Lecture 12

Notions of Generative Capacity

Last time we saw that even though syntax does not define regular string languages due to center embedding (and possibly other constructions we have not considered yet), context-free grammars are powerful enough to handle center embedding. And in an unexpected turn of events we quickly realized that context-free grammars can be viewed as a mechanism for generating strictly 2-local tree languages. So if this is on the right track, syntax does not look all that different from phonology: where a significant fragment of phonology is strictly local over strings, syntax may be strictly local over trees. But as you remember, not all of phonology turned out to be strictly local, so we should not be too eager to claim that syntax is strictly local, either. And as we will see today, this issue is a lot trickier to resolve for syntax.

1 Weak and Strong Generative Capacity

A strictly local tree grammar produces two distinct outputs — a tree language and the corresponding string language. These two notions are not in perfect alignment. Failure to produce the right string language necessarily implies failure to produce the right tree language, but the opposite does not hold. Moreover, a formalism that does not produce the right string language may still be closer to the correct tree language than one that gets the string language right but does so at the expense of ludicrous tree structures. So if we assume that trees are indeed an integral part of syntax — rather than just a convenient device for us to represent certain dependencies over strings — then we have to make sure that strictly local grammars are sufficiently powerful to capture syntax at both the string level and the tree level. In more technical terms, we have to ensure that they have adequate *weak and strong generative capacity*.

Weak Generative Capacity the class of string languages that are generated by the formalism

Strong Generative Capacity the class of tree languages that are generated by the formalism

Weak and strong generative capacity do not exhaust the full spectrum of levels of generative capacity. One could also posit *derivational capacity* as a metric for how succinctly a formalism produces a given tree, or *relational capacity* as a metric for the string-meaning pairings that can be computed. These notions are very little understood, though, so they play no big role at this point in the categorization of formalisms. Even strong generative capacity is still severely understudied, though a lot of progress has been made in recent years. This is partially due to theoretical computational linguistics undergoing a shift in interest from string languages to tree languages. Another driving factor, however, is that more and more application areas build on tree languages. XML, for example, is essentially a standard for dynamically specifying tree languages, which grants it an enormous amount of flexibility and makes it an ideal tool for APIs, among other things. So this is yet another case where theoretical and applied interests converge, leading towards a new focus on understudied concepts — in the case at hand, tree languages.

2 Generative Capacity of Tree-Based Grammar Formalisms

2.1 Strictly Local Tree Grammars

We already proved that $\mathrm{SL}^{\mathrm{T}}_{k-1} \subsetneq \mathrm{SL}^{\mathrm{T}}_{k}$ for all k, so the strong generative capacity of strictly local tree grammars increases with the size of their locality domain. In addition, the tree languages generated by CFGs are a proper subclass of $\mathrm{SL}^{\mathrm{T}}_{2}$ due to CFG's strict distinction between terminal and non-terminal nodes. With respect to strong generative capacity, we thus obtain a strict hierarchy CFG $\subsetneq \mathrm{SL}^{\mathrm{T}}_{2} \subsetneq \mathrm{SL}^{\mathrm{T}}_{3} \subsetneq \cdots$. The strong generative capacity of CFGs and strictly 1-local tree grammars is incomparable. In fact, the two classes have only one tree language in common, and that is the empty language. With the exception of these slightly deviant cases, though, we get a nice proper hierarchy that mirrors exactly the expressive hierarchy for strictly local string languages.

What does a CFG for the empty tree language look like? And what is the corresponding strictly 1-local tree grammar?

The fact that strong generative capacity increases with the size of the locality domain suggests that weak generative capacity does too. After all, that's what we saw with strictly local string languages. Surprisingly, though, this is not the case: weak generative capacity stays the same after $\mathrm{SL}^{\mathrm{T}}_{2}$.

Theorem 12.1. For all $k \geq 2$, $\mathrm{yd}(\mathrm{SL}^{\mathrm{T}}_{k})$ is the class of context-free languages. ⌟

We do not give a proof at this point since the theorem will fall out as a corollary of an even more general result later on. Instead, let us look at some of the surprising repercussions of this fact. A first important observation is that the string languages generated by strictly local tree grammars have very different closure properties compared to their tree languages.

