In the main chapter I reviewed the classical Chomsky hierarchy as it was defined in the late 1950s, with the aim of highlighting important and general intuitions about how generation by a grammar works. This supplement aims to show that once these crucial core ideas are highlighted, it becomes possible to identify other ways of fleshing them out: there is a broader landscape of formal grammars, extending beyond the particular formalisms comprising Chomsky’s original hierarchy from the 1950s. While it is these more recent (and less well-known) kinds of grammars that provide the closest fit to the particular empirical details of natural languages and contemporary linguistic theories, they remain variations on the same themes. The important lasting ideas that were brought out by study of the original hierarchy — intersubstitutability, the tracking of some distinctions and ignoring of others, and effects of using different forms of memory to carry out this tracking — underpin a thorough understanding of the relationship between any kind of grammar and the collection of expressions generated, and therefore continue to provide the formal grounding for the Chomskyan conception of linguistics, where grammars rather than collections of observed sentences are the primary objects of interest.

While the idea of intersubstitutability is most familiar from its connection to basic notions of phrase structure and constituency (and it is this connection that is most relevant in Chomsky’s early formal work), this concept is, as I have tried to stress, a very general one that gets at the core of how any sort of grammar differs from a mere collection of sentences. One thing I hope to show below is that the general form of this idea provides a perspective that unifies the “basic syntax” notions of phrase structure and constituency, with the phenomena that linguists in the Chomskyan tradition usually associate with operations like movement or agreement (which are less frequently connected with the formal study of grammars). For example, the conditions on reflexive-licensing lead to an important distinction between wash the clothes and wash himself: these two phrases are not intersubstitutable (even if, at some more basic level, there is a sense in which both are verb phrases), since only the former can combine with they will to form a sentence. Similarly, the conditions on NPI-licensing dictate that with three dogs and with any dogs are not intersubstitutable (even if there is a sense in which both are prepositional phrases). So the idea of intersubstitutability is useful beyond its applicability to identifying familiar categories such as nouns, adjectives, verb phrases, etc. More on this in §A.3 below.

The notion of intersubstitutable subexpressions is also closely related to the way grammars allow “infinite use of finite means”. By excluding the possibility that a grammar has infinitely many rules, we restrict the number of “cuts” that can be made (between nouns and verbs, between verbs and adjectives, etc.),
or the number of distinctions that can be used to delineate classes of intersubstitutable expressions — the intersubstitutability equivalence relation cannot be infinitely fine-grained. The infinity of generated expressions, however, comes not from an intersubstitutability relation that makes many such cuts or is sensitive to many such distinctions, but from one which, due to its coarseness, ignores or collapses some distinctions, specifically the distinction between two expressions that stand in a part-whole relationship to each other. The fact that the boy is intersubstitutable with the picture of the boy, for example, lets us immediately conclude that each of these is intersubstitutable with the picture of the picture of the boy, and so on.

That ideas such as these should have emerged from the original 1950s work and then continued, in new and varied guises, to sharpen linguistic insights, seems to be in line with the optimism Chomsky expressed shortly thereafter:

> "the systems that have so far proved amenable to serious abstract study are undoubtedly inadequate to represent the full complexity and richness of the syntactic devices available in natural language … Nevertheless … Certain basic properties of natural languages (e.g. bracketing into continuous phrases, categorization into lexical and phrase types, nesting of dependencies) appear in systems of the kind that have been surveyed. Hence the study of these systems has some direct bearing on the character of natural language. Furthermore, it is clear that profitable abstract study of systems as rich and intricate as natural language … will require sharper tools and deeper insights … and these can be acquired only through study of language-like systems that are simpler than the given natural languages.”

(Chomsky, 1963, p.415)

Chomsky apparently became less optimistic about this over time:

> “Now mathematical linguistics I think came to a rather critical point in its extremely brief career about two or three years ago, when some simple ideas were basically exhausted … The early work in mathematical linguistics was concerned with ideas which are simple enough to study from a mathematical point of view, but which are not complex enough to have very much to do with real language structure.”

(Chomsky quoted in Staal, 1968, p.19)

But I hope to show here that the situation was not quite so bleak.

