Constraints on strong generative power

David Chiang
University of Pennsylvania
Dept of Computer and Information Science
200 S 33rd St
Philadelphia, PA 19104 USA
dchiang@cis.upenn.edu

Abstract

We consider the question "How much strong generative power can be squeezed out of a formal system without increasing its weak generative power?" and propose some theoretical and practical constraints on this problem. We then introduce a formalism which, under these constraints, maximally squeezes strong generative power out of context-free grammar. Finally, we generalize this result to formalisms beyond CFG.

1 Introduction

"How much strong generative power can be squeezed out of a formal system without increasing its weak generative power?" This question, posed by Joshi (2000), is important for both linguistic description and natural language processing. The extension of tree adjoining grammar (TAG) to tree-local multicomponent TAG (Joshi, 1987), or the extension of context free grammar (CFG) to tree insertion grammar (Schabes and Waters, 1993) or regular form TAG (Rogers, 1994) can be seen as steps toward answering this question. But this question is difficult to answer with much finality unless we pin its terms down more precisely.

First, what is meant by strong generative power? In the standard definition (Chomsky, 1965) a grammar G weakly generates a set of sentences $L(G)$ and strongly generates a set of *structural descriptions* $\Sigma(G)$; the strong generative capacity of a formalism $\mathcal{F}$ is then $\{\Sigma(G) \mid \mathcal{F} \text{ provides } G\}$. There is some vagueness in the literature, however, over what structural descriptions are and how they can reasonably be compared across theories (Miller (1999) gives a good synopsis).

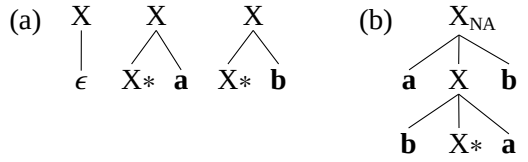


Figure 1: Example of weakly context-free TAG.

The approach that Vijay-Shanker et al. (1987) and Weir (1988) take, elaborated on by Becker et al. (1992), is to identify a very general class of formalisms, which they call linear context-free rewriting systems (CFRSs), and define for this class a large space of structural descriptions which serves as a common ground in which the strong generative capacities of these formalisms can be compared. Similarly, if we want to talk about squeezing strong generative power out of a formal system, we need to do so in the context of some larger space of structural descriptions.

Second, why is preservation of weak generative power important? If we interpret this constraint to the letter, it is almost vacuous. For example, the class of all tree adjoining grammars which generate context-free languages includes the grammar shown in Figure 1a (which generates the language $\{\mathbf{a}, \mathbf{b}\}^*$). We can also add the tree shown in Figure 1b without increasing the grammar's weak generative capacity; indeed, we can add any trees we please, provided they yield only **a**s and **b**s. Intuitively, the constraint of weak context-freeness has little force.

This intuition is verified if we consider that weak context-freeness is desirable for computational efficiency. Though a weakly context-free TAG might be *recognizable* in cubic time (if we know the equivalent CFG), it need not be *parsable* in cubic time—that is, given a string, to compute all its possible structural descriptions will take $O(n^6)$ time in general. If we are interested in computing structural descriptions from strings, then

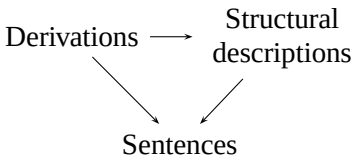


Figure 2: Simulation: structural descriptions as derived structures.

we need a tighter constraint than preservation of weak generative power.

In Section 3 below we examine some restrictions on tree adjoining grammar which are weakly context-free, and observe that their parsers all work in the same way: though given a TAG G, they implicitly parse using a CFG G' which derives the same strings as G, but also their corresponding structural descriptions under G, in such a way that preserves the dynamic-programming structure of the parsing algorithm.

Based on this observation, we replace the constraint of preservation of weak generative power with a constraint of *simulability*: essentially, a grammar G' simulates another grammar G if it generates the same strings that G does, as well as their corresponding structural descriptions under G (see Figure 2).

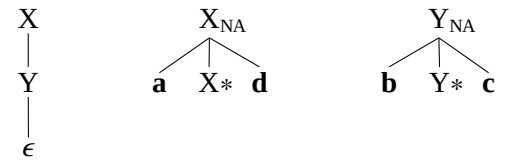
So then, within the class of context-free rewriting systems, how does this constraint of simulability limit strong generative power? In Section 4.1 we define a formalism called multicomponent multifoot TAG (MMTAG) which, when restricted to a regular form, characterizes precisely those CFRSs which are simulable by a CFG. Thus, in the sense we have set forth, this formalism can be said to squeeze as much strong generative power out of CFG as is possible. Finally, we generalize this result to formalisms beyond CFG.

