An Introduction to Synchronous CFGs

David Chiang

November 15, 2005

1 Introduction

Synchronous context-free grammars, roughly speaking, are to context-free grammars (CFGs) what finite-state transducers are to finite-state automata. Like finite-state transducers, they define mappings between two formal languages; like context-free grammars, they work in a recursive way to achieve more complex operations than finite-state devices. Thus they are useful in many situations where one might want to specify a recursive relationship between two languages. Originally, they were developed in the late 1960s for programming-language compilation. In natural language processing, they have been used for machine translation [14, 15, 3] and (less commonly, perhaps) semantic interpretation.

As a preview, consider the following English sentence and its (admittedly somewhat unnatural) equivalent in Japanese (with English glosses):

(1) The boy stated that the student said that the teacher danced

(2) shoonen-ga gakusei-ga sensei-ga odotta to itta to hanasita
the boy the student the teacher danced that said that stated

One might imagine writing a finite-state transducer to perform a word-for-word translation between English and Japanese sentences, but not to perform the kind of reordering seen here. But a synchronous CFG can do this.

The term *synchronous CFG* is recent and far from universal. They were originally known as *syntax-directed transduction grammars* [7] or *syntax-directed translation schemata* [1], the latter still probably being the most common name. In the NLP community they are also known as 2-multitext grammars [8], and inversion transduction grammars [14] are a special case of synchronous CFGs.

2 Definition

We give only an informal definition here. A synchronous CFG is like a CFG, but its productions have two right-hand sides—call them the *source* side and the *target* side—that are related in a certain way. Below is an example synchronous CFG for a fragment of English and Japanese:

$$\text{S} \rightarrow \langle \text{NP}\boxed{1}\ \text{VP}\boxed{2}, \text{NP}\boxed{1}\ \text{VP}\boxed{2} \rangle \tag{3}$$

$$\text{VP} \rightarrow \langle \text{V}\boxed{1}\ \text{NP}\boxed{2}, \text{NP}\boxed{2}\ \text{V}\boxed{1} \rangle \tag{4}$$

$$\text{NP} \rightarrow \langle \text{i}, \text{watashi} \rangle \tag{5}$$

$$\text{NP} \rightarrow \langle \text{the box}, \text{wa hako} \rangle \tag{6}$$

$$\text{V} \rightarrow \langle \text{open}, \text{akemasu} \rangle \tag{7}$$

The boxed numbers $\boxed{i}$ link up nonterminal symbols on the source side with nonterminal symbols on the target side: $\boxed{1}$ links to $\boxed{1}$, $\boxed{2}$ with $\boxed{2}$, and so on. We assume here that this linking is a one-to-one correspondence, and that a nonterminal X is always linked to another X, never to a $Y \neq X$.

How does this grammar work? Just as we start in a CFG with a start symbol and repeatedly rewrite nonterminal symbols using the productions, so in a synchronous CFG, we start with a pair of linked start symbols (I just chose the number 10 arbitrarily),

$$\langle \text{S}\boxed{10}, \text{S}\boxed{10} \rangle$$

and repeatedly rewrite pairs of nonterminal symbols using the productions—with two wrinkles. First, when we apply a production, we renumber the boxed indices consistently to fresh indices that aren't in our working string pair. Thus, applying production (3), we get

$$\Rightarrow \langle \text{NP}\boxed{11}\ \text{VP}\boxed{12}, \text{NP}\boxed{11}\ \text{VP}\boxed{12} \rangle$$

Second, we are only allowed to rewrite *linked* nonterminal symbols. Thus we can apply production (4) like so:

$$\Rightarrow \langle \text{NP}\boxed{11}\ \text{V}\boxed{13}\ \text{NP}\boxed{14}, \text{NP}\boxed{11}\ \text{NP}\boxed{14}\ \text{V}\boxed{13} \rangle$$

But now if we want to apply production (5), we can't apply it to $\text{NP}\boxed{11}$ on one side and $\text{NP}\boxed{14}$ on the other, like this:

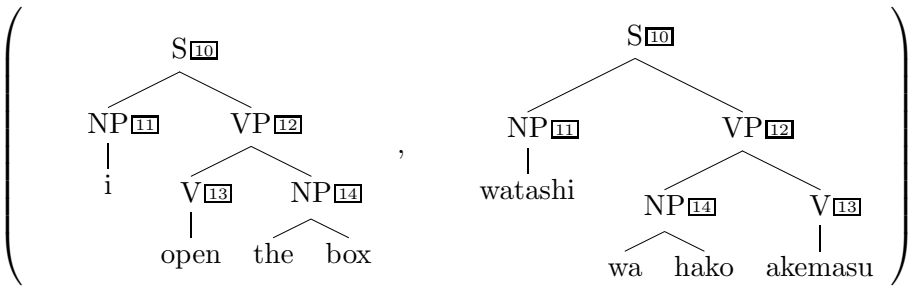
$$\Rightarrow \langle \text{i V}\boxed{13}\ \text{NP}\boxed{14}, \text{NP}\boxed{11}\ \text{watashi V}\boxed{13} \rangle \qquad \textbf{not allowed}$$

But we can apply it to any linked nonterminals, like so:

$$\Rightarrow \langle \text{i V}\boxed{13}\ \text{NP}\boxed{14}, \text{watashi NP}\boxed{14}\ \text{V}\boxed{13}\rangle$$
$$\Rightarrow \langle \text{i open NP}\boxed{14}, \text{watashi NP}\boxed{14}\ \text{akemasu}\rangle$$
$$\Rightarrow \langle \text{i open the box}, \text{watashi wa hako akemasu}\rangle$$

And now we have an English string and Japanese string which are translations of each other!

We can also view synchronous CFG derivations as pairs of trees, just as CFG derivations can be viewed as trees:



By this point, we often don't care about the boxed numbers and therefore drop them.

Here is a more complicated example, as promised:

$$\text{S} \rightarrow \langle \text{NP}\boxed{1}\ \text{VP}\boxed{2}, \text{NP}\boxed{1}\ \text{VP}\boxed{2}\rangle \tag{8}$$
$$\text{VP} \rightarrow \langle \text{V}\boxed{1}, \text{V}\boxed{1}\rangle \tag{9}$$
$$\text{VP} \rightarrow \langle \text{V}\boxed{1}\ \bar{\text{S}}\boxed{2}, \bar{\text{S}}\boxed{2}\ \text{V}\boxed{1}\rangle \tag{10}$$
$$\bar{\text{S}} \rightarrow \langle \text{Comp}\boxed{1}\ \text{S}\boxed{2}, \text{S}\boxed{2}\ \text{Comp}\boxed{1}\rangle \tag{11}$$
$$\text{Comp} \rightarrow \langle \text{that}, \text{to}\rangle \tag{12}$$
$$\text{NP} \rightarrow \langle \text{the boy}, \text{shoonen-ga}\rangle \tag{13}$$
$$\text{NP} \rightarrow \langle \text{the student}, \text{gakusei-ga}\rangle \tag{14}$$
$$\text{NP} \rightarrow \langle \text{the teacher}, \text{sensei-ga}\rangle \tag{15}$$
$$\text{V} \rightarrow \langle \text{danced}, \text{odotta}\rangle \tag{16}$$
$$\text{V} \rightarrow \langle \text{said}, \text{itta}\rangle \tag{17}$$
$$\text{V} \rightarrow \langle \text{stated}, \text{hanasita}\rangle \tag{18}$$

The derivation of sentences (1) and (2) is left as an exercise.

3 Properties

Synchronous CFGs have a number of properties that differ (perhaps surprisingly) from CFGs on the one hand or finite-state transducers on the other hand.

No Chomsky normal form. Define the *rank* of a right-hand side to be the number of nonterminals in it: for example, NP VP has rank two. Now define the rank of a CFG or synchronous CFG to be the maximum rank of any of its right-hand sides.

