Tree-Adjoining Grammars for Optimality Theory Syntax

Virginia Savova
Department of Cognitive Science
Johns Hopkins University
savova@jhu.edu

Robert Frank
Department of Cognitive Science
Johns Hopkins University
rfrank@jhu.edu

Abstract

This paper explores an optimality-theoretic approach to syntax based on Tree-Adjoining Grammars (TAG), where two separate optimizations are responsible for the construction of local pieces of tree structure (elementary trees) and the combination of these pieces of structure. The *local* optimization takes a non-recursive predicate-argument structure (PA-chunk) as an underlying representation and chooses the best tree structure realizing it. The *linking* optimization takes as an underlying representation a tree whose nodes are labeled by PA-chunks and chooses among a set of structurally isomorphic TAG derivation trees. We provide formal definitions of the OTAG system and prove equivalence in strong generative capacity between OTAG and TAG. Finally, we apply the mechanics of the formal system to the analysis of cross-serial dependencies in Swiss-German.

1 Introduction

Optimality Theory (OT) claims that linguistic expressions are restricted by a set of universal, mutually inconsistent and violable constraints (Prince and Smolensky, 1993). Conflicts result in the satisfaction of higher ranked constraints at the expense of their lower ranked adversaries. The variations among languages are attributed to differences in the constraint rankings. In OT, a grammatical linguistic expression is a winner of an optimization. Given an underlying representation (UR), a generator function (Gen) produces a (potentially infinite) set of surface realizations (SRs), and a process of optimization picks the SRs that minimally violate the constraints according to a language-particular ranking.

OT is a general framework that can give rise to a variety of specific formal instantiations depending on the types of representations and constraints invoked, but it is a largely unresolved question just what sort of formalism is appropriate for OT syntax. Since natural language syntax permits recursively embedded structures, this suggests that the OT optimizations ought to apply to unbounded domains. However, optimization over such structures can give rise to a system with excessive generative capacity, if the number of violations of a constraint can grow without bound as well (Frank and Satta, 1998; Wartena, 2000). Moreover, if we look at the properties of natural language syntax, it appears that the structural tradeoffs that arise from the resolution of constraint conflict take place over local domains.

We therefore propose an OT formalism based on Tree Adjoining Grammar, which we call Optimality Tree Adjoining Grammar (OTAG), where separate optimizations are responsible for the construction of local pieces of tree structure (elementary trees) and the combination of these pieces of structure.The first optimization (which we call *local optimization*) takes as UR a non-recursive predicate argument structure (PA-chunk) and chooses among a set of local trees generated by Gen as candidate SRs of this PA-chunk. The local optimization yields a finite tree language which serves as a set of elementary trees. The second type of optimization (which we refer to as *linking optimization*) takes as UR a tree whose nodes are labeled by PA-chunks (a derivation tree of sorts) and chooses among a set of structurally isomorphic TAG derivation trees, where each node in these trees is labeled by an elementary tree that is among the locally optimal outputs for the corresponding PA-chunk.

2 Definitions

Let us begin with a formal definition of an OT system, adapted from (Frank and Satta, 1998).

Def. 1 *An* optimality system *is a 4-tuple* $OS =$

$\{\Sigma, \Gamma, Gen, C\}$ *where* Σ *and* Γ *are the finite input and output alphabets, Gen is a relation over* $\Sigma^* \times \Gamma^*$*, and* C *is a finite set of total functions from* $\Sigma^* \times \Gamma^*$ *to* N.

As seen in this definition, Gen maps a UR to a set of SRs, while a constraint is a function from a candidate UR-SR pair to a natural number, which we take to represent the degree of violation incurred by that candidate on that constraint. An OS gives rise to a set of optimality grammars (OG), defined in (2):

Def. 2 *An* optimality grammar *OG is an OS together with a total ordering* R *on* C*, called a* ranking.

Frank and Satta's definition is not directly applicable to OT syntax because it defines the URs and the SRs as strings. We assume that in syntax, the SRs are trees, while the URs are predicate-argument (PA) structures in tree form. A PA structure may contain simple and nested predicates. A simple predicate is a predicate applied over atomic arguments, i.e., arguments that do not contain predicates, as in example (1).

(1) loves(John, Mary)

A nested predicate is a predicate applied to other predicates, like *says* in example (2).

(2) says(Bill, (loves(John, Mary)))

We postulate a grammatical component, the $\mathbf{PA-chunker}$, which breaks down a complex PA structure into simple PA structures by substituting non-atomic arguments with predicate labels, which are treated as atomic arguments in the local optimization.

