

Beyond context-free grammars

David Chiang

11 Mar 2008

1 Generative capacity

- Weak generative capacity: Grammar's ability to decide whether or not a sentence is English
 - Grammar checking
- Strong generative capacity: Grammar's ability to assign structural descriptions to grammatical English sentences
 - Interface to semantic analysis
 - Structural description can serve as a poor man's semantics in machine translation, question answering, lexical semantics, etc.

2 Weak generative capacity of context-free grammars

Center embedding Why context-free grammars and not finite-state machines?

- OK for finite state machines:
 - (1) the white male hired another white male
 - (2) the white male hired another white male who hired another white male
 - (3) the white male hired another white male (who hired another white male)ⁿ
- But what about:
 - (4) the white male hired another white male
 - (5) the white male whom a white male hired hired another white male
 - (6) the white male (whom a white male)ⁿ hiredⁿ hired another white male
- It can be shown that $\{a^n b^n \mid n \geq 0\}$ and similar languages cannot be described by a finite-state machine

Cross-serial dependencies

- Now we move on to Dutch!
 - (7) ...dat Jan Piet de kinderen zag helpen zwemmen
...that John Peter the children saw help swim
'... that John saw Peter help the children swim.'
- Can't be generated by a CFG with the right tree structure
- Swiss-German version is more airtight: set of strings that cannot be generated by a CFG with *any* tree structure

3 Tree-adjoining grammar

Definition

- Terminal alphabet
- Nonterminal alphabet
- Initial trees: leaf nodes can have nonterminal labels, in which case they are called *substitution nodes*
- Auxiliary trees: one leaf node has the same label as the root node and is called the *foot node*

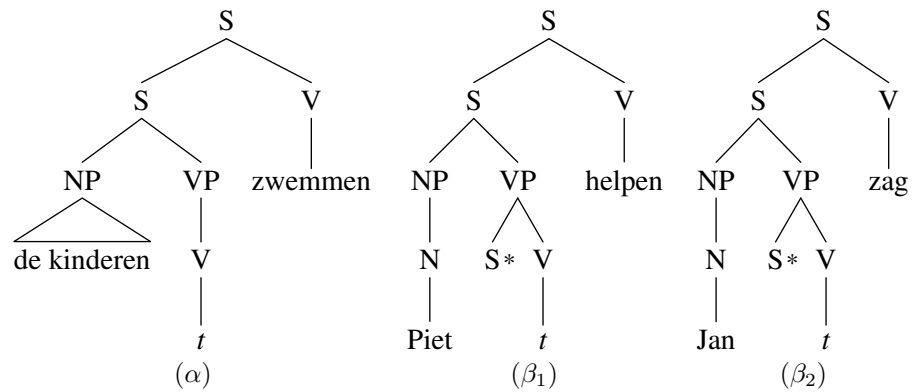
A derivation starts with an initial tree and proceeds by a series of rewriting operations, which can be either of:

- *substitution*: a substitution node labeled X is rewritten using an initial tree whose root label is X
- *adjunction*: a node labeled X is rewritten using an auxiliary tree whose root/foot label is X

Cross-serial dependencies

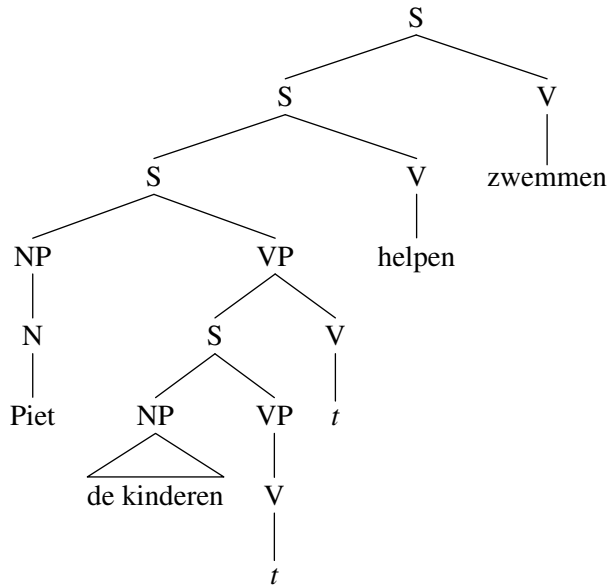
- A TAG for the Dutch example (treat the t symbols as empty strings):

