Dependencies in Syntax and Phonology: A Computational Comparison

Thomas Graf (and Jeff Heinz, sorta)
mail@thomasgraf.net
http://thomasgraf.net

Department of Linguistics
Stony Brook University

NECPHON
November 15 2014
Today’s Topic

The Received View on Syntax and Phonology

- Little cross-talk between syntax and phonology
- Properties of one supposedly have no bearing on the other
- Huge difference with respect to weak generative capacity

Counter Position

- Computationally, phonology and syntax are very similar.
- Over linguistically plausible models, both rely on dependencies of the same computational complexity.
- Main difference is data structures: \textit{strings} versus \textit{trees}
Today’s Topic

The Received View on Syntax and Phonology

- Little cross-talk between syntax and phonology
- Properties of one supposedly have no bearing on the other
- Huge difference with respect to weak generative capacity

Counter Position

- Computationally, phonology and syntax are very similar.
- Over linguistically plausible models, both rely on dependencies of the same computational complexity.
- Main difference is data structures: **strings** versus **trees**
Outline

1. The Received View
   - Linguistics
   - Measuring Generative Capacity

2. Phonology is Tier-Based Strictly Local
   - \( n \)-gram Grammars
   - Tiers for Non-Local Dependencies

3. A Closer Look at Syntax
   - Minimalist Grammars
   - A Quick Example
   - The Shortest Move Constraint

4. Underlyingly, Syntax is Tier-Based Strictly Local
   - MGs as Regular Derivation Tree Languages
   - Merge is Strictly Local
   - Move is Tier-Based Strictly Local
Linguistics: Syntax and Phonology are Unrelated

- **Different Frameworks**
  - Aspects ⇔ SPE
  - GB ⇔ Autosegmental/GP
  - Minimalism ⇔ OT

- **Empirical Separation**
  - What is the syntactic analog of umlaut or final devoicing?
  - What is the phonological analog of passive or the Person Case Constraint?
  - Cognitive impairments often impact only one of the two.
Formally, 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 program, human genome, Shakespeare’s oeuvre*
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)
Pinpointing Phonology in the Chomsky Hierarchy

- Regular languages are too powerful for phonology.
- Jeff Heinz has argued that phonology can be described by a natural generalization of $n$-gram grammars. (Heinz et al. 2011)
- **Idea:** non-local dependencies are **local on phonological tiers**
(Negative) Bigram Grammars

- Suppose we have a fixed alphabet $\Sigma$ (e.g. sounds of English).
- A bigram is a sequence $ab$ s.t. $a, b \in \Sigma$.
- A **bigram grammar** $G$ is a finite set of bigrams.
- $G$ generates the largest language of strings such that no string contains any bigrams of $G$ as a substring.
- Intuition: **bigrams are hard, local constraints**

### Example

<table>
<thead>
<tr>
<th>Rewrite rule</th>
<th>Constraint</th>
<th>Bigrams</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n \rightarrow m</td>
<td>b$</td>
<td>$*nb$</td>
</tr>
<tr>
<td>$z \rightarrow s</td>
<td>$</td>
<td>$*z$</td>
</tr>
<tr>
<td>[-voice] $\rightarrow \emptyset</td>
<td>$</td>
<td>$* [-voice]$</td>
</tr>
</tbody>
</table>
(Negative) Bigram Grammars

- Suppose we have a fixed alphabet $\Sigma$ (e.g. sounds of English).
- A bigram is a sequence $ab$ s.t. $a, b \in \Sigma$.
- A bigram grammar $G$ is a finite set of bigrams.
- $G$ generates the largest language of strings such that no string contains any bigrams of $G$ as a substring.
- Intuition: bigrams are hard, local constraints

Example

<table>
<thead>
<tr>
<th>Rewrite rule</th>
<th>Constraint</th>
<th>Bigrams</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n \rightarrow m \</td>
<td>_ \ b$</td>
<td>*nb</td>
</tr>
<tr>
<td>$z \rightarrow s \</td>
<td>_ \ $</td>
<td>*z$</td>
</tr>
<tr>
<td>[-voice] $\rightarrow \emptyset \</td>
<td>_ \ $</td>
<td>*[voice]$</td>
</tr>
</tbody>
</table>
Tiers for Long-Distance Dependencies

