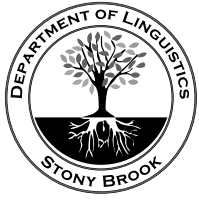
Strict Locality in Syntax

Kenneth Hanson

Stony Brook University

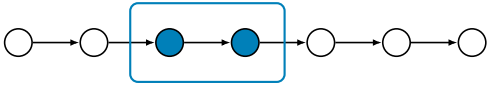


CLS 59

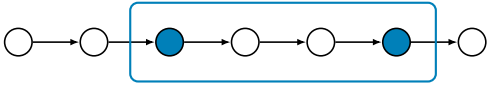
April 28, 2023

What is Locality?

Local ⟶ finitely bounded

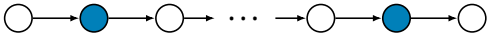


(window size 2)



(window size 4)

Long-distance ⟶ no finite bound



Local Patterns in Syntax

1. Lexical/category selection

Ex. Some verbs select a DP complement, others select a PP

2. Functional hierarchies

Ex. We might$_{T}$ have$_{Perf}$ been$_{Prog}$ being$_{Pass}$ watched.

3. Adjunct ordering

Ex. ✓ big red truck ?? red big truck

Can these phenomena be unified somehow?

Overview

Finitely bounded dependencies fall within the formal class of **strictly local (SL)** languages.

- In phonology: local phonotactics (Heinz 2018)
- In syntax: lexical selection (Graf 2018)

This talk: functional hierarchies and adjunct ordering are also SL.

→ Their existence is unsurprising from a computational perspective.

Overview (2)

Together, these patterns exhaust the logically possible SL patterns.

- **Functional hierarchies**: linear paths, optionality
- **Adjunct ordering**: linear paths, iteration
- **Lexical selection**: branching and looping paths

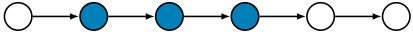
I will illustrate this using the **finite-state automaton (FSA)** representations of the SL grammars.

Proposal: SL computations are the basis for linguistic structure building across domains.

Roadmap

1. Introduction to SL
   - Examples from phonology
2. SL in syntax – intuitions
   - Lexical selection
   - Functional hierarchies
   - Adjunct ordering
3. SL in syntax – technical implementation
   - C-strings
   - Spines
4. SL as a structure-building operation

Strictly Local Languages



Strictly Local Languages

Defining characteristic: a string is well-formed iff all of its substrings of some fixed length are well-formed.

- G = set of well-formed substrings
- k = the length of the substrings
- **SL-k** SL for substrings of length k

Example: CV Alternation (SL-2)

✓ $CVCVC$ ✓ VCV ✗ $CVCCV$ ✗ $VCVV$

G = {C, V, CV, VC, C, V}

C stands for any consonant, V stands for any vowel, $ stands for beginning/end of string.

Strictly Local Languages (2)

Example: Japanese phonotactics (SL-2)

Syllable template: (C) (j) V (N)

Example words: aoi, kotowaza, sjunkan

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

Assimilation and geminate consonants are removed for simplicity. The full grammar is also SL-2.

Strictly Local Languages (2)

Example: Japanese phonotactics (SL-2)

Syllable template: (C) (j) V (N)

Example words: aoi, kotowaza, sjunkan

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

Intuition: check each substring in a moving window

$ s j u n k a n $

Assimilation and geminate consonants are removed for simplicity. The full grammar is also SL-2.

Strictly Local Languages (2)

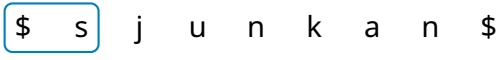
Example: Japanese phonotactics (SL-2)

Syllable template: (C) (j) V (N)

Example words: aoi, kotowaza, sjunkan

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

Intuition: check each substring in a moving window



Assimilation and geminate consonants are removed for simplicity. The full grammar is also SL-2.

Strictly Local Languages (2)

Example: Japanese phonotactics (SL-2)

Syllable template: (C) (j) V (N)

Example words: aoi, kotowaza, sjunkan

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

Intuition: check each substring in a moving window

$ s j u n k a n $

Assimilation and geminate consonants are removed for simplicity. The full grammar is also SL-2.

Strictly Local Languages (2)

Example: Japanese phonotactics (SL-2)

Syllable template: (C) (j) V (N)

Example words: aoi, kotowaza, sjunkan

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

Intuition: check each substring in a moving window

$ s j u n k a n $

Assimilation and geminate consonants are removed for simplicity. The full grammar is also SL-2.

Strictly Local Languages (2)

Example: Japanese phonotactics (SL-2)

Syllable template: (C) (j) V (N)

Example words: aoi, kotowaza, sjunkan

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

Intuition: check each substring in a moving window

$ s j u n k a n $

Assimilation and geminate consonants are removed for simplicity. The full grammar is also SL-2.

Strictly Local Languages (2)

Example: Japanese phonotactics (SL-2)

Syllable template: (C) (j) V (N)

Example words: aoi, kotowaza, sjunkan

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

Intuition: check each substring in a moving window

$ s j u n k a n $

Assimilation and geminate consonants are removed for simplicity. The full grammar is also SL-2.

Strictly Local Languages (2)

Example: Japanese phonotactics (SL-2)

Syllable template: (C) (j) V (N)

Example words: aoi, kotowaza, sjunkan

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

Intuition: check each substring in a moving window

$ s j u n k a n $

Assimilation and geminate consonants are removed for simplicity. The full grammar is also SL-2.

Strictly Local Languages (2)

Example: Japanese phonotactics (SL-2)

Syllable template: (C) (j) V (N)

Example words: aoi, kotowaza, sjunkan

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

Intuition: check each substring in a moving window

$ s j u n k a n $

Assimilation and geminate consonants are removed for simplicity. The full grammar is also SL-2.

