Strict Locality in Syntax

Kenneth Hanson
Stony Brook University

ABSTRACT

Lexical selection, functional hierarchies, and adjunct ordering are arguably distinct parts of syntax, yet are surprisingly similar in their computational properties. All three fall within the formal class *strictly local* (SL) and thus are maximally simple. Many phonological patterns are also SL, motivating a more detailed comparison. Towards this end, I develop a model based on *command strings* (Graf & Shafiei 2019) which allows syntactic and phonological grammars to be visualized using finite-state automata. Using this model, I show that the same basic patterns allowed within SL occur in both domains.

1 Introduction

One major goal of linguistic theory is to account for restrictions on the locality of linguistic dependencies: over what distances may they hold, and under what conditions? Computational complexity provides a natural and insightful way to describe such restrictions. Recent work has accumulated substantial evidence that linguistic patterns fall within very simple classes of formal languages, known as *subregular* languages. Among the subregular classes implicated for natural language is the class of *strictly local* (SL) languages, which have been used to model local phonotactics (Heinz 2018) as well as lexical selection in syntax (Graf 2018).

The SL languages encompass patterns which can be characterized by a finite set of permitted substructures, and are tied with the *strictly piecewise* (SP) languages for being the simplest non-trivial subregular class. As I argue in this paper, the class of SL phenomena in syntax is not limited to lexical selection, but also includes functional hierarchies, such as T < (Neg) < (Perf) < (Prog) < (Pass) < *v* of Adger (2003), as well as adjunct ordering restrictions, as in (1).

(1) Adjective ordering in English
  a. cute little spotted puppy
  b. ? little cute spotted puppy
  c. ? cute spotted little puppy
  d. ?? little spotted cute puppy

The primary purpose of this paper is to demonstrate the fundamental formal similarity of local syntactic patterns in syntax, and by virtue of their membership in the class of SL patterns, their similarity to local phonotactic patterns. So then, what exactly is an SL pattern? Intuitively, it is one in which contiguous chunks of structure of finite size are assembled to produce larger structures. For example, suppose we have an SL string pattern which allows the length-2 substrings ○⋆ and ⋆○. A string is well-formed according to this pattern iff all of its length-2 substrings are in the set {○⋆,⋆○}. Such strings include ○⋆○, ○⋆○⋆, ○⋆○⋆○, and so on. If we add the substrings ○● and ●○, then the pattern is expanded to include strings like ○●○●○ and ○⋆○●○⋆○●. By replacing the meaningless symbols in this example with what we take to be the units of linguistic structure, such as phonetic segments or syntactic heads, we have a simple yet useful formal model of language.

To see how this works, first consider an exceptionally simple phonotactic pattern: CV alternation and no other restrictions. This pattern is characterized by the substrings CV and VC. We say that the (finite) set {CV, VC} is an SL-2 grammar since it contains only substrings of length 2. By composing these substrings we can produce the strings CVC, CVCV, CVCVC, and so on. The set of all such strings (infinite, in this case) that the grammar generates is an SL-2 language.

Syntax also exhibits several SL patterns, which will be discussed in depth in Section 3. Let me give a brief sketch, beginning with functional hierarchies. A well-known example is the hierarchy of English clausal auxiliaries, in which negative *not*, perfect *have*, progressive *be*, and passive *be* can be used in any combination, as long as they occur in said order (Adger 2003). Abstracting away from movement, the hierarchy is modeled by the SL-2 grammar in (2). The grammar can be understood as saying: T may be followed by Neg, T may be followed by Perf, Neg may be followed by Perf, and so on. Such a grammar encodes the properties of strict ordering and optionality without the need for every head to be projected in every clause, as in cartographic proposals.

(2) SL-2 grammar for English clausal hierarchy

$$\left\{\begin{array}{lllll}
\text{T Neg} & & & & \\
\text{T Perf} & \text{Neg Perf} & & & \\
\text{T Prog} & \text{Neg Prog} & \text{Perf Prog} & & \\
\text{T Pass} & \text{Neg Pass} & \text{Perf Pass} & \text{Prog Pass} & \\
\text{T } v & \text{Neg } v & \text{Perf } v & \text{Prog } v & \text{Pass } v
\end{array}\right\}$$

Similar observations can be made for adjunct ordering restrictions. A simple example for English adjectives was shown in (1). It is not clear whether such facts should be handled within syntax. If we decide to do so, then from a formal perspective this is not a problem because adjunct orders are also SL-2. If we identify a set of slots, call them Adj_1, Adj_2, and Adj_3, etc., then we can state the pattern as in (3).

(3) SL-2 grammar for English adjective ordering

$$\left\{\begin{array}{lll}
\text{Adj}_1\ \text{Adj}_2 & & \\
\text{Adj}_1\ \text{Adj}_3 & \text{Adj}_2\ \text{Adj}_3 & \\
\text{Adj}_1\ \text{Adj}_4 & \text{Adj}_2\ \text{Adj}_4 & \text{Adj}_3\ \text{Adj}_4 \\
\ldots & &
\end{array}\right\}$$

The formal similarity of adjunct ordering and functional hierarchies has inspired syntacticians to treat the former as instances of the latter (Cinque 1999, et seq.). But such an approach presents several problems, as detailed by Larson (2021), who instead proposes to treat adjunct ordering as a form of selection. The present perspective suggests a different interpretation. Functional hierarchies need not be treated as a subtype of selection, nor adjunct ordering as a subtype of either. Rather, all three phenomena are instances of the same type of abstract pattern: a strictly local pattern. As mentioned earlier, local phonotactic patterns are also SL. Thus, the full class of phenomena that we should be comparing extends beyond just syntax.

