modifier auxiliary trees and a standard, dependent, notion for predicative trees.

3.2 Adding Statistical Parameters

In a similar vein, the statistical parameters of a stochastic lexicalized TAG (SLTAG) (Resnik, 1992; Schabes, 1992) specify the probability of adjunction of a given auxiliary tree at a specific node in another tree. This specification may again be interpreted with regard to differing derivations, obviously with differing impact on the resulting probabilities assigned to derivation trees. (In the extreme case, a constraint prohibiting adjoining corresponds to a zero probability in an SLTAG. The relation to the argument in the previous section follows thereby.) Consider a case in which linguistic modification of noun phrases by adjectives is modeled by adjunction of a modifying tree. Under the standard definition of derivation, multiple modifications of a single NP would lead to dependent adjunctions in which a first modifier adjoins at the root of a second. As an example, we consider again the grammar given in Figure 1, that admits of derivations for the strings "baked red potato" and "baked red pepper". Specifying adjunction probabilities on standard derivations, the distinction between the overall probabilities for these two strings depends solely on the adjunction probabilities of β_{re} (the tree for *red*) into α_{po} and α_{pe} (those for *potato* and *pepper*, respectively), as the tree β_b for the word *baked* is adjoined in both cases at the root of β_{re} in both standard derivations. In the extended derivations, on the other hand, both modifying trees are adjoined independently into the noun trees. Thus, the overall probabilities are determined as well by the probabilities of adjunction of the trees for *baked* into the nominal trees. It seems intuitively plausible that the most important relationships to characterize statistically are those between modifier and modified, rather than between two modifiers.[9] In the case at hand, the fact that one typically refers to the process of cooking potatoes as "baking", whereas the appropriate term for the corresponding cooking process applied to peppers is "roasting", would be more determining of the expected overall probabilities.

Note again that the distinction between modifier and predicative trees is important. The standard definition of derivation is entirely appropriate for adjunction probabilities for predicative trees, but not for modifier trees.

3.3 Adding Semantics

Finally, the formation of synchronous TAGs has been proposed to allow use of TAGs in semantic interpretation, natural language generation, and machine translation. In previous work (Shieber and Schabes, 1990), the definition of synchronous TAG derivation is given in a manner that requires multiple adjunctions at a single node. The need for such derivations follows from the fact that synchronous derivations are intended to model semantic relationships. In cases of multiple adjunction of modifier trees at a single node, the appropriate semantic relationships comprise separate modifications rather than cascaded ones, and this is reflected in the definition of synchronous TAG derivation.[10] Because of this, a parser for synchronous TAGs must recover, at least implicitly, the extended derivations of TAG derived trees. Shieber (Forthcoming) provides a more complete discussion of the relationship between synchronous TAGs and the extended definition of derivation with special emphasis on the ramifications for formal expressivity.

Note that the independence of the adjunction of modifiers in the syntax does not imply that semantically there is no precedence or scoping relation between them. As exemplified in Figure 5, the derived tree generated by multiple independent adjunctions at a single node still manifests nesting relationships among the adjoined trees. This fact

9 Intuition is an appropriate guide in the design of the SLTAG framework, as the idea is to set up a linguistically plausible infrastructure on top of which a lexically-based statistical model can be built. In addition, suggestive (though certainly not conclusive) evidence along these lines can be gleaned from corpora analyses. For instance, in a simple experiment in which medium frequency triples of exactly the discussed form "⟨*adjective*⟩ ⟨*adjective*⟩ ⟨*noun*⟩" were examined, the mean mutual information between the first adjective and the noun was found to be larger than that between the two adjectives. The statistical assumptions behind this particular experiment do not allow very robust conclusions to be drawn, and more work is needed along these lines.

10 The importance of the distinction between predicative and modifier trees with respect to how derivations are defined was not appreciated in the earlier work; derivations were taken to be of the independent variety in all cases. In future work, we plan to remedy this flaw.

may be used to advantage in the semantic half of a synchronous tree-adjoining grammar to specify the semantic distinction between, for example, the following two sentences:[11]

(12) a. Brockway ran over his polo mallet twice intentionally.
b. Brockway ran over his polo mallet intentionally twice.

We hope to address this issue in greater detail in future work on synchronous tree-adjoining grammars.

3.4 Desired Properties of Extended Derivations

We have presented several arguments that the standard notion of derivation does not allow for an appropriate specification of dependencies to be captured. An extended notion of derivation is needed that

1. Differentiates predicative and modifier auxiliary trees;
2. Requires dependent derivations for predicative trees;
3. Allows independent derivations for modifier trees; and
4. Unambiguously and nonredundantly specifies a derived tree.

Furthermore, following from considerations of the role of modifier trees in a grammar as essentially optional and freely applicable elements, we would like the following criterion to hold of extended derivations:

5. If a node can be modified at all, it can be modified any number of times, including zero times.

Recall that a derivation tree (as traditionally conceived) is a tree with unordered arcs where each node is labeled by an elementary tree of a TAG and each arc is labeled by a tree address specifying a node in the parent tree. In a standard derivation tree no two sibling arcs can be labeled with the same address. In an extended derivation tree, however, the condition is relaxed: No two sibling arcs *to predicative trees* can be labeled with the same address. Thus, for any given address there can be at most one predicative tree and several modifier trees adjoined at that node. As we have seen, this relaxed definition violates the fourth desideratum above; for instance, the derivation tree in Figure 3(b) ambiguously specifies both derived trees in Figure 2. In the next section, we provide a formal definition of extended derivations that satisfies all of the criteria above.

4. Formal Definition of Extended Derivations

In this section we introduce a new framework for describing TAG derivation trees that allows for a natural expression of both standard and extended derivations, and makes available even more fine-grained restrictions on derivation trees. First, we define ordered derivation trees and show that they unambiguously but redundantly specify derivations.[12] We characterize the redundant trees as those related by a sibling swapping

11 We are indebted to an anonymous reviewer of an earlier version of this paper for raising this issue crisply through examples similar to those given here.
12 Historical precedent for independent derivation and the associated ordered derivation trees can be found in the derivation trees postulated for string adjunct grammars (Joshi, Kosaraju, and Yamada, 1972a, pages

operation. Derivation trees proper are then taken to be the equivalence classes of ordered derivation trees where the equivalence relation is generated by the sibling swapping. By limiting the underlying set of ordered derivation trees in various ways, Vijay-Shanker's definition of derivation tree, a precise form of the extended definition, and many other definitions of derivation can be characterized in this way.

4.1 Ordered Derivation Trees

Ordered derivation trees, like the traditional derivation trees, are trees with nodes labeled by elementary trees where each arc is labeled with an address in the tree for the parent node of the arc. However, the arcs are taken to be ordered with respect to each other.

An ordered derivation tree is well-formed if for each of its arcs, linking parent node labeled γ to child node labeled γ' and itself labeled with address t, the tree γ' is an auxiliary tree that can be adjoined at the node t in the tree γ. (Alternatively, if substitution is allowed, γ' may be an initial tree that can be substituted at the node t in γ. Later definitions ignore this possibility, but are easily generalized.)

We define the function $\mathcal{D}$ from ordered derivation trees to the derived trees they specify, according to the following recursive definition:

$$\mathcal{D}(D) = \begin{cases} \gamma \quad \text{if } D \text{ is a trivial tree of one node labeled with the elementary tree } \gamma \\ \gamma[\mathcal{D}(D_1)/t_1, \mathcal{D}(D_2)/t_2, \ldots, \mathcal{D}(D_k)/t_k] \\ \qquad \text{if } D \text{ is a tree with root node labeled with the elementary tree } \gamma \\ \qquad\quad \text{and with } k \text{ child subtrees } D_1, \ldots, D_k \\ \qquad\quad \text{whose arcs are labeled with addresses } t_1, \ldots, t_k. \end{cases}$$

Here $\gamma[A_1/t_1, \ldots, A_k/t_k]$ specifies the simultaneous adjunction of trees A_1 through A_k at t_1 through t_k, respectively, in γ. It is defined as the iterative adjunction of the A_i in order at their respective addresses, with appropriate updating of the tree addresses of any later adjunction to reflect the effect of earlier adjunctions that occur at addresses dominating the address of the later adjunction.

4.2 Derivation Trees

It is easy to see that the derived tree specified by a given ordered derivation tree is unchanged if adjacent siblings whose arcs are labeled with different tree addresses are swapped. (This is not true of adjacent siblings whose arcs are labeled with the same address.) That is, if $t \neq t'$ then $\gamma[\ldots, A/t, B/t', \ldots] = \gamma[\ldots, B/t', A/t, \ldots]$. A graphical "proof" of this intuitive fact is given in Figure 4. A formal proof, although tedious and unenlightening, is possible as well. We provide it in an appendix, primarily because the definitional aspects of the TAG formulation may be of some interest.

This fact about the swapping of adjacent siblings shows that ordered derivation trees possess an inherent redundancy. The order of adjacent sibling subtrees labeled with different tree addresses is immaterial. Consequently, we can define true *derivation trees* to be the equivalence classes of the base set of ordered derivation trees under the equivalence relation generated by the sibling subtree swapping operation above. This is a well-formed definition by virtue of the proposition argued informally above.

This definition generalizes the traditional definition in not restricting the tree address labels in any way. It therefore satisfies criterion (3) of Section 3.4. Furthermore, by virtue of the explicit quotient with respect to sibling swapping, a derivation tree under this

99–100). In this system, siblings in derivation trees are viewed as totally, not partially, ordered. The systematic ambiguity introduced thereby is eliminated by stipulating that the sibling order be consistent with an arbitrary ordering on adjunction sites.

