Prefix Lexicalization of Synchronous CFGs using Synchronous TAG

Logan Born and **Anoop Sarkar**
Simon Fraser University
School of Computing Science
{loborn,anoop}@sfu.ca

Abstract

We show that an ε-free, chain-free synchronous context-free grammar (SCFG) can be converted into a weakly equivalent synchronous tree-adjoining grammar (STAG) which is prefix lexicalized. This transformation at most doubles the grammar's rank and cubes its size, but we show that in practice the size increase is only quadratic. Our results extend Greibach normal form from CFGs to SCFGs and prove new formal properties about SCFG, a formalism with many applications in natural language processing.

1 Introduction

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 the famous theorems due to Shamir and Chomsky-Schützenberger (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).

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 a machine translation task. Using a prefix lexicalized grammar ensures that target sentences 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. To achieve these results, new grammars had to be heuristically constrained to include only prefix lexicalized productions, as there is at present no way to automatically convert an existing SCFG to a prefix lexicalized form.

This work investigates the formal properties of prefix lexicalized synchronous grammars as employed by Watanabe et al. (2006) and Siahbani et al. (2013), which have received little theoretical attention compared to non-synchronous prefix lexicalized grammars. To this end, we first prove that SCFG is not closed under prefix lexicalization. Our main result is that there is a method for prefix lexicalizing an SCFG by converting it to an equivalent grammar in a different formalism, namely synchronous tree-adjoining grammar (STAG) in regular form. Like the GNF transformation for CFGs, our method at most cubes the grammar size, but we show empirically that the size increase is only quadratic for grammars used in existing NLP tasks. The rank is at most doubled, and we maintain $O(n^{3k})$ parsing complexity for grammars of rank k. We conclude that although SCFG does not have a prefix lexicalized normal form like GNF, our conversion to prefix lexicalized STAG offers a practical alternative.

2 Background

2.1 SCFG

An SCFG is a tuple $G = (N, \Sigma, P, S)$ 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 P is a finite set of synchronous rules of the form

$$\langle A_1 \to \alpha_1, A_2 \to \alpha_2 \rangle \tag{1}$$

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

[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).

Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers), pages 1160–1170
Melbourne, Australia, July 15 - 20, 2018.

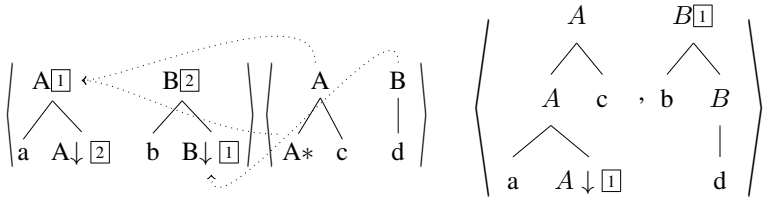


Figure 1: An example of synchronous rewriting in an STAG (left) and the resulting tree pair (right).

must be linked to exactly one nonterminal in α_2, and *vice versa*. We write these links using numerical annotations, as in (2).

(2) $\langle A \rightarrow A\boxed{1}B\boxed{2}, B \rightarrow B\boxed{2}A\boxed{1} \rangle$

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

In every step of an SCFG derivation, we rewrite one pair of linked nonterminals with a rule from P, in essentially the same way we would rewrite a single nonterminal in a non-synchronous CFG. For example, (3) shows linked A and B nodes being rewritten using (2):

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

Note how the $\boxed{1}$ and $\boxed{2}$ in rule (2) are renumbered to $\boxed{2}$ and $\boxed{3}$ during rewriting, to avoid an ambiguity with the $\boxed{1}$ already present in the derivation.

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

2.2 STAG

An 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

(4) $\langle t_1, t_2 \rangle$

where t_1 and t_2 are elementary trees as defined in Joshi et al. (1975). A *substitution site* is a leaf node marked by $\downarrow$ which may be rewritten by another tree; a *foot node* is a leaf marked by $*$ that may be used to rewrite a tree-internal node. Every substitution site in t_1 must be linked to exactly one nonterminal in t_2, and *vice versa*. As in SCFG, we write these links using numbered annotations; rank is defined for STAG the same way as for 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 adjunction operations defined for non-synchronous TAG. For example, Figure 1 shows linked A and B nodes being rewritten and the tree pair resulting from this operation. See Joshi et al. (1975) for details about the underlying TAG formalism.

