Chapter 15

Synchronous CFGs

Synchronous context-free grammars are a generalization of CFGs that generate pairs of related strings instead of single strings. 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 (Aho and Ullman, 1969). In natural language processing, they have been used for machine translation (Wu, 1997; Yamada and Knight, 2001; Chiang, 2005) and (less commonly, perhaps) semantic interpretation.

The term *synchronous CFG* is recent and far from universal. They were originally known as *syntax-directed transduction grammars* (Lewis and Stearns, 1968) or *syntax-directed translation schemata* (Aho and Ullman, 1969). *Inversion transduction grammars* (Wu, 1997) are a special case of synchronous CFGs.

15.1 Motivation

Earlier we used FSTs to perform many kinds of string transformations, but there are many kinds of transformations that they can't perform. Even something as simple as reversing the input string is beyond FST's power.

Question 27. Informally, what is the reason that there is no FST that can output the reverse of its input string?

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

(15.1) the boy stated that the student said that the teacher danced

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

This kind of reordering is beyond the power of FSTs, but a synchronous CFG can do this.

15.2 Definition

In a synchronous CFG, the productions have two right-hand sides—call them the *input rhs* and the *output rhs*—that are related in a certain way. Below is an example synchronous CFG for a fragment of English

and Japanese:

$$\text{S} \rightarrow \langle \text{NP}\boxed{1}\ \text{VP}\boxed{2}, \text{NP}\boxed{1}\ \text{VP}\boxed{2} \rangle \tag{15.2}$$

$$\text{VP} \rightarrow \langle \text{VB}\boxed{1}\ \text{NP}\boxed{2}, \text{NP}\boxed{2}\ \text{VB}\boxed{1} \rangle \tag{15.3}$$

$$\text{NP} \rightarrow \langle \text{I}, \text{watashi wa} \rangle \tag{15.4}$$

$$\text{NP} \rightarrow \langle \text{the box}, \text{hako wo} \rangle \tag{15.5}$$

$$\text{VB} \rightarrow \langle \text{open}, \text{akemasu} \rangle \tag{15.6}$$

The boxed numbers $\boxed{i}$ link up nonterminal symbols in the input rhs with nonterminal symbols in the output rhs: $\boxed{1}$ links to $\boxed{1}$, $\boxed{2}$ with $\boxed{2}$, and so on. Linked nonterminals must match (X with X), and the linking must be a one-to-one correspondence.

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 (15.2), 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 (15.3) like so:

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

But now if we want to apply production (15.4), we can't apply it to $\text{NP}\boxed{11}$ on one side and $\text{NP}\boxed{14}$ on the other, like this:

$$\not\Rightarrow \langle \text{I VB}\boxed{13}\ \text{NP}\boxed{14}, \text{NP}\boxed{11}\ \text{watashi wa VB}\boxed{13} \rangle$$

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

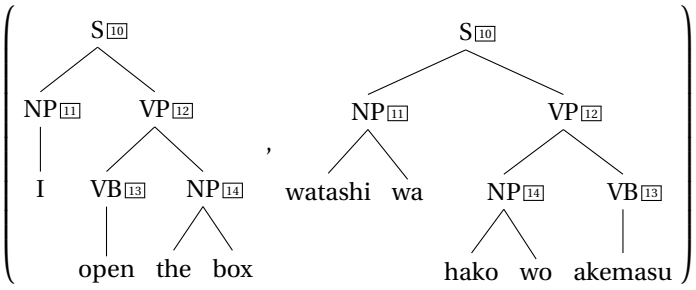
$$\Rightarrow \langle \text{I VB}\boxed{13}\ \text{NP}\boxed{14}, \text{watashi wa NP}\boxed{14}\ \text{VB}\boxed{13} \rangle$$

$$\Rightarrow \langle \text{I open NP}\boxed{14}, \text{watashi wa NP}\boxed{14}\ \text{akemasu} \rangle$$

$$\Rightarrow \langle \text{I open the box}, \text{watashi wa hako wo akemasu} \rangle$$

