An Introduction to Synchronous Grammars

David Chiang*

21 June 2006

1 Introduction

Synchronous context-free grammars are a generalization of context-free grammars (CFGs) that generate pairs of related strings instead of single strings. 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 [1]. In natural language processing, they have been used for machine translation [19, 20, 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* [10] 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 [11], and inversion transduction grammars [19] 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:

(3) $\mathrm{S} \rightarrow \langle \mathrm{NP}\boxed{1}\ \mathrm{VP}\boxed{2}, \mathrm{NP}\boxed{1}\ \mathrm{VP}\boxed{2} \rangle$

(4) $\mathrm{VP} \rightarrow \langle \mathrm{V}\boxed{1}\ \mathrm{NP}\boxed{2}, \mathrm{NP}\boxed{2}\ \mathrm{V}\boxed{1} \rangle$

(5) $\mathrm{NP} \rightarrow \langle \text{i}, \text{watashi wa} \rangle$

(6) $\mathrm{NP} \rightarrow \langle \text{the box}, \text{hako wo} \rangle$

(7) $\mathrm{V} \rightarrow \langle \text{open}, \text{akemasu} \rangle$

*Thanks to Philip Resnik, Anoop Sarkar, Kevin Knight, and Liang Huang for their helpful feedback.

The boxed numbers [1] link up nonterminal symbols on the source side with nonterminal symbols on the target side: [1] links to [1], [2] with [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),

$$\langle \text{S}[10], \text{S}[10] \rangle$$

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

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

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

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

But now if we want to apply production (5), we can't apply it to NP[11] on one side and NP[14] on the other, like this:

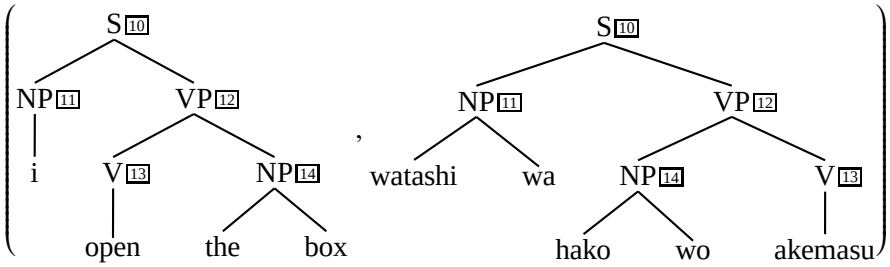
$$\Rightarrow \langle \text{i V}[13]\ \text{NP}[14], \text{NP}[11]\ \text{watashi wa V}[13] \rangle \qquad \textbf{not allowed}$$

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

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

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:



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

Here is a more complicated example, as promised:

(8) $\mathrm{S} \rightarrow \langle \mathrm{NP}\boxed{1}\ \mathrm{VP}\boxed{2}, \mathrm{NP}\boxed{1}\ \mathrm{VP}\boxed{2} \rangle$

(9) $\mathrm{VP} \rightarrow \langle \mathrm{V}\boxed{1}, \mathrm{V}\boxed{1} \rangle$

(10) $\mathrm{VP} \rightarrow \langle \mathrm{V}\boxed{1}\ \bar{\mathrm{S}}\boxed{2}, \bar{\mathrm{S}}\boxed{2}\ \mathrm{V}\boxed{1} \rangle$

(11) $\bar{\mathrm{S}} \rightarrow \langle \mathrm{Comp}\boxed{1}\ \mathrm{S}\boxed{2}, \mathrm{S}\boxed{2}\ \mathrm{Comp}\boxed{1} \rangle$

(12) $\mathrm{Comp} \rightarrow \langle \text{that}, \text{to} \rangle$

(13) $\mathrm{NP} \rightarrow \langle \text{the boy}, \text{shoonen-ga} \rangle$

(14) $\mathrm{NP} \rightarrow \langle \text{the student}, \text{gakusei-ga} \rangle$

(15) $\mathrm{NP} \rightarrow \langle \text{the teacher}, \text{sensei-ga} \rangle$

(16) $\mathrm{V} \rightarrow \langle \text{danced}, \text{odotta} \rangle$

(17) $\mathrm{V} \rightarrow \langle \text{said}, \text{itta} \rangle$

(18) $\mathrm{V} \rightarrow \langle \text{stated}, \text{hanasita} \rangle$

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

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 source and target trees. Or, 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 target tree given the source tree.