2.3 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 terminology captures the intuition that synchronous grammars model translational equivalence between a source phrase and its translation into a target language. Other authors refer to the two halves as 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).

We call a grammar ε-free if it contains no productions whose source or target side produces only the empty string ε.

2.4 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 when 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 (PL-SCFG) is

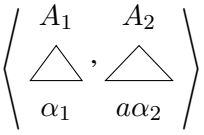
[2] All of the proofs in this work admit a symmetrical variant which prefix lexicalizes the source side instead of the target. We are not aware of any applications in NLP where source-side prefix lexicalization is useful, so we do not address this case.

of the form

$$\langle A_1 \to \alpha_1, A_2 \to a\alpha_2 \rangle \tag{5}$$

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

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

(6) 

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

3 Closure under Prefix Lexicalization

We now prove that the class SCFG is not closed under prefix lexicalization.

Theorem 1. *There exists an SCFG which cannot be converted to an equivalent PL-SCFG.*

Proof. The SCFG in (7) generates the language $L = \{\langle a^i b^j c^i, b^j a^i \rangle \mid i \geq 0, j \geq 1\}$, but this language cannot be generated by any PL-SCFG:

$$\begin{array}{ll} \langle S \to A\boxed{1}, & S \to A\boxed{1} \rangle \\ \langle A \to aA\boxed{1}c, & A \to A\boxed{1}a \rangle \\ \langle A \to bB\boxed{1}, & A \to bB\boxed{1} \rangle \\ \langle A \to b, & A \to b \rangle \\ \langle B \to bB\boxed{1}, & B \to bB\boxed{1} \rangle \\ \langle B \to b, & B \to b \rangle \end{array} \tag{7}$$

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 nontermials 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 $\alpha_5, \alpha_6, \alpha_8 \in (N \cup \Sigma)^*$, $\alpha_7 \in \Sigma(N \cup \Sigma)^*$

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, 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:

(8)
$$\begin{array}{ll} & \langle S\boxed{1}, S\boxed{1} \rangle \\ \Rightarrow^* & \langle \alpha_1 X\boxed{1} \alpha_2, b\alpha_3 Y\boxed{1} \alpha_4 \rangle \\ \Rightarrow^* & \langle \alpha_1 a^q X\boxed{1} c^q \alpha_2, b\alpha_3 a^r Y\boxed{1} a^s \alpha_4 \rangle \\ \Rightarrow^* & \langle \alpha_1 a^q \alpha_5 U\boxed{1} \alpha_6 c^q \alpha_2, b\alpha_3 a^r \alpha_7 V\boxed{1} \alpha_8 a^s \alpha_4 \rangle \\ \Rightarrow^* & \langle \alpha_1 a^q \alpha_5 b^k U\boxed{1} b^m \alpha_6 c^q \alpha_2, \\ & \quad b\alpha_3 a^r \alpha_7 b^n V\boxed{1} b^p \alpha_8 a^s \alpha_4 \rangle \end{array}$$

But since $n, r \geq 1$, the target string derived this way contains an a before a b and 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. ■

There also exist grammars which cannot be prefix lexicalized because they contain cyclic chain rules. If an SCFG can derive something of the form $\langle X\boxed{1}, Y\boxed{1} \rangle \Rightarrow^* \langle xX\boxed{1}, Y\boxed{1} \rangle$, then it can 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 unable to generate the original language where a source string may be unboundedly longer than its corresponding target. We call an SCFG chain-free if it does not contain a cycle of chain rules of this form. The remainder of this paper focuses on chain-free grammars, like (7), which cannot be converted to PL-SCFG despite containing no such cycles.

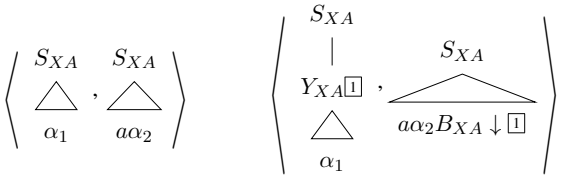
4 Prefix Lexicalization using STAG

We now present a method for prefix lexicalizing an SCFG by converting it to an STAG.