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:



(15.7)

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

Here is a more complicated example, a grammar deriving sentence pair (15.1):

$$\begin{aligned}
\text{S} &\rightarrow \langle \text{NP}\boxed{1}\ \text{VP}\boxed{2}, \text{NP}\boxed{1}\ \text{VP}\boxed{2} \rangle \\
\text{VP} &\rightarrow \langle \text{VB}\boxed{1}, \text{VB}\boxed{1} \rangle \\
\text{VP} &\rightarrow \langle \text{VB}\boxed{1}\ \text{SBAR}\boxed{2}, \text{SBAR}\boxed{2}\ \text{VB}\boxed{1} \rangle \\
\text{SBAR} &\rightarrow \langle \text{IN}\boxed{1}\ \text{S}\boxed{2}, \text{S}\boxed{2}\ \text{IN}\boxed{1} \rangle \\
\text{IN} &\rightarrow \langle \text{that}, \text{to} \rangle \\
\text{NP} &\rightarrow \langle \text{the boy}, \text{shoonen-ga} \rangle \\
\text{NP} &\rightarrow \langle \text{the student}, \text{gakusei-ga} \rangle \\
\text{NP} &\rightarrow \langle \text{the teacher}, \text{sensei-ga} \rangle \\
\text{VB} &\rightarrow \langle \text{danced}, \text{odotta} \rangle \\
\text{VB} &\rightarrow \langle \text{said}, \text{itta} \rangle \\
\text{VB} &\rightarrow \langle \text{stated}, \text{hanasita} \rangle
\end{aligned} \tag{15.8}$$

Question 28. Show how to derive sentence pair (15.1).

Question 29. How would you write a synchronous CFG that outputs the reverse of its input string?

We can add states to a synchronous CFG just like we did for CFGs. We introduce the following notational convention: if β is a string of terminals and r indexed nonterminals, then $\beta[q_1, \ldots, q_r]$ adds state q_i to the nonterminal indexed by i.

15.3 Weighted synchronous CFGs

In a *weighted synchronous CFG*, a weight is attached to each production. The weight of a whole derivation is just the product of the weights of the productions used in the derivation. Thus a weighted synchronous CFG generates a weighted set of pairs of derivations.

We can (but don't necessarily have to) think of a weighted synchronous CFG as a stochastic process, in at least two different ways. First, by analogy with PCFG, each production $A \rightarrow \langle \alpha, \alpha' \rangle$ could have probability $P(\alpha, \alpha' \mid A)$, so that the probability of the whole derivation is the joint probability of the input and output trees. Or, by analogy with FSTs, each production could have probability $P(\alpha' \mid A, \alpha)$, in which case the probability of the whole derivation would be the conditional probability of the output tree given the input tree.

For example, suppose we want to translate French into English. We could learn a weighted synchronous CFG that translates from English trees T_e into French trees T_f and computes $P(T_f \mid T_e)$. Then we would like to combine this grammar with a language model $P(T_e)$ so that we can compute:

$$\arg\max_{T_e} P(T_e \mid T_f) = \arg\max_{T_e} P(T_f, T_e) \tag{15.9}$$

$$= \arg\max_{T_e} P(T_f \mid T_e) P(T_e) \tag{15.10}$$

But suppose we are given only a French string, not a French tree, as input, and want only an English

string, not an English tree, as output. Then:

$$\arg\max_e P(e \mid f) = \arg\max_e P(f, e) \tag{15.11}$$

$$= \arg\max_e \sum_{T_f} \sum_{T_e} P(T_f, T_e) \tag{15.12}$$

$$= \arg\max_e \sum_{T_f} \sum_{T_e} P(T_f \mid T_e) P(T_e) \tag{15.13}$$

The summations are over trees that yield f and e, respectively. It is tractable to compute both summations for a fixed f, e, but maximizing over e is intractable. Therefore the standard practice is to use the **Viterbi approximation**:

$$= \arg\max_{T_f, T_e} P(T_f \mid T_e) P(T_e) \tag{15.14}$$

where again the maximization over T_f and T_e is over trees that yield f and e, respectively. We will see how to do this maximization below.