2 Characterizing structural descriptions

First we define context-free rewriting systems. What these formalisms have in common is that their derivation sets are all local sets (that is, generable by a CFG). These derivations are taken as structural descriptions. The following definitions are adapted from Weir (1988).

Definition 1 A *generalized context-free grammar* G is a tuple $\langle V, S, F, P\rangle$, where

1. V is a finite set of variables,
2. $S \in V$ is a distinguished start symbol,
3. F is a finite set of function symbols, and



$$\begin{array}{cc} S \to \alpha(X,Y) & \alpha(\langle x_1,x_2\rangle,\langle y_1,y_2\rangle) = x_1y_1y_2x_2 \\ X \to \beta_1(X) & \beta_1(\langle x_1,x_2\rangle) = \langle \mathbf{a}x_1, x_2\mathbf{d}\rangle \\ X \to \epsilon() & \epsilon() = \langle \epsilon,\epsilon\rangle \\ Y \to \beta_2(Y) & \beta_2(\langle y_1,y_2\rangle) = \langle \mathbf{b}y_1, y_2\mathbf{c}\rangle \\ Y \to \epsilon() & \epsilon() = \langle \epsilon,\epsilon\rangle \end{array}$$

Figure 3: Example of TAG with corresponding GCFG and interpretation. Here adjunction at foot nodes is allowed.

4. P is a finite set of productions of the form

$$A \to f(A_1, \ldots, A_n)$$

where $n \geq 0$, $f \in F$, and $A, A_i \in V$.

A generalized CFG G generates a set $\mathcal{T}(G)$ of *terms*, which are interpreted as derivations under some formalism. In this paper we require that G be free of spurious ambiguity, that is, that each term be uniquely generated.

Definition 2 We say that a formalism $\mathcal{F}$ is a *context-free rewriting system* (CFRS) if its derivation sets can be characterized by generalized CFGs, and its derived structures are produced by a function $[\![\cdot]\!]_{\mathcal{F}}$ from terms to strings such that for each function symbol f, there is a *yield function* $f^{\mathcal{F}}$ such that

$$[\![f(t_1, \ldots, t_n)]\!]_{\mathcal{F}} = f^{\mathcal{F}}([\![t_1]\!]_{\mathcal{F}}, \ldots, [\![t_n]\!]_{\mathcal{F}})$$

(A *linear* CFRS is subject to further restrictions, which we do not make use of.)

As an example, Figure 3 shows a simple TAG with a corresponding GCFG and interpretation.

A nice property of CFRS is that any formalism which can be defined as a CFRS immediately lends itself to several extensions, which arise when we give additional interpretations to the function symbols. For example, we can interpret the functions as ranging over probabilities, creating a stochastic grammar; or we can interpret them as yield functions of another grammar, creating a synchronous grammar.

Now we define strong generative capacity as the relationship between strings and structural descriptions.[1]

[1]This is similar in spirit, but not the same as, the notion of derivational generative capacity (Becker et al., 1992).

Definition 3 The *strong generative capacity* of a grammar G a CFRS $\mathcal{F}$ is the relation

$$\{\langle [\![t]\!]_{\mathcal{F}}, t\rangle \mid t \in \mathcal{T}(G)\}.$$

For example, the strong generative capacity of the grammar of Figure 3 is

$$\{\langle \mathbf{a}^m\mathbf{b}^n\mathbf{c}^n\mathbf{d}^m, \alpha(\beta_1^m(\epsilon()), \beta_2^n(\epsilon()))\rangle\}$$

whereas any equivalent CFG must have a strong generative capacity of the form

$$\{\langle \mathbf{a}^m\mathbf{b}^n\mathbf{c}^n\mathbf{d}^m, f^m(g^n(e()))\rangle\}$$

That is, in a CFG the n **b**s and **c**s must appear later in the derivation than the m **a**s and **d**s, whereas in our example they appear in parallel.

3 Simulating structural descriptions

We now take a closer look at some examples of "squeezed" context-free formalisms to illustrate how a CFG can be used to simulate formalisms with greater strong generative power than CFG.

3.1 Motivation

Tree substitution grammar (TSG), tree insertion grammar (TIG), and regular-form TAG (RF-TAG) are all weakly context free formalisms which can additionally be parsed in cubic time (with a caveat for RF-TAG below). For each of these formalisms a CKY-style parser can be written whose items are of the form $[X, i, j]$ and are combined in various ways, but always according to the schema

$$\frac{[X, i, j] \quad [Y, j, k]}{[Z, i, k]}$$

just as in the CKY parser for CFG. In effect the parser dynamically converts the TSG, TIG, or RF-TAG into an equivalent CFG—each parser rule of the above form corresponds to the rule schema $Z \to XY$.