Recall that any (non-synchronous) CFG can be converted into a (weakly) equivalent CFG with rank two or less (Chomsky normal form). Is this true for synchronous CFG? It turns out that any synchronous CFG of rank three can be converted into a synchronous CFG of rank two. For example, the production

$$\mathrm{A} \rightarrow \langle \mathrm{B}\boxed{1}\ \mathrm{C}\boxed{2}\ \mathrm{D}\boxed{3}, \mathrm{D}\boxed{3}\ \mathrm{B}\boxed{1}\ \mathrm{C}\boxed{2} \rangle$$

can be binarized into

$$\mathrm{A} \rightarrow \langle \mathrm{A}'\boxed{4}\ \mathrm{D}\boxed{3}, \mathrm{D}\boxed{3}\ \mathrm{A}'\boxed{4} \rangle$$
$$\mathrm{A}' \rightarrow \langle \mathrm{B}\boxed{1}\ \mathrm{C}\boxed{2}, \mathrm{B}\boxed{1}\ \mathrm{C}\boxed{2} \rangle$$

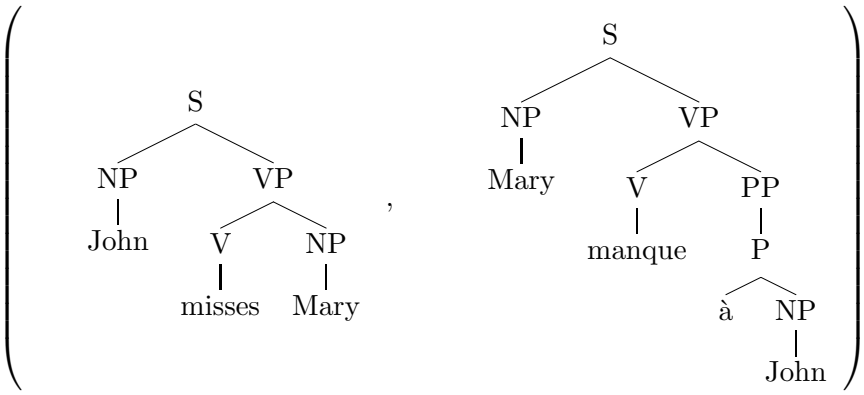
Note that we did not sever any links in doing so. But there are synchronous CFGs of rank four that can't be binarized—namely, any synchronous CFG containing a production similar to the following:

$$\mathrm{A} \rightarrow \langle \mathrm{B}\boxed{1}\ \mathrm{C}\boxed{2}\ \mathrm{D}\boxed{3}\ \mathrm{E}\boxed{4}, \mathrm{D}\boxed{3}\ \mathrm{B}\boxed{1}\ \mathrm{E}\boxed{4}\ \mathrm{C}\boxed{2} \rangle$$

Try to binarize it—no matter how you do it, you will always have to sever a link, which is not allowed. In general, there exists a synchronous CFG of rank $r+1$ that can't be converted into a synchronous CFG of rank r, for all $r \geq 3$ or $r = 1$ [1].

Closure under composition? We can think of a synchronous CFG as defining a relation on strings, just as a finite-state transducer defines a relation on strings. Recall that finite-state transducers are composable: if R and R' are definable by finite-state transducers, then so is $R \circ R'$. But this is not true for synchronous CFGs: if R and R' are string relations definable by synchronous CFGs, then $R \circ R'$ will not necessarily be. (The reason is related to the non-closure of CFLs under intersection.) However, if R and R' are *tree* relations definable by synchronous CFGs, then $R \circ R'$ *is* also definable by a synchronous CFG.

Sister-reordering only Synchronous CFGs can't do everything. If you think of a synchronous CFG as defining a (nondeterministic) transformation on trees (source to target), then all synchronous CFGs can do is relabel nodes and reorder sisters. So if you want to write a synchronous CFG that can translate between English and French trees:



then you're out of luck, because [$_{\text{NP}}$ John] and [$_{\text{NP}}$ Mary] aren't sister nodes, and therefore can't be swapped.

