Computational Unity Across Language Modules

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

Moscow State University

December 15, 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 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
1. Linguistic Subsystems: Syntax and Phonology

2. Memory Usage of Dependencies in Syntax and Phonology
   - Formal Language Theory
   - Phonology Uses Bounded Working Memory
   - Syntax Uses Unbounded Working Memory

3. A Linguistically Informed Look at Syntax
   - Minimalist Syntax
   - Syntactic Derivations and Working Memory

4. Deeper Down the Rabbit Hole
   - Why Trees in Syntax?
   - Where to go From Here
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: $\text{nef}+\text{ið} \rightarrow \text{nevið}$ (Icelandic)
  - word-final devoicing: $\text{rad} \rightarrow \text{rat}$ (German)
  - *intervocalic devoicing: $\text{aba} \rightarrow \text{apa}$ (unattested)
  - dissimilation: $\text{lun}+\text{alis} \rightarrow \text{lunarís}$ (Latin)
  - umlaut: $\text{mamm}+\text{u} \rightarrow \text{mömmu}$ (Icelandic)
  - *anti-umlaut: $\text{mömm}+\text{u} \rightarrow \text{mammu}$ (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:

  - island effects $\equiv$ blocking
  - negative concord $\equiv$ vowel harmony
  - gapping:ellipsis $\equiv$ deletion:
  - Principle C $\equiv$ ?
  - ? $\equiv$ hiatus
  - PCC $\equiv$ ?
  - :
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

\[
\begin{array}{c}
\text{TP} \\
\text{DP} \\
\text{John} \\
T' \\
T \\
\text{-s} \\
\text{t} \\
\text{VP} \\
V' \\
V \\
\text{DP} \\
\text{like} \\
\text{Mary}
\end{array}
\]

\[
\begin{array}{c}
\{(?,A)\} \\
\{(?,I)\} \\
\{(U),A\}(\{L,?,\},A)
\end{array}
\]
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

1. Linguistic Subsystems: Syntax and Phonology

2. Memory Usage of Dependencies in Syntax and Phonology
   - Formal Language Theory
   - Phonology Uses Bounded Working Memory
   - Syntax Uses Unbounded Working Memory

3. A Linguistically Informed Look at Syntax
   - Minimalist Syntax
   - Syntactic Derivations and Working Memory

4. Deeper Down the Rabbit Hole
   - Why Trees in Syntax?
   - Where to go From Here
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 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)
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.
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:** ∫ cannot be followed by s
**Memory:** 3 distinct states ✓, ∫, and *

```
∫ e g o ∫ a
```

```
∫ *
```

```
g e ∫ 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:** $\int$ cannot be followed by $s$

**Memory:** 3 distinct states $\checkmark$, $\int$, and $*$

\[
\begin{array}{cccc}
\checkmark & \checkmark & \int & \int \\
\int & \checkmark & \int & \int \\
\int & \int & \int & \int \\
\int & \int & \int & s \\
\int & e & g & o & \int & a \\
g & e & \int & o & s & a \\
\end{array}
\]
Example 1: Sibilant Harmony

**Condition:** ∫ cannot be followed by s

**Memory:** 3 distinct states ✓, ∫, and *