In this paper, I present a detailed analysis of strictly local patterns in syntax, comparing them to those in phonology. To do so, I develop a formal model of syntax based on Minimalist Grammars (MGs, Stabler 1997) and command strings (Graf & Shafiei 2019) which allows us to visualize syntactic patterns as finite-state automata. We will see that both local syntactic and phonological dependencies are essentially alike in their formal character, beyond the mere fact that they are SL.

These findings have several consequences. First, the fact that such phenomena are SL means that they are in principle easily learned (c.f. Lambert *et al.* 2021) and therefore do not need to be provided by Universal Grammar, a welcome result for Minimalist theory. Second, the abstract unity of linguistic patterns suggests that, rather than attempting to fold one empirical phenomenon into another, we should instead focus on how each instantiates what is made possible by the computational machinery that underlies all of them. It turns out that most long-distance linguistic dependencies fall in the class *tier-based strictly local* (TSL), a generalization of SL (Graf 2022a). This state of affairs is reminiscent of the idea from Minimalism that the Move operation is a subcase of Merge (Chomsky 2004). Thus, it appears that there is something truly fundamental about language which is expressed by the computational concept of strict locality.

The remainder of this paper is structured as follows. In Section 2, I introduce the formal definition for the SL string languages and show how they can be represented as finite-state automata. Next, I introduce the command string-based model of syntax and use it to analyze lexical selection, functional hierarchies, and adjunct ordering (Section 3). From there, we will discuss some questions that arise from the present investigation (Section 4). Section 5 concludes.

2 Strictly Local String Languages and Their FSA Representations

In this section, we build on the intuitive understanding of strictly local languages developed in the introduction with a more detailed example from phonology. As we will see in Section 3, the same abstract patterns that appear in this example also arise in syntax.

We begin with a few notes on formal languages. A *formal language* is a (possibly infinite) set of objects such as strings or trees. To be more precise, we use the term *string language* for a set of strings and *tree language* for a set of trees. The claim that local dependencies are SL applies when phonological and morphological patterns are represented as strings, and syntactic patterns as trees (Graf & Heinz 2015; Graf 2022a). This does not mean that other possible representations (such as autosegmental representations in phonology) do not have their own merits, but the strong parallel that emerges under this view makes it particularly compelling.[1]

[1]The reader may be aware of the fact that syntactic patterns extend into the mildly context-sensitive region of the Chomsky hierarchy when modeled with surface strings. But the vast majority of logically possible patterns in this class are completely unattested; indeed, most patterns expressible with the regular languages do not occur in syntax. Thus, one advantage of the tree-based model

A *strictly k-local* (SL-*k*) string language is characterized by the following property: a string is ruled out if it contains any banned substrings of length *k*; equivalently, it is ruled in if every such substring is licit. A language is *strictly local* if it is strictly *k*-local for some positive integer *k*. A negative SL-*k* grammar is the set of forbidden substrings, and a positive SL-*k* grammar is its complement, the set of permitted substrings. This set is notated *G*, a mnemonic for *grammar*, and must be finite. In this paper, I will present all grammars in their positive form.[2]

Earlier, we encountered an SL-2 grammar for simple CV alternation. Now we will look at a more interesting (albeit still simplified) model of Japanese phonotactics. Japanese has a basic syllable template (C) – (j) – V – (N), where 'C' stands for a consonant, 'V' for a vowel, 'j' for a palatal glide, and 'N' for a nasal coda. Some example words (in phonemic representation) include *aoi* (VVV) 'blue', *kotowaza* (CVCVCVCV) 'proverb/saying', and *sjunkan* (CjVNCVN) 'moment'. The set of all words that can be built from one or more iterations of this syllable template is an SL-2 language. We know this because each individual symbol determines which symbols may follow. For example, 'C' may be followed by 'j' or 'V', 'N' by 'C'/'j'/'V', and so on. Thus, the grammar includes length-2 substrings such as Cj, CV, and NC, but not substrings such as CC or jN.[3]

There is one more detail that we need to handle, which is what can happen at the edges of a string. For example, we cannot have a word of the form NCVC (made up of the illicit syllables N and CVC), even though the substrings NC, CV, and VC are all licit. Instead, we must explicitly list which symbols can occur at the beginning and end of the string. We do this by adding the right and left edge markers $\rtimes$/$\ltimes$ to each string, for example, $\rtimes$NCVC$\ltimes$. Then, we can include $\rtimes$C but not $\rtimes$N in the grammar since a word may start with C but not N. Similarly, we include V$\ltimes$ but not C$\ltimes$ since a word can end with V but not C. The full grammar is given in (4).

(4) SL-2 grammar for Japanese phonotactics

$$G = \left\{ \begin{array}{lllll} \rtimes\mathrm{C} & & & \mathrm{VC} & \mathrm{NC} \\ \rtimes\mathrm{j} & \mathrm{Cj} & & \mathrm{Vj} & \mathrm{Nj} \\ \rtimes\mathrm{V} & \mathrm{CV} & \mathrm{jV} & \mathrm{VV} & \mathrm{NV} \\ & & & \mathrm{VN} & \\ & & & \mathrm{V}\ltimes & \mathrm{N}\ltimes \end{array} \right\}$$

Before proceeding further, let us walk through the full process of determining whether a string is a member of an SL-*k* language. Examples of well-formed and ill-formed strings are given in (5). First, we add edge markers, then we check whether each of the length-2 substrings is a member of the grammar. In the first string, every substring is indeed in the grammar, so the string is included in the language. On the other hand, the substring jN in the second string is not in the grammar, so the string is excluded.

is that it allows us to place a much tighter bound on the range of syntactic patterns, allowing some context-free and context-sensitive patterns over surface strings while ruling out the rest.

