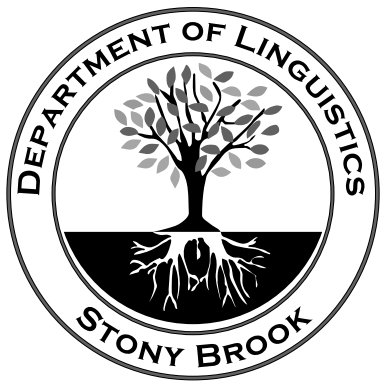
The Computational Nature of Language

Part 0: Overview

Thomas Graf



Stony Brook University
mail@thomasgraf.net
https://thomasgraf.net

ESSLLI 2021
July 26–30, 2021

Get the slides at:
https://esslli2021.thomasgraf.net

This course

- **Topic**
  - language as a cognitive ability
  - understanding this ability through computation
  - logic as a descriptive interface to computation
  - **subregular complexity**
- **Foundational**
  - no math required (I'm certain)
  - no linguistics required (I hope)
  - presupposes minimal logic background

$$\forall x \Big[a(x) \rightarrow \big(\exists y[b(y)] \vee \forall y[a(y)] \big) \Big]$$

- **Resources**
  - `https://esslli2021.thomasgraf.net`
  - slides (uploaded before each lecture)
  - many, many optional(!) readings

A puzzle

- German has a prefix **über**.
- This prefix can be freely combined with *morgen* ‘tomorrow’.

Example: German

morgen	tomorrow
über+*morgen*	the day after tomorrow
(**über**+)$^{\mathbf{n}}$*morgen*	(the day after)^{n} tomorrow

- Ilocano (Austronesian, Philippines) has a circumfix **ka- -an**.
- This prefix can be combined **only once** with *bigát* ‘tomorrow’.

Example: Ilocano

bigát	tomorrow
ka+*bigát*+**an**	the day after tomorrow
*(**ka**)$^{\mathbf{n}}$+*bigát*+(**an**)$^{\mathbf{n}}$	(the day after)^{n} tomorrow

A puzzle [cont.]

- The contrast between German and Ilocano is no accident.
- Across languages, circumfixes do not seem to be iterable.

The big question

- Why is prefix iteration possible, but not circumfix iteration?
- Why should languages be limited in this manner?

A puzzle [cont.]

- The contrast between German and Ilocano is no accident.
- Across languages, circumfixes do not seem to be iterable.

The big question

- Why is prefix iteration possible, but not circumfix iteration?
- Why should languages be limited in this manner?

