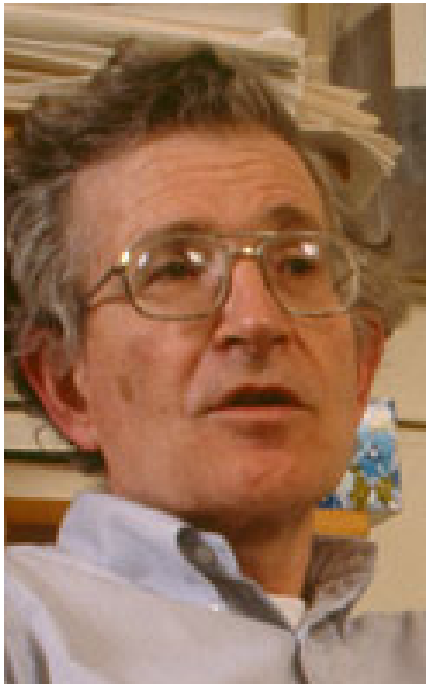
A Computational Common Ground for Syntax and Phonology

Thomas Graf
mail@thomasgraf.net
http://thomasgraf.net

Department of Linguistics

March 11 2015



It is important to learn to be surprised by simple things [. . .] The beginning of science is the recognition that the simplest phenomena of life raise quite serious problems: Why are they as they are, instead of some different way?

Chomsky 1988:43

The Syntax-Phonology Puzzle

- The grammar of sentences (syntax) and the grammar of sounds (phonology) are different lines of linguistic inquiry.
- Empirical evidence and mathematical theorems both show that syntax is much more complex than phonology.
- **Question:** Why should that be the case?

Answer: A Surprising Common Ground

- Phonology and syntax are not that different after all.
- They involve computations of comparable complexity.
- The main difference lies in their data structures.

Phonology: strings
Syntax: trees

Outline

Phonological Patterns

- Only certain sound sequences are licit.
- Vowel systems show regularities.
  a-i-u, a-e-i-o-u, *e-o-i
- Sounds can be affected by their contexts,
  but only in specific ways.

intervocalic voicing	*nef+iðr* → *neviðr*	Icelandic
word-final devoicing	*rad* → *rat*	German
*intervocalic devoicing	*aba* → *apa*	unattested
dissimilation	*lun+alis* → *lunaris*	Latin
umlaut	*mamm+u* → *mömmu*	Icelandic
*anti-umlaut	*mömm+u* → *mammu*	unattested

Syntactic Patterns

- Island effects

  (1) a. Which man did John say that Mary kissed?

  b. * Which man did John cry because Mary kissed?

- Center-embedding/Nested dependencies

  (2) a. The mouse that the cat that the dog chased ate is dead.

  b. * The mouse that the cat that the dog chased ate is dead.

- Crossing dependencies

  (3) a. The mouse, the cat, and the dog survived, slept, and chewed on a toy, respectively.

  b. * The mouse, the cat, and the dog survived, slept, and chewed on a toy, respectively.

Empirical Lay of the Land

- There are **no nested/crossing dependencies in phonology**. They are a peculiarity of syntax.
- **No syntactic analogues for phonological processes** like devoicing or umlaut have been clearly identified.
- This is just the tip of the iceberg:

$$\text{island effects} \stackrel{?}{\equiv} \text{blocking}$$

$$\text{negative concord} \stackrel{?}{\equiv} \text{vowel harmony}$$

$$\text{gapping:ellipsis} \equiv \text{deletion:?}$$

$$\text{Principle C} \equiv ?$$

$$? \equiv \text{hiatus}$$

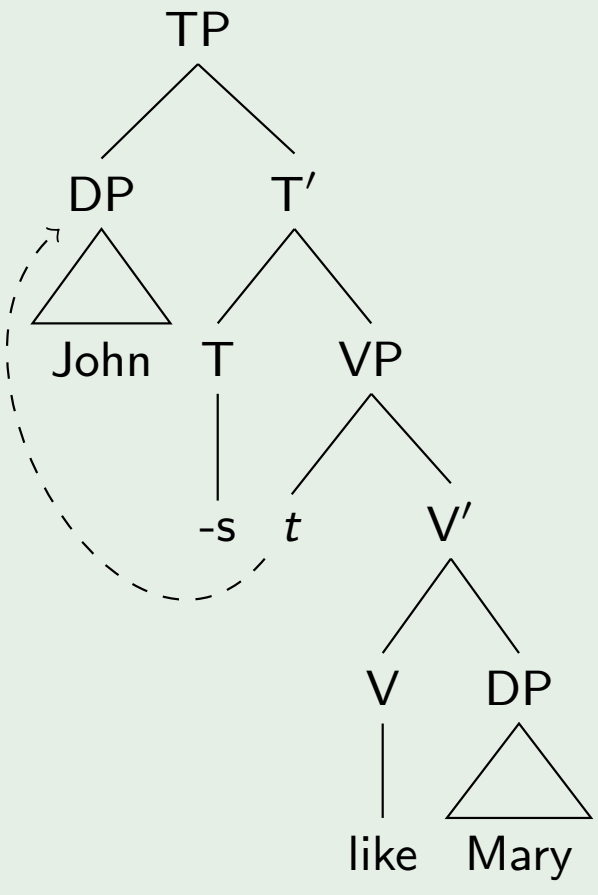
$$\text{PCC} \equiv ?$$

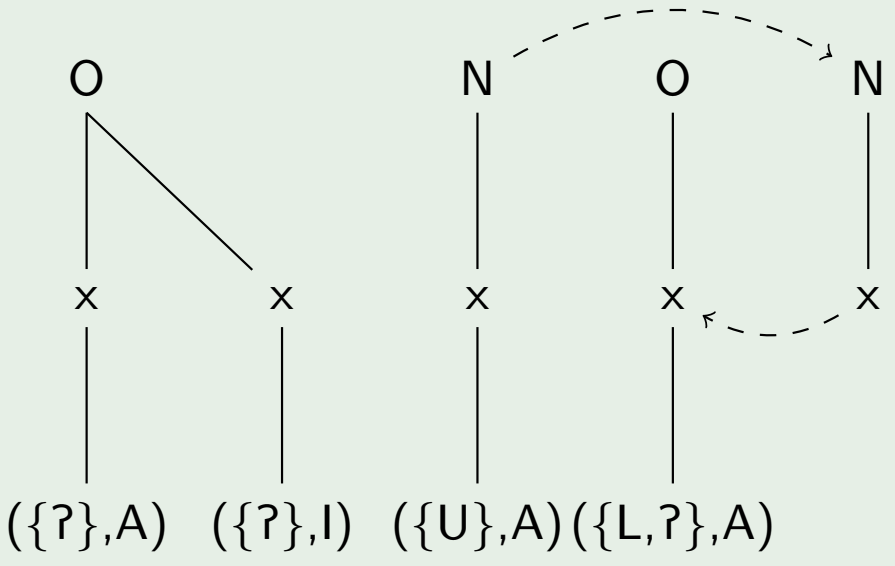
$$\vdots$$

The Theoretical Divide

Even when syntactic and phonological theories are meant to mirror each other closely, they end up looking very different.

Example: Government and Binding VS Government Phonology

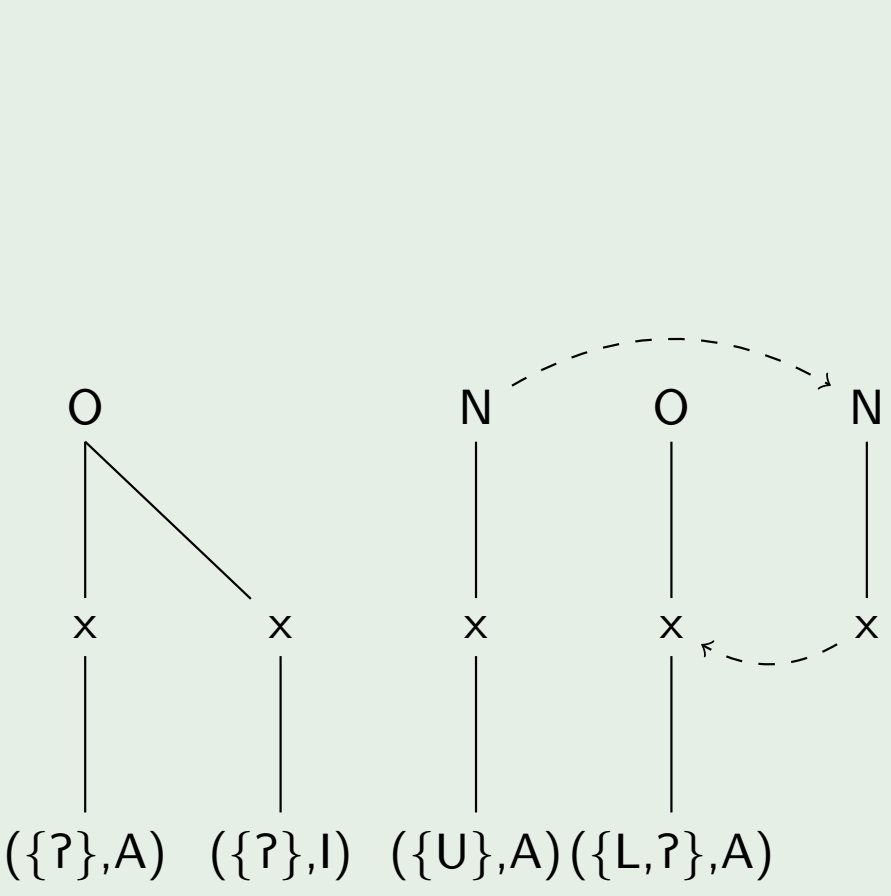


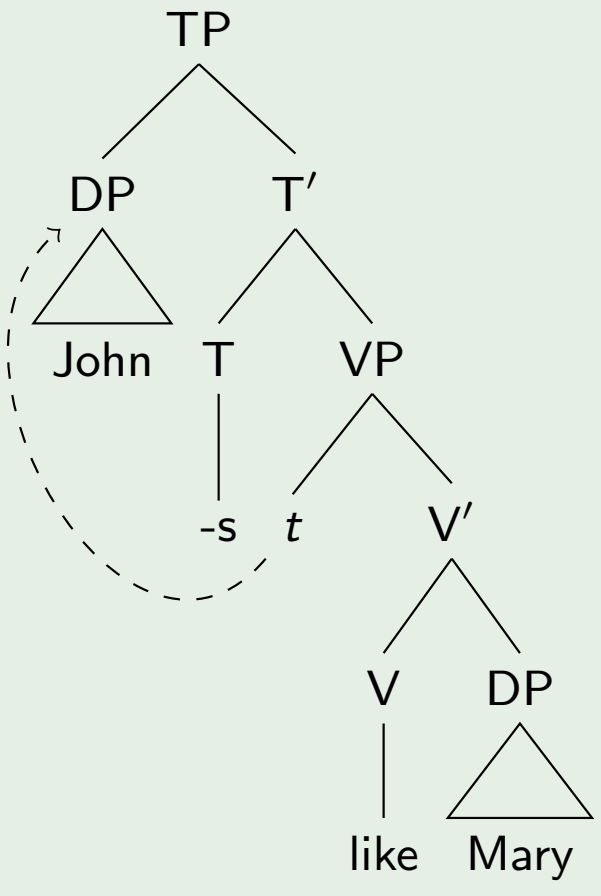


The Theoretical Divide

Even when syntactic and phonological theories are meant to mirror each other closely, they end up looking very different.

Example: Government and Binding VS Government Phonology





The General Upshot

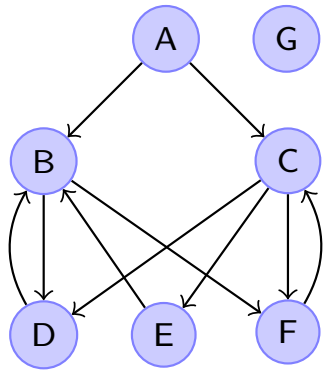
- Languages impose rules on sounds and sentences.
- But the rules/patterns are not the same across these two domains.
- This is reflected by linguistic theories, but also by the sociology of the field ("are you S-side or P-side?").
- **Next:** There are even **mathematical proofs** that separate syntax and phonology.