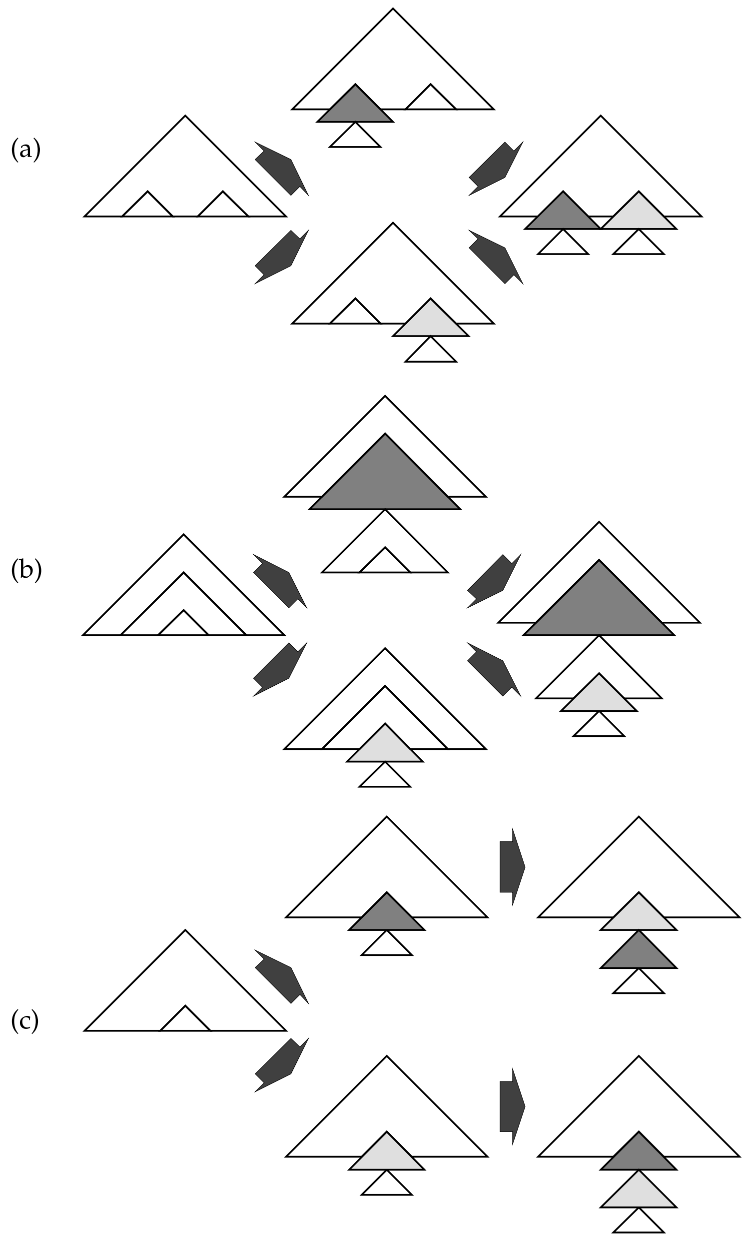

Figure 4
A graphical proof of the irrelevance of adjacent sibling swapping. These diagram show the effect of performing two adjunctions (of auxiliary trees depicted one as dark-shaded and one light-shaded), presumed to be specified by adjacent siblings in an ordered derivation tree. The adjunctions are to occur at two addresses (referred to in this caption as t and t', respectively). The two addresses must be such that either (a) they are distinct but neither dominates the other, (b) t dominates t' (or vice versa), or (c) they are identical. In case (a) the diagram shows that either order of adjunction yields the same derived tree. Adjunction at t and then t' corresponds to the upper arrows, adjunction at t' and then t the lower arrows. Similarly, in case (b), adjunction at t followed by adjunction at an appropriately updated t' yields the same result as adjunction first at t' and then at t. Clearly, adjunctions occurring before these two or after do not affect the interchangeability. Thus, if two adjacent siblings in a derivation tree specify adjunctions at distinct addresses t and t', the adjunctions can occur in either order. Diagram (c) demonstrates that this is not the case when t and t' are the same.

definition unambiguously and nonredundantly specifies a derived tree (criterion 4). It does not, however, differentiate predicative from modifier trees (criterion (1)), nor can it therefore mandate dependent derivations for predicative trees (criterion (2)).

This general approach can, however, be specialized to correspond to several previous definitions of derivation tree. For instance, if we further restrict the base set of ordered derivation trees so that no two siblings are labeled with the same tree address, then the equivalence relation over these ordered derivation trees allows for full reordering of all siblings. Clearly, these equivalence classes are isomorphic to the unordered trees, and we have reconstructed Vijay-Shanker's standard definition of derivation tree.

If we instead restrict ordered derivation trees so that no two siblings corresponding to predicative trees are labeled with the same tree address, then we have reconstructed a version of the extended definition argued for in this paper. Under this restriction, criteria (1) and (2) are satisfied, while maintaining (3) and (4).

By careful selection of other constraints on the base set, other linguistic restrictions might be imposed on derivation trees, still using the same definition of derivation trees as equivalence classes over ordered derivation trees. In the next section, we show that the definition of the previous paragraph should be further restricted to disallow the reordering of predicative and modifier trees. We also describe other potential linguistic applications of the ability to finely control the notion of derivation through the use of ordered derivation trees.

4.3 Further Restrictions on Extended Derivations

The extended definition of derivation tree given in the previous section effectively specifies the output derived tree by adding a partial ordering on sibling arcs that correspond to modifier trees adjoined at the same address. All other arcs are effectively unordered (in the sense that all relative orderings of them exist in the equivalence class).

Assume that in a given tree γ at a particular address t, the k modifier trees $\mu_1, \ldots, \mu_k$ are directly adjoined in that order. Associated with the subtrees rooted at the k elementary auxiliary trees in this derivation are k derived auxiliary trees ($A_1, \ldots, A_k$, respectively). The derived tree specified by this derivation tree, according to the definition of $\mathcal{D}$ given above, would have the derived tree A_1 directly below A_2 and so forth, with A_k at the top. Now suppose that in addition, a predicative tree π is also adjoined at address t. It must be ordered with respect to the μ_i in the derivation tree, and its relative order determines where in the bottom to top order in the derived tree the tree A_π associated with the subderivation rooted at π goes.

The question that we raise here is whether all $k + 1$ possible placements of the tree π relative to the μ_i are linguistically reasonable. We might allow all $k + 1$ orderings (as in the definition of the previous section), or we might restrict them by requiring, say, that the predicative tree always be adjoined before, or perhaps after, any modifier trees at a given address. We emphasize that this is a linguistic question, in the sense that the definition of extended derivation is well-formed whatever decision is made on this question.

Henceforth, we will assume that predicative trees are always adjoined after any modifier trees at the same address, so that they appear above the modifier trees in the derived tree. We call this "outermost predication" because a predicative tree appears wrapped around the outside of the modifier trees adjoined at the same address. (See Figure 5.) If we were to mandate innermost predication, in which a predicative tree is always adjoined before the modifier trees at the same address, the predicative tree would appear within all of the modifier trees, innermost in the derived tree.

Linguistically, the outermost method specifies that if both a predicative tree and a modifier tree are adjoined at a single node, then the predicative tree attaches higher

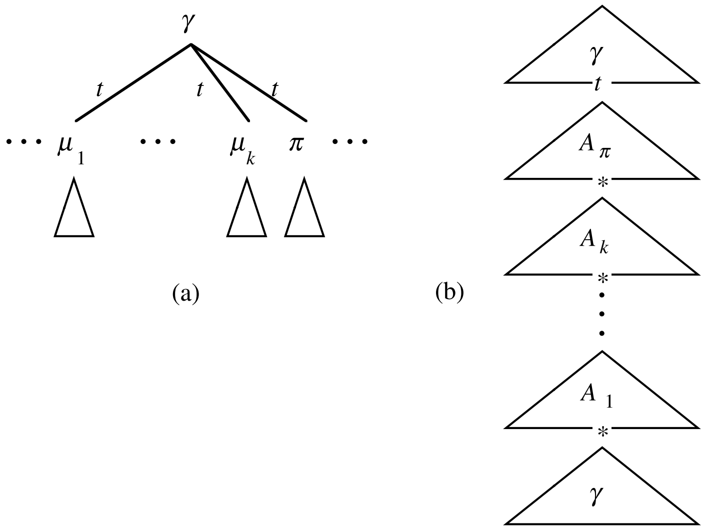

Figure 5
Schematic extended derivation tree and associated derived tree. In a derived tree, the predicative tree adjoined at an address t is required to follow all modifier trees adjoined at the same address, as in (a). The derived tree therefore appears as depicted in (b) with the predicative tree outermost.

than the modifier tree; in terms of the derived tree, it is as if the predicative tree were adjoined at the root of the modifier tree. This accords with the semantic intuition that in such a case (for English at least), the modifier is modifying the original tree, not the predicative one. (The alternate "reading", in which the modifier modifies the predicative tree, is still obtainable under an outermost-predication standard by having the modifier auxiliary tree adjoin dependently at the root node of the predicative tree.) In contrast, the innermost-predication method specifies that the modifier tree attaches higher, as if the modifier tree adjoined at the root of the predicative tree and was therefore modifying the predicative tree, contra semantic intuitions.

For this reason, we specify that outermost predication is mandated. This is easily done by further limiting the base set of ordered derivation trees to those in which predicative trees are ordered after modifier tree siblings.

(From a technical standpoint, by the way, the outermost-predication method has the advantage that it requires no changes to the parsing rules to be presented later, but only a single addition. The innermost-predication method induces some subtle interactions between the original parsing rules and the additional one, necessitating a much more complicated set of modifications to the original algorithm. In fact, the complexities in generating such an algorithm constituted the precipitating factor that led us to revise our original, innermost-predication, attempt at redefining tree-adjoining derivation. The linguistic argument, although commanding, became clear to us only later.)

Another possibility, which we mention but do not pursue here, is to allow for language-particular precedence constraints to restrict the possible orderings of derivation-tree siblings, in a manner similar to the linear precedence constraints of ID/LP format (Gazdar et al., 1985) but at the level of derivation trees. These might be interpreted as hard constraints or soft orderings depending on the application. This more fine-grained approach to the issue of ordering has several applications. Soft orderings might be used to account for ordering preferences among modifiers, such as the default ordering of English adjectives that accounts for the typical preference for "a large red ball" over "? a red large ball" and the typical ordering of temporal before spatial adverbial phrases in German.

Similarly, hard constraints might allow for the handling of an apparent counterexample to the outermost-predication rule.[13] One natural analysis of the sentence

(13) At what time did Brockway say Harrison arrived?

would involve adjunction of a predicative tree for the phrase "did Brockway say" at the root of the tree for "Harrison arrived". A *Wh* modifier tree "at what time" must be adjoined in as well. The example question is ambiguous, of course, as to whether it questions the time of the saying or of the arriving. In the former case, the modifier tree presumably adjoins at the root of the predicative tree for "did Brockway say" that it modifies. In the latter case, which is of primary interest here, it must adjoin at the root of the tree for "Harrison arrived". Thus, both trees would be adjoined at the same address, and the outermost-predication rule would predict the derived sentence to be "Did Brockway say at what time Harrison arrived." To get around this problem, we might specify hard ordering constraints for English that place all *Wh* modifier trees after all predicative trees, which in turn come after all non-*Wh* modifier trees. This would place the *Wh* modifier outermost as required.

13 Other solutions are possible that do not require extended derivations or linear precedence constraints. For instance, we might postulate an elementary tree for the verb *arrived* that includes a substitution node for a fronted adverbial *Wh* phrase.

Although we find this extra flexibility to be an attractive aspect of this approach, we stay with the more stringent outermost-predication restriction in the material that follows.

5. Compilation of TAGs to Linear Indexed Grammars

In this section, we present a technique for compiling tree-adjoining grammars into linear indexed grammars such that the linear-indexed grammar makes explicit the extended derivations of the TAG. This compilation plays two roles. First, it provides for a simple proof of the generative equivalence of TAGs under the standard and extended definitions of derivation, as described at the end of this section. Second, it can be used as the basis for a parsing algorithm that recovers the extended derivations for strings. The design of such an algorithm is the topic of Section 6.