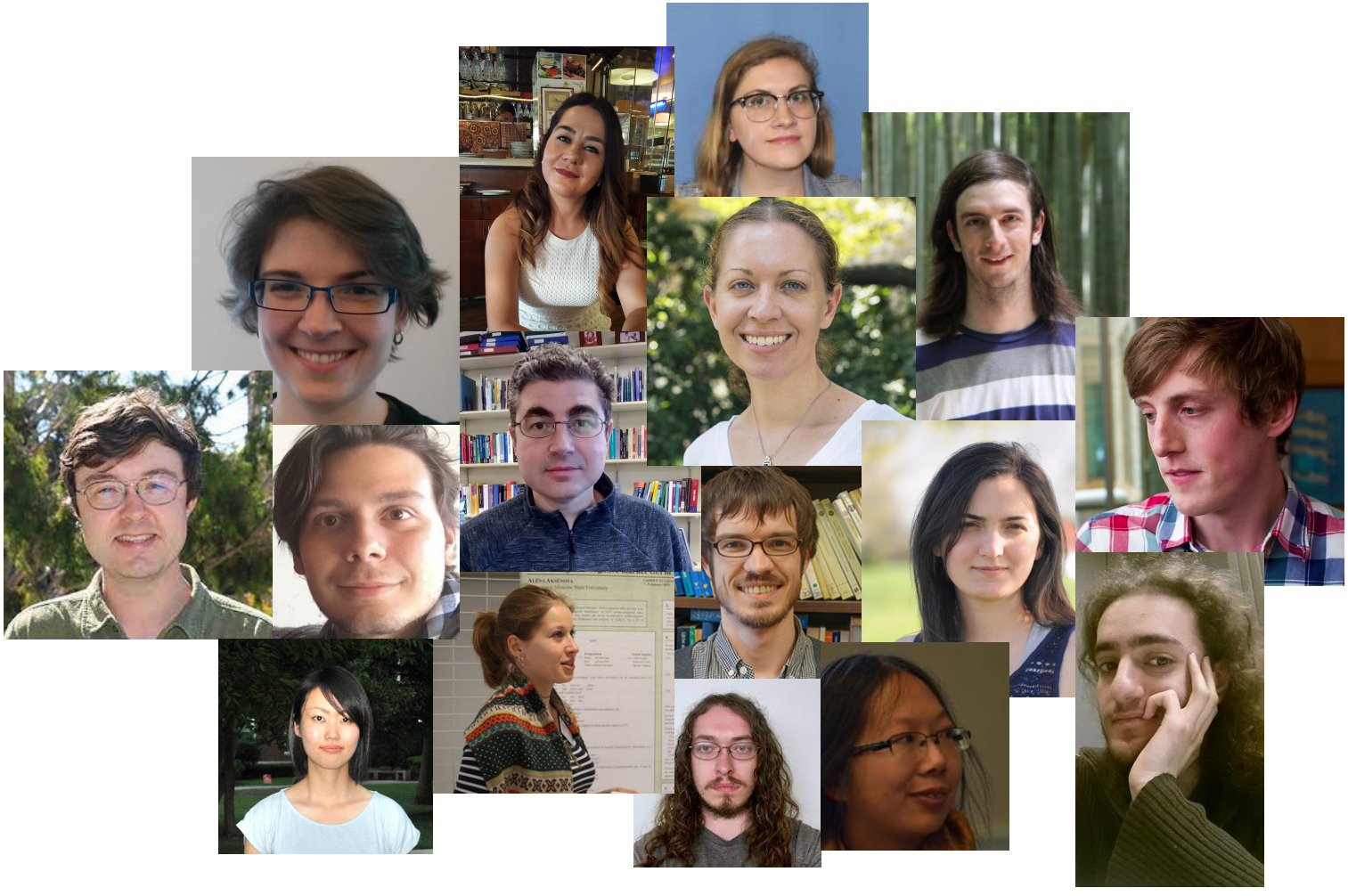
This course's answer

- Language involves computation, and it's not anything goes.
- The computational means of language are very limited.
- What would go beyond those limits simply does not happen.

Why should you care?

- **Linguistics**: a powerful new tool
  - derive typological gaps
  - "third factor" explanations (Chomsky 2005)
  - generalize across domains
    (phonology, morphology, syntax, semantics)
  - guide data gathering
  - inspire artificial language learning experiments
- **Math & Theoretical Computer Science**: low-hanging fruit
  - new applications for logic
  - conceptually simple classes
  - many formal issues for you to figure out
    closure properties, learnability, ...
- **NLP**: improved performance through domain knowledge
  - improve learning with more restricted hypothesis space
  - improve performance on resource-poor languages
  - test your models on patterns inspired by real-world phenomena

A Growing Community



Roadmap

Day 1 **Local dependencies in phonology** (grammar of sounds)

- Talking about structure with logic
- Strictly local string sets (SL)
- SL = conjunction of negative literals with successor
- voicing phenomena, vowel harmony

Day 2 **Unbounded dependencies in phonology**

- Tier-based strictly local string sets (TSL)
- Talking about mappings with logic
- Extensions of TSL
- sibilant harmony, Korean vowel harmony

Roadmap [cont.]

Day 3 **From phonology to syntax** (grammar of sentences)

- tree structures for sentences
- talking about trees with logic
- Minimalist grammars as a formal model
- string constraints on trees
  islands, wh-agreement, ...

Day 4 **TSL over trees**

- tree tiers for movement
- explaining the limits of movement
- phonology $\equiv$ syntax?