Lemma 12.2. The class of context-free languages is closed under union. ⌟

Proof. It suffices to show that for any two CFGs $G_1 := \langle \Sigma_1, S, R_1 \rangle$ and $G_2 := \langle \Sigma_2, S, R_2 \rangle$ there is a CFG G_3 that generates $L(G_1) \cup L(G_2)$. One can construct G_3 in a manner that is similar to the union of automata:

- for every rewrite rule $A \rightarrow A_1 \cdots A_n$ in R_i $(i \in \{1,2\})$, G_3 contains the rewrite rule $A^i \rightarrow A^i_1 \cdots A^i_1$,
- for every terminal symbol a of G_i $(i \in \{1,2\})$, G_3 contains the rewrite rule $a^i \rightarrow a$,
- the start symbol S of G_3 only occurs in the rewrite rules $S \rightarrow S^1$ and $S \rightarrow S^2$.

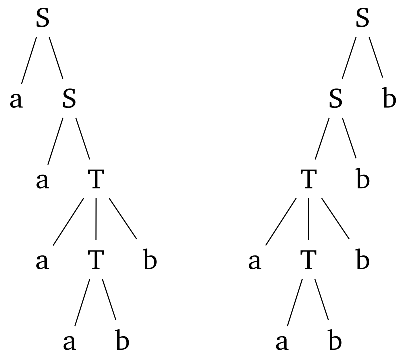
It is easy to see that G_3 generates every string that is a member of $L(G_1)$ or $L(G_2)$, and nothing else. □

Example 12.1 Constructing the Union of Two CFGs

Suppose $m > n \geq 1$. Then $a^m b^n$ and $a^n b^m$ are both context-free languages generated by the respective set of rewrite rules below:

$$\begin{array}{lll} S & \rightarrow & aS \\ S & \rightarrow & aT \\ T & \rightarrow & aTb \\ T & \rightarrow & ab \end{array} \qquad \begin{array}{lll} S & \rightarrow & Sb \\ S & \rightarrow & Tb \\ T & \rightarrow & aTb \\ T & \rightarrow & ab \end{array}$$

The trees for *aaaabb* and *aabbbb* are shown below.



The union of $a^m b^n$ and $a^n b^m$ is $a^+ b^+ \setminus a^n b^n$.

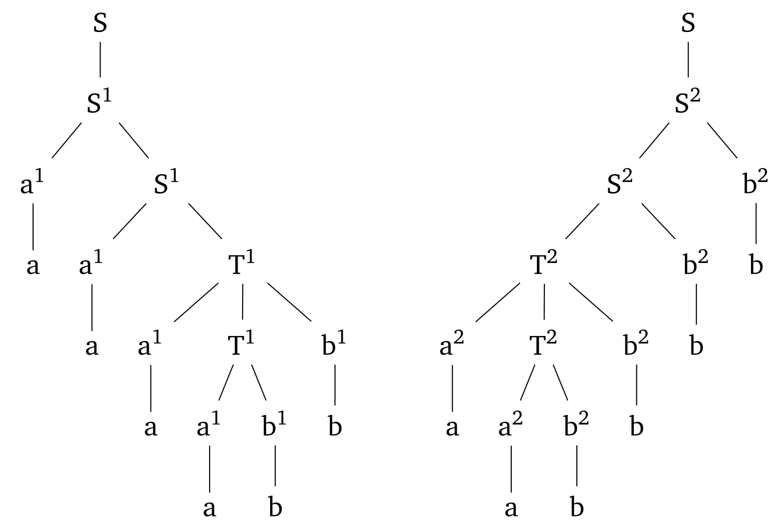
We now build a CFG G_3 for this language, meticulously following the steps in the proof. The first step collects all rewrite rules and adds a superscript to every symbol in the rule.

$$\begin{array}{lll} S^1 & \rightarrow & a^1 S^1 \\ S^1 & \rightarrow & a^1 T^1 \\ T^1 & \rightarrow & a^1 T^1 b^1 \\ T^1 & \rightarrow & a^1 b^1 \end{array} \qquad \begin{array}{lll} S^2 & \rightarrow & S^2 b^2 \\ S^2 & \rightarrow & T^2 b^2 \\ T^2 & \rightarrow & a^2 T^2 b^2 \\ T^2 & \rightarrow & a^2 b^2 \end{array}$$

Next we have to add a rewrite rule for every non-terminal that we superscripted.

$$\begin{array}{lll} a^1 & \rightarrow & \text{a} \\ a^2 & \rightarrow & \text{a} \\ b^1 & \rightarrow & \text{b} \\ b^2 & \rightarrow & \text{b} \end{array}$$

Finally, we add the rules $S \rightarrow S_1$ and $S \rightarrow S_2$. After the initial rewrite step of S, this grammar behaves almost exactly like G_1 or G_2, depending on what S was rewritten as.