[2]See Lambert *et al.* (2021) for a more detailed description of SL and related subregular classes and their correspondence to different types of mathematical logic.

[3]The full pattern, which includes geminate consonants, is also SL-2, as is the pattern over surface representations, which includes several types of assimilation.

(5) a. length-2 substrings of CjVNCVN (✓ e.g. *sjunkan* ‘moment’)

⋊C	Cj	jV	VN	NC	CV	VN	N⋉
✓	✓	✓	✓	✓	✓	✓	✓

b. length-2 substrings of C**VjN**CVN (✗ nonexistent)

⋊C	CV	Vj	**jN**	NC	CV	VN	N⋉
✓	✓	✓	✗	✓	✓	✓	✓

The reason that we apply the SL-2 grammar to patterns like CjVNCVN rather than actual Japanese words like *sjunkan* is that formal grammars traditionally deal with atomic symbols, not categories of symbols. To recognize the latter, we can compile the abstract categories C/V/N into the concrete symbols that they represent, replacing the substring CV, for example, with the substrings *ka, ki, ku, ta, ti, tu*, etc. The resulting grammar is considerably larger, but it is no more complex in computational terms—it is still only SL-2. The advantage to using the abstract grammar is that it is far more compact while still encapsulating the relevant patterns.

That said, the structure of an SL grammar when viewed as a set is still rather opaque. We can understand it more intuitively using its finite-state automaton (FSA) representation. An FSA is an abstract machine which reads a string one symbol at a time, and is visualized as a directed labeled graph. As an example, the FSA for the grammar from above is shown in Figure 1. The machine begins in the state labeled ‘Start’, as indicated by the unlabeled arrow. Upon reading a symbol, the machine advances to the state indicated by an edge labeled with the same symbol. If, after reading the entire string, the machine is in a final state (marked with double circles) then the string is accepted, otherwise it is rejected. For example, the string *sjunkan* takes the machine to state N, which is final state, so it is accepted.

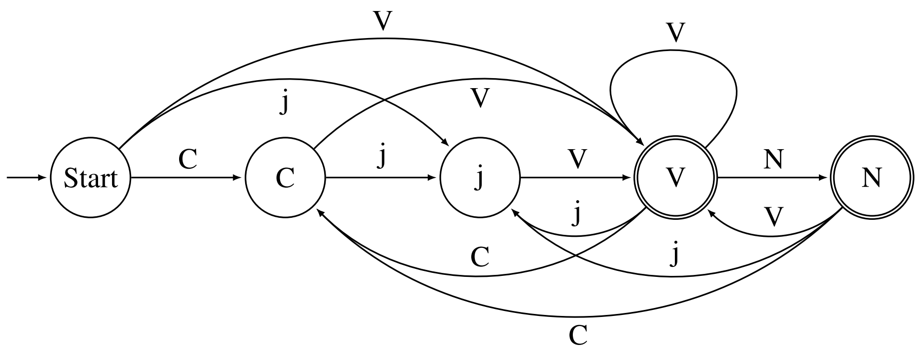

Figure 1: SL-2 FSA for Japanese phonotactics.

Every path through the FSA which ends in a final state corresponds to a string in the language that it implements. While the full class of languages implementable by FSAs (the regular languages) is much larger than SL, all FSAs here represent SL languages. In an SL-k FSA, each state represents the last $k-1$ symbols which have been read. Thus, for an SL-2 language, each state is named with a single symbol, and each pair of state and outgoing edge correspond to a substring of the grammar.

One interesting property of the SL languages is that, while they are extremely restricted in expressivity, evn an SL-2 FSA exhibit all of the basic substructures that are possible in the full class of FSAs: sequences, branches, and loops. For example, the syllable template in the current example corresponds to a sequence of states augmented with edges that bypass the optional elements, and the transition from one syllable to the next corresponds to edges that loop back to an earlier state. More complex phonotactic restrictions would also require branching paths, for example, to handle assimilation between adjacent segments. Languages like English with a more complex syllable structure require a somewhat larger window size, but k values larger than about 5 are hardly ever needed, and they also do not introduce any new types of graph structures.

So, we see that in spite of their simplicity, even SL-2 languages can do quite a lot. Furthermore, we will see that all of the same structures found in phonotactic patterns also occur in the syntactic patterns discussed in the next section, suggesting that the formal parallels between syntax and phonology run very deep.

3 Strictly Local Patterns in Syntax

Our next goal is to show that local dependencies in syntax are also SL, and that the exemplify the same abstract patterns found in phonology. To do this, we will model syntactic dependencies as strings which encode structural relations among elements in the syntactic structure, known as *command strings* (Graf & Shafiei 2019). This allows us to visualize syntactic patterns using FSAs just as we did for phonotactic patterns; tree grammars are not easily visualized in this manner, among other issues (see Section 4). This way of thinking about syntactic dependencies was inspired by Kobele (2021), which also treats the paths through a grammar as a kind of graph.

But first, we need a formal system that defines the class of syntactic structures, which make up a tree language. Following precedent in subregular syntax (Graf 2018, et seq.), I will make use of Minimalist Grammars (MGs, Stabler 1997), which are a formalization of ideas from Chomsky's (1995) Minimalist Program.

3.1 Minimalist Grammars and Derivation Trees

MGs consists of two operations, Merge and Move, along with a lexicon annotated with features to guide these operations. We will use the MG formalism to generate trees which record the steps of the derivation itself, called *derivation trees*, which can in turn be mapped to the corresponding phrase structure trees. In this paper, I use a particularly compact type of derivation tree in which the arguments of each head appear as its children (cf. Graf & Kostyszyn 2021).