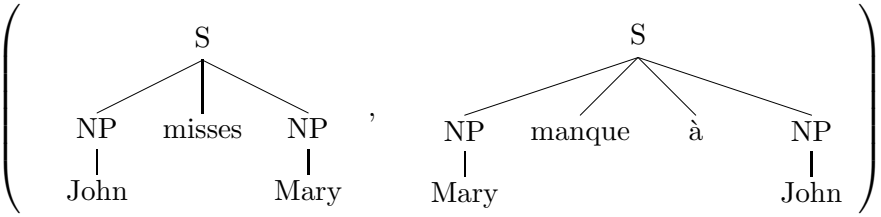
However, if you're willing to mangle the trees, then there is an easy solution:

$$\text{S} \rightarrow \langle \text{NP}\boxed{1} \text{ misses NP}\boxed{2}, \text{NP}\boxed{2} \text{ manque à NP}\boxed{1} \rangle \tag{19}$$

$$\text{NP} \rightarrow \langle \text{John}, \text{John} \rangle \tag{20}$$

$$\text{NP} \rightarrow \langle \text{Mary}, \text{Mary} \rangle \tag{21}$$

which will generate the tree pair



By flattening the trees, we made [$_{\text{NP}}$ John] and [$_{\text{NP}}$ Mary] sister nodes so that they could be swapped. (We also integrated the verb *misses/manque à* into the rule to make sure that this swapping doesn't occur with just any verb.)

4 Parsing

The parsing task for synchronous grammars comes in two flavors: we might be given a source sentence and asked to translate it into the target language, producing parse trees for both; or else we might be given a pair of sentences and asked to find parse trees for both. Surprisingly, perhaps, the latter problem, where we are given more information, is in general much more difficult.

4.1 Translation

Translation is a totally straightforward generalization of monolingual parsing. Basically, we view our synchronous CFG as an ordinary CFG by simply covering up the target right-hand sides; note that this could result in a CFG with duplicate rules, which should not be a problem. Then we parse. Finally, we transform the resulting source-language tree into a target-language tree by replacing each source right-hand side with its target right-hand side, reordering nodes as specified by the linking relation.

4.2 Bitext parsing

Recall that $\mathcal{O}(n^3)$-time parsing of general CFGs is related to the convertibility of general CFGs into Chomsky normal form. The fact that general synchronous CFGs *can't* be converted into synchronous CFGs of rank two is similarly related to the fact that parsing of *pairs* of sentences using a general synchronous CFG *can't* be done in $\mathcal{O}(n^k)$ time for any fixed k.[1] In general, k will depend on the rank of the synchronous CFG [9].

The case of synchronous CFGs of rank one is the simplest but not that interesting; in the case of synchronous CFGs of rank two or three, bitext parsing can be done in $\mathcal{O}(n^6)$ time [14]. Assume that each right-hand side (source or target) consists of

- two nonterminal symbols, or
- a single terminal symbol, or
- the empty string,

such that no production has two empty right-hand sides. Any synchronous CFG of rank two or three (or any inversion transduction grammar; see

[1] Unless $P = NP$.

the Appendix) can be put in this normal form. Then, here is a CKY-like algorithm for bitext parsing, assuming input strings $w = w_1 \cdots w_n$ and $w' = w'_1 \cdots w'_{n'}$. Following Shieber et al. [13], we present a parser as a deductive system: the axioms are taken as trivially provable items; the inference rules generate new items from old ones (if the top items are provable, then the bottom one is); and we accept the string pair if and only if we can prove the goal item.