- We can move to 3-grams, 4-grams, \ldots \ n\text{-}grams in order to regulate less local processes (e.g. umlaut).
- **Problem:** Still limited to locality domain of size \( n \)\n  \[ \Rightarrow \] unbounded processes cannot be captured
- **Solution:** segments can be on multiple tiers

Tier-Based Bigram Grammar

- Let \( T \subseteq \Sigma \) be a tier alphabet.
- A **tier-based bigram grammar** \( G \) is a pair of finite sets of bigrams over \( \Sigma \) and \( T \), respectively.
- \( G \) generates \( s \) iff
  - \( s \) has no \( \Sigma \)-bigram as a substring,
  - the restriction of \( s \) to elements of \( T \) has no \( T \)-bigram as a substring.
Tiers for Long-Distance Dependencies

- We can move to 3-grams, 4-grams, \ldots n-grams in order to regulate less local processes (e.g. umlaut).
- **Problem:** Still limited to locality domain of size $n$ \Rightarrow unbounded processes cannot be captured
- **Solution:** segments can be on multiple tiers

**Tier-Based Bigram Grammar**

- Let $T \subseteq \Sigma$ be a tier alphabet.
- A **tier-based bigram grammar** $G$ is a pair of finite sets of bigrams over $\Sigma$ and $T$, respectively.
- $G$ generates $s$ iff
  - $s$ has no $\Sigma$-bigram as a substring,
  - the restriction of $s$ to elements of $T$ has no $T$-bigram as a substring.
Example: Sibilant Harmony

Rewrite rule
\[ s \rightarrow \int \mid \int \ldots \]  

Constraint
\[ \ast \int \ldots s \]

Tier-Bigram
\[ \int s \]

T-Tier: \[ $ \int s $ \]

Σ-Tier: \[ $ e \int i s i $ \]
Example: Primary Stress Assignment

- Every word must have exactly one primary stress.
- Let $T$ be the set of symbols with primary stress.
- Then we need the following $T$-bigrams:
  - $$ “at least one primary stress”
  - $ab$ “not more than one primary stress” (for all $a, b \in T$)

<table>
<thead>
<tr>
<th>Stress Missing</th>
<th>Too Many Stresses</th>
<th>Just Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e i s i$</td>
<td>$'e 'i s i$</td>
<td>$'e i s i$</td>
</tr>
<tr>
<td>$e i s i$</td>
<td>$'e 'i s i$</td>
<td>$'e 'i s i$</td>
</tr>
</tbody>
</table>
Phonology is Subregular

- Tier-based $n$-gram grammars generate only a subclass of the regular languages.
- Only a few known phenomena might not to be tier-local, but the data is unclear (e.g. primary stress assignment in Cairene Arabic; Graf 2010)
- Hence **phonology is subregular**.

**Core insights**
- Most phonological dependencies are local.
- Non-local dependencies are local between the elements of the relevant type ($\approx$ tier).
A Closer Look at Syntax

The MCS-result treats 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)
- They are expressive enough for syntax.
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**.
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**.
Merge: Example 1

Assembling \([\text{DP the men}]\)

\[
\begin{array}{c}
\text{the} \\
N^+ \quad D^- \\
\text{men} \\
N^-
\end{array}
\]

- Features of opposite polarities annihilate
- Annihilation triggers Merge, which builds structure on top
Merge: Example 1

Assembling \[\text{DP the men}\]

\[
\begin{array}{ll}
\text{the} & \text{men} \\
N^+ D^- & N^- \\
\end{array}
\]

- Features of opposite polarities annihilate
- Annihilation triggers Merge, which builds structure on top
Merge: Example 1