Day 5.1 **SL in semantics** (grammar of meaning)

- quantifiers as string languages
- why *blik* cannot mean *not all*

Day 5.2 **Wrap-up**

- open questions
- how **you** can join the enterprise

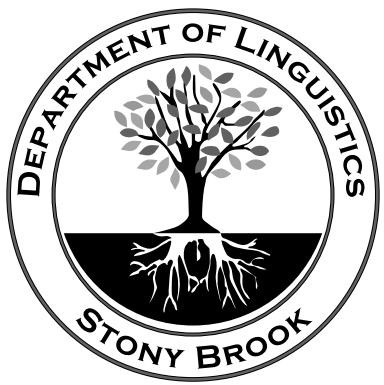
References

Chomsky, Noam. 2005. Three factors in language design. *Linguistic Inquiry* 36:1–22. URL `http://dx.doi.org/10.1162/0024389052993655`.

The Computational Nature of Language

Part 1: Strict Locality (SL)

Thomas Graf



Stony Brook University
mail@thomasgraf.net
https://thomasgraf.net

ESSLLI 2021
July 26–30, 2021

Get the slides at:
https://esslli2021.thomasgraf.net

Definition (Synchronous ISL transduction)

A node b in tree u can be **targeted** by an ISL-k context $\langle s, a, t\rangle$ iff there is some $p \in \mathbb{N}^*$ such that

node match $b = pa$, and

label match for all nodes g of s, $\ell_s(g) = \ell_u(pg)$,

full-width match for all nodes gi of s with $g \in \mathbb{N}^*$ and $i \in \mathbb{N}$, if pgj is a node of u $(j > i)$, then gj is a node of s.

Now suppose furthermore that n in u has $d \geq 0$ daughters. Given an ISL-k tree transducer τ, we use $\overleftarrow{\tau}(u, b)$ to denote the set of all trees $t[\Box_1 \leftarrow \overleftarrow{\tau}(u, b1), \ldots, \Box_d \leftarrow \overleftarrow{\tau}(u, bd)]$ such that there is a rewrite rule $\langle s, a, t\rangle$ in τ that targets node b in u. If this set is empty, $\overleftarrow{\tau}(u, b)$ is undefined. For any Σ-tree t, we may simply write $\overleftarrow{\tau}(t)$ instead of $\overleftarrow{\tau}(t, \varepsilon)$.

For any tree language L, the transduction computed by τ in *synchronous mode* is $\overleftarrow{\tau}(L) := \{\langle i, o\rangle \mid i \in L, o \in \overleftarrow{\tau}(i)\}$. A transduction is *synchronous input strictly k-local* (sISL-k) iff it can be computed by some ISL-k transducer in synchronous mode.

Definition (Synchronous ISL transduction)

A node b in tree u can be **targeted** by an ISL-k context $\langle s, a, t\rangle$ iff there is some $p \in \mathbb{N}^*$ such that

node match $b = pa$, and

label match fo

full-width matc and $i \in \mathbb{N}$, if

pg a node of s.

Now suppose fu ughters. Given an ISL-k tree tr te the set of all trees $t[\square_1 \leftarrow \overleftarrow{\tau}$ hat there is a rewrite rule $\langle s,$ f this set is empty, $\overleftarrow{\tau}(u, b)$ is undefined. For any Σ-tree t, we may simply write $\overleftarrow{\tau}(t)$ instead of $\overleftarrow{\tau}(t, \varepsilon)$.

Just kiddin'!

For any tree language L, the transduction computed by τ in *synchronous mode* is $\overleftarrow{\tau}(L) := \{\langle i, o\rangle \mid i \in L, o \in \overleftarrow{\tau}(i)\}$. A transduction is *synchronous input strictly k-local* (sISL-k) iff it can be computed by some ISL-k transducer in synchronous mode.

Outline

The big linguistic questions

- What are the laws that govern sounds, words, sentences, meaning?
- **Why** are those the laws?
- How **complex** are these laws? How hard are they to compute?
- How are they **learned**?
- Do we find **typological gaps**, i.e. patterns that should exist but don't appear in any language?
- What can we infer about human cognition?