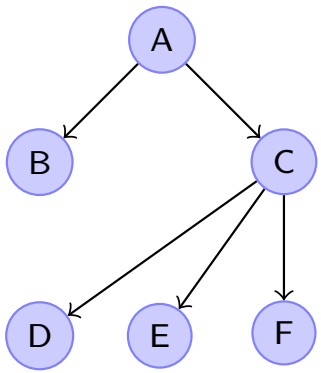
Outline

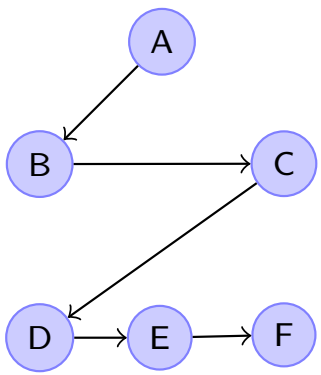
Language as Sets

Computationally, a language is simply
a set of objects of a specific type:

- **graph**: structure of connected nodes
  flow chart, *street network*, *Wikipedia*, *internet*, *video game AI*
- **tree**: connected graph where every node is reachable from at most one node
  family tree, *hard drive layout*, *XML file*
- **string**: sequence of nodes
  telephone number, *Python source code*, *Shakespeare's oeuvre*

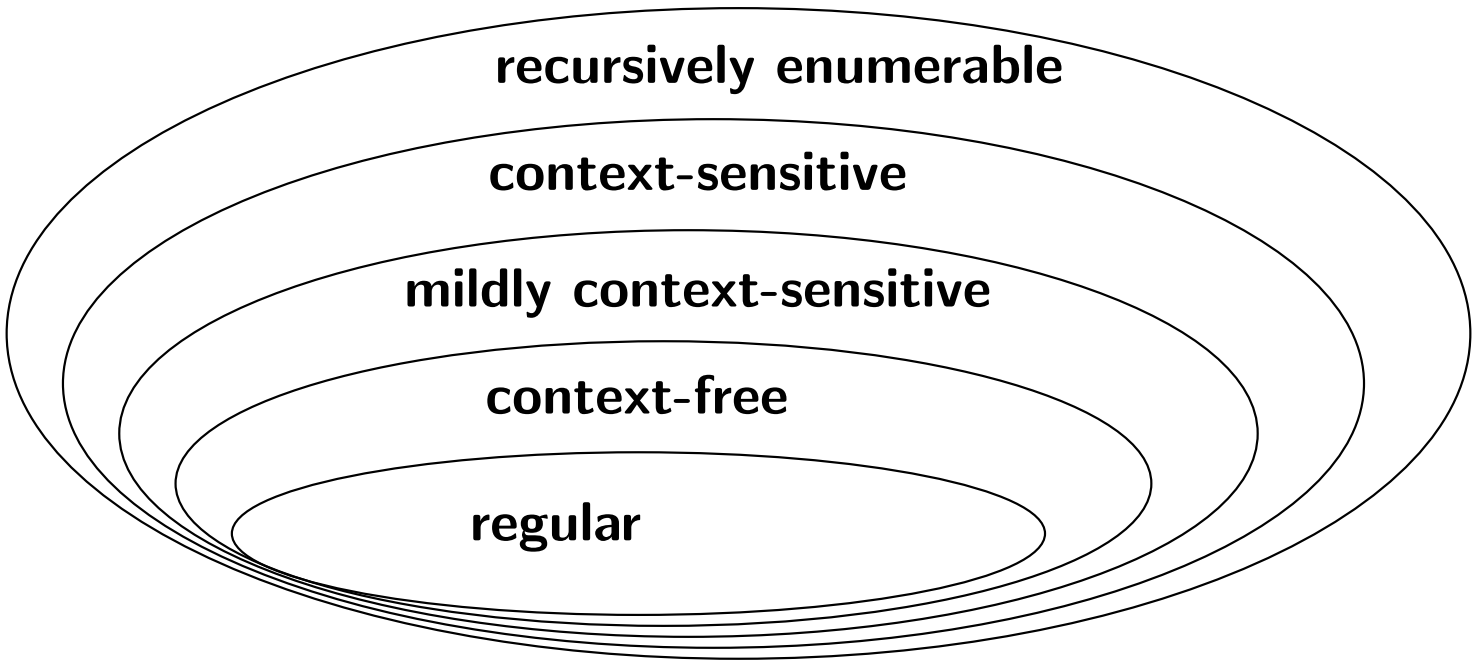






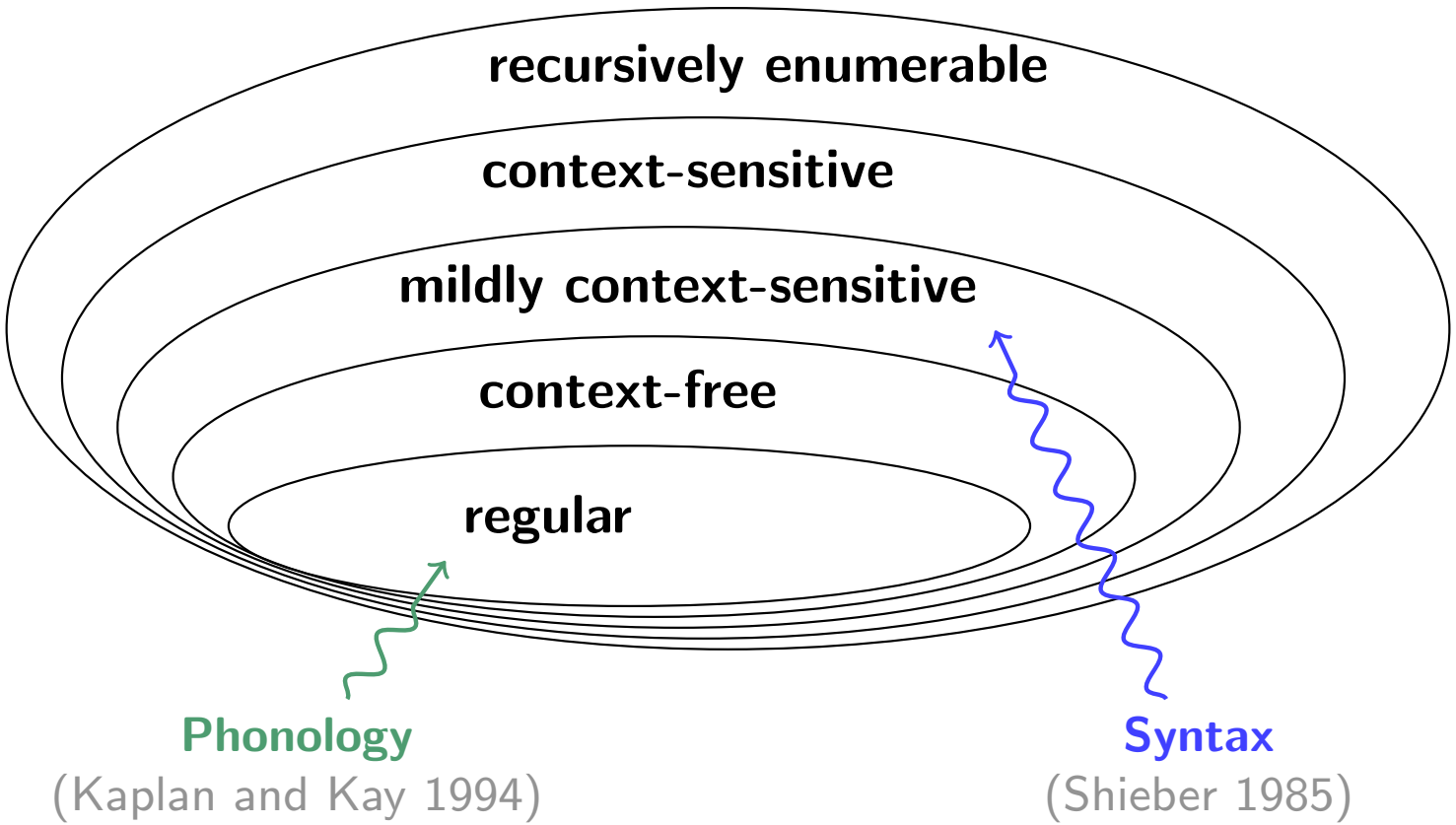
The Chomsky Hierarchy of String Languages

- The perceivable output of language is strings (sequences of sound waves, words, sentences).
- The complexity of string languages is measured by the (extended) **Chomsky hierarchy**. (Chomsky 1956, 1959)



The Chomsky Hierarchy of String Languages

- The perceivable output of language is strings (sequences of sound waves, words, sentences).
- The complexity of string languages is measured by the (extended) **Chomsky hierarchy**. (Chomsky 1956, 1959)



Languages and Automata

- For every language class there is a computational model that can generate all languages in the class, and only those.
- Such a model is called an **automaton**.
- Automata models tell us what kind of **memory structures** are needed in order to compute specific patterns.

Finite-State Automata

A **finite-state automaton** (FSA) assigns every node in a string one of finitely many *states*, depending on

- the label of the node, and
- the state of the preceding node (if it exists).

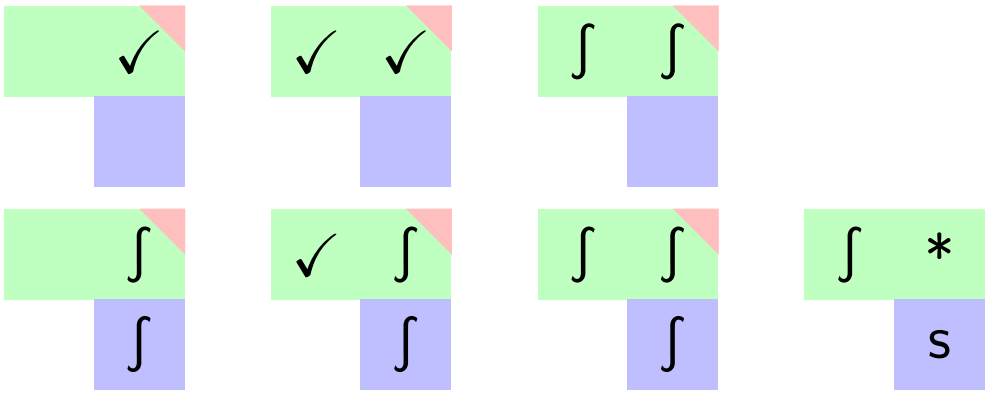
The FSA accepts the string if the last state is a *final state*.

Cognitive Intuition

- States are a metaphor for memory configurations.
- Every symbol in the input induces a change from one memory configuration into another.
- Only finitely many memory configurations are needed.
  Thus the amount of working memory used by the automaton is finitely bounded.

Example 1: Sibilant Harmony

Condition: ʃ cannot be followed by s
Memory: 3 distinct states ✓, ʃ, and *

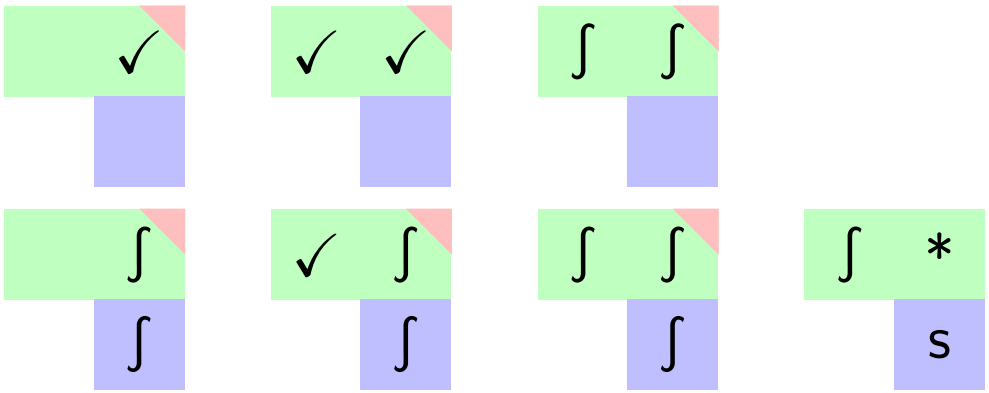


ʃ e g o ʃ a

g e ʃ o s a

Example 1: Sibilant Harmony

Condition: ʃ cannot be followed by s
Memory: 3 distinct states ✓, ʃ, and *

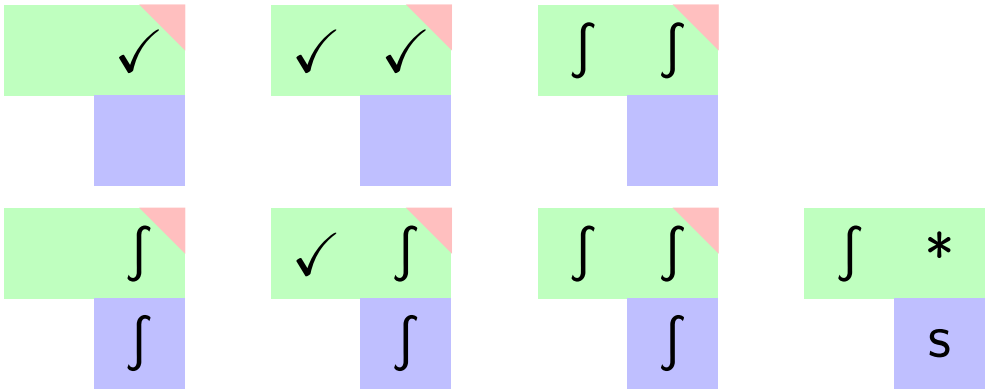


ʃ e g o ʃ a

g e ʃ o s a

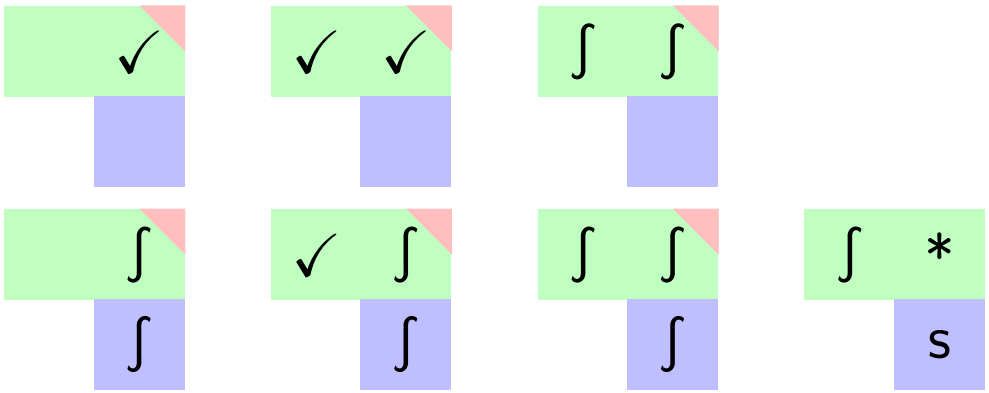
Example 1: Sibilant Harmony

Condition: ∫ cannot be followed by s
Memory: 3 distinct states ✓, ∫, and *



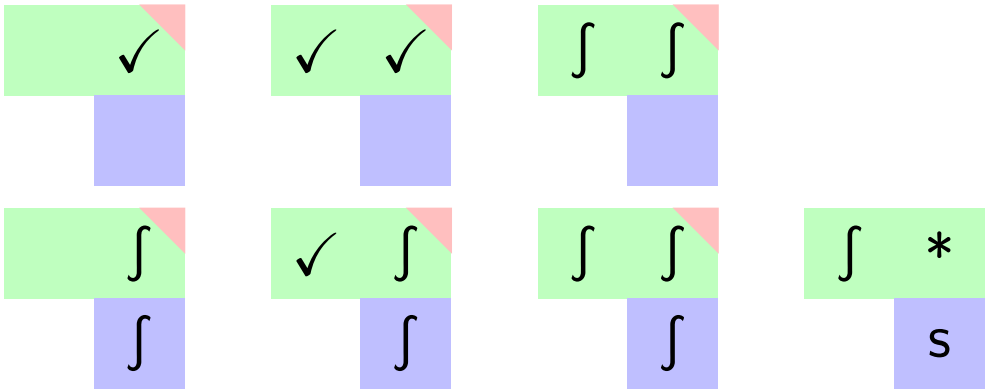
Example 1: Sibilant Harmony

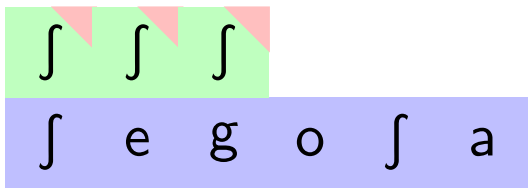
Condition: ʃ cannot be followed by s
Memory: 3 distinct states ✓, ʃ, and *