More importantly, given a grammar G and a string w, a parser can reconstruct all possible derivations of w under G by storing inside each chart item how that item was inferred. If we think of the parser as dynamically converting G into a CFG G', then this CFG is likewise able to compositionally reconstruct TSG, TIG, or RF-TAG derivations—we say that G' *simulates* G.

Note that the parser specifies how to convert G into G', but G' is not itself a parser. Thus these three formalisms have a special relationship to CFG that is independent of any particular parsing algorithm: for any TSG, TIG, or RF-TAG G, there is a CFG that simulates G. We make this notion more precise below.

3.2 Excursus: regular form TAG

Strictly speaking, the recognition algorithm Rogers gives cannot be extended to parsing; that is, it generates all possible derived trees for a given string, but not all possible derivations. It is correct, however, as a parser for a further restricted subclass of TAGs:

Definition 4 We say that a TAG is in *strict regular form* if there exists some partial ordering $\preceq$ over the nonterminal alphabet such that for every auxiliary tree β, if the root and foot of β are labeled X, then for every node η along β's spine where adjunction is allowed, $X \preceq \mathit{label}(\eta)$, and $X = \mathit{label}(\eta)$ only if η is a foot node. (In this variant adjunction at foot nodes is permitted.)

Thus the only kinds of adjunction which can occur to unbounded depth are off-spine adjunction and adjunction at foot nodes.

This stricter definition still has greater strong generative capacity than CFG. For example, the TAG in Figure 3 is in strict regular form, because the only nodes along spines where adjunction is allowed are foot nodes.

3.3 Simulability

So far we have not placed any restrictions on how these structural descriptions are computed. Even though we might imagine attaching arbitrary functions to the rules of a parser, an algorithm like CKY is only really capable of computing values of bounded size, or else structure-sharing in the chart will be lost, increasing the complexity of the algorithm possibly to exponential complexity.

For a parser to compute arbitrary-sized objects, such as the derivations themselves, it must use *back-pointers*, references to the values of subcomputations but not the values themselves. The only functions on a back-pointer the parser can compute online are the identity function (by copying the back-pointer) and constant functions (by replacing the back-pointer); any other function would have to dereference the back-pointer and destroy the structure of the algorithm. Therefore such functions must be computed offline.

Definition 5 A *simulating interpretation* $[\![\cdot]\!]$ is a bijection between two recognizable sets of terms such that

1. For each function symbol ϕ, there is a function $\bar{\phi}$ such that

$$[\![\phi(t_1, \ldots, t_n)]\!] = \bar{\phi}([\![t_1]\!], \ldots, [\![t_n]\!])$$

2. Each $\bar{\phi}$ is definable as:

$$\bar{\phi}(\langle x_{11}, \ldots, x_{1m_1})\rangle), \ldots, \langle x_{n1}, \ldots, x_{mn_m}\rangle) = \langle w_1, \ldots, w_m\rangle$$

where each w_i can take one of the following forms:

(a) a variable x_{ij}, or

(b) a function application $f(x_{i_1 j_1}, \ldots x_{i_n j_n})$, $n \geq 0$

3. Furthermore, we require that for any recognizable set T, $[\![T]\!]$ is also a recognizable set.

We say that $[\![\cdot]\!]$ is *trivial* if every $\bar{\phi}$ is definable as

$$\bar{\phi}(x_1, \ldots x_n) = f(x_{\pi(1)}, \ldots x_{\pi(n)})$$

where π is a permutation of $\{1, \ldots, n\}$.[2]

The rationale for requirement (3) is that it should not be possible, simply by imposing local constraints on the simulating grammar, to produce a simulated grammar which does not even come from a CFRS.[3]

Definition 6 We say that a grammar G from a CFRS $\mathcal{F}$ is (trivially) *simulable* by a grammar G' from another CFRS $\mathcal{F}$ if there is a (trivial) simulating interpretation $[\![\cdot]\!]_s : \mathcal{T}(G') \rightarrow \mathcal{T}(G)$ which satisfies $[\![t]\!]_{\mathcal{F}'} = [\![[\![t]\!]_s]\!]_{\mathcal{F}}$ for all $t \in \mathcal{T}(G')$.