Language & Computation

- Stand on the shoulders of giants.
- Computer scientists have figured out a lot about complexity, so let's apply their ideas to language.

The big linguistic questions

- What are the laws that govern sounds, words, sentences, meaning?
- **Why** are those the laws?
- How **complex** are these laws? How hard are they to compute?
- How are they **learned**?
- Do we find **typological gaps**, i.e. patterns that should exist but don't appear in any language?
- What can we infer about human cognition?

Language & Computation

- Stand on the shoulders of giants.
- Computer scientists have figured out a lot about complexity, so let's apply their ideas to language.

The quest for tight upper bounds

- Computer scientists have defined many notions of complexity.
- But the common classes don't provide a good fit for language.

regular $<$ context-free $<$ mildly context-sensitive $<$ $\cdots$

Kaplan and Kay (1994)

Phonology

Phonology laws governing the arrangement of sounds in a word

What's the problem?

- Regular is too loose an upper bound.
- It includes many patterns that never occur in language.
- Regular properly subsumes **first-order logic**, and that's already way too much!

Defining structures with first-order logic

- Logic isn't just for reasoning and meaning, it can also be used to talk about structure.
- The more powerful the logic, the more complex the conditions it can impose on those structures.

First-order logic (FO) for strings

- $\forall x[a(x)]$
  Every node must be labeled a.
- $\exists x \exists y[x \lhd y]$
  There are nodes x and y s.t. x immediately precedes ($\lhd$) y.
- $\exists x \forall y[\neg(x = y) \rightarrow x \prec y]$
  There is a node x that precedes ($\prec$) every y distinct from x.

A concrete example: Word-final devoicing

- Consonants can be **voiced** or **voiceless**.

voiced	**[z]**	**[d]**	**[g]**
voiceless	**[s]**	**[t]**	**[k]**

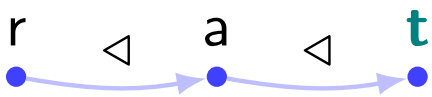
- Many languages (German, Russian, . . .)
  forbid some voiced sounds at the end of the word.
- **Caution:** this is a fact about pronunciation, not spelling!
- **Example:** German *Rad* ‘wheel’ is pronounced [ʁaːt], not [ʁaːd]

Remark for linguists

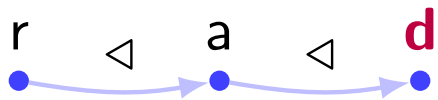
- Word-final devoicing is usually analyzed as a mapping from an underlying representation to a surface form.
- We’ll get to that, but for now we’ll look at it as a constraint on surface forms.

Word-final devoicing in FO

Allowed



Forbidden



FO formulas for word-final devoicing

1. $\mathrm{final}(\mathbf{x}) \Leftrightarrow \neg\exists y[\mathbf{x} \lhd y]$
   Node **x** is final iff there is no y immediately to the right of **x**.
2. $\forall(\mathbf{x})[\mathrm{final}(\mathbf{x}) \rightarrow \neg\mathbf{d}(\mathbf{x}) \wedge \neg\mathbf{z}(\mathbf{x})]$
   If node **x** is final, then it must not be labeled **d** or **z**.

- The left structure is a **model** of the FO constraint, the right one is not.
- Hence the left string is well-formed, the right one is ill-formed.

Also FO, yet unattested: conditional word-final devoicing

- Suppose we modify our FO constraint as follows:

$$\forall(\mathbf{x})\Big[\big(\text{final}(\mathbf{x}) \wedge \neg\exists\mathbf{y}[\neg\text{final}(\mathbf{y}) \wedge (\mathbf{d}(\mathbf{y}) \vee \mathbf{z}(\mathbf{y}))]\big) \rightarrow (\neg\mathbf{d}(\mathbf{x}) \wedge \neg\mathbf{z}(\mathbf{x}))\Big]$$

- In plain English:
  If node **x** is final, then it must not be labeled **d** or **z**,
  unless there is some non-final node **y** that is labeled **d** or **z**.
- **Result**: word-final devoicing would apply to *Ra***d** and *Wal***d**,
  but not to **D***unkelsteinerwal***d**.
- No known language has such conditional word-final devoicing!

