Properties of Prefix Lexicalized Synchronous Grammars

by

Logan Orion Born

B.Sc., University of Calgary, 2016

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Summer 2018

Approval

Name:	**Logan Orion Born**
Degree:	**Master of Science (Computing Science)**
Title:	**Properties of Prefix Lexicalized Synchronous Grammars**
Examining Committee:	**Chair:** Arrvindh Shriraman Associate Professor
	Anoop Sarkar Senior Supervisor Professor
	Fred Popowich Supervisor Professor
	Nick Sumner Internal Examiner Assistant Professor School of Computing Science
Date Defended:	**June 22, 2018**

Abstract

Synchronous grammars, which may be broadly characterized as sets of rules for generating sentence pairs, are used in linguistics and natural language processing (NLP) as a framework for modelling human languages. Prefix lexicalization is a property of some grammars in which each rule contains a terminal symbol (also called a lexical item) at its left edge. This work shows that the class of synchronous context-free grammars (SCFG) is not closed under prefix lexicalization, and presents an algorithm for converting a finitely ambiguous, ε-free SCFG to a weakly equivalent prefix lexicalized synchronous tree-adjoining grammar (STAG). This transformation only increases the rank of the grammar by 1; it at most cubes the grammar size, but we provide empirical results showing that the size increase is generally much smaller. We show that this result can also be extended to weighted synchronous grammars without affecting the weights they assign to string pairs. Theoretically, these results serve to generalize extended Greibach normal form from CFGs to SCFGs, and practically they have potential applications in machine translation. They also serve as an avenue for investigating the properties of related grammar formalisms in future work.

Keywords: Formal language theory; Synchronous grammars; Context-free grammars; Tree-adjoining grammars; Weighted grammars; Greibach normal form

Acknowledgements

Þæs ofereode, thesis *swa mæg.*
—Deor

I would like to express my sincere thanks to Anoop Sarkar for his continued support and advice over the course of this degree. Thanks especially for forcing me to take breaks from work by lending me your board games!

Thanks also to Fred Popowich and Nick Sumner for their many insightful and constructive comments as members of my examining committee.

I would also like to thank Dr. Peter Høyer, without whose influence I would probably not have pursued graduate studies. Your rigor, enthusiasm, and good spirit showed me what it means to be a great scientist. Thank you for working with me during my undergraduate studies, even though my work was so far removed from your own research!

Finally, I wish to thank my most amazing Sara, who has encouraged and taken care of me through this degree. Thank you for helping me to have a life outside of work, and for supporting me while I was too absorbed to do anything but work. You give me a confidence which I struggle to muster on my own, and without which I would not have made it to this point. *Ic lufie þē!*

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis presents a variety of formal and empirical results about prefix lexicalized synchronous grammars. We show that the class of synchronous context-free grammars is not closed under prefix lexicalization, unlike the class of non-synchronous context-free grammars which admits a prefix lexicalized normal form. We then present a method for prefix lexicalizing a synchronous context-free grammar by converting it to an equivalent grammar in a more powerful formalism. The remainder of the thesis considers the properties of grammars produced by this transformation, including their size, rank, generative capacity, and parse complexity. We show that this transformation can be applied to weighted grammars without affecting the weights of the string pairs which they generate, and we present techniques for normalizing weighted prefix lexicalized grammars into probabilistic grammars. We additionally discuss existing applications for these grammars and suggest further use cases to be investigated in future work.

1.1 Motivation

Greibach normal form (GNF; Greibach, 1965) is an important construction in formal language theory which allows every context-free grammar (CFG) to be rewritten so that the first character of each rule is a terminal symbol. A grammar in GNF is said to be *prefix lexicalized*, because the prefix of every production is a lexical item.

GNF has a variety of theoretical and practical applications, including for example the proofs of Shamir's Theorem and the Chomsky-Schützenberger Theorem (Shamir, 1967; Chomsky and Schützenberger, 1963; Autebert et al., 1997). Other applications of prefix lexicalization include proving coverage of parsing algorithms (Gray and Harrison, 1972) and decidability of equivalence problems (Christensen et al., 1995). Prefix lexicalization further guarantees finite ambiguity and prevents left-recursion, properties which can be useful for parsing depending on the parsing strategy used.

Similar prefix-lexicalized normal forms exist for a variety of grammar formalisms, including for example definite clause grammars (Dymetman, 1992), context-free valence grammars

(Fernau and Stiebe, 2002), and multiple context-free tree grammars (MCFTGs; Engelfriet et al., 2017).

By using prefix lexicalized synchronous context-free grammars (SCFGs), Watanabe et al. (2006) and Siahbani et al. (2013) obtain asymptotic and empirical speed improvements on hierarchical machine translation tasks. Using a prefix lexicalized grammar ensures that, given an input sentence, candidate translations can be generated from left to right, which allows the use of beam search to constrain their decoder's search space as it performs a left-to-right traversal of translation hypotheses. However, to achieve these results, new grammars had to be heuristically constructed to include only prefix lexicalized productions, as there is at present no way to automatically convert an existing SCFG to a prefix lexicalized form.

The above results serve to motivate the work in this thesis. From the perspective of formal language theory, we seek to generalize Greibach normal form to synchronous grammars, or to prove that no such normal form can exist for this class. From the perspective of natural language processing, we wish to provide a principled means of constructing prefix lexicalized grammars so that the techniques developed by Watanabe et al. (2006) and Siahbani et al. (2013) can be used without recourse to heuristically constructed grammars. In pursuit of these goals we also explore further theoretical and empirical properties of synchronous prefix lexicalized grammars, thereby enriching our understanding of a class of grammars with various applications in NLP.

1.2 Contributions

In brief, this work proves new properties about the class of synchronous context-free grammars (SCFG) and defines and characterizes a new subset of the class of synchronous tree-adjoining grammars (STAG). Concretely, our contributions include the following:

- We prove that SCFG is not closed under prefix lexicalization.

  More concretely, we show that an SCFG cannot be converted into an equivalent prefix lexicalized SCFG if it contains certain discontinuities or if its target projection is infinitely ambiguous.

- We give a method for converting an SCFG into an equivalent STAG which is prefix lexicalized.

  This can be considered a generalization of (extended) Greibach normal form to synchronous context-free grammars, which is a novel and theoretically interesting contribution in its own right, and which removes the need for heuristically constructed grammars in work such as Watanabe et al. (2006).

- We show that prefix lexicalized STAGs have formal and empirical properties which make them feasible to use in NLP applications.

In particular, we show that converting an SCFG into an equivalent prefix lexicalized STAG only increases its rank by 1. Furthermore we show that although STAG is generally asymptotically slower to parse than SCFG, the prefix lexicalized STAGs produced by our transformation have the same parse complexity as SCFGs. Lastly, although our lexicalization may at most cube the size of the input grammar, we show that in practice the size increase is only quadratic.

- We generalize existing techniques for normalizing weighted CFGs so that they can be applied to weighted STAGs.

  This allows us to further generalize our lexicalization to apply to weighted and probabilistic grammars.

1.3 Overview

Chapter 2 of this thesis reviews synchronous context-free and tree-adjoining grammars, as well as their weighted and probabilistic variants. This chapter also formally defines important concepts such as prefix lexicalization.

Chapter 3 presents the central result that an ϵ-free SCFG which has a finitely ambiguous target projection can be converted to an equivalent prefix lexicalized STAG, as well as the result that SCFG is not closed under prefix lexicalization. We prove various formal results about our lexicalization, including bounds on grammar rank and size. We show empirically that the size increase on grammars from real NLP tasks is significantly better than worst case, and we discuss how these results relate to the work of Watanabe et al. (2006).

Chapter 4 considers weighted and probabilistic grammars, and shows that our transformation may be applied to these grammars without changing the weights which they assign to string pairs. We discuss techniques for normalizing a weighted, prefix lexicalized STAG into a probabilistic prefix lexicalized STAG.

Chapter 5 reviews these results and concludes with a discussion of future work.

Chapter 2

Background

This chapter introduces the formalisms and terminology which will be used throughout the rest of this thesis.

2.1 Synchronous Grammar Formalisms

2.1.1 Synchronous Context-Free Grammar

A synchronous context-free grammar (SCFG) is a tuple $G = (N, \Sigma, P, S)$ where N is a finite non-terminal alphabet, Σ is a finite terminal alphabet, $S \in N$ is a distinguished nonterminal called the start symbol, and P is a finite set of synchronous productions of the form

$$\langle A_1 \to \alpha_1, A_2 \to \alpha_2 \rangle \tag{2.1}$$

for some $A_1, A_2 \in N$ and strings $\alpha_1, \alpha_2 \in (N \cup \Sigma)^*$.[1] Every nonterminal which appears in α_1 must be linked to exactly one nonterminal in α_2, and *vice versa*. We write these links using numerical annotations, as in (2.2).

$$\langle A \to A\boxed{1}B\boxed{2}, B \to B\boxed{2}A\boxed{1} \rangle \tag{2.2}$$

Links will not be explicitly written if they are irrelevant to the discussion or are obvious from context. This will most often be the case when one side of a rule has been abbreviated as a string $\alpha \in (N \cup \Sigma)^*$, as in (2.3):

$$\langle A_1 \to A_2, A_3 \to \alpha \rangle \tag{2.3}$$

[1] A variant formalism exists which requires that $A_1 = A_2$; this is called syntax-directed transduction grammar (Lewis and Stearns, 1968) or syntax-directed translation schemata (Aho and Ullman, 1969). This variant is weakly equivalent to SCFG, but SCFG has greater strong generative capacity (Crescenzi et al., 2015). Another related formalism is scattered context grammar (Greibach and Hopcroft, 1969), which uses rules similar to SCFG productions to generate strings rather than string pairs.

where $A_1, A_2, A_3 \in N$ and $\alpha \in (N \cup \Sigma)^*$. Although there is no explicit link written on A_2, we assume that it is linked to some nonterminal in α so that the rule is licit. (Likewise, we assume that all non-terminals appearing in an abbreviated string like α are properly linked, despite these links not being shown.)

An SCFG has rank k if no rule in the grammar contains more than k pairs of linked nonterminals.

In every step of an SCFG derivation, we rewrite one pair of linked nonterminals using a rule from P, in essentially the same way we would rewrite a single nonterminal in a non-synchronous CFG. For example, (2.4) shows linked A and B nodes being rewritten using the production in (2.2):

$$\langle X\boxed{1}A\boxed{2}, B\boxed{2}Y\boxed{1}\rangle \Rightarrow \langle X\boxed{1}A\boxed{2}B\boxed{3}, B\boxed{3}A\boxed{2}Y\boxed{1}\rangle \tag{2.4}$$

Note how the $\boxed{1}$ and $\boxed{2}$ from rule (2.2) are renumbered to $\boxed{2}$ and $\boxed{3}$ during rewriting, to avoid an ambiguity with the $\boxed{1}$ already present in the derivation. In (2.4), the symbol $\Rightarrow$ represents the effect of a single rewriting operation; when a derivation involves the application of multiple rewriting operations we may abbreviate the entire derivation using $\Rightarrow^*$.

An SCFG derivation is complete when it contains no more nonterminals to rewrite. A completed derivation represents a string pair generated by the grammar.

For reference, Figure 2.1 gives an example of an entire SCFG which contains 5 synchronous rules. This SCFG generates the language $\{\langle a^{2i+1}b^{2j+1}, b^{2j+1}a^{2i+1}\rangle | i, j \geq 0\}$.

$$\begin{array}{ll}
\langle S \rightarrow A\boxed{1}B\boxed{2}, & S \rightarrow B\boxed{2}A\boxed{1}\rangle \\
\langle A \rightarrow aA\boxed{1}a, & A \rightarrow aA\boxed{1}a\rangle \\
\langle A \rightarrow a, & A \rightarrow a\rangle \\
\langle B \rightarrow bB\boxed{1}b, & B \rightarrow bB\boxed{1}b\rangle \\
\langle B \rightarrow b, & B \rightarrow b\rangle
\end{array}$$

Figure 2.1: An SCFG which generates $\{\langle a^{2i+1}b^{2j+1}, b^{2j+1}a^{2i+1}\rangle | i, j \geq 0\}$.

2.1.2 Synchronous Tree-Adjoining Grammar

A synchronous tree-adjoining grammar (STAG; Shieber 1994) is a tuple $G = (N, \Sigma, T, S)$ where N is a finite nonterminal alphabet, Σ is a finite terminal alphabet, $S \in N$ is a distinguished nonterminal called the start symbol, and T is a finite set of synchronous tree pairs of the form

$$\langle t_1, t_2 \rangle \tag{2.5}$$

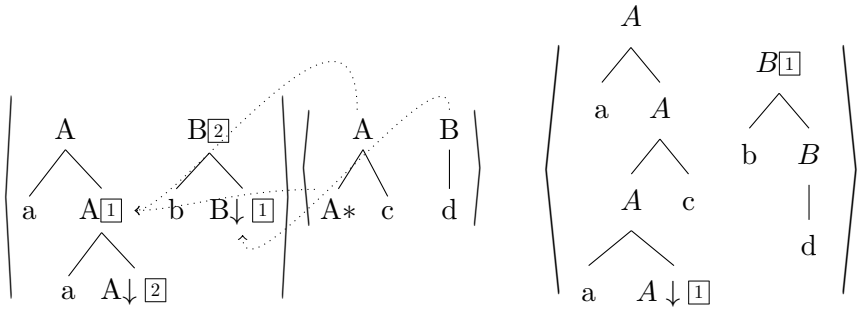


Figure 2.2: An example of synchronous rewriting in an STAG (left) and the resulting tree pair (right). The auxiliary tree rooted in A adjoins to the $A\boxed{1}$ node; note how the subtree rooted in $A\boxed{1}$ ends up attached below the foot node of the auxiliary tree. The initial tree rooted in B simply substitutes into the linked $B\boxed{1}$ node, overwriting it.

where t_1 and t_2 are elementary trees as defined in Joshi et al. (1975). An elementary tree over the alphabets N and Σ is a tree whose internal nodes are labeled by symbols in N and whose leaves are labeled by symbols in $(N \cup \Sigma)$. A *substitution site* is a nonterminal leaf node which is marked by $\downarrow$; a *foot node* is a nonterminal leaf node which is marked by $*$. An elementary tree can contain at most one foot node, and that node must be labeled by the same nonterminal symbol as the root node. If a tree contains a foot node, it is called an *auxiliary tree*; otherwise it is called an *initial tree*.

Every substitution site in t_1 must be linked to exactly one nonterminal node in t_2, and *vice versa*. As in an SCFG, we write these links using numbered annotations, and omit these annotations when they are not relevant to the discussion at hand. Rank is defined for an STAG in the same way it is defined for an SCFG.

In every step of an STAG derivation, we rewrite one pair of linked nonterminals with a tree pair from T, using the same substitution and adjoining operations defined for non-synchronous TAG. Substitution overwrites a substitution site with an initial tree rooted in the same nonterminal symbol as that substitution site; adjoining "splits" a tree apart at a given node and inserts an auxiliary tree rooted in the same nonterminal as that node into the gap that is created. Examples of both operations are shown in Figure 2.2.

We use a variant of STAG which disallows multiple adjunction, which means that any given node may be linked to at most one other node (Shieber, 1994). This restricts the generative capacity of the formalism and ensures that it only generates tree-adjoining languages (TALs).

For the sake of comparison, Figure 2.3 gives an example of an STAG which generates the language $\{\langle a^{2i+1}b^{2j+1}, b^{2j+1}a^{2i+1}\rangle | i, j \geq 0\}$. This is the same language generated by the SCFG in Figure 2.1. Note how the a characters are nested using wrapping adjunction, while the bs are nested using substitution.

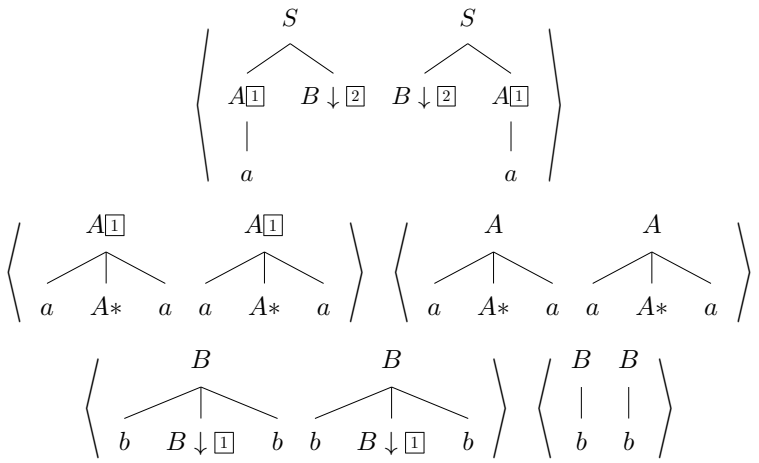


Figure 2.3: An STAG which generates $\{\langle a^{2i+1}b^{2j+1}, b^{2j+1}a^{2i+1}\rangle | i, j \geq 0\}$.

2.2 Other Terminology

We use *synchronous production* as a cover term for either a synchronous rule in an SCFG or a synchronous tree pair in an STAG.

Following Siahbani et al. (2013), we refer to the left half of a synchronous production as the source side, and the right half as the target side; this captures the intuition that synchronous grammars model translational equivalence between a source phrase and its translation into a target language. Other authors calls these the left and right components (Crescenzi et al., 2015) or, viewing the grammar as a transducer, the input and the output (Engelfriet et al., 2017). The *source projection* of a synchronous grammar G is the non-synchronous grammar obtained by deleting the target side of every production in G; the *target projection* is likewise obtained by deleting the source side of every production.

The language generated by a synchronous grammar G is defined as the set of all string pairs which can be derived from the start symbol using the productions in G; we write this as $L(G)$. This parallels the definition for a non-synchronous grammar, except that if G is non-synchronous then $L(G)$ is a set of strings rather than string pairs.

A pumpable cycle is a sequence of rules where the first rule in the sequence can rewrite some element produced by the last rule in the sequence. Such a cycle can be "pumped" (that is, repeated) to create derivations of arbitrary length, as in the pumping lemma for context-free languages (Bar-Hillel et al., 1961).

We call a grammar ε-free if it contains no productions whose source or target side produces only the empty string ε. A grammar G is finitely ambiguous if for every string pair in $L(G)$ there are only a finite number of ways to derive that string pair using the productions in G. Generally, a grammar is finitely ambiguous if it is ε-free and contains no pumpable cycles of chain rules (rules which rewrite a single nonterminal without introducing any other symbols).

2.3 Synchronous Prefix Lexicalization

Previous work (Watanabe et al., 2006; Siahbani et al., 2013) has shown that it is useful for the target side of a synchronous grammar to start with a terminal symbol. For this reason, we define a synchronous grammar to be prefix lexicalized just in case the leftmost character of the target side[2] of every synchronous production in that grammar is a terminal symbol.