Closure under union is surprising, but even more so is non-closure under intersection.

Lemma 12.3. The class of context-free languages is not closed under intersection. ⌟

Proof. We are not in a position to fully prove this yet. The basic idea is as follows: $a^+b^nc^n$ and $a^nb^nc^+$ are both context-free languages, but their intersection is $a^nb^nc^n$, which is not context-free. This can be shown via a pumping lemma, which we will encounter later on. □

How is it possible that the strictly local tree languages are closed under intersection but not union, while the very opposite holds for their string yields? Shouldn't closure under intersection for tree languages imply closure under intersection for the string yields, too? The answer is no, because the two intersection operations produce very different outputs. Consider the tree languages L_1 and L_2 that only contain the trees $[_S$ $[_A$ a]] and $[_S$ $[_B$ a]], respectively. Then $\text{yd}(L_1) = \text{yd}(L_2) = \{a\}$, so the intersection of their string languages is also $\{a\}$. But if we directly intersect L_1 and L_2, we get the empty set instead because they have not a single tree in common. And the string yield of the empty set is also the empty set, which definitely is not the same thing as $\{a\}$. So $\text{yd}(L_1) \cap \text{yd}(L_2) = \{a\} \neq \emptyset = \text{yd}(L_1 \cap L_2)$. It is this misalignment between intersecting tree languages and intersection their string yields that leads to the skewed closure property.

A similar problem arises with union. Union of strictly local tree languages does not necessarily preserve their strict locality because we might be able to build completely new trees from the union of the k-trees. But for union of the string yields, we can freely alter the tree structure, including relabeling interior nodes to keep the k-grams distinct. So once again $\text{yd}(L_1) \cup \text{yd}(L_2)$ is not necessarily the same as $\text{yd}(L_1 \cup L_2)$, and closure under union for string languages cannot be lifted to the level of tree languages.

Keeping all of this mind, you should not be too surprised anymore that context-free languages are also closed under relabelings even though the strictly local tree languages are not.

Lemma 12.4. The class of context-free languages is closed under relabelings. ⌟

Proof. Suppose L is generated by CFG $G := \langle \Sigma, S, R \rangle$ and the relabeling τ is specified as a finite set of pairs $\langle a, b \rangle$ such that $a \in \Sigma_T$. Then the image of L under τ is generated by the grammar G' with $R' := R \cup \{a \rightarrow b \mid \langle a, b \rangle \in \tau\}$. □

Theorem 12.5. The class of context-free languages is closed under union and relabelings. It is not closed under intersection and relative complement. ⌟

The difference in closure properties is both a strength and a weakness. On the one hand, it makes things a lot trickier, and one always has to pay attention what kind of objects are being manipulated, tree languages or string languages. On the other hand, it solves a problem we would be facing otherwise: if syntax defines context-free string languages, shouldn't we expect the union of two natural languages to be a natural language? In our discussion of phonology we already pointed out that closure under union is not a property of natural languages because it allows constraints to apply disjunctively. For syntax, the fact that some languages only enforce person agreement between the subject and the verb and some only number agreement would predict via closure under union that there is a language where the subject has to agree in person or number, but not both. This seems unlikely, and it is readily accounted for if we take syntax to generate tree languages, where closure under union does not hold.

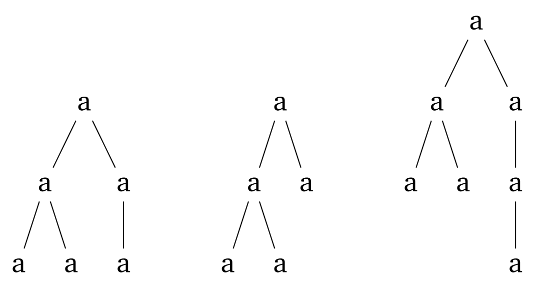
2.2 Refined Strictly Local Tree Grammars

Now that we have a tree analogue for strictly local grammars, it makes sense to define an analogue for refined strictly local grammars and ask how their generative capacity compares to that of standard strictly local tree grammars.