An example for the sentence "The cat chased the rat" is shown in Figure 2. On the left is a phrase structure tree (using bare phrase structure, but with X′-style labels for the internal nodes), and on the right is the corresponding derivation tree with MG feature annotation. The rightmost child of each node in the derivation tree is its complement; others are specifiers. Positive features are 'selector' features and negative features are 'category' features. For example, the feature specification for the v head is $\langle \text{lv}^- \ \text{D}^+ \ \text{V}^+ \rangle$, encoding the fact that v takes a DP specifier and a VP complement. I omit all movement-related features since they are not our focus.[4]

[4]I have flipped the usual MG feature ordering for expository convenience. Normally, features are

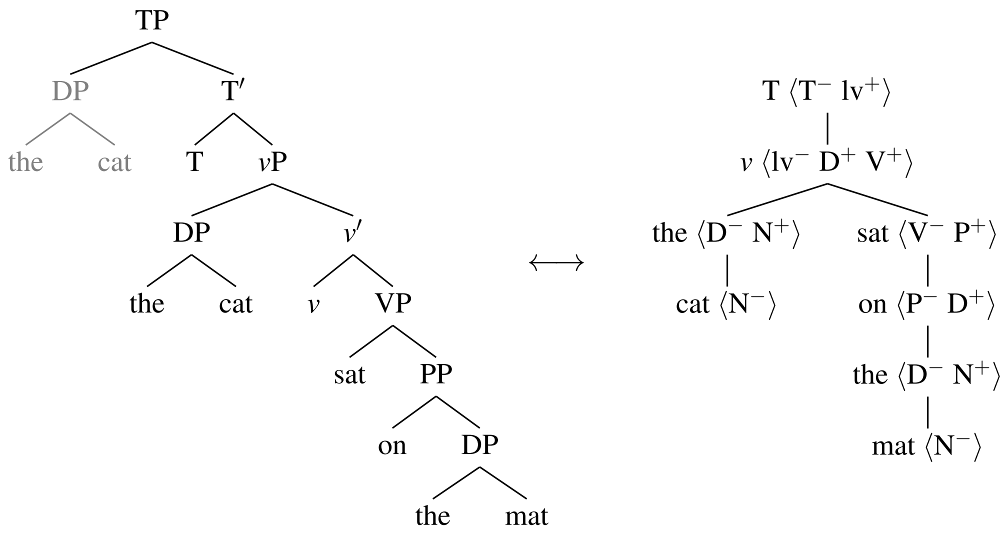

Figure 2: Phrase structure tree and corresponding derivation tree.

Some additional comments are in order. First, all elements in the derivation tree appear in their base position only. This is not a problem since we are dealing only with local dependencies. Second, the ordering among the children of a node represents a hierarchical ordering among its arguments; it does not directly encode linear order, which is instead taken to be part of the mapping to the phrase structure tree (or from the phrase structure tree to the surface string).

3.2 Command Strings and Lexical Selection

Now that we have our tree language, we will extract *string paths* from the tree which encapsulate the relevant dependencies, in this case, that between a head and its arguments. Consider again the derivation tree from above, repeated in Figure 3. The first such path begins at the root and zip-zags towards the rightmost leaf, visiting every node that dominates the leaf as well as any left siblings of those nodes—call this the main *spine* of the tree. The main spine for our example is $T \cdot v \cdot the \cdot sat \cdot on \cdot the \cdot mat$. Additionally, let us assume that each left branch begins a spine of its own. In this example, the only other spine is $the \cdot cat$.

These strings are a type of *command string*, or *c-string*, and represent a derivational ordering of nodes in a tree (Graf & Shafiei 2019). C-strings were originally developed to model long-distance dependencies, and are motivated by the fact that, roughly speaking, syntactic dependencies only hold among c-commanding nodes. They can just as easily be used to model local dependencies like selection. Furthermore, c-string grammars can be compiled into a tree grammar which recognizes the derivation tree directly; see Graf & De Santo (2019) for details. Thus, the c-string approach can be seen as a method for decomposing a tree grammar into a more human-friendly format.

ordered according to their checking order in a bottom up derivation. From the top-down perspective taken here, the reverse order is more intuitive.

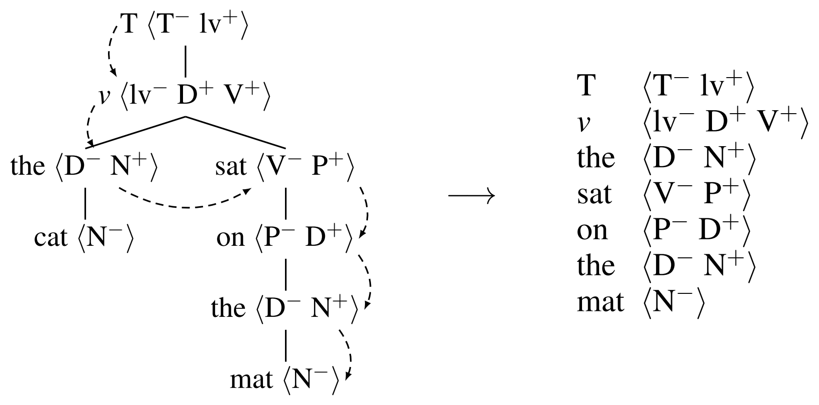

Figure 3: Derivation tree and c-string. Left: the dashed line represents the c-string for the main spine of the tree. Right: the c-string itself.

The ordering in a c-string is a mix of c-command and m-command, since the head precedes both its complement and specifier. This turns out to be convenient for our purposes. If a node has n selector features, then we need only to look at the next n nodes to confirm that their category features match (treating left branches separately, as discussed below). For example, *on* $\langle \mathrm{P}^- \ \mathrm{D}^+ \rangle$ must be followed by a node with category D^-, and transitive v $\langle \mathrm{lv}^- \ \mathrm{D}^+ \ \mathrm{V}^+ \rangle$ must be followed by nodes of categories D^- and V^-, in that order. Thus, for any MG lexicon with a maximum of n arguments per head, we need an SL grammar with $k = n + 1$.