This $P(T_e)$ could be modeled by a PCFG. However, n-gram language models continue to be far more effective than tree language models. So it is common to use an n-gram language model instead, even though it is a string language model, and assume that $P(T_e) \approx P(e)$.

15.4 Binarization

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

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

can be binarized into

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

Note that we did not sever any links in doing so.

Question 30. Show that any synchronous CFG production of rank three can be converted into a set of productions of rank two. Assume that the production doesn't have any terminal symbols.

But there are synchronous CFGs of rank four that can't be binarized—namely, any synchronous CFG containing a production with one of the following forms:

$$\begin{aligned} A &\to \langle B\boxed{1}\ C\boxed{2}\ D\boxed{3}\ E\boxed{4}, D\boxed{3}\ B\boxed{1}\ E\boxed{4}\ C\boxed{2} \rangle \\ A &\to \langle B\boxed{1}\ C\boxed{2}\ D\boxed{3}\ E\boxed{4}, C\boxed{2}\ E\boxed{4}\ B\boxed{1}\ D\boxed{3} \rangle \end{aligned} \tag{15.15}$$

Question 31. Show that the first form can't be binarized in such a way that preserves all the links.

In general, let r-SCFG stand for the set of string relations generated by synchronous CFGs of rank r. Then:

$$1\text{-SCFG} \subsetneq 2\text{-SCFG} = 3\text{-SCFG} \subsetneq 4\text{-SCFG} \subsetneq \dots \tag{15.16}$$

despite the fact that non-synchronous CFGs of rank 2 and higher are all weakly equivalent (Aho and Ullman, 1969). There is an efficient algorithm for minimizing the rank of a synchronous CFG (Zhang and Gildea, 2007).

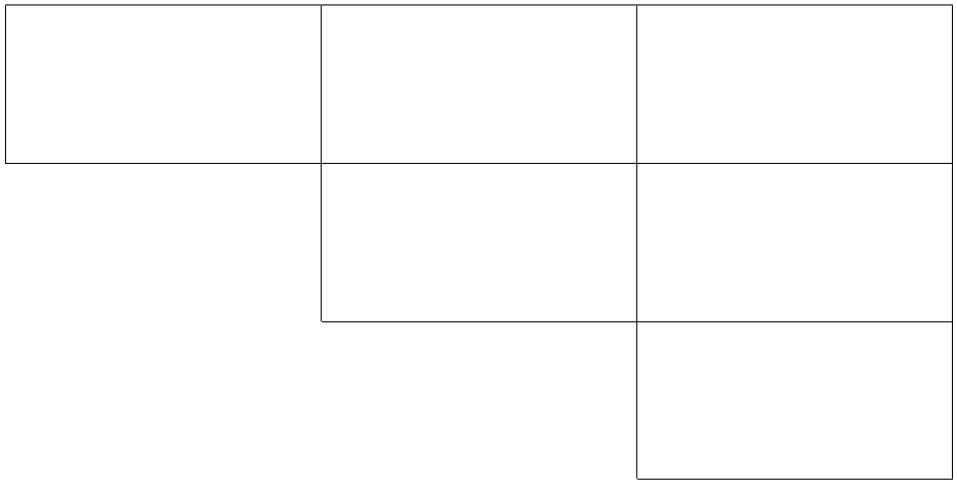
15.5 Translation

Translation, analogous to application of FSTs, is the problem of finding all possible output strings for a given input string. If the input side of the grammar is in Chomsky normal form, we can use a simple variant of CKY to do translation.