Strictly Local Languages (2)

Example: Japanese phonotactics (SL-2)

Syllable template: (C) (j) V (N)

Example words: aoi, kotowaza, sjunkan

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

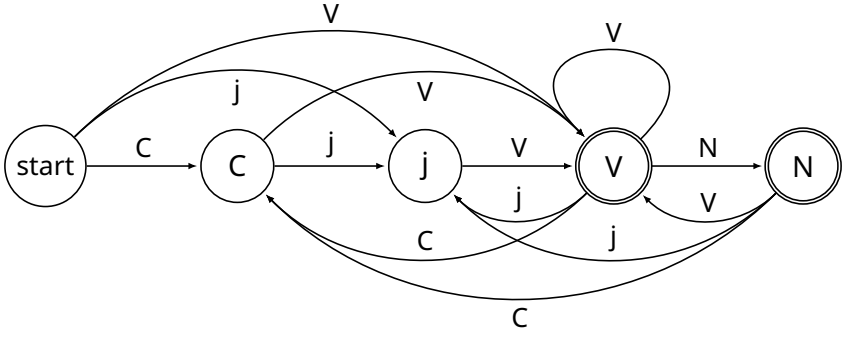
Intuition: check each substring in a moving window

$ s j u n k a n $

Assimilation and geminate consonants are removed for simplicity. The full grammar is also SL-2.

Strictly Local Languages (3)

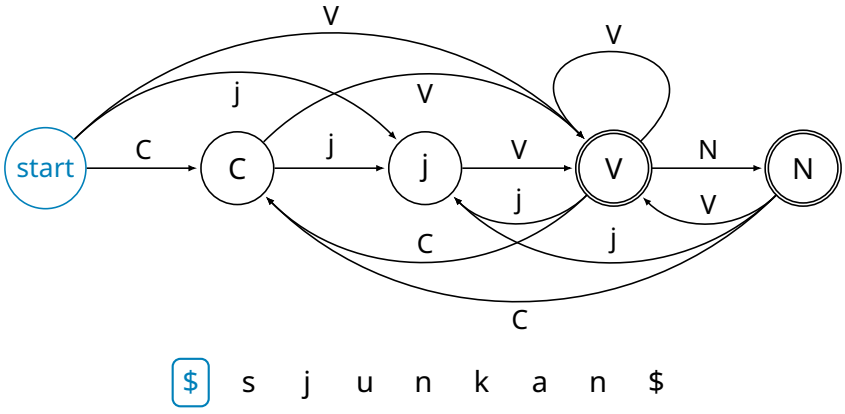
We can visualize an SL grammar using a **finite-state automaton (FSA)**.



SL is a subclass of the languages expressible by FSAs.

Strictly Local Languages (3)

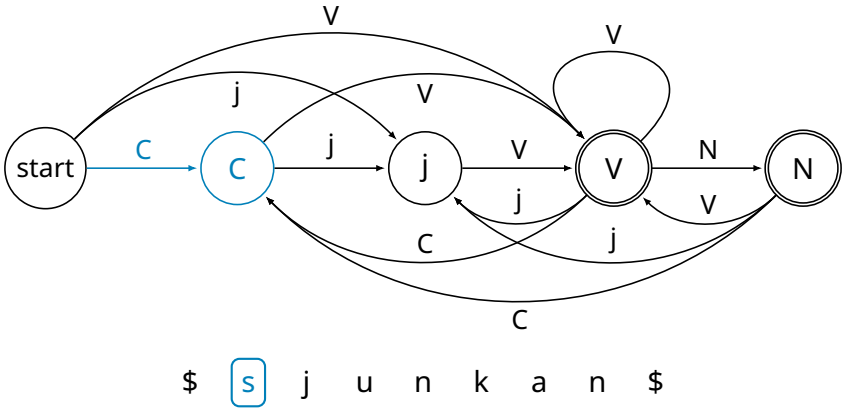
We can visualize an SL grammar using a **finite-state automaton (FSA)**.



SL is a subclass of the languages expressible by FSAs.

Strictly Local Languages (3)

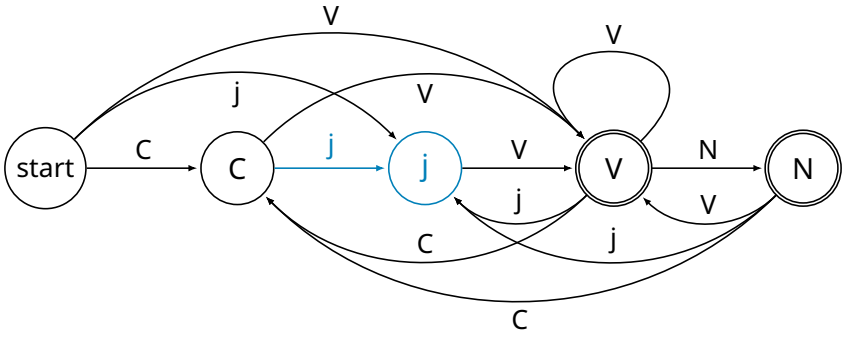
We can visualize an SL grammar using a **finite-state automaton (FSA)**.



SL is a subclass of the languages expressible by FSAs.

Strictly Local Languages (3)

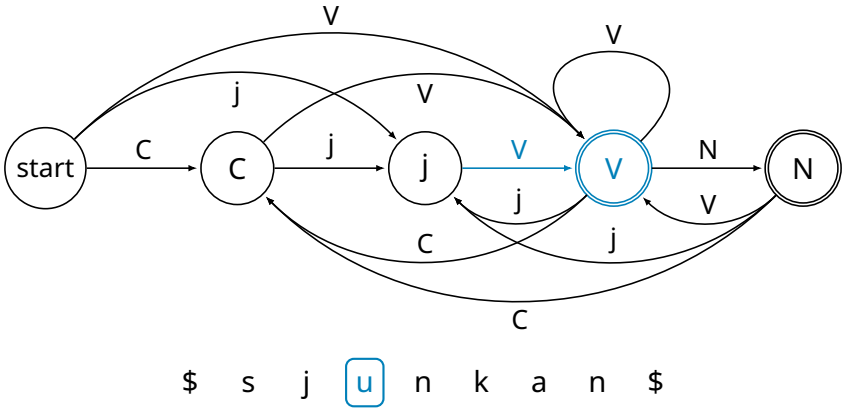
We can visualize an SL grammar using a **finite-state automaton (FSA)**.