Assembling \([DP \text{ the men}]\)

\[
\begin{array}{c}
\text{the} \\
\text{N}^+ \text{ D}^-
\end{array}
\begin{array}{c}
\text{men} \\
\text{N}^-
\end{array}
\]

- Features of opposite polarities annihilate
- Annihilation triggers Merge, which builds structure on top
Merge: Example 1

Assembling \([\text{DP the men}]\)

\[ \begin{array}{c}
\text{D} \\
\text{the} & \text{men} \\
\text{N}^+ & \text{D}^- & \text{N}^- 
\end{array} \]

- Features of opposite polarities annihilate
- Annihilation triggers Merge, which builds structure on top
Merge: Example 1

Assembling \([\text{DP the men}]\)

- Features of opposite polarities annihilate
- Annihilation triggers Merge, which builds structure on top
Assembling \([ VP \text{ the men like which men} ]\)

- the and men merged as before
- same steps for which men
- like merged with which men
- like merged with the men
Merge: Example 2

**Assembling** \([VP \ the \ men \ like \ which \ men]\)

- **the** and **men** merged as before
- same steps for **which men**
- **like** merged with **which men**
- **like** merged with **the men**
Assembling \[ VP \text{ the men like which men } \]

- *the* and *men* merged as before
- same steps for *which men*
- *like* merged with *which men*
- *like* merged with *the men*
Assembling \([VP \text{ the men like which men}]\)

- **the** and **men** merged as before
- **same steps for which men**
- **like** merged with **which men**
- **like** merged with **the men**
Merge: Example 2

Assembling $[VP \text{ the men like which men}]$

- *the* and *men* merged as before
- same steps for *which men*
- *like* merged with *which men*
- *like* merged with *the men*
Merge: Example 2

Assembling \([_{VP} \text{the men like which men}]\)

- \(the\) and \(men\) merged as before
- same steps for \(which\) \(men\)
- \(like\) merged with \(which\) \(men\)
- \(like\) merged with \(the\) \(men\)
Merge: Example 2

Assembling \([VP \text{ the men like which men}]\)

- The and men merged as before
- Same steps for which men
- Like merged with which men
- Like merged with the men
Merge: Example 2

Assembling \[
\text{VP the men like which men}
\]

- **the** and **men** merged as before
- same steps for **which men**
- **like** merged with **which men**
- **like** merged with **the men**
Merge: Example 2

Assembling \[ \text{VP the men like which men} \]

- the and men merged as before
- same steps for which men
- like merged with which men
- like merged with the men
Merge: Example 2 [cont.]

The tree diagram shows the structure of a sentence involving merging of Merge[D]
and Merge[N]. The diagram is as follows:

```
  V
 /\  
D  V
 /\  
D  N
 |   |
the men like which men
N+ D- N- D+ D+ V- N+ D-
```

The tree illustrates the process of merging in a sentence where 'the men' and the phrase 'which men' merge with 'like'.
Assembling “which men do the men like?”

- Merge *do*
- Move triggered by features of opposite polarity
Assembling “which men do the men like?”

- **Move**
- **Merge** *do*
- **Move triggered by features of opposite polarity**
Move

Assembling “which men do the men like?”

- **Merge** *do*
- **Move** triggered by features of opposite polarity
Assembling “which men do the men like?”

- Merge *do*
- Move triggered by features of opposite polarity
**Move**

Assembling “which men do the men like?”

- **Merge** *do*
- **Move triggered by features of opposite polarity**
Derivation Trees with Move

```
C
  /     \
 /       \\
/         \\nD which men
  /     \
 /       \\
/         \\nN+ D- wh- N-

C
  /     \\n /       \\
/         \\nD do
  /     \\n /       \\
/         \\nV+ wh+ C-

V
  /     \\n /       \\
/         \\nD
  /     \\n /       \\
/         \\nV V C
  /     \\n /       \\
/         \\nD V C

Move[wh]
  /     \
 /       \\
/         \\nV+ wh+ C-

Merge[V]
  /     \
 /       \\
/         \\nV+ wh+ C-

Merge[N]
  /     \
 /       \\
/         \\nN+ D- the

Merge[D]
  /     \\n /       \\
/         \\nd men

Merge[D]
  /     \\n /       \\
/         \\ndo

Merge[N]
  /     \\n /       \\
/         \\nd like

Merge[N]
  /     \\n /       \\
/         \\nd which

Merge[N]
  /     \\n /       \\
/         \\nd men

N+ D- N-

D+ D+ V-

N+ D- wh- N-
```
An Important Restriction on MGs

In order to ensure that MGs generate only MCS-languages, movement must be restricted.