The Problem

- A good scientific theory must explain why languages have word-final devoicing but not conditional word-final devoicing.
- Even if you only care about engineering,
  you don't want to spend resources on unnecessary crud.

- We cannot simply say that phonology is regular or FO.
  That is **too loose an upper bound**.
- We need more restrictive models within Regular
  $\Rightarrow$ **subregular**

Restricting FO

- From the perspective of logic, the search for weak subregular classes is the search for weak fragments of FO.
- But where should we look?
- The problem with conditional word-final devoicing is that it is, well, conditional.
- We need a fragment that limits conditionals.

Literals

- An n-**literal** is an FO formula of the form

$$\exists x_1, x_2, \ldots, x_n [x_1 \triangleleft x_2 \wedge \cdots \wedge x_{n-1} \triangleleft x_n \wedge a(x_1) \wedge b(x_2) \wedge \cdots \wedge z(x_n)]$$

- **Intuition:** n-literals $\equiv$ n-grams

Example

1. The bigram ab corresponds to the 2-literal

$$\exists x_1, x_2 [x_1 \triangleleft x_2 \wedge a(x_1) \wedge b(x_2)]$$

2. The 4-gram $abac$ corresponds to the 4-literal

$$\exists x_1, x_2, x_3, x_4 [x_1 \triangleleft x_2 \wedge x_2 \triangleleft x_3 \wedge x_3 \triangleleft x_4 \wedge a(x_1) \wedge b(x_2) \wedge a(x_3) \wedge c(x_4)]$$

Conjunction of negative literals (CNL)

CNL A formula is a **conjunction of negative n-literals** (CNL-n) iff it is of the form

$$\neg\phi_1 \wedge \neg\phi_2 \wedge \cdots \wedge \neg\phi_n$$

where each ϕ_i is an n-literal.

- If a pattern can be described as a CNL-n formula, it is **strictly n-local (SL-n)**.

Word-final devoicing is SL-2

- Suppose we use $\ltimes$ to mark the right edge of a string:

$$\neg\exists x_1, x_2[x_1 \lhd x_2 \wedge \mathbf{d}(x_1) \wedge \ltimes(x_2)] \wedge \neg\exists x_1, x_2[x_1 \lhd x_2 \wedge \mathbf{z}(x_1) \wedge \ltimes(x_2)]$$

- Or more succinctly:

$$\neg\mathbf{d}\ltimes \wedge \neg\mathbf{z}\ltimes$$

A model checker for CNL

It is easy to verify if a string is a model of a CNL-n formula.

1. Move a sliding window of size n from left to right.
2. If we find a forbidden literal, the string is illicit.
3. If we make it to the end, the string is fine.

Example: Model checker for word-final devoicing

$$* \rtimes \; r \; a \; \mathbf{d} \ltimes \qquad \begin{array}{l} \neg \mathbf{d} \ltimes \wedge \\ \neg \mathbf{z} \ltimes \end{array}$$

A model checker for CNL

It is easy to verify if a string is a model of a CNL-n formula.

1. Move a sliding window of size n from left to right.
2. If we find a forbidden literal, the string is illicit.
3. If we make it to the end, the string is fine.

Example: Model checker for word-final devoicing

$* \rtimes$ r a **d** $\ltimes$

$\neg$**d** $\ltimes \wedge$

$\neg$**z** $\ltimes$

A model checker for CNL

It is easy to verify if a string is a model of a CNL-n formula.

1. Move a sliding window of size n from left to right.
2. If we find a forbidden literal, the string is illicit.
3. If we make it to the end, the string is fine.