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.

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.

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:

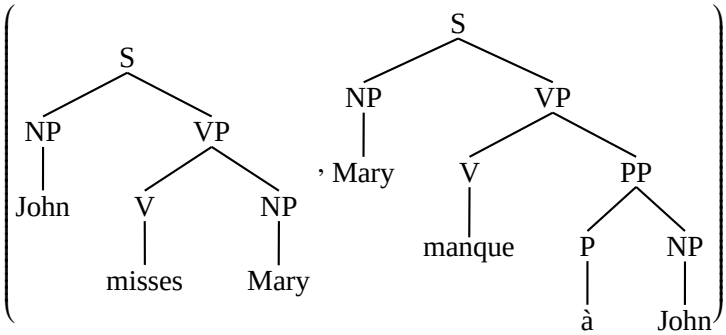
(19) $$\mathrm{A} \rightarrow \langle \mathrm{B}\boxed{1}\ \mathrm{C}\boxed{2}\ \mathrm{D}\boxed{3}\ \mathrm{E}\boxed{4}, \mathrm{D}\boxed{3}\ \mathrm{B}\boxed{1}\ \mathrm{E}\boxed{4}\ \mathrm{C}\boxed{2} \rangle$$

Try to binarize it—no matter how you do it, you will always have to sever a link, which is not allowed. In general, let *r*-SCFG stand for the set of string relations generated by synchronous CFGs of rank *r*; then:

(20) $$\text{1-SCFG} \subsetneq \text{2-SCFG} = \text{3-SCFG} \subsetneq \text{4-SCFG} \subsetneq \ldots$$

despite the fact that non-synchronous CFGs of rank 2 and higher are all weakly equivalent [1]. There is an efficient algorithm for minimizing the rank of a synchronous CFG [6].

No raising or lowering. The properties above considered synchronous CFGs as defining relations on strings. If you think of a synchronous CFG as defining a relation on trees, then all synchronous CFGs can do is relabel nodes and reorder sisters. So if you want to write a synchronous CFG that can swap subjects and objects, as in the following English and French trees:



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

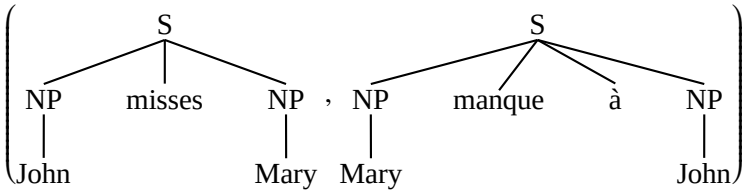
However, if you're willing to mangle the trees, then there is an easy solution:

(21) $$\mathrm{S} \rightarrow \langle \mathrm{NP}\boxed{1}\ \text{misses}\ \mathrm{NP}\boxed{2}, \mathrm{NP}\boxed{2}\ \text{manque à}\ \mathrm{NP}\boxed{1} \rangle$$

(22) $$\mathrm{NP} \rightarrow \langle \text{John}, \text{John} \rangle$$

(23) $$\mathrm{NP} \rightarrow \langle \text{Mary}, \text{Mary} \rangle$$

which will generate the tree pair



By flattening the trees, we made [$_{\mathrm{NP}}$ John] and [$_{\mathrm{NP}}$ 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.)

Synchronous tree-substitution grammar (Section 5.1) would be able to do this without flattening.

4 Algorithms

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.

Following Shieber et al. [18], we present each parsing algorithm as a deductive system, which consists of a set of *axioms* that are taken as the items that are trivially provable, and a set of *inference rules* that say what items are provable from what other items (if the top items are provable, then the bottom one is), and a designated *goal* item that, if proven, means that the string belongs to the language. The algorithms we present are simple enough to run as follows: start with the set of axioms, and then go through all the possible items in a predefined order (which will be given), trying to prove each one from the items that have been proven so far. Then, if the goal item was proved, accept the string; otherwise, reject it.