**Shortest Move Constraint (SMC)**

If two lexical items in a tree both have a negative Move feature as their first currently unchecked feature, then these features must be distinct.
An Important Restriction on MGs

In order to ensure that MGs generate only MCS-languages, movement must be restricted.

**Shortest Move Constraint (SMC)**

If two lexical items in a tree both have a negative Move feature as their first currently unchecked feature, then these features must be distinct.

```
which < men
\_ < like \_ < which \_ < men
N^+ D^- wh^- N^- D^+ D^+ V^- N^+ D^- wh^- N^-  
```
An Important Restriction on MGs

In order to ensure that MGs generate only MCS-languages, movement must be restricted.

Shortest Move Constraint (SMC)

If two lexical items in a tree both have a negative Move feature as their first currently unchecked feature, then these features must be distinct.

```
which < men > like < which < men
N^+ D^- wh^- N^- D^+ D^+ V^- N^+ D^- wh^- N^-```


An Important Restriction on MGs

In order to ensure that MGs generate only MCS-languages, movement must be restricted.

**Shortest Move Constraint (SMC)**

If two lexical items in a tree both have a negative Move feature as their first currently unchecked feature, then these features must be distinct.
An Important Restriction on MGs

In order to ensure that MGs generate only MCS-languages, movement must be restricted.

**Shortest Move Constraint (SMC)**

If two lexical items in a tree both have a negative Move feature as their first currently unchecked feature, then these features must be distinct.

```
N^+ D^- wh^- N^- D^+ D^+ V^- N^+ D^- wh^- N^- 
```

“Don’t cross the streams!”
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
  ⇒ **phrase structure trees** and **derivation trees** as two possible views of tree-based syntax
The Complexity of MGs

- Since syntax is described by trees, we should look at tree languages instead of string languages.
- Every MG can be identified with
  - its set of phrase structure trees, or
  - its set of derivation trees
- The set of phrase structure trees is not regular.
  (Doner 1970; Thatcher 1967)
- But the set of derivation trees is regular.
  (Michaelis 2001; Kobele et al. 2007; Graf 2012)

The Big Question
Are MG derivation tree languages tier-based strictly local?
The Complexity of MGs

- Since syntax is described by trees, we should look at tree languages instead of string languages.
- Every MG can be identified with
  - its set of phrase structure trees, or
  - its set of derivation trees
- The set of phrase structure trees is not regular. (Doner 1970; Thatcher 1967)
- But the set of derivation trees is regular. (Michaelis 2001; Kobele et al. 2007; Graf 2012)

The Big Question
Are MG derivation tree languages tier-based strictly local?
Tree $n$-gram Grammars

- We need to lift $n$-grams from strings to trees.
- Instead of strings of length $n$, use subtrees of depth $n$.
- Each subtree encodes a constraint on the derivation.

Example: Bigram template for merging only matching LIs