Axioms	$\overline{[A, i, j, i', j']}$	$A \to \langle w_{j+1}, w'_{j'+1} \rangle$
	$\overline{[A, i, i, i', j']}$	$A \to \langle \epsilon, w'_{j'+1} \rangle$
	$\overline{[A, i, j, i', i']}$	$A \to \langle w_{j+1}, \epsilon \rangle$
Inference rules	$\dfrac{[B, i, j, i', j'] \quad [C, j, k, j', k']}{[A, i, k, i', k']}$	$A \to \langle B\boxed{1}\ C\boxed{2}, B\boxed{1}\ C\boxed{2} \rangle$
	$\dfrac{[B, i, j, j', k'] \quad [C, j, k, i', j']}{[A, i, k, i', k']}$	$A \to \langle B\boxed{1}\ C\boxed{2}, B\boxed{2}\ C\boxed{1} \rangle$
Goal	$[S, 0, n, 0, n']$	

Viewed this way, the algorithm looks pretty much like ordinary CKY, except that we have to keep track of spans on both the source and target sides.

Shieber et al. provide a general search algorithm for deductive parsers, but CKY is usually thought of as building a chart that is partitioned into *cells* $\langle i, j \rangle$ and filling the cells in order of increasing $j - i$. This guarantees fast matching of items and correct calculation of quantities like Viterbi or inside/outside probabilities. Here, we need cells $\langle i, j, i, j' \rangle$ which are to be filled in order of increasing $j - i + j' - i'$. The search algorithm, then, looks like this:

1: **for all** i, j, i', j' **do**
2: $\quad c[i, j, i', j'] = \emptyset$
3: **for all** axioms $[A, i, j, i', j']$ **do**
4: $\quad$ add $[A, i, j, i', j']$ to $c[i, j, i', j']$

5: **for** $m = 1$ to $n + n'$ **do**
6: **for all** i, k, i', k' s.t. $k - i + k' - i' = m$, $0 \leq i \leq k \leq n$, $0 \leq i' \leq k' \leq n'$ **do**
7: **for** nonterminal symbols $A \in V$ **do**
8: **if** $[A, i, k, i', k']$ can be proven from items in c **then**
9: add $[A, i, k, i', k']$ to $c[i, k, i', k']$
10: **if** $[S, 0, n, 0, n'] \in c[0, n, 0, n']$ **then**
11: accept $\langle w, w' \rangle$

For synchronous CFGs of fixed rank greater than three, bitext parsing is still polynomial, but slower.

Appendix: Variants

This appendix is a field guide to variations and extensions of synchronous CFG. It will probably be of interest only to those who are interested in exploring the literature further.

One left-hand side or two? The production

$$\text{VP} \rightarrow \langle \text{V}\boxed{1}\ \text{NP}\boxed{2}, \text{NP}\boxed{2}\ \text{V}\boxed{1} \rangle \tag{22}$$

is also often written as

$$\langle \text{VP} \rightarrow \text{V}\boxed{1}\ \text{NP}\boxed{2}, \text{VP} \rightarrow \text{NP}\boxed{2}\ \text{V}\boxed{1} \rangle \tag{23}$$

This opens the possibility of using different nonterminal symbols on source and target sides, for example:

$$\langle \text{A} \rightarrow \text{B}\boxed{1}\ \text{C}\boxed{2}, \text{D} \rightarrow \text{F}\boxed{2}\ \text{E}\boxed{1} \rangle \tag{24}$$

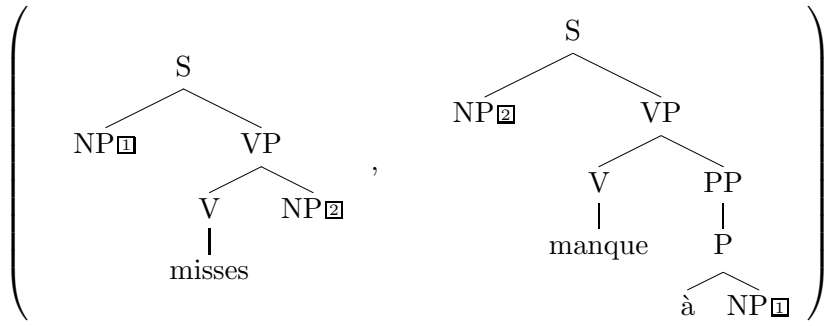
This allows more flexibility in writing the grammar, because the source side and target side are more decoupled from each other, and it also is arguably more conceptually elegant, because it better captures the idea that a synchronous CFG is just two CFGs linked together. The disadvantages are that these productions are harder to squeeze into a three-inch-wide column, and the translation algorithm is not as easy to formulate. In fact, the easiest way to formulate it is probably to convert the grammar into our original notation. This is done by creating a compound nonterminal alphabet:

$$\text{A/D} \rightarrow \langle \text{B/E}\boxed{1}\ \text{C/F}\boxed{2}, \text{C/F}\boxed{2}\ \text{B/E}\boxed{1} \rangle \tag{25}$$

and then proceed as above.