As an example, a CFG which simulates the TAG of Figure 3 is shown in Figure 4. Note that if we give additional interpretations to the simulated yield functions α, β_1, and β_2, this CFG can compute any probabilities, translations, etc., that the original TAG can.

Note that if G' trivially simulates G, they are very nearly strongly equivalent, except that the yield functions of G' might take their arguments in a different order than G, and there might be several yield functions of G' which correspond to a single yield function of G used in several different contexts. In fact, for technical reasons we will use this notion instead of strong equivalence for testing the strong generative power of a formal system.

Thus the original problem, which was, given a formalism $\mathcal{F}$, to find a formalism that has as much strong generative power as possible but remains weakly equivalent to $\mathcal{F}$, is now recast as the following problem: find a formalism that trivially simulates as many grammars as possible but remains simulable by $\mathcal{F}$.

[2]Simulating interpretations and trivial simulating interpretations are similar to the generalized and "ungeneralized" syntax-directed translations, respectively, of Aho and Ullman (1969; 1971).

[3]Without this requirement, there are certain pathological cases that cause the construction of Section 4.2 to produce infinite MM-TAGs.

$S \rightarrow \alpha^{0\bullet}$	$\alpha(x_1, x_2) \leftarrow\!\shortmid \langle x_1, x_2\rangle$
$\alpha^{0\bullet} \rightarrow \alpha^{0}_{\bullet}$	$\langle \epsilon(), x_2\rangle \leftarrow\!\shortmid \langle -, x_2\rangle$
$\alpha^{0}_{\bullet} \rightarrow \alpha^{1\bullet}$	$\langle -, x_2\rangle \leftarrow\!\shortmid \langle -, x_2\rangle$
$\alpha^{1\bullet} \rightarrow \alpha^{1}_{\bullet}$	$\langle -, \epsilon()\rangle \leftarrow\!\shortmid \langle -, -\rangle$
$\alpha^{1}_{\bullet} \rightarrow \epsilon$	$\langle -, -\rangle \leftarrow\!\shortmid \langle -, -\rangle$
$\alpha^{0\bullet} \rightarrow \beta^0_1[\alpha^0]$	$\langle \beta_1(x_1), x_2\rangle \leftarrow\!\shortmid \langle x_1, x_2\rangle$
$\beta^0_1[\alpha^0] \rightarrow \mathbf{a}\,\beta^2_1[\alpha^0]\,\mathbf{d}$	$\langle x_1, x_2\rangle \leftarrow\!\shortmid \langle x_1, x_2\rangle$
$\beta^2_1[\alpha^0] \rightarrow \beta^0_1[\alpha^0]$	$\langle \beta_1(x_1), x_2\rangle \leftarrow\!\shortmid \langle x_1, x_2\rangle$
$\beta^2_1[\alpha^0] \rightarrow \alpha^0_{\bullet}$	$\langle \epsilon(), x_2\rangle \leftarrow\!\shortmid \langle -, x_2\rangle$
$\alpha^{1\bullet} \rightarrow \beta^0_2[\alpha^1]$	$\langle -, \beta_2(x_2)\rangle \leftarrow\!\shortmid \langle -, x_2\rangle$
$\beta^0_2[\alpha^1] \rightarrow \mathbf{b}\,\beta^2_2[\alpha^1]\,\mathbf{c}$	$\langle -, x_2\rangle \leftarrow\!\shortmid \langle -, x_2\rangle$
$\beta^2_2[\alpha^1] \rightarrow \beta^1_2[\alpha^1]$	$\langle -, \beta_2(x_2)\rangle \leftarrow\!\shortmid \langle -, x_2\rangle$
$\beta^2_2[\alpha^1] \rightarrow \alpha^1_{\bullet}$	$\langle -, \epsilon()\rangle \leftarrow\!\shortmid \langle -, -\rangle$

Figure 4: CFG which simulates the grammar of Figure 3. Here we leave the yield functions anonymous; $y \leftarrow\!\shortmid x$ denotes the function which maps x to y.

3.4 Results

The following is easy to show:

Proposition 1 Simulability is reflexive and transitive.

Because of transitivity, it is impossible that a formalism which is simulable by $\mathcal{F}$ could simulate a grammar that is not simulable by $\mathcal{F}$. So we are looking for a formalism that can trivially simulate exactly those grammars that $\mathcal{F}$ can.

In Section 4.1 we define a formalism called multicomponent multifoot TAG (MMTAG), and then in Section 4.2 we prove the following result:

Proposition 2 A grammar G from a CFRS is simulable by a CFG if and only if it is *trivially* simulable by an MMTAG in regular form.