Linear indexed grammars (LIG) constitute a grammatical framework based, like context-free, context-sensitive, and unrestricted rewriting systems, on rewriting strings of nonterminal and terminal symbols. Unlike these systems, linear indexed grammars, like the indexed grammars from which they are restricted, allow stacks of marker symbols, called *indices*, to be associated with the nonterminal symbols being rewritten. The linear version of the formalism allows the full index information from the parent to be used to specify the index information for only one of the child constituents. Thus, a linear indexed production can be given schematically as:

$$N_0[..\beta_0] \rightarrow N_1[\beta_1] \cdots N_{s-1}[\beta_{s-1}]\ N_s[..\beta_s]\ N_{s+1}[\beta_{s+1}] \cdots N_k[\beta_k]$$

The N_i are nonterminals, the β_i strings of indices. The ".." notation stands for the remainder of the stack below the given string of indices. Note that only one element on the right-hand side, N_s, inherits the remainder of the stack from the parent. (This schematic rule is intended to be indicative, not definitive. We ignore issues such as the optionality of the inherited stack, how terminal symbols fit in, and so forth. Vijay-Shanker and Weir (1990) present a complete discussion.)

Vijay-Shanker and Weir (1990) present a way of specifying any TAG as a linear indexed grammar. The LIG version makes explicit the standard notion of derivation being presumed. Also, the LIG version of a TAG grammar can be used for recognition and parsing. Because the LIG formalism is based on augmented rewriting, the parsing algorithms can be much simpler to understand and easier to modify, and no loss of generality is incurred. For these reasons, we use the technique in this work.

The compilation process that manifests the standard definition of derivation can be most easily understood by viewing nodes in a TAG elementary tree as having both a top and bottom component, identically marked for nonterminal category, that dominate (but may not immediately dominate) each other. (See Figure 6.) The rewrite rules of the corresponding linear indexed grammar capture the immediate domination between a bottom node and its child top nodes directly, and capture the domination between top and bottom parts of the same node by optionally allowing rewriting from the top of a node to an appropriate auxiliary tree, and from the foot of the auxiliary tree back to the bottom of the node. The index stack keeps track of the nodes that adjunction has occurred on so that the recognition to the left and the right of the foot node will occur under identical assumption of derivation structure.

The TAG grammar is encoded as a LIG with two nonterminal symbols t and b corresponding to the top and bottom components, respectively, of each node. The stack indices correspond to the individual nodes of the elementary trees of the TAG grammar. Thus, there are as many stack index symbols as there are nodes in the elementary trees

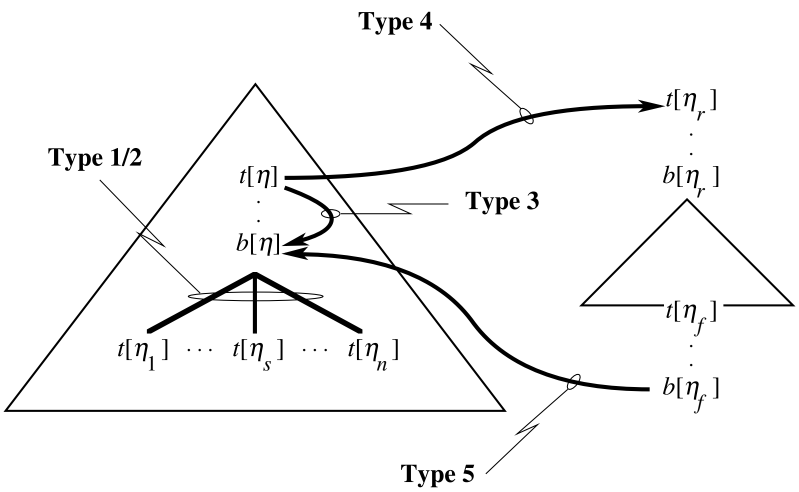

Figure 6
Schematic structure of adjunction with top and bottom of each node separated

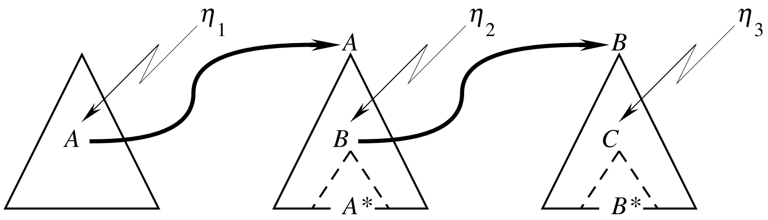

Figure 7
A stack of indices $[\eta_1\eta_2\eta_3]$ captures the adjunction history that led to the reaching of the node η_3 in the parsing process. Parsing of an elementary tree α proceeded to node η_1 in that tree, at which point adjunction of the tree containing η_2 was pursued by the parser. When the node η_2 was reached, the tree containing η_3 was implicitly adjoined. Once this latter tree is completely parsed, the remainder of the tree containing η_2 can be parsed from that point, and so on.

of the grammar, and each such index (i.e., node) corresponds unambiguously to a single address in a single elementary tree. (In fact, the symbols can be thought of as pairs of an elementary tree identifier and an address within that tree, and our implementation encodes them in just that way.) The index at the top of the stack corresponds to the node being rewritten. Thus, a LIG nonterminal with stack $t[\eta]$ corresponds to the top component of node η, and $b[\eta_1\eta_2\eta_3]$ corresponds to the bottom component of η_3. The indices η_1 and η_2 capture the history of adjunctions that are pending completion of the tree in which η_3 is a node. Figure 7 depicts the interpretation of a stack of indices.

In summary, given a tree-adjoining grammar, the following LIG rules are generated:

1. *Immediate domination dominating foot:* For each auxiliary tree node η that dominates the foot node, with children $\eta_1, \ldots, \eta_s, \ldots, \eta_n$, where η_s is the child that also dominates the foot node, include a production

$$b[..\eta] \rightarrow t[\eta_1] \cdots t[\eta_{s-1}]t[..\eta_s]t[\eta_{s+1}] \cdots t[\eta_n]$$

2. *Immediate domination not including foot:* For each elementary tree node η that does not dominate a foot node, with children $\eta_1, \ldots, \eta_n$, include a production

$$b[\eta] \rightarrow t[\eta_1] \cdots t[\eta_n]$$

3. *No adjunction:* For each elementary tree node η that is not marked for substitution or obligatory adjunction, include a production

$$t[..\eta] \rightarrow b[..\eta]$$

4. *Start root of adjunction:* For each elementary tree node η on which the auxiliary tree β with root node η_r can be adjoined, include the following production:

$$t[..\eta] \rightarrow t[..\eta\eta_r]$$

5. *Start foot of adjunction:* For each elementary tree node η on which the auxiliary tree β with foot node η_f can be adjoined, include the following production:

$$b[..\eta\eta_f] \rightarrow b[..\eta]$$

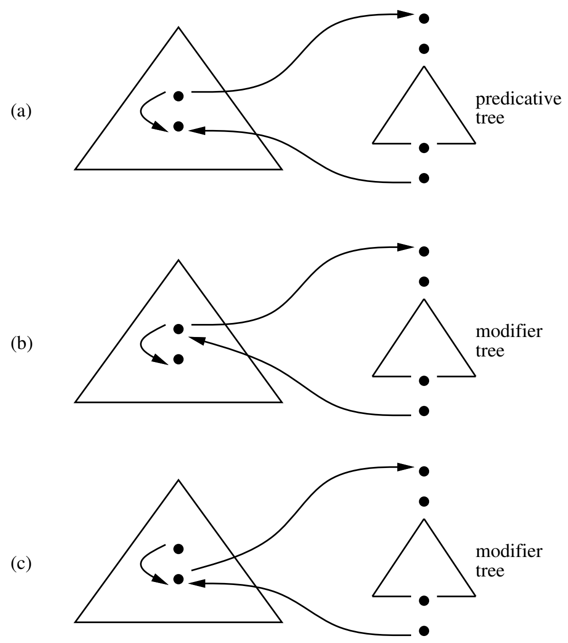

Figure 8
Schematic structure of possible predicative and modifier adjunctions with top and bottom of each node separated.

6. *Start substitution:* For each elementary tree node η marked for substitution on which the initial tree α with root node η_r can be substituted, include the production

$$t[\eta] \rightarrow t[\eta_r]$$

We will refer to productions generated by Rule i above as Type i productions. For example, Type 3 productions are of the form $t[..\eta] \rightarrow b[..\eta]$. For further information concerning the compilation see the work of Vijay-Shanker and Weir (1990). For present purposes, it is sufficient to note that the method directly embeds the standard notion of derivation in the rewriting process. To perform an adjunction, we move (by Rule 4) from the node adjoined at to the top of the root of the auxiliary tree. At the root, additional adjunctions might be performed. When returning from the foot of the auxiliary tree back to the node where adjunction occurred, rewriting continues at the bottom of the node (see Rule 5), not the top, so that no more adjunctions can be started at that node. Thus, the dependent nature of predicative adjunction is enforced because *only a single adjunction can occur at any given node*.

In order to permit extended derivations, we must allow for multiple modifier tree adjunctions at a single node. There are two natural ways this might be accomplished, as depicted in Figure 8.

1. *Modified start foot of adjunction rule:* Allow moving from the bottom of the foot of a modifier auxiliary tree to the top (rather than the bottom) of the node at which it adjoined (Figure 8b).

2. *Modified start root of adjunction rule:* Allow moving from the bottom (rather than the top) of a node to the top of the root of a modifier auxiliary tree (Figure 8c).

As can be seen from the figures, both of these methods allow recursion at a node, unlike the original method depicted in Figure 8a. Thus multiple modifier trees are allowed to adjoin at a single node. Note that since predicative trees fall under the original rules, at most a single predicative tree can be adjoined at a node. The two methods correspond exactly to the innermost- and outermost-predication methods discussed in Section 4.3. For the reasons described there, the latter is preferred. [14]

In summary, independent derivation structures can be allowed for modifier auxiliary trees by starting the adjunction process from the bottom, rather than the top of a node for those trees. Thus, we split Type 4 LIG productions into two subtypes for predicative and modifier trees, respectively.

4a. *Start root of predicative adjunction:* For each elementary tree node η on which the predicative auxiliary tree β with root node η_r can be adjoined, include the following production:

$$t[..\eta] \rightarrow t[..\eta\eta_r]$$

4b. *Start root of modifier adjunction:* For each elementary tree node η on which the modifier auxiliary tree β with root node η_r can be adjoined, include the following production:

$$b[..\eta] \rightarrow t[..\eta\eta_r]$$

Once this augmentation has been made, we no longer need to allow for adjunctions at the root nodes of modifier auxiliary trees, as repeated adjunction is now allowed for by the new rule 4b. Consequently, grammars should forbid adjunction of a modifier tree β_1 at the root of a modifier tree β_2 except where β_1 is intended to modify β_2 directly.