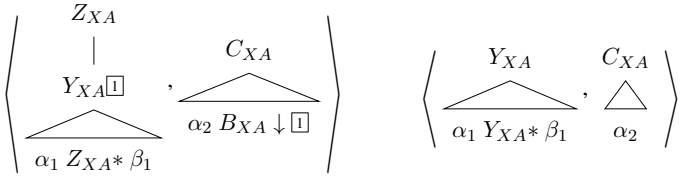
$$\langle X\boxed{1}, A\boxed{1}\rangle \Rightarrow \langle \alpha_1 Y_1\boxed{1}\beta_1, B_1\boxed{1}\gamma_1\rangle \Rightarrow \langle \alpha_1\alpha_2 Y_2\boxed{1}\beta_2\beta_1, B_2\boxed{1}\gamma_2\gamma_1\rangle$$

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

Figure 2: A target-side terminal leftmost derivation. $a \in \Sigma$; $X, A, Y_i, B_i \in N$; and $\alpha_i, \beta_i, \gamma_i \in (N \cup \Sigma)^*$.



(a) $\langle X \to \alpha_1, A \to a\alpha_2\rangle$ (b) $\langle Y \to \alpha_1, B \to a\alpha_2\rangle$



(c) $\langle Y \to \alpha_1 Z\boxed{1}\beta_1, B \to C\boxed{1}\alpha_2\rangle$ (d) $\langle X \to \alpha_1 Y\boxed{1}\beta_1, A \to C\boxed{1}\alpha_2\rangle$

Figure 3: Tree-pairs in G_{XA} and the rules in G from which they derive.

Theorem 2. *Given a rank-k SCFG G which is ε-free and chain-free, an STAG H exists such that H is prefix lexicalized and $L(G) = L(H)$. The rank of H is at most $2k$, and $|H| = O(|G|^3)$.*

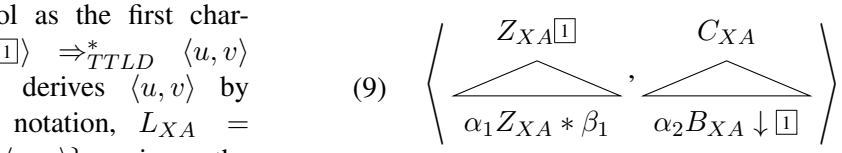
Proof. Let $G = (N, \Sigma, P, S)$ be an ε-free, chain-free SCFG. We provide a constructive method for prefix lexicalizing the target side of G.

We begin by constructing an intermediate grammar G_{XA} for each pair of nonterminals $X, A \in N \setminus \{S\}$. For each pair $X, A \in N \setminus \{S\}$, G_{XA} will be constructed to generate the language of sentential forms derivable from $\langle X\boxed{1}, A\boxed{1}\rangle$ via a target-side terminal leftmost derivation (TTLD). A TTLD is a derivation of the form in Figure 2, where the leftmost nonterminal in the target string is expanded until it produces a terminal symbol as the first character. We write $\langle X\boxed{1}, A\boxed{1}\rangle \Rightarrow^*_{TTLD} \langle u, v\rangle$ to mean that $\langle X\boxed{1}, A\boxed{1}\rangle$ derives $\langle u, v\rangle$ by way of a TTLD; in this notation, $L_{XA} = \{\langle u, v\rangle | \langle X\boxed{1}, A\boxed{1}\rangle \Rightarrow^*_{TTLD} \langle u, v\rangle\}$ is the language of sentential forms derivable from $\langle X\boxed{1}, A\boxed{1}\rangle$ via a TTLD.