The "if" direction ($\Leftarrow$) implies (because simulability is reflexive) that RF-MMTAG is simulable by a CFG, and therefore cubic-time parsable. (The proof below does give an effective procedure for constructing a simulating CFG for any RF-MMTAG.) The "only if" direction ($\Rightarrow$) shows that, in the sense we have defined, RF-MMTAG is the most powerful such formalism.

We can generalize this result using the notion of a meta-level grammar (Dras, 1999).

Definition 7 If $\mathcal{F}_1$ and $\mathcal{F}_2$ are two CFRSs, $\mathcal{F}_2 \circ \mathcal{F}_1$ is the CFRS characterized by the interpretation function $[\![\cdot]\!]_{\mathcal{F}_2 \circ \mathcal{F}_1} = [\![\cdot]\!]_{\mathcal{F}_2} \circ [\![\cdot]\!]_{\mathcal{F}_1}$.

$\mathcal{F}_1$ is the meta-level formalism, which generates derivations for $\mathcal{F}_2$. Obviously $\mathcal{F}_1$ must be a tree-rewriting system.

Proposition 3 For any CFRS $\mathcal{F}'$, a grammar G from a (possibly different) CFRS is simulable by a grammar in $\mathcal{F}'$ if and only if it is *trivially* simulable by a grammar in $\mathcal{F}' \circ$ RF-MMTAG.

The "only if" direction ($\Rightarrow$) follows from the fact that the MMTAG constructed in the proof of Proposition 2 generates the same derived trees as the CFG. The "if" direction ($\Leftarrow$) is a little trickier because the constructed CFG inserts and relabels nodes.

4 Multicomponent multifoot TAG

4.1 Definitions

MMTAG resembles a cross between set-local multicomponent TAG (Joshi, 1987) and ranked node rewriting grammar (Abe, 1988), a variant of TAG in which auxiliary trees may have multiple foot nodes. It also has much in common with d-tree substitution grammar (Rambow et al., 1995).

Definition 8 An *elementary tree set* $\vec{\alpha}$ is a finite set of trees (called the *components* of $\vec{\alpha}$) with the following properties:

1. Zero or more frontier nodes are designated *foot nodes*, which lack labels (following Abe), but are marked with the diacritic $*$;
2. Zero or more (non-foot) nodes are designated *adjunction nodes*, which are partitioned into one or more disjoint sets called *adjunction sites*. We notate this by assigning an index i to each adjunction site and marking each node of site i with the diacritic $\boxed{i}$.
3. Each component is associated with a symbol called its *type*. This is analogous to the left-hand side of a CFG rule (again, following Abe).
4. The components of $\vec{\alpha}$ are connected by *d-edges* from foot nodes to root nodes (notated by dotted lines) to form a single tree structure. A single foot node may have multiple d-children, and their order is significant. (See Figure 5 for an example.)

A *multicomponent multifoot tree adjoining grammar* is a tuple $\langle \Sigma, P, S \rangle$, where:

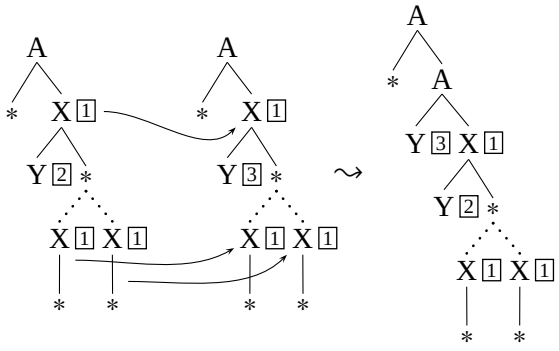


Figure 5: Example of MMTAG adjunction. The types of the components, not shown in the figure, are all X.

1. Σ is a finite alphabet;
2. P is a finite set of tree sets; and
3. $S \in \Sigma$ is a distinguished start symbol.

Definition 9 A component α is *adjoinable* at a node η if η is an adjunction node and the type of α equals the label of η.

The result of *adjoining* a component α at a node η is the tree set formed by separating η from its children, replacing η with the root of α, and replacing the ith foot node of α with the ith child of η. (Thus adjunction of a one-foot component is analogous to TAG adjunction, and adjunction of a zero-foot component is analogous to substitution.)

A tree set $\vec{\alpha}$ is *adjoinable* at an adjunction site $\vec{\eta}$ if there is a way to adjoin each component of $\vec{\alpha}$ at a different node of $\vec{\eta}$ (with no nodes left over) such that the dominance and precedence relations within $\vec{\alpha}$ are preserved. (See Figure 5 for an example.)