This simple modification to the compilation process from TAG to LIG fully specifies the modified notion of derivation. Note that the extra criterion (5) noted in Section 3.4 is satisfied by this definition: Modifier adjunctions are inherently repeatable and eliminable as the movement through the adjunction "loop" ends up at the same point that it begins. The recognition algorithms for TAG based on this compilation, however, must be adjusted to allow for the new rule types.

This compilation makes possible a simple proof of the weak-generative equivalence of TAGs under the standard and extended derivations.[15] Call the set of languages generable by a TAG under the standard definition of derivation TAL_s and under the extended definition TAL_e. Clearly, $TAL_s \subseteq TAL_e$ since the standard definition can be mimicked by making all auxiliary trees predicative. The compilation above provides the inclusion $TAL_e \subseteq LIL$, where LIL is the set of linear indexed languages. The final inclusion $LIL \subseteq TAL_s$ has been shown indirectly by Vijay-Shanker (1987) using embedded push-down automata and modified head grammars as intermediaries. From these inclusions, we can conclude that $TAL_s = TAL_e$.

14 The more general definition allowing predicative trees to occur anywhere within a sequence of modifier adjunctions would be achieved by adding both types of rules.

15 We are grateful to K. Vijay-Shanker for bringing this point to our attention.

6. Recognition and Parsing

A recognition algorithm for TAGs can be constructed based on the above translation into corresponding LIGs as specified by Rules 1 through 6 in the previous section. The algorithm is not a full recognition algorithm for LIGs, but rather, is tuned for exactly the types of rules generated as output of this compilation process. In this section, we present the recognition algorithm and modify it to work with the extended derivation compilation.

We will use the following notations in this and later sections. The symbol P will serve as a variable over the two LIG grammar nonterminals t and b. The substring of the string $w_1 \cdots w_n$ being parsed between indices i and j will be notated as $w_{i+1} \cdots w_j$, which we take to be the empty string when i is greater than or equal to j. We will use Γ, Δ, and Θ for sequences containing terminals and LIG nonterminals with their stack specifications. For instance, Γ might be $t[\eta_1]t[..\eta_2]t[\eta_3]$.

The parsing algorithm can be seen as a tabular parsing method based on deduction of items, as in Earley deduction (Pereira and Warren, 1983). We will so describe it, by presenting inference rules over items of the form

$$\langle P[\eta] \rightarrow \Gamma \bullet \Delta, i, j, k, l \rangle$$

Such items play the role of the items of Earley's algorithm. Unlike the items of Earley's algorithm, however, an item of this form does not embed a grammar rule proper, that is $P[\eta] \rightarrow \Gamma\Delta$ is not necessarily a rule of the grammar. Rather, it is what we will call a *reduced* rule; for reasons described below, the nonterminals in Γ and Δ as well as the nonterminal $P[\eta]$ record only the top element of each stack of indices. We will use the notation $\overline{P[\eta] \rightarrow \Gamma\Delta}$ for the unreduced form of the rule whose reduced form is $P[\eta] \rightarrow \Gamma\Delta$. For instance, the rule specified by the notation $\overline{t[\eta_1] \rightarrow t[\eta_2]}$ might be the rule $t[..\eta_1] \rightarrow t[..\eta_1\eta_2]$. The reader can easily verify that the TAG to LIG compilation is such that there is a one-to-one correspondence between the generated rules and their reduced form. Consequently, this notation is well-defined.

The dot in the items is analogous to that found in Earley and LR items as well. It serves as a marker for how far recognition has proceeded in identifying the subconstituents for this rule. The indices i, j, k, and l specify the portion of the string $w_1 \cdots w_n$ covered by the recognition of the item. The substring between i and l (i.e., $w_{i+1} \cdots w_l$) has been recognized, perhaps with a region between j and k where the foot of the tree below the node η has been recognized. (If the foot node is not dominated by Γ, we take the values of j and k to be the dummy value '$-$'.)

6.1 The Inference Rules

In this section, we specify several inference rules for parsing a LIG generated from a TAG, which we recall in this section. One explanatory comment is in order, however, before the rules are presented. The rules of a LIG associate with each constituent a nonterminal and a stack of indices. It seems natural for a parsing algorithm to maintain this association by building items that specify for each constituent the full information of nonterminal and index stack. However, this would necessitate storing an unbounded amount of information for each potential constituent, resulting in a parsing algorithm that is potentially quite inefficient when nondeterminism arises during the parsing process, and perhaps non-effective if the grammar is infinitely ambiguous. Instead, the parse items manipulated by the inference rules that we present do not keep all of this information for each constituent. Rather, the items keep only the single top stack element for each constituent (in addition to the nonterminal symbol). This drastically decreases the number of possible items, and accounts for the polynomial character of the resultant algorithm.[16] Side conditions make up for some of the loss of information, thereby maintaining correctness. For instance, the Type 4 Completor rule specifies a relation between η and η_f that takes the place of popping an element off of the stack associated with η. However, the side conditions are strictly weaker than maintaining full stack information. Consequently, the algorithm, though correct, does not maintain the valid prefix property. See the work of Schabes (1991) for further discussion and alternatives.

Scanning and prediction work much as in Earley's original algorithm.

- *Scanner:*

$$\frac{\langle b[\eta] \rightarrow \Gamma \bullet a\Delta, i, j, k, l\rangle}{\langle b[\eta] \rightarrow \Gamma a \bullet \Delta, i, j, k, l+1\rangle} \qquad a = w_{l+1}$$

  Note that the only rules that need be considered are those where the parent is a bottom node, as terminal symbols occur on the right-hand side only of Type 1 or 2 productions. Otherwise, the rule is exactly as that for Earley's algorithm except that the extra foot indices (j and k) are carried along.

- *Predictor:*

$$\frac{\langle P[\eta] \rightarrow \Gamma \bullet P'[\eta']\Delta, i, j, k, l\rangle}{\langle P'[\eta'] \rightarrow \bullet \Theta, l, -, -, l\rangle} \qquad \overline{P'[\eta'] \rightarrow \Theta}$$

  This rule serves to form predictions for any type production in the grammar, as the variables P and P' range over the values t and b. In the predicted item, the foot is not dominated by the (empty) recognized input, so that the dummy value '$-$' is used for the foot indices. Note that the predicted item records the reduced form of an unreduced rule $\overline{P'[\eta'] \rightarrow \Theta}$ of the grammar.

Completion of items (moving of the dot from left to right over a nonterminal) breaks up into several cases, depending on which production type is being completed. This is because the addition of the extra indices and the separate interpretations for top and bottom productions require differing index manipulations to be performed. We will list the various steps, organized by what type of production they participate in the completion of.

Productions that specify immediate domination (from Rules 1 and 2) are completed whenever the top of the child node is fully recognized.

- *Type 1 and 2 Completor:*

$$\frac{\langle b[\eta_1] \rightarrow \Gamma \bullet t[\eta]\Delta, m, j', k', i\rangle \quad \langle t[\eta] \rightarrow \Theta \bullet, i, j, k, l\rangle}{\langle b[\eta_1] \rightarrow \Gamma t[\eta] \bullet \Delta, m, j \cup j', k \cup k', l\rangle}$$

  Here, $t[\eta]$ has been fully recognized as the substring between i and l. The item expecting $t[\eta]$ can be completed. One of the two antecedent items

16 Vijay-Shanker and Weir (1990) first proposed the recording of only the top stack element in order to achieve efficient parsing. The algorithm they presented is a bottom-up general LIG parsing algorithm. Schabes (1991) sketches a proof of an $O(n^6)$ bound for an Earley-style algorithm for TAG parsing that is more closely related to the algorithm proposed here.

might also dominate the foot node of the tree to which η and η_1 belong, and would therefore have indices for the foot substring. The operations $j \cup j'$ and $k \cup k'$ are used to specify whichever of j or j' (and respectively for k or k') contain foot substring indices. The formal definition of $\cup$ is as follows:

$$j \cup j' = \begin{cases} j & \text{if } j' = - \\ j' & \text{if } j = - \\ j & \text{if } j' = j \\ \text{undefined} & \text{otherwise} \end{cases}$$

The remaining rules (3 through 6) are each completed by a particular completion instance.

- *Type 3 Completor:*

$$\frac{\langle t[\eta] \rightarrow \bullet\, b[\eta], i, -, -, i\rangle \quad \langle b[\eta] \rightarrow \Theta\, \bullet, i, j, k, l\rangle}{\langle t[\eta] \rightarrow b[\eta]\, \bullet, i, j, k, l\rangle}$$

  This rule is used to complete a prediction that no (predicative) adjunction occurs at node η. Once the part of the string dominated by $b[\eta]$ has been found, as evidenced by the second antecedent item, the prediction of no adjunction can be completed.

- *Type 4 Completor:*

$$\frac{\begin{array}{r} \langle t[\eta] \rightarrow \bullet\, t[\eta_r], i, -, -, i\rangle \\ \langle t[\eta_r] \rightarrow \Theta\, \bullet, i, j, k, l\rangle \\ \langle b[\eta] \rightarrow \Delta\, \bullet, j, p, q, k\rangle \end{array}}{\langle t[\eta] \rightarrow t[\eta_r]\, \bullet, i, p, q, l\rangle} \qquad t[..\eta] \rightarrow t[..\eta\eta_r]$$

  Here, an adjunction has been predicted at η, and the adjoined derived tree (between $t[\eta]$ and $b[\eta]$) and the derived material that η itself dominates (below $b[\eta]$) have both been completed. Thus $t[\eta]$ is completely recognized. Note that the side condition (the unreduced form of the reduced rule in the first antecedent item) is placed merely to guarantee that η_r is the root node of an adjoinable auxiliary tree.

- *Type 5 Completor:*

$$\frac{\langle b[\eta_f] \rightarrow \bullet\, b[\eta], i, -, -, i\rangle \quad \langle b[\eta] \rightarrow \Theta\, \bullet, i, j, k, l\rangle}{\langle b[\eta_f] \rightarrow b[\eta]\, \bullet, i, i, l, l\rangle} \qquad b[..\eta\eta_f] \rightarrow b[..\eta]$$

  When adjunction has been performed, and recognition up to the foot node η_f has been performed, it is necessary to recognize all the material under the foot node. When that is done, the foot node prediction can be completed. Note that it must be possible to have adjoined the auxiliary tree at node η as specified in the production in the side condition.

- *Type 6 Completor:*

$$\frac{\langle t[\eta] \rightarrow \bullet\, t[\eta_r], i, -, -, i\rangle \quad \langle t[\eta_r] \rightarrow \Theta\, \bullet, i, -, -, l\rangle}{\langle t[\eta] \rightarrow t[\eta_r]\, \bullet, i, -, -, l\rangle} \qquad t[\eta] \rightarrow t[\eta_r]$$

  Completion of the material below the root node η_r of an initial tree allows for the completion of the node at which substitution occurred.