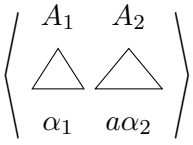
Formally, this means that every synchronous rule in a prefix lexicalized SCFG (or PL-SCFG) is of the form

$$\langle A_1 \to \alpha_1, A_2 \to a\alpha_2 \rangle \tag{2.6}$$

for some $A_1, A_2 \in N$, $\alpha_1, \alpha_2 \in (\Sigma \cup N)^*$, and $a \in \Sigma$.

Every synchronous tree pair in a prefix lexicalized STAG (or PL-STAG) is of the form

(2.7)



for some $A_1, A_2 \in N$, $\alpha_1, \alpha_2 \in (\Sigma \cup N)^*$, and $a \in \Sigma$. The triangle notation in (2.7) is used throughout this thesis to abbreviate the internal structure of a tree when this structure is not relevant to the discussion at hand.

2.4 Weighted Grammar Formalisms

Every SCFG or STAG G over an alphabet Σ partitions the set of string pairs $\Sigma^* \times \Sigma^*$ into two groups: the set of string pairs which are in $L(G)$, and the set of string pairs which are not in $L(G)$. Such a coarse partition does not accurately represent the intricacies of human language, however; within a single language some strings may be comparatively more acceptable than others, and when translating between languages multiple translations are usually possible, some being more likely than others.

For this reason a weighted or probabilistic grammar may be used in place of a simple SCFG or STAG: such a grammar has a weight associated with each production, representing the likelihood of that production being used in a derivation. Multiplying the weight of each production used in a derivation gives an overall weight for that derivation, and summing over the weights of all derivations for a given string pair gives a value representing the likelihood of that string pair.

[2]All of the proofs in this document admit a symmetrical variant which lexicalizes the source side instead of the target. We are not aware of any applications where source-side lexicalization is useful, so we do not address this case. For the same reason, we do not address the stricter case where both source and target trees are prefix lexicalized.

2.4.1 Weighted SCFG

Formally, every synchronous rule r in a weighted SCFG (WSCFG) is annotated with a weight w_r. If D is a derivation in a WSCFG, then the weight of D is written $W(D)$ and is the product of the weight of every rule used in the course of D:

$$W(D) = \prod_{r \in D} w_r \tag{2.8}$$

Let $D(u, v)$ be the set of all derivations in the grammar G which produce the string pair $\langle u, v \rangle$. Then the weight of $\langle u, v \rangle$ is the sum of the weights of every derivation in $D(u, v)$:

$$W(u, v) = \sum_{D \in D(u,v)} W(D) = \sum_{D \in D(u,v)} \prod_{r \in D} w_r \tag{2.9}$$

A probabilistic SCFG (PSCFG) is a WSCFG where, for every pair of nonterminals A_1 and A_2, the grammar weights define a probability distribution over the rules which rewrite $\langle A_1, A_2 \rangle$. If $\pi(A_1, A_2)$ is the set of all rules of the form $\langle A_1 \to \alpha_1, A_2 \to \alpha_2 \rangle$ for any $\alpha_1, \alpha_2 \in (N \cup \Sigma)^*$, then a PSCFG satisfies

$$\sum_{r \in \pi(A_1, A_2)} w_r = 1 \tag{2.10}$$

for every choice of $A_1, A_2 \in N$.

In a PSCFG, the probability of a derivation D is written $P(D)$ and is defined analagously to the weight of a derivation in a WSCFG; likewise the probability $P(u, v)$ of a string pair $\langle u, v \rangle$ is defined like the weight of a string pair in a WSCFG.

Figure 2.4 shows an example of a WSCFG which generates the language $\{\langle a^{2i+1}b^{2j+1}, b^{2j+1}a^{2i+1} \rangle \mid i, j \geq 0\}$ and assigns the weight $2^i \cdot 3^j$ to every string pair $\langle a^{2i+1}b^{2j+1}, b^{2j+1}a^{2i+1} \rangle$.

$\langle S \to A\boxed{1}B\boxed{2},$	$S \to B\boxed{2}A\boxed{1} \rangle$	$w = 1$
$\langle A \to aA\boxed{1}a,$	$A \to aA\boxed{1}a \rangle$	$w = 2$
$\langle A \to a,$	$A \to a \rangle$	$w = 1$
$\langle B \to bB\boxed{1}b,$	$B \to bB\boxed{1}b \rangle$	$w = 3$
$\langle B \to b,$	$B \to b \rangle$	$w = 1$

Figure 2.4: A WSCFG which generates $\{\langle a^{2i+1}b^{2j+1}, b^{2j+1}a^{2i+1} \rangle | i, j \geq 0\}$. The weight of a rule is written beside that rule; the grammar assigns weight $2^i \cdot 3^j$ to the string pair $\langle a^{2i+1}b^{2j+1}, b^{2j+1}a^{2i+1} \rangle$.

2.4.2 Weighted STAG

A weighted STAG (WSTAG) is similarly defined as an STAG where each synchronous tree pair t is assigned a weight w_t; the weight of a derivation (or, equivalently, of a derived tree pair) is the product of the weights of every elementary tree pair involved in the derivation, and the weight of a string pair is the sum of the weights of every derivation which produces that string pair.

Analogously, a probabilistic STAG (PSTAG) is a WSTAG where, for every pair of nonterminals A_1 and A_2, the grammar weights define a probability distribution over the tree pairs rooted in A_1 and A_2.

We give an example of a weighted STAG in Figure 2.5. This grammar generates the same language (with the same weights) as the SCFG in Figure 2.4.

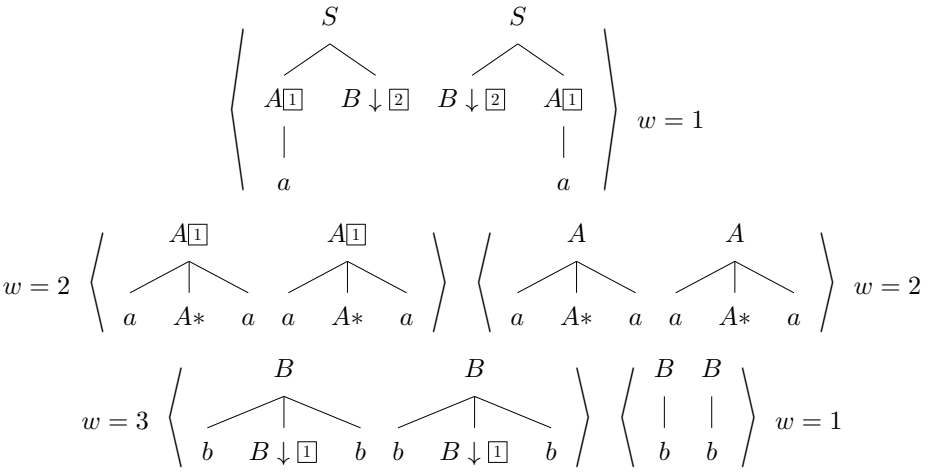


Figure 2.5: A WSTAG which generates $\{\langle a^{2i+1}b^{2j+1}, b^{2j+1}a^{2i+1}\rangle | i, j \geq 0\}$. The weight of a tree pair is written beside that tree pair; the grammar assigns weight $2^i \cdot 3^j$ to the string pair $\langle a^{2i+1}b^{2j+1}, b^{2j+1}a^{2i+1}\rangle$.

2.4.3 Weights from Arbitrary Algebras

In many NLP applications, grammar weights will belong to $\mathbb{R}$, but in general this need not be the case. In algebraic formal language theory (e.g. Stanat 1972) a more general case has been considered where grammar weights may belong to arbitrary algebraic structures. For example, given a commutative semiring $(\mathbb{S}, \oplus, \otimes)$, we may define a weighted grammar with weights belonging to $\mathbb{S}$. The weight of a derivation or string in such a grammar is exactly as defined above, except that we compute the sums and products using the semiring operations $\oplus$ and $\otimes$. We require the semiring to be commutative because, in general, the rewriting operations involved in a derivation are not ordered relative to one another and there is no principled way to impose an ordering over them. We do not consider weights

from more general structures than semirings, as we are not aware of any applications where such weights are used.

2.5 Conclusion

This chapter has introduced the concepts needed to prove the results in the following chapters. We will now consider the central question of this thesis, which is whether or not the class SCFG admits a prefix lexicalized normal form, and how to convert grammars into this form if it exists.

Chapter 3

Prefix Lexicalizing SCFG

This chapter considers the problem of prefix lexicalizing the class SCFG. Section 3.1 proves that SCFG is not closed under prefix lexicalization; we then provide an algorithm which prefix lexicalizes an SCFG by converting it to an equivalent STAG in Section 3.2. Section 3.3 considers the formal properties of the grammars produced by our lexicalization, and Section 3.4 details an empirical investigation into the size increase which our lexicalization incurs. Section 3.5 discusses how to use these grammars for hierarchical translation in place of SCFGs, and Section 3.6 closes with an examination of other related work.

3.1 Closure under Prefix Lexicalization

Theorem 1. *There exists an SCFG which cannot be converted to an equivalent PL-SCFG; thus SCFG is not closed under prefix lexicalization.*

Proof. The SCFG in (3.1) generates the language $L = \{\langle a^i b^j c^i, b^j a^i \rangle |\ i \geq 0, j \geq 1\}$, but we will show that this language cannot be generated by any PL-SCFG:

$$
\begin{array}{ll}
\langle S \to A\boxed{1}, & S \to A\boxed{1} \rangle \\
\langle A \to aA\boxed{1}c, & A \to A\boxed{1}a \rangle \\
\langle A \to bB\boxed{1}, & A \to bB\boxed{1} \rangle \\
\langle A \to b, & A \to b \rangle \\
\langle B \to bB\boxed{1}, & B \to bB\boxed{1} \rangle \\
\langle B \to b, & B \to b \rangle
\end{array}
\tag{3.1}
$$

Suppose, for the purpose of contradiction, that some PL-SCFG does generate L; call this grammar G. Then the following derivations must all be possible in G for some nonterminals U, V, X, Y:

(i) $\langle U\boxed{1}, V\boxed{1}\rangle \Rightarrow^* \langle b^k U\boxed{1}b^m, b^n V\boxed{1}b^p\rangle$, where $k+m=n+p$ and $n \geq 1$

(ii) $\langle X\boxed{1}, Y\boxed{1}\rangle \Rightarrow^* \langle a^q X\boxed{1}c^q, a^r Y\boxed{1}a^s\rangle$, where $q = r+s$ and $r \geq 1$

(iii) $\langle S\boxed{1}, S\boxed{1}\rangle \Rightarrow^* \langle \alpha_1 X\boxed{1}\alpha_2, b\alpha_3 Y\boxed{1}\alpha_4\rangle$, where $\alpha_1, ..., \alpha_4 \in (N \cup \Sigma)^*$

(iv) $\langle X\boxed{1}, Y\boxed{1}\rangle \Rightarrow^* \langle \alpha_5 U\boxed{1}\alpha_6, \alpha_7 V\boxed{1}\alpha_8\rangle$, where $\alpha_5, \alpha_6, \alpha_8 \in (N \cup \Sigma)^*$ and $\alpha_7 \in \Sigma \cdot (N \cup \Sigma)^*$

(i) and (ii) follow from the same arguments used in the pumping lemma for (non-synchronous) context free languages (Bar-Hillel et al., 1961): strings in L can contain arbitrarily many as, bs, and cs, so there must exist some pumpable cycles which generate these characters. In (i), $k+m=n+p$ because the final derived strings must contain an equal number of bs, and $n \geq 1$ because G is prefix lexicalized; in (ii) the constraints on q, r and s follow likewise from L. (iii) follows from the fact that, in order to pump on the cycle in (ii), this cycle must be reachable from the start symbol. (iv) follows from the fact that a context-free production cannot generate a discontinuous span. Once the cycle in (i) has generated a b in the source string, it is impossible for (ii) to generate an a on one side of the b and a c on the other. Therefore (i) must always be derived strictly later than (ii), as shown in (iv).

Now we obtain a contradiction. Given that G can derive all of (i) through (iv), the following derivation is also possible:

$$
\begin{aligned}
& \langle S\boxed{1}, S\boxed{1}\rangle \\
\Rightarrow^* \quad & \langle \alpha_1 X\boxed{1}\alpha_2, b\alpha_3 Y\boxed{1}\alpha_4\rangle \\
\Rightarrow^* \quad & \langle \alpha_1 a^q X\boxed{1}c^q\alpha_2, b\alpha_3 a^r Y\boxed{1}a^s\alpha_4\rangle \\
\Rightarrow^* \quad & \langle \alpha_1 a^q \alpha_5 U\boxed{1}\alpha_6 c^q \alpha_2, b\alpha_3 a^r \alpha_7 V\boxed{1}\alpha_8 a^s \alpha_4\rangle \\
\Rightarrow^* \quad & \langle \alpha_1 a^q \alpha_5 b^k U\boxed{1}b^m \alpha_6 c^q \alpha_2, b\alpha_3 a^r \alpha_7 b^n V\boxed{1}b^p \alpha_8 a^s \alpha_4\rangle
\end{aligned}
\tag{3.2}
$$

But since $n, r \geq 1$ the derived target string contains an a before a b and thus does not belong to L.

This is a contradiction: if G is a PL-SCFG then it must generate (i) through (iv), but if so then it also generates strings which do not belong to L. Thus no PL-SCFG can generate L, and SCFG must not be closed under prefix lexicalization. ■

The language in 3.1 is resistant to prefix lexicalization because context-free productions cannot generate discontinuous spans: this fact enforces an ordering on how various parts of a string pair can be generated relative to other parts of that string pair, as shown in (iv). This in turn permits spurious derivations of the shape in (3.2) which generate strings outside the desired language. It is also interesting to note that the grammar in (3.1) has rank 1: it is therefore among the simplest of grammars in the class SCFG. This reinforces the intuition that discontinuities, and not high rank or complex reorderings, are the quality which makes an SCFG resistant to prefix lexicalization.

There also exist grammars which cannot be prefix lexicalized because their target projections are not finitely ambiguous. If an SCFG can derive a cycle such as $\langle X\boxed{1}, Y\boxed{1}\rangle \Rightarrow^* \langle xX\boxed{1}, Y\boxed{1}\rangle$ for some $x \in \Sigma$ and $X, Y \in N \setminus \{S\}$, then this cycle can be pumped to generate arbitrarily many symbols in the source string without adding anything to the target string. Prefix lexicalizing the grammar would force it to generate some terminal symbol in the target string at each step of the derivation, making it impossible to generate the original language in which the length of a source string may be unboundedly greater than the length of its corresponding target string. The target projection of such a grammar will not be finitely ambiguous, because it will contain a cycle of chain rules which can be pumped to create infinitely many derivations for a single string. Since lexicalized grammars must be finitely ambiguous, lexicalizing such a grammar will by necessity change the language it generates.

For this reason, the rest of this thesis will focus on lexicalizing grammars with source discontinuities rather than those with infinitely ambiguous target projections. Unless otherwise specified, all grammars discussed from this point on are assumed to have a finitely ambiguous target projection; we make the additional requirement that grammars are ε-free, to avoid the case where trees are "lexicalized" by the empty string.[1] We do not consider this restriction to be a major drawback, because when dealing with natural language it is reasonable to assume finite ambiguity: to take an example from translation, although languages may vary in the number of words used to express a given concept, the length of a sentence in one language will at least be proportional to the length of its translation, and will not be unboundedly longer or shorter.

3.2 Prefix Lexicalization using STAG

We now show that an ε-free SCFG with a finitely ambiguous target projection can be prefix lexicalized by converting it to an equivalent STAG.

Theorem 2. *Given a rank-k SCFG G which is ε-free and has a finitely ambiguous target projection, an STAG H exists such that H is prefix lexicalized and $L(G) = L(H)$. The rank of H is $k+1$, and $|H| = O(|G|^3)$.*

Proof. Let $G = (N, \Sigma, P, S)$ be an ε-free SCFG with a finitely ambiguous target projection. We provide a constructive method for prefix lexicalizing the target side of G.

We begin by constructing an intermediate grammar $G_{A_1 A_2}$ for each pair of nonterminals $A_1, A_2 \in N \setminus \{S\}$. For each $A_1, A_2 \in N \setminus \{S\}$, $G_{A_1 A_2}$ will be constructed to generate the

[1] In some applications (such as word alignment) it is acceptable to have a tree lexicalized by the empty string, for example to represent the deletion of a word between two languages. For the sake of formal correctness, however, we prefer to focus on the case where every tree is lexicalized by an overt string from Σ, so that our transformation is a lexicalization in the strict formal sense.

$$\langle A_1 \boxed{1}, A_2 \boxed{1} \rangle \Rightarrow \langle \alpha_1 B_{11} \boxed{1} \beta_1, B_{21} \boxed{1} \gamma_1 \rangle \Rightarrow \langle \alpha_1 \alpha_2 B_{12} \boxed{1} \beta_2 \beta_1, B_{22} \boxed{1} \gamma_2 \gamma_1 \rangle$$

$$\Rightarrow^* \langle \alpha_1 \cdots \alpha_t B_{1t} \boxed{1} \beta_t \cdots \beta_1, B_{2t} \boxed{1} \gamma_t \cdots \gamma_1 \rangle \Rightarrow \langle \alpha_1 \cdots \alpha_t \alpha_{t+1} \beta_t \cdots \beta_1, a \gamma_{t+1} \gamma_t \cdots \gamma_1 \rangle$$

Figure 3.1: A target-side terminal leftmost derivation. $a \in \Sigma$, $A_1, A_2, B_{1i}, B_{2i} \in N$ for $1 \leq i \leq t$, and $\alpha_i, \beta_i, \gamma_i \in (N \cup \Sigma)^*$ for $1 \leq i \leq t+1$.

language of sentential forms derivable from $\langle A_1, A_2 \rangle$ via a target-side terminal leftmost derivation (TTLD). A TTLD is a derivation of the form in Figure 3.1, where the leftmost nonterminal in the target string is expanded until it produces a terminal symbol as the first character. We write $\langle A_1, A_2 \rangle \Rightarrow^*_{TTLD} \langle u, v \rangle$ to mean that $\langle A_1, A_2 \rangle$ derives $\langle u, v \rangle$ by way of a TTLD; in this notation, $L_{A_1 A_2} = \{\langle u, v \rangle | \langle A_1, A_2 \rangle \Rightarrow^*_{TTLD} \langle u, v \rangle\}$.

Given a pair $A_1, A_2 \in N \setminus \{S\}$ we formally define $G_{A_1 A_2}$ as an STAG over the terminal alphabet $\Sigma_{A_1 A_2} = (N \cup \Sigma)$ and nonterminal alphabet $N_{A_1 A_2} = \{Y_{A_1 A_2} | Y \in N\}$, with start symbol $S_{A_1 A_2}$. $N_{A_1 A_2}$ contains nonterminals indexed by $A_1 A_2$ to ensure that two intermediate grammars $G_{A_1 A_2}$ and $G_{B_1 B_2}$ do not interact as long as $\langle A_1, A_2 \rangle \neq \langle B_1, B_2 \rangle$. $G_{A_1 A_2}$ contains four kinds of tree pairs: [2]

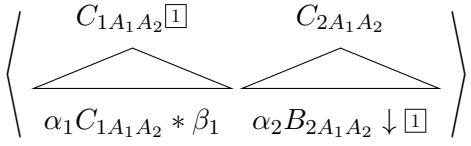
- For each rule in G of the form $\langle A_1 \to \alpha_1, A_2 \to a \alpha_2 \rangle$, $a \in \Sigma$, $\alpha_1, \alpha_2 \in (N \cup \Sigma)^*$, we add a tree pair of the form in Figure 3.2(a).