Example 1: Sibilant Harmony

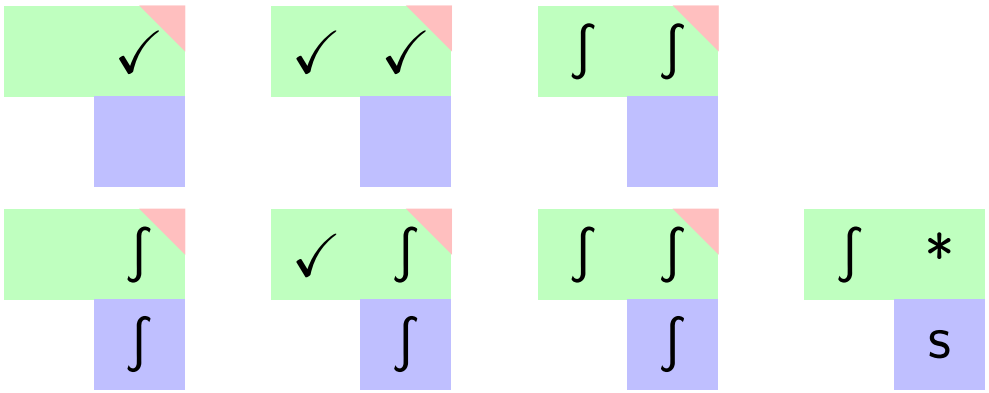
Condition: ∫ cannot be followed by s
Memory: 3 distinct states ✓, ∫, and *

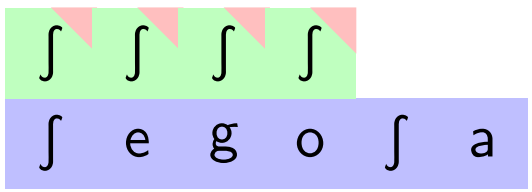




Example 1: Sibilant Harmony

Condition: ∫ cannot be followed by s
Memory: 3 distinct states ✓, ∫, and *

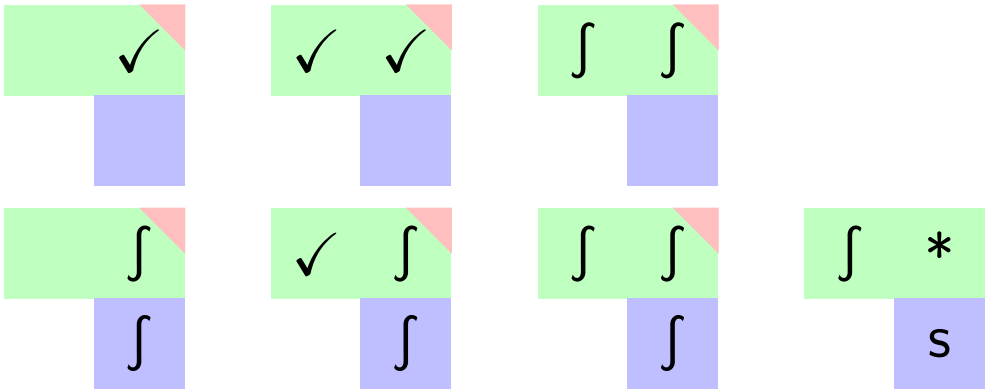


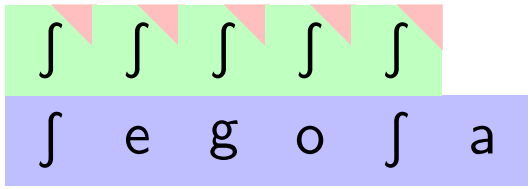


g e ∫ o s a

Example 1: Sibilant Harmony

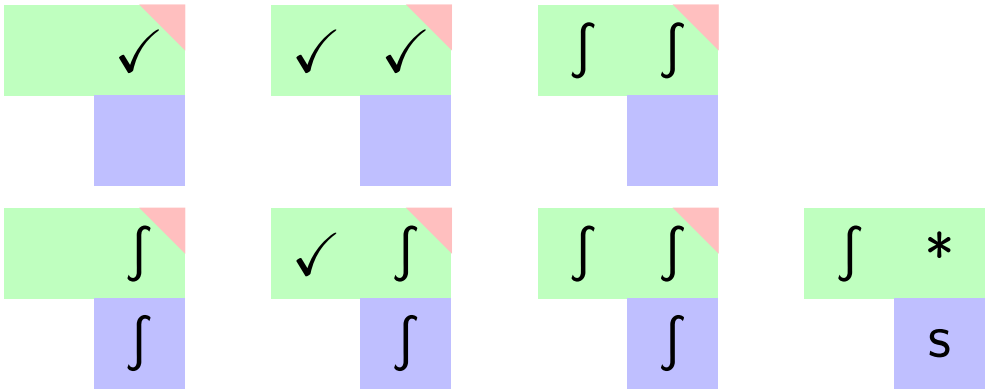
Condition: ∫ cannot be followed by s
Memory: 3 distinct states ✓, ∫, and *

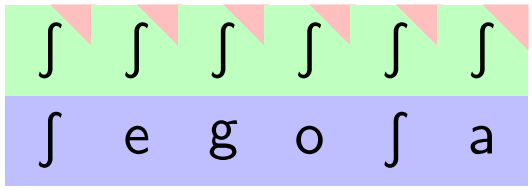




Example 1: Sibilant Harmony

Condition: ∫ cannot be followed by s
Memory: 3 distinct states ✓, ∫, and *

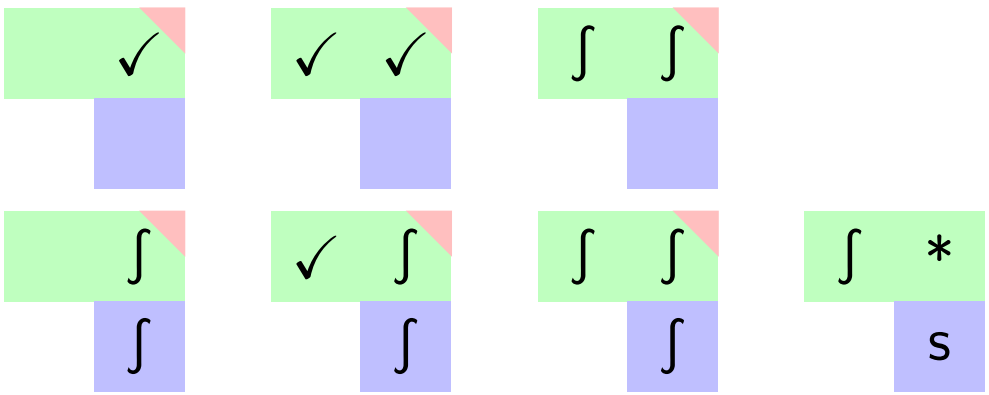




g e ∫ o s a

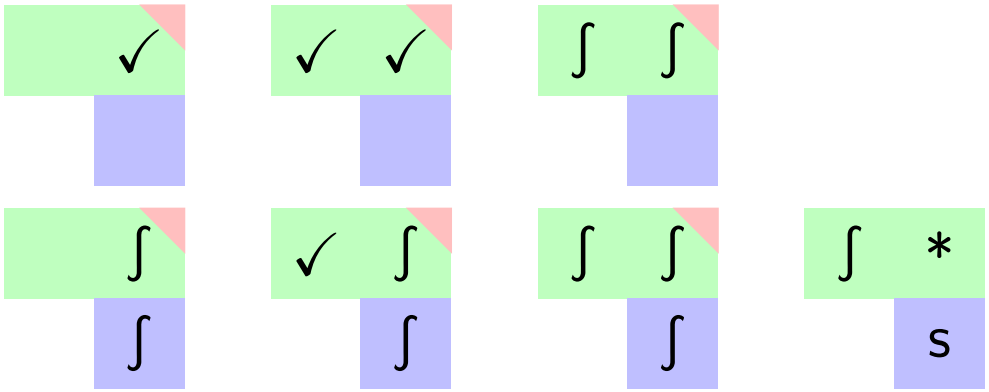
Example 1: Sibilant Harmony

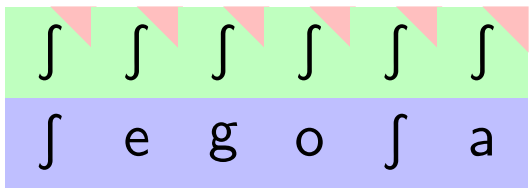
Condition: ∫ cannot be followed by s
Memory: 3 distinct states ✓, ∫, and *