```
Merge
   head    arg
```

where head and arg lack matching Merge features
An LI’s Merge features are checked by its arguments.

The distance between a head and the head of its argument is bounded by some factor $k$ (which depends on how many arguments a head may take).

Hence Merge dependencies are $n$-local for some fixed $n$.

It follows that **Merge is tier-based strictly local**.
Constraints on Move

Suppose our MG is in **single movement normal form**.

<table>
<thead>
<tr>
<th>Occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given lexical item $l$ with negative Move feature $f^-$, a node $m$ is an occurrence of $l$ iff $m$ is the lowest node dominating $l$ that can check $f^-$.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Movement Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Move</strong></td>
</tr>
<tr>
<td>Every lexical item with a negative Move feature has exactly one occurrence.</td>
</tr>
<tr>
<td><strong>SMC</strong></td>
</tr>
<tr>
<td>Every Move node is an occurrence of exactly one lexical item.</td>
</tr>
</tbody>
</table>
Tiers for Movement

- There is no upper bound on the distance between a lexical item and its occurrence.
- Consequently, **Move dependencies are not strictly local**.
- What if every movement type (wh, topic, ...) induces its own tier? Would that make Move dependencies local?
Tiers for Movement

- There is no upper bound on the distance between a lexical item and its occurrence.
- Consequently, **Move dependencies are not strictly local**.
- What if every movement type (wh, topic, ...) induces its own tier? Would that make Move dependencies local?

```
Move
  | Merge
  |
Move  Merge
  |      |
  |
Move  Merge
  |      |
  |
Merge  Move f
  |      |
  |
Merge  Merge
  |      |
  |
Merge  Merge
  |      |
  |
Merge  Merge
  |      |
  |
Merge  Merge
  |      |
  |
 a  b  d  e
```
There is no upper bound on the distance between a lexical item and its occurrence.

Consequently, Move dependencies are not strictly local.

What if every movement type (wh, topic, . . .) induces its own tier? Would that make Move dependencies local?
Tiers for Movement

- There is no upper bound on the distance between a lexical item and its occurrence.
- Consequently, **Move dependencies are not strictly local.**
- What if every movement type (wh, topic, . . . ) induces its own tier? Would that make Move dependencies local?
Tree Bigrams for Move

Move amounts to the following constraints over each tier:

- **Move**
  Every lexical item has a mother labeled Move.

- **SMC**
  Among the daughters of a Move node there exists exactly one that is a lexical item.

![Tree Bigram Templates](image-url)

```
<table>
<thead>
<tr>
<th>Move1</th>
<th>Move2</th>
<th>SMC</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>LI</td>
<td></td>
</tr>
<tr>
<td>LI</td>
<td>...</td>
<td>LI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
```

```
The Problem With Our Bigrams

- No limit on number of daughters per Move node in tier
  $\Rightarrow$ Move2 and SMC correspond to infinitely many bigrams
- But a bigram grammar must be finite!

A Hint from Multidimensional Trees

- We think of trees as nodes ordered by dominance and precedence.
- Jim Rogers (2003) formalizes trees as strings (sequences of siblings) related by dominance.
- Analogously, a tree-tier may consist of string-tiers related by dominance!
Checking the Example Derivation

General Verification Procedure

- Take derivation and project Move tiers.
- In every Move tier, project LI-tiers.
- For every node, build a bigram consisting of the node and the LI-tier of its daughter string.
From Templates to Tree Bigrams with String Tiers

- Each string of siblings is given an LI-tier.
- The tree bigrams only reference the LI-tier.

### Old Tree Bigram Templates

<table>
<thead>
<tr>
<th>Move1</th>
<th>Move2</th>
<th>SMC</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>LI</td>
<td>Move</td>
</tr>
<tr>
<td>LI</td>
<td>...</td>
<td>LI</td>
</tr>
</tbody>
</table>

### New Tree Bigrams with String Tiers

<table>
<thead>
<tr>
<th>Move1</th>
<th>Move2</th>
<th>SMC1</th>
<th>SMC2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>LI</td>
<td>Move</td>
<td>Move</td>
</tr>
<tr>
<td>LI</td>
<td>LI</td>
<td>$</td>
<td>LI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$</td>
<td>LI</td>
</tr>
</tbody>
</table>

$LI$
Conclusion

- Syntax and phonology look very different, but computationally they are very similar.
- Phonology is tier-based strictly local over strings.
- Syntax is tier-based strictly local over derivation trees.

**Intuition**

- Non-local dependencies are not particularly complex.
- They are *local over some relativization domain*. 