Thus the standard CKY algorithm for non-synchronous CFGs in Chomsky normal form would be:

$$\text{Axioms} \qquad \frac{}{A \to \alpha} \qquad \text{for all } (A \to \alpha) \in G$$

$$\text{Inference rules} \qquad \frac{A \to w_{i+1}}{[A, i, i+1]}$$

$$\frac{[B, i, j] \quad [C, j, k] \quad A \to BC}{[A, i, k]}$$

$$\text{Goal} \qquad [S, 0, n]$$

where the item $[A, i, j]$ means that we have recognized an A spanning words $i + 1$ through j (inclusive) of the input string.

Whenever we prove an item, we store inside the item a list of the tuples of items that it can be proved from. In this way, once the goal is proved, we can go back and reconstruct its whole proof, and from that, the parse tree. Usually there will be many possible proofs: in this case, we can reconstruct whichever proof we like, but very often we will use the probabilities to reconstruct the single best or k best proofs.

A variety of search algorithms can be used to find a proof of the goal, but if we want to calculate probabilities of subderivations online (that is, during parsing), because, for example, we need them for pruning, then certain conditions have to be met. The simplest thing to do is to follow the traditional CKY algorithm: for each $1 \le \ell \le n$ in increasing order, try to prove all items of the form $[A, i, j]$ where $j - i = \ell$.

4.1 Translation

Translation is a totally straightforward generalization of monolingual parsing because every valid derivation of the source-side grammar leads to one or more valid translations. So we just modify our CKY algorithm to have different axioms:

$$\text{Axioms} \quad \frac{}{A \to \alpha} \quad \text{for all } (A \to \langle \alpha, \alpha' \rangle) \in G$$

That's it! This algorithm just hides the target sides of the rules right at the beginning and then parses using the source-side grammar. When we reconstruct the derivations, we will find the target sides of the rules again, and use that information to reconstruct the translations.

That is assuming, of course, that the source-side grammar is in Chomsky normal form. Remember that not every synchronous CFG can be put into this form. For a general synchronous CFG, we must use a parsing algorithm that can operate on general CFGs directly. The simplest such algorithm is probably a bottom-up Earley algorithm:

Axioms $$\frac{}{[A \to \bullet\alpha, i, i]} \qquad \text{for all } 0 \le i \le n \text{ and } (A \to \langle \alpha, \alpha' \rangle) \in G$$

Inference rules $$\frac{[A \to \alpha \bullet w_{j+1}\beta, i, j]}{[A \to \alpha w_{j+1} \bullet \beta, i, j+1]}$$

$$\frac{[A \to \alpha \bullet B\beta, i, j] \quad [B \to \gamma \bullet, j, k]}{[A \to \alpha B \bullet \beta, i, k]}$$

Goal $$[S \to \alpha \bullet, 0, n]$$

where the item $[A \to \alpha \bullet \beta, i, j]$ means that we have recognized the prefix of A up to the dot ($\bullet$) spanning words $i+1$ through j (inclusive) of the input string.

Unfortunately, now the CKY-style method described above of proving items in order of increasing $j - i$ will not work, but a more general chart parsing method will. See Shieber et al. [18] for more details.

But the key idea here is that once again the target sides of the rules are hidden away at the beginning, and parsing proceeds as usual with the source-side grammar. This particular parser effectively binarizes the grammar on the fly, and even if the synchronous grammar is not binarizable, this will work.

4.2 Bitext parsing

Now we turn to the problem of bitext parsing, which is to decide whether a synchronous grammar accepts or rejects a given *pair* of input strings. This is typically used when training some translation models: for example, using the Inside-Outside algorithm to estimate probabilities from a parallel text.