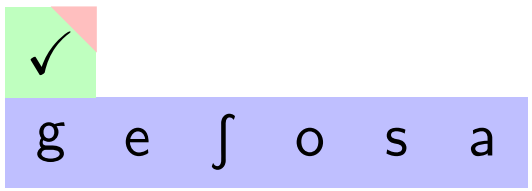


Example 1: Sibilant Harmony

Condition: ∫ cannot be followed by s
Memory: 3 distinct states ✓, ∫, and *

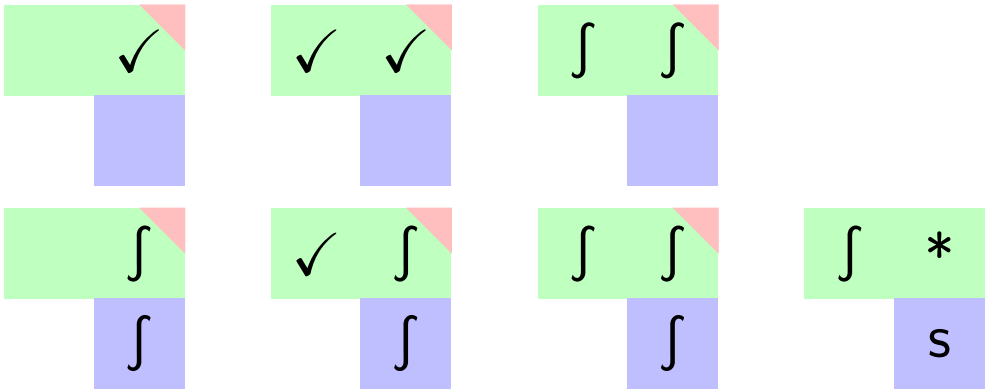


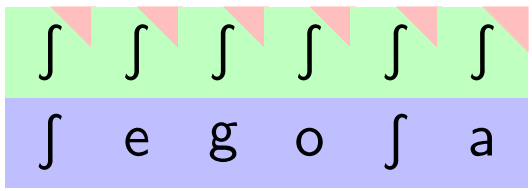


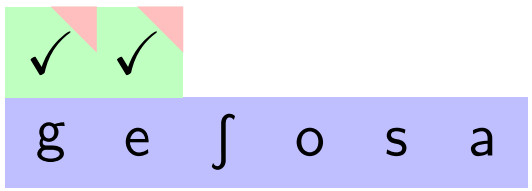


Example 1: Sibilant Harmony

Condition: ∫ cannot be followed by s
Memory: 3 distinct states ✓, ∫, and *

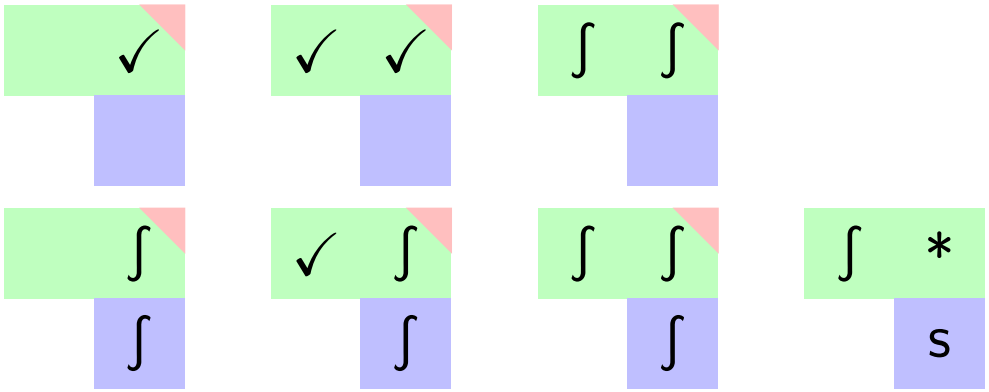


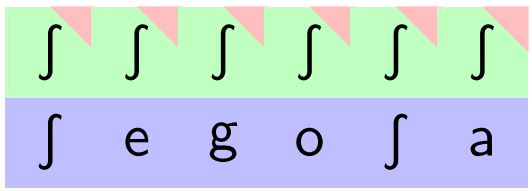


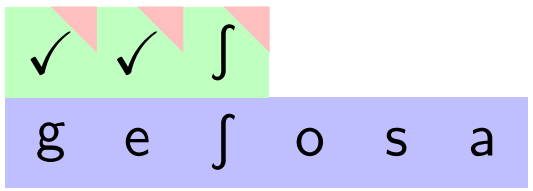


Example 1: Sibilant Harmony

Condition: ∫ cannot be followed by s
Memory: 3 distinct states ✓, ∫, and *

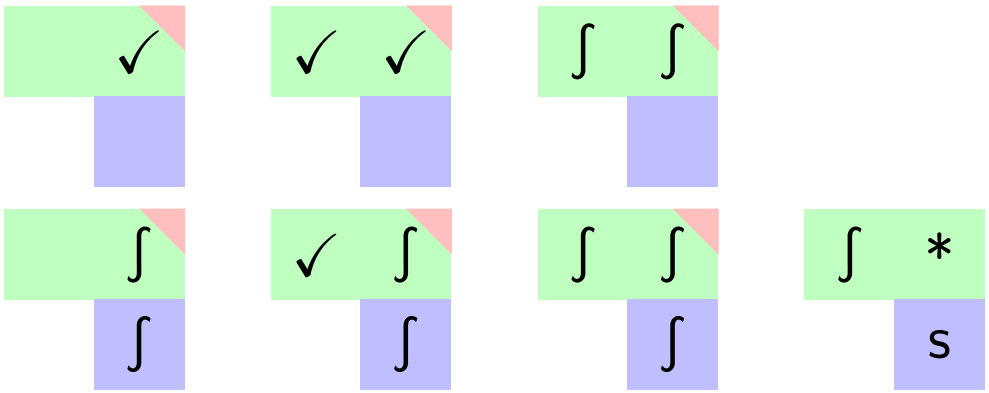


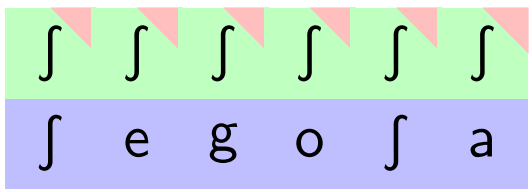


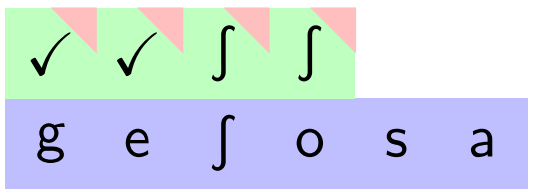


Example 1: Sibilant Harmony

Condition: ∫ cannot be followed by s
Memory: 3 distinct states ✓, ∫, and *

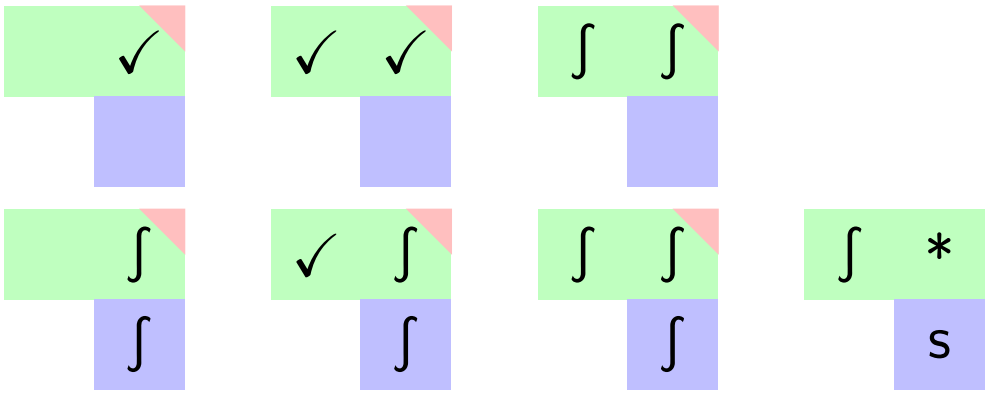


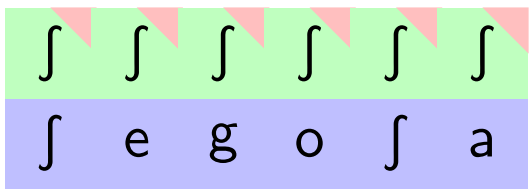


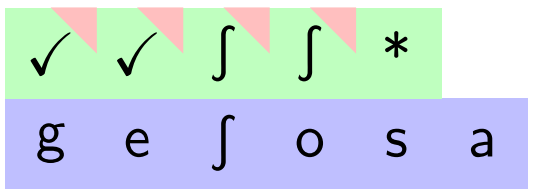


Example 1: Sibilant Harmony

Condition: ∫ cannot be followed by s
Memory: 3 distinct states ✓, ∫, and *

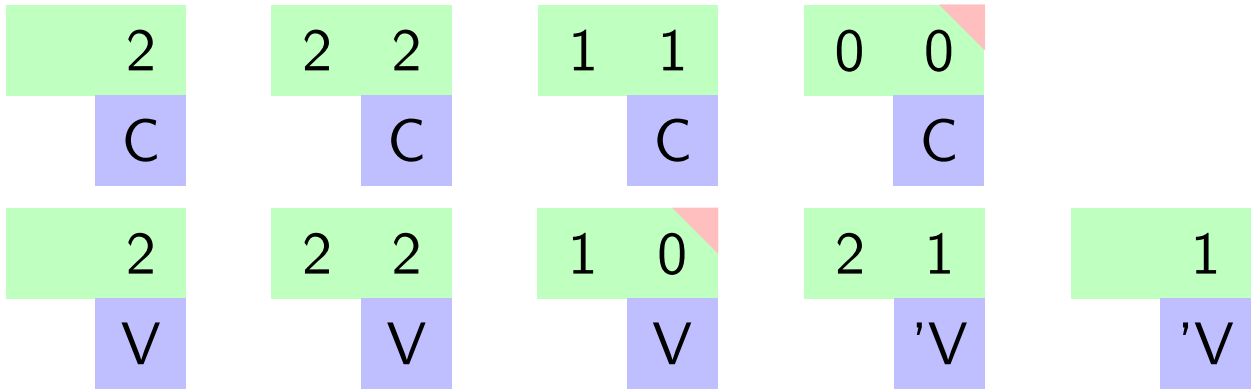






Example 2: Penultimate Stress

Condition: Put stress on the penultimate (= last but one) vowel
Memory: 2 distinct states 2 and 1

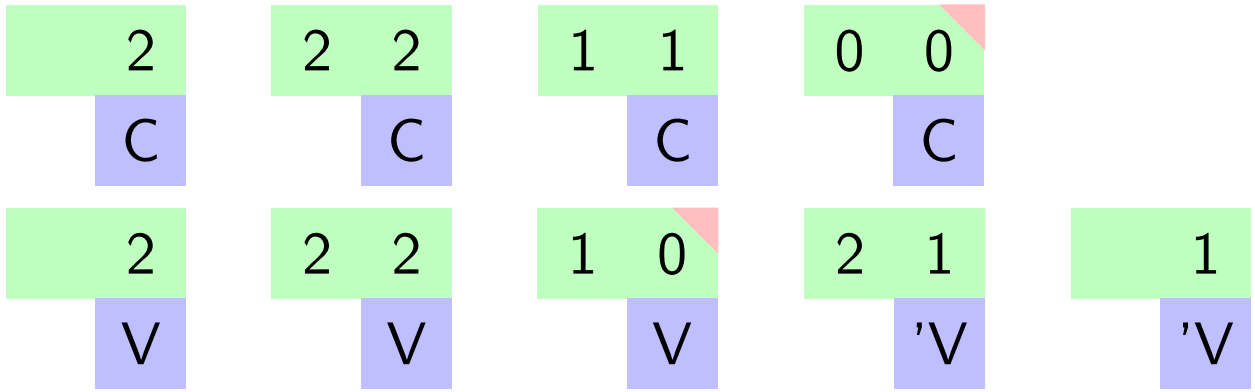


k 'e l o n t

k e l 'o n t

Example 2: Penultimate Stress

Condition: Put stress on the penultimate (= last but one) vowel
Memory: 2 distinct states 2 and 1

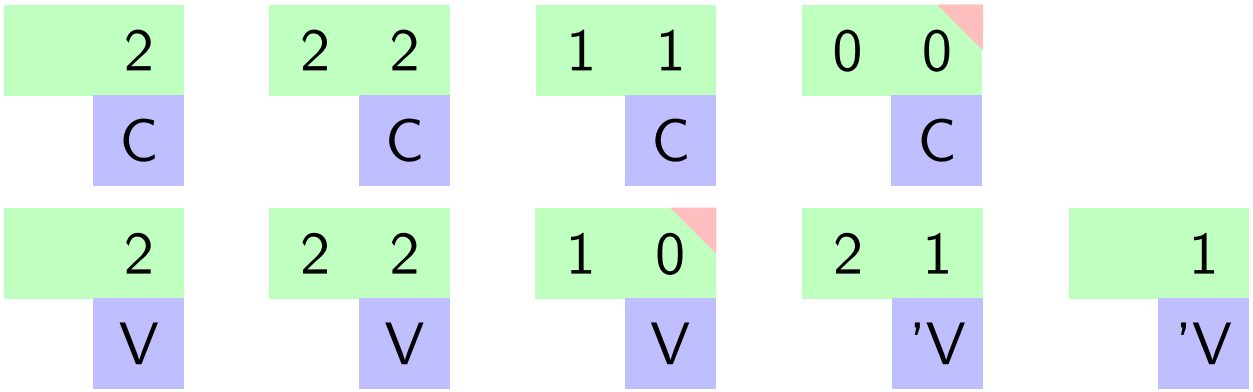


k 'e l o n t

k e l 'o n t

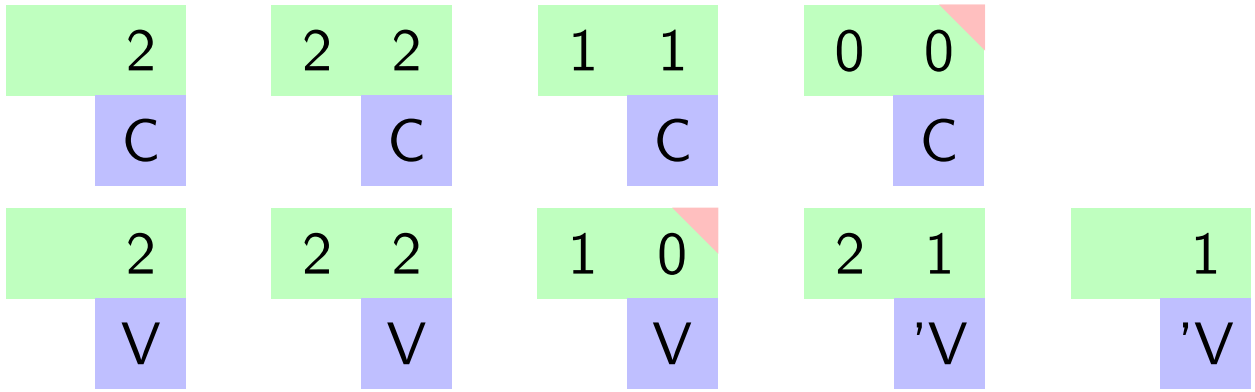
Example 2: Penultimate Stress

Condition: Put stress on the penultimate (= last but one) vowel
Memory: 2 distinct states 2 and 1



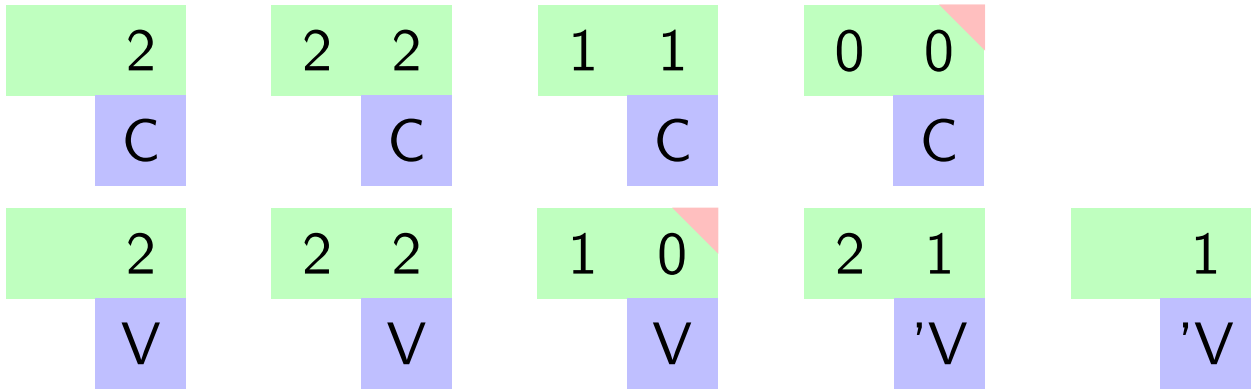
Example 2: Penultimate Stress

Condition: Put stress on the penultimate (= last but one) vowel
Memory: 2 distinct states 2 and 1