Def. 3 *A* PA-chunker *is a function from a nested PA structure* P *to a set of pairs containing a simple PA-structure (PA-chunks)* S *and a label* l *for that structure, such that*

i. each predicate in P *is a predicate in exactly one of the PA-chunks in* S*;*

ii. the atomic arguments of each predicate in P *are the same as the arguments of that predicate in corresponding PA-chunk in* S*; and*

iii. each complex argument A *of a predicate* π *in* P *is replaced in the PA-chunk containing* π *in* S *by the label uniquely associated with the simple PA-chunk in* S *corresponding to* A.

For example, the nested PA structure in (2) will give rise to the set of simple PA structures in (3) (where X and Y are predicate labels).

(3) {([says (Bill, X)], Y), ([loves (John, Mary)], X)}

In our setting, PA-chunks are the URs for optimizations over bounded domains whose outputs are local trees. The URs for optimizations over unbounded domains are tree structures over nodes labeled by a PA-chunk and all winning surface realizations of that PA-chunk (in the form of syntactic trees).

With this in mind, we define Optimality Tree Adjoining Systems (OTAS) as follows:

Def. 4 *An* Optimality Tree Adjoining System *is a 9-tuple*
$OTAS = \{\Sigma, \Gamma, \Pi, Chunk, Loc, Gen_C, Gen_K, C, K\}$ *where*

- Σ *and* Γ *are finite input and output alphabets;*
- Π *is a set of predicate labels;*
- $Chunk$ *and* Loc *are finite sets of finite trees labeled by* $\Sigma \cup \Pi$ *and* Γ *respectively;*
- Gen_C *is a relation over* $Chunk \times Loc$*;*
- Gen_K *is a relation over* $\Psi \times \Xi$*, where*
  - *i.* Ψ *is the set of finite trees each of whose nodes are labeled by members of* $Chunk \times \Pi \times 2^{Loc}$ *where for each* $\tau \in \Psi$*, a node labeled* (σ, π, γ) *is a daughter of node* (σ', π', γ') *iff* σ' *contains label* π*;*
  - *ii.* Ξ *is the set of finite trees labeled by* Γ*;*
- C *is a finite set of total functions from* $Chunk \times Loc$ *to* N*;*
- K *is a finite set of total functions from* $\Psi \times \Xi$ *to* N *(with* Ψ *and* Ξ *defined as above).*

The alphabets Σ and Γ are the sets of symbols in the representations making up the UR and SR, respectively. In our current conception, Σ consists of the set of predicate and argument symbols, while Γ contains the set of terminal and non-terminal symbols.[1] $Chunk$ will contain the set of URs that feed the local optimization, the set of PA-chunks, while Loc contains the SRs that can be the output of this process, the possible syntactic realizations of the PA-chunks. Gen_C maps a PA-chunk $\sigma \in Chunk$ to corresponding SR $\gamma \in Loc$. Gen_K maps any tree structure whose nodes are labeled by (local-UR, pred-label, locally-optimal-SRs) triples to a recursive surface tree realization. C is the set of constraints on local trees, while K is the constraints over recursive trees.[2] According to definition (2), an OT grammar is obtained by imposing a unique ranking on the set of constraints. In OTAG, a ranking must be specified for each type of optimization.

Def. 5 *An* OTAG Grammar *(OTG) is an OTAS with a pair of rankings* R_C, R_K *on* C *and* K.

[1]To keep things relatively simple, our definition neither enforces the arity requirements of predicate symbols nor the proper placement of predicate labels, terminal and non-terminal symbols in building members of $Chunk$ or Loc.

[2]Note that we are assuming that set of possible realizations of a member of $Chunk$ is finite. This is reasonable under the assumption that there is a finite set of winners for each optimization.

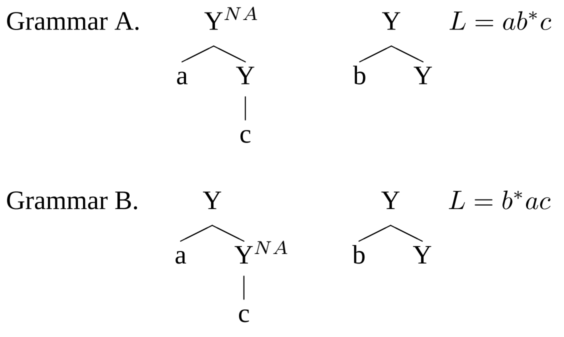

Figure 1: Related TAG grammars.