- For each rule in G of the form $\langle B_1 \to \alpha_1, B_2 \to a \alpha_2 \rangle$, $a \in \Sigma$, $\alpha_1, \alpha_2 \in (N \cup \Sigma)^*$, $B_1, B_2 \in N \setminus \{S\}$, we add a tree pair of the form in Figure 3.2(b).

- For each rule in G of the form $\langle B_1 \to \alpha_1 C_1 \boxed{1} \beta_1, B_2 \to C_2 \boxed{1} \alpha_2 \rangle$, $\alpha_1, \alpha_2, \beta_1 \in (N \cup \Sigma)^*$, $B_1, B_2, C_1, C_2 \in N \setminus \{S\}$, we add a tree pair of the form in Figure 3.2(c).

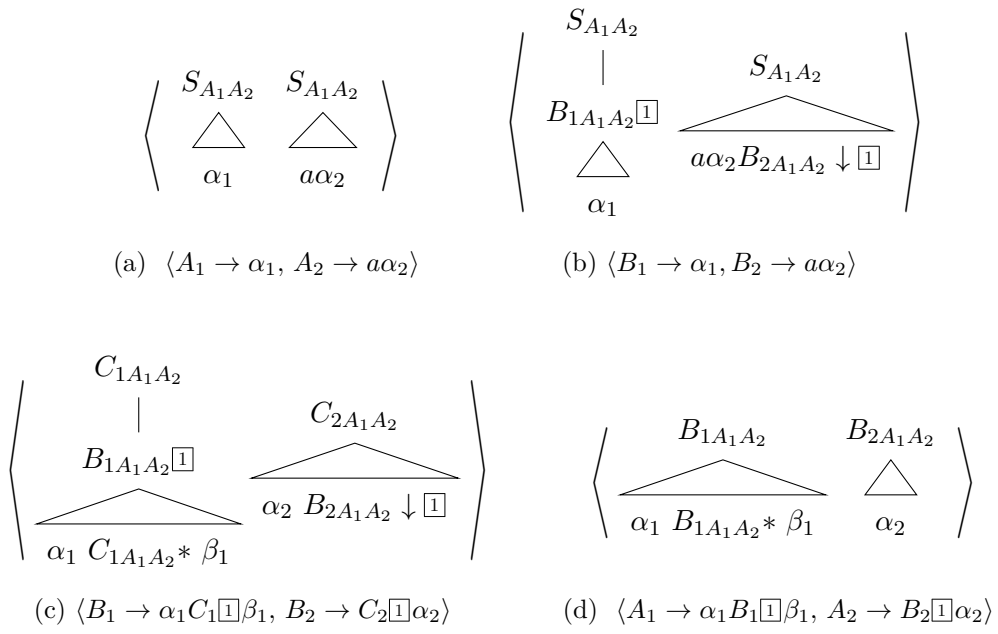
  As a special case, if $B_1 = C_1$ we collapse the root node and adjunction site to produce a tree pair of the following form:

(3.3)



- For each rule in G of the form $\langle A_1 \to \alpha_1 B_1 \boxed{1} \beta_1, A_2 \to B_2 \boxed{1} \alpha_2 \rangle$, $\alpha_1, \alpha_2, \beta_1 \in (N \cup \Sigma)^*$, $B_1, B_2 \in N \setminus \{S\}$, we add a tree pair of the form in Figure 3.2(d).

[2] In all cases, we assume that symbols in N (not $N_{A_1 A_2}$) retain any links they bore in the original grammar, even though they belong to the terminal alphabet in $G_{A_1 A_2}$ and therefore do not participate in rewriting operations. In the final constructed grammar, these symbols will belong to the nonterminal alphabet again, and the links will function normally.



(a) $\langle A_1 \to \alpha_1,\ A_2 \to a\alpha_2\rangle$ (b) $\langle B_1 \to \alpha_1, B_2 \to a\alpha_2\rangle$

(c) $\langle B_1 \to \alpha_1 C_1\boxed{1}\beta_1,\ B_2 \to C_2\boxed{1}\alpha_2\rangle$ (d) $\langle A_1 \to \alpha_1 B_1\boxed{1}\beta_1,\ A_2 \to B_2\boxed{1}\alpha_2\rangle$

Figure 3.2: Tree pairs in $G_{A_1A_2}$ and the rules in G from which they derive.

Lemma 1. *$G_{A_1A_2}$ generates the language $L_{A_1A_2}$.*

Proof. This can be shown by induction over derivations of increasing length. The proof is straightforward but tedious, so we provide only a sketch; the complete proof is provided in Appendix A.

As a base case, observe that a tree of the shape in Figure 3.2(a) corresponds straightforwardly to the derivation

$$\langle A_1\boxed{1}, A_2\boxed{1}\rangle \Rightarrow \langle \alpha_1, a\alpha_2\rangle \tag{3.4}$$

which is a TTLD starting from the pair $\langle A_1, A_2\rangle$. By construction, therefore, every TTLD of the shape in (3.4) corresponds to some tree in $G_{A_1A_2}$ of shape 3.2(a); likewise every derivation in $G_{A_1A_2}$ comprising a single tree of shape 3.2(a) corresponds to a TTLD of the shape in (3.4).

As a second base case, note that a tree of the shape in Figure 3.2(b) corresponds to the last step of a TTLD like (3.5):

$$\langle A_1\boxed{1}, A_2\boxed{1}\rangle \Rightarrow^*_{TTLD} \langle uB_1\boxed{1}v, B_2\boxed{1}w\rangle \Rightarrow \langle u\alpha_1 v, a\alpha_2 w\rangle \tag{3.5}$$

In the other direction, the last step of any TTLD of the shape in (3.5) will involve some rule of the shape $\langle B_1 \to \alpha_1, B_2 \to a\alpha_2\rangle$ for some $B_1, B_2 \in N \setminus \{S\}$; by construction $G_{A_1A_2}$ must contain a corresponding tree pair of shape 3.2(b).

Together, these base cases establish a one-to-one correspondence between single-tree derivations in $G_{A_1A_2}$ and the last step of a TTLD starting from $\langle A_1, A_2\rangle$.

Now, assume that the last n steps of every TTLD starting from $\langle A_1, A_2\rangle$ correspond to some derivation over n trees in $G_{A_1A_2}$, and *vice versa*. Then the last $n+1$ steps of that TTLD will also correspond to some $n+1$ tree derivation in $G_{A_1A_2}$, and *vice versa*.

To see this, consider the step $n+1$ steps before the end of the TTLD. This step may be in the middle of the derivation, or it may be the first step of the derivation. If it is in the middle, then this step must involve a rule of the shape

$$\langle B_1 \to \alpha_1 C_1 \boxed{1} \beta_1, B_2 \to C_2 \boxed{1} \alpha_2 \rangle \tag{3.6}$$

for some nonterminals $B_1, B_2, C_1, C_2 \in N \setminus \{S\}$. The existence of such a rule in G implies the existence of a corresponding tree pair in $G_{A_1A_2}$ of the shape in Figure 3.2(c). Adding this tree to the existing n-tree derivation yields a new $n+1$ tree derivation corresponding to the last $n+1$ steps of the TTLD.[3] In the other direction, if the $n+1$th tree[4] of a derivation in $G_{A_1A_2}$ is of the shape in Figure 3.2(c), then this implies the existence of a production in G of the shape in (3.6). By assumption the first n trees of the derivation in $G_{A_1A_2}$ correspond to some TTLD in G; by prepending the rule from (3.6) to this TTLD we obtain a new TTLD of length $n+1$ which corresponds to the entire $n+1$ tree derivation in $G_{A_1A_2}$.

Finally, consider the case where the TTLD is only $n+1$ steps long. The first step must involve a rule of the form

$$\langle A_1 \to \alpha_1 B_1 \boxed{1} \beta_1, A_2 \to B_2 \boxed{1} \alpha_2 \rangle \tag{3.7}$$

for some $B_1, B_2 \in N \setminus \{S\}$. The existence of such a rule implies the existence of a corresponding tree pair in $G_{A_1A_2}$ of the shape in Figure 3.2(d). Adding this pair to the derivation which corresponds to the last n steps of the TTLD yields a new $n+1$ tree derivation corresponding to the entire $n+1$ step TTLD. In the other direction, if the last tree pair of an $n+1$ tree derivation in $G_{A_1A_2}$ is of the shape in Figure 3.2(d), then this implies the existence of a production in G of the shape in (3.7). By assumption the first n trees of the derivation in $G_{A_1A_2}$ correspond to some TTLD in G; by prepending the rule from (3.7) to

[3] It is easy to verify by inspection of Figure 3.2 that whenever one rule from G can be applied to the output of another rule, then the tree pairs in $G_{A_1A_2}$ which correspond to these rules can compose with one another. Thus we can add the new tree to the existing derivation and be assured that it will compose with one of the trees that is already present.

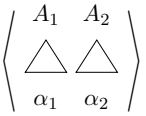
[4] Although trees in $G_{A_1A_2}$ may contain symbols from the nonterminal alphabet of G, these symbols belong to the *terminal* alphabet in $G_{A_1A_2}$. Only nonterminals in $N_{A_1A_2}$ will be involved in this derivation, and by construction there is at most one such nonterminal per tree. Thus a well-formed derivation structure in $G_{A_1A_2}$ will never branch, and we can refer to the $n+1$th tree pair as the one which is at depth n in the derivation structure.

this TTLD we obtain a new TTLD of length $n+1$ which corresponds to the entire $n+1$ tree derivation in $G_{A_1A_2}$.

Taken together, these cases establish a one-to-one correspondence between derivations in $G_{A_1A_2}$ and TTLDs which start from the pair $\langle A_1, A_2 \rangle$. In turn they show, in sketch form, that $G_{A_1A_2}$ generates the desired language $L_{A_1A_2}$. □

Once we have constructed an intermediate grammar $G_{A_1A_2}$ for each pair $A_1A_2 \in N \backslash \{S\}$, we obtain the final STAG H as follows:

1. Convert the input SCFG G to an equivalent STAG. For each rule $\langle A_1 \to \alpha_1, A_2 \to \alpha_2 \rangle$, where $A_i \in N$, $\alpha_i \in (N \cup \Sigma)^*$, create a tree pair of the form

   (3.8) 

   where each pair of linked nonterminals in the original rule become a pair of linked substitution sites in the tree pair. The terminal and nonterminal alphabets and start symbol are unchanged. Call the resulting STAG H.

2. For all $A_1, A_2 \in N \setminus \{S\}$, add all of the tree pairs from the intermediate grammar $G_{A_1A_2}$ to the new grammar H. Expand N to include the new nonterminal symbols in $N_{A_1A_2}$.

3. For every $A_1, A_2 \in N$, in all tree pairs where the target tree's leftmost leaf is labeled with A_2 and this node is linked to an A_1, replace this occurrence of A_2 with $S_{A_1A_2}$. Also replace the linked A_1 node in the source tree.

4. For every $A_1, A_2 \in N$, let $R_{A_1A_2}$ be the set of all tree pairs rooted in $S_{A_1A_2}$, and let $T_{A_1A_2}$ be the set of all tree pairs whose target tree's leftmost leaf is labeled with $S_{A_1A_2}$. For every $\langle s, t \rangle \in T_{A_1A_2}$ and every $\langle s', t' \rangle \in R_{A_1A_2}$, substitute s' and t' into the linked $S_{A_1A_2}$ nodes in s and t, respectively. Add the derived trees to H.

5. For all $A_1, A_2 \in N$, let $T_{A_1A_2}$ be defined as above. Remove all tree pairs in $T_{A_1A_2}$ from H.

6. For all $A_1, A_2 \in N$, let $R_{A_1A_2}$ be defined as above. Remove all tree pairs in $R_{A_1A_2}$ from H.

We now claim that H generates the same language as the original grammar G, and that all of the target side trees in H are prefix lexicalized.

The first claim follows directly from the construction. Step 1 merely rewrites the grammar in a new formalism. From Lemma 1 it is clear that steps 2–3 do not change the generated

language: the set of string pairs generable from a pair of $S_{A_1A_2}$ nodes is identical to the set generable from $\langle A_1, A_2 \rangle$ in the original grammar. Step 4 replaces some nonterminals by all possible alternatives; steps 5–6 then remove the trees which were used in step 4, but since all possible combinations of these trees have already been added to the grammar, removing them will not alter the language.

The second claim follows from inspection of the tree pairs generated in Figure 3.2. Observe that, by construction, for all $A_1, A_2 \in N$ every target tree rooted in $S_{A_1A_2}$ is prefix lexicalized. Thus the trees created in step 4 are all prefix lexicalized variants of non-lexicalized tree pairs; steps 5–6 then remove the non-lexicalized trees from the grammar. ■

Figure 3.3 gives some concrete examples showing the construction of an intermediate grammar from an SCFG. Figures 3.4, 3.5 and 3.6 demonstrate the entire transformation, in steps, as it is applied to a small grammar.

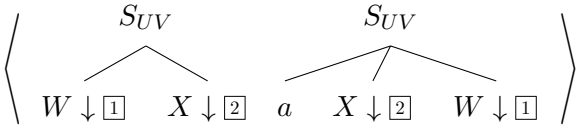
3.3 Complexity & Formal Properties

The set of all grammars produced by our transformation is a subset of the class of prefix lexicalized STAGs in regular form (regular form for TAG is defined in Rogers 1994); we abbreviate this set as PL-RSTAG. This section discusses some formal properties of PL-RSTAG.

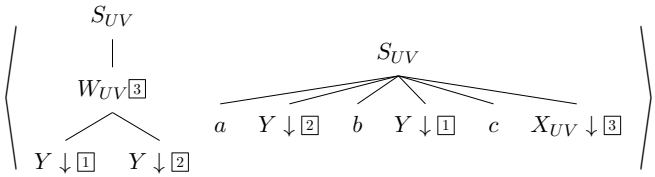
Generative Capacity PL-RSTAG is weakly equivalent to the class of ε-free SCFGs with finitely ambiguous target projections: this follows immediately from the proof that our transformation does not change the language generated by the input SCFG. As an aside, observe that the source and target projections of any STAG produced by our transformation will be TAGs in regular form. Every TAG in regular form is strongly equivalent to some CFG (Rogers, 1994), which gives some intuitive insight into why the STAGs produced by our transformation necessarily generate synchronous context-free languages despite TAGs being able to generate strictly more languages than CFGs in general.[5]

[5]This observation appears at a glance to contradict our previous result that SCFG is not closed under prefix lexicalization: since each side of the transformed STAG is strongly equivalent to some CFG, it should be possible to construct a prefix lexicalized SCFG with these equivalent grammars as source and target projections. This is not possible in the general case, however, because the two grammars will not necessarily derive strings in the same order as one another. That is, the CFG which is equivalent to the STAG's source projection may generate the parts of a string in one order, while the CFG which is equivalent to the target projection generates the parts of the corresponding string in a different order. Thus the nonterminals which would need to be linked together will never be present on both sides of the derivation at the same time as one another and so it will be impossible to link them.

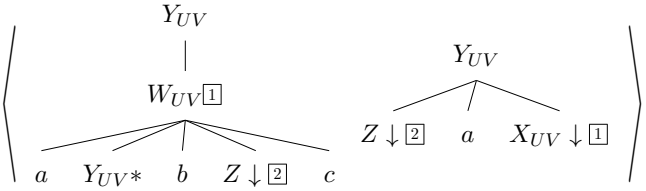
$\langle U \to W\boxed{1}X\boxed{2}, V \to aX\boxed{2}W\boxed{1}\rangle$:



$\langle W \to Y\boxed{1}Y\boxed{2}, X \to aY\boxed{2}bY\boxed{1}c\rangle$:



$\langle W \to aY\boxed{1}bZ\boxed{2}c, X \to Y\boxed{1}Z\boxed{2}a\rangle$:



$\langle U \to aW\boxed{1}bX\boxed{2}c, V \to W\boxed{1}X\boxed{2}a\rangle$:

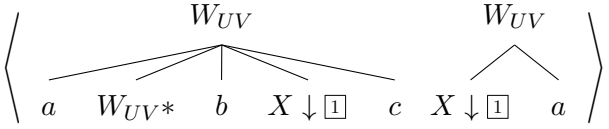
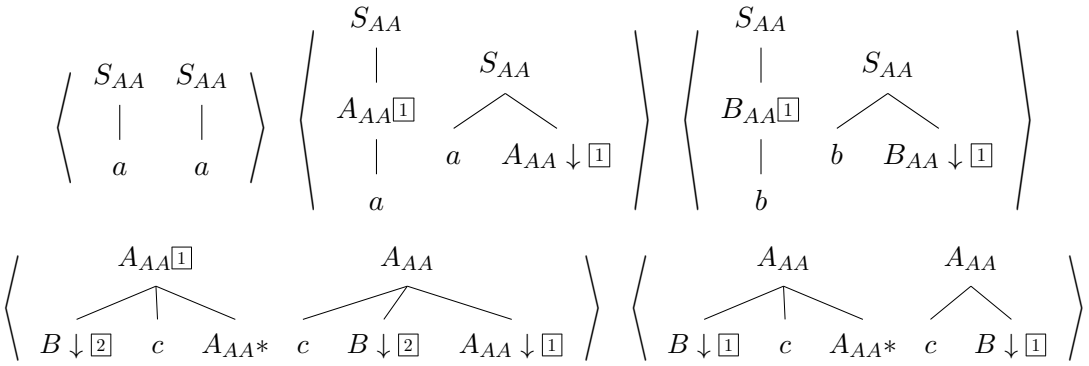


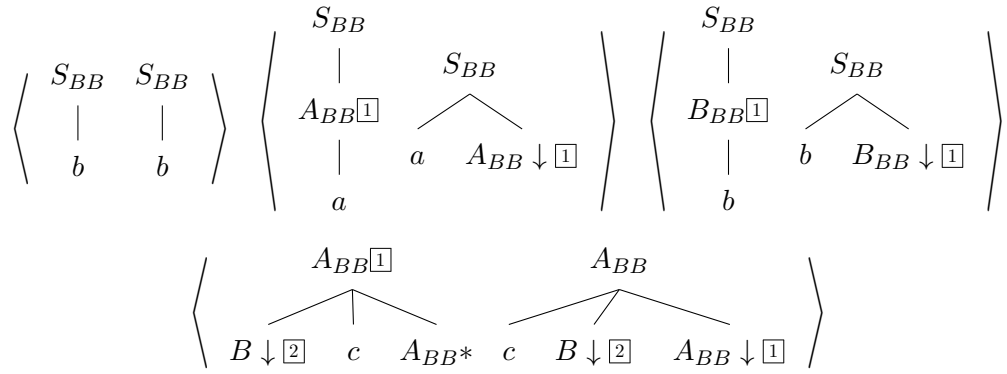
Figure 3.3: Examples showing steps in the construction of an intermediate grammar G_{UV} for a pair of nonterminals U and V. SCFG rules are shown on the left, and a tree pair added to G_{UV} on the basis of each rule is shown to the right. Observe that nonterminals which are not indexed by UV (those which belong to strings abbreviated by Greek letters in Figure 3.2) retain all their original links in the new tree pairs. The nonterminals which are indexed by UV have been added during the construction of the tree pairs. Wrapping adjunction will occur at the new adjunction sites (links $\boxed{3}$ and $\boxed{1}$ in pairs two and three, respectively) which will permit this grammar to generate the same strings as the SCFG it is based on.

$$\langle S \to B\boxed{2}cA\boxed{1}, S \to A\boxed{1}cB\boxed{2} \rangle$$
$$\langle A \to B\boxed{2}cA\boxed{1}, A \to A\boxed{1}cB\boxed{2} \rangle$$
$$\langle A \to a, A \to a \rangle$$
$$\langle B \to b, B \to b \rangle$$

(a)



(b)



(c)

Figure 3.4: An SCFG (a) and the intermediate grammars G_{AA} (b) and G_{BB} (c) produced while lexicalizing it. The grammars G_{AB} and G_{BA} are omitted, as there are no rules in the original grammars which rewrite the pairs $\langle A, B \rangle$ or $\langle B, A \rangle$.