On its face this problem might seem easier because we are given more information as inputs, but in fact it is considerably harder. Here, we can't ignore the target-side rules as we did above, and have to confront the fact that general synchronous CFGs are not binarizable. Even in the case of a rank-two synchronous CFG (or equivalent), bitext parsing takes $O(n^6)$ time [19]. For a fixed synchronous CFG, bitext parsing takes $O(n^k)$ time where k depends on the rank of the synchronous CFG [14], but there is no k such that we can do bitext-parsing on an *arbitrary* synchronous CFG in $O(n^k)$ time.[1]

We present here Wu's bitext-parsing algorithm for the rank two case. Assume that each right-hand side (source or target) consists of

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

[1] Unless $P = NP$.

such that no production has two empty right-hand sides. Any synchronous CFG of rank two or three (or any inversion transduction grammar; see 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 $w' = w'_1 \cdots w'_{n'}$.

Axioms

$$A \to \langle \alpha, \alpha' \rangle \qquad \text{for all } A \to \langle \alpha, \alpha' \rangle \in G$$

Inference rules

$$\frac{A \to \langle w_{i+1}, w'_{i'+1} \rangle}{[A, i, i+1, i', i'+1]}$$

$$\frac{A \to \langle \epsilon, w'_{i'+1} \rangle}{[A, i, i, i', i'+1]}$$

$$\frac{A \to \langle w_{i+1}, \epsilon \rangle}{[A, i, i+1, i', i']}$$

$$\frac{[B, i, j, i', j'] \quad [C, j, k, j', k'] \quad A \to \langle B\boxed{1}\ C\boxed{2}, B\boxed{1}\ C\boxed{2} \rangle}{[A, i, k, i', k']}$$

$$\frac{[B, i, j, j', k'] \quad [C, j, k, i', j'] \quad A \to \langle B\boxed{1}\ C\boxed{2}, C\boxed{2}\ B\boxed{1} \rangle}{[A, i, k, i', k']}$$

Goal

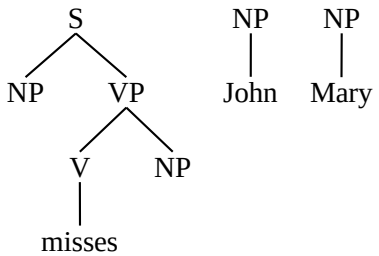
$$[S, 0, n, 0, n']$$

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. The items should be proved in order of increasing $j - i + j' - i'$.

5 Extensions

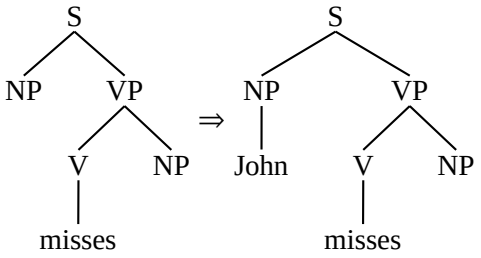
5.1 Synchronous tree-substitution grammar

Synchronous tree-substitution grammars [15, 4] are able to perform subject-object swapping as described above without flattening trees. In a (non-synchronous) tree-substitution grammar (TSG), the productions are *elementary trees*, tree fragments whose leaf nodes can be either nonterminal symbols or terminal symbols. For example:

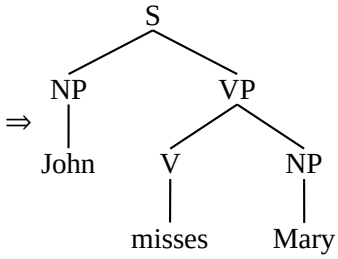


In a TSG derivation, we start with an elementary tree that is rooted in the start symbol, and then repeatedly choose a leaf nonterminal symbol *X* and attach to it an elementary tree rooted in *X*. For example:

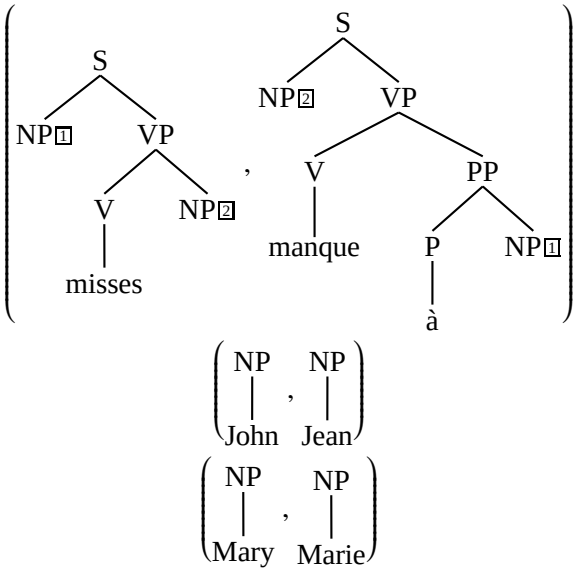
(24)