Example: Model checker for word-final devoicing

$* \rtimes$ r a **d** $\ltimes$

$\neg$**d** $\ltimes \wedge$

$\neg$**z** $\ltimes$

A model checker for CNL

It is easy to verify if a string is a model of a CNL-n formula.

1. Move a sliding window of size n from left to right.
2. If we find a forbidden literal, the string is illicit.
3. If we make it to the end, the string is fine.

Example: Model checker for word-final devoicing

$$* \rtimes \boxed{\text{r a}}\ \mathbf{d} \ltimes$$

$$\neg\mathbf{d} \ltimes \wedge$$

$$\neg\mathbf{z} \ltimes$$

A model checker for CNL

It is easy to verify if a string is a model of a CNL-n formula.

1. Move a sliding window of size n from left to right.
2. If we find a forbidden literal, the string is illicit.
3. If we make it to the end, the string is fine.

Example: Model checker for word-final devoicing

$$* \rtimes \; r \; a \; \boxed{d} \ltimes$$

$$\neg d \ltimes \wedge$$

$$\neg z \ltimes$$

A model checker for CNL

It is easy to verify if a string is a model of a CNL-n formula.

1. Move a sliding window of size n from left to right.
2. If we find a forbidden literal, the string is illicit.
3. If we make it to the end, the string is fine.

Example: Model checker for word-final devoicing

$$* \rtimes \; r \; a \; \boxed{\mathbf{d} \ltimes}$$

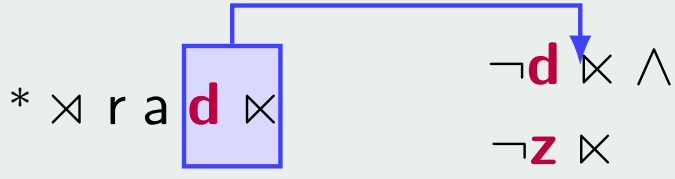
$$\neg \mathbf{d} \ltimes \wedge$$
$$\neg \mathbf{z} \ltimes$$

A model checker for CNL

It is easy to verify if a string is a model of a CNL-n formula.

1. Move a sliding window of size n from left to right.
2. If we find a forbidden literal, the string is illicit.
3. If we make it to the end, the string is fine.

Example: Model checker for word-final devoicing



A model checker for CNL

It is easy to verify if a string is a model of a CNL-n formula.

1. Move a sliding window of size n from left to right.
2. If we find a forbidden literal, the string is illicit.
3. If we make it to the end, the string is fine.

Example: Model checker for word-final devoicing

$$* \rtimes \; r \; a \; \mathbf{d} \; \ltimes \qquad \begin{matrix} \neg \mathbf{d} \ltimes \wedge \\ \neg \mathbf{z} \ltimes \end{matrix} \qquad \rtimes \; r \; a \; \mathbf{t} \; \ltimes$$

A model checker for CNL

It is easy to verify if a string is a model of a CNL-n formula.

1. Move a sliding window of size n from left to right.
2. If we find a forbidden literal, the string is illicit.
3. If we make it to the end, the string is fine.

Example: Model checker for word-final devoicing

$* \rtimes$ r a **d** $\ltimes$ $\quad$ $\neg$**d** $\ltimes \wedge$ $\neg$**z** $\ltimes$ $\quad$ $\rtimes$ r a **t** $\ltimes$

A model checker for CNL

It is easy to verify if a string is a model of a CNL-n formula.

1. Move a sliding window of size n from left to right.
2. If we find a forbidden literal, the string is illicit.
3. If we make it to the end, the string is fine.

Example: Model checker for word-final devoicing

$* \rtimes$ r a **d** $\ltimes$ $\quad$ $\neg$**d** $\ltimes \wedge$ $\neg$**z** $\ltimes$ $\quad$ $\rtimes$ r a **t** $\ltimes$

A model checker for CNL

It is easy to verify if a string is a model of a CNL-n formula.

1. Move a sliding window of size n from left to right.
2. If we find a forbidden literal, the string is illicit.
3. If we make it to the end, the string is fine.