Such a grammar for a fragment of English is shown in (6). As in our earlier example, the grammar abstracts away from the concrete lexical items to their categories. The first two lines, for example, indicate that a verb that takes a DP complement may be followed by a simplex D head (such as a pronoun) or a D that takes an NP complement (such as an article). Technically, all substrings should be the same length, but for reasons of readability I instead present a mixed grammar. To obtain the 'real' grammar we must pad substrings with length $< k$ on both sides with every possible combination of other symbols.[5]

(6) SL-3 grammar for lexical selection in English

$$G = \left\{ \begin{array}{l} \langle \mathrm{V}^- \ \mathrm{D}^+ \rangle \cdot \langle \mathrm{D}^- \rangle \\ \langle \mathrm{V}^- \ \mathrm{D}^+ \rangle \cdot \langle \mathrm{D}^- \ \mathrm{N}^+ \rangle \\ \langle \mathrm{P}^- \ \mathrm{D}^+ \rangle \cdot \langle \mathrm{D}^- \rangle \\ \langle \mathrm{P}^- \ \mathrm{D}^+ \rangle \cdot \langle \mathrm{D}^- \ \mathrm{N}^+ \rangle \\ \langle \mathrm{D}^- \ \mathrm{N}^+ \rangle \cdot \langle \mathrm{N}^- \rangle \\ \langle \mathrm{D}^- \ \mathrm{N}^+ \rangle \cdot \langle \mathrm{N}^- \ \mathrm{P}^+ \rangle \\ \langle \mathrm{lv}^- \ \mathrm{D}^+ \ \mathrm{V}^+ \rangle \cdot \langle \mathrm{D}^- \ \mathrm{N}^+ \rangle \cdot \langle \mathrm{V}^- \ \mathrm{D}^+ \rangle \\ \langle \mathrm{lv}^- \ \mathrm{D}^+ \ \mathrm{V}^+ \rangle \cdot \langle \mathrm{D}^- \rangle \cdot \langle \mathrm{V}^- \ \mathrm{D}^+ \rangle \\ \langle \mathrm{lv}^- \ \mathrm{D}^+ \ \mathrm{V}^+ \rangle \cdot \langle \mathrm{D}^- \ \mathrm{N}^+ \rangle \cdot \langle \mathrm{V}^- \rangle \\ \ldots \end{array} \right\}$$

[5]As pointed out to me by Bob Frank, this c-string grammar does not distinguish a head followed by its specifier and complement from a head followed by its complement and *the complement of its complement*. For example, it allows v to select a DP complement which selects a VP complement. This corner case can be avoided by enriching the c-string to distinguish complements from specifiers, something that the corresponding tree grammar does in any case.

The full grammar is quite unwieldy when presented in this format. Let us instead visualize the grammar as an FSA, as shown in Figure 4. This time, the individual states are labeled with a string of selector features, since it is these that determine which nodes may be read next. For example, suppose the FSA is in state lv^+, indicating that it has just read a T node and is looking for its *v* complement. If it reads a transitive *v* node, it will transition to state $\text{D}^+\ \text{V}^+$, which means that it is looking for a DP specifier followed by a VP complement. Upon encountering a node of category $\text{D}^-\ \text{N}^+$, the specifier dependency is fulfilled, and it moves to state V^+. The next node is a complement, so if it reads a node with features $\text{V}^-\ \text{P}^+$ then it moves to state P^+, *not* the end state. The full path through the automaton for our running example is shown in Figure 5.[6]

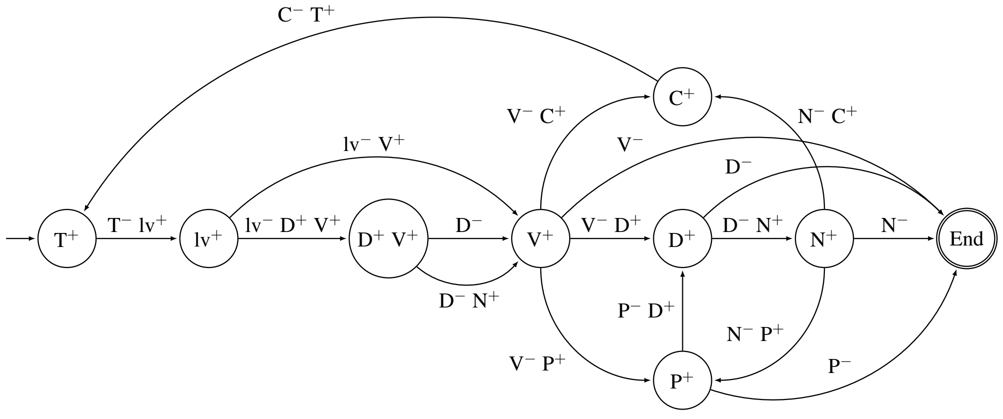

Figure 4: FSA for lexical selection in English.

Input	State
n/a	T^+
$\langle \text{T}^-\ \text{lv}^+ \rangle$	lv^+
$\langle \text{lv}^-\ \text{D}^+\ \text{V}^+ \rangle$	$\text{D}^+\ \text{V}^+$
$\langle \text{D}^-\ \text{N}^+ \rangle$	V^+
$\langle \text{V}^-\ \text{P}^+ \rangle$	P^+
$\langle \text{P}^-\ \text{D}^+ \rangle$	D^+
$\langle \text{D}^-\ \text{N}^+ \rangle$	N^+
$\langle \text{N}^- \rangle$	End

Figure 5: State path for input string “The cat sat on the mat”.