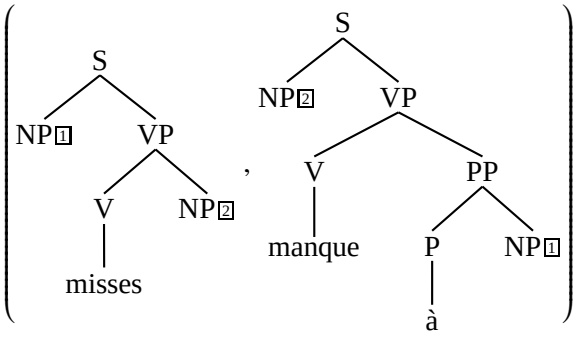
(25)



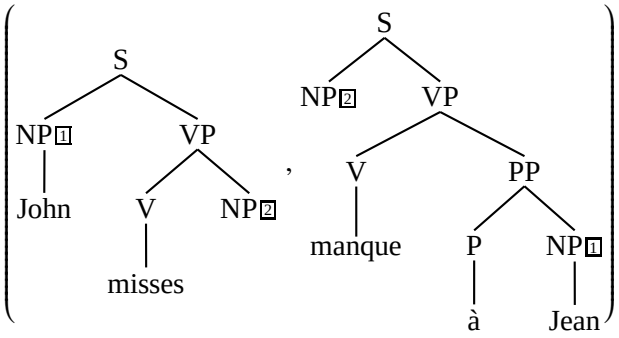
In a synchronous TSG, the productions are pairs of elementary trees, and the leaf nonterminals are linked just as in synchronous CFG:



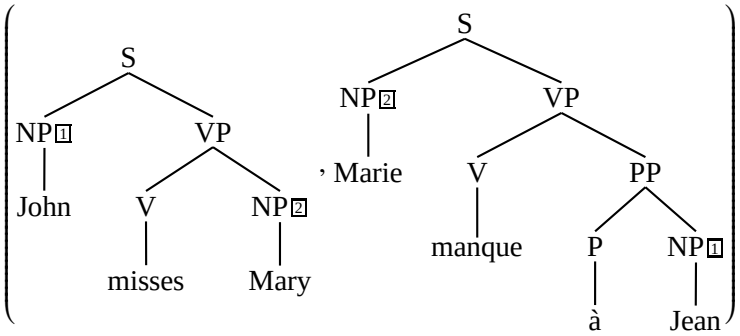
A synchronous TSG derivation starts with a pair of elementary trees and repeatedly rewrites linked nonterminals:



(26) ⇒



(27) ⇒



Thus, if we care about the trees produced, synchronous TSG is a more powerful alternative to synchronous CFG.

5.2 Synchronous tree-adjoining grammar (TAG)

Here is an example of something synchronous CFG and synchronous TSG can't do.

(28) (that) John saw Peter help the children swim

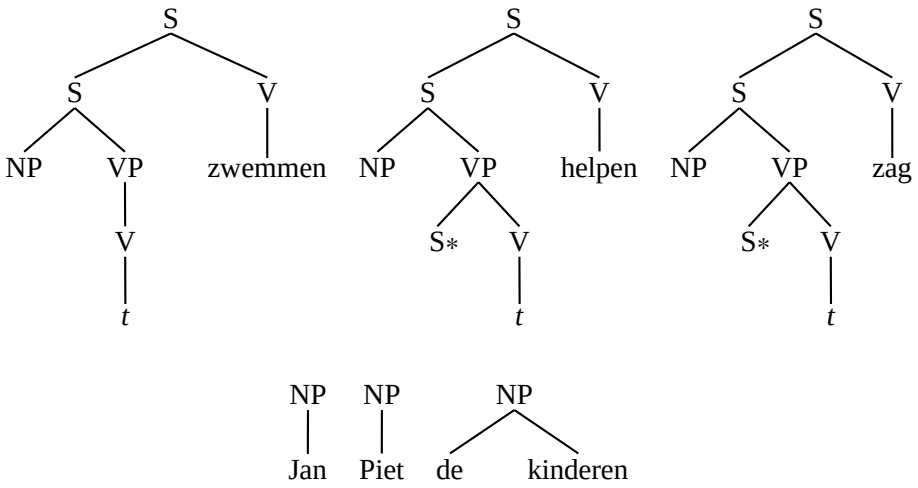
(29) (dat) Jan Piet de kinderen zag helpen zwemmen
John Peter the children saw help swim