Incomplete linking Above we said that we would assume that the linking relation between nonterminal symbols on the source and target right-hand sides should be a one-to-one correspondence. We might relax this requirement to allow some nonterminal symbols to be unlinked. In that case, we would also have to add unpaired CFG productions with which to rewrite the unlinked nonterminal symbols. This would allow unbounded amounts of material to appear on the source side with no counterpart on the target side, or vice versa. We could convert any such grammar into a weakly-equivalent completely-linked synchronous CFG by liberal use of productions with empty right-hand sides.

Synchronous tree-adjoining grammar (TAG) [11] This is a more powerful formalism than synchronous CFG. Note that the 1994 formulation of synchronous TAG supersedes the original flawed formulation [12], although the earlier paper is still important for its discussion of applications. Both synchronous TAG and its weaker cousin, synchronous tree-substitution grammar [10, 5], are able to perform subject-object swapping as described above without flattening trees. Tree-substitution grammar is beyond the scope of this introduction, but here is the relevant production:



An interesting property (interesting to me, anyway) of formalisms in this territory is that two non-synchronous formalisms can be weakly equivalent but their synchronous versions might not be [2].

Inversion transduction grammar [14] Inversion transduction grammars (ITGs) are synchronous CFGs in which the links between nonterminals in a production are restricted to two possible configurations: either the first nonterminal links to the first, the second to the second, etc., or else the first links to the last, the second to the second-to-last, etc. Thus the boxed numbers can be replaced with a simpler notation involving two kinds

of brackets:

$$\begin{aligned} \mathrm{S} &\rightarrow [\mathrm{NP}, \mathrm{VP}] \\ \mathrm{VP} &\rightarrow \langle \mathrm{V}, \mathrm{NP} \rangle \\ \mathrm{NP} &\rightarrow \text{i/watashi} \\ \mathrm{NP} &\rightarrow \text{the-box/wa-hako} \\ \mathrm{V} &\rightarrow \text{open/akemasu} \end{aligned}$$

This grammar is equivalent to the synchronous CFG above (where we have turned *the box* and *wa hako* into single terminal symbols, to conform with the ITG notation while keeping things simple). Any ITG can be converted into a synchronous CFG of rank two.

This work was the first serious attempt, to my knowledge, to apply synchronous CFGs to statistical machine translation.

Multitext grammar [8] Multitext grammars (MTGs) are more general than synchronous CFGs because they allow not just two, but any number of right-hand sides, to specify an n-ary relation on strings. The two-dimensional case of multitext grammars is equivalent to synchronous CFGs, however. The following 2-multitext grammar rule is equivalent to our production (19):

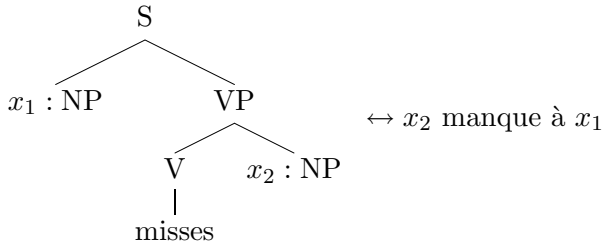
$$\mathrm{S} \rightarrow \bowtie \begin{matrix} [1,2,3] \\ [4,2,3,1] \end{matrix} \begin{pmatrix} \mathrm{NP} & \text{misses} & \epsilon & \mathrm{NP} \\ \mathrm{NP} & \text{manque} & \text{à} & \mathrm{NP} \end{pmatrix} \tag{26}$$

Note that the symbols inside the round parentheses don't appear in the order they will end up in; it is the numbers in square brackets, which correspond to our boxed numbers, that appear in order. Note, moreover, that MTG links terminals as well as nonterminals, but the empty string is not linked, nor is it assigned a number.