Given $X, A \in N \setminus \{S\}$ we formally define G_{XA} as an STAG over the terminal alphabet $\Sigma_{XA} = N \cup \Sigma$ and nonterminal alphabet $N_{XA} = \{Y_{XA} | Y \in N\}$, with start symbol S_{XA}. N_{XA} contains nonterminals indexed by XA to ensure that two intermediate grammars G_{XA} and G_{YB} do not interact as long as $\langle X, A\rangle \neq \langle Y, B\rangle$. G_{XA} contains four kinds of tree pairs: [3]

- For each rule in G of the form $\langle X \to \alpha_1, A \to a\alpha_2\rangle$, $a \in \Sigma$, $\alpha_i \in (N \cup \Sigma)^*$, we add a tree pair of the form in Figure 3(a).
- For each rule in G of the form $\langle Y \to \alpha_1, B \to a\alpha_2\rangle$, $a \in \Sigma$, $\alpha_i \in (N \cup \Sigma)^*$, $Y, B \in N \setminus \{S\}$, we add a tree pair of the form in Figure 3(b).
- For each rule in G of the form $\langle Y \to \alpha_1 Z\boxed{1}\beta_1, B \to C\boxed{1}\alpha_2\rangle$, $Y, Z, B, C \in N \setminus \{S\}$, $\alpha_i, \beta_i \in (N \cup \Sigma)^*$, we add a tree pair of the form in Figure 3(c).

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

  (9) 

- For each rule in G of the form $\langle X \to \alpha_1 Y\boxed{1}\beta_1, A \to C\boxed{1}\alpha_2\rangle$, $Y, C \in N$, $\alpha_i, \beta_i \in (N \cup \Sigma)^*$, we add a tree pair of the form in Figure 3(d).

[3] In all cases, we assume that symbols in N (not N_{XA}) retain any links they bore in the original grammar, even though they belong to the terminal alphabet in G_{XA} 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.

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

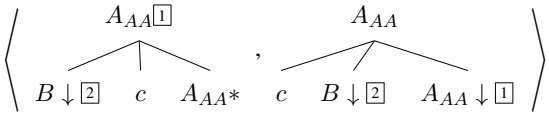


Figure 4: An SCFG rule and a tree pair based off that rule, taken from an intermediate grammar G_{AA}. The tree pair is formed according to the pattern illustrated in Figure 3(c). Observe that the B nodes retain the link they bore in the original rule. This link is not functional in the intermediate grammar (that is, it cannot be used for synchronous rewriting) because $B \notin N_{AA}$, but it will be functional when this tree pair is added to the final grammar H.

Figure 4 gives a concrete example of constructing an intermediate grammar tree pair on the basis of an SCFG rule.

Lemma 1. *G_{XA} generates the language L_{XA}.*

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

As a base case, observe that a tree of the shape in Figure 3(a) corresponds straightforwardly to the derivation

$$\langle X\boxed{1}, A\boxed{1} \rangle \Rightarrow \langle \alpha_1, a\alpha_2 \rangle \tag{10}$$

which is a TTLD starting from $\langle X, A \rangle$. By construction, therefore, every TTLD of the shape in (10) corresponds to some tree in G_{XA} of shape 3(a); likewise every derivation in G_{XA} comprising a single tree of shape 3(a) corresponds to a TTLD of the shape in (10).

As a second base case, note that a tree of the shape in Figure 3(b) corresponds to the last step of a TTLD like (11):

$$\langle X\boxed{1}, A\boxed{1} \rangle \Rightarrow^*_{TTLD} \langle uY\boxed{1}v, B\boxed{1}w \rangle \Rightarrow \langle u\alpha_1 v, a\alpha_2 w \rangle \tag{11}$$

In the other direction, the last step of any TTLD of the shape in (11) will involve some rule of the shape $\langle Y \to \alpha_1, B \to a\alpha_2 \rangle$; by construction G_{XA} must contain a corresponding tree pair of shape 3(b).

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

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

$$\langle Y \to \alpha_1 Z\boxed{1}\beta_1, B \to C\boxed{1}\alpha_2 \rangle \tag{12}$$

The existence of such a rule in G implies the existence of a corresponding tree in G_{XA} of the shape in Figure 3(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.[4] In the other direction, if the $n+1$th tree[5] of a derivation in G_{XA} is of the shape in Figure 3(c), then this implies the existence of a production in G of the shape in (12). By assumption the first n trees of the derivation in G_{XA} correspond to some TTLD in G; by prepending the rule from (12) to this TTLD we obtain a new TTLD of length $n+1$ which corresponds to the entire $n+1$ tree derivation in G_{XA}.