The recognition process for a string $w_1 \cdots w_n$ starts with some items that serve as axioms for these inference rules. For each rule $\overline{t[\eta_s] \to \Gamma}$ where η_s is the root node of an initial tree which node is labeled with the start nonterminal, the item $\langle t[\eta_s] \to \bullet \, \Gamma, 0, -, -, 0\rangle$ is an axiom. If from these axioms an item of the form $\langle t[\eta_s] \to \Gamma \bullet, 0, -, -, n\rangle$ can be proved according to the rules of inference above, the string is accepted; otherwise it is rejected.

Alternatively, the axioms can be stated as if there were extra rules $S \to t[\eta_s]$ for each η_s a start-nonterminal-labeled root node of an initial tree. In this case, the axioms are items of the form $\langle S \to \bullet \, t[\eta_s], 0, -, -, 0\rangle$ and the string is accepted upon proving $\langle S \to t[\eta_s] \bullet, 0, -, -, n\rangle$. In this case, an extra prediction and completion rule is needed just for these rules, since the normal rules do not allow S on the left-hand side. This point is taken up further in Section 6.4.

Generation of items can be cached in the standard way for inference-based parsing algorithms (Shieber, 1992); this leads to a tabular or chart-based parsing algorithm.

6.2 The Algorithm Invariant

The algorithm maintains an invariant that holds of all items added to the chart. We will describe the invariant using some additional notational conventions. Recall that $\overline{P[\eta] \to \Gamma}$ is the LIG production in the grammar whose reduced form is $P[\eta] \to \Gamma$. The notation $\Gamma[\gamma]$ where γ is a sequence of stack symbols (i.e., nodes), specifies the sequence Γ with γ replacing the occurrence of .. in the stack specifications. For example, if Γ is the sequence $t[\eta_1]t[..\eta_2]t[\eta_3]$, then $\Gamma[\gamma] = t[\eta_1]t[\gamma\eta_2]t[\eta_3]$. A single LIG derivation step will be notated with $\Rightarrow$ and its reflexive transitive closure with $\Rightarrow^*$.

The invariant specifies that $\langle P[\eta] \to \Gamma \bullet \Delta, i, j, k, l\rangle$ is in the chart only if[17]

1. If node η dominates the foot node η_f of the tree to which it belongs, then there exists a string of stack symbols (i.e., nodes) γ such that
   - (a) $\overline{P[\eta] \to \Gamma\Delta}$ is a LIG rule in the grammar, where $\overline{\Gamma}$ is the unreduced form of Γ.
   - (b) $\overline{\Gamma}[\gamma\eta] \Rightarrow^* w_{i+1} \cdots w_j b[\gamma\eta_f] w_{k+1} \cdots w_l$.
   - (c) $b[\gamma\eta_f] \Rightarrow^* w_{j+1} \cdots w_k$.
2. If node η does not dominate the foot node η_f of the tree to which it belongs or there is no foot node in the tree, then
   - (a) $\overline{P[\eta] \to \Gamma\Delta}$ is a LIG rule in the grammar, where $\overline{\Gamma}$ is the unreduced form of Γ.
   - (b) $\overline{\Gamma} \Rightarrow^* w_{i+1} \cdots w_l$.
   - (c) j and k are not bound.

According to this invariant, for a node η_s which is the root of an initial tree, the item $\langle t[\eta_s] \to \Gamma \bullet, 0, -, -, n\rangle$ is in the chart only if $t[\eta_s] \Rightarrow \overline{\Gamma} \Rightarrow^* w_1 \cdots w_n$. Thus, soundness of the algorithm as a recognizer follows.

17 The invariant is not stated as a biconditional because this would require strengthening of the antecedent condition. The natural strengthening, following the standard for Earley's algorithm, would be to add a requirement that the item be consistent with left context, as

(d) $\eta_s \Rightarrow^* w_1 \cdots w_i P[\gamma\eta]$

but this is too strong. This condition implies that the algorithm possesses the valid prefix property, which it does not. The exact statement of the invariant condition that would allow for exact specifications of the item semantics is the topic of ongoing research. However, the current specification is sufficient for proving soundness of the algorithm.

6.3 Modifications for Extended Derivations

Extending the algorithm to allow for the new types of production (specifically, as derived by Rule 4b) requires adding a completion rule for Type 4b productions. For the new type of production, a completion rule of the following form is required:

- *Type 4b Completor:*

$$\frac{\begin{array}{l}\langle b[\eta] \rightarrow \ \bullet\, t[\eta_r], i, -, -, i\rangle \\ \qquad\quad \langle t[\eta_r] \rightarrow \Theta \,\bullet\, , i, j, k, l\rangle \\ \qquad\qquad\quad \langle b[\eta] \rightarrow \Delta \,\bullet\, , j, p, q, k\rangle\end{array}}{\langle b[\eta] \rightarrow t[\eta_r]\,\bullet\, , i, p, q, l\rangle} \qquad b[..\eta] \rightarrow t[..\eta\eta_r]$$

In addition to being able to complete Type 4b items, we must also be able to complete other items using completed Type 4b items. This is an issue in particular for completor rules that might move their dot over a $b[\eta]$ constituent, in particular, the Type 3 and 5 Completors. However, these rules have been stated so that the antecedent item with right-hand side $b[\eta]$ already matches Type 4b items. Furthermore, the general statement, including index manipulation is still appropriate in the context of Type 4b productions. Thus, no further changes to the recognition inference rules are needed for this purpose.

However, a bit of care must be taken in the interpretation of the Type 1/2 Completor. Type 4b items that require completion bear a superficial resemblance to Type 1 and 2 items, in that both have a constituent of the form $t[\_]$ after the dot. In Type 4b items, the constituent is $t[\eta_r]$, in Type 4a items $t[\eta]$. But it is crucial that the Type 1/2 Completor not be used to complete Type 4b items. A simple distinguishing characteristic is that in Type 1 and 2 items to be completed, the node η after the dot is never a root node (as it is immediately dominated by η_1), whereas in Type 4b items, the node η_r after the dot is always a root node (of a modifier tree). Simple side conditions can distinguish the cases.

Figure 9 contains the final versions of the inference rules for recognition of LIGs corresponding to extended TAG derivations.

6.4 Maintaining Derivation Structures

One of the intended applications for extended derivation TAG parsing is the parsing of synchronous TAGs. Especially important in this application is the ability to generate the derivation trees while parsing proceeds.

A synchronous TAG is composed of two base TAGs (which we will call the *source* and *target* TAG) whose elementary trees have been paired one-to-one. A synchronous TAG whose source TAG is a grammar for a fragment of English, and whose target TAG is a grammar for a logical form language may be used to generate logical forms for each sentence of English that the source grammar admits (Shieber and Schabes, 1990). Similarly, with source and target swapped, the synchronized grammar may be used to generate English sentences corresponding to logical forms (Shieber and Schabes, 1991). If the source and target grammars specify fragments of natural languages, an automatic translation system is specified (Abeillé, Schabes, and Joshi, 1990).

Abstractly viewed, the processing of a synchronous grammar proceeds by parsing an input string according to the source grammar, thereby generating a derivation tree for the string; mapping the derivation tree into a derivation tree for the target grammar; and generating a derived tree (hence, derived string) according to the target grammar.

One frequent worry about synchronous TAGs as used in their semantic interpretation mode is whether it is possible to perform incremental interpretation. The abstract view of processing just presented seems to require that a full derivation tree be developed before interpretation into the logical form language can proceed. Incremental

- *Scanner:*

$$\frac{\langle b[\eta] \rightarrow \Gamma \bullet a\Delta, i, j, k, l\rangle}{\langle b[\eta] \rightarrow \Gamma a \bullet \Delta, i, j, k, l+1\rangle} \qquad a = w_{l+1}$$

- *Predictor:*

$$\frac{\langle P[\eta] \rightarrow \Gamma \bullet P'[\eta']\Delta, i, j, k, l\rangle}{\langle P'[\eta'] \rightarrow \ \bullet \Theta, l, -, -, l\rangle} \qquad \overline{P'[\eta'] \rightarrow \Theta}$$

- *Type 1 and 2 Completor:*

$$\frac{\langle b[\eta_1] \rightarrow \Gamma \bullet t[\eta]\Delta, m, j', k', i\rangle \quad \langle t[\eta] \rightarrow \Theta \bullet , i, j, k, l\rangle}{\langle b[\eta_1] \rightarrow \Gamma t[\eta] \bullet \Delta, m, j \cup j', k \cup k', l\rangle} \qquad \eta \text{ not a root node}$$

- *Type 3 Completor:*

$$\frac{\langle t[\eta] \rightarrow \ \bullet b[\eta], i, -, -, i\rangle \quad \langle b[\eta] \rightarrow \Theta \bullet , i, j, k, l\rangle}{\langle t[\eta] \rightarrow b[\eta] \bullet , i, j, k, l\rangle}$$

- *Type 4a Completor:*

$$\frac{\begin{array}{l}\langle t[\eta] \rightarrow \ \bullet t[\eta_r], i, -, -, i\rangle \\ \qquad \langle t[\eta_r] \rightarrow \Theta \bullet , i, j, k, l\rangle \\ \qquad\qquad \langle b[\eta] \rightarrow \Delta \bullet , j, p, q, k\rangle\end{array}}{\langle t[\eta] \rightarrow t[\eta_r] \bullet , i, p, q, l\rangle} \qquad t[..\eta] \rightarrow t[..\eta\eta_r]$$

- *Type 4b Completor:*

$$\frac{\begin{array}{l}\langle b[\eta] \rightarrow \ \bullet t[\eta_r], i, -, -, i\rangle \\ \qquad \langle t[\eta_r] \rightarrow \Theta \bullet , i, j, k, l\rangle \\ \qquad\qquad \langle b[\eta] \rightarrow \Delta \bullet , j, p, q, k\rangle\end{array}}{\langle b[\eta] \rightarrow t[\eta_r] \bullet , i, p, q, l\rangle} \qquad b[..\eta] \rightarrow t[..\eta\eta_r]$$

- *Type 5 Completor:*

$$\frac{\langle b[\eta_f] \rightarrow \ \bullet b[\eta], i, -, -, i\rangle \quad \langle b[\eta] \rightarrow \Theta \bullet , i, j, k, l\rangle}{\langle b[\eta_f] \rightarrow b[\eta] \bullet , i, i, l, l\rangle} \qquad b[..\eta\eta_f] \rightarrow b[..\eta]$$

- *Type 6 Completor:*

$$\frac{\langle t[\eta] \rightarrow \ \bullet t[\eta_r], i, -, -, i\rangle \quad \langle t[\eta_r] \rightarrow \Theta \bullet , i, -, -, l\rangle}{\langle t[\eta] \rightarrow t[\eta_r] \bullet , i, -, -, l\rangle} \qquad t[\eta] \rightarrow t[\eta_r]$$

Figure 9
Inference Rules for Extended Derivation TAG Recognition

interpretation, on the other hand, would allow partial interpretation results to guide the parsing process on-line, thereby decreasing the nondeterminism in the parsing process. Whether incremental interpretation is possible depends precisely on the extent to which the three abstract phases of synchronous TAG processing can in fact be interleaved. In previous work, we left this issue open. In this section, we allay these worries by showing how the extended TAG parser just presented can build derivation trees incrementally as parsing proceeds. Once this has been demonstrated, it should be obvious that these derivation trees could be transferred to target derivation trees during the parsing process, and immediately generated from. Thus, incremental interpretation is demonstrated to be possible in the synchronous TAG framework. In fact, the technique presented in this section has allowed for the first implementation of synchronous TAG processing, due to Onnig Dombalagian. This implementation was directly based on the inference-based TAG parser mentioned in Section 6.5 and presented in full elsewhere (Schabes and Shieber, 1992).