```
    ✓
    ✓ ✓
    ✓ ✓
    ✓ ∫
    ✓ ∫
    ✓ ∫
    ∫ *

    ∫ ∫
    ∫ e g o ∫ a

    g e ∫ o s a
```
Example 1: Sibilant Harmony

**Condition:** \( \text{j} \) cannot be followed by \( s \)

**Memory:** 3 distinct states \( \checkmark, \text{j}, \) and \( * \)
Example 1: Sibilant Harmony

**Condition:** \( \mathcal{S} \) cannot be followed by \( s \)

**Memory:** 3 distinct states \( \checkmark, \mathcal{S}, \) and \( * \)

\[
\begin{array}{cccc}
\text{\checkmark} & \text{\checkmark} & \mathcal{S} & \mathcal{S} \\
\mathcal{S} & \mathcal{S} & \mathcal{S} & \mathcal{S} \\
\mathcal{S} & \mathcal{S} & \mathcal{S} & * \\
\mathcal{S} & \mathcal{S} & \mathcal{S} & s \\
\mathcal{S} & \mathcal{S} & \mathcal{S} & \mathcal{S} \\
\mathcal{S} & \mathcal{S} & \mathcal{S} & \mathcal{S} \\
\mathcal{S} & \mathcal{S} & \mathcal{S} & \mathcal{S} \\
\mathcal{S} & \mathcal{S} & \mathcal{S} & \mathcal{S} \\
\end{array}
\]

\[
\text{g e f o s a}
\]
Example 1: Sibilant Harmony

**Condition:** \( \mathfrak{j} \) cannot be followed by \( s \)

**Memory:** 3 distinct states \( \checkmark, \mathfrak{j}, \) and \( * \)
Example 1: Sibilant Harmony

**Condition:** ∫ cannot be followed by s

**Memory:** 3 distinct states ✓, ∫, and *

```
∫ ✓
✓ ✓
✓ ✓
∫ ✓
✓ ✓
∫ ✓
∫ ✓
∫ ✓
∫ *
∫ ∫ ∫ ∫ ∫ ∫
∫ e g o ∫ a
g e ∫ o s a
```
Example 1: Sibilant Harmony

**Condition:** ∫ cannot be followed by s

**Memory:** 3 distinct states ✓, ∫, and *

```
  ✓  ✓  ✓
  ∫  ∫  ∫
  ∫  ∫  ∫  *

∫ e g o ∫ a

g e ∫ o s a
```
Example 1: Sibilant Harmony

**Condition:** $\overline{f}$ cannot be followed by $s$

**Memory:** 3 distinct states $\checkmark$, $\overline{f}$, and $*$

$\overline{f}$

$\overline{f}$

$\overline{f}$

$\overline{f}$

$\overline{f}$

$\overline{f}$

$e\ g\ o\ f\ a$

$g\ e\ \overline{f}\ o\ s\ a$
Example 1: Sibilant Harmony

**Condition:** ∫ cannot be followed by s

**Memory:** 3 distinct states ✓, ∫, and *

```
✓   ✓ ✓   ✓   ✓   ✓
∫   ∫   ∫   ∫   ∫
∫   ∫   ∫   ∫   ∫
∫   ∫   ∫   ∫   ∫
✓ ✓
∫ e g o ∫ a
g e ∫ o s a
```
Example 1: Sibilant Harmony

**Condition:** \( \emptyset \) cannot be followed by s

**Memory:** 3 distinct states ✓, \( \emptyset \), and *

\[
\begin{array}{c}
✓ \\
✓
\end{array}
\quad
\begin{array}{c}
✓ \checkmark
d\
✓
\end{array}
\quad
\begin{array}{c}
\checkmark
\checkmark
\
\checkmark
\end{array}
\quad
\begin{array}{c}
\checkmark
\checkmark
\
\checkmark
\end{array}

\begin{array}{c}
\checkmark
\
\checkmark
\end{array}
\quad
\begin{array}{c}
\checkmark
\
\checkmark
\end{array}
\quad
\begin{array}{c}
\checkmark
\checkmark
\
\checkmark
\end{array}
\quad
\begin{array}{c}
\checkmark
\checkmark
\
s
\end{array}
\quad
\begin{array}{c}
\checkmark
\
\checkmark
\end{array}
\quad
\begin{array}{c}
\checkmark
\
\checkmark
\end{array}
\quad
\begin{array}{c}
\checkmark
\
\checkmark
\end{array}

\begin{array}{c}
\checkmark
\checkmark
\
\checkmark
\end{array}
\quad
\begin{array}{c}
\checkmark
\
\checkmark
\end{array}
\quad
\begin{array}{c}
\checkmark
\
\checkmark
\end{array}
\quad
\begin{array}{c}
\checkmark
\
\checkmark
\end{array}

\begin{array}{c}
\checkmark
\
e
g\ o\ a
\end{array}
\quad
\begin{array}{c}
\checkmark
\checkmark
\
ge\ i\ o\ s\ a
\end{array}
Example 1: Sibilant Harmony

**Condition:** \( \text{ʃ} \) cannot be followed by \( s \)

**Memory:** 3 distinct states \( \checkmark, \text{ʃ}, \) and \( * \)
Example 1: Sibilant Harmony

**Condition:** $\mathcal{S}$ cannot be followed by $s$

**Memory:** 3 distinct states $\checkmark$, $\mathcal{S}$, and $*$
**Example 2: Penultimate Stress**

**Condition:** Put stress on the penultimate (= last but one) vowel

**Memory:** 2 distinct states 2 and 1