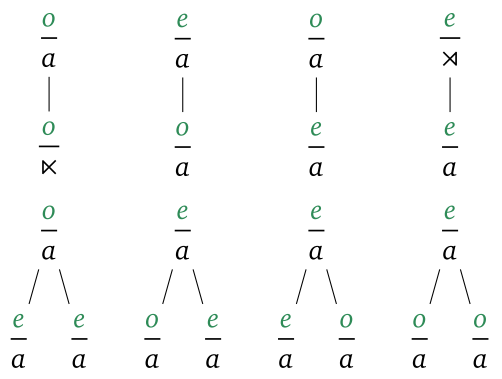
Definition 12.6 (Refined Tree Grammar). A *refined strictly k-local tree grammar G* is a finite set of k-trees over alphabet $\Sigma \times Q$. Such a grammar generates the tree language $L^T(G) := \left\{ t \mid \exists r \in Q^{\hat{t}}, k\text{-trees}(r) \subseteq G \right\}$. Its string language is $L^S(G) = \text{yd}(L^T(G))$. A tree language is refined strictly k-local (in $\text{SL}^{\text{T,R}}$) iff it is generated by a refined strictly k-local tree grammar. The class of all refined strictly k-local tree languages is $\text{SL}^{\text{T,R}} := \bigcup_{k \geq 0} \text{SL}^{\text{T,R}}_k$.

Example 12.2 Two Refined Strictly 2-Local Tree Languages

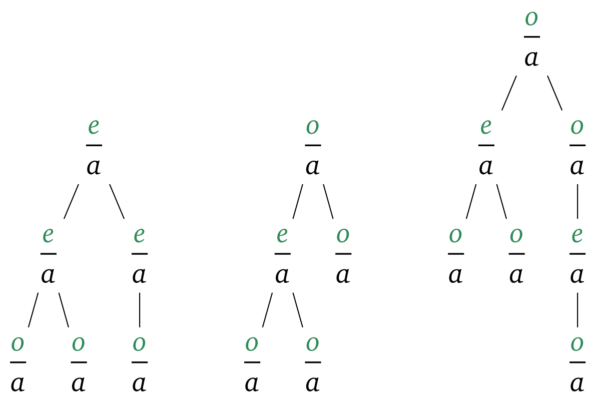
In lecture 11, we saw example of two tree languages that are not strictly local. The first one was the set of unary branching trees over alphabet $\{a\}$ that contain an even number of nodes. This language is refined strictly 2-local, though, as the construction for the string case can be lifted directly to unary branching trees. Let us look at a slight generalization instead, the set of all at most binary branching trees over alphabet $\{a\}$ with an even number of nodes. This language contains the leftmost tree but not the other two.



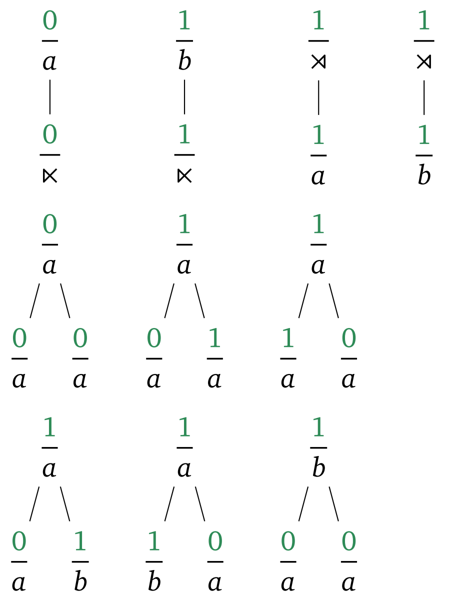
In order to generate this tree language, one needs the refined 2-trees below:



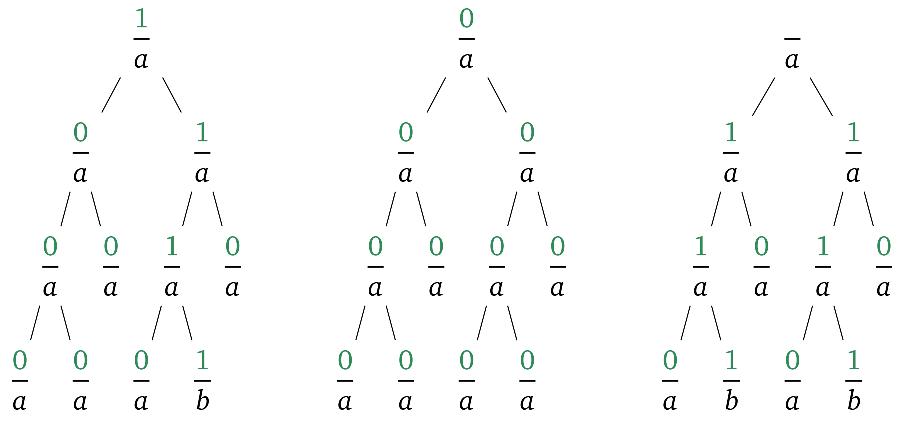
The state assignments for the three trees above now show that only the leftmost is generated by the grammar.



The other language we looked at was $L_{b=1}$, the set of all strictly binary branching trees over $\{a, b\}$ that contain exactly one instance of b. In this case, the states simply keep track of whether a b has been seen yet.



The state assignment once again shows the distinction between well-formed and ill-formed trees.



During our investigation of refined strictly local string languages, we saw that $\mathrm{SL}^R_k = \mathrm{SL}^R_{k+1}$ for all $k \geq 2$, which implies $\mathrm{SL}^R_2 = \mathrm{SL}^R$. The equivalence holds because the extra information that is afforded by the increased locality domain can be encoded in the hidden alphabet layer. For the very same reason, every strictly local language turns out to be refined strictly 2-local. Adapting the constructions used in the proof from strings to trees is straight-forward, and consequently we find the same relations among refined strictly local tree languages.

Theorem 12.7. $\mathrm{SL}^{\mathrm{T}} \subsetneq \mathrm{SL}^{\mathrm{T,R}}_2 = \mathrm{SL}^{\mathrm{T,R}}$ ┘

These relations will allow us to prove rather elegantly that all $\mathrm{SL}^{\mathrm{T}}_k$ have the same weak generative capacity. For this is just a corollary of the fact that the class of context-free languages is exactly $\mathrm{yd}(\mathrm{SL}^{\mathrm{T,R}}_2)$.

Theorem 12.8 (Thatcher 1967). CFGs are weakly equivalent to $\mathrm{SL}^{\mathrm{T,R}}$. ⌟

Proof. Since every CFG defines a strictly 2-local tree language and $\mathrm{SL}^{\mathrm{T}} \subsetneq \mathrm{SL}_2^{\mathrm{T,R}}$, every context-free language is included in $\mathrm{yd}(\mathrm{SL}_2^{\mathrm{T,R}})$. In the other direction, let $L \in \mathrm{SL}^{\mathrm{T,R}}$. Then there is some refined strictly 2-local tree grammar G_2 with $L^T(G_2) = L$. We construct a CFG G_C such that $L^S(G_2) = L^S(G_C)$.

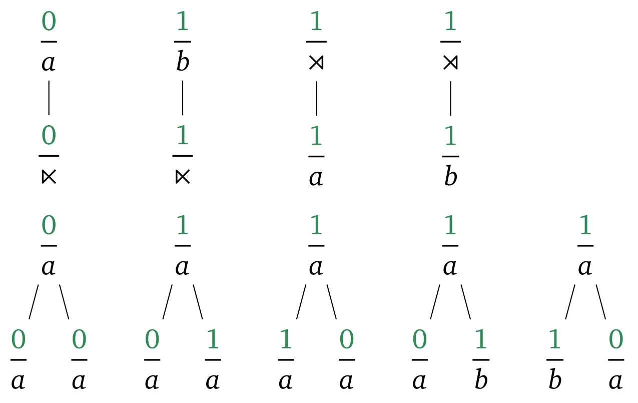
Let $k \in G_2$.