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

$$\langle X \to \alpha_1 Y\boxed{1}\beta_1, A \to C\boxed{1}\alpha_2 \rangle \tag{13}$$

The existence of such a rule implies the existence of a corresponding tree in G_{XA} of the shape in Figure 3(d). Adding this tree 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 of an $n+1$ tree derivation in G_A is of the shape in Figure 3(d), then this implies the

[4] It is easy to verify by inspection of Figure 3 that whenever one rule from G can be applied to the output of another rule, then the tree pairs in G_{XA} 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.

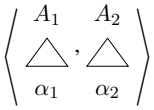
[5] Although trees in G_{XA} may contain symbols from the nonterminal alphabet of G, these symbols belong to the *terminal* alphabet in G_{XA}. Only nonterminals in N_{XA} will be involved in this derivation, and by construction there is at most one such nonterminal per tree. Thus a well-formed derivation structure in G_{XA} 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.

existence of a production in G of the shape in (13). By assumption the first n trees of the derivation in G_{XA} correspond to some TTLD in G; by prepending the rule from (13) to this TTLD we obtain a new TTLD of length $n+1$ which corresponds to the entire $n+1$ tree derivation in G_{XA}.

Taken together, these cases establish a one-to-one correspondence between derivations in G_{XA} and TTLDs which start from $\langle X, A\rangle$; in turn they confirm that G_{XA} generates the desired language L_{XA}. □

Once we have constructed an intermediate grammar G_{XA} for each $X, A \in N \setminus \{S\}$, 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

   (14) 

   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 $X, A \in N \setminus \{S\}$, add all of the tree pairs from the intermediate grammar G_{XA} to the new grammar H. Expand N to include the new nonterminal symbols in N_{XA}.

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

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

5. For all $X, A \in N$, let T_{XA} be defined as above. Remove all tree pairs in T_{XA} from H.

6. For all $X, A \in N$, let R_{XA} be defined as above. Remove all tree pairs in R_{XA} from H.

We now claim that H generates the same language as the original grammar G, and all of the target 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_{XA} nodes is identical to the set generable from $\langle X, A\rangle$ 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. Observe that, by construction, for all $X, A \in N$ every target tree rooted in S_{XA} 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 5 gives an example of this transformation applied to a small grammar. Note how A nodes at the left edge of the target trees end up rewritten as S_{AA} nodes, as per step 4 of the transformation.

5 Complexity & Formal Properties

Our conversion generates a subset of the class of prefix lexicalized STAGs in regular form, which we abbreviate to PL-RSTAG (regular form for TAG is defined in Rogers 1994). This section discusses some formal properties of PL-RSTAG.

Generative Capacity PL-RSTAG is weakly equivalent to the class of ε-free, chain-free SCFGs: this follows immediately from the proof that our transformation does not change the language generated by the input SCFG. Note that every TAG in regular form generates a context-free language (Rogers, 1994).

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, as noted for example in Figure 4. Thus subtrees which are gener-