If we assume that both languages can stack an unbounded number of clauses this way, then this translation is provably beyond the power of synchronous CFG and TSG. We need a more powerful formalism called *synchronous tree-adjoining grammar* (TAG), first defined by Shieber and Schabes [17], but note that the 1994 formulation [16] supersedes it.

In a tree-adjoining grammar [7, 8], the elementary trees come in two flavors, *initial trees*, which are just as in TSG, and *auxiliary trees*, which have a leaf node called the *foot node* marked with an asterisk (∗). Here

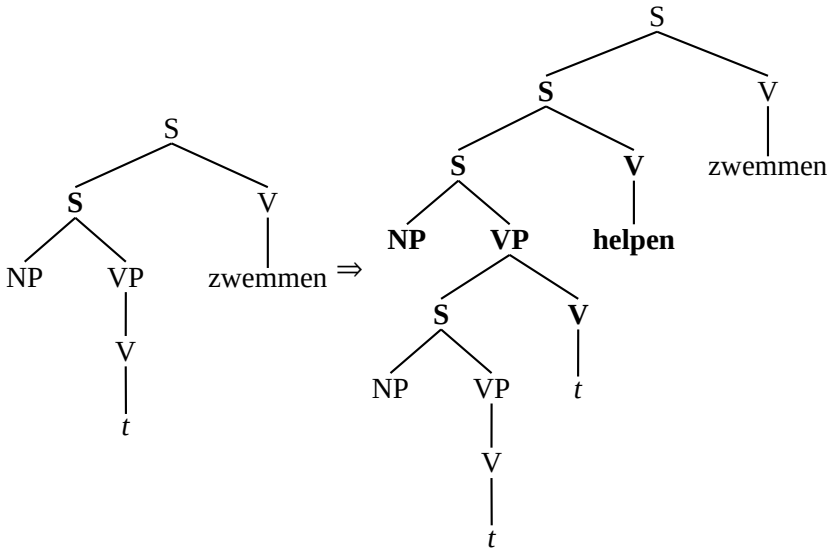
is an example TAG:

(30)



A TAG derivation again starts with an initial tree, and at each step either *substitution* or *adjunction* happens. Substitution is the same as the rewriting operation in TSG. In adjunction, any nonterminal symbol X (not necessarily a leaf node) can be rewritten with an auxiliary tree β: the X node gets split in half and β inserted in between, the root of β fusing with the top half of X and the foot of β fusing with the bottom half of X. For example (rewritten node and result of rewriting shown in bold):

(31)



The rest of the derivation of sentence (29) is left as an exercise.

Synchronous TAG generalizes TAG in exactly the same way that synchronous CFG and TSG generalize CFG and TSG: we start with a pair of initial trees and then repeatedly rewrite pairs of linked nonterminal symbols. The synchronous TAG for sentences (28) and (29) is also left as an exercise for the reader.

Exercises

1. Derive sentences (1) and (2) using the grammar (8–18).

2. Show that:

   (a) 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.

   (b) Show that production (19) can't be converted into any set of productions of rank two.

3. 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: $\{a^i b^j c^j\}$ and $\{a^j b^j c^k\}$ are context-free languages but intersect to form a non-context-free language.