Consider the following synchronous CFG:

$$\begin{aligned}
\text{S} &\xrightarrow{1} \langle \text{NP}\boxed{1}\ \text{VP}\boxed{2}, \text{NP}\boxed{1}\ \text{VP}\boxed{2} \rangle \\
\text{VP} &\xrightarrow{1} \langle \text{VB}\boxed{1}\ \text{NP}\boxed{2}, \text{NP}\boxed{2}\ \text{VB}\boxed{1} \rangle \\
\text{VB} &\xrightarrow{1} \langle \text{see}, \text{veo} \rangle \\
\text{VB} &\xrightarrow{1} \langle \text{love}, \text{amo} \rangle \\
\text{NP} &\xrightarrow{0.3} \langle \text{I}, \text{yo} \rangle \\
\text{NP} &\xrightarrow{0.7} \langle \text{I}, \epsilon \rangle \\
\text{NP} &\xrightarrow{0.1} \langle \text{you}, \text{te} \rangle \\
\text{NP} &\xrightarrow{0.9} \langle \text{you}, \text{la} \rangle \\
\text{NP} &\xrightarrow{1} \langle \text{her}, \text{la} \rangle
\end{aligned} \tag{15.17}$$

Question 32. Run the Viterbi CKY algorithm (just probabilities, no back-pointers) using the *input* side of the grammar on the input string *I see her*.



Remember that in CFG parsing, we represented back-pointers as CFG rules, augmenting nonterminal symbols with spans i, j to specify which cells a pointer points to. Similarly, in synchronous CFG translation, we represent back-pointers as synchronous CFG rules.

Question 33. Using the same grammar and input string as before, write down the back-pointers for the two missing cells:

$\text{NP} : \text{NP}_{0,1} \rightarrow \langle \text{I}, \epsilon \rangle$	$\emptyset$	
	$\text{VB} : \text{VB}_{1,2} \rightarrow \langle \text{see}, \text{veo} \rangle$	$\text{VP} : \text{VP}_{1,3} \rightarrow \langle \text{VB}_{1,2}\ \boxed{1}\ \text{NP}_{2,3}\ \boxed{2}, \text{NP}_{2,3}\ \boxed{2}\ \text{VB}_{1,2}\ \boxed{1} \rangle$

What is the best translation?

If we maintain *sets* of back-pointers instead of just the best back-pointer, then we get a representation of all possible translations of the input string.

Question 34. Using the same grammar and input string as before, write down the sets of all possible back-pointers for the two missing cells:

$\text{NP}: \left\{ \begin{array}{l} \text{NP}_{0,1} \rightarrow \langle \text{I}, \text{yo} \rangle \\ \text{NP}_{0,1} \rightarrow \langle \text{I}, \epsilon \rangle \end{array} \right\}$	$\emptyset$	
	$\text{VB}: \{\text{VB}_{1,2} \rightarrow \langle \text{see}, \text{veo} \rangle\}$	$\text{VP}: \{\text{VP}_{1,3} \rightarrow \langle \text{VB}_{1,2}\,\boxed{1}\,\text{NP}_{2,3}\,\boxed{2}, \text{NP}_{2,3}\,\boxed{2}\,\text{VB}_{1,2}\,\boxed{1} \rangle\}$

How many translations are there, and what are they?

15.6 Composition (optional)

Recall that finite-state transducers are composable: if R and R' are definable by finite-state transducers, then so is $R \circ R'$. Is this true for synchronous CFGs too?

We can think of a synchronous CFG as defining a relation on trees or a relation on strings. If we think of synchronous CFGs as defining tree relations, they are composable just like FSTs. This is what we would need, for example, when combining a synchronous CFG translation model with a CFG language model. The composition construction is quite similar to that for FSTs, but we don't discuss it here.

However, if we think of synchronous CFGs as string relations, they behave very differently: if R and R' are string relations definable by synchronous CFGs, then $R \circ R'$ will not necessarily be.