We associate with each item a set of *operations* that have been implicitly carried out by the parser in recognizing the substring covered by the item. An operation can be characterized by a derivation tree and a tree address at which the derivation tree is to be placed; it corresponds roughly to a branch of a derivation tree. Prediction items have the empty set of operations. Type 4 and 6 completion steps build new elements of the sets as they correspond to actually carrying out adjunction and substitution operations, respectively. Other completion steps merely pool the operations from their constituent parts.

In describing the building of derivation trees, we will use normal set notation for the sets of derivation trees. We will assume that for each node η, there are functions $tree(\eta)$ and $addr(\eta)$ that specify, respectively, the initial tree that η occurs in and its address in that tree. Finally, we will use a constructor function for derivation trees $deriv(\gamma, S)$, where γ specifies an elementary tree and S specifies a set of operations on it. An operation is built with $op(t, D)$ where t is a tree address and D is a derivation tree to be operated at that address.

Figure 10 lists the previously presented recognition rules augmented to build derivation structures as the final component of each item. The axioms for this inference system are items of the form $\langle S \rightarrow \bullet\, t[\eta_s], 0, -, -, 0, \{\}\rangle$, where we assume as in Section 6.1 that there are extra rules $S \rightarrow t[\eta_s]$ for each η_s a start-nonterminal-labeled root node of an initial tree. We require an extra rule for prediction and completion to handle this new type of rule. The predictor rule is the obvious analog:

- *Start Rule Predictor:*

$$\frac{\langle S \rightarrow \Gamma \bullet P'[\eta']\Delta, i, j, k, l, S\rangle}{\langle P'[\eta'] \rightarrow \bullet\, \Theta, l, -, -, l, \{\}\rangle} \qquad \overline{P'[\eta'] \rightarrow \Theta}$$

In fact, the existing predictor rule could have been easily generalized to handle this case.

The completor for these start rules is the obvious analog to a Type 6 completor, except in the handling of the derivation. It delivers, instead of a set of derivation operations, a single derivation tree.

- *Start Rule Completor:*

$$\frac{\langle S \rightarrow \bullet\, t[\eta_s], i, -, -, i, \{\}\rangle \quad \langle t[\eta_s] \rightarrow \Theta \bullet, i, -, -, l, S\rangle}{\langle S \rightarrow t[\eta_s] \bullet, i, -, -, l, deriv(tree(\eta_s), S)\rangle}$$

- *Scanner:*

$$\frac{\langle b[\eta] \rightarrow \Gamma \bullet a\Delta, i, j, k, l, S\rangle}{\langle b[\eta] \rightarrow \Gamma a \bullet \Delta, i, j, k, l+1, S\rangle} \qquad a = w_{l+1}$$

- *Predictor:*

$$\frac{\langle P[\eta] \rightarrow \Gamma \bullet P'[\eta']\Delta, i, j, k, l, S\rangle}{\langle P'[\eta'] \rightarrow \ \bullet \Theta, l, -, -, l, \{\}\rangle} \qquad \overline{P'[\eta'] \rightarrow \Theta}$$

- *Type 1 and 2 Completor:*

$$\frac{\langle b[\eta_1] \rightarrow \Gamma \bullet t[\eta]\Delta, m, j', k', i, S_1\rangle \quad \langle t[\eta] \rightarrow \Theta \bullet, i, j, k, l, S_2\rangle}{\langle b[\eta_1] \rightarrow \Gamma t[\eta] \bullet \Delta, m, j \cup j', k \cup k', l, S_1 \cup S_2\rangle} \qquad \eta \text{ not a root node}$$

- *Type 3 Completor:*

$$\frac{\langle t[\eta] \rightarrow \ \bullet b[\eta], i, -, -, i, \{\}\rangle \quad \langle b[\eta] \rightarrow \Theta \bullet, i, j, k, l, S\rangle}{\langle t[\eta] \rightarrow b[\eta] \bullet, i, j, k, l, S\rangle}$$

- *Type 4a Completor:*

$$\frac{\begin{array}{r}\langle t[\eta] \rightarrow \ \bullet t[\eta_r], i, -, -, i, \{\}\rangle \\ \langle t[\eta_r] \rightarrow \Theta \bullet, i, j, k, l, S_1\rangle \\ \langle b[\eta] \rightarrow \Delta \bullet, j, p, q, k, S_2\rangle\end{array}}{\langle t[\eta] \rightarrow t[\eta_r] \bullet, i, p, q, l, \{op(addr(\eta), deriv(tree(\eta_r), S_1))\} \cup S_2\rangle} \qquad t[..\eta] \rightarrow t[..\eta\eta_r]$$

- *Type 4b Completor:*

$$\frac{\begin{array}{r}\langle b[\eta] \rightarrow \ \bullet t[\eta_r], i, -, -, i, \{\}\rangle \\ \langle t[\eta_r] \rightarrow \Theta \bullet, i, j, k, l, S_1\rangle \\ \langle b[\eta] \rightarrow \Delta \bullet, j, p, q, k, S_2\rangle\end{array}}{\langle t[\eta] \rightarrow t[\eta_r] \bullet, i, p, q, l, \{op(addr(\eta), deriv(tree(\eta_r), S_1))\} \cup S_2\rangle} \qquad b[..\eta] \rightarrow t[..\eta\eta_r]$$

- *Type 5 Completor:*

$$\frac{\langle b[\eta_f] \rightarrow \ \bullet b[\eta], i, -, -, i, \{\}\rangle \quad \langle b[\eta] \rightarrow \Theta \bullet, i, j, k, l, S\rangle}{\langle b[\eta_f] \rightarrow b[\eta] \bullet, i, i, l, l, S\rangle} \qquad b[..\eta\eta_f] \rightarrow b[..\eta]$$

- *Type 6 Completor:*

$$\frac{\langle t[\eta] \rightarrow \ \bullet t[\eta_r], i, -, -, i, \{\}\rangle \quad \langle t[\eta_r] \rightarrow \Theta \bullet, i, -, -, l, S\rangle}{\langle t[\eta] \rightarrow t[\eta_r] \bullet, i, -, -, l, \{op(addr(\eta), deriv(tree(\eta_r), S))\}\rangle} \qquad t[\eta] \rightarrow t[\eta_r]$$

Figure 10
Inference Rules for Extended Derivation TAG Parsing

The string is accepted upon proving $\langle S \rightarrow t[\eta_s] \bullet, 0, -, -, n, D\rangle$, where D is the derivation developed during the parse.

6.5 Complexity Considerations

The inference system of Section 6.3 essentially specifies a parsing algorithm with complexity of $O(n^6)$ in the length of the string. Adding explicit derivation structures to the items, as in the inference system of the previous section eliminates the polynomial character of the algorithm, in that there may be an unbounded number of derivations corresponding to any given item of the original sort. Even for finitely ambiguous grammars, the number of derivations may be exponential. Nonetheless, this fact does not vitiate the usefulness of the second algorithm, which maintains derivations explicitly. The point of this augmentation is to allow for incremental interpretation — for interleaved processing of a post-syntactic sort — so as to guide the parsing process in making choices on-line. By using the extra derivation information, the parser should be able to eliminate certain nondeterministic paths of computation; otherwise, there is no reason to do the interpretation incrementally. But this determinization of choice presumably decreases the complexity. Thus, the extra information is designed for use in cases where the full search space is not intended to be explored.

Of course, a polynomial shared-forest representation of the exponential number of derivations could have been maintained (by maintaining back pointers among the items in the standard fashion). For performing incremental interpretation for the purpose of determinization of parsing, however, the non-shared representation is sufficient, and preferable on grounds of ease of implementation and expository convenience.

As a proof of concept, the parsing algorithm just described was implemented in Prolog on top of a simple, general-purpose, agenda-based inference engine. Encodings of explicit inference rules are essentially interpreted by the inference engine. The Prolog database is used as the chart; items not already subsumed by a previously generated item are asserted to the database as the parser runs. An agenda is maintained of potential new items. Items are added to the agenda as inference rules are triggered by items added to the chart. Because the inference rules are stated explicitly, the relation between the abstract inference rules described in this paper and the implementation is extremely transparent. As a meta-interpreter, the prototype is not particularly efficient. (In particular, the implementation does not achieve the theoretical $O(n^6)$ bound on complexity, because of a lack of appropriate indexing.) Code for the prototype implementation is available for distribution electronically from the authors.

7. Conclusion

The precise formulation of derivation for tree-adjoining grammars has important ramifications for a wide variety of uses of the formalism, from syntactic analysis to semantic interpretation and statistical language modeling. We have argued that the definition of tree-adjoining derivation must be reformulated in order to take greatest advantage of the decoupling of derivation tree and derived tree by manifesting the proper linguistic dependencies in derivations. The particular proposal is both precisely characterizable through a definition of TAG derivations as equivalence classes of ordered derivation trees, and computationally operational, by virtue of a compilation to linear indexed grammars together with an efficient algorithm for recognition and parsing according to the compiled grammar.

Acknowledgments

Order of authors is not intended as an indication of precedence of authorship. Much of the work reported in this paper was performed while the first author was at the Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA. The first author was supported in part by DARPA Grant N0014-90-31863, ARO Grant DAAL03-89-C-0031 and NSF Grant IRI-90-16592. The second author was supported in part by Presidential Young Investigator award IRI-91-57996 from the National Science Foundation and a matching grant from Xerox Corporation. The authors wish to thank Aravind Joshi for his support of the research, and Aravind Joshi, Judith Klavans, Anthony Kroch, Shalom Lappin, Kathy McCoy, Fernando Pereira, James Pustejovsky, and K. Vijay-Shanker for their helpful discussions of the issues involved. We are indebted to David Yarowsky for aid in the design of the experiment mentioned in

footnote 9 and for its execution.

References

Abeillé, Anne, Yves Schabes, and Aravind K. Joshi. 1990. Using lexicalized tree adjoining grammars for machine translation. In *Proceedings of the 13th International Conference on Computational Linguistics*, volume 3, pages 1–6, Helsinki University, Helsinki, Finland, August.