Example: Model checker for word-final devoicing

$* \rtimes$ r a **d** $\ltimes$ $\quad$ $\neg$**d** $\ltimes \wedge$ $\neg$**z** $\ltimes$ $\quad$ $\rtimes$ r a **t** $\ltimes$

A model checker for CNL

It is easy to verify if a string is a model of a CNL-n formula.

1. Move a sliding window of size n from left to right.
2. If we find a forbidden literal, the string is illicit.
3. If we make it to the end, the string is fine.

Example: Model checker for word-final devoicing

$* \rtimes$ r a **d** $\ltimes$

$\neg$**d** $\ltimes \wedge$
$\neg$**z** $\ltimes$

$\rtimes$ r a **t** $\ltimes$

Another example: Intervocalic voicing is SL-3

- Captured by forbidding voiceless consonants between vowels
- **Description:** don't have $\mathrm{V}[-\mathrm{voice}]\mathrm{V}$
- **Suppose**:
  - $[-\mathrm{voice}] = \{\mathsf{s},\mathsf{ʃ}\}$
  - $\mathrm{V} = \{\mathsf{a},\mathsf{i},\mathsf{o},\mathsf{u}\}$
- **Compiled out**: $\neg$**asa** $\wedge$ $\neg$**aʃa** $\wedge$ $\neg$**asi** $\wedge$ $\neg$**aʃi** $\wedge \ldots$

Example: Northern Italian

$*\rtimes$ a s o l a $\ltimes$

Another example: Intervocalic voicing is SL-3

- Captured by forbidding voiceless consonants between vowels
- **Description:** don't have $\mathrm{V}[-\mathrm{voice}]\mathrm{V}$
- **Suppose**:
  - $[-\mathrm{voice}] = \{\mathsf{s},\mathsf{ʃ}\}$
  - $\mathrm{V} = \{\mathsf{a},\mathsf{i},\mathsf{o},\mathsf{u}\}$
- **Compiled out**: $\neg$**asa** $\wedge$ $\neg$**aʃa** $\wedge$ $\neg$**asi** $\wedge$ $\neg$**aʃi** $\wedge \ldots$

Example: Northern Italian

$* \rtimes$ **a s o** | **a** $\ltimes$

Another example: Intervocalic voicing is SL-3

- Captured by forbidding voiceless consonants between vowels
- **Description:** don't have $\mathrm{V}[-\mathrm{voice}]\mathrm{V}$
- **Suppose**:
  - $[-\mathrm{voice}] = \{\mathsf{s},\mathsf{ʃ}\}$
  - $\mathrm{V} = \{\mathsf{a},\mathsf{i},\mathsf{o},\mathsf{u}\}$
- **Compiled out**: $\neg\mathbf{asa} \wedge \neg\mathbf{aʃa} \wedge \neg\mathbf{asi} \wedge \neg\mathbf{aʃi} \wedge \ldots$

Example: Northern Italian

* ⋊ **a** **s** **o** | **a** ⋉

Another example: Intervocalic voicing is SL-3

- Captured by forbidding voiceless consonants between vowels
- **Description:** don't have $\mathrm{V}[-\mathrm{voice}]\mathrm{V}$
- **Suppose**:
  - $[-\mathrm{voice}] = \{\mathsf{s},\mathsf{ʃ}\}$
  - $\mathrm{V} = \{\mathsf{a},\mathsf{i},\mathsf{o},\mathsf{u}\}$
- **Compiled out**: ¬**asa** ∧ ¬**aʃa** ∧ ¬**asi** ∧ ¬**aʃi** ∧ . . .