(8)



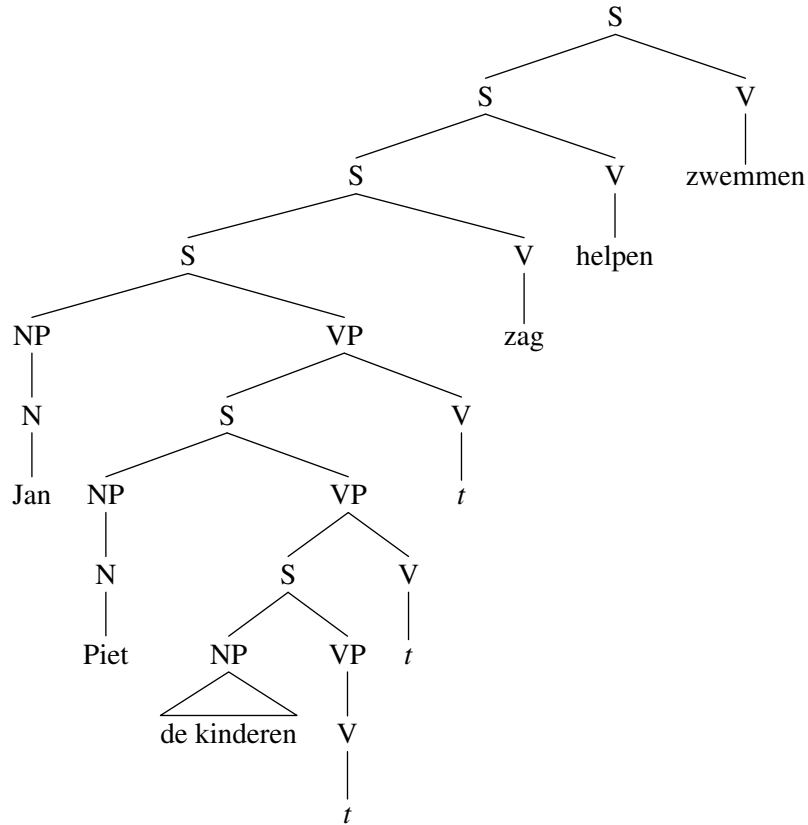
- Result of adjoining β_1 into α :

(9)



- Result of adjoining β_2 into that:

(10)



Parsing TAGs

- Algorithms for CFGs generalize to TAGs
- But wherever a CFG parser has two string positions, the TAG parser has four
- Hence, TAG parsing is $\mathcal{O}(n^6)$ time
- But there are restricted versions that bring this back down to $\mathcal{O}(n^4)$ (retaining the cross-serial dependencies) or to $\mathcal{O}(n^3)$ (weakly but not strongly equivalent to CFG)

4 Synchronous grammars

We saw already how logical forms can be built from syntactic parse trees compositionally. Here, we look at how to similarly build translations into another language, using an extension to CFGs called *synchronous CFGs*. These are similar to the tree transducers from CSCI 562.

Synchronous CFGs are a generalization of 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. In natural language processing, they have been used for machine translation 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* or *syntax-directed translation schemata*, the latter still probably being the most common name. In the NLP community they are also known as 2-multitext grammars, and inversion transduction grammars are a special case of synchronous CFGs.

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:

$$S \rightarrow \langle \text{NP}_{\boxed{1}} \text{VP}_{\boxed{2}}, \text{NP}_{\boxed{1}} \text{VP}_{\boxed{2}} \rangle \quad (11)$$

$$\text{VP} \rightarrow \langle \text{V}_{\boxed{1}} \text{NP}_{\boxed{2}}, \text{NP}_{\boxed{2}} \text{V}_{\boxed{1}} \rangle \quad (12)$$

$$\text{NP} \rightarrow \langle \text{i, watashi wa} \rangle \quad (13)$$

$$\text{NP} \rightarrow \langle \text{the box, hako wo} \rangle \quad (14)$$

$$\text{V} \rightarrow \langle \text{open, akemasu} \rangle \quad (15)$$

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 $Y \neq X$.