**A.1 Strict locality**

The idea of strict locality does not appear in the original Chomsky hierarchy, but has turned out to be useful in a few different ways. Chomsky mentions the relevant class of automata, but only very briefly — presumably because they are even less powerful, in generative capacity terms, than FSAs.¹

A strictly-local grammar is based on the simple idea of $n$-grams. An $n$-gram is a sequence of $n$ consecutive symbols. I’ll restrict attention to the special case of 2-strictly-local grammars, so the relevant $n$-grams will be sequences of length two, known as bigrams; generalizing to larger $n$-grams is straightforward. A 2-strictly-local grammar is just a list of allowable bigrams, plus some specification of the symbols that strings can start and end with. A convenient way to formulate this is to use the special symbol $\#$ to indicate the edges of the string, so that the bigrams $\#a$ and $a\#$ represent a appearing at the beginning and end of a string, respectively.

An example of a strictly-local grammar (SLG) is given in Figure A-1, in two formats: as a list of allowable bigrams, and as a diagram. This grammar generates all strings using the symbols \{a, b, c\} that begin with a, end with c, do not contain two adjacent bs and do not contain adjacent occurrences of a and c (in either order).

¹See Chomsky (1956, §2.4) on “$n$-order approximations” and Chomsky (1963, p.336) on “$k$-limited automata”. The first significant formal investigation of strict locality was by McNaughton and Papert (1971), who call the relevant concept “locally testable in the strict sense” (p.17).
Construed as a list of bigrams, it’s natural to interpret an SLG representationally, as admissibility conditions that are applied to each bigram of a string. But we also can interpret SLGs derivationally. Specifically, we can imagine a “write head” in the position indicated by • in (A.1), which repeatedly chooses a symbol to write next on the basis of the symbol immediately to its left; to begin, it chooses from bigrams that have # in their first position, and when it choose a bigram that has # in its second position the process ends. Importantly, the head is “stateless”, in the sense that there is nothing being rewritten as there is in the grammars of the original Chomsky hierarchy; the constraints on what can be written stem only from the other already-written symbols, not from a different book-keeping symbol (i.e. nonterminal symbol) that is being replaced. The line-by-line derivation of \textit{abccbaabc} would proceed as follows:

(A.1)
\begin{itemize}
  \item \textit{a}
  \item \textit{ab}
  \item \textit{abc}
  \item \textit{abcc}
  \item \textit{abccc}
  \item \textit{abcccb}
  \item \textit{abcccbba}
  \item \textit{abcccbbaab}
  \item \textit{abcccbbaabc}
  \item \textit{abcccbbaabab}
\end{itemize}

The diagram in Figure A-1 makes it clear that there are similarities between the generative process associated with SLGs and that of FSGs. The crucial difference is that an SLG directly specifies allowable transitions between symbols, whereas an FSG specifies transitions between states — the relationship between a surface string and the sequence of state-transitions taken by an FSG is (in general) opaque, because the states themselves are “hidden”, but observing a surface string generated by an SLG tells us exactly which grammatical rules (bigrams) are being used to generate it.\footnote{The relationship between SLGs and FSGs corresponds almost exactly to the relationship between (visible) Markov models and hidden Markov models in the realm of probabilistic grammars (e.g. Manning and Schütze, 1999, pp.317–325).}

Because of this “surface-oriented” nature of SLGs, there is a particularly simple and transparent relationship between a string and the ways it can be used. Recall that in FSGs and CFGs, it is a string’s forward set (a set of states) or inside set (a set of nonterminals) that dictates the combinatory potential of the string. An SLG has no such internal book-keeping symbols, so it is the surface symbols themselves that dictate a string’s combinatory potential. Specifically, an SLG treats any two strings that end with the same symbol as intersubstitutable prefixes. For example, the two strings \textit{ab} and \textit{baccb} will be treated as intersubstitutable prefixes by the SLG in Figure A-1, and indeed by every SLG, since they both end with \textit{b}; for any other string \textit{u}, the two strings \textit{ab} ++ \textit{u} and \textit{baccb} ++ \textit{u} will either both be generated or both not be generated, depending only on whether the transition from the symbol \textit{b} to the first symbol of the string \textit{u} is allowed.