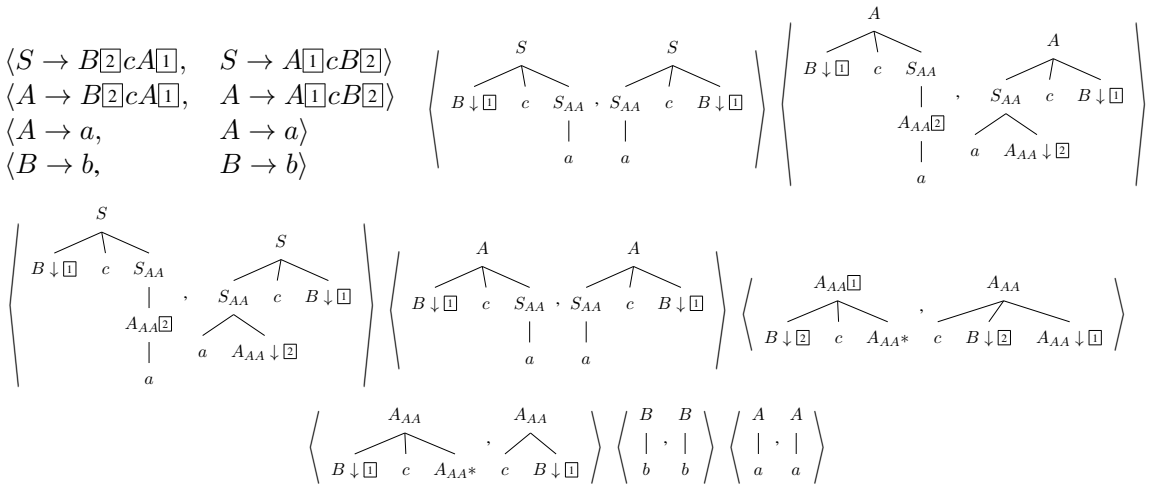


Figure 5: An SCFG and the STAG which prefix lexicalizes it. Non-productive trees have been omitted.

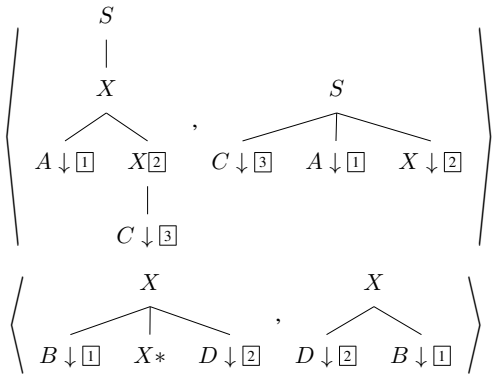
Grammar	$\|G\|$	$\|H\|$	% of G prefix lexicalized	$\log_{\|G\|}(\|H\|)$
Siahbani and Sarkar (2014a) (Zh-En)	18.5M	23.6T	63%	1.84
Example (7)	6	14	66%	1.47
ITG (10000 translation pairs)	10,003	170,000	99.97%	1.31

Table 1: Grammar sizes before and after prefix lexicalization, showing $O(n^2)$ size increase instead of the worst case $O(n^3)$. $|G|$ and $|H|$ give the grammar size before and after prefix lexicalization; $\log_{|G|} |H|$ is the increase as a power of the initial size. We also show the percentage of productions which are already prefix lexicalized in G.

ated 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] This result holds despite the fact that our transformation is only applicable to chain-free grammars: chain rules cannot introduce any reorderings, since by definition they involve only a single pair of linked nonterminals.

Grammar Rank If the input SCFG G has rank k, then the STAG H produced by our transformation has rank at most $2k$. To see this, observe that the construction of the intermediate grammars increases the rank by at most 1 (see Figure 3(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 $2k$.

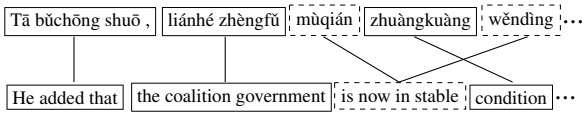
In the general case, rank-k STAG is more powerful than rank-k SCFG; for example, a rank-4 SCFG is required to generate the reordering in $\langle S \rightarrow A\boxed{1}B\boxed{2}C\boxed{3}D\boxed{4}, S \rightarrow C\boxed{3}A\boxed{1}D\boxed{4}B\boxed{2}\rangle$ (Wu, 1997), but this reordering is captured by the following rank-3 STAG:



For this reason, we speculate that it is possible to further transform the grammars produced by our lexicalization in order to reduce their rank, but the details of this transformation remain as future work.

This potentially poses a solution to an issue raised by Siahbani and Sarkar (2014b). On a Chinese-English translation task, they find that sentences like (15) involve reorderings which cannot be captured by a rank-2 prefix lexicalized SCFG:

(15)



If rank-k PL-RSTAG is more powerful than rank-k

[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 remembered and used to reconstruct the reorderings which occurred between the two sentences.

SCFG, using a PL-RSTAG here would permit capturing more reorderings without using grammars of higher rank.

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 and Experiments 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 compared to $|G|$, a tighter bound is given by $O(|G|^2|N|^2)$.