SL is a subclass of the languages expressible by FSAs.

Strictly Local Languages (3)

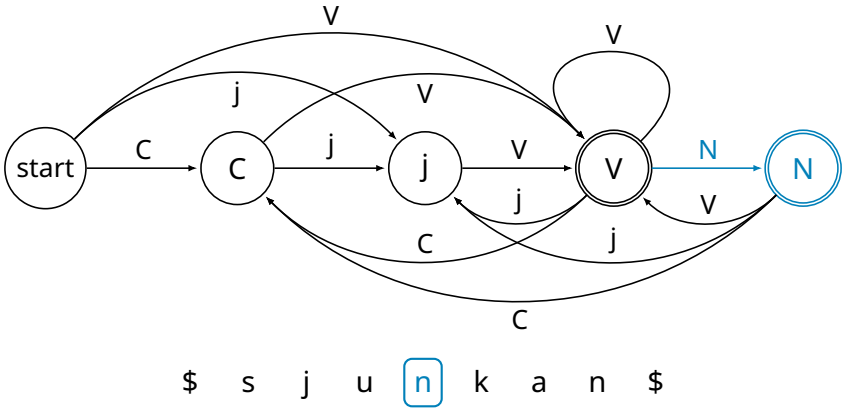
We can visualize an SL grammar using a **finite-state automaton (FSA)**.



SL is a subclass of the languages expressible by FSAs.

Strictly Local Languages (3)

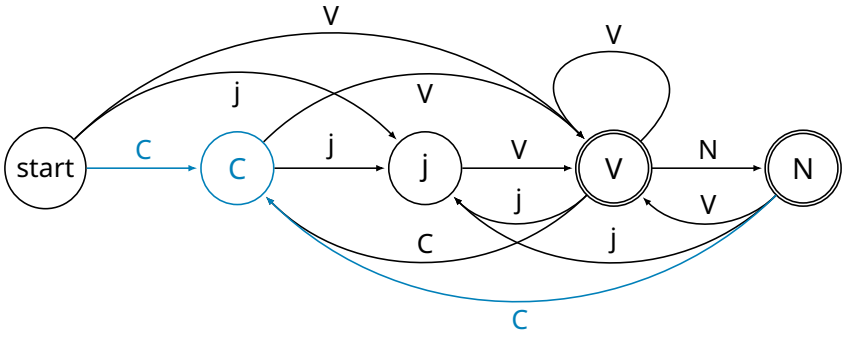
We can visualize an SL grammar using a **finite-state automaton (FSA)**.



SL is a subclass of the languages expressible by FSAs.

Strictly Local Languages (3)

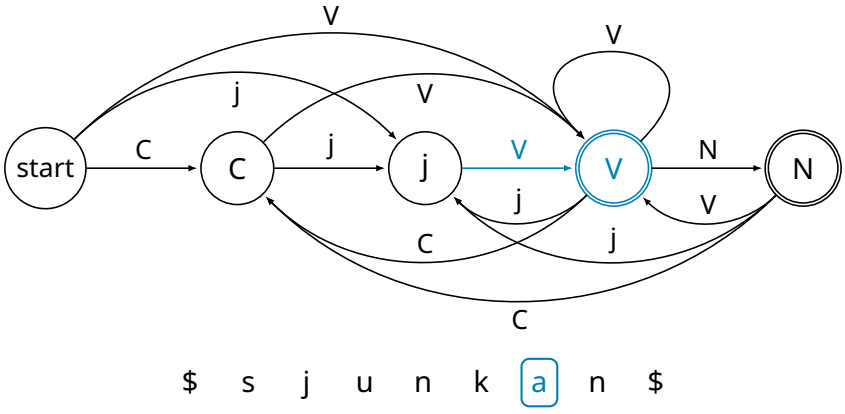
We can visualize an SL grammar using a **finite-state automaton (FSA)**.



SL is a subclass of the languages expressible by FSAs.

Strictly Local Languages (3)

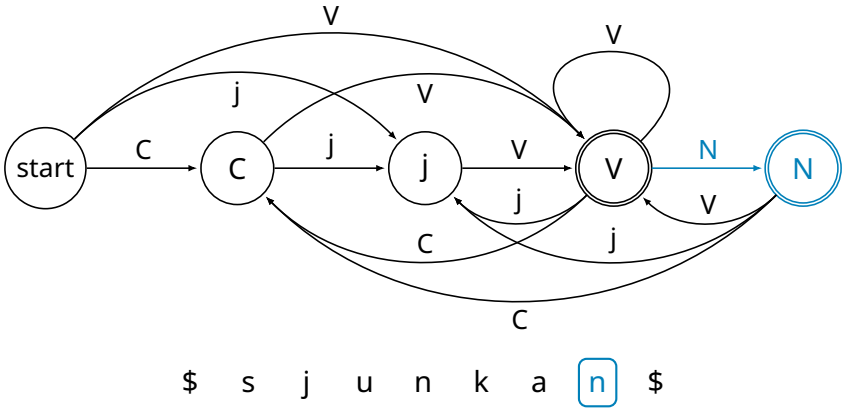
We can visualize an SL grammar using a **finite-state automaton (FSA)**.



SL is a subclass of the languages expressible by FSAs.