A useful way to bring out the distinction between SLGs and FSGs is to consider the distinction between immediate precedence and unbounded precedence. The SLG in Figure A-2 enforces an immediate precedence requirement: it generates strings using the symbols \{x, y, z\} satisfying the requirement that \textit{z} can only appear when it is immediately preceded by \textit{y}. This is straightforwardly achieved by including \textit{yz} as the only bigram.

\begin{figure}
\centering
\begin{tabular}{c}
\hline
\textbf{#a} & \textbf{ab} & \textbf{ba} & \textbf{cb} & \textbf{c#} \\
\hline
\textbf{aa} & \textbf{bc} & \textbf{cc} \\
\hline
\end{tabular}
\caption{Allowable-bigrams representation}
\end{figure}

\begin{figure}
\centering
\begin{tabular}{c}
\hline
\textbf{#} & \arrow{a} & \arrow{b} & \arrow{c} & \textbf{#} \\
\hline
\end{tabular}
\caption{Graphical representation}
\end{figure}

\textbf{Figure A-1:} Two representations of a strictly-local grammar
Figure A-2: Two representations of a strictly-local grammar enforcing immediate precedence

start symbol: P
P → x
P → x P
P → y
P → y Q
Q → x
Q → x Q
Q → y
Q → y Q
Q → z
Q → z Q

(a) Rewrite-rule representation

(b) Graphical representation

Figure A-3: Two representations of a finite-state grammar enforcing unbounded precedence with z in the second position. More abstractly, we can notice that this pattern is the sort that an SLG will be able to express because it only requires distinguishing between strings that end in y (and therefore allow z as the next symbol) and strings that do not. It is “safe” to ignore all distinctions between strings that share a final symbol.

Consider now switching to a requirement that z can only appear if it is preceded, not necessarily immediately, by y. The FSG in Figure A-3 straightforwardly enforces this unbounded precedence constraint, using the distinction between states P and Q to track whether or not a licensing y has been encountered. Crucially, a pair of prefixes such as xxxx (which has forward set {P}) and xyxx (forward set {Q}) are not treated as intersubstitutable — as is necessary if we are going to enforce the requirement, since only one of the two licenses z as the next symbol — even though they both end with the symbol x. See Figure A-4 for an illustration of this, in comparison to the SLG-enforceable immediate precedence requirement.

But now consider trying to enforce this unbounded precedence requirement in an SLG. Since xxxx and xyxx end with the same symbol, they will be treated as intersubstitutable by any SLG that we might construct: the bigram xz will be either allowed, in which case the grammar will incorrectly generate xxxxz, or not allowed, in which case the grammar will incorrectly fail to generate xyxz. Put differently, for an SLG to generate the string xyxz it must allow (among others) the bigrams #x, xx, xz and z#; and once it allows these, it will inevitably also generate xxxxz.

In short, FSGs can represent dependencies across unbounded distances, and SLGs cannot. While the possible symbols that an FSG might generate in the nth position of a string depend entirely on the state(s) reached after generating the preceding n - 1 symbols, the state(s) reached might be sensitive to symbols that are
Figure A-4: Comparison of the strictly-local and finite-state perspectives on the potential combinations of the prefixes $xyxy$ (top), $xyxx$ (middle) and $xxxx$ (bottom) with the suffix $z$. In a strictly-local grammar, any two prefixes that end with the same symbol will be combinable with all the same suffixes, as indicated by the bigrams shown inside dotted lines; the SLG in Figure A-2 does not allow prefixes ending in $x$ to combine with $z$. The FSG in Figure A-3, on the other hand, distinguishes between prefixes that contain $y$, which have forward set $\{Q\}$ and can be followed by $z$, and prefixes that do not contain $y$, which have forward set $\{P\}$ and cannot be followed by $z$.

The concept of strict-locality has turned out to be useful in at least three ways.

First, the class of strictly-local grammars, along with slight extensions of this concept, has proven to be a useful framework for analyzing phonotactic constraints. The constraint violated by $bnick$ but not $blick$, for example, might require that a stop is not immediately followed by a nasal, which can be enforced by a SLG. See e.g. Rogers et al. (2013), Chandlee (2014), Heinz (2018).

Second, being able to view these constraints on symbol adjacency as a (very simple) sort of grammar allows us to better understand the relationship between grammars in general and mechanisms that it is tempting to describe as “purely statistical”. The best-known instantiations of the strict-locality idea bundled it up with its statistical/probabilistic variant\(^3\) (e.g. “statistical learning” in psychology (e.g. Saffran et al., 1996), or “$n$-gram models” in natural language processing (e.g. Manning and Schütze, 1999)), and the fact that the non-probabilistic analogs (i.e. SLGs as presented in this section) were not often discussed in introductions to formal grammar gave a misleading impression that these kinds of theories or models worked with “no grammar”. With SLGs as a part of our typology of grammars, however, it becomes clear that those systems are the probabilistic instantiations of SLGs, just as more complex probabilistic systems are the probabilistic instantiations of FSGs or CFGs.