[6]As an SL-3 FSA, each state is completely determined by the most recent 2 symbols read. Technically, an SL-3 FSA should have distinct states corresponding to every possible length-2 substring. The FSA shown is the canonical FSA, in which redundant states have been merged. This makes the representation especially compact.

So, we have seen how the FSA checks selectional dependencies along the main spine of the tree. But what about the other spines? I cannot go into detail here for reasons of space, but the general idea is that the computation branches at each left child. The very same FSA is used to check the spine of that subtree, starting in a state corresponding to the selector features of its head. In the case of the previous example, this would happen at the point when the specifier labeled D^- N^+ is read, and the new computation would start at state N^+.

This was the most complex example in this paper, but the effort was worthwhile, because we can now make an important observation: not only is lexical selection SL, it displays exactly the same kinds of substructure that we observed for phonotactics. First, we observe a major sequence corresponding to the longest possible chain of distinct categories within the clause, with opportunities to skip ahead in the chain depending on the number of arguments each node takes. Next, there are points at which the computation may loop back to an earlier state, corresponding to recursive embedding of categories. Finally, there are branching paths starting at categories V and N, which allow for a larger range of possible complements.

Such a result is not necessarily to be expected in studies of computational complexity. Normally, the claim that phenomenon X is in class Y is meant as an upper bound—it does not at all imply that every possibility in class Y be attested in phenomenon X. But local linguistic dependencies do seem to make use of a large swath of the logical possibilities within SL, making it a kind of lower bound as well. This means that SL truly is a robust formal characterization for local linguistic patterns.

3.3 Functional Hierarchies and Adjunct Ordering

Having analyzed lexical selection in detail, let us return to the other examples of strictly local syntactic patterns which we encountered briefly in the introduction, beginning with functional hierarchies. A well-known example, the English clausal hierarchy, shown in (7). Various auxiliaries may occur between the T and *v* heads, each optional yet with a fixed relative position. (Some functional categories like T and *v* are not optional, though this makes them relatively uninteresting.) A sampling of the possible combinations is shown in Figure 6. To control for movement, a modal verb in inserted in T in all examples.

(7) English clausal hierarchy (Adger 2003)
T < (Neg) < (Perf) < (Prog) < (Pass) < *v*

Most syntacticians are probably familiar with the problem that functional hierarchies present for models of grammar built around selection. It feels artificial to say that Pass selects for VP, Prog for PassP or VP, Perf for ProgP/PassP/VP, and so on, in a way that just so happens to enforce this order. Instead, it may be more reasonable to assume that such hierarchies are distinct from selection. At the same time, a functional hierarchy is simply an SL pattern, which means that it can be recognized or generated by the same kind of computational device regardless of the stance one takes on this theoretical issue.

As discussed earlier, each individual element in a functional hierarchy determines which elements may come next, making it an SL-2 pattern. The grammar for the English clausal hierarchy is repeated in (8). Unlike with selection, the categories

	T	Neg	Perf	Prog	Pass	*v*+V	
John	will					eat	the pie.
John	will		have	been		eating	the pie.
John	will	not		be		eating	the pie.
The pie	will	not	have		been	eaten.	
. . .							
The pie	will	not	have	been	being	eaten.	

Figure 6: Some combinations of auxiliaries in the English clausal hierarchy.

of the nodes themselves are sufficient to describe the pattern—for example, every T head behaves alike with respect to the hierarchy, so there is no need to include their MG feature specification in the grammar.

(8) SL-2 grammar for English clausal hierarchy

$$G = \left\{ \begin{array}{lllll} \text{T Neg} & & & & \\ \text{T Perf} & \text{Neg Perf} & & & \\ \text{T Prog} & \text{Neg Prog} & \text{Perf Prog} & & \\ \text{T Pass} & \text{Neg Pass} & \text{Perf Pass} & \text{Prog Pass} & \\ \text{T } v & \text{Neg } v & \text{Perf } v & \text{Prog } v & \text{Pass } v \end{array} \right\}$$

The corresponding FSA is shown in Figure 7. The shape of this FSA is extremely similar to that of the phonotactic grammar from earlier, as both exemplify a sequential structure with optional slots. Recall how the SL grammar makes it possible to project only heads which are necessary in a given clause. In the FSA representation, this corresponds to following edges which skip ahead in the sequence.

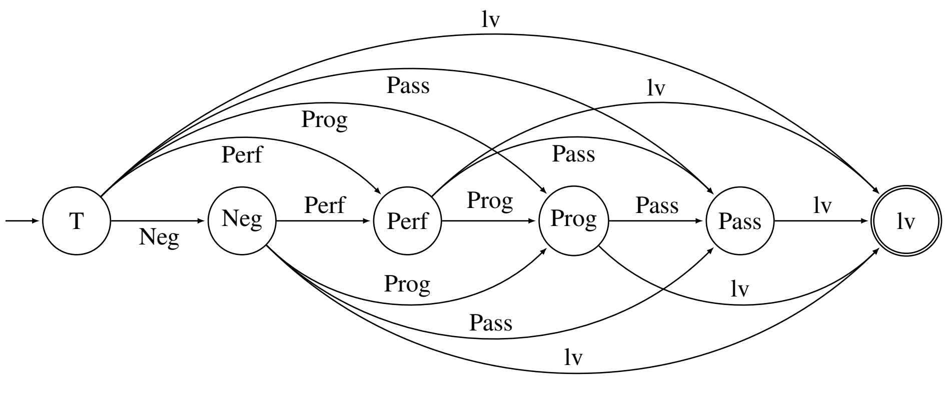

Figure 7: FSA for English clausal hierarchy

Adjunct ordering is similar to a functional hierarchy, though there are some differences. First, violations of the normal order do not sound nearly as bad as typical grammatical violations, as shown in earlier and repeated in (9). Indeed, with the right context, they may be completely natural (10). Additionally, adjuncts in general can be iterated (11), though not every adjective or adverb can do this.