```
2  2  2  1  1  0  0
C  C  C  C  C  C

2  2  2  1  0  2  1
V  V  V  'V 'V 1
```

```
k'elont
```

```
k'el'ont
```
Example 2: Penultimate Stress

**Condition:** Put stress on the penultimate (last but one) vowel

**Memory:** 2 distinct states 2 and 1

```
 2    2    1    1    0    0
C    C    C    C    C    C

2    2    1    0    2    1
V    V    V    V    'V    'V
```

```
k'el on 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

```
2  C
2  C
1  C
0  C

2  V
2  V
1  V
2  'V
1  'V
```

```
2
k 'el 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

```
 2  2  1  0
C  C  C  C
2  2  1  1
V  V  V  'V
```

```
 2  1
k  e  l  o  n  t
```

```
 2  1
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

```
<table>
<thead>
<tr>
<th>2</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>V</td>
<td>V</td>
<td>V</td>
<td>'V</td>
</tr>
</tbody>
</table>

```

2

kelont

kelont
### Example 2: Penultimate Stress

**Condition:** Put stress on the penultimate (last but one) vowel

**Memory:** 2 distinct states 2 and 1

<table>
<thead>
<tr>
<th>2</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>V</td>
<td>V</td>
<td>V</td>
<td>'V</td>
</tr>
</tbody>
</table>

```
2110
kelont
```
Example 2: Penultimate Stress

**Condition:** Put stress on the penultimate (last but one) vowel

**Memory:** 2 distinct states 2 and 1

```
2  1  0  0
C  C  C  C
V  V  V  V
k 'e l o n t
```

```
2  2  2
C  C  C
V  V  V
```

```
1  0
V  V
```

```
0  0
C  C
V  V
```

```
2  1  1
V  V  'V
```

```
1  1  0  0
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

```
2  2  1  1  0  0  0
C  C  C  C  V  V  V

2  2  2  1  0  2  1
V  V  V  V  'V  'V

k 'el'ont
```
Example 2: Penultimate Stress

**Condition:** Put stress on the penultimate (last but one) vowel

**Memory:** 2 distinct states 2 and 1

```
2   2   1   1   0   0   0
C   C   C   C   C   C
2   2   1   0   2   1   1
V   V   V   V   'V   'V

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

```
2 2 2 1 1 0 0 0
C C C C C C C
V V V V V V V
k 'e l o n t
```

```
2 1 1 0 0 0
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