Third, a relationship between SLGs and CFGs paves the way for many important parts of the theory of string-generating grammars to be generalized over to tree-generating grammars, which I will outline in §A.2 and §A.3.

---

\(^3\)This is even true of Chomsky’s brief early mentions of the idea; see footnote 1.
A.2 Generalizing to trees: From CFGs to strictly-local tree grammars

Recall that CFGs involve hierarchical structures only implicitly: the objects that the grammar actually manipulates are strings, and hierarchy arises implicitly as a result of CFGs’ ability to build strings up “from the inside out”, working with infix-circumfix structure rather than simply prefix-suffix structure. But from a linguistic perspective, it’s natural to want to work with trees directly. Transformations, for example, are typically thought of as operations on trees, unlike everything discussed here so far, which concerns operations on strings.\footnote{Early formalizations of transformations used a set of strings as the object that transformations operate on; this works, but is rather cumbersome (e.g. Lasnik and Kupin, 1977).}

Other researchers quickly took on the challenge of generalizing the basic ideas emerging from the Chomsky hierarchy from strings to trees (Thatcher, 1967; Thatcher and Wright, 1968; Doner, 1970; Rounds, 1968, 1970; Thatcher, 1973). Rogers (1997, 1998) later brought the resulting ideas into closer proximity to the linguistics community. This work not only drew on Chomsky’s string-manipulating grammars in its technical details, but was also often explicitly inspired by the fact that Chomsky had turned to tree-manipulating transformational grammars as a better tool for linguistic theorizing. For example, Thatcher (1967, p.317) writes in the first paragraph of his paper that “From the point of view of transformational grammars, sets of trees are of prime importance as opposed to sets of strings”; and the introduction to Rounds 1968 briefly summarizes the classical Chomsky hierarchy and the motivation for the addition of transformations for linguistic purposes, leading up to “Related sentences can then be derived by transformations, which are mappings defined on the set of P-markers, or trees, of the kernel sentences . . . In this thesis, we propose a mathematical model of these mappings” (p.5).\footnote{It appears, in hindsight, surprising that this line of work, beginning in the late 1960s, did not have more of an impact on linguistics. From a modern computational perspective, the generalization from strings to trees seems particularly natural given the close relationship between them when viewed as recursive data structures. A possible source of the disconnect is that even strings were not typically viewed as a recursive data structure in much of the early work on string grammars; the recursive perspective apparently became more prominent only when researchers considered finite-state automata from an algebraic viewpoint, construing a string as a term in a monadic algebra which has the automaton’s states as its elements. From this point it becomes natural to consider the generalization beyond monadic algebras, which yields tree-shaped terms. See Thatcher and Wright (1968, pp.57–59) for a succinct account of this.}

While the theory of tree grammars does build on the ideas from string grammars discussed above, it (perhaps somewhat counter-intuitively) does not involve refinements or extensions of the implicit notion of hierarchy present in the way CFGs generate strings; it does not involve going further in the direction that we go in by moving from FSGs to CFGs. The tree diagrams we use to illustrate the workings of CFGs are useful for indicating the context-free way in which strings get composed out of substrings, which differs from the finite-state way or the strictly-local way in which strings get composed out of substrings. The theory of tree grammars reveals that we can generalize all of those different ways of composing larger objects out of smaller ones, and find devices that compose trees out of subtrees in the context-free way, or the finite-state way, or the strictly-local way. It turns out that with the shift to a more complex sort of generated object (trees rather than strings), the less powerful kinds of combinatoric mechanisms (strictly-local or finite-state, rather than context-free) suffice to generate patterns with relevance to syntactic theory. A result of this — again, perhaps counter-intuitively at first — is that the first steps in appreciating tree grammars rely more on having a solid grasp of the “low end” of the hierarchy of string grammars, i.e. FSGs and SLGs, than on an understanding the “high end” (CFGs and CSGs).

To begin, notice that when we write down the CFG in (2) for \( a^n b^n \), we’re doing something very similar to specifying a collection of treelets, which can be stuck together by linking a node at the top of one treelet with a node of the same label at the bottom of another treelet.

\[
(A.2) \\
\begin{array}{c}
  \text{s} \\
  a \quad \text{s} \quad b \\
  \quad a \quad b \\
\end{array}
\]

Notice that I’ve written \( s \) here, rather than \( S \) as in (2) above, to reflect the fact that the \( s \) in these treelets is (just like \( a \) and \( b \)) a symbol that is contained in the tree that is generated by the grammar, as opposed
to being a grammar-internal book-keeping symbol like S above. By chaining together these treelets in the
intended way, we can construct infinitely many trees, such as those in (A.3). Strings such as aabb and aaabb
are playing no role here.