With these definitions in place, we can now define the notion of optimization in an OTG. Let us begin with local optimization:

Def. 6 *The* local optimum, $LOpt(p)$, *associated with a simple predicate argument structure* p *is defined recursively, as in (Frank and Satta, 1998):*

$$LOpt^i(p) = \begin{cases} Gen_C(p) & \textit{if } i = 0; \\ argmin_{c_i}(LOpt^{i-1}(p)) & \textit{if } i \geq 1 \end{cases}$$

$$LOpt(p) = LOpt^m(p) \textit{ where } m = |C|$$

Given such a set of local optima, we can now define the linking optimization process. Assume that we have a recursive predicate argument structure Π. The input to the linking optimization is a tree whose labels are taken from the following set of locally optimal pairings:

$$\Lambda = \{(p, \pi, LOpt(p)) | (p, \pi) \in \text{ PA-Chunk}(p)\}$$

Given such a Λ, there will be a unique tree τ such that (p, π, γ) is a daughter of node (p', π', γ') iff p' contains predicate label π. Linking optimization is now defined over this τ as in definition 6, using Gen_K and constraint set K.

3 Substitution, adjoining and the Linking Optimization

In traditional TAG, grammars sharing the same set of local trees can generate different languages. An example of this situation is depicted in Figure 1, where we see two grammars that differ only in the locus of adjoining constraints and generate distinct languages. Since the linking optimization in OTAG constrains how the elementary trees that result from the local optimization are put together, the languages of these grammars could also generated by two OTAGs derived from the same OTAG system with different constraint rankings (Figure 2).

The constraints on adjoining are implemented in the set of violable constraints K, which prohibit or require adjoining at a set of nodes. In the grammar illustrated here, C_1 requires some adjoining to take place, C_2 forbids adjoining at the root Y node of the *ac* elementary tree, and C_3 forbids adjoining at the lower Y node of the same tree. When C_1 is ranked above either or both of C_2 or C_3, the higher ranked of this latter pair of constraints determines where adjoining applies, whereas when C_1 is lowest ranked, no adjoining takes place at all. Constraint reranking, then, achieves the effect of altering the loci of adjoining constraints. In principle, the linking optimization may apply globally, evaluating the whole UR against a derivation, but that would lead to the possibility of conditioning an adjunction at high levels on lower level adjunctions. In order to limit the generative power of OTAG, we require that the linking optimization apply cyclically. Each cycle adjoins a set of auxiliary trees into a single local tree, and these cycles proceed in a bottom-up fashion through the PA-chunk structure that is the input to the linking optimization. The result of a linking optimization may be used for a subsequent cycle, when a derived auxiliary is adjoined. This constraint enforces a strong parallelism between the OTAG derivation and the TAG derivation. They differ only by the presence of an optimization step in OTAG, which determines where the auxiliary tree is adjoined into another elementary tree. In other words, an OTAG derivation tree represents a series of optimal adjoining operations.

With this restriction in place, it turns out that the resulting formalism is exactly as powerful as the TAG formalism. Specifically, we can prove the following theorems (see appendix for proofs):

Theorem 1 *For any TAG G, there is a OTAG G' such that T(G) = T(G').*

Theorem 2 *For any OTAG G', there is a TAG G such that T(G') = T(G).*

4 OTAG in action: An illustrative example

To illustrate the practical application of the formalism, we will go through the steps of a derivation of the Swiss-German cross-serial construction, and the corresponding

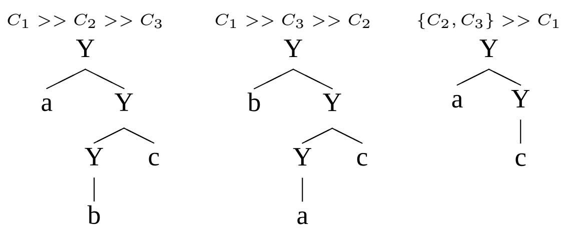

Figure 2: Output of OTAG grammars that differ only in constraint ranking.

German and English constructions. Swiss-German exhibits cross-serial dependencies that can be modeled by the language $LCross = a^n b^m c^n d^m | m, n \in N$ (Shieber, 1985).

(4) De Jan säit, dass mer em Hans es huus
John-NOM says that we Hans-DAT the house
hälfed aastriiche. (Swiss German)
helped paint
'John says that we helped Hans paint the house.'

Compare this to the English and German equivalents.

(5) John says that we helped Hans paint the house.

(6) Jan sagt, daß wir Hans das Haus
John says that we Hans the house-Acc
anstreichen hilften. (German)
paint helped
'John says that we help Hans paint the house.'

The German sentence exhibits center embedding - the innermost verb case-marking the innermost noun, the outermost verb case-marking the outermost noun. In the English case, there is no embedding at all: verbs always immediately precede their associated arguments.