Strictly Local Languages (3)

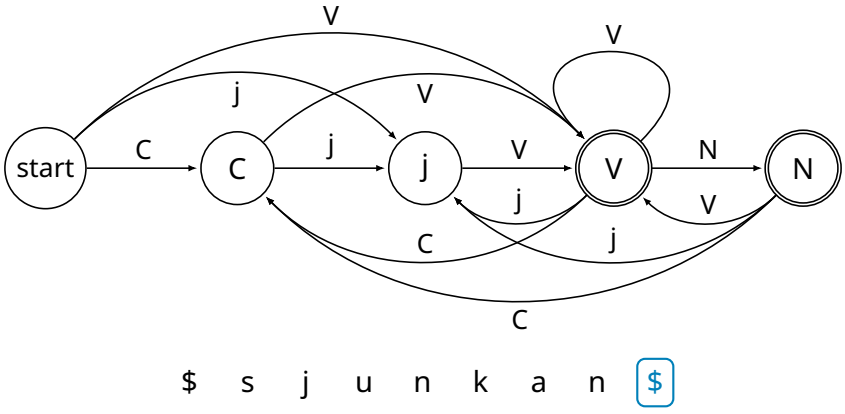
We can visualize an SL grammar using a **finite-state automaton (FSA)**.



SL is a subclass of the languages expressible by FSAs.

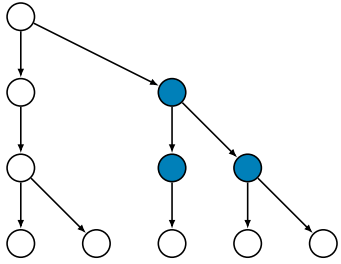
Strictly Local Languages (3)

We can visualize an SL grammar using a **finite-state automaton (FSA)**.



SL is a subclass of the languages expressible by FSAs.

SL in Syntax



Preview

Plan of attack:

1. Functional hierarchies (SL-2)
2. Adjunct ordering (SL-2)
3. Lexical selection (SL-3)

Functional Hierarchies

Example: English clausal hierarchy

T < (Neg) < (Perf) < (Prog) < (Pass) < v

	T	Neg	Perf	Prog	Pass	v+V	
John	will					eat	ice cream.
John	will		has			eaten	ice cream.
John	will			is		eating	ice cream.
John	will		have	been		eating	ice cream.
…							
The ice cream	will	not	have	been	being	eaten.	

Functional Hierarchies (2)

Functional hierarchies are awkward to express using selection.

- T may select Neg/Perf/Prog/Pass/lv
- Neg may select Perf/Prog/Pass/lv
- …

This is unsatisfying for a variety of reasons:

- Each item has multiple lexical entries with no difference in meaning.
- Other than T, each category has only a single member.
- The analysis obscures the real pattern.

Functional Hierarchies (3)

Alternative: stipulate that every projection is always (vacuously) present.

Neither of these (unsatisfying) options is necessary.

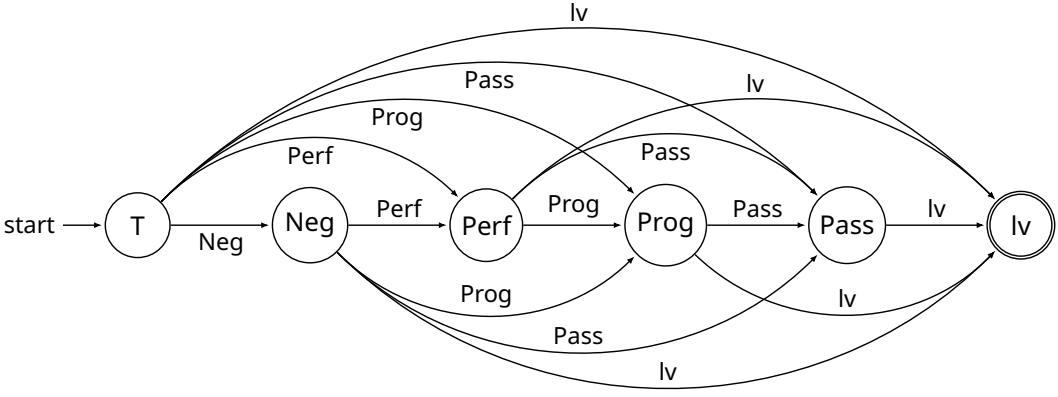
This is simply another SL-2 pattern:

- The identity of an element completely determines what may come next.

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

Functional Hierarchies (4)

FSA for English Clausal Hierarchy



Adjunct Ordering

Adjectives and adverbs often have a preferred order, though it is difficult to say exactly what this order is.

- ✓ *cute little spotted puppy*
- ? *little cute spotted puppy*
- ? *cute spotted little puppy*
- ?? *little spotted cute puppy*

Items in the same group can be iterated.

- ✓ *cute cute cute little spotted puppy*

Adjunct Ordering (2)

Some descriptions of adjective ordering:

- opinion < size < physical quality < shape < age … (Cambridge Dictionary)
- size < length < height < speed < depth < width … (Scott 2002)
- more subjective < less subjective (Scontras et al. 2017)