We now define a regular form for MMTAG that is analogous to strict regular form for TAG. A *spine* is the path from the root to a foot of a single component. Whenever adjunction takes place, several spines are inserted inside or concatenated with other spines. To ensure that unbounded insertion does not take place, we impose an ordering on spines, by means of functions ρ_i that map the type of a component to the rank of that component's ith spine.

Definition 10 We say that an adjunction node $\eta \in \vec{\eta}$ is *safe* in a spine if it is the lowest node (except the foot) in that spine, and if each component under that spine consists only of a member of $\vec{\eta}$ and zero or more foot nodes.

We say that an MMTAG G is in *regular form* if there are functions ρ_i from Σ into the domain of some partial ordering $\preceq$ such that for each component α of type X, for each adjunction node $\eta \in \alpha$, if the jth child of η dominates the ith foot node of α (that is, another component's jth spine would adjoin into the ith spine), then $\rho_i(X) \preceq \rho_j(label(\eta))$, and $\rho_i(X) = \rho_j(label(\eta))$ only if η is safe in the ith spine.

Thus the only kinds of adjunction which can occur to unbounded depth are off-spine adjunction and safe adjunction. The adjunction shown in Figure 5 is an example of safe adjunction.

4.2 Proof of Proposition 2

(⇐) First we describe how to construct a simulating CFG for any RF-MMTAG; then this direction of the proof follows from the transitivity of simulability.

When a CFG simulates a regular form TAG, each nonterminal must encapsulate a stack (of bounded depth) to keep track of adjunctions. In the multicomponent case, these stacks must be generalized to trees (again, of bounded size).

So the nonterminals of G' are of the form $[\eta, t]$, where t is a derivation fragment of G with a dot $(\cdot)$ at exactly one node $\vec{\alpha}$, and η is a node of $\vec{\alpha}$. Let $\overline{\eta}$ be the node in the derived tree where η ends up.

A fragment t can be put into a normal form as follows:

1. For every $\vec{\alpha}$ above the dot, if $\overline{\eta}$ does not lie along a spine of $\vec{\alpha}$, delete everything above $\vec{\alpha}$.
2. For every $\vec{\alpha}$ not above or at the dot, if $\overline{\eta}$ does not lie along a d-edge of $\vec{\alpha}$, delete $\vec{\alpha}$ and everything below and replace it with $\top$ if $\overline{\eta}$ dominates $\vec{\alpha}$; otherwise replace it with $\bot$.
3. If there are two nodes $\vec{\alpha}_1$ and $\vec{\alpha}_2$ along a path which name the same tree set and $\overline{\eta}$ lies along the same spine or same d-edge in both of them, collapse $\vec{\alpha}_1$ and $\vec{\alpha}_2$, deleting everything in between.

Basically this process removes all unboundedly long paths, so that the set of normal forms is finite. In the rule schemata below, the terms in the left-hand sides range over normalized terms, and their corresponding right-hand sides are renormalized. Let $up(t)$ denote the tree that results from moving the dot in t up one step.

The value of a subderivation t' of G' under $[\![\cdot]\!]_s$ is a tuple of partial derivations of G, one for each $\top$ symbol in the root label of t', in order. Where we do not define a yield function for a production below, the identity function is understood.

For every set $\vec{\alpha}$ with a single, S-type component rooted by η, add the rule

$$S \rightarrow [\eta, \vec{\alpha}(\top, \ldots, \top)]$$
$$\vec{\alpha}(x_1, \ldots, x_n) \leftrightarrow \langle x_1, \ldots, x_n \rangle$$

For every non-adjunction, non-foot node η with children $\eta_1, \ldots, \eta_n$ $(n \geq 0)$,

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

For every component with root η' that is adjoinable at η,

$$[\eta, up(t)] \rightarrow [\eta', t]$$

If η' is the root of the whole set $\vec{\alpha}'$, this rule rewrites a $\top$ to several $\top$ symbols; the corresponding yield function is then

$$\langle \ldots, \vec{\alpha}'(x_1, \ldots, x_n), \ldots \rangle \leftrightarrow \langle \ldots, x_1, \ldots, x_n, \ldots \rangle$$

For every component with ith foot η'_i that is adjoinable at a node with ith child η_i,

$$[\eta'_i, t] \rightarrow [\eta_i, up(t)]$$

This last rule skips over deleted parts of the derivation tree, but this is harmless in a regular form MMTAG, because all the skipped adjunctions are safe.

(⇒) First we describe how to decompose any given derivation t' of G' into a set of elementary tree sets.