(A.3)

\[ s \]
\[ \begin{array}{c}
  s \\
  a \quad s \\
  a \\
\end{array} \quad \begin{array}{c}
  s \\
  a \quad s \\
  a \\
\end{array} \]

It turns out that the mechanism we are appealing to here is very similar to the mechanism underlying SLGs;
it generates trees in the same way that SLGs generate strings. Notice that since a, b and s are all surface
symbols now, the distinction between terminal symbols (a and b) and nonterminal symbols (S) in the CFG
in (2) has been lost: we need something to tell us that an s node can appear at the root of a generated
tree, and that an a or b node can appear as a leaf node — but an s node cannot appear as a leaf node, for
example. For this we can add treelets containing # markers that work analogously to the way we indicated
possible starting and ending symbols in SLGs.

(A.4)

\[ a \quad b \]
\[ \begin{array}{c}
  a \quad s \quad b \\
  a \\
\end{array} \quad \begin{array}{c}
  a \quad b \\
  a \\
\end{array} \quad \begin{array}{c}
  a \quad s \\
  a \\
\end{array} \quad \begin{array}{c}
  a \quad b \\
  a \\
\end{array} \quad \begin{array}{c}
  a \quad s \\
  a \\
\end{array} \quad \begin{array}{c}
  a \quad b \\
  a \\
\end{array} \]

We can think of a tree grammar like this as specifying a number of “admissibility conditions”, similar to
the representational understanding of SLGs: to check whether a string is generated by a SLG, it suffices to
check each length-two substring; to check whether a tree is generated by the grammar in (A.4), it suffices to
check each depth-two subtree. Or, we could think of it as a “stateless write head” like in (A.1), gradually
writing out a tree, where its choice of what to write next depends only on the last symbol it wrote.

(A.5)

\[ \bullet \quad s \\
\[ \begin{array}{c}
  s \\
  a \quad s \\
  a \\
\end{array} \quad \begin{array}{c}
  s \\
  a \quad s \\
  a \\
\end{array} \quad \begin{array}{c}
  s \\
  a \quad s \\
  a \\
\end{array} \quad \begin{array}{c}
  s \\
  a \quad s \\
  a \\
\end{array} \]

The difference from SLGs is that this write head can “branch”, i.e. write things that, in effect, produce two
new write heads, as opposed to one which is just shifted along linearly as in (A.1). But this is just the
difference between trees and strings: a large string is comprised of a symbol and a smaller substring; a large
tree is comprised of a symbol and one or more smaller subtrees.\(^6\)

I will call (A.4) a \textit{strictly local tree grammar} (SLTG), distinguished from (but importantly related to) the
\textit{strictly local (string) grammars} (SLGs) discussed in §A.1. While SLTGs are of course also closely related, in
a different way, to context-free (string) grammars, focusing instead on the parallel to SLGs is the key step
in moving to the general idea of tree grammars.

\(^6\)The tree-shaped generative process underlying CFGs is a direct consequence of the fact that the right hand side of a rule
can have more than one nonterminal symbol; recall how the calculation of inside sets differs from that of forward sets.
It may be helpful to note that the collection of treelets — or “tree bigrams” — in (A.4) can be represented graphically as shown in Figure A-5, just as Figure A-1 and Figure A-2 show graphical representations of collections of (flat) bigrams. The arcs in the diagrams in Figure A-1 and Figure A-2 lead from one symbol to one other symbol; in Figure A-5, each treelet is represented by a “hyperarc” leading from one symbol to one or more symbols (Klein and Manning, 2004).

### Table A-1

<table>
<thead>
<tr>
<th></th>
<th>Strings as generated objects</th>
<th>Trees as generated objects</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Strictly-local</strong></td>
<td>SLGs (§A.1)</td>
<td>SLTGs (§A.2)</td>
</tr>
<tr>
<td><strong>Finite-state</strong></td>
<td>FSGs (§2)</td>
<td>FSTGs</td>
</tr>
<tr>
<td><strong>Context-free</strong></td>
<td>CFGs (§3)</td>
<td>CFTGs</td>
</tr>
</tbody>
</table>

A.3 Beyond strict-locality on trees

Once we observe that the simple tree grammars of the sort in (A.4) generate trees in the same way that SLGs generate strings, we can think of a range of possible kinds of grammars laid out as in Table A-1. Each row of the table corresponds to a particular way of composing larger objects out of smaller ones. The kinds of grammars that make up the original Chomsky hierarchy (e.g. FSGs, CFGs) sit on different rows, but all sit in the same column because in all cases the generated objects are strings. In the previous subsection we saw that a certain simple way of constructing trees, which is implicit in the workings of context-free (string) grammars, is exactly what we get by applying strictly-local composition mechanisms to tree-shaped objects, i.e. moving into the rightmost column of Table A-1.