Synchronous CFG derivations 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 (11), 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 (12) 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 (13), we can't apply it to $\text{NP}_{\boxed{11}}$ on one side and $\text{NP}_{\boxed{14}}$ on the other, like this:

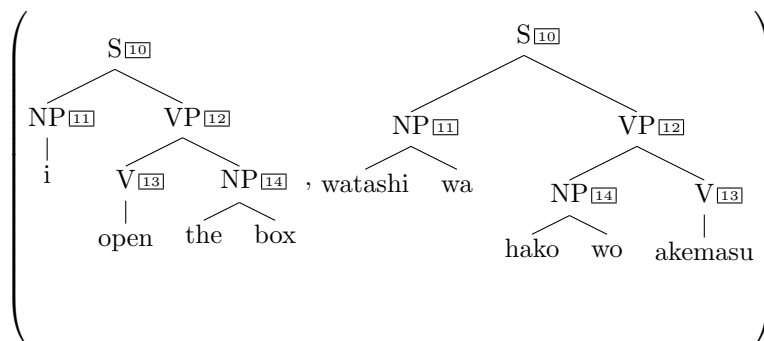
$$\Rightarrow \langle \text{i V}_{\boxed{13}} \text{NP}_{\boxed{14}}, \text{NP}_{\boxed{11}} \text{watashi wa V}_{\boxed{13}} \rangle \quad \text{not allowed}$$

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

$$\begin{aligned} &\Rightarrow \langle \text{i V}_{\boxed{13}} \text{NP}_{\boxed{14}}, \text{watashi wa NP}_{\boxed{14}} \text{V}_{\boxed{13}} \rangle \\ &\Rightarrow \langle \text{i open NP}_{\boxed{14}}, \text{watashi wa NP}_{\boxed{14}} \text{akemasu} \rangle \\ &\Rightarrow \langle \text{i open the box, watashi wa hako wo akemasu} \rangle \end{aligned}$$

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.

Example

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

- $$S \rightarrow \langle NP_1 VP_2, NP_1 VP_2 \rangle \tag{18}$$
- $$VP \rightarrow \langle V_1, V_1 \rangle \tag{19}$$
- $$VP \rightarrow \langle V_1 \bar{S}_2, \bar{S}_2 V_1 \rangle \tag{20}$$
- $$\bar{S} \rightarrow \langle Comp_1 S_2, S_2 Comp_1 \rangle \tag{21}$$
- $$Comp \rightarrow \langle \text{that, to} \rangle \tag{22}$$
- $$NP \rightarrow \langle \text{the boy, shoonen-ga} \rangle \tag{23}$$
- $$NP \rightarrow \langle \text{the student, gakusei-ga} \rangle \tag{24}$$
- $$NP \rightarrow \langle \text{the teacher, sensei-ga} \rangle \tag{25}$$
- $$V \rightarrow \langle \text{danced, odotta} \rangle \tag{26}$$
- $$V \rightarrow \langle \text{said, itta} \rangle \tag{27}$$
- $$V \rightarrow \langle \text{stated, hanasita} \rangle \tag{28}$$

Synchronous TAG We can also define *synchronous TAG* by analogy with synchronous CFG. In synchronous CFG we linked the nonterminal symbols between the two sides of a rule; in a synchronous TAG, we link the adjunction/substitution sites. How would you define a synchronous TAG to translate our Dutch example into English?

Parsing synchronous CFG

- Parsing synchronous CFG is as easy as parsing CFG: just parse the source side, then convert to target language
- But in certain cases where we need to access the target string (constrain it to a fixed string, or score it using a language model), parsing can jump up to $\mathcal{O}(n^6)$ time like TAG.
- In those same cases, synchronous TAG would jump to $\mathcal{O}(n^{12})!$