Let $t = [\![t']\!]_s$. (Note the convention that primed variables always pertain to the simulating grammar, unprimed variables to the simulated grammar.) If, during the computation of t, a node η' creates the node η, we say that η' is *productive* and *produces* η. Without loss of generality, let us assume that there is a one-to-one correspondence between productive nodes and nodes of t.[4]

To start, let η be the root of t, and $\eta_1, \ldots, \eta_n$ its children.

Define the *domain* of η_i as follows: any node in t' that produces η_i or any of its descendants is in the domain of η_i, and any non-productive node whose parent is in the domain of η_i is also in the domain of η_i.

For each η_i, excise each connected component of the domain of η_i. This operation is the reverse of adjunction (see Figure 6): each component gets

[4] If G' does not have this property, it can be modified so that it does. This may change the derived trees slightly, which makes the proof of Proposition 3 trickier.

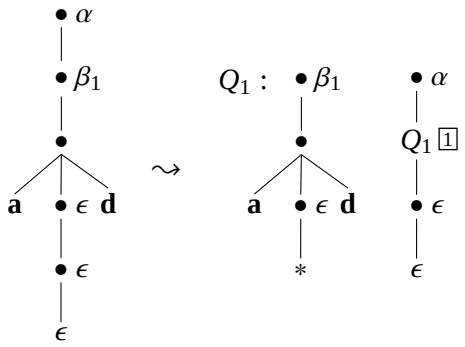


Figure 6: Example derivation (left) of the grammar of Figure 4, and first step of decomposition. Non-adjunction nodes are shown with the placeholder • (because the yield functions in the original grammar were anonymous), the Greek letters indicating what is produced by each node. Adjunction nodes are shown with labels Q_i in place of the (very long) true labels.

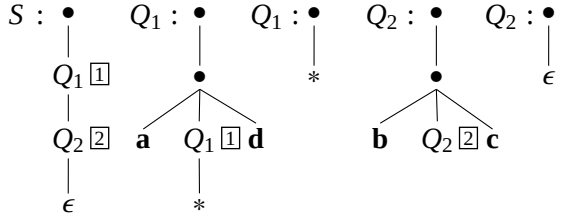


Figure 7: MMTAG converted from CFG of Figure 4 (cf. the original TAG in Figure 3). Each components' type is written to its left.

foot nodes to replace its lost children, and the components are connected by d-edges according to their original configuration.

Meanwhile an adjunction node is created in place of each component. This node is given a label (which also becomes the type of the excised component) whose job is to make sure the final grammar does not overgenerate; we describe how the label is chosen below. The adjunction nodes are partitioned such that the ith site contains all the adjunction nodes created when removing η_i.

The tree set that is left behind is the elementary tree set corresponding to η (rather, the function symbol that labels η); this process is repeated recursively on the children of η, if any.

Thus any derivation of G' can be decomposed into elementary tree sets. Let $\hat{G}$ be the union of the decompositions of all possible derivations of G' (see Figure 7 for an example).

Labeling adjunction nodes For any node η', and any list of nodes $\langle \eta'_1, \ldots, \eta'_n \rangle$, let the *signature* of η' with respect to $\langle \eta'_1, \ldots, \eta'_n \rangle$ be $\langle A, a_1, \ldots, a_m \rangle$, where A is the left-hand side of the GCFG production that generated η', and $a_i = \langle j, k \rangle$ if η' gets its ith field from the kth field of η'_j, or $*$ if η' produces a function symbol in its ith field.

So when we excise the domain of η_i, the label of the node left behind by a component α is $\langle s, s_1, \ldots, s_n \rangle$, where s is the signature of the root of α with respect to the foot nodes and $s_1, \ldots, s_n$ are the signatures of the foot nodes with respect to their d-children. Note that the number of possible adjunction labels is finite, though large.

$\hat{G}$ trivially simulates G. Since each tree of $\hat{G}$ corresponds to a function symbol (though not necessarily one-to-one), it is easy to write a trivial simulating interpretation $[\![\cdot]\!] : \mathcal{T}(\hat{G}) \to \mathcal{T}(G)$. To see that $\hat{G}$ does not overgenerate, observe that the nonterminal labels inside the signatures ensure that every derivation of $\hat{G}$ corresponds to a valid derivation of G', and therefore G. To see that $[\![\cdot]\!]$ is one-to-one, observe that the adjunction labels keep track of how G' constructed its simulated derivations, ensuring that for any derivation $\hat{t}$ of $\hat{G}$, the decomposition of the derived tree of $\hat{t}$ is $\hat{t}$ itself. Therefore two derivations of $\hat{G}$ cannot correspond to the same derivation of G', nor of G.