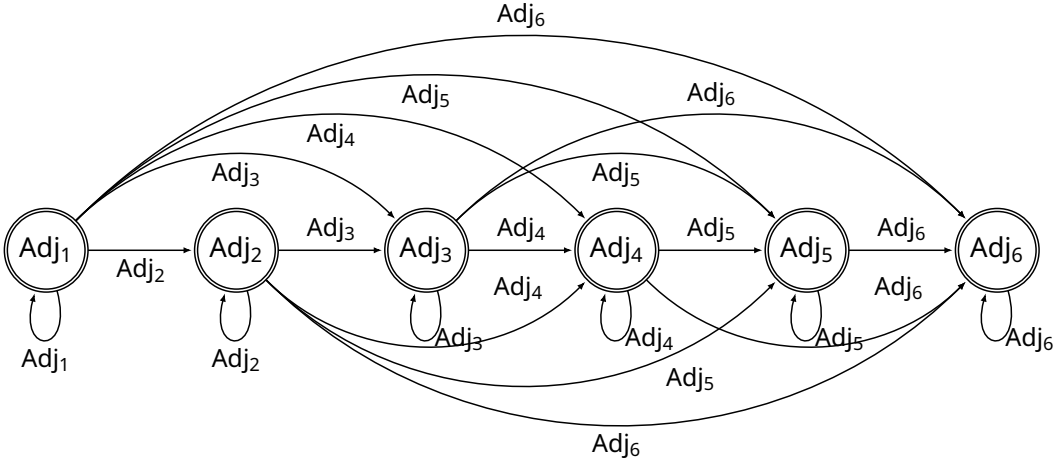
From a mathematical perspective, all of these are **preorders** (Larson 2021).
→ They can be encoded with an SL-2 grammar.

- We could do this with a functional hierarchy (Cinque 1999, et seq.).
- Or we can do it directly.

A preorder is a relation that is reflexive and transitive.

Adjunct Ordering (3)

FSA schematic for adjunct ordering



How does this work in the syntax?

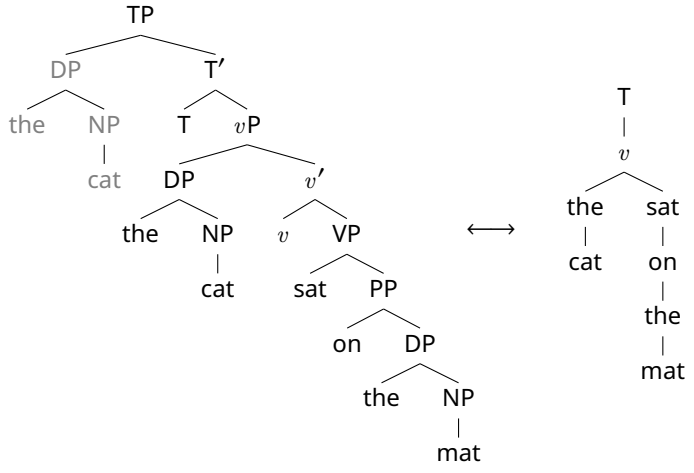
Syntactic representations are trees, not strings.

Two possible approaches:

- Define a set of treelets that combine for form larger trees (Rogers 1997)
- Extract string paths from the tree, and enforce an SL grammar on those strings (Graf and Shafiei 2019).

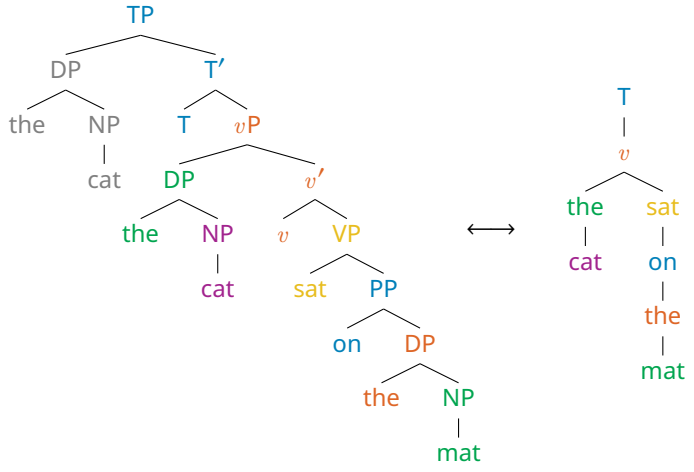
I will take the latter approach.

Dependency Trees



Every element of the tree is a lexical item. The rightmost daughter of a node is its complement; other daughters are specifiers. See Graf and Kostyszyn (2021) and references therein.

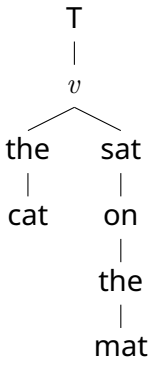
Dependency Trees



Every element of the tree is a lexical item. The rightmost daughter of a node is its complement; other daughters are specifiers. See Graf and Kostyszyn (2021) and references therein.

Category and Selector Features

F^- = category feature F^+ = selector feature

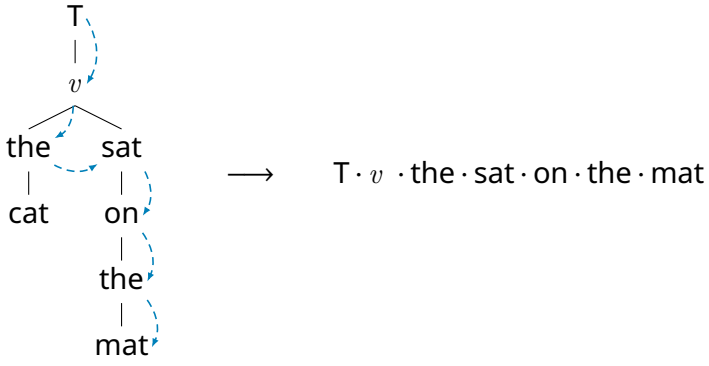


T	$\langle T^- \; Iv^+ \rangle$
v	$\langle Iv^- \; D^+ \; V^+ \rangle$
sat	$\langle V^- \; P^+ \rangle$
on	$\langle P^- \; D^+ \rangle$
the	$\langle D^- \; N^+ \rangle$
cat	$\langle N^- \rangle$
mat	$\langle N^- \rangle$

Note: the feature system is based on Minimalist Grammars (Stabler 1997).

Lexical Selection

We will extract a string path from the tree in which selectors and selectees appear close to each other.



This string is called a **command string (c-string)**. Details to come shortly.