(9) Violations of adjunct ordering are mildly unacceptable
- a. cute little spotted puppy
- b. ? little cute spotted puppy
- c. ? cute spotted little puppy
- d. ?? little spotted cute puppy

(10) Alternative orders are available

a.	cute little puppy	(attractiveness < size)
b.	big ugly bird	(size < attractiveness)

(11) Adjuncts are iterable
- a. cute little spotted puppy
- b. cute cute little spotted puppy
- c. cute cute cute little spotted puppy

As mentioned at the outset, it is not clear whether adjunct ordering should be treated as belonging to syntax at all. While common, they do not appear to be universal; all of the orders in (9) are fine in German, for example (Thomas Graf, p.c.). In cases where there is a preferred order, it is difficult to pin down what exactly it is. Scontras *et al.* (2017) argue, backed by experimental evidence, that rather than there being a fixed ranking among categories like 'size', 'shape', and 'material', as in traditional descriptions, there is a general principle in which more subjective descriptors precede less subjective ones. This is an appealing idea, since it allows for the order to be dynamically determined by pragmatic factors. Then, grammar needs only to implement this order, whatever it may be. Larson (2021) proposes a way to do this via selection; here, we will simply use an SL-2 grammar. We modify the earlier grammar to include iteration by adding substrings repeating each symbol twice, as shown in (12); the new substrings are highlighted.

(12) SL-2 grammar for English adjective ordering, with iteration

$$\left\{\begin{array}{llll} \textbf{Adj}_\mathbf{1}\ \textbf{Adj}_\mathbf{1} & & & \\ \text{Adj}_1\ \text{Adj}_2 & \textbf{Adj}_\mathbf{2}\ \textbf{Adj}_\mathbf{2} & & \\ \text{Adj}_1\ \text{Adj}_3 & \text{Adj}_2\ \text{Adj}_3 & \textbf{Adj}_\mathbf{3}\ \textbf{Adj}_\mathbf{3} & \\ \text{Adj}_1\ \text{Adj}_4 & \text{Adj}_2\ \text{Adj}_4 & \text{Adj}_3\ \text{Adj}_4 & \textbf{Adj}_\mathbf{4}\ \textbf{Adj}_\mathbf{4} \\ \ldots & & & \end{array}\right\}$$

These new substrings correspond to self loops on each state in the FSA, as shown in Figure 8. As Larson (2021) notes, an adjunct ordering is a *preorder*, analogous to the integers ordered by the $\leq$ relation. A preorder is both reflexive and transitive; reflexivity corresponds to self-loops on each state in the FSA ($X \leq X$), and transitivity means that every path between two elements is accompanied by an edge that connects them directly (if $X \leq Y$ and $Y \leq Z$ then $X \leq Z$). So, we see that the distinctive structure of adjunct ordering is just one instance of the class of SL-2 structures, which are in turn part of the larger class of SL-*k* structures.

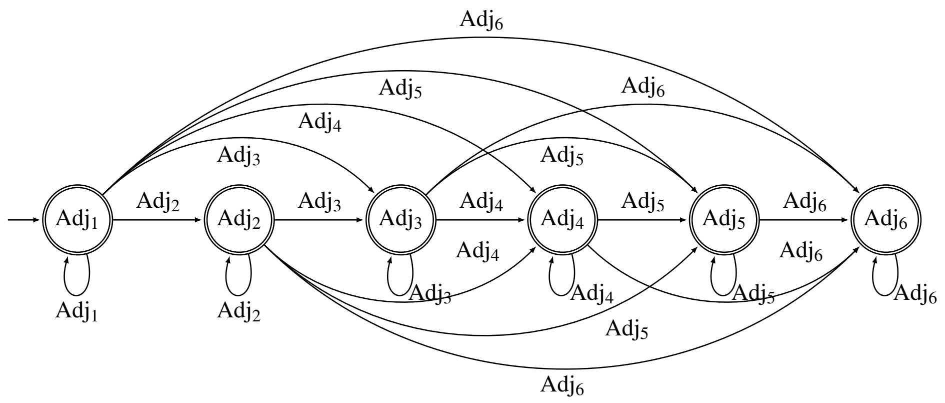

Figure 8: FSA schematic for adjunct ordering.

4 Discussion

To briefly summarize what we have seen in this paper, functional hierarchies and adjunct orderings are SL-2 patterns, while lexical selection, assuming a maximum of two arguments per head, is SL-3. In addition, by looking at their FSA representations, we have seen how these patterns exhibit the same sorts of structures found in phonological patterns. At this point, I wish to address some questions that the reader may have:

1. How could we model syntactic dependencies using an SL tree grammar rather than c-strings?
2. Are there other local patterns in syntax worth distinguishing from those analyzed here?
3. Is a single grammar which models all of these phenomena still SL?
4. How do long-distance dependencies fit into the picture?

1. SL tree grammars. An SL tree grammar works much like an SL string grammar. Instead of a set of substrings of a certain length, we define a set of subtrees of a certain height (the width of the subtrees may vary), which can be assembled in the same manner. The use of SL tree grammars goes back at least to Rogers (1997), and they are also extremely intuitive. However, there are some technical and practical problems with extending them to long-distance dependencies which do not affect c-strings; see Hanson (2023) for an example relating to coordination. At present, it seems that both approaches merit further exploration.