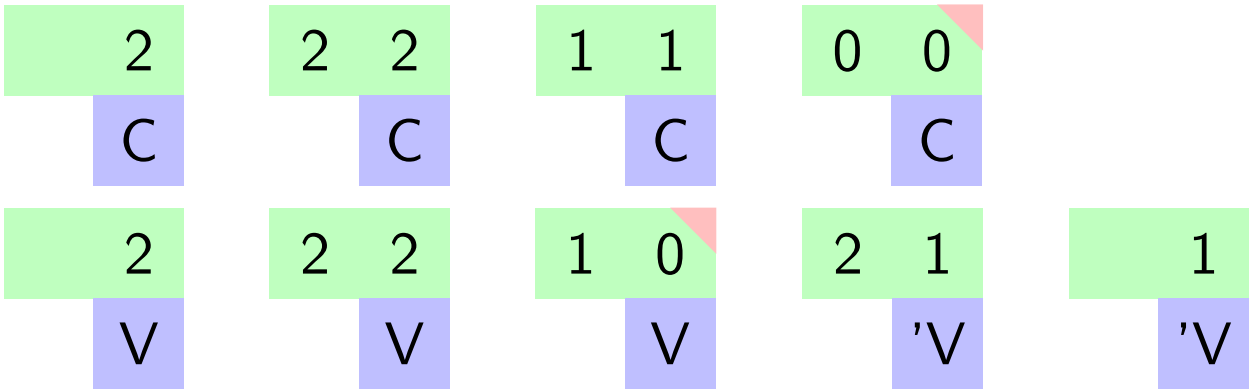
Example 2: Penultimate Stress

Condition: Put stress on the penultimate (= last but one) vowel
Memory: 2 distinct states 2 and 1



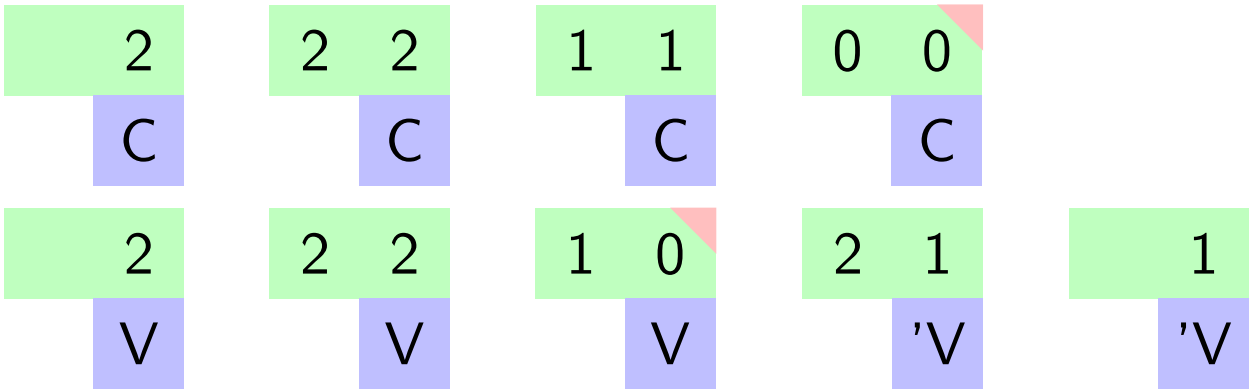
Example 2: Penultimate Stress

Condition: Put stress on the penultimate (= last but one) vowel
Memory: 2 distinct states 2 and 1



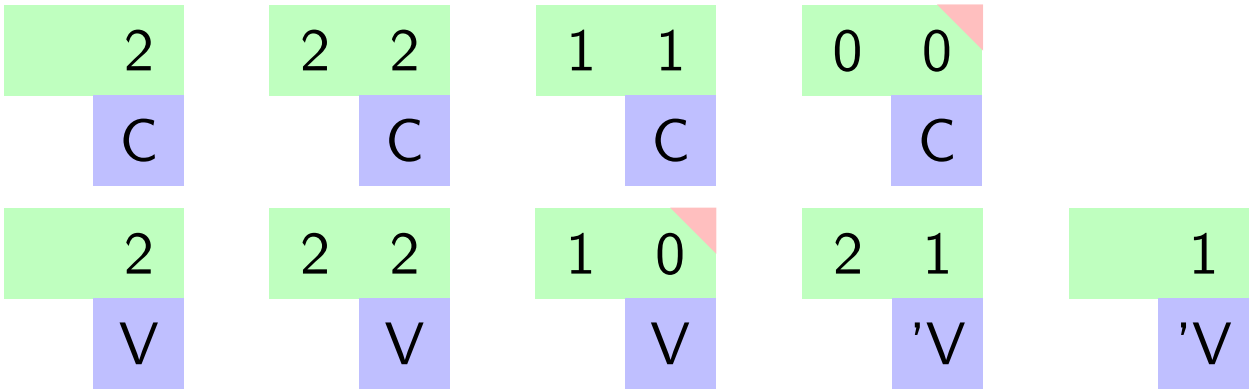
Example 2: Penultimate Stress

Condition: Put stress on the penultimate (= last but one) vowel
Memory: 2 distinct states 2 and 1



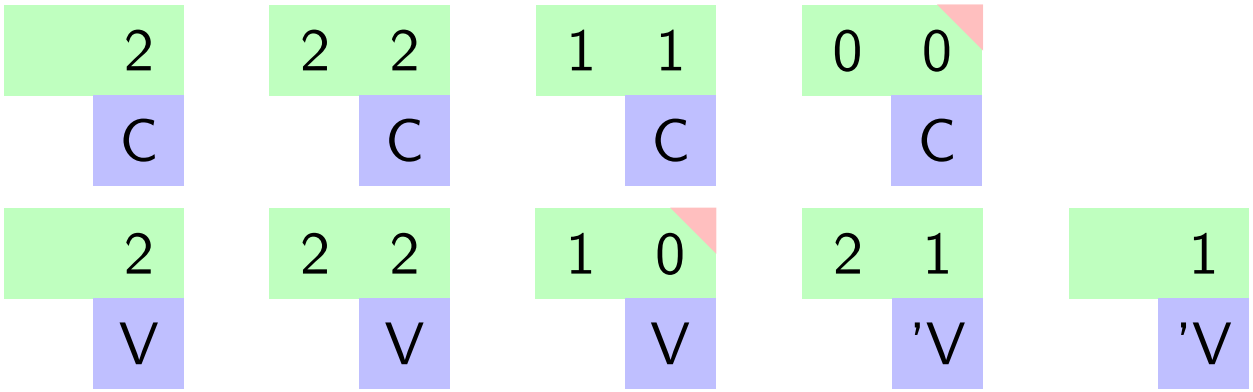
Example 2: Penultimate Stress

Condition: Put stress on the penultimate (= last but one) vowel
Memory: 2 distinct states 2 and 1



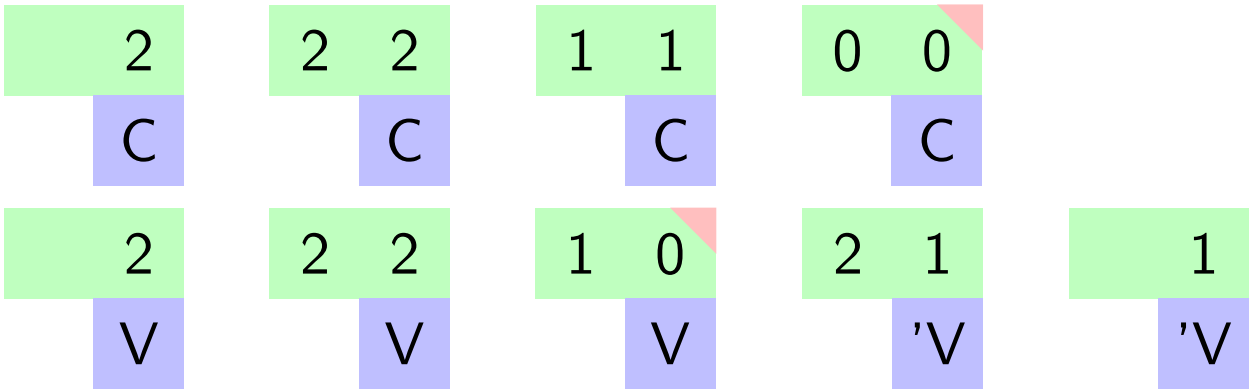
Example 2: Penultimate Stress

Condition: Put stress on the penultimate (= last but one) vowel
Memory: 2 distinct states 2 and 1



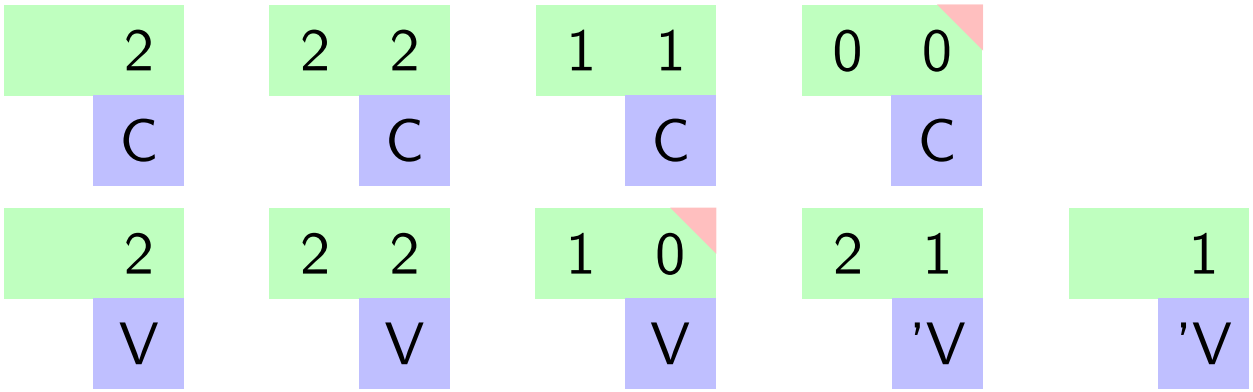
Example 2: Penultimate Stress

Condition: Put stress on the penultimate (= last but one) vowel
Memory: 2 distinct states 2 and 1



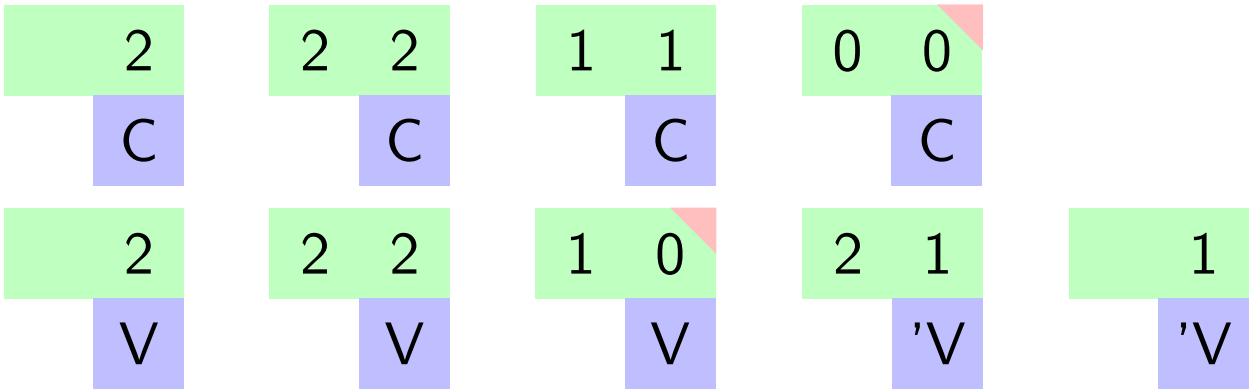
Example 2: Penultimate Stress

Condition: Put stress on the penultimate (= last but one) vowel
Memory: 2 distinct states 2 and 1



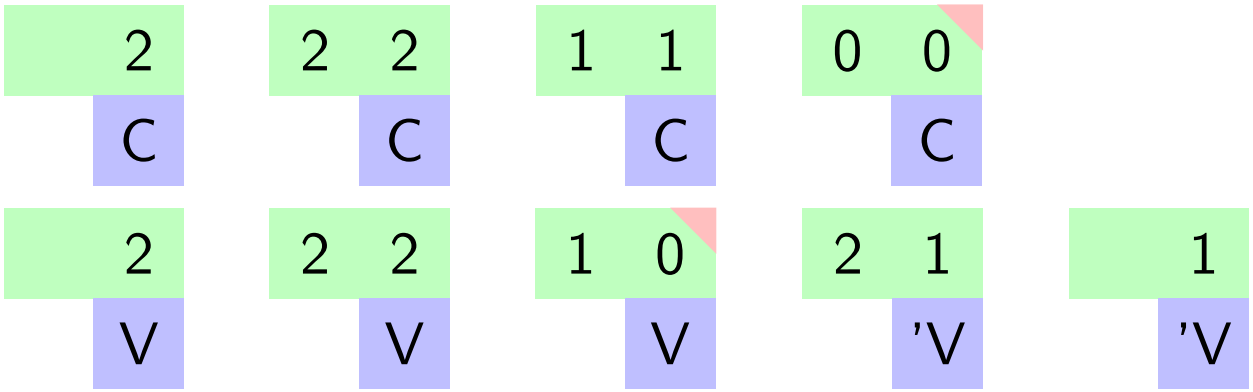
Example 2: Penultimate Stress

Condition: Put stress on the penultimate (= last but one) vowel
Memory: 2 distinct states 2 and 1



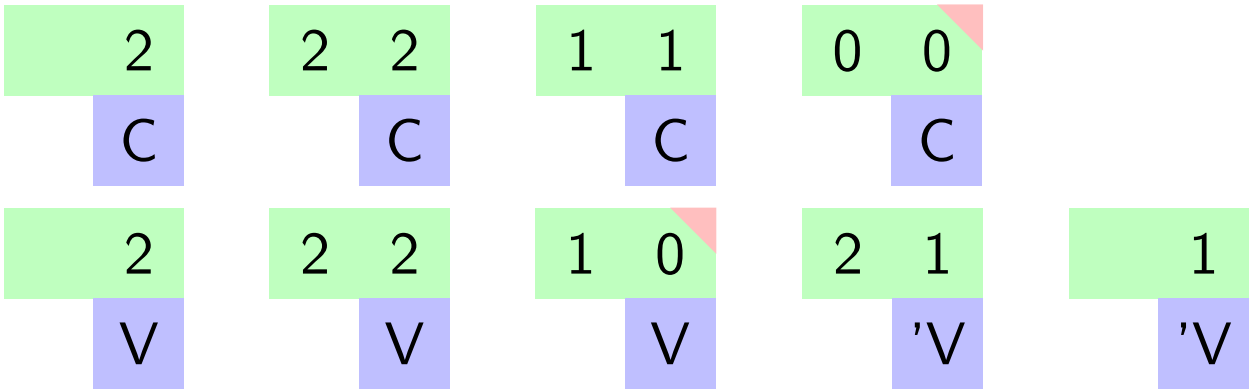
Example 2: Penultimate Stress

Condition: Put stress on the penultimate (= last but one) vowel
Memory: 2 distinct states 2 and 1