This raises a few very natural questions. First, we know that there are many constraints on strings that cannot be expressed by SLGs; so what sorts of constraints on trees can and cannot be expressed by SLTGs? Second, what sorts of constraints on trees can be expressed by adapting more powerful composition mechanisms (e.g. finite-state and context-free) to the generation of trees? And third, where do linguistically-relevant patterns fall in this hierarchy of tree grammars? A solid understanding of the relationships between the different rows of the table — which are most simply illustrated with reference to the corresponding kinds of string-generating grammars, as the earlier parts of this chapter have attempted to do — is useful for approaching these questions.

My goal in what follows is mainly to give a partial answer to the first question: I will give an example of a constraint on trees that cannot be enforced by an SLTG. I leave the details of exactly what the more powerful kinds of tree grammars look like beyond the scope of this chapter. But by emphasizing the parallels between the limits of what SLGs can do with strings and the limits of what SLTGs can do with trees, I hope to give the reader a sense of the cross-cutting dimensions in Table A-1.

To begin, consider the new SLTG in (A.6), which is an expanded variant of the one in (A.4). Although it looks complicated, it simply expresses all combinations of a small number of choices, which I will explain carefully.

---

7 See for example Comon et al. (2007) or Gécseg and Steinby (1984) for textbook presentations. Thatcher (1973) also provides a good informal overview.
In addition to $s$, in this grammar we have $l$ and $r$ as possible symbols at non-leaf nodes. Notice that in (A.4), the $s$-rooted ternary treelet creates a loop, effectively expressing the possibility of “transitioning from $s$ to $s$” as we work down the spine of the tree. (Figure A-5 shows this loop graphically.) Similarly, the nine ternary treelets in (A.6) express all possible transitions from one of the non-leaf symbols ($s$, $l$ and $r$) to another; the treelet with $s$ at its root and $r$ as its middle daughter, for example, can be thought of as expressing a transition from $s$ to $r$. The $s$-rooted binary treelet in (A.4), on the other hand, expresses the possibility of ending (the spine of) the tree from an $s$ node, and the three binary treelets in (A.6) similarly express the possibility of ending from $s$, $l$ or $r$.

In this grammar we also have more possible symbols at leaf nodes: in addition to $a$ and $b$, we have $c$ and $d$. As in (A.4), every $s$ node has an $a$ leaf as its leftmost daughter and a $b$ leaf as its rightmost daughter. The ways that $l$ (“left”) and $r$ (“right”) differ from $s$ involve the placement of these new symbols $c$ and $d$ at leaf nodes: every $l$ node has a $c$ leaf as its leftmost daughter, and every $r$ node has a $d$ leaf as its rightmost daughter.

The net effect, then, of the additional treelets in (A.6) relative to (A.4), is that $c$ leaf nodes can appear in amongst the $a$s down the left edge of the tree, and $d$ leaf nodes in amongst the $b$s — with the parent node of a $c$ leaf labeled $l$, and the parent node of a $d$ leaf labeled $r$. For example, this extended grammar will generate the trees in (A.7), where the new symbols are highlighted for clarity.

(A.6)

\[
\begin{array}{cccccccc}
\# & \# & \# & a & b & c & d \\
\mid & \mid & \mid & \mid & \mid & \mid & \mid \\
s & l & r & \# & \# & \# & \#
\end{array}
\]

\[
\begin{array}{cccccccc}
s & s & s & s & s \\
s & a & s & b & a & l & b & a & r & b & a & b \\
\mid & \mid & \mid & \mid & \mid & \mid & \mid \\
l & l & l & l & l \\
c & s & b & c & l & b & c & r & b & c & b \\
\mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \middle
Now suppose we only wanted to generate the subset of these trees satisfying the requirement that any $d$ node must be c-commanded by a $c$ node; the first tree in (A.7) should be generated, but the second should not. This amounts to saying that any $r$ node on the spine of the tree must be dominated by an $l$ node.