2. Other local patterns in syntax. I believe that certain "last resort" operations, such as do-support, are sufficiently distinct from ordinary selection that they would be interesting to analyze in the present framework—see Kobele (2021) regarding how to construct an appropriate MG lexicon—but for reasons of space I cannot do this here.

3. Is the full grammar SL? The answer is a tentative yes (again, abstracting away from inherently long-distance phenomena such as movement), though there are potential pitfalls. For example, classic MGs do not include an adjunction operation, and depending on how one is added, the result can be a tree language in which Merge is no longer SL, as was shown by Graf (2018). In any case, with the specific type of derivation tree used in this paper, I believe that both selection and adjunction remain SL. I leave a proper investigation of this issue to future research.

4. Long-distance dependencies. It is an interesting fact that most long distance dependencies can be modeled by a generalization of the SL languages known as the *tier-based strictly local* (TSL) languages. In the case of syntax, this includes movement (Graf 2022b), case (Vu *et al.* 2019; Hanson 2023), and agreement (Hanson n.d.). For some phenomena, somewhat more powerful subregular classes seem to be necessary—see Graf (2022a) for a summary. It would be interesting to study some of the long-distance phenomena discussed in the subregular syntax literature through their FSA representations as we did here.

5 Conclusion

The SL languages occupy one of the lowest positions in the hierarchy of formal languages—the bare minimum of complexity needed to do much of anything at all. Yet, in spite of this simplicity, they can in fact do quite a lot. As we have seen, local syntactic patterns are extremely similar to one another, as well as those in phonology. Furthermore, most long-distance dependencies are TSL, and hence also related, suggesting that strict locality is a fundamental aspect of linguistic patterns, both within and beyond syntax.

While the perspective taken here mirrors current syntactic theory in many ways, there are important differences. For example, Chomsky's (1995) Merge is set-theoretic in nature while SL and TSL, being based in formal language theory, are more closely aligned with Chomsky's earlier work (Chomsky 1956; Chomsky 1959). It may be time to revisit this older perspective as we attempt to understand the nature of linguistic structure.

Acknowledgments

This work was partly supported by the National Science Foundation under Grant No. BCS-1845344. I thank Thomas Graf, Jeff Heinz, and audiences at SYNC 2023 and CLS 2023 for their comments and feedback.

References

Adger, D. 2003. *Core syntax: A minimalist approach*. Oxford University Press.

Chomsky, N. 1956. Three models for the description of language. *IRE Transactions on Information Theory* 2. 113–124.

Chomsky, N. 1959. On certain formal properties of grammars. *Information and Control* 2. 137–167.

Chomsky, N. 1995. *The Minimalist Program*. MIT Press.

Chomsky, N. 2004. Beyond explanatory adequacy. In *Structures and Beyond: The Cartography of Syntactic Structures*, ed. by A. Belletti, volume 3, p. 104–131. Oxford University Press.

Cinque, G. 1999. *Adverbs and functional heads: A cross-linguistic perspective*. Oxford University Press.

Graf, T. 2018. Why movement comes for free once you have adjunction. *Proceedings of CLS* 53. 117–136.

Graf, T. 2022a. Subregular linguistics: bridging theoretical linguistics and formal grammar. *Theoretical Linguistics* 48. 145–184.

Graf, T. 2022b. Typological implications of tier-based strictly local movement. In *Proceedings of the Society for Computation in Linguistics 2022*, 184–193.

Graf, T., & A. De Santo. 2019. Sensing tree automata as a model of syntactic dependencies. In *Proceedings of the 16th Meeting on the Mathematics of Language*, 12–26, Toronto, Canada. Association for Computational Linguistics.

Graf, T., & J. Heinz. 2015. Commonality in disparity: The computational view of syntax and phonology. Slides of a talk given at GLOW 2015, April 18, Paris, France.

Graf, T., & K. Kostyszyn. 2021. Multiple wh-movement is not special: The subregular complexity of persistent features in Minimalist Grammars. In *Proceedings of the Society for Computation in Linguistics 2021*, 275–285, Online. Association for Computational Linguistics.

Graf, T., & N. Shafiei. 2019. C-command dependencies as TSL string constraints. In *Proceedings of the Society for Computation in Linguistics 2019*, 205–215.

Hanson, K. 2023. A TSL analysis of Japanese case. In *Proceedings of the Society for Computation in Linguistics 2023*.

Hanson, K. n.d. A computational perspective on the typology of agreement. In preparation.

Heinz, J. 2018. The computational nature of phonological generalizations. *Phonological Typology, Phonetics and Phonology* 126–195.

Kobele, G. 2021. Minimalist grammars and decomposition.

Lambert, D., J. Rawski, & J. Heinz. 2021. Typology emerges from simplicity in representations and learning. *Journal of Language Modelling* 9. 151–194.

Larson, R. K. 2021. Rethinking cartography. *Language* 97. 245–268.

Rogers, J. 1997. Strict LT_2 : Regular :: Local : Recognizable. In *Logical Aspects of Computational Linguistics: First International Conference, LACL '96 (Selected Papers)*, ed. by C. Retoré, volume 1328 of *Lectures Notes in Computer Science/Lectures Notes in Artificial Intelligence*, 366–385. Springer.

Scontras, G., J. Degen, & N. D. Goodman. 2017. Subjectivity predicts adjective ordering preferences. *Open Mind* 1. 53–66.

Stabler, E. P. 1997. Derivational minimalism. In *Logical Aspects of Computational Linguistics*, ed. by C. Retore. Springer.

Vu, M. H., N. Shafiei, & T. Graf. 2019. Case assignment in TSL syntax: A case study. In *Proceedings of the Society for Computation in Linguistics 2019*, 267–276.