```
2 2 1 1 1 0 0 0
C C C V V V 'V 'V
k e l o n t
```

```
2 2 2 1 0 0 1
C C C V V ',V ',V
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

```
C  C  C
V  V  V
```

```
2  1  0  0  0
k 'e l o n t
```

```
0  0
C
```

```
2  1
'V
1
```

```
2  2  1  1
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

<table>
<thead>
<tr>
<th>2</th>
<th>2 2</th>
<th>1 1</th>
<th>0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>2 2</td>
<td>1 0</td>
<td>2 1</td>
</tr>
<tr>
<td>V</td>
<td>V</td>
<td>V</td>
<td>'V</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2 1 1 0 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>kel'ont</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2 2 2 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>kel'o'nt</td>
</tr>
</tbody>
</table>
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.
\[\Rightarrow\] Finitely bounded memory cannot handle unbounded embedding.
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.

<table>
<thead>
<tr>
<th></th>
<th>Phonology</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lang. Class</td>
<td>regular</td>
<td>mildly context-sensitive</td>
</tr>
<tr>
<td>Memory</td>
<td>finitely bounded</td>
<td>unbounded</td>
</tr>
</tbody>
</table>

The Big Question

Why doesn’t phonology have access to unbounded memory?
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.

<table>
<thead>
<tr>
<th>Phonology</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lang. Class</td>
<td>regular</td>
</tr>
<tr>
<td>Memory</td>
<td>finitely bounded</td>
</tr>
<tr>
<td></td>
<td>mildly context-sensitive</td>
</tr>
<tr>
<td></td>
<td>unbounded</td>
</tr>
</tbody>
</table>

The Big Question

Why doesn’t phonology have access to unbounded memory?
Outline

1 Linguistic Subsystems: Syntax and Phonology

2 Memory Usage of Dependencies in Syntax and Phonology
   - Formal Language Theory
   - Phonology Uses Bounded Working Memory
   - Syntax Uses Unbounded Working Memory

3 A Linguistically Informed Look at Syntax
   - Minimalist Syntax
   - Syntactic Derivations and Working Memory

4 Deeper Down the Rabbit Hole
   - Why Trees in Syntax?
   - Where to go From Here
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)
Syntax as Chemistry of Language

Minimalist grammars treat syntax like chemistry.

<table>
<thead>
<tr>
<th>Chemistry</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>atoms</td>
<td>words</td>
</tr>
<tr>
<td>electrons</td>
<td>features</td>
</tr>
<tr>
<td>molecules</td>
<td>sentences</td>
</tr>
<tr>
<td>stable</td>
<td>grammatical</td>
</tr>
<tr>
<td>unstable</td>
<td>ungrammatical</td>
</tr>
</tbody>
</table>

- 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**.
Minimalist grammars treat syntax like chemistry.

### Chemistry
- atoms
- electrons
- molecules
- stable
- unstable

### Syntax
- words
- features
- sentences
- grammatical
- 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
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

⇒ 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.*
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

```
    C^-
     |  wh^+ C^-, wh^-
     |  c^+ wh^+ C^-, wh^-, c^-
     |  T^+ c^+ wh^+ C^-  T^-, wh^-, c^-
     |  nom^+ T^-, wh^-, nom^-, c^-
     |  V^+ nom^+ T^- c^-  V^-, wh^-, nom^-
     |  D^- nom^-  D^+ V^-, wh^-
     |  D^+ D^+ V^-  D^-, wh^-
     |  N^+ D^- wh^-  N^-
```

- `do` Move
  - `ed` Merge
    - Mary Merge
      - `kiss` Merge
        - which man
Example: State-Assignment of Minimalist Derivation

**Condition:** All features must be checked except $C^-$

**Memory:** one state for every possible list of unchecked features

```
C^-
  | wh^+ C^-, wh^-
  | c^+ wh^+ C^-, wh^-, c^-
    | T^+ c^+ wh^+ C^- T^-, wh^-, c^-
    | nom^+ T^-, wh^-, nom^-, c^-
      | V^+ nom^+ T^- c^- V^-, wh^-, nom^-
        | D^- nom^- D^- V^-, wh^-
          | D^- D^+ V^- D^-, wh^-
            | N^+ D^- wh^- N^-
```
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

<table>
<thead>
<tr>
<th>Phonology</th>
<th>finite working memory computations over <strong>strings</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>finite working memory computations over <strong>trees</strong></td>
</tr>
</tbody>
</table>
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

1. Linguistic Subsystems: Syntax and Phonology

2. Memory Usage of Dependencies in Syntax and Phonology
   - Formal Language Theory
   - Phonology Uses Bounded Working Memory
   - Syntax Uses Unbounded Working Memory

3. A Linguistically Informed Look at Syntax
   - Minimalist Syntax
   - Syntactic Derivations and Working Memory

4. Deeper Down the Rabbit Hole
   - Why Trees in Syntax?
   - Where to go From Here
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