The next questions concern the following synchronous CFG:

(32) $\text{S} \rightarrow \langle \text{Subj}\boxed{1}\ \text{VP}\boxed{2}, \text{Subj}\boxed{1}\ \text{VP}\boxed{2} \rangle$

(33) $\text{VP} \rightarrow \langle \text{V}\boxed{1}\ \text{Obj}\boxed{2}, \text{Obj}\boxed{2}\ \text{V}\boxed{1} \rangle$

(34) $\text{VP} \rightarrow \langle \text{V}\boxed{1}\ \text{Obj}\boxed{2}, \text{Obj}\boxed{2}\ \text{V}\boxed{1} \rangle$

(35) $\text{V} \rightarrow \langle \text{see}, \text{veo} \rangle$

(36) $\text{V} \rightarrow \langle \text{love}, \text{amo} \rangle$

(37) $\text{Subj} \rightarrow \langle \text{i}, \text{yo} \rangle$

(38) $\text{Subj} \rightarrow \langle \text{i}, \epsilon \rangle$

(39) $\text{Obj} \rightarrow \langle \text{you}, \text{te} \rangle$

(40) $\text{Obj} \rightarrow \langle \text{you}, \text{la} \rangle$

(41) $\text{Obj} \rightarrow \langle \text{her}, \text{la} \rangle$

4. Translation

   (a) Draw a parse tree for the Spanish sentence "te amo" using the Spanish side of the above grammar, and then translate the tree into English.

   (b) How many parse trees are there for the English sentence "I see you" using the English side of the above grammar? Remember to keep duplicate rules distinct!

   (c) Translate these parse trees into Spanish.

5. Bitext parsing. Use the CKY-style algorithm above to parse the sentence pair "I see her", "la veo" using the above grammar. Show the full chart, not just the successful search path.

6. Now let's add some more rules to our grammar.

   (42) $\text{VP} \rightarrow \langle \text{V}\boxed{1}\ \text{VP}\boxed{2}, \text{V}\boxed{1}\ \text{VP}\boxed{2} \rangle$

   (43) $\text{V} \rightarrow \langle \text{can}, \text{puedo} \rangle$

   (44) $\text{V} \rightarrow \langle \text{see}, \text{ver} \rangle$

   (a) Add a rule to the grammar that will allow it to correctly translate "I can see her" as "Puedo ver la."

(b) Suppose you want to add a rule to the grammar that will allow it to translate “I can see her” equally correctly as “La puedo ver.” But you want to maintain consistency with the analyses in the grammar already. What difficulty do you encounter?

7. Synchronous TAGs.

   (a) Complete the TAG derivation in (31) to yield sentence (29).

   (b) Based on what you know about synchronous CFGs and TSGs, try to turn the TAG (30) into a synchronous TAG that can generates sentence pairs like (28) and (29).

Appendix: Other notations

This is a field guide to other formalisms or variants of formalisms that are equivalent to synchronous CFG or TSG that you may encounter in the literature.

One left-hand side or two? The production

(45) $$\text{VP} \rightarrow \langle \text{V}\boxed{1}\ \text{NP}\boxed{2}, \text{NP}\boxed{2}\ \text{V}\boxed{1} \rangle$$

is also often written as

(46) $$\langle \text{VP} \rightarrow \text{V}\boxed{1}\ \text{NP}\boxed{2}, \text{VP} \rightarrow \text{NP}\boxed{2}\ \text{V}\boxed{1} \rangle$$

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

(47) $$\langle \text{A} \rightarrow \text{B}\boxed{1}\ \text{C}\boxed{2}, \text{D} \rightarrow \text{F}\boxed{2}\ \text{E}\boxed{1} \rangle$$

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:

(48) $$\text{A/D} \rightarrow \langle \text{B/E}\boxed{1}\ \text{C/F}\boxed{2}, \text{C/F}\boxed{2}\ \text{B/E}\boxed{1} \rangle$$

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.

Inversion transduction grammar [19] 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:

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

This grammar is equivalent to the synchronous CFG above (where we have fused some terminal symbols 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.