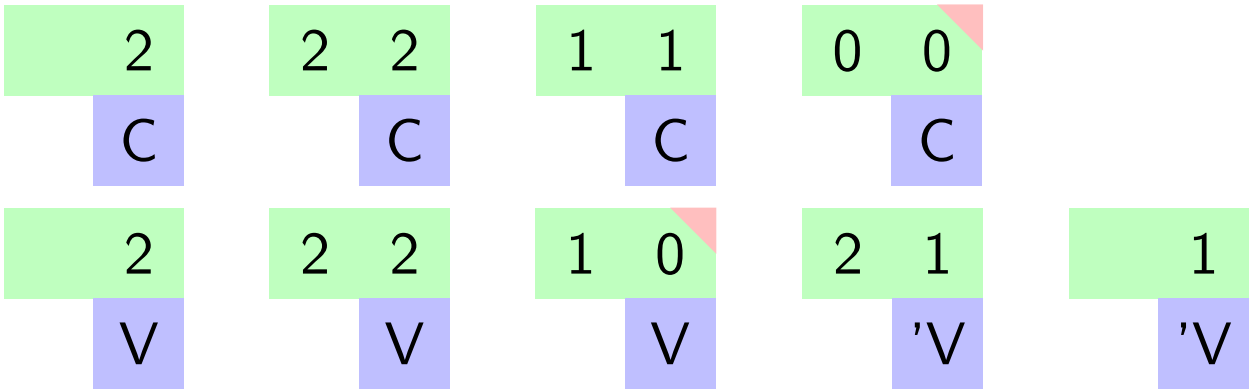
Example 2: Penultimate Stress

Condition: Put stress on the penultimate (= last but one) vowel
Memory: 2 distinct states 2 and 1



Example 2: Penultimate Stress

Condition: Put stress on the penultimate (= last but one) vowel
Memory: 2 distinct states 2 and 1



Syntax is not Finite-State

Nesting and crossing dependencies require **unbounded memory**.
(Chomsky 1956, 1959; Huybregts 1984; Shieber 1985; Radzinski 1991; Michaelis and Kracht 1997; Kobele 2006)

that J surprised me is annoying
that that J surprised me surprised me is annoying
that that that J surprised me surprised me surprised me is annoying
⋮
that^n J (surprised me)^n is annoying
⋮

The Limits of Finite-State Memory

For each level of embedding, we need at least 1 more state.
⇒ Finitely bounded memory cannot handle unbounded embedding.

Syntax is not Finite-State

Nesting and crossing dependencies require **unbounded memory**.
(Chomsky 1956, 1959; Huybregts 1984; Shieber 1985; Radzinski 1991; Michaelis and Kracht 1997; Kobele 2006)

that J surprised me is annoying
that that J surprised me surprised me is annoying
that that that J surprised me surprised me surprised me is annoying
⋮
that^n J (surprised me)^n is annoying
⋮

The Limits of Finite-State Memory

For each level of embedding, we need at least 1 more state.
⇒ Finitely bounded memory cannot handle unbounded embedding.

Syntax is not Finite-State

Nesting and crossing dependencies require **unbounded memory**.
(Chomsky 1956, 1959; Huybregts 1984; Shieber 1985; Radzinski 1991; Michaelis and Kracht 1997; Kobele 2006)

that J surprised me is annoying
that that J surprised me surprised me is annoying
that that that J surprised me surprised me surprised me is annoying
⋮
that^n J (surprised me)^n is annoying
⋮

The Limits of Finite-State Memory

For each level of embedding, we need at least 1 more state.
⇒ Finitely bounded memory cannot handle unbounded embedding.

Interim Summary

- String languages can be classified according to their complexity and matched up with specific automata models.
- These automata give us some basic cognitive facts about memory usage and architecture.
- The string patterns we find in phonology and syntax differ significantly with respect to these parameters.

	Phonology	**Syntax**
Lang. Class	regular	mildly context-sensitive
Memory	finitely bounded	unbounded

The Big Question

Why doesn't phonology have access to unbounded memory?

Interim Summary

- String languages can be classified according to their complexity and matched up with specific automata models.
- These automata give us some basic cognitive facts about memory usage and architecture.
- The string patterns we find in phonology and syntax differ significantly with respect to these parameters.

	Phonology	**Syntax**
Lang. Class	regular	mildly context-sensitive
Memory	finitely bounded	unbounded

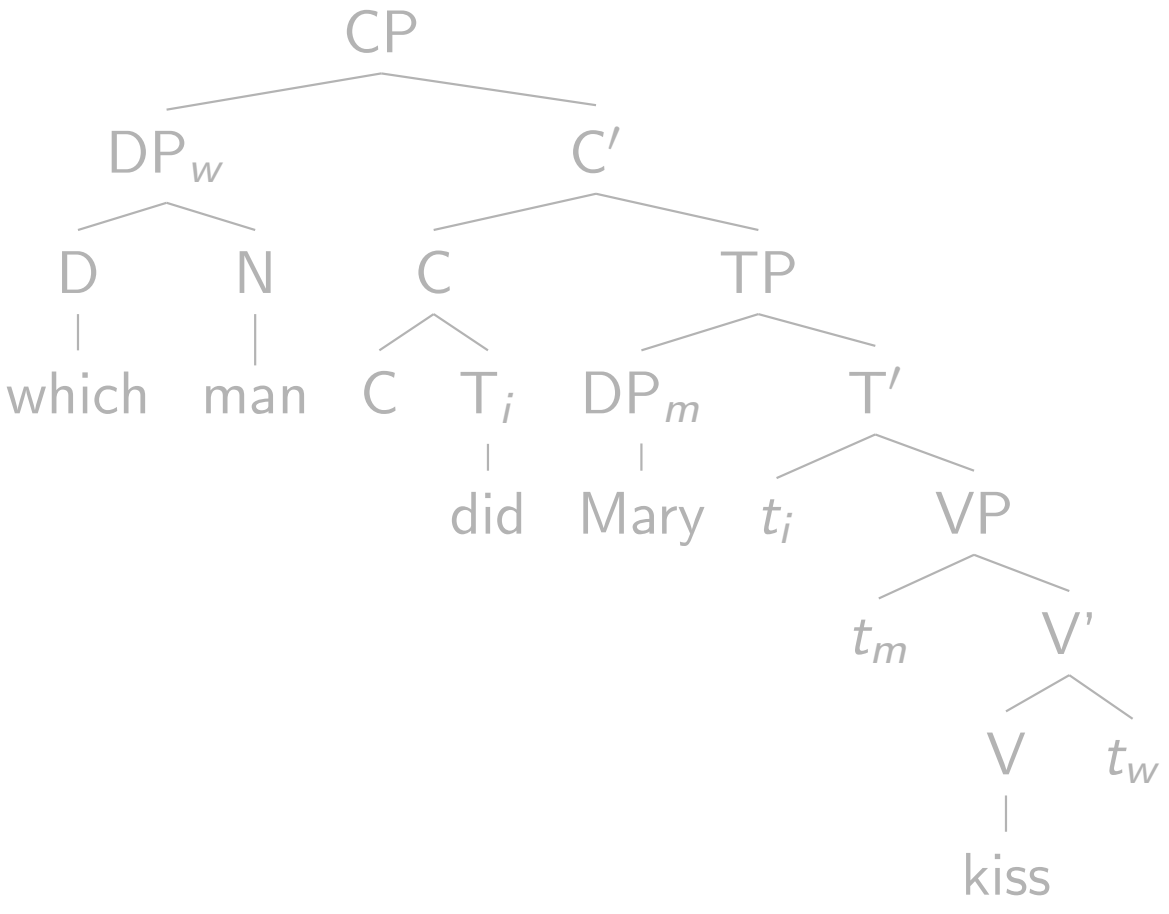
The Big Question

Why doesn't phonology have access to unbounded memory?

Outline

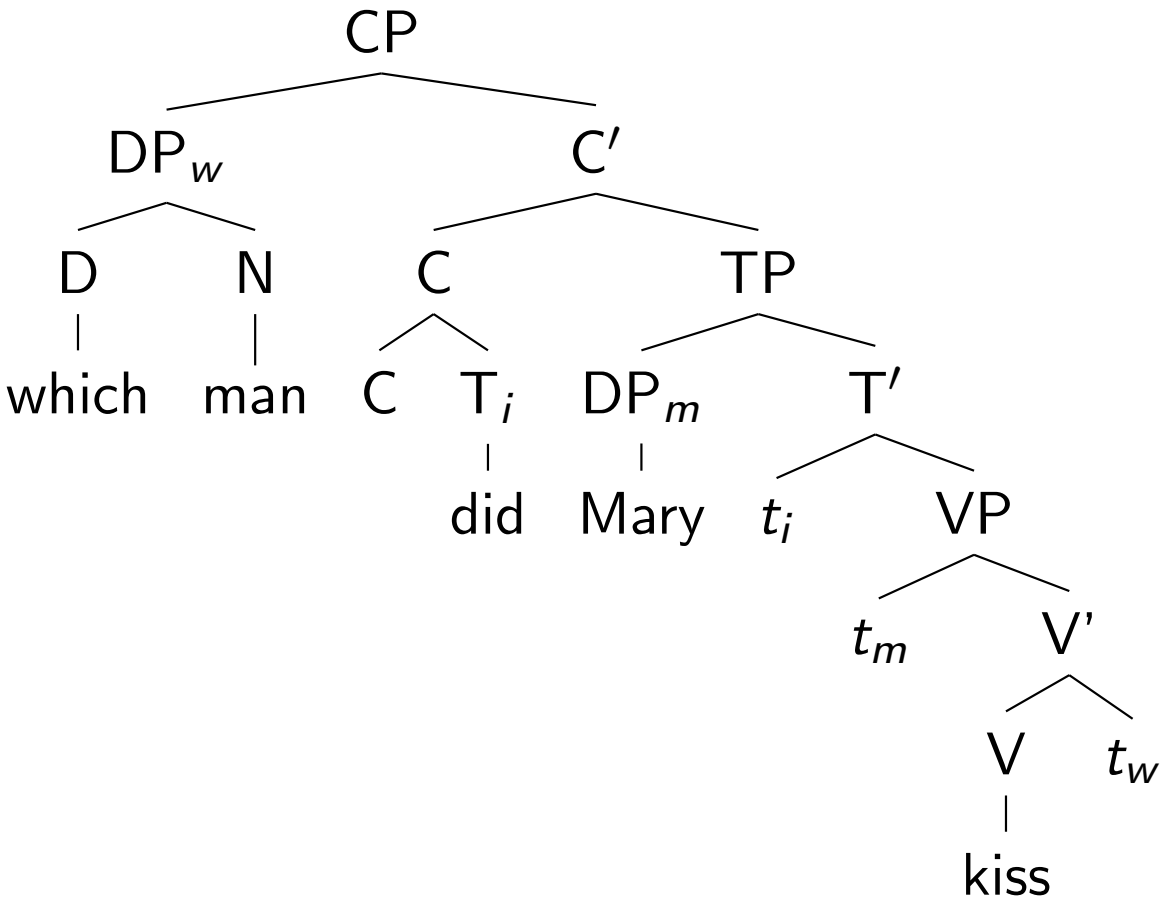
A Closer Look at Syntax

So far we have looked at syntactic patterns as string dependencies.
But **syntacticians work with trees**, not strings.



A Closer Look at Syntax

So far we have looked at syntactic patterns as string dependencies.
But **syntacticians work with trees**, not strings.

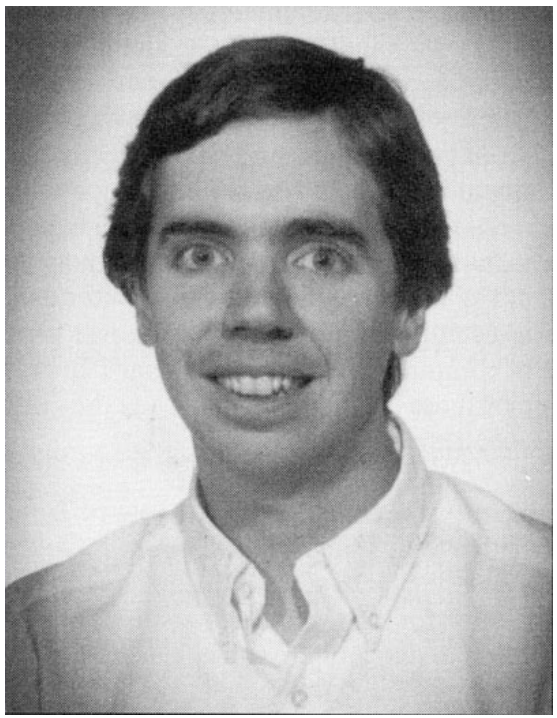


Minimalist Grammars

- **Minimalism** is the dominant syntactic theory. (Chomsky 1995)
- Can Minimalism change the computational picture of syntax? Maybe, but first we need a precise specification.
- **Minimalist grammars** are such a formalization, developed by Ed Stabler. (Stabler 1997)

Minimalist Grammars

- **Minimalism** is the dominant syntactic theory. (Chomsky 1995)
- Can Minimalism change the computational picture of syntax? Maybe, but first we need a precise specification.
- **Minimalist grammars** are such a formalization, developed by Ed Stabler. (Stabler 1997)



Syntax as Chemistry of Language

Minimalist grammars treat syntax like chemistry.