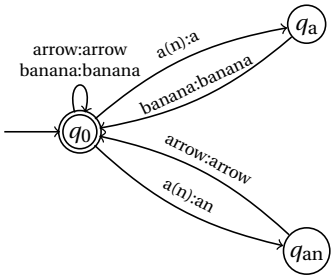
Question 35. Give an example of two synchronous CFGs that define string relations that compose to form a string relation that can't be defined by a synchronous CFG. Hint: Remember that $\{a^i b^j c^j\}$ and $\{a^j b^j c^k\}$ are context-free languages but intersect to form a non-context-free language.

But just as it was possible to intersect a CFG with a FSA, it is possible to compose a synchronous CFG with a FST, or vice-versa. The construction to do this is similar to intersecting a (non-synchronous) CFG with a FSA.

Consider the following synchronous CFG G that translates from French to English, but translates the French determiner *une* using a pseudoword *a(n)*, and a FST M that changes *a(n)* into *a* or *an* as appropriate.

$$\begin{aligned} &\text{NP} \rightarrow \langle \text{DT}\,\boxed{1}\,\text{NN}\,\boxed{2}, \text{DT}\,\boxed{1}\,\text{NN}\,\boxed{2} \rangle \\ &\text{DT} \rightarrow \langle \text{une}, \text{a(n)} \rangle \\ &\text{NN} \rightarrow \langle \text{flèche}, \text{arrow} \rangle \\ &\text{NN} \rightarrow \langle \text{banane}, \text{banana} \rangle \end{aligned} \tag{15.18}$$

(15.19)



We want to compose $M \circ G$, that is, to feed the output of G into the input of M. Assume that M has no input ϵ-transitions. As in intersection, we are going to annotate the nonterminal symbols of G to simulate the action of M. Each nonterminal gets a pair of states indicating where M could be before and after reading the *output* yield of the symbol.

We consider each rule of the grammar one by one. First, consider the production $\text{DT} \rightarrow \langle \text{une}, \text{a(n)} \rangle$. Suppose that M is in state q_0. Upon reading the output yield of this rule, it could either output *a* and move to state q_{a}, or output *an* and move to state q_{an}. So we make two new rules,

$$\text{DT}_{q_0,q_{\text{a}}} \rightarrow \langle \text{une}, \text{a} \rangle$$
$$\text{DT}_{q_0,q_{\text{an}}} \rightarrow \langle \text{une}, \text{an} \rangle$$

But what if M starts in state q_{a} or state q_{an}? In that case, M would reject the string, so we don't make any new rules for these cases. By similar reasoning, we make rules

$$\text{NN}_{q_0,q_0} \rightarrow \langle \text{flèche}, \text{arrow} \rangle$$
$$\text{NN}_{q_{\text{an}},q_0} \rightarrow \langle \text{flèche}, \text{arrow} \rangle$$
$$\text{NN}_{q_0,q_0} \rightarrow \langle \text{banane}, \text{banana} \rangle$$
$$\text{NN}_{q_{\text{a}},q_0} \rightarrow \langle \text{banane}, \text{banana} \rangle$$

Now consider the production $\text{NP} \rightarrow \text{DT NN}$. Imagine that M is in state q_0 and reads the yield of DT. What state will it be in after? Based on the annotated nonterminals created so far, it could either move to state q_{a} or q_{an}. In either case, upon reading NN, it will end up back in state q_0. So we make two rules,

$$\text{NP}_{q_0,q_0} \rightarrow \text{DT}_{q_0,q_{\text{a}}}\ \text{NN}_{q_{\text{a}},q_0}$$
$$\text{NP}_{q_0,q_0} \rightarrow \text{DT}_{q_0,q_{\text{an}}}\ \text{NN}_{q_{\text{an}},q_0}$$

As before, we loop through all the rules repeatedly until the set of annotated nonterminals stops growing. In this case, we are done.