- If k is of the form $[_{\frac{q}{\rtimes}}\ \frac{q_1}{A_1}]$, then G_C contains the rewrite rule $S \rightarrow \langle A_1, q_1 \rangle$.
- If k is of the form $[_{\frac{q}{A}}\ \frac{q_1}{\ltimes}]$, then G_C contains the rewrite rule $\langle A, q \rangle \rightarrow A$.
- In all other cases, k is of the form $[_{\frac{q}{A}}\ \frac{q_1}{A_1} \cdots \frac{q_n}{A_n}]$ $(n \leq 1)$ and G_C contains the rewrite rule $\langle A, q \rangle \rightarrow \langle A_1, q_1 \rangle \cdots \langle A_n, q_n \rangle$.

Let τ be the projection that maps every non-terminal symbol $\langle A, q \rangle$ to A. Furthermore, $t[l \leftarrow l^2]$ is the tree that is obtained from t by replacing every leaf a with the tree $[_a a]$. Note that $\mathrm{yd}(t) = \mathrm{yd}(t[\leftarrow l^2])$. Close inspection of the translation above reveals that $s \in L^T(G_2)$ iff there is a tree $t \in L^T(G_C)$ such that $\tau(t) = [_S s[l \leftarrow l^2]]$. It follows that $\mathrm{yd}(s) = \mathrm{yd}(t)$, and by extension $L^S(G_2) = L^S(G_C)$. □

Example 12.3 Translating a Refined Tree Grammar Into a CFG

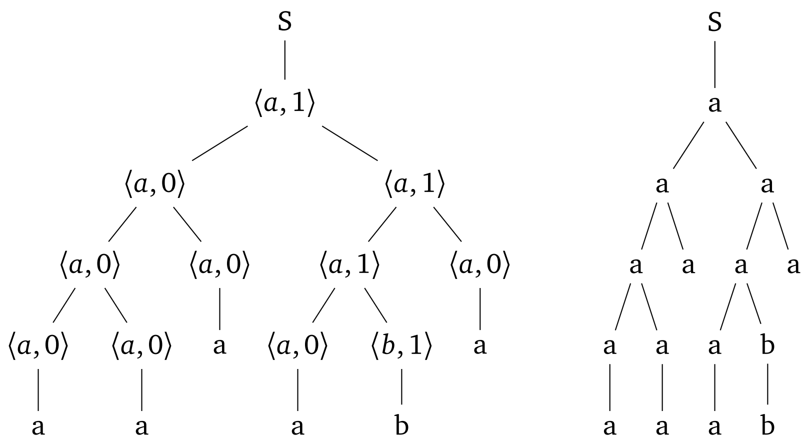
The previous example gave a refined strictly 2-local grammar for $L_{b=1}$, the set of all strictly binary branching trees over $\{a, b\}$ that contain exactly one instance of b. We consider a minor variant where only leafs can be labeled b. To this end we have to remove only one 2-tree from the original grammar.



The procedure described in the proof translates these nine k-grams into nine rewrite rules.

$$\begin{array}{lcl} \langle a,0 \rangle & \rightarrow & a \\ \langle b,1 \rangle & \rightarrow & b \end{array} \qquad \begin{array}{lcl} S & \rightarrow & \langle a,1 \rangle \\ S & \rightarrow & \langle b,1 \rangle \end{array} \qquad \begin{array}{lcl} \langle a,0 \rangle & \rightarrow & \langle a,0 \rangle\ \langle a,0 \rangle \\ \langle a,1 \rangle & \rightarrow & \langle a,0 \rangle\ \langle a,1 \rangle \\ \langle a,1 \rangle & \rightarrow & \langle a,1 \rangle\ \langle a,0 \rangle \\ \langle a,1 \rangle & \rightarrow & \langle a,0 \rangle\ \langle b,1 \rangle \\ \langle a,1 \rangle & \rightarrow & \langle b,1 \rangle\ \langle a,0 \rangle \end{array}$$

The well-formed tree from the previous example now has the left tree as its correspondent, while the right tree shows the image under τ.



We now have a very intriguing result: CFGs, strictly local tree grammars, and refined strictly local tree grammars all have the same weak generative capacity. The choice between them thus cannot be motivated by well-formedness data since the well-formedness of a sentence in, say, English only tells us that it belongs to the string language defined by English, and all these formalisms define exactly the same string languages. If we want to make an informed choice between these formalisms, it will have to be in terms of the tree languages they can define. That is a much more subtle issue that requires a lot of linguistic finesse. But more on that next time.