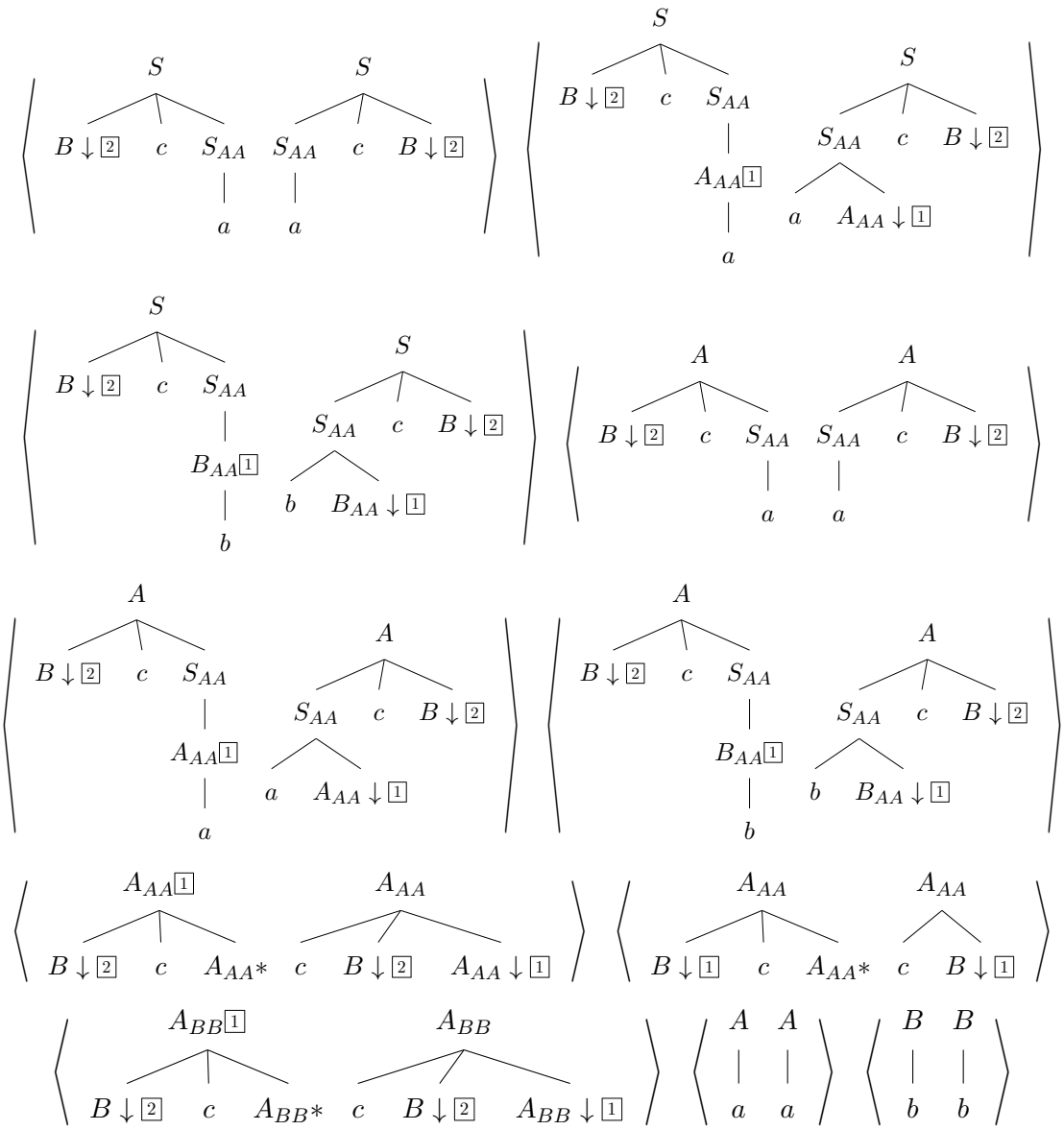


Figure 3.5: The complete STAG produced by lexicalizing the grammar in Figure 3.4.

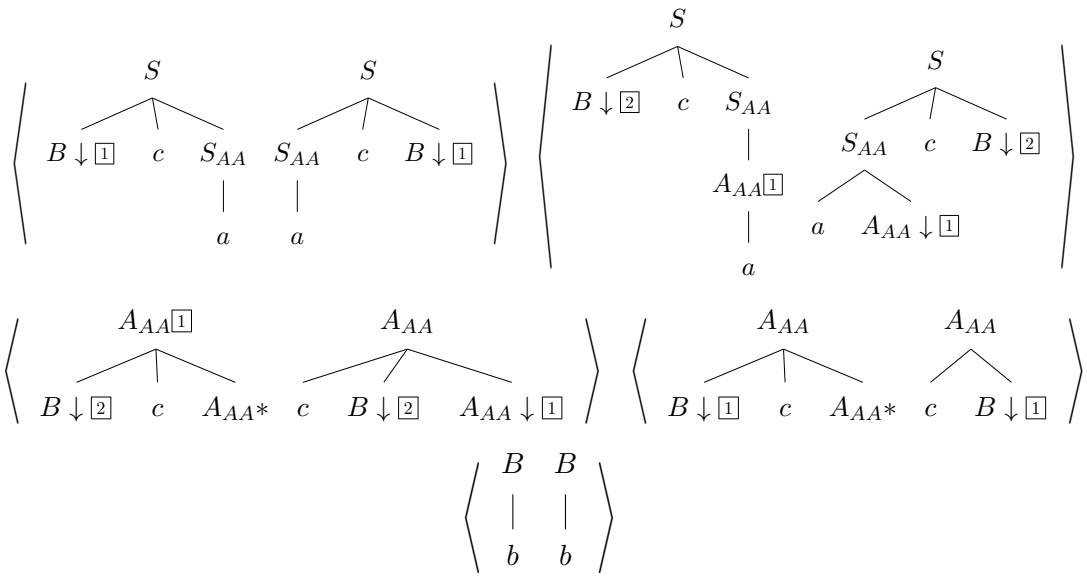


Figure 3.6: The STAG from Figure 3.5 with unreachable and unproductive trees omitted.

Alignments and Reordering PL-RSTAG generates the same set of reorderings (alignments) as SCFG. Observe that our transformation does not cause nonterminals which were linked in the original grammar to become unlinked. Thus subtrees which are generated by linked nonterminals in the original grammar will still be generated by linked nonterminals in the final grammar, so no reordering information is lost or added.[6]

Grammar Rank If the input SCFG G has rank k, then applying our transformation directly to G will produce an STAG H with rank at most $2k$. To see this, observe that the construction of the intermediate grammars increases the rank by at most 1, as seen in Figure 3.2(b). When a prefix lexicalized tree is substituted at the left edge of a non-lexicalized tree, the link on the substitution site will be consumed, but up to $k+1$ new links will be introduced by the substituting tree, so that the final tree will have rank at most $2k$.

By modifying the input grammar slightly before applying our transformation, however, it is possible to obtain an STAG of rank $k+1$ rather than $2k$. To achieve this, for every rule in the original grammar of the form in (3.9)

$$\langle A_1 \to \alpha_1, A_2 \to a\alpha_2 \rangle \tag{3.9}$$

[6]Although we consume one link whenever we substitute a prefix lexicalized tree at the left edge of an unlexicalized tree, that link can still be recorded and used to reconstruct the reorderings which occurred between the two sentences.

for some $a \in \Sigma$, $\alpha_i \in (N \cup \Sigma)^*$, we remove this rule from the grammar and replace it with the following pair of rules:

$$\langle A_1 \rightarrow A'_1 \boxed{1}, A_2 \rightarrow a A'_2 \boxed{1} \rangle \tag{3.10}$$

$$\langle A'_1 \rightarrow \alpha_1, A'_2 \rightarrow \alpha_2 \rangle \tag{3.11}$$

where A'_1 and A'_2 are new nonterminals which appear only in this pair of productions.

Now every prefix lexicalized rule in the original grammar has rank 1, which means that all of the initial trees constructed in the intermediate grammars will have rank at most 2. Thus when we substitute these trees at the left edge of unlexicalized tree pairs, we only increase the rank to $k + 1$ rather than to $2k$.

Parse Complexity Because the grammar produced is in regular form, each side can be parsed in time $O(n^3)$ (Rogers, 1994), for an overall parse complexity of $O(n^{3k})$, where n is sentence length and k is the grammar rank.

Grammar Size If H is the PL-RSTAG produced by applying our transformation to an SCFG G, then H contains $O(|G|^3)$ elementary tree pairs, where $|G|$ is the number of synchronous productions in G. When the set of nonterminals N is small relative to $|G|$, a tighter bound is given by $O(|N|^2|G|^2)$.

To derive these bounds, observe that each intermediate grammar contains $O(|G|)$ tree pairs, since we create at most two pairs per production in G. There are $|N|^2$ such grammars, so that adding the trees from each intermediate grammar will increase the size to $O(|N|^2|G|)$ tree pairs. Finally, since there are $O(|G|)$ prefix lexicalized tree pairs in each intermediate grammar, substituting these at the left edge of productions which are not lexicalized will increase the grammar size by another factor of $O(|G|)$ to $O(|N|^2|G|^2)$. In the worst case where each pair of nonterminals appears as the left hand side of some production, this simplifies to $O(|G|^3)$.

To supplement these asymptotic bounds, we have also performed an empirical evaluation to determine the actual size increase incurred by our transformation on some grammars taken from existing NLP applications. These evaluations are detailed in the following section.

3.4 Experiments

The asymptotic bounds in Section 3.3 show that applying our transformation to an SCFG containing $|G|$ synchronous productions produces an STAG containing at most $O(|G|^3)$ tree pairs. In this section, we empirically show that the size increase on grammars which are used in real-world NLP applications is much smaller, always within $O(|G|^2)$.

General Results and Effect of $|N|$ Table 3.1 shows the actual size increase incurred by applying our transformation to two likely use cases: here $|G|$ is the size of the initial grammar, $|H|$ is the size after applying our transformation, and $\log_{|G|} |H|$ is a measure of the size increase as a power of the original grammar size. The table also shows p_{pl}, which is the percentage of rules in the original SCFG which were already prefix lexicalized before applying our transformation.

The first grammar used for this experiment is a stochastic bracketing ITG (SBITG; Wu 1997), a simple kind of SCFG which can be used for word or phrase alignment. Previous work (Zhang and Gildea, 2005) has shown that lexicalizing these grammars leads to better alignments, suggesting SBITGs as a likely use case for our transformation. We also consider the grammar used by Siahbani and Sarkar (2014b), which was created for a Chinese-English translation task known to involve complex reorderings which cannot be modeled by a prefix lexicalized SCFG (Siahbani and Sarkar, 2014a). This therefore represents another likely use case for the transformation developed in this work.

Grammar	$\|G\|$	$\|H\|$	p_{pl}	$\log_{\|G\|} \|H\|$
SBITG (10000 translation pairs)	10k	170k	99.9%	1.31
Siahbani and Sarkar (2014b) (Zh-En)	18.5M	23.6T	37.1%	1.84

Table 3.1: Grammar sizes before and after prefix lexicalization, showing sub-quadratic growth instead of the worst case cubic growth. $|G|$ and $|H|$ are the grammar size before and after lexicalization; p_{pl} is the percentage of the rules in the original SCFG which were already prefix lexicalized before applying our transformation; $\log_{|G|} |H|$ is the size increase expressed as a power of the initial size.

These experiments show that, on the two use cases we have considered, the grammar size increase incurred by our transformation is significantly smaller than the worst case. We believe that this is due to the fact that $|N|$ is small relative to $|G|$ in both cases: since the overall size increase is bounded by $O(|N|^2|G|^2)$, we should expect at most quadratic rather than cubic growth when $|N|$ is small. Given that stochastic bracketing ITGs and ITGs used for machine translation tend to have at most two nonterminals, it is not surprising that our transformation causes such small growth when applied to such grammars.

We offer this as an explanation for the small growth shown in Table 3.1, but we did not perform additional experiments to precisely determine the effect of $|N|$ on our transformation. This is because we are not aware of any real-world NLP applications which use grammars with a large $|N|$: formal grammars tend to be used as a means for compactly generalizing across common linguistic patterns, so it is in a grammar designer (or learner)'s interests to make rules as generic as possible. That is, there is incentive to use few, generic constructions rather than many, finely-grained ones, and any grammar constructed in this way may be expected to grow approximately quadratically under our transformation.

Effect of $|G|$ To test the effect of initial grammar size on our transformation, we sampled grammars of various sizes from the SCFG used by Siahbani and Sarkar (2014b). Each sample was constructed to contain 50% prefix lexicalized rules and 50% non-prefix lexicalized rules; in this way it is guaranteed that the size of the grammar is the only factor that varies between samples. Each sample was subjected to the prefix lexicalizing transformation described in this chapter, and the size of the resulting STAG H was calculated. Note that useless tree pairs (those which are unreachable or unproductive) were not pruned from the final STAG, so the numbers reported in this section represent the size of the grammar exactly as it is returned by our algorithm.

Figure 3.7 shows the results of this experiment. We find that small grammars grow disproportionately slower than larger ones: our smallest sample (1,000 synchronous rules) grows by a power of 1.92 while the largest (100,000 synchronous rules) grows by a power of nearly 1.95. The size increase levels off as the initial grammar size increases, tending towards an asymptote around 1.95.

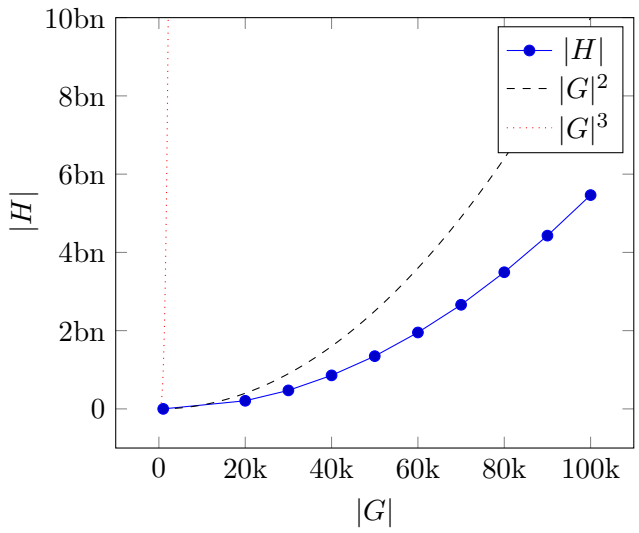
Effect of p_{pl} We also investigated the effect of p_{pl}, the percentage of productions in the initial SCFG which were already prefix lexicalized before applying our transformation. We extracted samples of 15,000 synchronous rules from the grammar used by Siahbani and Sarkar (2014b), where each sample varied in the number of prefix lexicalized rules it contained. Each sample was subjected to our prefix lexicalization and the size increase was recorded; as in the previous experiment, unproductive and unreachable trees were left in the transformed grammar. The results of this experiment are shown in Figure 3.8.

Observe that the size of the final STAG does not vary linearly with respect to p_{pl}. Counterintuitively, we observe the largest growth when exactly half of the original SCFG is already prefix lexicalized, and considerably smaller growth when most of the grammar is or is not lexicalized.

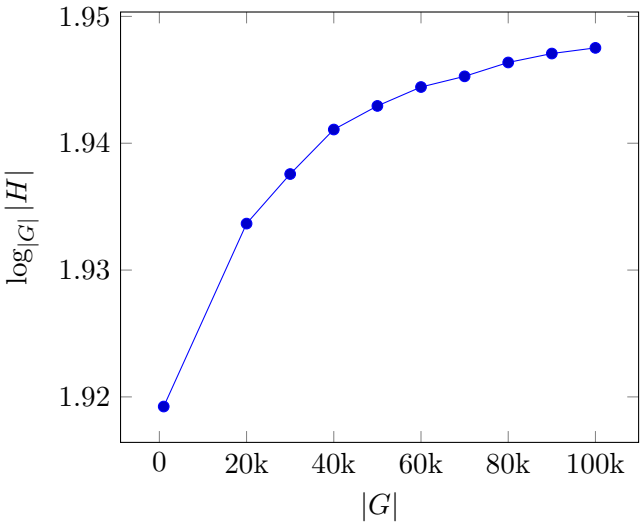
We believe this results from an interaction between two separate factors affecting the size of the final grammar. For every prefix lexicalized rule in the initial SCFG, we create several lexicalized tree pairs in the intermediate grammars, which increases the final blowup when these trees are substituted into unlexicalized tree pairs. At the same time, for every rule which is *not* prefix lexicalized in the original SCFG, we must construct a family of lexicalized tree pairs around this rule, again increasing the size of the final grammar. When half of the input SCFG is prefix lexicalized, there are both a large number of rules needing lexicalization and a large number of rules with which to lexicalize them. Thus the blowup is greatest in this case, analogous to how the function $f(x) = x(1-x)$ takes its maximum at $x = \frac{1}{2}$.

We have now established the general properties of the grammars produced by our transformation. Next, we briefly discuss how to use these grammars in the tasks considered by

Watanabe et al. (2006) and Siahbani et al. (2013), and we consider other work related to these results.