Gazdar, Gerald, Ewan Klein, Geoffrey K. Pullum, and Ivan A. Sag. 1985. *Generalized Phrase Structure Grammar*. Blackwell Publishing, Oxford, England.

Joshi, A. K., S. R. Kosaraju, and H. M. Yamada. 1972a. String adjunct grammars: I. Local and distributed adjunction. *Information and Control*, 21(2):93–116, September.

Joshi, A. K., S. R. Kosaraju, and H. M. Yamada. 1972b. String adjunct grammars: II. Equational representation, null symbols, and linguistic relevance. *Information and Control*, 21(3):235–260, October.

Joshi, Aravind K., L. S. Levy, and M. Takahashi. 1975. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1):136–163.

Kroch, Anthony S. 1989. Asymmetries in long distance extraction in a TAG grammar. In M. Baltin and A. Kroch, editors, *Alternative Conceptions of Phrase Structure*. University of Chicago Press, Chicago, Illinois, pages 66–98.

Kroch, Anthony S. and Aravind K. Joshi. 1985. The linguistic relevance of tree adjoining grammar. Technical Report MS-CIS-85-18, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania, April.

Pereira, Fernando C. N. and David H. D. Warren. 1983. Parsing as deduction. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pages 137–144, Cambridge, Massachusetts, June 15-17.

Quirk, Randolph, Sidney Greenbaum, Geoffrey Leech, and Jan Svartvik. 1985. *A Comprehensive Grammar of the English Language*. Longman, London, England.

Resnik, Philip. 1992. Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. In *Proceedings of the 14th International Conference on Computational Linguistics*, pages 418–424, Nantes, France, 23–28 August.

Schabes, Yves. 1991. The valid prefix property and left to right parsing of tree-adjoining grammar. In *Proceedings of the Second International Workshop on Parsing Technologies*, Cancun, Mexico, February.

Schabes, Yves. 1992. Stochastic lexicalized tree-adjoining grammars. In *Proceedings of the 14th International Conference on Computational Linguistics*, pages 426–432, Nantes, France, 23–28 August.

Schabes, Yves and Stuart M. Shieber. 1992. An alternative conception of tree-adjoining derivation. Technical Report 08-92, Harvard University, Cambridge, Massachusetts.

Schabes, Yves and Richard C. Waters. 1993a. Lexicalized context-free grammars. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 121–129, Columbus, Ohio, June.

Schabes, Yves and Richard C. Waters. 1993b. Stochastic lexicalized context-free grammars. In *Proceedings of the Third International Workshop on Parsing Technologies*, pages 257–266, Tilburg, The Netherlands and Durbuy, Belgium, August.

Shieber, Stuart M. 1992. *Constraint-Based Grammar Formalisms*. MIT Press, Cambridge, Massachusetts.

Shieber, Stuart M. Forthcoming. Restricting the weak-generative capacity of synchronous tree-adjoining grammars. *Computational Intelligence*.

Shieber, Stuart M. and Yves Schabes. 1990. Synchronous tree-adjoining grammars. In *Proceedings of the 13th International Conference on Computational Linguistics*, volume 3, pages 253–258, Helsinki University, Helsinki, Finland, August.

Shieber, Stuart M. and Yves Schabes. 1991. Generation and synchronous tree adjoining grammars. *Computational Intelligence*, 4(7):220–228.

Vijay-Shanker, K. 1987. *A Study of Tree Adjoining Grammars*. Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania.

Vijay-Shanker, K. and Aravind K. Joshi. 1985. Some computational properties of tree adjoining grammars. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, pages 82–93, Chicago, Illinois, July.

References

Vijay-Shanker, K. and Aravind K. Joshi. 1988. Feature structure based tree adjoining grammars. In *Proceedings of the 12th International Conference on Computational Linguistics*, Budapest, Hungary, August.

Vijay-Shanker, K. and David J. Weir. 1990. Polynomial parsing of extensions of context-free grammars. In Masaru Tomita, editor, *Current Issues in Parsing Technologies*. Kluwer Academic Publishers, Dordrecht, The Netherlands, pages 191–206.

A. Proof of Redundancy of Adjacent Sibling Swapping

A.1 Preliminaries

A.1.1 Tree Addresses. We define *tree addresses* (variables over which are conventionally notated $p, q, \ldots, t, u, v$ and their subscripted and primed variants) as the finite, possibly empty, sequences of positive integers (conventionally i, j, k), with $\_ \cdot \_$ as the sequence concatenation operator. We uniformly abuse notation by conflating the distinction between singleton sequences and their one element.

We use the notation $p \prec q$ to notate that tree address p is a proper prefix of q, and $p \preceq q$ for improper prefix. When $p \preceq q$, we write $q - p$ for the (possibly empty) sequence obtained from q by removing p from the front, e.g. $1 \cdot 2 \cdot 3 \cdot 4 - 1 \cdot 2 = 3 \cdot 4$.

A.1.2 Trees. We will take trees (conventionally A, B, E, T; also α, β, γ in the prior text) to be finite partial functions from tree addresses to symbols, such that the functions are

Prefix closed: For any tree T, if $T(p \cdot i)$ is defined then $T(p)$ is defined.

Left closed: For any tree T, if $T(p \cdot i)$ is defined and $i > 1$ then $T(p \cdot (i-1))$ is defined.

We will refer to the domain of a tree T, the tree addresses for which T is defined, as the *nodes* of T. A node p of T is a *frontier node* if $T(p \cdot i)$ is undefined for all i. A node of T is an *interior node* if it is not a frontier node. We say that a node p of T is labeled with a symbol s if $T(p) = s$.

A.2 Tree-Adjoining Grammars and Derivations

A.2.1 Tree-Adjoining Grammars. In the following definitions, we restrict attention to tree-adjoining grammars in which adjunction is the only operation; substitution is not allowed. The definitions are, however, easily augmented to include substitution. We define a *tree-adjoining grammar* to be given by a quintuple $\langle \Sigma, N, \mathcal{I}, \mathcal{A}, S \rangle$ where

- Σ is a finite set of *terminal symbols*.
- N is a finite set of *nonterminal symbols* disjoint from Σ.
- ($V = \Sigma \cup N$ is the *vocabulary* of the grammar.)
- S is a distinguished nonterminal symbol, the *start symbol*.
- $\mathcal{I}$ is a finite set of trees, the *initial trees*, where
  - — interior nodes are labeled by nonterminal symbols, and
  - — frontier nodes are labeled by terminal symbols or the special symbol ϵ. (We require that $\epsilon \notin V$, as ϵ intuitively specifies the empty string.)
- $\mathcal{A}$ is a finite set of trees, the *auxiliary trees*, where
  - — interior nodes are labeled by nonterminal symbols, and
  - — frontier nodes are labeled by terminal symbols or ϵ, except for one node, called the *foot node*, which is labeled with a nonterminal symbol.
- ($\mathcal{E} = \mathcal{I} \cup \mathcal{A}$ is the set of *elementary trees* of the grammar.)

By convention, the address of the foot node of a tree A is notated f_A.

A.2.2 Adjunction. The adjunction of an auxiliary tree A at address t in tree E notated $E[A/t]$ is defined to be the smallest (least defined) tree T such that

$$T(r) = \begin{cases} E(r) & \text{if } t \not\preceq r & (1) \\ A(u) & \text{if } r = t \cdot u \text{ and } f_A \not\prec u & (2) \\ E(t \cdot u) & \text{if } r = t \cdot f_A \cdot u & (3) \end{cases}$$

These cases are disjoint except at addresses t and $t \cdot f_A$. We have

$$T(t) = E(t)$$

by clause (1), and

$$T(t) = A(t)$$

by clause (2). Similarly, we have

$$T(t \cdot f_A) = A(f_A)$$

by clause (2) and

$$T(t \cdot f_A) = E(t)$$

by clause (3). So for an adjunction to be well defined, it must be the case that

$$E(t) = A(t) = A(f_A)$$

that is, the node at which adjunction occurs must have the same label as the root and foot of the auxiliary tree adjoined. This is, of course, standard in definitions of TAG.

Alternatively, this constraint can be added as a stipulation and the definition modified as follows:

$$T(r) = \begin{cases} E(r) & \text{if } t \not\preceq r \\ A(u) & \text{if } r = t \cdot u \text{ and } f_A \not\preceq u \\ E(t \cdot u) & \text{if } r = t \cdot f_A \cdot u \end{cases}$$

We will use this latter definition below.

A.2.3 Ordered Derivation Trees. Ordered derivation trees are ordered trees composed of nodes, conventionally notated as η, possibly in its subscripted and primed variants. (For ordered derivation trees, we will be less formal as to their mathematical structure. In particular, the formalization of the previous section need not apply; the definitions that follow define all of the structure that we will need.) The parent of a node η in a derivation tree will be written $parent(\eta)$, and the tree in $\mathcal{E}$ that the node marks adjunction of will be notated $tree(\eta)$. The tree $tree(\eta)$ is to be adjoined into its parent $tree(parent(\eta))$ at an address specified on the arc in the tree linking the two; this address is notated $addr(\eta)$. (Of course the root node has no parent or address; the $parent$ and $addr$ functions are partial.)

An ordered derivation tree is well-formed if for each arc in the derivation tree from η to $parent(\eta)$ labeled with $addr(\eta)$, the tree $tree(\eta)$ is an auxiliary tree that can be adjoined at the node $addr(\eta)$ in $tree(parent(\eta))$.

We repeat from Section 4.1 the definition of the function $\mathcal{D}$ from derivation trees to the derived trees they specify, in the notation of this appendix:

$$\mathcal{D}(D) = \begin{cases} tree(\eta) \quad \text{if } D \text{ is a trivial tree of one node } \eta \\ tree(\eta)[\mathcal{D}(D_1)/t_1, \mathcal{D}(D_2)/t_2, \ldots, \mathcal{D}(D_k)/t_k] \\ \qquad \text{if } D \text{ is a tree with root node } \eta \\ \qquad\quad \text{and with } k \text{ child subtrees } D_1, \ldots, D_k \\ \qquad\quad \text{whose arcs are labeled with addresses } t_1, \ldots, t_k. \end{cases}$$

As in Section 4.1, $E[A_1/t_1, \ldots, A_k/t_k]$ specifies the simultaneous adjunction of trees A_1 through A_k at t_1 through t_k, respectively, in E. It is defined as the iterative adjunction of the A_i in order at their respective addresses, with appropriate updating of the tree addresses of later adjunctions to reflect the effect of earlier adjunctions. In particular, the following inductive definition suffices; the base case holds for the adjunction of zero auxiliary trees.

$$\begin{aligned}
&E[\,] = E \\
&E[A_1/t_1, A_2/t_2, \ldots, A_k/t_k] \\
&\qquad = (E[A_1/t_1])[A_2/update(t_2, A_1, t_1), \ldots, A_k/update(t_k, A_1, t_1)]
\end{aligned}$$

where

$$update(s, A, t) = \begin{cases} s & \text{if } t \not\prec s \\ t \cdot f_A \cdot (s - t) & \text{if } t \prec s \end{cases}$$

In the following section, we leave out parentheses in specifying sequential adjunctions such as $(E[A_1/t_1])[A_2/t_2]$ under a convention of left associativity of the $[\_/\_]$ operator.