Lexical Selection (2)

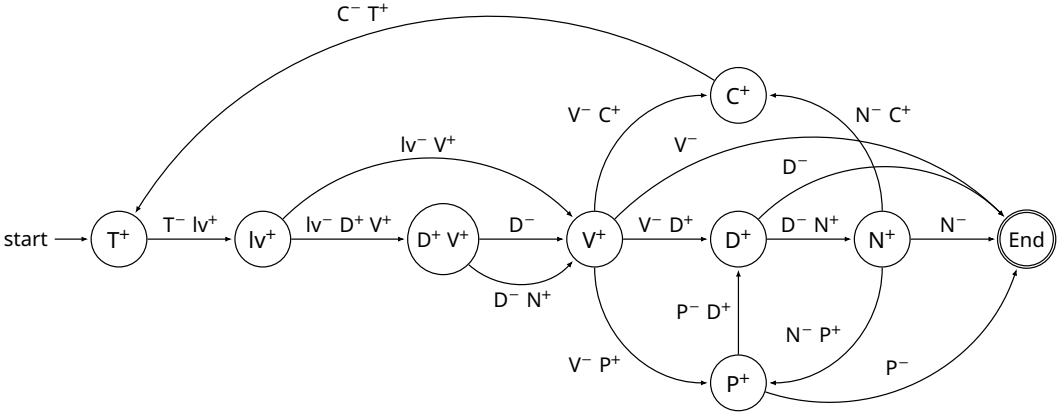
What the SL grammar looks like

k = 3

$$G = \left\{ \begin{array}{ccc} \langle V^- D^+ \rangle & \langle D^- N^+ \rangle \cdots ? & \\ \cdots & \langle V^- D^+ \rangle & \langle D^- N^+ \rangle \\ \langle D^- N^+ \rangle & \langle N^- P^+ \rangle & \cdots \\ \cdots & \langle D^- N^+ \rangle & \langle N^- P^+ \rangle \\ \langle lv^- D^+ V^+ \rangle & \langle D^- N^+ \rangle & \langle V^- D^+ \rangle \\ \cdots & \langle lv^- D^+ V^+ \rangle & \langle D^- N^+ \rangle \\ \vdots & \vdots & \vdots \end{array} \right\}$$

Lexical Selection (3)

FSA for Lexical Selection



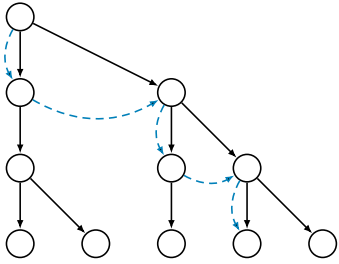
Interim Summary

Lexical selection, functional hierarchies, and adjunct ordering are all SL.

It is not necessary to literally treat a functional hierarchy as selection, or adjunct ordering as a functional hierarchy.

At the same time, all three are instances of the same kind of abstract pattern.

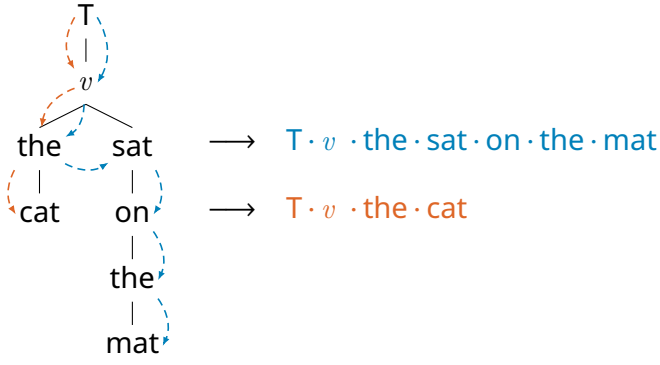
The Technical Details



C-Strings

The **command string (c-string)** of a node is a path from the root which visits the node's c-commanders along the way on the way.

Example: c-strings for *cat* and *mat*



C-Strings (2)

In the dependency tree, the c-string of a node includes

- all of its ancestors
- and the left siblings of its ancestors

…ordered so that precedence and dominance are preserved.

The order is a blend of c-command and m-command.

- Heads precede specifiers (m-command)
- Specifiers precede complements (c-command)

This is a natural **derivational order** w.r.t. feature checking.

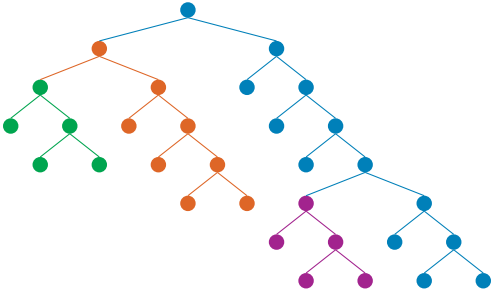
- The complement is the first to have all of its features checked, followed by the specifier, and then the head.

See Graf and Shafiei (2019) for details.

Spines

Which c-strings do we use?

Those that trace the complement spine of the tree, or of a subtree.



A computational device called a **sensing tree automaton** can enforce the c-string grammars for each spine in the tree (Graf and De Santo 2019).

SL as Structure Building Computation

Structure Building with SL

The same grammar that can recognize whether a structure is well-formed can also generate structures that conform to the grammar.

→ SL can be thought of as the basis of linguistic structure building.

Structure Building with SL

The same grammar that can recognize whether a structure is well-formed can also generate structures that conform to the grammar.

→ SL can be thought of as the basis of linguistic structure building.

Structure Building with SL

The same grammar that can recognize whether a structure is well-formed can also generate structures that conform to the grammar.

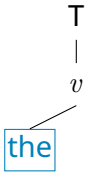
→ SL can be thought of as the basis of linguistic structure building.

Structure Building with SL

The same grammar that can recognize whether a structure is well-formed can also generate structures that conform to the grammar.

→ SL can be thought of as the basis of linguistic structure building.

$ s j

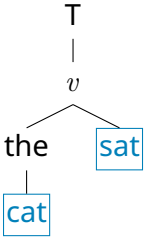


Structure Building with SL

The same grammar that can recognize whether a structure is well-formed can also generate structures that conform to the grammar.