Table 1 shows the actual size increase on a variety of grammars: here $|G|$ is the size of the initial grammar, $|H|$ is the size after applying our transformation, and the increase is expressed as a power of the original grammar size. We apply our transformation to the grammar from Siahbani and Sarkar (2014a), which was created for a Chinese-English translation task known to involve complex reorderings that cannot be captured by PL-SCFG (Siahbani and Sarkar, 2014b). We also consider the grammar in (7) and an ITG (Wu, 1997) containing 10,000 translation pairs, which is a grammar of the sort that has previously been used for word alignment tasks (cf Zhang and Gildea 2005). We always observe an increase within $O(|G|^2)$ rather than the worst-case $O(|G|^3)$, because $|N|$ is small relative to $|G|$ in most grammars used for NLP tasks.

We also investigated how the proportion of prefix lexicalized rules in the original grammar affects the overall size increase. We sampled grammars with varying proportions of prefix lexicalized rules from the grammar in Siahbani and Sarkar (2014a); Table 2 shows the result of lexicalizing these samples. We find that the worst case size increase occurs when 50% of the original grammar is already prefix lexicalized. This is because the size increase depends on both the number of prefix lexicalized trees in the intermediate grammars (which grows with the proportion of lexicalized rules) and the number of productions which need to be lexicalized (which shrinks as the proportion of prefix lexicalized rules increases). At 50%, both factors contribute appreciably to the grammar size, analogous to how the function $f(x) = x(1-x)$ takes its maximum at $x = 0.5$.

$\|G\|$	$\|H\|$	% of G prefix lexicalized	$\log_{\|G\|}(\|H\|)$
15k	42.4M	10%	1.83
15k	74.9M	20%	1.89
15k	97.8M	30%	1.91
15k	112M	40%	1.93
15k	118M	50%	1.93
15k	114M	60%	1.93
15k	102M	70%	1.92
15k	78.2M	80%	1.89
15k	43.6M	90%	1.83

Table 2: Effect of prefix lexicalized rules in G on final grammar size.

6 Applications

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.

Combined with the transformation in Section 4, 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, 2014a). Now, an SCFG tailored for a translation task can be transformed directly to PL-RSTAG and used for decod-

ing; unlike a heuristically induced PL-SCFG, the transformed PL-RSTAG will generate the same language as the original SCFG which is known to handle more reorderings.

Note that, since applying our transformation may double the rank of a grammar, this method may prove prohibitively slow. This highlights the need for future work to examine the generative power of rank-k PL-RSTAG relative to rank-k SCFG in the interest of reducing the rank of the transformed grammar.

7 Related Work

Our work continues the study of TAGs and lexicalization (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 (CFTGs).

Other extensions of GNF to new grammar formalisms include Dymetman (1992) for definite clause grammars, Fernau and Stiebe (2002) for CF valence grammars, and Engelfriet et al. (2017) for multiple CFTGs. Although multiple CFTG subsumes SCFG (and STAG), Engelfriet et al.'s result appears to guarantee only that *some* side of every synchronous production will be lexicalized, whereas our result guarantees that it is always the target side that will be prefix lexicalized.

Lexicalization of synchronous grammars was addressed by Zhang and Gildea (2005), but they consider lexicalization rather than prefix lexicalization, and they only consider SCFGs of rank 2. They motivate their results using a word alignment task, which may be another possible application for our lexicalization.

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_{XA}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) for the claim that our transformed grammars generate context-free languages despite allowing wrapping adjunction; an alternative proof could employ the results of Swanson et al. (2013), who develop their own context-free TAG variant known as osTAG.

Kaeshammer (2013) introduces the class of synchronous linear context-free rewriting systems to model reorderings which cannot be captured by a rank-2 SCFG. In the event that rank-k PL-RSTAG is more powerful than rank-k SCFG, our work can be seen as an alternative approach to the same problem.

Finally, Nesson et al. (2008) present an algorithm for reducing the rank of an STAG on-the-fly during parsing; this presents a promising avenue for proving a smaller upper bound on the rank increase caused by our transformation.