A.3 Effect of Sibling Swaps

In this section, we show that the derived tree specified by a given ordered derivation tree is unchanged if adjacent siblings whose arcs are labeled with different tree addresses are swapped. This will be shown as the following proposition.

Proposition
If $t \neq t'$ then $E[\ldots, A/t, B/t', \ldots] = E[\ldots, B/t', A/t, \ldots]$.

We start with a lemma, the case for only two adjunctions.

Lemma
If $t \neq t'$ then $E[A/t, B/t'] = E[B/t', A/t]$.

Proof
There are three major cases, depending on the relationship of t and t':

Case $t \prec t'$: Let $s = t' - t$. Then

$$\begin{aligned}
E[A/t, B/t'](r) &= E[A/t][B/update(t', A, t)](r) \\
&= E[A/t][B/t \cdot f_A \cdot s](r) \\
&= \begin{cases} E[A/t](r) & \text{if } t \cdot f_A \cdot s \not\preceq r \\ B(u) & \text{if } r = t \cdot f_A \cdot s \cdot u \text{ and } f_B \not\preceq u \\ E[A/t](t \cdot f_A \cdot s \cdot u) & \text{if } r = t \cdot f_A \cdot s \cdot f_B \cdot u \end{cases} \\
&= \begin{cases} E(r) & \text{if } t \cdot f_A \cdot s \not\preceq r \text{ and } t \not\preceq r \\ A(v) & \text{if } t \cdot f_A \cdot s \not\preceq r \text{ and } r = t \cdot v \\ E(t \cdot v) & \text{if } t \cdot f_A \cdot s \not\preceq r \text{ and } r = t \cdot f_A \cdot v \\ B(u) & \text{if } r = t \cdot f_A \cdot s \cdot u \text{ and } f_B \not\preceq u \\ E(t \cdot s \cdot u) & \text{if } r = t \cdot f_A \cdot s \cdot f_B \cdot u \end{cases} \\
&= \begin{cases} E(r) & \text{if } t \not\preceq r \\ A(v) & \text{if } r = t \cdot v \\ E(t \cdot v) & \text{if } s \not\preceq v \text{ and } r = t \cdot f_A \cdot v \\ B(u) & \text{if } r = t \cdot f_A \cdot s \cdot u \text{ and } f_B \not\preceq u \\ E(t \cdot s \cdot u) & \text{if } r = t \cdot f_A \cdot s \cdot f_B \cdot u \end{cases}
\end{aligned}$$

If siblings are swapped,

$$\begin{aligned}
E[B/t', A/t](r) &= E[B/t'][A/update(t, B, t')](r) \\
&= E[B/t'][A/t](r) \\
&= \begin{cases} E[B/t \cdot s](r) & \text{if } t \not\preceq r \\ A(v) & \text{if } r = t \cdot v \text{ and } f_A \not\preceq v \\ E[B/t \cdot s](t \cdot v) & \text{if } r = t \cdot f_A \cdot v \end{cases} \\
&= \begin{cases} E(r) & \text{if } t \not\preceq r \\ A(v) & \text{if } r = t \cdot v \\ E(t \cdot v) & \text{if } r = t \cdot f_A \cdot v \text{ and } t \cdot s \not\preceq t \cdot v \\ B(u) & \text{if } r = t \cdot f_A \cdot v \text{ and } t \cdot v = t \cdot s \cdot u \text{ and } f_B \not\preceq u \\ E(t \cdot s \cdot u) & \text{if } r = t \cdot f_A \cdot v \text{ and } t \cdot v = t \cdot f_B \cdot u \end{cases} \\
&= \begin{cases} E(r) & \text{if } t \not\preceq r \\ A(v) & \text{if } r = t \cdot v \\ E(t \cdot v) & \text{if } s \not\preceq v \text{ and } r = t \cdot f_A \cdot v \\ B(u) & \text{if } r = t \cdot f_A \cdot s \cdot u \text{ and } f_B \not\preceq u \\ E(t \cdot s \cdot u) & \text{if } r = t \cdot f_A \cdot s \cdot f_B \cdot u \end{cases}
\end{aligned}$$

Case $t' \prec t$: Analogously.

Case $t \not\prec t'$ and $t' \not\prec t$:

$$\begin{aligned}
E[A/t, B/t'](r) &= E[A/t][B/update(t', A, t)](r) \\
&= E[A/t][B/t'](r) \\
&= \begin{cases} E[A/t](r) & \text{if } t' \not\preceq r \\ B(u) & \text{if } r = t' \cdot u \text{ and } f_B \not\preceq u \\ E[A/t](t' \cdot u) & \text{if } r = t' \cdot f_B \cdot u \end{cases} \\
&= \begin{cases} E(r) & \text{if } t' \not\preceq r \text{ and } t \not\preceq r \\ A(v) & \text{if } t' \not\preceq r \text{ and } r = t \cdot v \text{ and } f_A \not\preceq v \\ E(t \cdot v) & \text{if } t' \not\preceq r \text{ and } r = t \cdot f_A \cdot v \\ B(u) & \text{if } r = t' \cdot u \text{ and } f_B \not\preceq u \\ E(t' \cdot u) & \text{if } r = t' \cdot f_B \cdot u \end{cases}
\end{aligned}$$

Note that this is unchanged (up to variable renaming) under swapping of A for B and t for t'. That is $E[A/t, B/t'](r) = E[B/t', A/t](r)$. □

We now return to the main proposition.

Proposition
If $t \neq t'$ then $E[\ldots, A/t, B/t', \ldots] = E[\ldots, B/t', A/t, \ldots]$.

Proof
The effect of the adjunctions before the two specified in the swap is obviously the same on all following adjunctions, so we need only show that

$$E[A/t, B/t', C_1/t_1, \ldots, C_k/t_k] = E[B/t', A/t, C_1/t_1, \ldots, C_k/t_k]$$

without loss of generality. We examine the effect of the A and B adjunctions on the tree address t_i for each C_i separately. In the case of the former adjunction order

$$\begin{aligned}
&E[A/t, B/t', \ldots, C_i/t_i, \ldots] \\
&\quad = E[A/t][B/update(t', A, t), \ldots, C_i/update(t_i, A, t), \ldots] \\
&\quad = E[A/t][B/update(t', A, t)][\ldots, C_i/update(update(t_i, A, t), B, update(t', A, t)), \ldots] \\
&\quad = E[A/t, B/t'][\ldots, C_i/update(update(t_i, A, t), B, update(t', A, t)), \ldots]
\end{aligned}$$

and for the latter adjunction order:

$$
\begin{aligned}
&E[B/t', A/t, \ldots, C_i/t_i, \ldots] \\
&\quad = E[B/t'][A/update(t, B, t'), \ldots, C_i/update(t_i, B, t'), \ldots] \\
&\quad = E[B/t'][A/update(t, B, t')][\ldots, C_i/update(update(t_i, B, t'), A, update(t, B, t')), \ldots] \\
&\quad = E[B/t', A/t][\ldots, C_i/update(update(t_i, B, t'), A, update(t, B, t')), \ldots] \\
&\quad = E[A/t, B/t'][\ldots, C_i/update(update(t_i, B, t'), A, update(t, B, t')), \ldots]
\end{aligned}
$$

This last step holds by virtue of the lemma.

Thus, it suffices to show that

$$update(update(t_i, A, t), B, update(t', A, t)) = update(update(t_i, B, t'), A, update(t, B, t'))$$

Again, we perform a case analysis depending on the prefix relationships of t, t', and t_i. Note that we make use of the fact that if $t \prec t'$ then $(t' - t) \cdot s = t' \cdot s - t$.

Case $t \prec t'$:

Subcase $t' \prec t_i$:

$$
\begin{aligned}
&update(update(t_i, A, t), B, update(t', A, t)) \\
&\quad = update(t \cdot f_A \cdot (t_i - t), B, t \cdot f_A \cdot (t' - t)) \\
&\quad = t \cdot f_A \cdot (t' - t) \cdot f_B \cdot (t_i - t') \\
&\quad = t \cdot f_A \cdot (t' \cdot f_B \cdot (t_i - t') - t) \\
&\quad = update(t' \cdot f_B \cdot (t_i - t'), A, t) \\
&\quad = update(update(t_i, B, t'), A, update(t, B, t'))
\end{aligned}
$$

Subcase $t' \not\prec t_i$ and $t \prec t_i$:

$$
\begin{aligned}
&update(update(t_i, A, t), B, update(t', A, t)) \\
&\quad = update(t \cdot f_A \cdot (t_i - t), B, t \cdot f_A \cdot (t' - t)) \\
&\quad = t \cdot f_A \cdot (t_i - t) \\
&\quad = update(t_i, A, t) \\
&\quad = update(update(t_i, B, t'), A, update(t, B, t'))
\end{aligned}
$$

Subcase $t' \not\prec t_i$ and $t \not\prec t_i$:

$$
\begin{aligned}
&update(update(t_i, A, t), B, update(t', A, t)) \\
&\quad = update(t_i, B, t \cdot f_A \cdot (t' - t)) \\
&\quad = t_i \\
&\quad = update(t_i, A, t \cdot f_B \cdot (t' - t)) \\
&\quad = update(update(t_i, B, t'), A, update(t, B, t'))
\end{aligned}
$$

Case $t' \prec t$: The proof is as for the previous case with t for t' and vice versa.

Case $t \not\prec t'$ and $t' \not\prec t$:

Subcase $t \prec t_i$: We can conclude from the assumptions that $t' \not\prec t_i$. Then

$$
\begin{aligned}
&update(update(t_i, A, t), B, update(t', A, t)) \\
&\quad = update(t \cdot f_A \cdot (t_i - t), B, t') \\
&\quad = t \cdot f_A \cdot (t_i - t) \\
&\quad = update(t_i, A, t) \\
&\quad = update(update(t_i, B, t'), A, update(t, B, t'))
\end{aligned}
$$

References

Subcase $t \not\prec t_i$ **and** $t' \prec t_i$**:** The proof is as for the previous subcase with t for t' and vice versa.

Subcase $t \not\prec t_i$ **and** $t' \not\prec t_i$**:**

$$
\begin{aligned}
& update(update(t_i, A, t), B, update(t', A, t)) \\
& \quad = update(t_i, B, t') \\
& \quad = t_i \\
& \quad = update(t_i, A, t) \\
& \quad = update(update(t_i, B, t'), A, update(t, B, t'))
\end{aligned}
$$

□

An Alternative Conception of Tree-Adjoining Derivation

Yves Schabes and Stuart M. Shieber

Version of August 26, 1993