Finally, make a new start symbol S', and write rules connecting S' to the start symbol of G and the initial/final states of M (one for each final state):

$$S' \rightarrow \langle \text{NP}_{q_0,q_0}\,\boxed{1}, \text{NP}_{q_0,q_0}\,\boxed{1} \rangle \tag{15.20}$$

Composing $G \circ M$, that is, feeding the output of M into the input of G, is analogous. In that case we would assume that M has no output ϵ-transitions.

Just as intersection gave us a more abstract way of thinking about parsing, so composition gives us a more abstract way of thinking about translation. We saw how we can use CKY to build a synchronous CFG that represents all possible translations of an input string. This is none other than the composition of the

input string and the grammar. That is, given an input string f, form the trivial FST M_f that maps f to itself and nothing else, and compose $G \circ M_f$. If G is rank-two, then the complexity of translation is exactly the same as the complexity of parsing. If G has rank greater than two, then translation-as-composition will be slower than parsing.

15.7 Translation with a language model (optional)

Suppose we want to do translation with a weighted synchronous CFG but also want to use a g-gram language model, such that the score of a derivation is the score according to the synchronous CFG multiplied by the language model score. We can do this using the tools we have established so far. Let M_f be as before, and let M_{LM} be our g-gram language model, represented as a FST. Then we form the composition $M_{\text{LM}} \circ G \circ M_f$.

Thus, each nonterminal A gets annotated to become $A^{i,j}_{q,r}$, where:

- i, j are positions of f, resulting from composition with M_f, and
- q, r are states of M_{LM}, which each encapsulates a $(g-1)$-gram context of output symbols.

Usually this means that A generates the input span $f_{i+1} \cdots f_j$ with some output translation, and q encapsulates the $(g-1)$ words *preceding* the translation and r encapsulates the last $(g-1)$ words of the translation. If we are parsing bottom-up, this means that we start by translating individual words of f and then gradually build up larger and larger translations of spans of f. But for each input span that we translate, we constantly have to guess all possible values of q, that is, all the possible $(g-1)$-gram contexts that might precede the translation.

In practice, this is not going to be very efficient. What most MT systems use is another composition algorithm (Wu, 1996) that does not require so much guesswork. Instead of annotating each nonterminal $A^{i,j}$ with a pair of states q, r, it annotates it with the first $(g-1)$ and last $(g-1)$ output translations of $A^{i,j}$. Most systems also use beam search, and an algorithm called **cube pruning** (Chiang, 2007) that further speeds up the beam search, again exploiting the structure of the language model.

Bibliography

Aho, A. V. and J. D. Ullman (1969). "Syntax Directed Translations and the Pushdown Assembler". In: *J. Comp. Sys. Sci.* 3, pp. 37–56.

Chiang, David (2005). "A Hierarchical Phrase-Based Model for Statistical Machine Translation". In: *Proc. ACL*, pp. 263–270.

— (2007). "Hierarchical Phrase-Based Translation". In: *Computational Linguistics* 33.2, pp. 201–228.

Lewis P. M., II and R. E. Stearns (1968). "Syntax-Directed Transduction". In: *Journal of the ACM* 15, pp. 465–488.

Satta, Giorgio and Enoch Peserico (2005). "Some Computational Complexity Results for Synchronous Context-Free Grammars". In: *Proc. HLT-EMNLP*, pp. 803–810.

Wu, Dekai (1996). "A Polynomial-Time Algorithm for Statistical Machine Translation". In: *Proc. ACL*, pp. 152–158.

— (1997). "Stochastic Inversion Transduction Grammars and Bilingual Parsing of Parallel Corpora". In: *Computational Linguistics* 23, pp. 377–404.

Yamada, Kenji and Kevin Knight (2001). "A Syntax-based Statistical Translation Model". In: *Proc. ACL*, pp. 523–530.

Zhang, Hao and Daniel Gildea (2007). "Factorization of Synchronous Context-Free Grammars in Linear Time". In: *Workshop on Syntax and Structure in Statistical Translation*.