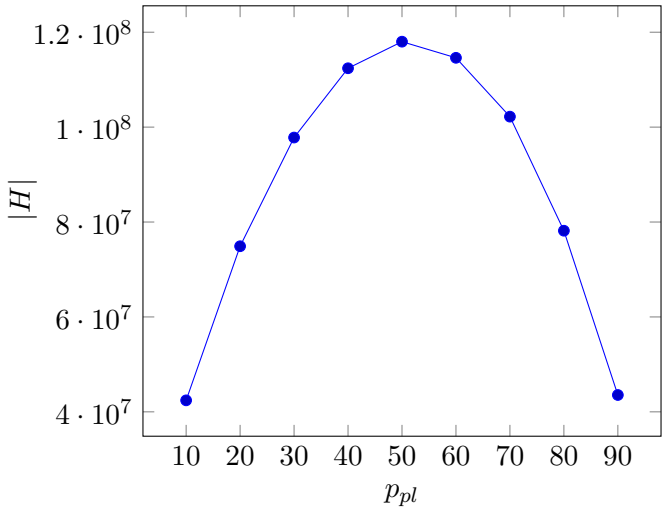


(a) Final grammar size ($|H|$) vs. initial grammar size ($|G|$). Lines depicting $O(n^2)$ growth and $O(n^3)$ (worst-case) growth are also given for reference.

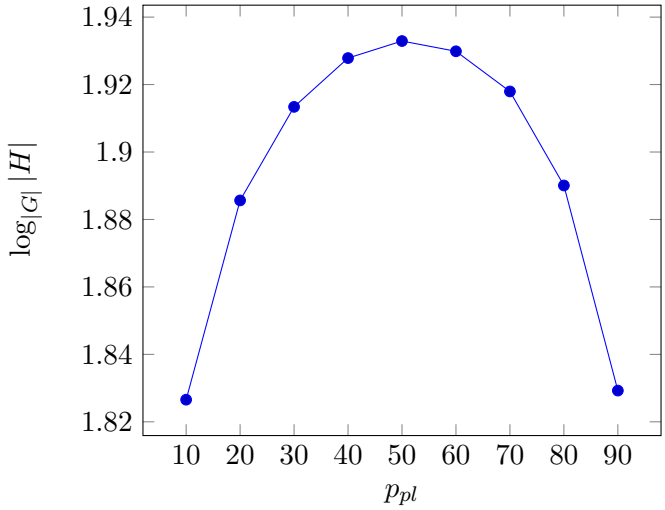


(b) Size increase ($\log_{|G|} |H|$) vs. initial grammar size ($|G|$).

Figure 3.7: Effect of $|G|$ (initial grammar size) on overall size increase.



(a) Final grammar size ($|H|$) vs. percentage of prefix lexicalized rules in the initial grammar (p_{pl}).



(b) Size increase ($\log_{|G|} |H|$) vs. percentage of prefix lexicalized rules in the initial grammar (p_{pl}).

Figure 3.8: Effect of p_{pl} (the percentage of prefix lexicalized rules in the initial grammar) on overall size increase.

3.5 Applications to Translation

The LR decoding algorithm from Watanabe et al. (2006) relies on prefix lexicalized rules to generate a prefix of the target sentence during machine translation. At each step, a translation hypothesis is expanded by rewriting the leftmost nonterminal in its target string using some grammar rule; the prefix of this rule is appended to the existing translation and the remainder of the rule is pushed onto a stack, in reverse order, to be processed later. Translation hypotheses are stored in stacks according to the length of their translated prefix, and beam search is used to traverse these hypotheses and find a complete translation. During decoding, the source side is processed by an Earley-style parser, with the dot moving around to process nonterminals in the order they appear on the target side.

Since the trees on the target side of our transformed grammar are all of depth 1, and none of these trees can compose via the adjunction operation, they can be treated like context-free rules and used as-is in this decoding algorithm. The only change required to adapt LR decoding to use a PL-RSTAG is to make the source side use a TAG parser instead of a CFG parser; an Earley-style parser for TAG already exists (Joshi and Schabes, 1997), so this is a minor adjustment. Pseudocode for this decoding algorithm is provided in appendix B, with the changes required to handle PL-RSTAG highlighted.

Combined with the transformation in Section 3.2, this suggests a method for using LR decoding without sacrificing translation quality. Previously, LR decoding required the use of heuristically generated PL-SCFGs, which cannot model some reorderings (Siahbani and Sarkar, 2014b). Now, an SCFG tailored for a translation task can be transformed directly to PL-RSTAG and used for decoding; unlike a heuristically induced PL-SCFG, the transformed PL-RSTAG will generate the same language as the original SCFG so translation quality should not be affected. It remains to empirically test this hypothesis, however, and to determine whether the increased grammar rank or larger set of non-terminals offsets the speed improvements gained by using LR decoding. See Appendix B for comments on the work involved in such an evaluation.

3.6 Related Work

The transformation discussed in this chapter continues a long line of work studying lexicalized TAGs (e.g. Joshi et al. 1975; Schabes and Waters 1993). Schabes and Waters (1995) show that TAG can strongly lexicalize CFG, whereas CFG only weakly lexicalizes itself; we show a similar result for SCFGs. Kuhlmann and Satta (2012) show that TAG is not closed under strong lexicalization, and Maletti and Engelfriet (2012) show how to strongly lexicalize TAG using simple context-free tree grammars.

We employ STAG to overcome the limited power of SCFG. STAG has previously been used in a similar fashion by Johnson and Charniak (2004) and Zwarts et al. (2010), who employ STAGs to repair speech errors which cannot be accurately modeled by SCFGs.

Other extensions of GNF to new grammar formalisms include Dymetman (1992) (for definite clause grammars), Hoogeboom (2002) (for CF valence grammars), and Engelfriet et al. (2017) (for multiple CFGs).

Lexicalization of synchronous grammars was addressed in Zhang and Gildea (2005) for SCFGs of rank 2, but these authors consider lexicalization rather than prefix lexicalization. Furthermore, their transformation is not a lexicalization in the formal sense, because they do not introduce a terminal symbol into every production; rather, they annotate the nonterminals in the grammar to record which terminal symbols they will eventually produce.

Analogous to our closure result, Aho and Ullman (1969) prove that SCFG does not admit a normal form with bounded rank like Chomsky normal form.

Blum and Koch (1999) use intermediate grammars like our $G_{A_1 A_2}$s to transform a CFG to GNF. Another GNF transformation (Rosenkrantz, 1967) is used by Schabes and Waters (1995) to define tree insertion grammars (which are also weakly equivalent to CFG).

We rely on Rogers 1994 to show that our transformed grammars generate context-free languages despite allowing wrapping adjunction; an alternative argument could employ the results of Swanson et al. 2013, who develop their own context-free TAG variant known as osTAG.

3.7 Conclusion

This chapter has shown that SCFG is not closed under prefix lexicalization, and has presented a technique for converting an SCFG into an equivalent prefix lexicalized STAG. We have given an overview of the formal properties of the resulting grammar and considered possible applications for the transformation. We now turn to the problem of lexicalizing grammars with weighted productions.

Chapter 4

Weighted Grammar Lexicalization

Section 3.2 demonstrated a method for prefix lexicalizing an ε-free SCFG with a finitely ambiguous target projection by converting it to an equivalent STAG. This chapter demonstrates that the same transformation may be applied to a weighted or probabilistic SCFG without affecting the weight assigned to string pairs generated by that grammar.

4.1 Weighted SCFG

We begin by considering the case where our transformation is applied to a WSCFG. Recall that every tree pair in the intermediate grammars and the final prefix lexicalized STAG H will be created on the basis of some synchronous rule from the original WSCFG G. Suppose that each of these tree pairs is assigned the same weight as the WSCFG rule on which it is based. Then the following result obtains:

Lemma 2. *For every $A_1, A_2 \in N$, whenever a derivation in the intermediate grammar $G_{A_1 A_2}$ corresponds to a TTLD in G, both derivations have the same weight.*

Proof. This is easily seen by adapting the proof in Appendix A to record the weight of the string pair being derived at each step of the induction. Every time a derivation in $G_{A_1 A_2}$ (resp. a TTLD in G) is extended by adding a new tree pair (WSCFG rule), this pair (rule) will by construction have the same weight as the corresponding WSCFG rule (tree pair). Thus the weight of the corresponding derivations will remain equal at every step of the induction. This result holds for the usual case where the grammar weights belong to $\mathbb{R}$, but can trivially be generalized to a grammar with weights in a commutative semiring. □

Now, suppose as well that when we lexicalize a tree pair by substituting a lexicalized tree pair at the left corner, we assign the combined tree pair a weight equal to the product of the weights of the component tree pairs. Then it follows immediately that the weight assigned to a given string pair by the final prefix lexicalized WSTAG H will be the same as the weight assigned by the original grammar G.

To see this, note first that rewriting the original SCFG as an STAG trivially preserves the weight of every derivation. Next, recall that Lemma 1 established a one-to-one correspondence between derivations in the intermediate grammars and those in the original WSCFG G, while Lemma 2 shows that this correspondence conserves weight. Thus when we replace a pair of linked $\langle A_1, A_2 \rangle$ nodes with a pair of $\langle S_{A_1 A_2}, S_{A_1 A_2} \rangle$ nodes during the transformation, we change neither the number of derivations which can proceed from that pair of nodes nor the weight of these derivations. Finally, by multiplying together the tree weights whenever we substitute a prefix lexicalized tree pair into these $\langle S_{A_1 A_2}, S_{A_1 A_2} \rangle$ nodes, we perform the same computation which would have been performed if these tree pairs composed during a normal derivation. We have simply performed this computation during grammar construction, rather than during the derivation as would be more common.

Thus for any weighted SCFG, our transformation does not affect the weight of the string pairs generated by the grammar.

4.2 Probabilistic SCFG

Since every PSCFG is also a WSCFG, the preceding result guarantees that we can apply our transformation to a PSCFG without affecting the probabilities of the strings generated by the grammar. However, there is no guarantee that the final STAG produced by our transformation will still be a PSTAG — although the weight of each string pair generated by the final grammar will be unchanged, the weights on the individual tree pairs may be changed so that they no longer sum to 1.

It is easy to find cases where this issue occurs: for example, when the PSCFG in (4.1) is lexicalized, the final STAG will contain exactly six tree pairs rooted in $\langle B_{AA}, B_{AA} \rangle$, shown in Figure 4.1. Observe that the weights on these tree pairs do not sum to 1, so this will be a WSTAG rather than a PSTAG:

$$
\begin{array}{ll}
\langle S \to A\boxed{1}, S \to A\boxed{1} \rangle & p = 1 \\
\langle A \to a, A \to a \rangle & p = 0.75 \\
\langle A \to B\boxed{1}A\boxed{2}, A \to B\boxed{1}A\boxed{2} \rangle & p = 0.25 \\
\langle B \to b, B \to b \rangle & p = 1
\end{array}
\tag{4.1}
$$

Thus if we wish the final grammar to be probabilistic, we must perform renormalization after applying our transformation.

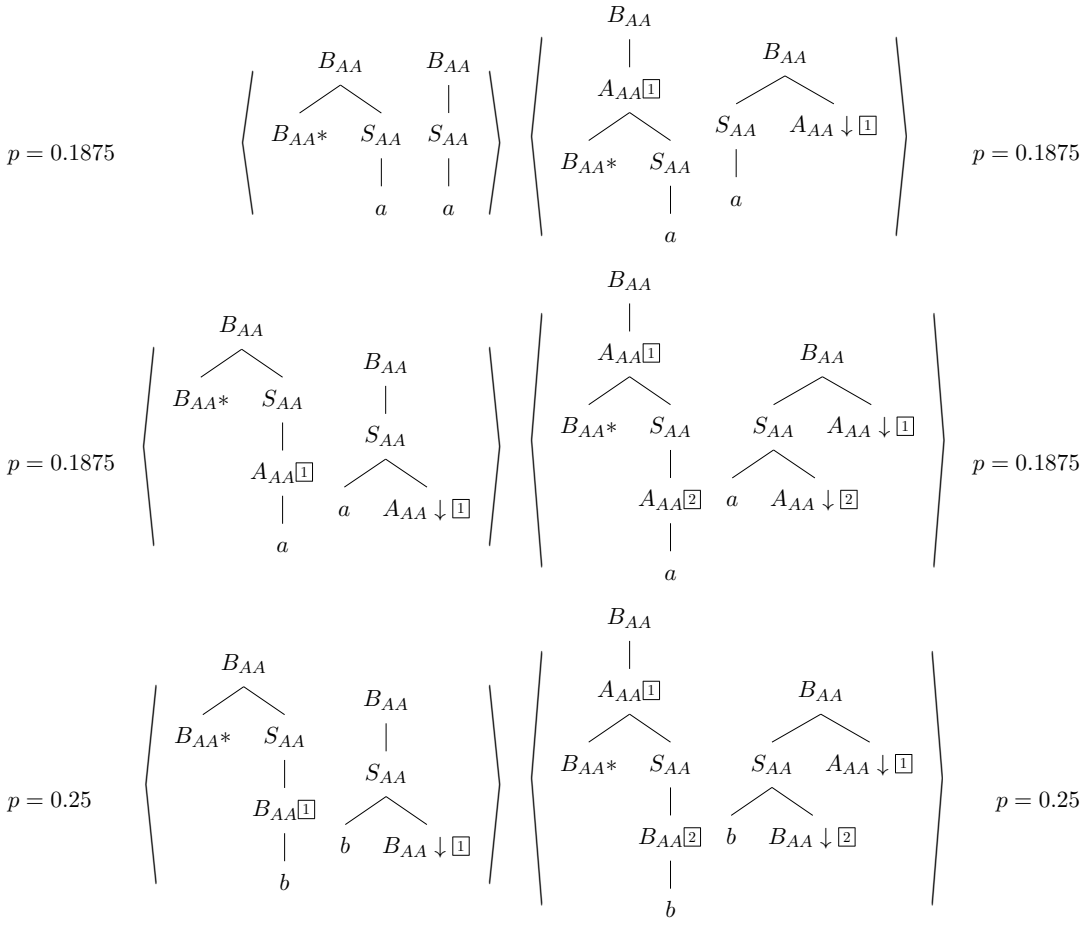


Figure 4.1: Tree pairs rooted in B_{AA} created by prefix lexicalizing (4.1)

Given a WSTAG H obtained by lexicalizing the PSCFG G, we renormalize H as follows. First, let $\Omega(H)$ be the set of all tree pairs which can be derived by the grammar H, and let $\Omega(G)$ be defined likewise for G. Define H's *partition function* as

$$Z_H = \sum_{\langle \sigma, \tau \rangle \in \Omega(H)} w(\sigma, \tau) \tag{4.2}$$

This is the sum of the weights of every tree pair generated by the grammar H. Likewise, Z_G is the partition function for G, and is equal to the sum of the weights of every string pair generated by G.

Because the weight assigned to string pairs by H is the same as the weight assigned by the original grammar G, the following equality holds:

$$Z_H = \sum_{\langle \sigma, \tau \rangle \in \Omega(H)} w(\sigma, \tau) = \sum_{\langle \sigma, \tau \rangle \in \Omega(G)} p(\sigma, \tau) = Z_G \tag{4.3}$$

That is, the partition function for the WSTAG H equals the partition function for the original PSCFG G. From this it is easily shown that $Z_H \leq 1$, because G is a probabilistic grammar and therefore $Z_G \leq 1$ (cf. Smith and Johnson 2007).

Next, given a pair of nonterminals $X, Y \in N$, let $\Omega(X, Y)$ be the set of all tree pairs rooted in $\langle X, Y \rangle$ that can be derived using H. We define a partition function for the pair X, Y as

$$Z_{XY} = \sum_{\langle \sigma, \tau \rangle \in \Omega(X,Y)} w(\sigma, \tau) \tag{4.4}$$

Now we show that Z_{XY} is finite for every pair $X, Y \in N$. This is necessary because we will later use $1/Z_{XY}$ as a normalizing constant. In the simplest case, $Z_{SS} = Z_H$ is simply the partition function for the entire grammar, which is already known to be at most 1. Next suppose $\langle \sigma, \tau \rangle$ is an initial tree pair from H rooted in $\langle S, S \rangle$ with weight $w_{\langle \sigma, \tau \rangle}$. Let σ_i be the ith nonterminal[1] occurring in σ, and let τ_i be the nonterminal in τ which is linked to σ_i. Then, if there are n nonterminals in $\langle \sigma, \tau \rangle$, the sum of the weights of all tree pairs rooted in $\langle \sigma, \tau \rangle$ is $w_{\langle \sigma, \tau \rangle} \cdot Z_{\sigma_1 \tau_1} \cdots Z_{\sigma_n \tau_n}$. The following inequality holds:

$$Z_{SS} \geq w_{\langle \sigma, \tau \rangle} \cdot Z_{\sigma_1 \tau_1} \cdots Z_{\sigma_n \tau_n} \tag{4.5}$$

[1]Here we require only that some total ordering is imposed on the nonterminals in σ, such that "the ith nonterminal" is well-defined. The actual choice of ordering is not relevant to the proof. One option is to order the nonterminals according to the order in which they are encountered in a depth-first traversal of the tree.

The inequality is tight just in case $\langle \sigma, \tau \rangle$ is the only tree pair rooted in $\langle S, S \rangle$. Since Z_{SS} is finite, and $Z_{\sigma_i \tau_i} > 0$ for $1 \leq i \leq n$ it follows that every $Z_{\sigma_i \tau_i}$ must also be finite.[2] Furthermore, we may assume without loss of generality that H contains no unreachable trees, so that for any pair of nonterminals $\langle X, Y \rangle$ occurring at the root of some tree pair in H, there exists a sequence of adjunction and substitution operations which derive a tree pair containing $\langle X, Y \rangle$ starting from a tree pair rooted in $\langle S, S \rangle$. By induction, we can obtain an inequality resembling (4.5) for each pair of nonterminals occurring along this sequence, including $\langle X, Y \rangle$ itself. Thus $Z_{XY} \leq 1$ for every $X, Y \in N$.

Now, for every tree pair $\langle \sigma, \tau \rangle$ in H, let σ_i, τ_i, n, and $w_{\langle \sigma, \tau \rangle}$ be defined as before. Let $\langle X, Y \rangle$ be the pair of nonterminals at the root of $\langle \sigma, \tau \rangle$, and define a new weight $w'_{\langle \sigma, \tau \rangle}$ as

$$w'_{\langle \sigma, \tau \rangle} = \frac{w_{\langle \sigma, \tau \rangle} \prod_{i=1}^{n} Z_{\sigma_i \tau_i}}{Z_{XY}} \tag{4.6}$$

Since we proved that the partition functions are all finite and nonzero, the value $w'_{\langle \sigma, \tau \rangle}$ will be well defined. Furthermore, these new weights define a probability distribution over every pair of nonterminals in H. To see that this is true, let $\omega(X, Y)$ denote the set of all elementary tree pairs in H rooted in $\langle X, Y \rangle$, and observe that the following equality is true for every pair $X, Y \in N$:

$$\sum_{\langle \sigma, \tau \rangle \in \omega(X,Y)} w'_{\langle \sigma, \tau \rangle} = \frac{1}{Z_{XY}} \sum_{\langle \sigma, \tau \rangle \in \omega(X,Y)} w_{\langle \sigma, \tau \rangle} \prod_{i=1}^{n} Z_{\sigma_i \tau_i} = \frac{1}{Z_{XY}} \sum_{\langle \sigma, \tau \rangle \in \Omega(X,Y)} w(\sigma, \tau) = 1 \tag{4.7}$$

Thus reweighting every tree pair $\langle \sigma, \tau \rangle$ in H using the weight $w'_{\langle \sigma, \tau \rangle}$ will make the grammar a PSTAG instead of a WSTAG.