Let us consider the necessary steps in an OTAG analysis of these data. First, we must isolate the local winners. As we know, they are SRs corresponding to PA-chunks. Table 4 shows the simple predicates and the corresponding yield of the local winners in English, German, and Swiss-German. The symbol _ marks the insertion site for the other SR. The question we need to tackle is what kind of trees yield these strings. We notice that the German and Swiss-German cases differ from the English case by the position of the verb with respect to its arguments. One way to account for this difference would be to invoke a Headedness constraint on the local trees, Head-Left, and a counter-constraint, e.g., Head-Right. We also invoke a local Markedness constraint such as "Move V" which conflicts with a Faithfulness constraint "*trace" (a.k.a. "Stay!", cf. Grimshaw1977). These constraints are defined as follows:

- Move V: Raise V to T.
- *trace: No traces.

In German, unlike English, "*trace" is ranked lower than "Move V". Note that the overt difference between English and German can be explained by assuming the verb *help* raises to node Y, without assuming anything about the verb *paint*. However, our OTAG analysis forces us to make a theoretical commitment that *paint* also raises, since the tree it is part of is a winner of a local optimization under the same constraint hierarchy.

We can now characterize the Swiss-German case in a way consistent with our theory of the English and German cases. At this point, we are going to make use of the

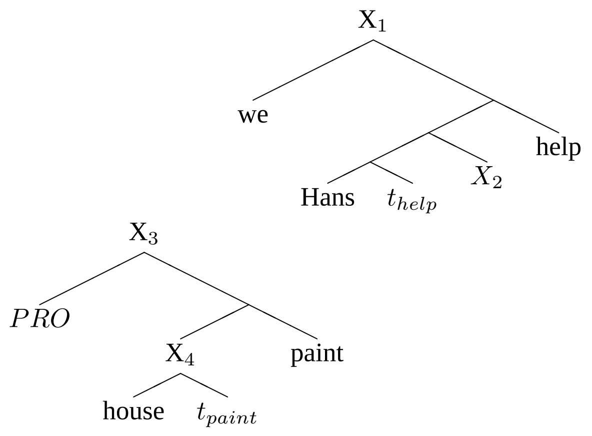

Figure 3: Adjoining occurs at X_4 in Swiss-German, X_3 in German

linking optimization to distinguish German from Swiss-German in particular. Descriptively, Swiss-German differs from German by the fact that *help* intervenes between *paint* and its argument. This is exactly what we expect if we assume that adjoining in Swiss-German takes place at a lower node than adjoining in German. In the analysis of English and German, the node X_3 was the adjoining site. By supposing that instead, the adjoining site for Swiss-German is X_4, we obtain the desired cross-serial dependency. To enforce this difference in adjoining sites, we need to postulate two constraints that play a role in the linking optimization by favoring nodes X_3 and X_4, respectively. A linguistically motivated constraint favoring X_3 may be related to the relationship between Hans and PRO resulting from the adjoining. In English and German, but not in Swiss-German, Hans c-commands PRO in the output of the linking optimization. Another plausible constraint is a subcategorization constraint on the adjoining tree. Suppose the adjoining tree is of type A and node X_3 is of a particular type N. Thus, the linking optimization may involve a constraint "C-PRO: PRO must be c-commanded" and a constraint "A-to-N: Adjoin trees of type A to nodes of type N" ranked differently with respect to each other. In our case, let us suppose "trees of type A" means "Auxiliary trees of type VP" and "Nodes of type N" means "Highest VP node of initial tree." To recount, here is how our model analysis would play out. Table 4 presents the local optimizations with candidate structures, including the winners for English (E), German (G) and Swiss-German (SG).

Note that at this point the local optimization contains two constraints more than necessary to account for the data. We can prune the analysis by removing any pair of constraints that favor opposite candidates. For example,

PA-chunks	English	German	Swiss-German
$([paint(Hans, house)], X)$	paint the house	das Haus anstreichen	es huus aastriiche
helped(we, Hans, X)	We helped Hans _	wir Hans _ hilften	mer em Hans_ hälfed _

Table 1: PA-chunks

(paint(Hans, house)],X**)**	Head-Left	Head-Right	*trace	Move V
E: [PRO [paint house]]		*		*
G, SG:[PRO [[t_{paint} house] paint]]	*		*	
help(we, Hans, X)				
E: [we [[help Hans]]]		*		*
G, SG:[we [[[t_{help} Hans]] help]]	*		*	

Table 2: Local optimizations

paint(Hans, house)	Head-Left	Head-Right
E: [PRO [paint [t_{paint} house]]]	*	
G, SG: [PRO [[t_{paint} house] paint]]		*
help(we, Hans, X**)**		
E: [we [help [t_{help} Hans]]]	*	
G, SG: [we [[[t_{help} Hans]] help]]		*