→ SL can be thought of as the basis of linguistic structure building.

$ s j u

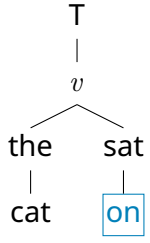


Structure Building with SL

The same grammar that can recognize whether a structure is well-formed can also generate structures that conform to the grammar.

→ SL can be thought of as the basis of linguistic structure building.

$ s j u n

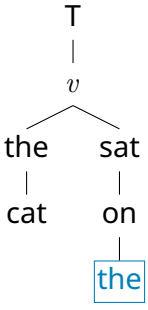


Structure Building with SL

The same grammar that can recognize whether a structure is well-formed can also generate structures that conform to the grammar.

→ SL can be thought of as the basis of linguistic structure building.

$ s j u n k

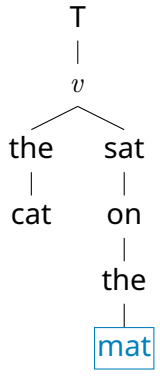


Structure Building with SL

The same grammar that can recognize whether a structure is well-formed can also generate structures that conform to the grammar.

→ SL can be thought of as the basis of linguistic structure building.

$ s j u n k a

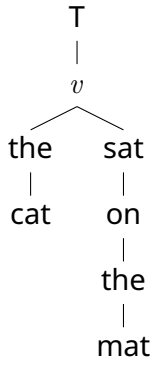


Structure Building with SL

The same grammar that can recognize whether a structure is well-formed can also generate structures that conform to the grammar.

→ SL can be thought of as the basis of linguistic structure building.

$ s j u n k a n

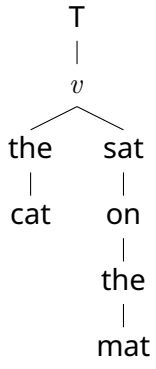


Structure Building with SL

The same grammar that can recognize whether a structure is well-formed can also generate structures that conform to the grammar.

→ SL can be thought of as the basis of linguistic structure building.

$ s j u n k a n $



Structure Building with SL (2)

Some characteristics of the model:

- It is a graph-theoretic model
- It has a derivational interpretation (the moving window)
- Unlike set-theoretic Merge (Chomsky 1995) but reminiscent of Chomsky's earlier work (Chomsky 1956; Chomsky 1959)

Conclusion

Local dependencies in syntax are SL.

- Functional hierarchies and adjunct hierarchies are unsurprising from a computational perspective — they are just further examples of SL patterns.

Syntax and phonology are very similar in computational terms.

- This similarity is especially clear using the c-string perspective.

SL computations are a good candidate for the basis of linguistic structure building.

- The same grammar that recognizes a string or tree also contains the information needed to build the structure.

Thank you!

Acknowledgments

- This work was supported by NSF Grant BCS-1845344.
- The idea of treating syntax as an FSA was partly inspired by a draft book chapter by Kobele (2021).
- Thanks to the Stony Brook Mathematical Linguistics Reading Group and the audience at SYNC 2023 for comments and discussion.

References

Chomsky, Noam (1956). “Three models for the description of language”. In: *IRE Transactions on information theory* 2.3.

— (1959). “On certain formal properties of grammars”. In: *Information and control* 2.2.

— (1995). *The Minimalist Program*. MIT Press.

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

Garcia, Pedro et al. (1990). “Learning Locally Testable Languages in the Strict Sense”. In: *Proceedings of the Workshop on Algorithmic Learning Theory*.

Graf, Thomas (2018). “Why movement comes for free once you have adjunction”. In: *Proceedings of CLS* 53.

— (2022a). “Subregular linguistics: bridging theoretical linguistics and formal grammar”. In: *Theoretical Linguistics* 48.3-4. DOI: doi:10.1515/tl-2022-2037.

— (2022b). “Typological implications of tier-based strictly local movement”. In: *Proceedings of the Society for Computation in Linguistics 2022*.

Graf, Thomas and Aniello De Santo (2019). “Sensing Tree Automata as a Model of Syntactic Dependencies”. In: *Proceedings of the 16th Meeting on the Mathematics of Language*. Toronto, Canada: Association for Computational Linguistics.

References (2)

Graf, Thomas and Kalina Kostyszyn (Feb. 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*. Online: Association for Computational Linguistics.

Graf, Thomas and Nazila Shafiei (2019). "C-command dependencies as TSL string constraints". In: *Proceedings of the Society for Computation in Linguistics (SCiL) 2019*.

Hanson, Kenneth (2023). "A TSL Analysis of Japanese Case". In: *Proceeding of SCiL 2023*. Forthcoming.

Heinz, Jeffrey (July 2010). "String Extension Learning". In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Uppsala, Sweden: Association for Computational Linguistics.

— (2018). "The computational nature of phonological generalizations". In: *Phonological Typology, Phonetics and Phonology*.

Kobele, Greg (2021). *Minimalist Grammars and Decomposition*. Submitted.

Lambert, Dakotah et al. (2021). "Typology emerges from simplicity in representations and learning". In: *Journal of Language Modelling* 9.1.

Larson, Richard K (2021). "Rethinking cartography". In: *Language* 97.2.

McNaughton, Robert and Seymour A. Papert (1971). *Counter-Free Automata*. The MIT Press.

Rogers, James (1997). "Strict LT_2 : Regular :: Local : Recognizable". In: *Logical Aspects of Computational Linguistics: First International Conference, LACL '96 (Selected Papers)*. Vol. 1328. Lectures Notes in Computer Science/Lectures Notes in Artificial Intelligence. Springer.

References (3)

Scontras, Gregory et al. (2017). “Subjectivity predicts adjective ordering preferences”. In: *Open Mind* 1.1.