Chemistry	Syntax
atoms	words
electrons	features
molecules	sentences
stable	grammatical
unstable	ungrammatical

- Every word is a collection of features.
- Every feature has either positive or negative polarity.
- Features of opposite polarity annihilate each other.
- Feature annihilation drives the structure-building operations **Merge** and **Move**.

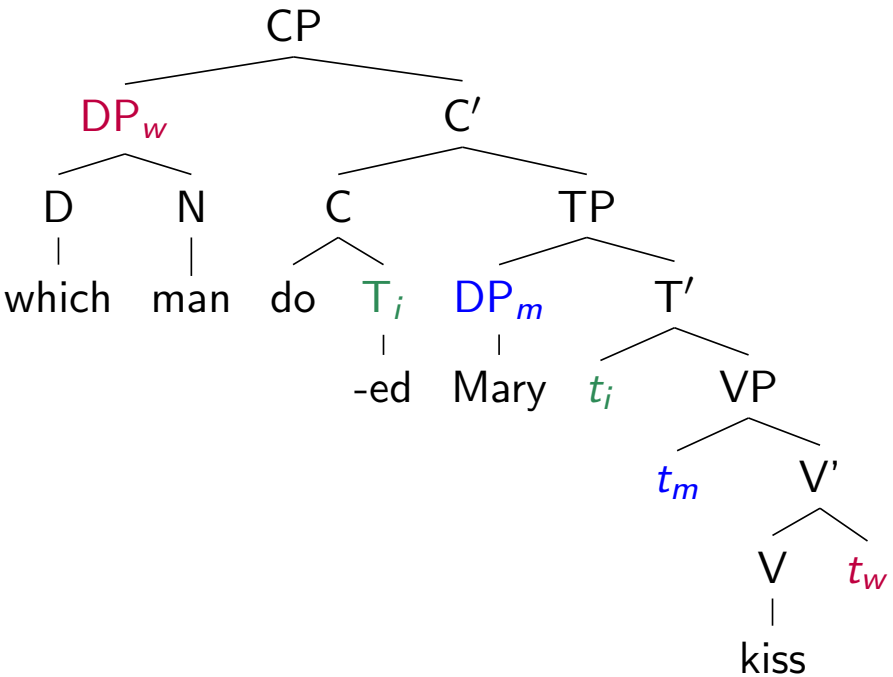
Syntax as Chemistry of Language

Minimalist grammars treat syntax like chemistry.

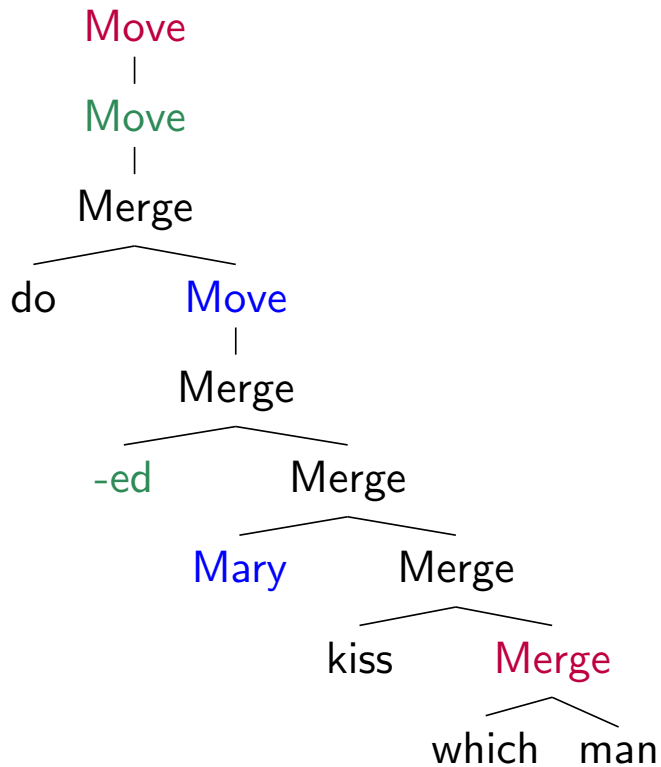
Chemistry	**Syntax**
atoms	words
electrons	features
molecules	sentences
stable	grammatical
unstable	ungrammatical

- Every word is a collection of features.
- Every feature has either positive or negative polarity.
- Features of opposite polarity annihilate each other.
- Feature annihilation drives the structure-building operations **Merge** and **Move**.

MG Syntax in Action



Phrase Structure Tree



Derivation Tree

What's the Point?

- Sentences aren't just strings, they contain hidden structure.
- Syntacticians usually look at the tree structure
  that is built by the operations Merge and Move.
- **But:** the history of how such a structure is built is also a tree
  $\Rightarrow$ **phrase structure trees** and **derivation trees** as
  two possible views of tree-based syntax

Finite-State Tree Automata

A **finite-state tree automaton** (FSTA) assigns every node in a tree one of finitely many *states*, depending on

- the label of the node, and
- the states of the nodes immediately below it (if they exist).

The FSTA accepts the tree if the highest state is a *final state*.

Reminder: FSA Definition

A finite-state automaton (FSA) assigns every node in a string one of finitely many states, depending on

- the label of the node, and
- the state of the preceding node (if it exists).

The FSA accepts the string if the last state is a *final state*.

Finite-State Tree Automata

A **finite-state tree automaton** (FSTA) assigns every node in a tree one of finitely many *states*, depending on

- the label of the node, and
- the states of the nodes immediately below it (if they exist).

The FSTA accepts the tree if the highest state is a *final state*.

Reminder: FSA Definition

A finite-state automaton (FSA) assigns every node in a string one of finitely many states, depending on

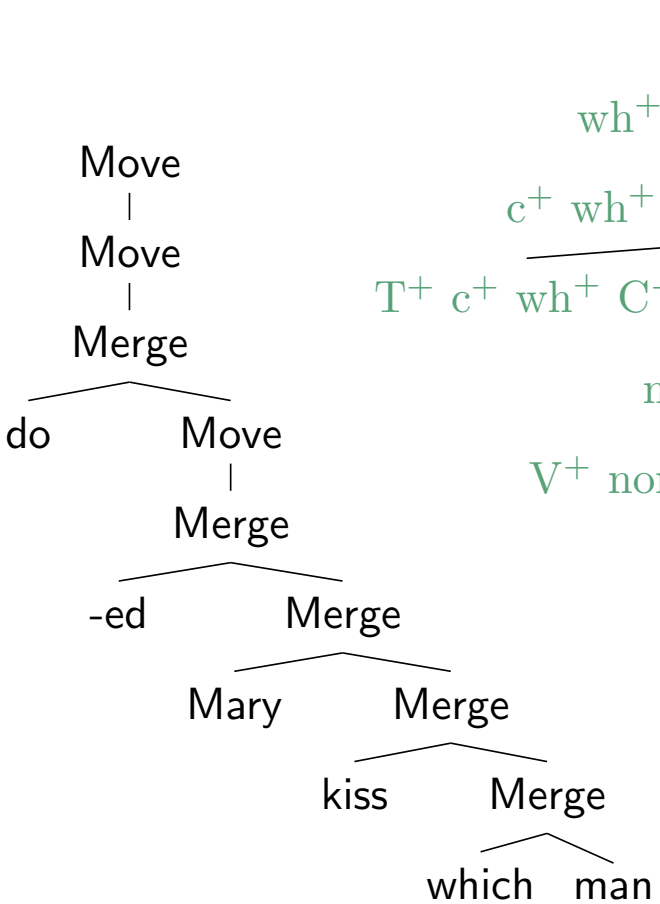
- the label of the node, and
- the state of the preceding node (if it exists).

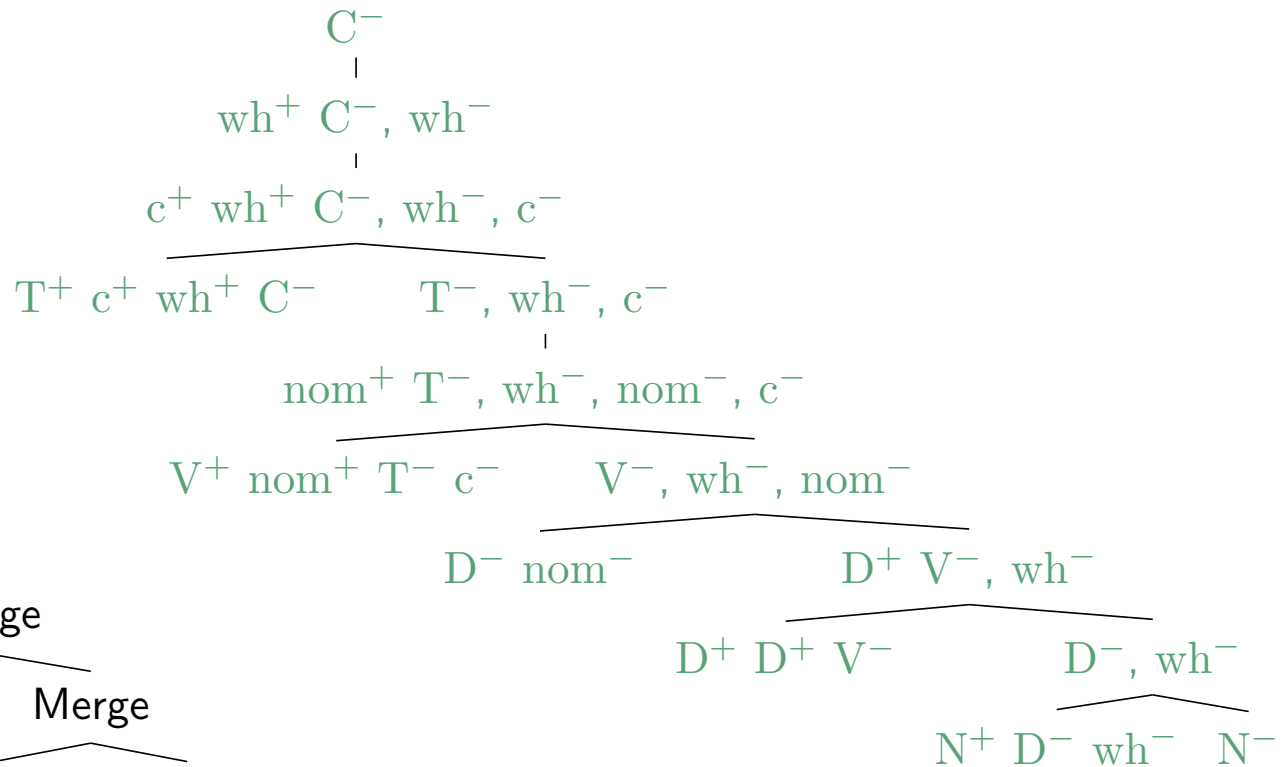
The FSA accepts the string if the last state is a *final state*.

Example: State-Assignment of Minimalist Derivation

Condition: All features must be checked except C^-

Memory: one state for every possible list of unchecked features

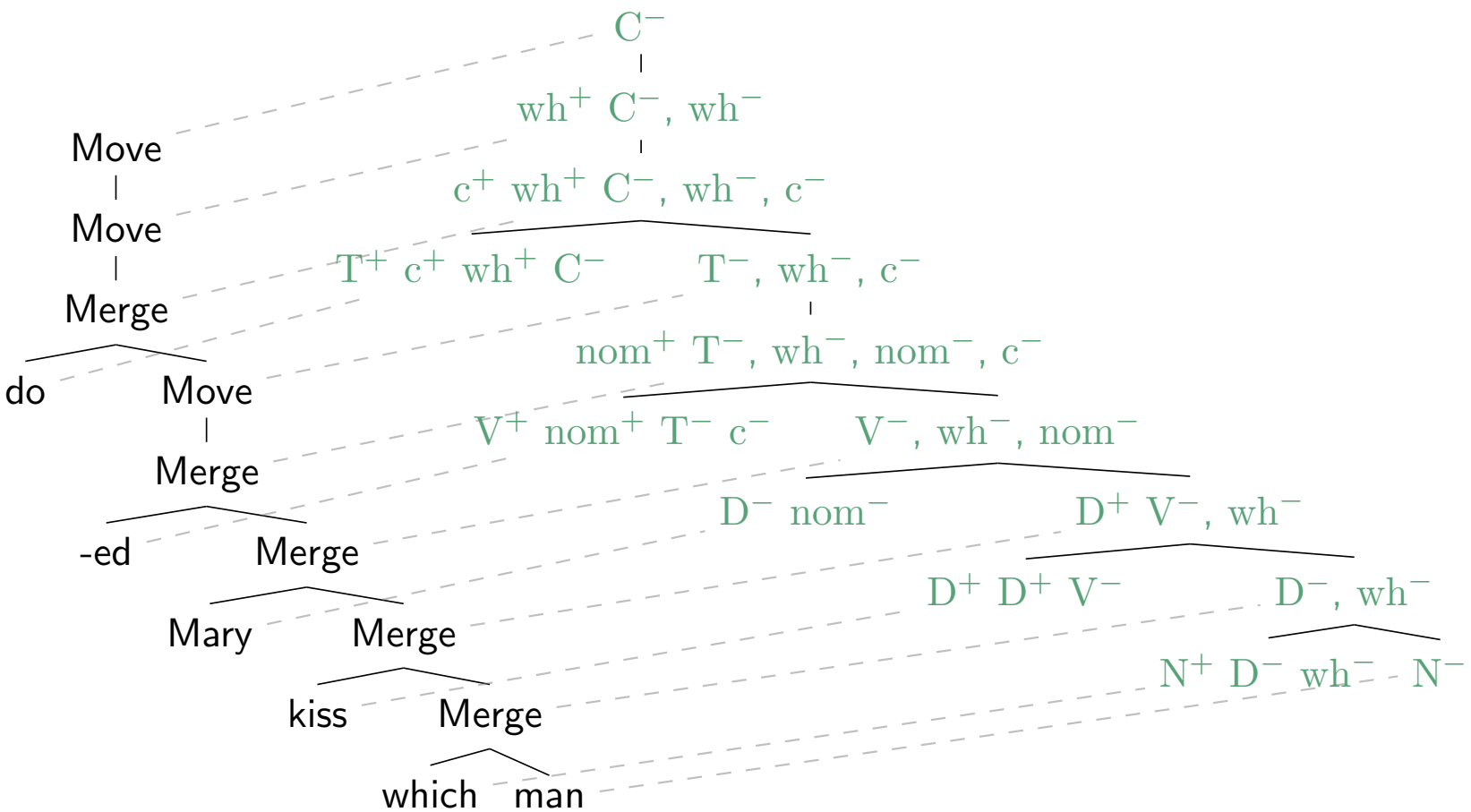




Example: State-Assignment of Minimalist Derivation

Condition: All features must be checked except C^-

Memory: one state for every possible list of unchecked features



Minimalism and FSTAs

- Phrase structure trees cannot be handled by FSTAs.
  (Harkema 2001; Michaelis 2001)
- But FSTAs are powerful enough for derivations trees.
  (Michaelis 2001; Kobele et al. 2007; Graf 2012)
- Since derivation trees are just a more abstract data structure for encoding syntactic dependencies, this means that **all syntactic dependencies can be computed with a finite amount of working memory**.