$\hat{G}$ is finite. Briefly, suppose that the number of components per tree set is unbounded. Then it is possible, by intersecting G' with a recognizable set, to obtain a grammar whose simulated derivation set is non-recognizable. The idea is that multicomponent tree sets give rise to dependent paths in the derivation set, so if there is no bound on the number of components in a tree set, neither is there a bound on the length of dependent paths. This contradicts the requirement that a simulating interpretation map recognizable sets to recognizable sets.

Suppose that the number of nodes per component is unbounded. If the number of components per tree set is bounded, so must the number of adjunction nodes per component; then it is possible, again by intersecting G' with a recognizable set, to obtain a grammar which is infinitely ambiguous with respect to simulated derivations, which contradicts the requirement that simulating interpretations be bijective.

$\hat{G}$ is in regular form. A component of $\hat{G}$ corresponds to a derivation fragment of G' which takes fields from several subderivations and processes them, combining some into a larger structure and copying some straight through to the root. Let $\rho_i(X)$ be the number of fields that a component of type X copies from its ith foot up to its root. This information is encoded in X, in the signature of the root. Then $\hat{G}$ satisfies the regular form constraint, because when adjunction inserts one spine into another spine, the the inserted spine must copy at least as many fields as the outer one. Furthermore, if the adjunction site is not safe, then the inserted spine must additionally copy the value produced by some lower node.

5 Discussion

We have proposed a more constrained version of Joshi's question, "How much strong generative power can be squeezed out of a formal system without increasing its weak generative power," and shown that within these constraints, a variant of TAG called MMTAG characterizes the limit of how much strong generative power can be squeezed out of CFG. Moreover, using the notion of a meta-level grammar, this result is extended to formalisms beyond CFG.

It remains to be seen whether RF-MMTAG, whether used directly or for specifying meta-level grammars, provides further practical benefits on top of existing "squeezed" grammar formalisms like tree-local MCTAG, tree insertion grammar, or regular form TAG.

This way of approaching Joshi's question is by no means the only way, but we hope that this work will contribute to a better understanding of the strong generative capacity of constrained grammar formalisms as well as reveal more powerful formalisms for linguistic analysis and natural language processing.

Acknowledgments

This research is supported in part by NSF grant SBR-89-20230-15. Thanks to Mark Dras, William Schuler, Anoop Sarkar, Aravind Joshi, and the anonymous reviewers for their valuable help. *S. D. G.*

References

Naoki Abe. 1988. Feasible learnability of formal grammars and the theory of natural language acquisition. In *Proceedings of the Twelfth International Conference on Computational Linguistics (COLING-88)*, pages 1–6, Budapest.

A. V. Aho and J. D. Ullman. 1969. Syntax directed translations and the pushdown assembler. *J. Comp. Sys. Sci*, 3:37–56.

A. V. Aho and J. D. Ullman. 1971. Translations on a context free grammar. *Information and Control*, 19:439–475.

Tilman Becker, Owen Rambow, and Michael Niv. 1992. The derivational generative power of formal systems, or, Scrambling is beyond LCFRS. Technical Report IRCS-92-38, Institute for Research in Cognitive Science, University of Pennsylvania.

Noam Chomsky. 1965. *Aspects of the Theory of Syntax*. MIT Press, Cambridge, MA.

Mark Dras. 1999. A meta-level grammar: redefining synchronous TAG for translation and paraphrase. In *Proceedings of the 37th Annual Meeting of the Assocation for Computational Linguistics*, pages 80–87, College Park, MD.

Aravind K. Joshi. 1987. An introduction to tree adjoining grammars. In Alexis Manaster-Ramer, editor, *Mathematics of Language*. John Benjamins, Amsterdam.

Aravind K. Joshi. 2000. Relationship between strong and weak generative power of formal systems. In *Proceedings of the Fifth International Workshop on TAG and Related Formalisms (TAG+5)*, pages 107–113.

Philip H. Miller. 1999. *Strong Generative Capacity: The Semantics of Linguistic Formalism*. Number 103 in CSLI lecture notes. CSLI Publications, Stanford.

Owen Rambow, K. Vijay-Shanker, and David Weir. 1995. D-tree grammars. In *Proceedings of the 33rd Annual Meeting of the Assocation for Computational Linguistics*, pages 151–158, Cambridge, MA.

James Rogers. 1994. Capturing CFLs with tree adjoining grammars. In *Proceedings of the 32nd Annual Meeting of the Assocation for Computational Linguistics*, pages 155–162, Las Cruces, NM.

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

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

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