Scott, Gary-John (2002). “Stacked adjectival modification and the structure of nominal phrases”. In: *G. Cinque (ed.) Functional Structure in DP and IP. New York: Oxford University Press, 2002.*

Stabler, Edward P. (1997). “Derivational minimalism”. In: *Logical Aspects of Computational Linguistics*. Springer.

Vu, Mai Ha et al. (2019). “Case assignment in TSL syntax: A case study”. In: *Proceedings of the Society for Computation in Linguistics (SCiL) 2019*.

Extras

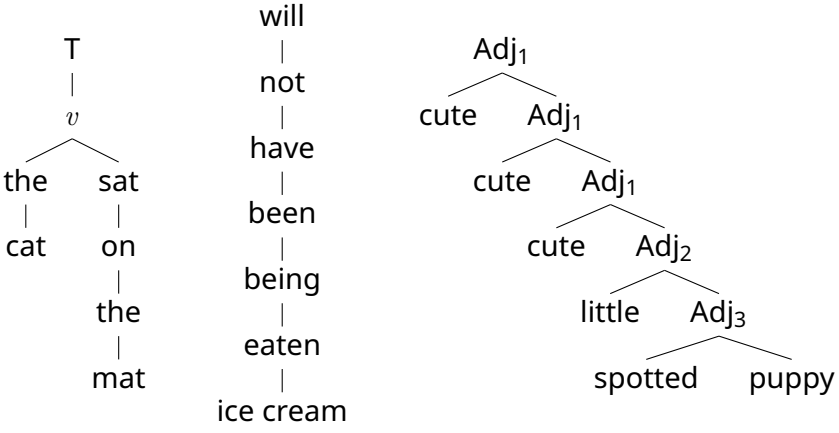
Long-Distance Dependencies

What about long-distance dependencies?

Most long-distance dependencies are tier-based strictly local (TSL), a generalization of SL in which non-salient items are ignored (Heinz 2018; Graf 2022a).

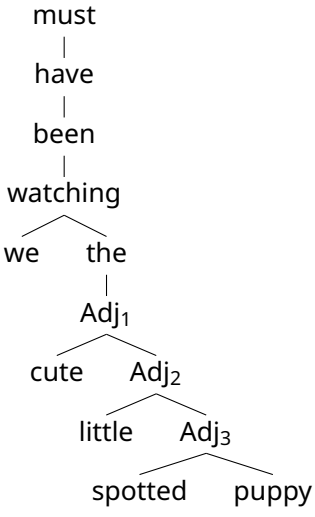
	Local (SL)	Long-Distance (TSL)
Phonology	Local Phonotactics (Heinz 2018)	Harmony/Disharmony (Heinz 2018)
Syntax	Category Selection (Graf 2018) Functional Hierarchies Adjunct Ordering	Movement (Graf 2022b) Case (Vu et al. 2019; Hanson 2023) NPI Licensing (Graf and Shafiei 2019) Binding (Graf and Shafiei 2019) Phi-Agreement (Hanson, in prep.)

Dependency Tree Comparison



Putting Everything Together

We must have been watching the cute little spotted puppy.



Computational Complexity

Strictly local languages are extremely restricted in expressive power.
They lie at the very bottom of the hierarchy of formal languages (McNaughton and Papert 1971).

1. The patterns they can encode are extremely restricted.
2. They are efficient to process.
3. They are easy to learn, though you need to guess the window size k.

Computational Complexity (2)

1. The patterns that SL languages encode are extremely restricted.
   - No relationship between symbols at arbitrary distance
   - No boolean conditions: e.g. you can have ABC and XYZ, but not both
   - No counting of substrings: e.g. you can have ABC only up to three times

Computational Complexity (3)

2. SL languages are efficient to process.

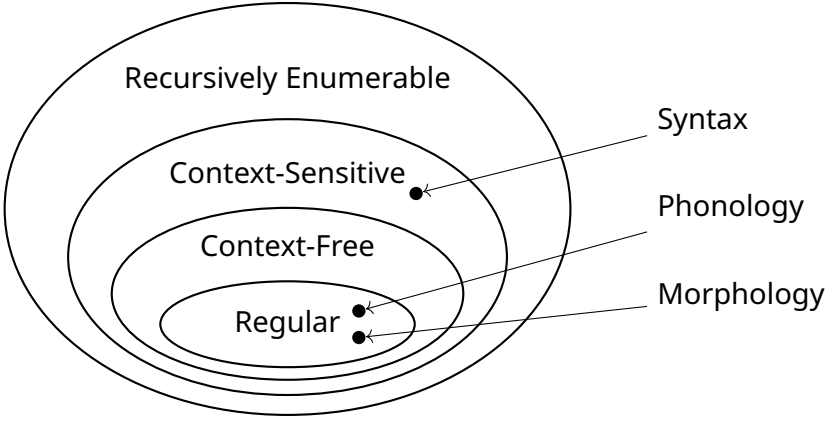
   - The size of the grammar is at most $|\Sigma|^k$, where Σ is the set of symbols.
   - Testing or generating a string takes linear time, e.g. when implemented as a finite state machine.

Computational Complexity (4)

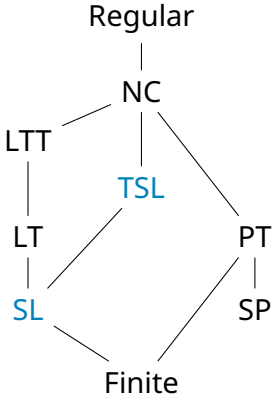
3. SL languages are easy to learn.

- Just keep track of all attested substrings of size k.
  $\rightarrow$ *string extension learning* (Garcia et al. 1990; Heinz 2010)
- The required computations are cognitively plausible (Lambert et al. 2021).
- The time to process the input data is linear.
- Very little data is needed (compared to more expressive classes).

The Chomsky Hierarchy



The Subregular Hierarchy



Subregular String-to-String Functions

TODO