Table 3: Local optimization simplified

we have the option of scrapping either the pair Head-Left, Move V or the pair Head-Right, *trace from the constraint set. If we get rid of the former pair, we will essentially be claiming that movement of the verb happens in order to position the head to the right of the verb phrase. Alternatively, if we remove the latter constraint pair, we will be suggesting that movement of the verb can only happen to the right and hence necessarily violates Head-Left. There is no reason to dismiss either scenario right away. On the other hand, some new data might discredit either alternative and persuade us to keep all constraints in the set. Finally, a third scenario may involve obligatory verb movement in both English and German/Swiss-German. In this case, the only relevant players in the constraint set are Head-Left and Head-Right, which force the movement to take the preferred direction. The optimization would include only candidate representations in which movement has occurred (i.e. Loc would be restricted to such structures, Table 3).

Another issue in the local optimization is the realization of the argument "Hans" as PRO in one sentence, but as *Hans* in the other. This issue can only be solved by exploiting the possibility of multiple winners in the local optimizations. In other words PRO and the full argument must be indistinguishable from the point of view of the local optimization, but one or the other must be preferred in the linking optimization. The argument is simple. By virtue of our definition of the PA chunker, the predicate argument structure *paint(Hans, house)* is independent of the larger complex predicate it was embedded in. Consequently, the same predicate argument structure would qualify as an UR of *Hans paints the house* since the latter is a grammatical structure, Hans may equally surface as PRO or simply *Hans*. We need to update our Table once again by adding two more competitors, as shown in Table 4. This competition is resolved in the subsequent linking optimizations as seen in Table 5. The constraint "*Repeat" penalizes the repetition of a nominal element. Admittedly, this is a very crude way of enforcing the presence of PRO in the final structure. A more sophisticated way of defining *Repeat could refer to the relationship between trees with argument Arg in SpecVP on one hand, and trees with the same argument Arg in a complement position on the other. For example: *Repeat: Do not adjoin trees with complement Arg to trees with Arg in SpecVP This formulation is a better match for the type of constraints we have used in our formal treatment of OTAG so far.

The role of *Repeat here is to show how multiple winners in the local optimization allow us to sneak in solutions to differences in the form of main versus embedded clauses. Recall that, if the PA-chunker is only given the simple predicate argument structure to start with, the linking optimization will involve adjoining of the null tree. Consequently, "*Repeat" will not play a role, as shown in Table 6. At the same time, any constraint related to PRO would disadvantage PRO in this setting and the full argument would surface. This completes our illustrative

paint(Hans, house)	Head-Left	Head-Right
E: [PRO [paint [[t_{paint} house]]]	*	
G, SG: [PRO [[t_{paint} house] paint]]		*
E: [Hans [paint [t_{paint} house]]]	*	
G, SG: [Hans [[t_{paint} house] paint]]		*

Table 4: Full NP and PRO are tied in the local optimization

help(we, Hans, X). paint(Hans, house)	$C - PRO$	A-to-N	*Repeat
E: [we [[help Hans] [PRO [paint house]]]]		*	
G: [we [[[Hans t help] [PRO [[house t_{paint}] paint]]]help]]		*	
SG: [PRO [[we [[[Hans t_{help}] [house t_{paint}]] help]]paint]]	*		
* [we [[help Hans] [Hans [paint house]]]]		*	*
* [we [[[Hans t_{help}] [Hans [[house t_{paint}] paint]]]help]]		*	*
* [Hans [[we [[[Hans thelp] [house t_{paint}]]help]]paint]]		*	*

Table 5: Linking optimization licenses PRO in subordinate clause

$\emptyset$. paint(Hans, house)	$C - PRO$	*Repeat
* [PRO [paint house]]	*	
*[PRO [[house t_{paint}] paint]]	*	
E: [Hans [paint house]]		
G, SG: [Hans [[house t_{paint}] paint]]		

Table 6: Linking Optimization eliminates PRO in main clause

analysis of the Swiss-German construction and its cross-linguistic counterparts. The important points to remember are:

- When analyzing a complex structure, complex PA structures are broken into chunks.
- Predicate labels in the PA chunks constrain what adjoins into what in the linking optimization.
- Adjustments in the ranking among constraints in the local optimization permit different structural variants to win.
- Both main clause and the embedded clause variants of a PA chunk must be possible winners in the local optimization.
- If the embedded clause is not grammatical as a main clause, the linking optimization must include a constraint that favors the embedded clause over the main clause.