A New Perspective on Syntax and Phonology

Phonology finite working memory computations over **strings**

Syntax finite working memory computations over **trees**

Minimalism and FSTAs

- Phrase structure trees cannot be handled by FSTAs. (Harkema 2001; Michaelis 2001)
- But FSTAs are powerful enough for derivations trees. (Michaelis 2001; Kobele et al. 2007; Graf 2012)
- Since derivation trees are just a more abstract data structure for encoding syntactic dependencies, this means that **all syntactic dependencies can be computed with a finite amount of working memory**.

A New Perspective on Syntax and Phonology

Phonology finite working memory computations over **strings**

Syntax finite working memory computations over **trees**

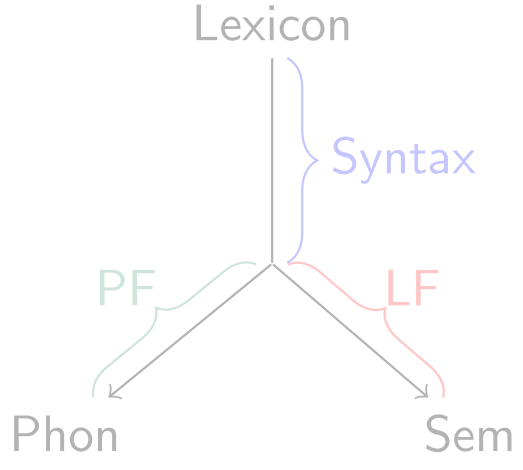
Outline

Why Trees?

The New Big Question

Why does phonology operate over strings and syntax over trees?

- **Axiom 1:** Inferring tree structure from strings is hard, so it should be avoided if possible.
- **Axiom 2:** If possible, stick with finitely bounded memory.
- **Conjecture:** Syntax must use **trees because of semantics**!

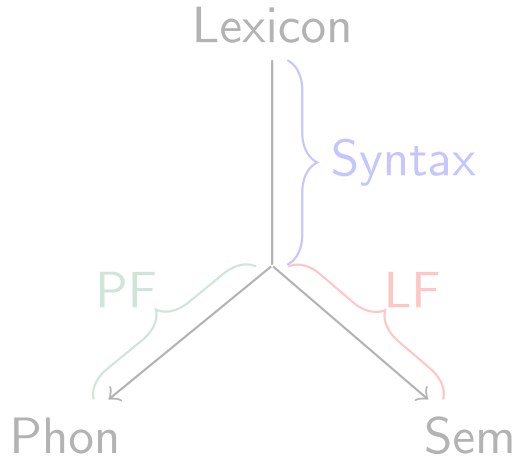


Why Trees?

The New Big Question

Why does phonology operate over strings and syntax over trees?

- **Axiom 1:** Inferring tree structure from strings is hard, so it should be avoided if possible.
- **Axiom 2:** If possible, stick with finitely bounded memory.
- **Conjecture:** Syntax must use **trees because of semantics**!

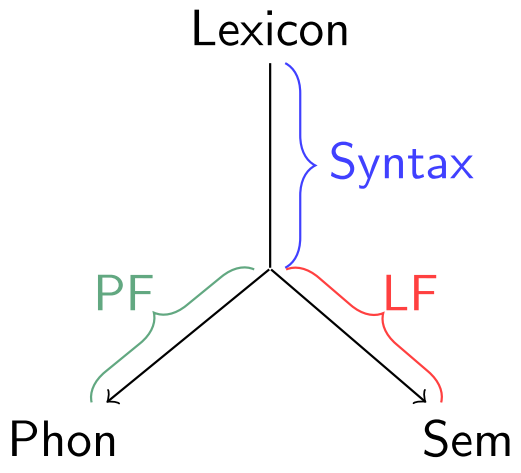


Why Trees?

The New Big Question

Why does phonology operate over strings and syntax over trees?

- **Axiom 1:** Inferring tree structure from strings is hard, so it should be avoided if possible.
- **Axiom 2:** If possible, stick with finitely bounded memory.
- **Conjecture:** Syntax must use **trees because of semantics**!



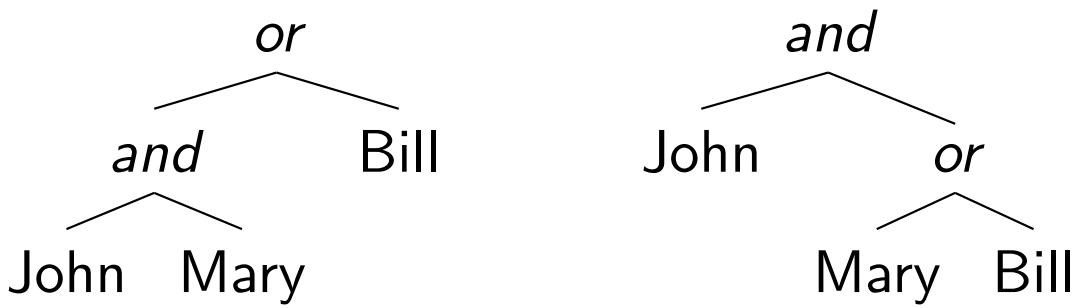
Representing Semantic Scope

Semantic scope can be represented in two ways:

- linearly via bracketing or Polish notation

bracketing	John *and* (Mary *or* Bill)	(John *and* Mary) *or* Bill
Polish	*and* John *or* Mary Bill	*or and* John Mary Bill

- graphically via trees



Memory Usage of Semantic Scope

- Bracketing and Polish notation are context-free
  $\Rightarrow$ unbounded memory
- Tree representation is finite-state $\Rightarrow$ finitely bounded memory

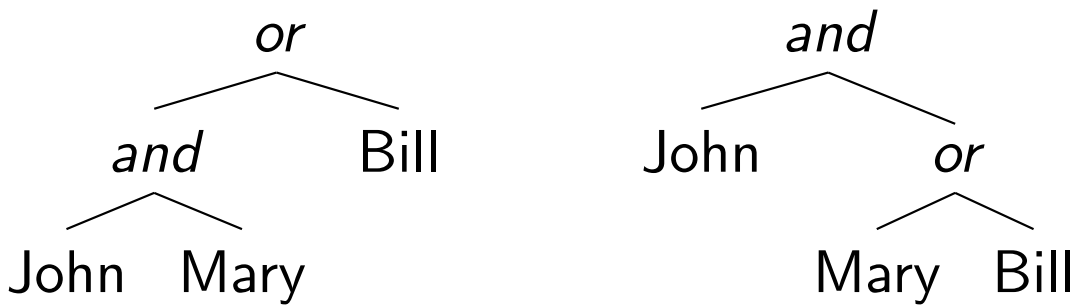
Representing Semantic Scope

Semantic scope can be represented in two ways:

- linearly via bracketing or Polish notation

bracketing	John *and* (Mary *or* Bill)	(John *and* Mary) *or* Bill
Polish	*and* John *or* Mary Bill	*or and* John Mary Bill

- graphically via trees



Memory Usage of Semantic Scope

- Bracketing and Polish notation are context-free
  $\Rightarrow$ unbounded memory
- Tree representation is finite-state $\Rightarrow$ finitely bounded memory

A Tighter Bound

Playing Devil's Advocate

Finitely bounded memory usage is not a strong restriction, syntax and phonology could occupy very different places within that class.

- **Reply:** No, there's more to this!
- Jeff Heinz has argued that phonology can be described by a small subclass of this space. (Heinz et al. 2011)
- MG derivations belong to the tree-analogue of his class! (Graf 2014)

How Deep does the Rabbit Hole Go?

- **Beyond Syntax**
  Some preliminary research of mine on generalized quantifiers suggests that large part of semantics also obey the restriction to finite working memory.
- **Beyond Language**
  Do we find similar restrictions in non-linguistic cognitive domains, e.g. music?
- **Beyond Humans**
  Birdsong has crossing dependencies like syntax, but seems to lack compositional semantics. What should we make of this?
- **Beyond Cognition**
  Protein folding also involves crossing dependencies. Is there some more abstract "folding structure" that is finite-state?

Conclusion

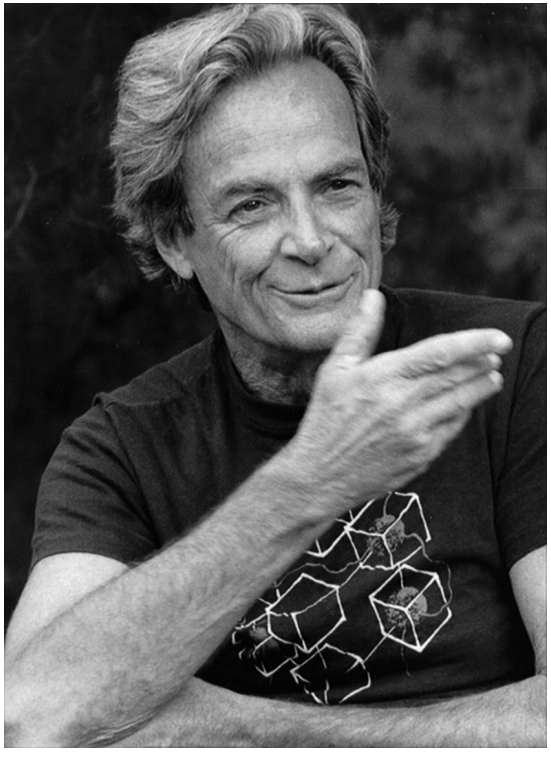
- Phonology and syntax look very different.
- But they are remarkably similar on a computational level.

Phonology finite working memory computations over **strings**

Syntax finite-working memory computations over **trees**

- The need for trees in syntax is due to semantic expressivity.
- Finite-working memory computations may lie at the center of many other cognitive and biological domains.

Final Words



Our imagination is stretched to the utmost, not, as in fiction, to imagine things which are not really there, but just to comprehend those things which are there.

(Feynman 1964:127f)

References I

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

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

Chomsky, Noam. 1988. *Language and problems of knowledge: The managua lectures*. Cambridge, MA: MIT Press.

Chomsky, Noam. 1995. *The minimalist program*. Cambridge, Mass.: MIT Press.

Feynman, Richard. 1964. *The character of physical law*. Cambridge, MA: MIT Press.

Graf, Thomas. 2012. Locality and the complexity of minimalist derivation tree languages. In *Formal Grammar 2010/2011*, ed. Philippe de Groot and Mark-Jan Nederhof, volume 7395 of *Lecture Notes in Computer Science*, 208–227. Heidelberg: Springer.

Graf, Thomas. 2014. Dependencies in syntax and phonology: A computational comparison. Slides of a talk given at NECPHON 2014, November 15, NYU, New York, New York.

Harkema, Henk. 2001. A characterization of minimalist languages. In *Logical aspects of computational linguistics (LACL'01)*, ed. Philippe de Groote, Glyn Morrill, and Christian Retoré, volume 2099 of *Lecture Notes in Artificial Intelligence*, 193–211. Berlin: Springer.

References II

Heinz, Jeffrey, Chetan Rawal, and Herbert G. Tanner. 2011. Tier-based strictly local constraints in phonology. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, 58–64.

Huybregts, M. A. C. 1984. The weak adequacy of context-free phrase structure grammar. In *Van periferie naar kern*, ed. Ger J. de Haan, Mieke Trommelen, and Wim Zonneveld, 81–99. Dordrecht: Foris.

Kaplan, Ronald M., and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics* 20:331–378.

Kobele, Gregory M. 2006. *Generating copies: An investigation into structural identity in language and grammar*. Doctoral Dissertation, UCLA.

Kobele, Gregory M., Christian Retoré, and Sylvain Salvati. 2007. An automata-theoretic approach to minimalism. In *Model Theoretic Syntax at 10*, ed. James Rogers and Stephan Kepser, 71–80.

Michaelis, Jens. 2001. Transforming linear context-free rewriting systems into minimalist grammars. *Lecture Notes in Artificial Intelligence* 2099:228–244.

Michaelis, Jens, and Marcus Kracht. 1997. Semilinearity as a syntactic invariant. In *Logical Aspects of Computational Linguistics*, ed. Christian Retoré, volume 1328 of *Lecture Notes in Artifical Intelligence*, 329–345. Springer.

References III

Radzinski, Daniel. 1991. Chinese number names, tree adjoining languages, and mild context sensitivity. *Computational Linguistics* 17:277–300.

Shieber, Stuart M. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy* 8:333–345.

Stabler, Edward P. 1997. Derivational minimalism. In *Logical aspects of computational linguistics*, ed. Christian Retoré, volume 1328 of *Lecture Notes in Computer Science*, 68–95. Berlin: Springer.