Note that the partition functions Z_{XY} are related to one another by equations of the form

$$Z_{XY} = \sum_{\langle \sigma, \tau \rangle \in \omega(X,Y)} w_{\langle \sigma, \tau \rangle} \prod_{i=1}^{n} Z_{\sigma_i \tau_i} \tag{4.8}$$

These comprise a system of nonlinear polynomial equations which must be solved in order to compute the normalized weights for H. In the general case, a numerical solver may be required to compute the solution to this system; we will not comment on the most effective way to solve these equations, but will direct the reader to Nederhof and Satta (2008), Smith and Johnson (2007), and Chi (1999), who address related problems for non-synchronous grammars.

[2] We assume without loss of generality that the partition functions are nonzero. If Z_{XY} is zero for some $X, Y \in N$, then every tree pair rooted in $\langle X, Y \rangle$ will be unproductive and can be removed from the grammar. By removing all unproductive and unreachable tree pairs from the grammar we ensure that only nonterminals with nonzero partition functions will remain.

There are, fortunately, cases in which the partition functions are easily solved through purely analytic means; these include grammars with one non-terminal (ignoring the start symbol), and those with two non-terminals (ignoring the start symbol) where each nonterminal appears on only one side of the grammar. Stochastic bracketing ITGs belong to the first category, so given the grammar depicted in 4.2 we may easily solve for the normalized grammar weights to obtain the PSTAG shown in Figure 4.3.

Note as well that renormalizing the grammar weights requires dividing the original weights by the values of the partition functions Z_{XY}. This procedure is therefore only applicable to grammars whose weights belong to an algebraic structure where multiplicative inverses are well-defined. Thus, although our transformation will preserve the weights of string pairs whenever the original grammar weights come from a commutative semiring, we cannot necessarily renormalize the grammar in this case, because elements in a semiring do not have multiplicative inverses. In NLP applications, however, weights are typically taken from a field (most commonly $\mathbb{R}$), making normalization possible.

We have now shown that our prefix lexicalizing transformation may be applied to weighted or probabilistic SCFGs, and that the resulting WSTAG will assign the same weights to string pairs as did the original SCFG. We can also normalize the resulting WSTAG to obtain a PSTAG; finding the normalized weights may require solving a system of nonlinear polynomial equations, but for grammars such as stochastic bracketing ITGs the system is simple enough to solve analytically.

$$
\begin{array}{ll}
\langle S \to X\boxed{1}, S \to X\boxed{1} \rangle & p = 1 \\
\langle X \to X\boxed{1}X\boxed{2}, X \to X\boxed{1}X\boxed{2} \rangle & p = p_1 \\
\langle X \to X\boxed{1}X\boxed{2}, X \to X\boxed{2}X\boxed{1} \rangle & p = p_2 \\
\langle X \to a_1, X \to b_1 \rangle & p = u_1 \\
\langle X \to a_2, X \to b_2 \rangle & p = u_2 \\
\qquad\qquad \cdots & \\
\langle X \to a_n, X \to b_n \rangle & p = u_n
\end{array}
$$

Figure 4.2: A stochastic bracketing ITG. The grammar consists of two reordering rules with probabilities p_1 and p_2, plus n translation pairs.

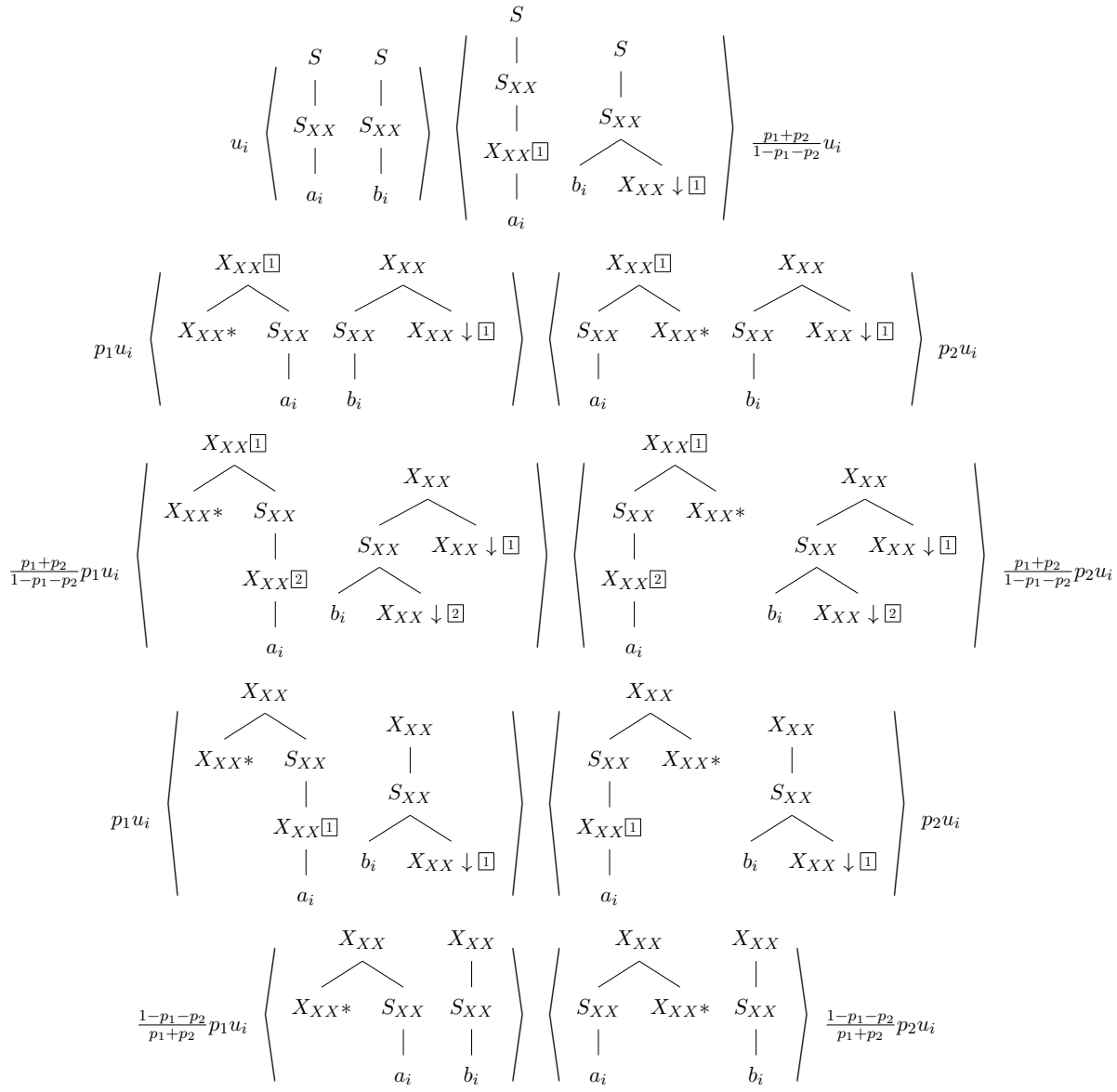


Figure 4.3: Tree-pairs produced by prefix lexicalizing the ITG in Figure 4.2. A copy of the above trees is created for each translation pair a_i, b_i; assuming a vocabulary of size n, this gives a final grammar of $10n$ tree pairs, compared to the original grammar size of $n + 2$ rules.

Chapter 5

Conclusion & Future Work

We have demonstrated a method for prefix lexicalizing an SCFG by converting it to an equivalent STAG. This process is applicable to any SCFG which is ε-free and has a finitely ambiguous target projection. Our transformation may be considered a generalization of the (extended) Greibach normal form transformation to synchronous grammars, and like the original GNF transformation for CFGs our construction at most cubes the grammar size. However, when applied to the kinds of synchronous grammars used in machine translation, the size is merely squared. Our transformation preserves all of the alignments generated by SCFG, and only increases the rank of the grammar by 1. By using a version of STAG that is weakly equivalent to SCFG, we additionally manage to retain $O(n^{3k})$ parsing complexity for grammars of rank k.

We have also shown that our transformation may be applied to a weighted or probabilistic SCFG without affecting the weights of the string pairs generated by the grammar. Renormalizing the grammar after the transformation may require solving a system of non-linear polynomial equations, but for common grammars such as stochastic bracketing ITGs these equations are easy to solve analytically.

As future work we wish to re-examine the questions in this thesis from the perspective of algebraic formal language theory, in order to give our results a better formal grounding. More precisely, we wish to consider the results of Ésik and Leiß (2005), who show the existence of a normal form in all semirings where least-fixed-points satisfy certain properties. The semiring of context-free languages over an alphabet Σ satisfies these properties, which implies the existence of the classic GNF theorem for CFGs. Notably, however, the normal form developed by Ésik and Leiß is "applicable to all algebraically complete semirings" (Ésik and Leiß 2005:192), not just the context-free languages; thus there is a good theoretical basis for considering this normal form a generalization of GNF.

This suggests the following course for future work. First, we seek to formally define the semiring of synchronous context-free languages over an alphabet Σ, which is a natural extension of the semiring of context-free languages over the same alphabet. Next, we wish to determine what are the formal properties of this semiring: is it algebraically complete,

and does it satisfy the properties required for the existence of Ésik and Leiß's normal form? If not, this yields more insight into the reasons why SCFG does not admit an analogue to GNF. If the semiring *does* satisfy these properties, then we may determine what Ésik and Leiß's normal form looks like when applied to this semiring. From the results in this thesis we know that this normal form must not involve a prefix lexicalized target projection; whatever form it takes, however, that form would have a strong theoretical claim to being a generalization of GNF to synchronous CFGs.

We additionally intend to empirically evaluate prefix lexicalized STAGs on tasks such as word alignment to see how they compare to other lexicalized (but not prefix lexicalized) grammars. Since existing lexicalizations for synchronous grammars (e.g. Zhang and Gildea 2005) are not actually lexicalizations in the formal sense, it is not clear how they will compare to prefix lexicalized grammars in their ability to capture structural information about natural languages. Such an evaluation will also help to establish whether prefix lexicalization is useful in tasks which do not make explicit use of the structural properties of the grammar.

The prefix lexicalized STAGs discussed in this thesis may also be of use in proving an unpublished conjecture due to Aravind Joshi. This conjecture speculates that multicomponent STAGs (MC-STAGs) with prefix lexicalized target sides are less powerful than general MC-STAGs, in that they have smaller generative capacity and are therefore easier to parse. MC-STAGs are STAGs where the source and target side of a tree pair may comprise a set of trees rather than a single tree; during a derivation, all of the trees in such a set must compose at the same time as one another. Proving this conjecture would first require formalising prefix lexicalization for MC-STAGs: a plausible definition could impose an order on elements in the multicomponent set, and require that the first tree in this ordering be prefix lexicalized according to the definition used elsewhere in this work. A proof of the conjecture could then proceed by showing a mapping between MC-STAGs in this format and prefix lexicalized STAGs of the kind discussed in this thesis. Since the grammars in this thesis are known to be equivalent to SCFGs, such a mapping would imply that prefix lexicalized MC-STAGs are also no more powerful than SCFGs.

Proving this conjecture would have important ramifications for the field of linguistics, where MC-STAG is used to model the syntax and semantics of human language. Parsing MC-TAG is known to be NP-complete even in the non-synchronous case (Nesson, 2009). Additionally, many of the linguistic theories which use MC-STAG contain tree sets where some tree has a lexical item in its left corner (typically this is a λ-term from some expression in the lambda calculus). However prefix lexicalization is defined for MC-STAG, these tree sets are likely to satisfy the definition. Thus proving that MC-STAG is more computationally tractable when trees are prefix lexicalized would add to the psychological plausibility of these linguistic analyses, e.g. Han and Hedberg 2008; Storoshenko 2017; Han and Sarkar 2017 among many others.

Bibliography

Alfred V. Aho and Jeffrey D. Ullman. Syntax directed translations and the pushdown assembler. Journal of Computer and System Sciences, 3(1):37–56, 1969. ISSN 0022-0000.

Jean-Michel Autebert, Jean Berstel, and Luc Boasson. Context-free languages and pushdown automata. In Grzegorz Rozenberg and Arto Salomaa, editors, Handbook of Formal Languages, Vol. 1, pages 111–174. Springer-Verlag New York, Inc., New York, NY, USA, 1997. ISBN 3-540-60420-0. URL http://dl.acm.org/citation.cfm?id=267846.267849.

Yehoshua Bar-Hillel, M. Perles, and Eliahu Shamir. On formal properties of simple phrase structure grammars. Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung, 14:143–172, 1961. Reprinted in Y. Bar-Hillel. (1964). Language and Information: Selected Essays on their Theory and Application, Addison-Wesley 1964, 116–150.

Norbert Blum and Robert Koch. Greibach normal form transformation revisited. Information and Computation, 150(1):112–118, 1999. doi: 10.1006/inco.1998.2772. URL https://doi.org/10.1006/inco.1998.2772.

Zhiyi Chi. Statistical properties of probabilistic context-free grammars. Computational Linguistics, 25, 1999.

Noam Chomsky and Marcel-Paul Schützenberger. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg, editors, Computer Programming and Formal Systems, volume 35 of Studies in Logic and the Foundations of Mathematics, pages 118–161. Elsevier, 1963.

Søren Christensen, Hans Hüttel, and Colin Stirling. Bisimulation equivalence is decidable for all context-free processes. Information and Computation, 121(2):143–148, 1995. ISSN 0890-5401.

Pierluigi Crescenzi, Daniel Gildea, Andrea Marino, Gianluca Rossi, and Giorgio Satta. Synchronous context-free grammars and optimal linear parsing strategies. Journal of Computer and System Sciences, 81(7):1333–1356, 2015. ISSN 0022-0000. doi: 10.1016/j.jcss.2015.04.003. URL http://www.sciencedirect.com/science/article/pii/S0022000015000409.

Marc Dymetman. A generalized Greibach normal form for definite clause grammars. In Proceedings of the 14th Conference on Computational Linguistics - Volume 1, COLING '92, pages 366–372, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics. doi: 10.3115/992066.992126. URL https://doi.org/10.3115/992066.992126.

Joost Engelfriet, Andreas Maletti, and Sebastian Maneth. Multiple context-free tree grammars: Lexicalization and characterization. arXiv preprint, 2017. URL http://arxiv.org/abs/1707.03457.

Zoltán Ésik and Hans Leiß. Algebraically complete semirings and Greibach normal form. Annals of Pure and Applied Logic, 133(1):173–203, 2005. ISSN 0168-0072. doi: https://doi.org/10.1016/j.apal.2004.10.008. URL http://www.sciencedirect.com/science/article/pii/S0168007204001332. Festschrift on the occasion of Helmut Schwichtenberg's 60th birthday.

Henning Fernau and Ralf Stiebe. Sequential grammars and automata with valences. Theoretical Computer Science, 276(1):377–405, 2002. ISSN 0304-3975. doi: 10.1016/S0304-3975(01)00282-1. URL http://www.sciencedirect.com/science/article/pii/S0304397501002821.

James N. Gray and Michael A. Harrison. On the covering and reduction problems for context-free grammars. Journal of the ACM, 19(4):675–698, October 1972. ISSN 0004-5411. doi: 10.1145/321724.321732. URL http://doi.acm.org/10.1145/321724.321732.

Sheila A. Greibach. A new normal-form theorem for context-free phrase structure grammars. Journal of the ACM, 12(1):42–52, 1965. doi: 10.1145/321250.321254. URL http://doi.acm.org/10.1145/321250.321254.

Sheila A. Greibach and John E. Hopcroft. Scattered context grammars. Journal of Computer and System Sciences, 3(3):233–247, 1969. doi: 10.1016/S0022-0000(69)80015-2. URL https://doi.org/10.1016/S0022-0000(69)80015-2.

Chung-Hye Han and Nancy Hedberg. Syntax and semantics of it-clefts: A tree adjoining grammar analysis. Journal of Semantics, 25(4):345–380, 2008. doi: 10.1093/jos/ffn007. URL http://dx.doi.org/10.1093/jos/ffn007.

Chung-hye Han and Anoop Sarkar. Coordination in TAG without the conjoin operation. In Proceedings of the 13th International Workshop on Tree Adjoining Grammars and Related Formalisms, pages 43–52. Association for Computational Linguistics, 2017. URL http://aclweb.org/anthology/W17-6205.

Hendrik Jan Hoogeboom. Context-free valence grammars - revisited. In Werner Kuich, Grzegorz Rozenberg, and Arto Salomaa, editors, Developments in Language Theory: 5th International Conference, DLT 2001 Wien, Austria, July 16–21, 2001 Revised Papers, pages 293–303. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. ISBN 978-3-540-46011-4. doi: 10.1007/3-540-46011-X_25. URL https://doi.org/10.1007/3-540-46011-X_25.

Mark Johnson and Eugene Charniak. A TAG-based noisy-channel model of speech repairs. In Donia Scott, Walter Daelemans, and Marilyn A. Walker, editors, Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics, 21-26 July, 2004, Barcelona, Spain, pages 33–39. ACL, 2004. URL http://aclweb.org/anthology/P/P04/P04-1005.pdf.

Aravind Joshi and Yves Schabes. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, Handbook of Formal Languages, Vol. 3: Beyond Words, chapter 2, pages 69–124. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

Aravind Joshi, Leon Levy, and Masako Takahashi. Tree adjunct grammars. Journal of Computer and System Sciences, 10(1):136–163, 1975. doi: 10.1016/S0022-0000(75)80019-5. URL https://doi.org/10.1016/S0022-0000(75)80019-5.

Marco Kuhlmann and Giorgio Satta. Tree-adjoining grammars are not closed under strong lexicalization. Computational Linguistics, 38(3):617–629, 2012. doi: 10.1162/COLI_a_00090. URL https://doi.org/10.1162/COLI_a_00090.

Philip M. Lewis and Richard E. Stearns. Syntax-directed transduction. Journal of the ACM, 15(3):465–488, July 1968. ISSN 0004-5411. doi: 10.1145/321466.321477. URL http://doi.acm.org/10.1145/321466.321477.

Andreas Maletti and Joost Engelfriet. Strong lexicalization of tree adjoining grammars. In The 50th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, July 8-14, 2012, Jeju Island, Korea - Volume 1: Long Papers, pages 506–515. The Association for Computational Linguistics, 2012. ISBN 978-1-937284-24-4. URL http://www.aclweb.org/anthology/P12-1053.

Mark-Jan Nederhof and Giorgio Satta. Computing partition functions of PCFGs. Research on Language and Computation, 6(2):139–162, Oct 2008. ISSN 1572-8706. doi: 10.1007/s11168-008-9052-8. URL https://doi.org/10.1007/s11168-008-9052-8.

Rebecca Nesson. Synchronous and Multicomponent Tree-Adjoining Grammars: Complexity, Algorithms and Linguistic Applications. Dissertation, Harvard University, 2009. URL https://pdfs.semanticscholar.org/754b/6acaf2660748967d1937a25222538207aabc.pdf.