5 Conclusion

Our proposal is a step towards a restrictive and adequate framework for handling syntactic phenomena in the spirit of OT. We have demonstrated that the generative power of any grammar specified within the framework is limited to the class of MCSLs, which many believe is the complexity class of natural languages. The main theoretical advantage of the OTAG formalism is the locality imposed by the optimization over simple predicates in the first stage of the derivation of an arbitrarily complex structure. Another, more practical advantage stems from the relative transparency of the components of the framework. Our formalism relies on a specific kind of underlying representation, a specific way to handle recursion, and a general template for constraints. Clearly, further work is needed to test the viability of this framework for a broader range of empirical phenomena.

Acknowledgments

Special thanks to Paul Smolensky for helpful comments on an earlier draft. The research in this paper was supported in part by NSF grant SBR-9972807.

References

Robert Frank and Giorgio Satta. 1998. Optimality theory and the generative complexity of constraint violability. *Comput. Linguist.*, 24(2):307–315.

Alan Prince and Paul Smolensky. 1993. Optimality theory: Constraint interaction in generative grammar. Technical report, Rutgers University Center for Cognitive Science.

Stuart Shieber. 1985. Evidence against the contextfreeness of natural language. *Linguistics and Philosophy*, 8:333–343.

Christian Wartena. 2000. A note on the complexity of optimality systems. Ms. Universität Potsdam, Rutgers Optimality Archive (ROA-385-03100).

Appendix: Proofs of theorems

We define a TAG G as a tuple (A, I, R), where A is the set of auxiliary trees, I is the set of initial trees, and R is the set of adjoining constraints associated with nodes of $A \cup I$. We require that A contain a distinguished null auxiliary tree ϵ, capable of adjoining at any node. With such an ϵ tree, we can assume without loss of generality that every legal TAG/OTAG derivation involves adjoining to every node of every tree involved in the derivation. An adjoining constraint $r \in R$ specifies a set of trees S and a node d such that S cannot adjoin at d ($r = *S@d$). Such a constraint corresponds to the usual notion of selective adjoining constraint. Obligatory adjoining constraints can be modeled as a constraint which forbids adjoining of ϵ. Null adjoining constraints permit adjoining of only the tree ϵ. On the OTAG side, we will assume a constraint *NIL that penalizes SRs in the linking optimization in which trees present in the UR do not participate in the TAG derivation yielding the surface tree. Finally, we use the notation $T(G)$ to refer to the set of well-formed derivation trees in a TAG or OTAG.

Theorem 1. For any TAG G, there is a OTAG G' such that T(G) = T(G').

Given a TAG $G = (I, A, R)$, we define OTAG $G' = (\Sigma, \Gamma, \Pi, Chunk, Loc, C, K)$, such that $Loc = A \cup I$ and $K_{G'} = \{*NIL\} \cup \{k_r | r \in R\}$ where k_r penalizes a candidate if it involves an adjoining that would violate TAG adjoining constraint r.[3]

Claim 1 $D \in T(G) \rightarrow D \in T(G')$

Proof by induction on the depth of D (represented (D)):

Base case Let $(D) = 0 . D$ consists of a node t whose only children are instances of the empty tree ϵ. Let t be a tree with nodes $\{$1...n$\}$. $D \notin T(G')$ iff one of the following is true:

1. $\{t, \epsilon\} \notin Loc$. But ϵ is always in A. Moreover, $D \in T(G)$ by hypothesis, which is true only if $t \in A \cup I$. Since $Loc = A \cup I, t \in Loc$.
2. $\exists k_1 ... k_n \in K | k_i = *\{\epsilon\}@i$, for i a node $\in t$. This is true only if $\exists \{r_1 ... r_n\} \in R | r_i = *\{\epsilon\}@i$, i a node $\in t$. But if $\{r_1 ... r_n\} \in R$ was true $D \in T(G)$ would be false.

Hence $\{k_1 ... k_n\}$ do not exist and $D \in T(G')$

Induction hypothesis Suppose Claim 1 is true for all $(D) \leq k$. Let t be the root of D and $\{1...n\}$ the set of nodes in t. Let $\{D_1 ... D_n\}$ be a set of derivations with roots $\{a_1 ... a_n\} \in A$ such that a_i is adjoined to node i in t. Observe that $(D_i) \leq k$ for $1 \leq i \leq n$. $D \notin T(G')$ iff one of the following is true:

1. $t \notin Loc$. But $D \in T(G)$ by hypothesis, which is true only if $t \in A \cup I$. Since $Loc = A \cup I, t \in Loc$;
2. $\{D_1 ... D_n\} \notin T(G')$.But $\{D_1 ... D_n\} \in T(G')$ by the induction hypothesis;
3. $\exists k_i \in K | k_i = *a_i@i$. This is true only if $\exists r_i \in R | r_i = *a_i@i$. But if this were true, $D \in T(G)$ would be false.