8 Conclusion and 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- and chain-free. Like the original GNF transformation for CFGs our construction at most cubes the grammar size, though 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 retains properties such as $O(n^{3k})$ parsing complexity for grammars of rank k. We plan to verify whether rank-k PL-RSTAG is more powerful than rank-k SCFG in future work, and to reduce the rank of the transformed grammar if possible. We further plan to empirically evaluate our lexicalization on an alignment task and to offer a comparison against the lexicalization due to Zhang and Gildea (2005).

Acknowledgements

The authors wish to thank the anonymous reviewers for their helpful comments. The research was also partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC RGPIN-2018-06437 and RGPAS-2018-522574) to the second author. We dedicate this paper to the memory of Prof. Aravind Joshi; a short hallway conversation with him at ACL 2014 was the seed for this paper.

References

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

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

Yehoshua Bar-Hillel, M. Perles, and Eliahu Shamir. 1961. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung* 14:143–172. 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. 1999. Greibach normal form transformation revisited. *Information and Computation* 150(1):112–118. https://doi.org/10.1006/inco.1998.2772.

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

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

Pierluigi Crescenzi, Daniel Gildea, Andrea Marino, Gianluca Rossi, and Giorgio Satta. 2015. Synchronous context-free grammars and optimal linear parsing strategies. *Journal of Computer and System Sciences* 81(7):1333–1356. https://doi.org/10.1016/j.jcss.2015.04.003.

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

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

Henning Fernau and Ralf Stiebe. 2002. Sequential grammars and automata with valences. *Theoretical Computer Science* 276(1):377–405. https://doi.org/10.1016/S0304-3975(01)00282-1.

James N. Gray and Michael A. Harrison. 1972. On the covering and reduction problems for context-free grammars. *Journal of the ACM* 19(4):675–698. https://doi.org/10.1145/321724.321732.

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

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

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

Miriam Kaeshammer. 2013. Synchronous linear context-free rewriting systems for machine translation. In M. Carpuat, L. Specia, and D. Wu, editors, *Proceedings of the Seventh Workshop on Syntax, Semantics and Structure in Statistical Translation, SSST@NAACL-HLT 2013, Atlanta, GA, USA, 13 June 2013*. Association for Computational Linguistics, pages 68–77. http://aclweb.org/anthology/W/W13/W13-0808.pdf.

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

Philip M. Lewis and Richard E. Stearns. 1968. Syntax-directed transduction. *Journal of the ACM* 15(3):465–488. https://doi.org/10.1145/321466.321477.

Andreas Maletti and Joost Engelfriet. 2012. 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*. The Association for Computational Linguistics, pages 506–515. http://www.aclweb.org/anthology/P12-1053.

Rebecca Nesson, Giorgio Satta, and Stuart M. Shieber. 2008. Optimal k-arization of synchronous tree-adjoining grammar. In *Proceedings of ACL-08: HLT*. Association for Computational Linguistics, Columbus, Ohio, pages 604–612. http://www.aclweb.org/anthology/P/P08/P08-1069.pdf.

James Rogers. 1994. Capturing CFLs with tree adjoining grammars. In *Proceedings of the 32nd Annual Meeting of the Association for Computational*

Linguistics. Association for Computational Linguistics, Stroudsburg, PA, USA, ACL '94, pages 155–162. https://doi.org/10.3115/981732.981754.

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

Yves Schabes and Richard C. Waters. 1993. 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.*. ACL, pages 121–129. http://aclweb.org/anthology/P/P93/P93-1017.pdf.

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

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

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

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

Maryam Siahbani and Anoop Sarkar. 2014a. 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*.

Maryam Siahbani and Anoop Sarkar. 2014b. 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*. ACL, pages 221–226. http://aclweb.org/anthology/D/D14/D14-1028.pdf.

Ben Swanson, Elif Yamangil, Eugene Charniak, and Stuart M. Shieber. 2013. 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*. The Association for Computational Linguistics, pages 302–310. http://aclweb.org/anthology/P/P13/P13-1030.pdf.

Taro Watanabe, Hajime Tsukada, and Hideki Isozaki. 2006. 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. http://aclweb.org/anthology/P06-1098.

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

Hao Zhang and Daniel Gildea. 2005. 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*. The Association for Computational Linguistics, pages 475–482. http://aclweb.org/anthology/P/P05/P05-1059.pdf.