James Rogers. Capturing CFLs with tree adjoining grammars. In Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics, ACL '94, pages 155–162, Stroudsburg, PA, USA, 1994. Association for Computational Linguistics. doi: 10.3115/981732.981754. URL http://dx.doi.org/10.3115/981732.981754.

Daniel J. Rosenkrantz. Matrix equations and normal forms for context-free grammars. Journal of the Association for Computing Machinery, 14(3):501–507, 1967.

Anoop Sarkar. Practical experiments in parsing using tree adjoining grammars. In Proceedings of the 5th International Workshop on Tree Adjoining Grammars and Related Formalisms, pages 193–198, 2000.

Yves Schabes and Richard C. Waters. Lexicalized context-free grammars. In L. Schubert, editor, 31st Annual Meeting of the Association for Computational Linguistics, 22-26 June 1993, Ohio State University, Columbus, Ohio, USA, Proceedings, pages 121–129. ACL, 1993. URL http://aclweb.org/anthology/P/P93/P93-1017.pdf.

Yves Schabes and Richard C. Waters. Tree insertion grammar: Cubic-time, parsable formalism that lexicalizes context-free grammar without changing the trees produced. Computational Linguistics, 21(4):479–513, 1995.

Eliahu Shamir. A representation theorem for algebraic and context-free power series in noncommuting variables. Information and Control, 11(1/2):239–254, 1967. doi: 10.1016/S0019-9958(67)90529-3. URL https://doi.org/10.1016/S0019-9958(67)90529-3.

Stuart M. Shieber. Restricting the weak-generative capacity of synchronous tree-adjoining grammars. Computational Intelligence, 10:371–385, 1994. doi: 10.1111/j.1467-8640.1994.tb00003.x. URL http://dx.doi.org/10.1111/j.1467-8640.1994.tb00003.x.

M. Siahbani, R. M. Seraj, B. Sankaran, and A. Sarkar. Incremental translation using hierarchichal phrase-based translation system. In 2014 IEEE Spoken Language Technology Workshop (SLT), pages 71–76, Dec 2014. doi: 10.1109/SLT.2014.7078552.

Maryam Siahbani and Anoop Sarkar. Two improvements to left-to-right decoding for hierarchical phrase-based machine translation. In A. Moschitti, B. Pang, and W. Daelemans, editors, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL, pages 221–226. ACL, 2014a. ISBN 978-1-937284-96-1. URL http://aclweb.org/anthology/D/D14/D14-1028.pdf.

Maryam Siahbani and Anoop Sarkar. Expressive hierarchical rule extraction for left-to-right translation. In Proceedings of the 11th Biennial Conference of the Association for Machine Translation in the Americas (AMTA-2014), Vancouver, Canada, 2014b.

Maryam Siahbani, Baskaran Sankaran, and Anoop Sarkar. Efficient left-to-right hierarchical phrase-based translation with improved reordering. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pages 1089–1099. Association for Computational Linguistics, 2013. URL http://www.aclweb.org/anthology/D13-1110.

Noah A. Smith and Mark Johnson. Weighted and probabilistic context-free grammars are equally expressive. Computational Linguistics, 33(4):477–491, December 2007. ISSN 0891-2017. doi: 10.1162/coli.2007.33.4.477. URL http://dx.doi.org/10.1162/coli.2007.33.4.477.

Donald F. Stanat. A homomorphism theorem for weighted context-free grammars. Journal of Computer and System Sciences, 6(3):217–232, 1972. ISSN 0022-0000. doi: 10.1016/S0022-0000(72)80003-5. URL http://www.sciencedirect.com/science/article/pii/S0022000072800035.

Dennis Ryan Storoshenko. Scope, time, and predicate restriction in Blackfoot using MC-STAG. In Proceedings of the 13th International Workshop on Tree Adjoining Grammars and Related Formalisms, pages 53–60. Association for Computational Linguistics, 2017. URL http://aclweb.org/anthology/W17-6206.

Ben Swanson, Elif Yamangil, Eugene Charniak, and Stuart M. Shieber. A context free TAG variant. In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 1: Long Papers, pages 302–310. The Association for Computational Linguistics, 2013. ISBN 978-1-937284-50-3. URL http://aclweb.org/anthology/P/P13/P13-1030.pdf.

Taro Watanabe, Hajime Tsukada, and Hideki Isozaki. Left-to-right target generation for hierarchical phrase-based translation. In N. Calzolari, C. Cardie, and P. Isabelle, editors, ACL 2006, 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Sydney, Australia, 17-21 July 2006. The Association for Computational Linguistics, 2006. URL http://aclweb.org/anthology/P06-1098.

Dekai Wu. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. Computational Linguistics, 23(3):377–403, 1997.

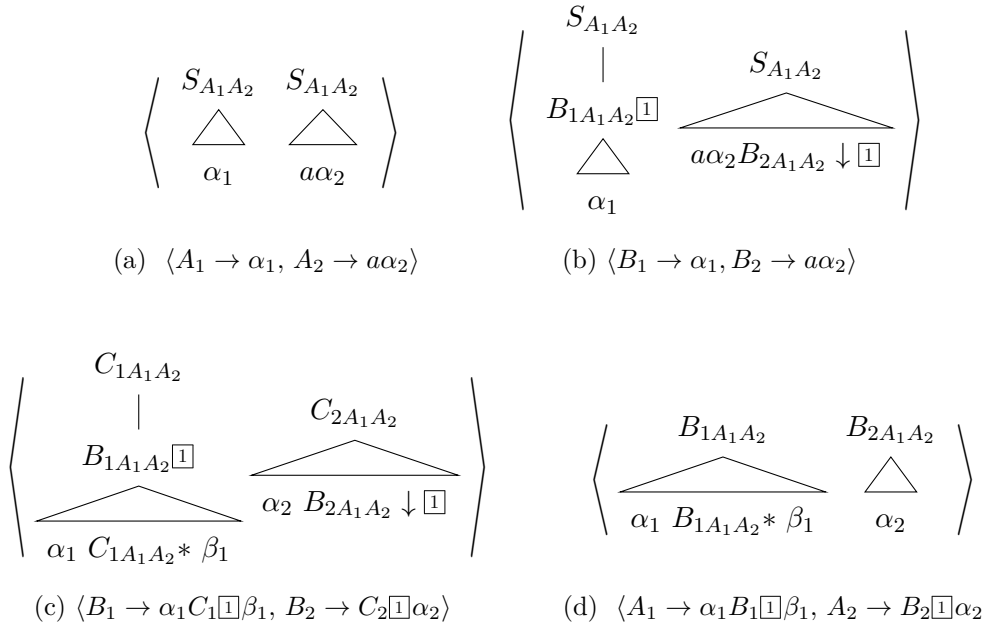
Hao Zhang and Daniel Gildea. Stochastic lexicalized inversion transduction grammar for alignment. In K. Knight, H. T. Ng, and K. Oflazer, editors, ACL 2005, 43rd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, 25-30 June 2005, University of Michigan, USA, pages 475–482. The Association for Computational Linguistics, 2005. URL http://aclweb.org/anthology/P/P05/P05-1059.pdf.

Simon Zwarts, Mark Johnson, and Robert Dale. Detecting speech repairs incrementally using a noisy channel approach. In Chu-Ren Huang and Dan Jurafsky, editors, COLING 2010, 23rd International Conference on Computational Linguistics, Proceedings of the Conference, 23-27 August 2010, Beijing, China, pages 1371–1378. Tsinghua University Press, 2010. URL http://aclweb.org/anthology/C10-1154.

Appendix A

Proof of Lemma 1

This appendix contains the complete proof for Lemma 1. We repeat Figure 3.2 from Chapter 3 as figure A.1 for ease of reference.



(a) $\langle A_1 \to \alpha_1,\ A_2 \to a\alpha_2\rangle$ (b) $\langle B_1 \to \alpha_1, B_2 \to a\alpha_2\rangle$

(c) $\langle B_1 \to \alpha_1 C_1\boxed{1}\beta_1,\ B_2 \to C_2\boxed{1}\alpha_2\rangle$ (d) $\langle A_1 \to \alpha_1 B_1\boxed{1}\beta_1,\ A_2 \to B_2\boxed{1}\alpha_2\rangle$

Figure A.1: Tree-pairs in $G_{A_1A_2}$ and the rules in G from which they derive.

Lemma 3. *$G_{A_1A_2}$ generates the language $L_{A_1A_2} = \{\langle u, v\rangle | \langle A_1\boxed{1}, A_2\boxed{1}\rangle \Rightarrow^*_{TTLD} \langle u, v\rangle\}$.*

Proof. We prove this lemma by induction over derivations of increasing length. We show first that every TTLD starting from $\langle A_1\boxed{1}, A_2\boxed{1}\rangle$ corresponds to a unique derivation in $G_{A_1A_2}$; we then show the other direction, that each derivation in $G_{A_1A_2}$ corresponds to a TTLD starting from $\langle A_1\boxed{1}, A_2\boxed{1}\rangle$.

A.1 TTLD to STAG

We first consider the direction from TTLDs in G to derivations in $G_{A_1A_2}$. For the sake of brevity, the rest of this section uses TTLD as shorthand for "TTLD starting from $\langle A_1\boxed{1}, A_2\boxed{1}\rangle$". We show that the last n steps of every TTLD correspond to some derivation over n trees from $G_{A_1A_2}$. We show as well that whenever the derivation in $G_{A_1A_2}$ is complete (there are no open substitution sites left) it generates the same sentential form as the TTLD.

Base Cases As a base case, consider a TTLD of length 1, as in (A.1):

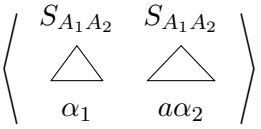
$$\langle A_1\boxed{1}, A_2\boxed{1}\rangle \Rightarrow \langle \alpha_1, a\alpha_2\rangle \tag{A.1}$$

where $a \in \Sigma$, and $\alpha_i \in (N \cup \Sigma)^*$. Such a derivation involves the application of one rule which must be of the form in (A.2):

$$\langle A_1 \rightarrow \alpha_1, A_2 \rightarrow a\alpha_2\rangle \tag{A.2}$$

By construction, we know that if such a rule exists in G, then $G_{A_1A_2}$ must contain a corresponding tree pair of the shape depicted in Figure 3.2(a). This implies that the following is a valid derived tree in $G_{A_1A_2}$:

(A.3)



This derived tree produces the same string pair as the TTLD in (3.4). Thus we see that for every single-step TTLD in G there exists a (unique) derivation in $G_{A_1A_2}$ which produces the same sentential form.

As a second base case, consider a TTLD of length > 1. This will be a derivation of the form in (A.4)

$$\langle A_1\boxed{1}, A_2\boxed{1}\rangle \Rightarrow^*_{TTLD} \langle uB_1\boxed{1}v, B_2\boxed{1}w\rangle \Rightarrow \langle u\alpha_1 v, a\alpha_2 w\rangle \tag{A.4}$$

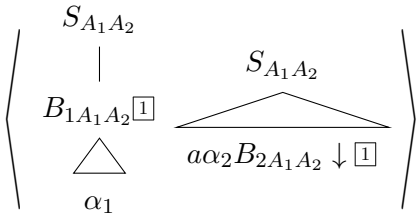
where $B_1, B_2 \in N \setminus \{S\}$, $a \in \Sigma$, and $u, v, w, \alpha_i \in (N \cup \Sigma)^*$. Now the last step of this TTLD must involve the application of some rule of the form in (A.5)

$$\langle B_1 \rightarrow \alpha_1, B_2 \rightarrow a\alpha_2\rangle \tag{A.5}$$

By construction, we know that if such a rule exists in G, then $G_{A_1A_2}$ must contain a corresponding tree pair of the shape in Figure 3.2(b). This implies that the following is a

valid derived tree pair in $G_{A_1A_2}$:

(A.6)



This tree pair does not constitute a complete derivation, however, as there remains an open substitution site in the target-side tree. This derivation produces the sentential form $\langle \alpha_1, a\alpha_2 B_{2A_1A_2} \rangle$, which is the same form produced by the last step of the TTLD in question, up to the addition of a $B_{2A_1A_2}$ in the target string. Finally, note that this derivation contains an open $B_{1A_1A_2}$ adjunction site on the source side linked to the $B_{2A_1A_2}$ substitution site on the target side.

Taken together, these base cases show the following:

- Every TTLD of length 1 has a corresponding derivation in $G_{A_1A_2}$ which produces the same sentential form as that TTLD.
- For every TTLD of length > 1, the last step of that TTLD corresponds to some single-tree derivation in $G_{A_1A_2}$. This correspondence satisfies the following:
  - the last step of the TTLD produces the same sentential form as the derivation in $G_{A_1A_2}$, up to the addition of some nonterminal in the target string;
  - if the last step of the TTLD involves overwriting some pair of nonterminals $\langle B_1\boxed{1}, B_2\boxed{1} \rangle$, then the derivation in $G_{A_1A_2}$ contains a $B_{1A_1A_2}$ adjunction site in the source tree linked to a $B_{2A_1A_2}$ substitution site in the target tree.

Inductive Step Assume that the following inductive hypotheses are true for some n:

> For every TTLD of length $> n$, the last n steps of that TTLD correspond to some derivation in $G_{A_1A_2}$ over n trees. This correspondence satisfies the following:
>
> - the last n steps of the TTLD produce the same sentential form as the derivation in $G_{A_1A_2}$, up to the addition of some nonterminal in the target string;
> - if the nth-from-last step of the TTLD involves overwriting some pair of nonterminals $\langle B_1\boxed{1}, B_2\boxed{1} \rangle$, then the derivation in $G_{A_1A_2}$ contains a $B_{1A_1A_2}$ adjunction site in the source tree linked to a $B_{2A_1A_2}$ substitution site in the target tree.
>
> Also, for every TTLD of length exactly n, that TTLD corresponds to some derivation in $G_{A_1A_2}$ over n trees. This correspondence satisfies the following:
>
> - the TTLD produces the same sentential form as the derivation in $G_{A_1A_2}$.

- the derivation in $G_{A_1A_2}$ contains no open adjunction or substitution sites.

We now prove that if these hypotheses hold for some n, then they must also hold for $n+1$. There are two cases to consider: either a TTLD contains more than $n+1$ steps, or it contains exactly $n+1$ steps.

First Case: TTLD of length $> n+1$ Consider the last $n+1$ steps of such a TTLD, as shown in (A.7)

$$\langle B_1\boxed{1}, B_2\boxed{1}\rangle \Rightarrow \langle \alpha_1 C_1\boxed{1}\beta_1, C_2\boxed{1}\alpha_2\rangle \Rightarrow^*_{TTLD} \langle \alpha_1\gamma_1\beta_1, a\gamma_2\alpha_2\rangle \tag{A.7}$$

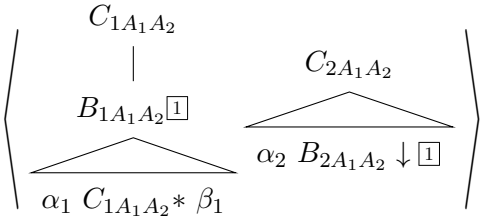
where $B_1, B_2, C_1, C_2 \in N \setminus \{S\}$, $a \in \Sigma$, and $\alpha_i, \beta_i, \gamma_i \in (N \cup \Sigma)^*$. By the first inductive hypothesis, the last n of these steps correspond to some derivation over n trees in $G_{A_1A_2}$. Since the first of these n steps must involve rewriting the C_2 which is at the left edge of the target string, the inductive hypothesis implies that the derivation in $G_{A_1A_2}$ contains a $C_{2A_1A_2}$ substitution site linked to a $C_{1A_1A_2}$ adjunction site. Furthermore, by the inductive hypothesis this derivation produces the same sentential form as the last n steps of the TTLD, up to the addition of a $C_{2A_1A_2}$ at the edge of the target string.

Now, from (A.7) we also see that the step $n+1$ operations before the end of the TTLD involves a rule of the form

$$\langle B_1 \rightarrow \alpha_1 C_1\boxed{1}\beta_1, B_2 \rightarrow C_2\boxed{1}\alpha_2\rangle \tag{A.8}$$

for some $B_1, B_2, C_1, C_2 \in N \setminus \{S\}$. By construction, the existence of this rule in G implies that $G_{A_1A_2}$ contains a tree pair of the shape in Figure 3.2(c), repeated here as (A.9)

(A.9)



This tree pair can be added to the n-tree derivation which the inductive hypothesis tells us must exist: the source tree can adjoin to the open $C_{1A_1A_2}$ adjunction site, and the target tree can substitute into the $C_{2A_1A_2}$ substitution site.

The result will be a new $n+1$ tree derivation which satisfies the following:

- it produces the same sentential form as the last $n+1$ steps of the TTLD. This can be verified by observing that all adjunction sites in $G_{A_1A_2}$ are near the root of the tree, so that when the new source tree adjoins it must necessarily wrap α_1 and β_1 to either side of the existing source string to produce the required form; on the target side, the new tree will overwrite the $C_{2A_1A_2}$ node at the right edge of the string so that α_2 will also be in the correct position.

- it contains an open $B_{1A_1A_2}$ adjunction site on the source side linked to a $B_{2A_1A_2}$ substitution site on the target side, as can be seen by inspection of (A.9)

Therefore we see that the first inductive hypothesis will also hold for a derivation of length $n+1$ given that it holds for a derivation of length n.

Second Case: TTLD of length $n+1$ Consider a completed TTLD of length $n+1$, as shown in (A.10)

$$\langle A_1\boxed{1}, A_2\boxed{1}\rangle \Rightarrow \langle \alpha_1 B_1\boxed{1}\beta_1, B_2\boxed{1}\alpha_2\rangle \Rightarrow^*_{TTLD} \langle \alpha_1\gamma_1\beta_1, a\gamma_2\alpha_2\rangle \tag{A.10}$$

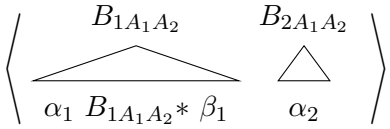
where $B_1, B_2 \in N \setminus \{S\}$, $a \in \Sigma$, and $\alpha_i, \beta_i, \gamma_i \in (N \cup \Sigma)^*$. By the first inductive hypothesis, the last n steps of this TTLD correspond to some derivation over n trees in $G_{A_1A_2}$. Since the first of these n steps must involve rewriting the B_2 which is at the left edge of the target string, the derivation in $G_{A_1A_2}$ must contain a $B_{2A_1A_2}$ substitution site linked to a $B_{1A_1A_2}$ adjunction site. Furthermore, this derivation must produce the same string as the last n steps of the TTLD, up to the addition of $B_{2A_1A_2}$ at the right edge of the target string.

Now, from (A.10) we also see that the first step of the derivation involves a rule of the form

$$\langle A_1 \rightarrow \alpha_1 B_1\boxed{1}\beta_1, A_2 \rightarrow B_2\boxed{1}\alpha_2\rangle \tag{A.11}$$

By construction, the existence of this rule in G implies that $G_{A_1A_2}$ contains a tree pair of the shape in Figure 3.2(d), repeated here as (A.12)

(A.12)



This tree pair can be added to the n-tree derivation which the inductive hypothesis tells us must exist: the source tree can adjoin to the open $B_{1A_1A_2}$ adjunction site, and the target tree can substitute into the $B_{2A_1A_2}$ substitution site.