Hence k_i do not exist and $D \in T(G')$

Claim 2 $W \in T(G') \rightarrow W \in T(G)$

Proof by induction on the depth of W:

Base case (W) = 0. W consists of one optimization adjoining the empty tree ϵ into some $w \in Loc$. $W \notin T(G)$ iff one of the following is true:

1. $w \notin A \cup I$. But $Loc = A \cup I$ and $w \in Loc$. Hence $w \in A \cup I$.
2. $\exists r_1 ... r_n \in R | r_i = *\{\epsilon\}@i$, for i a node $\in t$. This is true only if $\exists \{k_1 ... k_n\} \in K | k_i = *\{\epsilon\}@i$, for i a node $\in t$. But if $\{k_1 ... k_n\} \in K$ was true, $W \in T(G')$ would be false.

Hence $\{r_1 ... r_n\}$ do not exist and $W \in T(G)$

Induction hypothesis Suppose Claim 2 is true for any derivation W, $(W) \leq k$. Let w be the root of W and $\{1...n\}$ the set of nodes in w. Let $\{W_1 ... W_n\}$ be a set of derivations with roots $\{z_1 ... z_n\} \in Loc$ such that z_i is adjoined at node i. Observe that $(W_i) \leq k$ for all $1 \leq i \leq n$. $W \notin T(G)$ iff one of the following is true:

1. $w \notin A \cup I$. But $w \in T(G')$ by hypothesis, which is true only if $w \in Loc$. Since $Loc = A \cup I, w \in A \cup I$;
2. $\{W_1 ... W_n\} \notin T(G)$.But $\{W_1 ... W_n\} \in T(G)$ by hypothesis;
3. $\exists r_i \in R | r_i = *z_i@i$. This is true only if $\exists k_i \in K | k_i = *z_i@i$. But if $k_i \in K$ was true $W \in T(G')$ would be false.

Hence r_i do not exist and $W \in T(G)$. ■

Theorem 2. For any OTAG G', there is a TAG G such that T(G') = T(G).

Here, we will also give a general procedure for converting a OTAG into an equivalent TAG. Before we proceed, it would be useful to informally consider the two cases that cause complications in this conversion. Both cases are easily illustrated with a minimal OTAG. Suppose Loc contains only two trees: the initial tree t and the auxiliary tree a. In addition, let t contain only two nonterminal nodes (n1, n2). Case 1: Now suppose that the constraint set K of our OTAG G contains two OA constraints, k_1 and k_2, such that k_1 and k_2 require the adjoining of the same tree a at different nodes (n_1, n_2) of the tree t $(k_1 = *(A - a)@n_1; k_2 = *(A - a)@n_2;)$.

[3]We do not define $\Sigma, \Pi, Chunk$, or C since the there is no counterpart to the local optimization in TAG. Since the set of elementary trees is finite, we can assume the existence of some set of constraint C that will produce this set of trees from appropriate URs.

Furthermore, suppose $*Null >> k_1 >> k_2$. This constraint ranking would enforce the adjoining of a into $n_2(t)$ only if another instance of a is adjoined at $n_1(t)$. Case 2 is similar: Suppose that the constraint set K of our OTAG G contains two NA constraints, k_1 and k_2 against adjoining any auxiliary tree a at either one of two different nodes (n_1, n_2) of the same tree t. Furthermore, suppose $*Null >> k_1 >> k_2$. This constraint ranking would allow adjoining into $n_1(t)$ only if adjoining has taken place already at $n_2(t)$. It is clear from these cases that a simple translation of constraints into adjoining constraints is not sufficient. The violated OA constraint k_2 cannot be emulated by an OA constraint forcing a to adjoin at n_2 (because the adjoining fails when a is not adjoined at n_1); nor does it correspond to a SA constraint that merely allows adjoining of a at n_2 (because it the adjoining is obligatory whenever an instance of a is already adjoined at n_1). Thus, instead of picking a single type of constraint to place on each elementary tree, we need to multiply out the trees in Loc affected by problematic constraint sets of this type. The tree t corresponds to a subset of two trees in the elementary tree set of the corresponding TAG: One tree has an OA constraint on node n_1. The other has an NA constraint on node n_2. Similarly, the violated NA constraint k_2 cannot be emulated by a NA constraint against a on n_1 (because the adjoining could occur if an instance of a is already adjoined at n_2). Neither can it be completely disregarded, because it prevents a from adjoining into n_1 if a has not adjoined to n_2 beforehand. The tree t maps to a subset of two trees in the elementary tree set of the corresponding TAG: One tree has an NA constraint on n_1, the other has an OA constraint on n_2. Let G' be a $OTAG = \{\Sigma, \Gamma, \Pi, Chunk, Loc, Gen_C, Gen_K, C, K\}$ with rankings R_C and R_K. Then TAG $G = \{A, I, R\}$, obtained based on the outcome of all linking optimizations involving the adjoining of a set S of trees from Loc into some tree t in Loc (note that $|S| \leq$ the number of non-terminals in t).