```
          or
         / \
        /   \
      or    and
     /     /  /
    and  Bill John
   /     /    /
  John  Mary Bill
```

Memory Usage of Semantic Scope

- Bracketing and Polish notation are context-free  
  ⇒ unbounded memory
- Tree representation is finite-state  
  ⇒ finitely bounded memory
Representing Semantic Scope

Semantic scope can be represented in two ways:

- linearly via bracketing or Polish notation
  
  **bracketing**  
  \[ (\text{John} \text{ and} (\text{Mary or Bill})) \text{ or Bill} \]
  
  **Polish**  
  \[ \text{and John or Mary Bill} \text{ or and John Mary Bill} \]

- graphically via trees

 Memory Usage of Semantic Scope

- Bracketing and Polish notation are context-free  
  \[ \Rightarrow \text{unbounded memory} \]

- Tree representation is finite-state  
  \[ \Rightarrow \text{finitely bounded memory} \]
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?
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.
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)


Tiers in Phonology: Sibilant Harmony

Rewrite rule

\[ s \rightarrow \int \mid \int \cdots \cdot \]

Constraint

\[ \ast \int \cdots s \int s \]

Tier_1 - Bigram

Tier_1 contains all sibilants

Tier_0 contains all segments

Tier_1:

\[
\begin{array}{c}
\$ \int s \$ \\
\mid \mid \mid \mid \mid \\
\end{array}
\]

Tier_0:

\[
\begin{array}{c}
\$ e \int i s i \$ \\
\mid \mid \mid \mid \mid \\
\end{array}
\]

Tier_1:

\[
\begin{array}{c}
\$ \int \int \$ \\
\mid \mid \mid \mid \mid \\
\end{array}
\]

Tier_0:

\[
\begin{array}{c}
\$ e \int i \int i \$ \\
\mid \mid \mid \mid \mid \\
\end{array}
\]
Tiers for Movement

```
Move
/\  
merge
 Move
/  
merge
 Move
/  
merge
 Merge
```

a b c d e f
Tiers for Movement

```
Move
  | Merge
  |
Move  Merge
  |    |
  | Move f
  |     |
  | Merge
  |     |
  | Merge
  |   a b
d e
```
Tiers for Movement
Example of Ill-Formed Derivation
Example of Ill-Formed Derivation
Example of Ill-Formed Derivation
Example of Ill-Formed Derivation
Example of Ill-Formed Derivation
Example of Ill-Formed Derivation

References
Example of Ill-Formed Derivation
Example of Ill-Formed Derivation

Merge

$\quad \text{Move} \quad g$

Merge

$\quad \text{Merge} \quad c \quad \text{Move} \quad f$

Merge $\quad a \quad b$

Merge $\quad d \quad e$

Move

$a \quad f$

Move

$g$

Move

$g$

Move

SMC1

SMC2
Example of Well-Formed Derivation

```
$  Move  $  Move  
  LI  $  LI  LI
Move  SMC1  SMC2
```

```
Move  Merge
  |  
  Merge  Move  f
  |  
  Merge  c  Merge
  |  
  a  b  d  e
```
Example of Well-Formed Derivation
Example of Well-Formed Derivation
Example of Well-Formed Derivation

```
References

Move       Move
|         |
Merge      Merge

Move       Merge
|         |
Move       f
|
Merge     
|    |
c      d
|
Merge

a  b  d  e
```

```
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|
Move
|  

SMC1
SMC2
$$
LI
$$
$$
LI
$$
```
Example of Well-Formed Derivation