There is no way to enforce this kind of c-command requirement in a SLTG. It is important and enlightening to observe that the reason for this is exactly the same as the reason why an SLG cannot enforce an unbounded-precedence requirement on $y$ and $z$, which instead required an FSG as shown above in Figure A-3. The strictly local nature of these tree grammars means that what can appear in a particular position can depend only on the immediately “preceding” — which here amounts to dominating — symbol. Following the pattern of the SLG in Figure A-2 that requires immediate precedence, we could construct a SLTG that requires each $r$ to be immediately dominated by $l$, by removing the $s$-dominating-$r$ and $r$-dominating-$r$ ternary treelets from (A.6); this would enforce a kind of local c-command between $c$ and $d$, ruling out both trees in (A.7).

As discussed above, the intersubstitutability of the prefixes $xyxx$ and $xxxx$ makes it impossible for an SLG to allow $z$ to come after $xyxx$ but disallow $z$ from coming after $xxxx$; the SLG has no way to “keep track of” the licensing $y$. Similarly, an SLTG cannot generate the first tree in (A.7) without also generating the second; it cannot allow an $r$-rooted tree to appear with surroundings corresponding to the $siss$ sequence on the spine, but disallow an $r$-rooted tree from appearing with surroundings corresponding to $ssss$. The SLTG has no way to keep track of the licensing $l$ node.

Put differently, any two subtrees that have the same symbol at their root are treated as intersubstitutable by an SLTG: any two trees with $s$ as their root symbol, for example, will be treated identically, regardless of whether there is an $r$ somewhere inside them, which will make it impossible to prevent those that do have an $r$ inside them from appearing with surroundings that contain no $l$.

This abstract example is intended to be reminiscent of many familiar syntactic requirements; for example, the requirement that an NPI be c-commanded by an appropriate licensor. In the abstract example, the crucial point is that a subtree whose root is labeled $s$ but contains an “unlicensed” $d$ needs to be kept distinct from a subtree whose root is labeled $s$ but does not contain an unlicensed $d$. Analogously, enforcing the NPI-licensing requirement means keeping the two trees in (A.8) distinct, since (as foreshadowed in the introduction) the former cannot in general be substituted for the latter (e.g. in the sentence “John went for a walk with three dogs”). And enforcing the appropriate licensing of reflexives requires keeping the two trees in (A.9) distinct, since they also have different distributions.

(A.8)

```
  PP  PP
 /\   /\?
with with
  DP  DP
   any    dogs
      three    dogs
```

(A.9)

```
  VP  VP
 /\   /\?
wash  wash
   DP  DP
    the    himself
     clothes
```

In (A.9), the two trees’ distributions differ in a way that is sensitive to the locality of a potential licensor, and in this sense (A.9) is unlike (A.8). But the fundamental point is that (A.8) and (A.9) each exhibit a pair of trees that we might wish to treat as non-intersubstitutable, even though they have the same root symbol.

The solution to this is to introduce grammar-internal book-keeping symbols that mediate the composition of trees out of smaller trees, in the same way that states and nonterminals mediate the composition of strings out of smaller strings in FSGs and CFGs. Recall from (A.1) and (A.5) that SLGs and SLTGs make no use
of grammar-internal symbols (like A and B); there are only surface symbols (like a and b). For the cases sketched in (A.7), (A.8) and (A.9), moving up from SLTGs to the finite-state mode of tree composition suffices (i.e. the context-free mode is not necessary) (recall Table A-1). We saw earlier in Figure A-4 how an FSG can enforce unbounded precedence by using (sets of) states to categorize subparts of strings, in particular by putting prefixes in different categories even if they end with the same symbol. Similarly, a finite-state tree grammar (FSTG) can enforce unbounded c-command by using (sets of) states to categorize subparts of trees, in ways that are not tied to the identity of the trees’ surface symbols. This contrasts with the way an SLTG could only enforce a bounded/local version of the c-command constraint by requiring that each r is immediately dominated by an l, like the SLG in Figure A-2 and Figure A-4 which requires that each z is immediately preceded by a y. Figure A-6 provides an illustration highlighting the analogy to Figure A-4.