Conversion algorithm:

Step 1: Create a table T_t of size $n \times p$ associated with each tree t in Loc, where n is the number of nodes in t, and p is the number of possible multisets of trees Z drawn from Loc of cardinality n. In each cell (j, k), enter all trees $z \in Z$ adjoined to node j in some linking optimization over Υ, where Υ is a UR tree whose nodes are labeled with triples $(\sigma_i, \pi, \gamma_i)$ and $\cup(\gamma_i) = k$.

Step 2: For every tree $t \in Loc$, create a set of elementary trees E_t containing distinct copies of t for each cell of T_t. For each such $t_{(i,j)} \in E_t$, create adjoining constraints $r = *A - T_t(i,j)@h$, where h is the name of the copy of node i in $t_{(i,j)}$.

Claim 1: $W \in T(G') \rightarrow W \in T(G)$
Proof by induction on depth of W.

Base case Let $(W) = 0$. W involves one optimization adjoining of only instances of the empty tree ϵ into some $w \in Loc$. $W \notin T(G)$ iff one of the following is true:

1. $w \notin A \cup I$. But $A \cup I \supseteq Loc$ and $w \in Loc$. Hence $w \in A \cup I$.
2. $\exists r_1...r_n \in R | r_i = *\{\epsilon\}@i$, i a node $\in t$. This is true only if ϵ never adjoins into w in the linking optimization of G'. But if this were the case, $W \in T(G')$ would be false.

Hence $\{r_1...r_n\}$ do not exist and $W \in T(G)$

Induction hypothesis Suppose Claim 1 is true for any derivation $(W) \leq k$. Let w be the root of W and $\{1...n\}$ the set of nodes in w. Let $\{W_1...W_n\}$ be a set of derivations,$(W_i) \leq k$ with roots $\{z_1...z_n\} \in Loc$ such that z_i is adjoined at node i. $W \notin T(G)$ iff one of the following is true:

1. $w \notin A \cup I$. But $w \in T(G')$ by hypothesis, which is true only if $w \in Loc$. Since $A \cup I$ contains copies of all the trees in $Loc, w \in A \cup I$;
2. $\{W_1...W_n\} \notin T(G)$. But $\{W_1...W_n\} \in T(G)$ by hypothesis;
3. $\exists r_i \in R | r_i = *z_i@i$. This is true only if G' disallows adjoining of z_i to i, in which case $W \in T(G')$ would be false.

Hence r_i do not exist and $W \in T(G)$.

Claim 2: $D \in T(G) \rightarrow D \in T(G')$
Proof by induction on depth of D:

Base case Let $(D) = 0$. D consists of a node t whose only children are the empty tree ϵ. Let t be a tree with nodes {1...n}. $D \notin T(G')$ iff one of the following is true:

1. $\{t, \epsilon\} \notin Loc$. But ϵ is always in A. Moreover, $D \in T(G)$ by hypothesis, which is true only if $t \in A \cup I$. Since $A \cup I$ contains only copies of trees in $Loc, t \in Loc$.
2. $\exists k_1...k_n \in K | k_i = *\{\epsilon\}@i$, i a node $\in t$. This is true only if $\exists \{r_1...r_n\} \in R | r_i = *\{\epsilon\}@i$, i a node $\in t$. But if $\{r_1...r_n\} \in R$ was true $D \in T(G)$ would be false.

Hence $\{k_1...k_n\}$ do not exist and $D \in T(G')$

Induction hypothesis Suppose Claim 1 is true for any $(D) \leq k$. Let t be the root of D and $\{1...n\}$ the set of nodes in t. Let $\{D_1...D_n\}$ be a set of derivations,$(D_i) \leq k$ with roots $\{a_1...a_n\} \in A$ such that a_i is adjoined to node i. $D \notin T(G')$ iff one of the following is true:

1. $t \notin Loc$. But $D \in T(G)$ by hypothesis, which is true only if $t \in A \cup I$. Since $A \cup I$ contains only copies of trees in Loc, $t \in Loc$;
2. $\{D_1...D_n\} \notin T(G')$.But $\{D_1...D_n\} \in T(G')$ by hypothesis;
3. $\exists k_i \in K | k_i = *a_i@i$. This is true only if $\exists r_i \in R | r_i = *a_i@i$. But if $r_i \in R$ was true $D \in T(G)$ would be false.

Hence k_i do not exist and $D \in T(G')$. ■