The result will be a new $n+1$ tree derivation which satisfies the following:

- it produces the same sentential form as the entire $n+1$ step TTLD. This can be verified by observing that all adjunction sites in $G_{A_1A_2}$ are near the root of the tree, so that when the new source tree adjoins it will wrap α_1 and β_1 to either side of the existing source string to produce the required form; on the target side, the new tree will overwrite the $B_{2A_1A_2}$ node at the right edge of the string so that α_2 will also be in the correct position.
- it is a completed derivation, as there are no open adjunction or substitution sites.

Therefore it follows that the second inductive hypothesis also holds for $n+1$ given that the first hypothesis holds for n.

Conclusion Taken together, the preceding two cases show that there is a derivation in $G_{A_1A_2}$ corresponding to every TTLD starting from $\langle A_1\boxed{1}, A_2\boxed{1}\rangle$. To obtain a one-to-one correspondence, we now prove the other direction, that for every derivation in $G_{A_1A_2}$ there exists a corresponding TTLD in G.

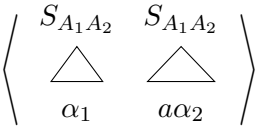
A.2 STAG to TTLD

We now show that the first n steps of every derivation in $G_{A_1A_2}$ correspond to the last n steps of a TTLD in G, and every complete derivation in $G_{A_1A_2}$ corresponds to a TTLD starting from $\langle A_1\boxed{1}, A_2\boxed{1}\rangle$.

Preliminaries In TAG, derivations are generally assumed to be unordered, and all operations are taken to occur at once. In the case of a grammar like $G_{A_1A_2}$, however, we may talk about the "first" and "last" operations, because every tree has rank at most 1. Concretely, we shall say that the first tree pair in a derivation is the one rooted in the start symbol $S_{A_1A_2}$. Then the second tree pair in that derivation is the one which substitutes or adjoins to the first; the third tree pair substitutes or adjoins to the second; and so on.

Base Cases As a base case, consider a derivation in $G_{A_1A_2}$ comprising a single tree pair of the shape given in Figure 3.2(a), repeated here:

(A.13)



where $a \in \Sigma$, and $\alpha_i \in (N \cup \Sigma)^*$. By construction, we know that this tree pair must have been added to $G_{A_1A_2}$ on the basis of some rule in G. In particular, there must be a corresponding rule in G of the shape in (A.14)

(A.14) $$\langle A_1 \to \alpha_1, A_2 \to a\alpha_2\rangle$$

where $a \in \Sigma$, and $\alpha_i \in (N \cup \Sigma)^*$.

Using (A.14), we may construct a TTLD of length 1, shown in (A.15):

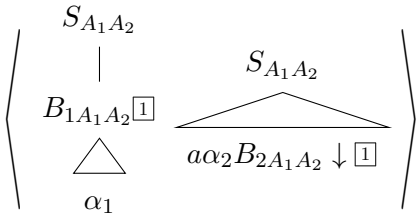
(A.15) $$\langle A_1\boxed{1}, A_2\boxed{1}\rangle \Rightarrow \langle \alpha_1, a\alpha_2\rangle$$

This is a completed TTLD which generates the same string pair as the derivation in $G_{A_1A_2}$ shown in (A.13). Thus we see that for every completed single-tree derivation in $G_{A_1A_2}$, there exists a corresponding TTLD in G which produces the same string.

As a second base case, consider a derivation in $G_{A_1A_2}$ comprising more than one tree pair. This derivation must start with some tree pair rooted in $S_{A_1A_2}$; furthermore, since it includes

more than one tree pair in total, it cannot start with a pair of the shape in 3.2(a), because such a pair has no open substitution or adjunction sites. The only remaining possibility is for the derivation to start with a tree pair of the shape in 3.2(b), repeated below:

(A.16)



where $B_1, B_2 \in N \setminus \{S\}$, $a \in \Sigma$, and $\alpha_i \in (N \cup \Sigma)^*$. By construction, we know that this tree pair must have been added to $G_{A_1A_2}$ on the basis of some rule in G. In particular, there must be a corresponding rule in G of the shape in (A.17)

(A.17) $$\langle B_1 \to \alpha_1, B_2 \to a\alpha_2 \rangle$$

where $B_1, B_2 \in N \setminus \{S\}$, $a \in \Sigma$, and $\alpha_i \in (N \cup \Sigma)^*$.

Using (A.17), we may construct the derivation in (A.18):

(A.18) $$\langle B_1\boxed{1}, B_2\boxed{1} \rangle \Rightarrow \langle \alpha_1, a\alpha_2 \rangle$$

This is a valid TTLD; furthermore this derivation produces the string pair $\langle \alpha_1, a\alpha_2 \rangle$, which is the same pair produced by the first tree in the derivation in $G_{A_1A_2}$, up to the removal of $B_{2A_1A_2}$ from the right edge of the target string. Note that (A.18) starts by rewriting the pair $\langle B_1\boxed{1}, B_2\boxed{1} \rangle$, and that (A.16) correspondingly contains a $B_{1A_1A_2}$ adjunction site linked to a $B_{2A_1A_2}$ substitution site.

Taken together, the two base cases show the following:

- Every completed, single-tree-pair derivation in $G_{A_1A_2}$ has a corresponding TTLD in G which produces the same sentential form as that derivation.
- For every derivation in $G_{A_1A_2}$ comprising more than one tree pair, the first tree pair in that derivation corresponds to the end of some TTLD in G. This correspondence satisfies the following:
  - the last step of the TTLD produces the same sentential form as the first tree pair of the derivation in $G_{A_1A_2}$, up to the removal of some nonterminal from the target string;
  - if the first tree pair in the derivation in $G_{A_1A_2}$ contains a $B_{1A_1A_2}$ adjunction site in the source tree linked to a $B_{2A_1A_2}$ substitution site in the target tree, then the last step of the TTLD involves overwriting the pair of nonterminals $\langle B_1\boxed{1}, B_2\boxed{1} \rangle$.

Inductive Step Assume that the following inductive hypotheses are true for some n:

For every derivation in $G_{A_1A_2}$ comprising $> n$ tree pairs, the first n tree pairs in that derivation correspond to some TTLD in G involving n rule applications. This correspondence satisfies the following:

- the first n tree pairs produce the same sentential form as the TTLD, up to the removal of some nonterminal from the right edge of the target string;
- if the nth tree pair of the derivation in $G_{A_1A_2}$ contains a $B_{1A_1A_2}$ adjunction site in the source tree linked to a $B_{2A_1A_2}$ substitution site in the target tree, then the first step of the TTLD involves overwriting the pair of nonterminals $\langle B_1\boxed{1}, B_2\boxed{1}\rangle$.

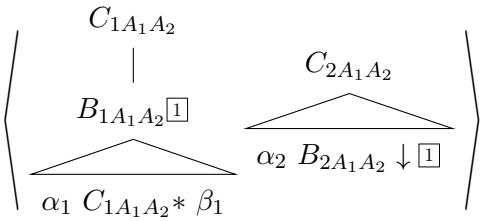
Also, for every derivation in $G_{A_1A_2}$ of length exactly n, that derivation corresponds to some TTLD involving n rule applications. This correspondence satisfies the following:

- the TTLD produces the same sentential form as the derivation in $G_{A_1A_2}$.
- the TTLD starts from $\langle A_1\boxed{1}, A_2\boxed{1}\rangle$.

We now prove that if these hypotheses hold for some n, then they must also hold for $n+1$. There are two cases to consider: either a derivation in $G_{A_1A_2}$ involves more than $n+1$ tree pairs, or it involves exactly $n+1$ pairs.

First Case: $> n+1$ tree pairs Consider the $n+1$th tree pair in such a derivation. This must be of the shape in Figure 3.2(c), repeated below as (A.19). This is because this is the only kind of tree pair in $G_{A_1A_2}$ which both (i) contains open substitution/adjunction sites to perpetuate the derivation (since by assumption it is longer than $n+1$ operations) and (ii) is not rooted in $S_{A_1A_2}$, and is therefore able to appear in the middle of a derivation.

(A.19)



Since the $n+1$th pair must compose with the nth pair, the nth pair must contain an open adjunction site labeled $C_{1A_1A_2}$ linked to a substitution site labeled $C_{2A_1A_2}$, where $C_{1A_1A_2}$ and $C_{2A_1A_2}$ are the nonterminals at the root of the $n+1$th pair's source and target trees respectively.

Furthermore, by the first inductive hypothesis, the first n tree pairs in this derivation must correspond to some n-step TTLD in G. Since the nth pair has open $C_{1A_1A_2}$ and $C_{2A_1A_2}$ sites, we know by the same hypothesis that the corresponding TTLD starts from $\langle C_1\boxed{1}, C_2\boxed{1}\rangle$, as in (A.20):

$$\langle C_1\boxed{1}, C_2\boxed{1}\rangle \Rightarrow^*_{TTLD} \langle \gamma_1, a\gamma_2\rangle \tag{A.20}$$

Now, by construction we know that if $G_{A_1A_2}$ contains a tree pair of the shape in (A.19), then G must contain a production of the shape in (A.21):

(A.21) $$\langle B_1 \to \alpha_1 C_1 \boxed{1} \beta_1, B_2 \to C_2 \boxed{1} \alpha_2 \rangle$$

By applying the rule in (A.21), followed by the rest of the derivation in (A.20), we obtain a new $n+1$-step TTLD shown in (A.22):

(A.22) $$\langle B_1 \boxed{1}, B_2 \boxed{1} \rangle \Rightarrow \langle \alpha_1 C_1 \boxed{1} \beta_1, C_2 \boxed{1} \alpha_2 \rangle \Rightarrow^*_{TTLD} \langle \alpha_1 \gamma_1 \beta_1, a \gamma_2 \alpha_2 \rangle$$

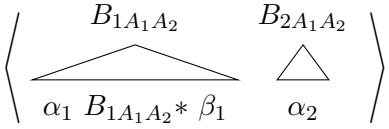
The new TTLD in (A.22) satisfies the following:

- it produces the same sentential form as the first $n+1$ tree pairs of the derivation in $G_{A_1A_2}$, up to the removal of a nonterminal from the right edge of the target string. This can be verified by observing that prepending the new production to the existing TTLD wraps α_1 and β_1 around the existing source string in the same way that adjoining the $n+1$th source tree wraps α_1 and β_1 around the rest of the tree; on the target side, α_2 is appended to the right edge in the same position that the $n+1$th target tree appends $\alpha_2 B_{2A_1A_2}$.
- it starts from the pair $\langle B_1 \boxed{1}, B_2 \boxed{1} \rangle$, where $B_{1A_1A_2}$ and $B_{2A_1A_2}$ are the labels on the adjunction and substitution sites in the $n+1$th tree pair.

Therefore we see that the first inductive hypothesis holds for derivations of length $n+1$ given that it holds for derivations of length n. In other words, we have so far proven that for every derivation in $G_{A_1A_2}$, every step up to the last step of the derivation corresponds to some TTLD in G. We now prove the final case, which shows that the last steps of the derivations also correspond.

Second Case: exactly $n+1$ tree pairs Consider a completed derivation in $G_{A_1A_2}$ containing $n+1$ tree pairs. The last tree pair must be of the shape in Figure 3.2(d), repeated below as (A.23), because this is the only tree pair which can compose with a derivation without introducing any new adjunction or substitution sites.

(A.23)



Since the $n+1$th tree pair must compose with the nth pair, the nth pair must contain an open adjunction site labeled $B_{1A_1A_2}$ linked to a substitution site labeled $B_{2A_1A_2}$, where $B_{1A_1A_2}$ and $B_{2A_1A_2}$ are the nonterminals at the root of the $n+1$th pair's source and target trees respectively.

Furthermore, by the first inductive hypothesis, the first n tree pairs in this derivation must correspond to some n-step TTLD in G. Since the nth pair has open $B_{1A_1A_2}$ and $B_{2A_1A_2}$ sites,

we know by the same hypothesis that the corresponding TTLD starts from $\langle B_1\boxed{1}, B_2\boxed{1}\rangle$, as in (A.24):

$$\langle B_1\boxed{1}, B_2\boxed{1}\rangle \Rightarrow^*_{TTLD} \langle \gamma_1, a\gamma_2\rangle \tag{A.24}$$

Now, by construction we know that if the $n+1$th tree pair is of the shape in (A.23), then G must contain a production of the shape in (A.25):

$$\langle A_1 \rightarrow \alpha_1 B_1\boxed{1}\beta_1, A_2 \rightarrow B_2\boxed{1}\alpha_2\rangle \tag{A.25}$$

By applying the rule in (A.25), followed by the rest of the derivation in (A.24), we obtain a new $n+1$-step TTLD shown in (A.26):

$$\langle A_1\boxed{1}, A_2\boxed{1}\rangle \Rightarrow \langle \alpha_1 B_1\boxed{1}\beta_1, B_2\boxed{1}\alpha_2\rangle \Rightarrow^*_{TTLD} \langle \alpha_1\gamma_1\beta_1, a\gamma_2\alpha_2\rangle \tag{A.26}$$

The new TTLD in (A.26) satisfies the following:

- it produces the same sentential form as the first $n+1$ tree pairs of the derivation in $G_{A_1A_2}$. This can be verified by observing that prepending the new production to the existing TTLD wraps α_1 and β_1 around the existing source string in the same way that adjoining the $n+1$th source tree wraps α_1 and β_1 around the rest of the tree; on the target side, α_2 is appended to the right edge in the same position that the $n+1$th target tree appends α_2.
- it starts from the pair $\langle A_1\boxed{1}, A_2\boxed{1}\rangle$.

Therefore we see that the second inductive hypothesis holds for derivations of length $n+1$ given that the first hypothesis holds for derivations of length n.

Conclusion Taken together, the preceding two cases show that there is a TTLD in G corresponding to every derivation in $G_{A_1A_2}$. Furthermore every completed derivation in $G_{A_1A_2}$ corresponds to a TTLD which starts from the pair $\langle A_1\boxed{1}, A_2\boxed{1}\rangle$.

Combining the results from both of the preceding sections, we see that there is a one-to-one correspondence between completed derivations in $G_{A_1A_2}$ and TTLDs in G which start from $\langle A_1\boxed{1}, A_2\boxed{1}\rangle$. By extension, we have shown that $G_{A_1A_2}$ generates precisely the language $L_{A_1A_2} = \{\langle u, v\rangle | \langle A_1\boxed{1}, A_2\boxed{1}\rangle \Rightarrow^*_{TTLD} \langle u, v\rangle\}$.

□

Appendix B

LR Decoding with STAG

Algorithm 1 gives the pseudocode for LR decoding, adapted from Siahbani et al. (2014) which is a refinement of Watanabe et al. (2006). The section in red highlights the changes necessary for this algorithm to accommodate PL-RSTAG instead of PL-SCFG. Note how nearly the entire algorithm is unaffected by the change in grammar formalism: the only major change is to use an Earley-style TAG parser (cf. Joshi and Schabes 1997) on the source grammar rather than a CFG parser.

This algorithm has been implemented by Siahbani et al., and an Earley-style TAG parser has been implemented by Sarkar (2000). Thus, to empirically evaluate the transformation described in this work, it should suffice to adapt the parser to ensure cubic-time performance (according to a method already outlined in Rogers 1994) and then to combine these existing implementations.

Algorithm 1 LR-Hiero Decoding for PL-RSTAG

Input sentence: $\mathbf{f} = f_0 f_1 ... f_n$
$\mathcal{F} = \text{FutureCost}(\mathbf{f})$ ▷ (Precompute future cost for spans)
$S_0 = \{\}$ ▷ (Create empty initial stack)
$h_0 = (\langle s \rangle, [[0, n]], \emptyset, \mathcal{F}_{[0,n]})$ ▷ (Create initial hypothesis: $\langle s \rangle$ is the initial translation prefix containing a beginning-of-string marker s; $[[0, n]]$ is a list of uncovered spans in the source sentence; $\emptyset$ is the set of covered source words; $\mathcal{F}_{[0,n]}$ is the hypothesis cost, initialized to a precomputed future cost estimate.)
Add h_0 to S_0 ▷ (Push initial hypothesis onto first stack)
for $i = 1, ..., m$ **do**
    cubeList $= \{\}$
    for $p = \max(i - \text{MRL}, 0), ..., i - 1$ **do** ▷ (MRL is the max rule length; we ignore stacks with index less than $i - \text{MRL}$ because there is no rule in the grammar long enough to expand them to fit into stack i)
        $\{G\} = \text{Grouped}(S_p)$ ▷ (Group hypotheses in S_p based on their first uncovered span)
        for $g \in \{G\}$ **do** ▷ (Each group $g \in G$ is a tuple (g_{span}, g_{hyps}) where g_{hyps} is a list of hypotheses with the same first uncovered span g_{span})
            $[u, v] = g_{span}$
            R = GetSpanRules($[u, v]$) ▷ (Find all rules which can cover the span $[u, v]$)
            for $R_s \in R$ **do**
                cube $= [g_{hyps}, R_s]$ ▷ Create a "cube" which combines an existing hypothesis with a rule that can be used to expand that hypothesis.
                Add *cube* to *cubeList*
    $S_i = \text{Merge}(\textit{cubeList}, \mathcal{F})$ ▷ (Create stack S_i and add new hypotheses to it)
return arg $\min_{h \in S_n} h_c$

function MERGE(*cubeList*,$\mathcal{F}$)
    heapQ $= \{\}$
    for each (H, R) in *cubeList* **do**
        $h' = \text{getBestHypotheses}((H, R), \mathcal{F})$ ▷ (Given a hypothesis-rule pair and an estimate for future costs, create a new hypothesis. The source side of R might combine with the hypothesis's existing parse tree via adjunction: thus this step requires a TAG parser to determine which spans remain uncovered in the source string, rather than a CFG parser as used in the original algorithm.)
        push(*heapQ*, $(h'_c, h', [H, R])$) ▷ (Push new hypothesis to the queue. h'_c is the cost of the new hypothesis.)
    hypList $= \{\}$
    while $|\textit{heapQ}| > 0$ and $|\textit{hypList}| < K$ **do** ▷ (This loop pops the K best hypotheses (and their neighbors) from *heapQ* and adds them to *hypList*. The neighbors of $[H, R]$ are hypotheses which expand H using a rule/tree pair with the same source side as R but a different target side.)
        $(h'_c, h', [H, R]) = \text{pop}(\textit{heapQ})$ ▷ (Pop the best hypothesis)
        push(*heapQ*, GetNeighbors($[H, R]$)) ▷ (Push neighbors to queue)
        Add h' to *hypList*
    return *hypList* ▷ (Returns the best hypotheses.)

Appendix C

Code

Code, data, and instructions for reproducing the numbers graphed in Section 3.4 is publicly available at `https://github.com/MrLogarithm/scfg-plex`.