The artificial example in (A.7) and Figure A-6 provides a particularly clear connection back to SL and FS string grammars, but it is also simple in some inessential ways that should not be allowed to obscure the general point. For example, the l and r nodes in the example allow the distribution of the crucial c and d nodes to be mediated purely with reference to symbols on a tree’s spine, but if all non-leaf nodes in these trees were simply labeled s then the requirement would still be enforcable by a FSTG. This is due to the fact that the licensing c node is only finitely far off the spine; note that if it were any further embedded, it would not c-command into the spine. It is also not essential that the trees here have a very simple shape, with a non-branching central spine: with reference to the diagrams in Figure A-6, the relevant FSTG can be constructed such that partial trees that contain a c that does not c-command the “hole” bring us to state P, like those that do not contain a c at all. And while the hypothetical relationship here between d and c is a simple one involving no locality or sensitivity to intervenors (i.e. it is analogous to the NPI example in (A.8) rather than the reflexives example in (A.9)), a FSTG can incorporate such requirements in the same ways that a FSG can incorporate constraints on an unbounded precedence requirement.

In broad terms, the distinction between what SLTGs can and cannot do corresponds roughly to the distinction between local selectional requirements (e.g. between a verb and its arguments) and the kind of long-distance hierarchical dependencies usually associated with c-command or movement. This jibes with the fact that in many forms of generative grammar, selectional requirements have been enforced in a base component that took the form of something close to a context-free grammar, with other dependencies mediated by more powerful transformational operations. While the discussion here has not addressed movement itself (i.e. phonological displacement), note that the first tree in (A.7) bears no necessary connection to the string acaaaaabdbbbb. The linearization of this generated tree might rearrange its leaf nodes in some way that is mediated by the FSTG-enforced hierarchical dependency between the c and d nodes; wh-movement, for example, arises when the dependency between an interrogative complementizer and a wh-phrase that it c-commands triggers this kind of rearrangement. But FSTGs provide some machinery that allows us to focus on the dependency itself, whatever its effects on externalization/linearization might be.

A.4 Conclusion

To conclude I reiterate some main themes that, I hope to have shown, were first brought out by investigations of the Chomsky hierarchy, but transcend many of the details and idealizations shared by all the formalisms that featured in that early work.

First, the notion of intersubstitututability of subexpressions, or categorization (in the broad sense incorporating the way FSGs assign forward sets to string, for example), is tightly related to the very idea of a grammar itself. Grammar formalisms differ in the kinds of subexpressions that they categorize (e.g. strings, trees) and in the ways that they compose these subexpressions (e.g. prefix-suffix combinations, infix-circumfix combinations), but this composition is mediated by categorization.

Second, any interesting system of categorization involves isolating out the properties of a subexpression that affect its combinatory potential, and those that don’t; those properties that need to be remembered or tracked, and those that can be safely ignored or forgotten. In the boring extreme case where everything

---

8 See e.g. Kobele et al. (2007); Kobele (2011); Graf (2011, 2018) for work connecting FSTGs to minimalist syntax along these lines.
Figure A-6: Comparison of the strictly-local and finite-state perspectives on the construction of the two trees in (A.7); one tree with a licensing c-commanding c for the d (top) and one without (bottom). A strictly-local tree grammar (SLTG) only assesses a local configuration of nodes, as indicated by the “tree bigrams” shown inside dotted lines, and so an SLTG cannot distinguish between the top and bottom trees. A finite-state tree grammar (FSTG), however, can distinguish between partial trees (grown top-down) that contain a licensing c and those that do not, for example by associating the former with the set of states \{Q\} and the latter with \{P\}, as indicated. Compare with Figure A-4.
is remembered and nothing is forgotten, a grammar reduces to a list of stored complete expressions (recall Figure 6); at the other extreme, a grammar that remembers nothing treats all subexpressions interchangeably, and therefore generates a set of expressions that exhibits no regularities. An interesting grammar is one that sits in between these two extremes, yielding constrained productivity.

Finally, these ideas are not only widely-applicable in principle, but remain at the core of other grammar formalisms beyond those on the classical Chomsky hierarchy, some of which I reviewed in this supplement, and which can be seen as minor variations on the original themes. In particular, the notion of strict locality — a very simple instance of categorization that is tied closely to surface symbols — provides a path to new kinds of grammars (the tree grammars of §A.2 and §A.3) that go beyond CFGs while maintaining the linguistic interpretability that is lost in the higher regions of the Chomsky hierarchy (§4). This paints a picture that is somewhat more optimistic about lasting contributions of the Chomsky hierarchy than linguists have generally been since the 1960s — not more optimistic about the role string-generating grammars can play in linguistic theory, but more optimistic about the role that insights gleaned from the careful study of string-generating grammars can play in an understanding of any kind of grammar.

References