ISI models Two syntax-based machine translation systems have come out of ISI: that of Yamada and Knight [15] and the current one [4], based on "GHKM" rules [6]. Both are weakly equivalent to synchronous CFG. The first, however, generates the productions dynamically and doesn't actually represent them using any notation.

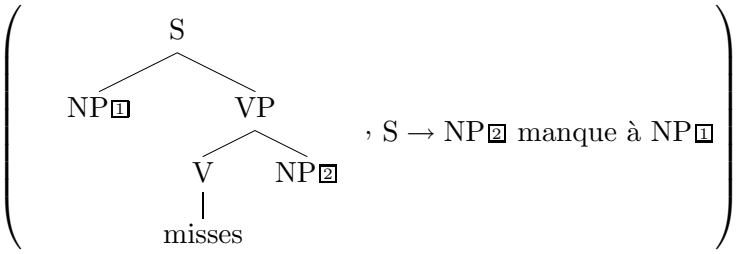
The GHKM notation is equivalent to synchronous CFG if we are willing to flatten trees, then we can say that their notation is equivalent to

synchronous CFG. An example production in their notation would be:



which would, after flattening, be equivalent to our production (19).

If we don't want to flatten trees, then the GHKM notation is equivalent to a synchronous grammar that has a CFG on one side and a tree-substitution grammar on the other. Thus the above rule would become:



References

[1] A. V. Aho and J. D. Ullman. Syntax directed translations and the pushdown assembler. *Journal of Computer and System Sciences*, 3:37–56, 1969.

[2] David Chiang. Putting some weakly context-free formalisms in order. In *Proceedings of the Sixth International Workshop on TAG and Related Formalisms (TAG+)*, pages 11–18, 2002.

[3] David Chiang. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting of the ACL*, pages 263–270, 2005.

[4] Steve DeNeefe and Kevin Knight. ISI's 2005 statistical machine translation entries. In *Proceedings of IWSLT 2005*, 2005.

[5] Jason Eisner. Learning non-isomorphic tree mappings for machine translation. In *Companion Volume to the Proceedings of the 41st Annual Meeting of the ACL*, pages 205–208, 2003. Poster/demonstration session.

[6] Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. What's in a translation rule? In *Proceedings of HLT-NAACL 2004*, pages 273–280, 2004.

[7] P. M. Lewis II and R. E. Stearns. Syntax-directed transduction. *Journal of the ACM*, 15:465–488, 1968.

[8] I. Dan Melamed. Multitext grammars and synchronous parsers. In *Proceedings of HLT-NAACL 2003*, pages 79–86, 2003.

[9] Giorgio Satta and Enoch Peserico. Some computational complexity results for synchronous context-free grammars. In *Proceedings of HLT/EMNLP 2005*, pages 803–810, 2005.

[10] Yves Schabes. *Mathematical and Computational Aspects of Lexicalized Grammars*. PhD thesis, University of Pennsylvania, 1990.

[11] Stuart M. Shieber. Restricting the weak generative capacity of synchronous tree-adjoining grammars. *Computational Intelligence*, 10(4):371–385, 1994. Special Issue on Tree Adjoining Grammars.

[12] Stuart M. Shieber and Yves Schabes. Synchronous tree-adjoining grammars. In *Proceedings of the Thirteenth International Conference on Computational Linguistics (COLING)*, volume 3, pages 1–6, 1990.

[13] Stuart M. Shieber, Yves Schabes, and Fernando C. N. Pereira. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24:3–36, 1995.

[14] Dekai Wu. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23:377–404, 1997.

[15] Kenji Yamada and Kevin Knight. A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting of the ACL*, pages 523–530, 2001.