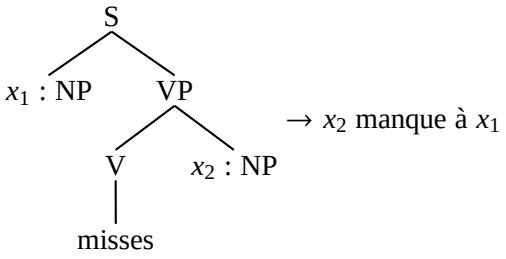
Multitext grammar [11] Multitext grammars (MTGs) are more general than synchronous CFGs because they allow not just two, but any number of right-hand sides, to specify an *n*-ary relation on strings. The two-dimensional case of multitext grammars is equivalent to synchronous CFGs. The following 2-multitext grammar rule is equivalent to our production (21):

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

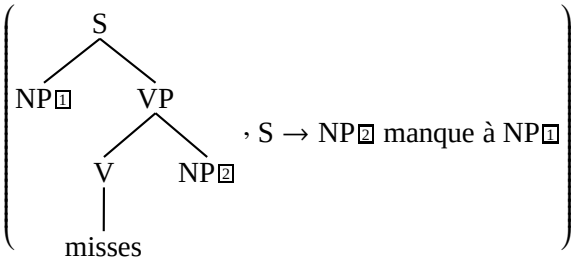
Note that the symbols inside the round parentheses don't appear in the order they will end up in; it is the numbers in square brackets, which correspond to our boxed numbers, that appear in order. Note, moreover, that MTG links terminals as well as nonterminals, but the empty string is not linked, nor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

The advantage of this notation is that you can compactly draw a synchronous derivation using a single tree. MTGs can also be extended to allow constituents to have an arbitrary number of discontiguous components [12]. The notation used in this extension, called generalized MTG, is much more closely related to the notation we have used here for synchronous CFG. Generalized MTGs also can simulate synchronous TAGs. They are equivalent to *simple range concatenation transducers* [2] and would also be equivalent to the synchronous versions of a wide range of formalisms discussed by Rambow and Satta [13].

Linear nondeleting tree transducers Two syntax-based machine translation systems at ISI, that of Yamada and Knight [20] and another based on the work of Galley et al. [5], are based on *tree-to-string transducers* [9]. The transducers they use are both *linear* and *nondeleting*, making them effectively equivalent to synchronous grammars. The Galley et al. formalism is equivalent to a synchronous grammar that has a CFG on one side and a tree-substitution grammar on the other. Here is an example rule in their notation:



This would be equivalent to:



The transducers can also allow copying (nonlinearity, or duplicate variables on the target side), deletion (missing variables on the target side), or other extensions, but this is beyond the scope of this survey.

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] Eberhard Bertsch and Mark-Jan Nederhof. On the complexity of some extensions of RCG parsing. In *Proc. Seventh International Workshop on Parsing Technologies (IWPT)*, pages 66–77, 2001.

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

[4] Jason Eisner. Learning non-isomorphic tree mappings for machine translation. In *Comp. Vol. Proc. 41st Annual Meeting of the ACL*, pages 205–208, 2003.

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

[6] Daniel Gildea, Giorgio Satta, and Hao Zhang. Factoring synchronous grammars by sorting. In *Comp. Vol. Proc. COLING-ACL 2006*, 2006.

[7] A. K. Joshi, L.S. Levy, and M. Takahashi. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10:136–163, 1975.

[8] Aravind K. Joshi and Yves Schabes. Tree-adjoining grammars. In Grzegorz Rosenberg and Arto Salomaa, editors, *Handbook of Formal Languages and Automata*, volume 3, pages 69–124. Springer-Verlag, Heidelberg, 1997.

[9] Kevin Knight and Jonathan Graehl. An overview of probabilistic tree transducers for natural language processing. In *Proc. Sixth International Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*, Lecture Notes in Computer Science. Springer-Verlag, 2005.

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

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

[12] I. Dan Melamed, Giorgio Satta, and Ben Wellington. Generalized multitext grammars. In *Proc. 42nd Annual Meeting of the ACL*, pages 661–668, 2004.

[13] Owen Rambow and Giorgio Satta. Independent parallelism in finite copying parallel rewriting systems. *Theoretical Computer Science*, 223:87–120, 1999.

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

[15] Yves Schabes. *Mathematical and Computational Aspects of Lexicalized Grammars*. PhD thesis, University of Pennsylvania, 1990. Available as technical report MS-CIS-90-48.

[16] 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.

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

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

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

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