Example: Northern Italian

* ⋊ **a s o** | **a** ⋉

Another example: Intervocalic voicing is SL-3

- Captured by forbidding voiceless consonants between vowels
- **Description:** don't have $\mathrm{V}[-\mathrm{voice}]\mathrm{V}$
- **Suppose**:
  - $[-\mathrm{voice}] = \{\mathsf{s},\mathsf{ʃ}\}$
  - $\mathrm{V} = \{\mathsf{a},\mathsf{i},\mathsf{o},\mathsf{u}\}$
- **Compiled out**: $\neg$**asa** $\wedge$ $\neg$**aʃa** $\wedge$ $\neg$**asi** $\wedge$ $\neg$**aʃi** $\wedge \ldots$

Example: Northern Italian

* ⋊ **a** **s** **o** l **a** ⋉ ⋊ **a** z **o** l **a** ⋉

Another example: Intervocalic voicing is SL-3

- Captured by forbidding voiceless consonants between vowels
- **Description:** don't have $\mathrm{V}[-\mathrm{voice}]\mathrm{V}$
- **Suppose**:
  - $[-\mathrm{voice}] = \{\mathsf{s},\mathsf{ʃ}\}$
  - $\mathrm{V} = \{\mathsf{a},\mathsf{i},\mathsf{o},\mathsf{u}\}$
- **Compiled out**: $\neg\mathbf{asa} \wedge \neg\mathbf{aʃa} \wedge \neg\mathbf{asi} \wedge \neg\mathbf{aʃi} \wedge \ldots$

Example: Northern Italian

* ⋊ **a s o** l **a** ⋉ ⋊ **a** z **o** l **a** ⋉

Another example: Intervocalic voicing is SL-3

- Captured by forbidding voiceless consonants between vowels
- **Description:** don't have $\mathrm{V}[-\mathrm{voice}]\mathrm{V}$
- **Suppose**:
  - $[-\mathrm{voice}] = \{\mathsf{s},\mathsf{ʃ}\}$
  - $\mathrm{V} = \{\mathsf{a},\mathsf{i},\mathsf{o},\mathsf{u}\}$
- **Compiled out**: ¬**asa** ∧ ¬**aʃa** ∧ ¬**asi** ∧ ¬**aʃi** ∧ . . .

Example: Northern Italian

* ⋊ **a** **s** **o** l **a** ⋉ ⋊ **a** z **o** l **a** ⋉

Another example: Intervocalic voicing is SL-3

- Captured by forbidding voiceless consonants between vowels
- **Description:** don't have $\mathrm{V}[-\mathrm{voice}]\mathrm{V}$
- **Suppose**:
  - $[-\mathrm{voice}] = \{\mathsf{s},\mathsf{ʃ}\}$
  - $\mathrm{V} = \{\mathsf{a},\mathsf{i},\mathsf{o},\mathsf{u}\}$
- **Compiled out**: $\neg\mathbf{asa} \wedge \neg\mathbf{aʃa} \wedge \neg\mathbf{asi} \wedge \neg\mathbf{aʃi} \wedge \ldots$

Example: Northern Italian

* ⋊ **a s o** l **a** ⋉ ⋊ **a** z **o** l **a** ⋉

Another example: Intervocalic voicing is SL-3

- Captured by forbidding voiceless consonants between vowels
- **Description:** don't have $\mathrm{V}[-\mathrm{voice}]\mathrm{V}$
- **Suppose**:
  - $[-\mathrm{voice}] = \{\mathsf{s},\mathsf{ʃ}\}$
  - $\mathrm{V} = \{\mathsf{a},\mathsf{i},\mathsf{o},\mathsf{u}\}$
- **Compiled out**: ¬**asa** ∧ ¬**aʃa** ∧ ¬**asi** ∧ ¬**aʃi** ∧ . . .

Example: Northern Italian

* ⋊ **a** **s** **o** l **a** ⋉ ⋊ **a** z **o** l **a** ⋉

Another example: Intervocalic voicing is SL-3

- Captured by forbidding voiceless consonants between vowels
- **Description:** don't have $\mathrm{V}[-\mathrm{voice}]\mathrm{V}$
- **Suppose**:
  - $[-\mathrm{voice}] = \{\mathsf{s},\mathsf{ʃ}\}$
  - $\mathrm{V} = \{\mathsf{a},\mathsf{i},\mathsf{o},\mathsf{u}\}$
- **Compiled out**: $\neg\mathbf{asa} \wedge \neg\mathbf{aʃa} \wedge \neg\mathbf{asi} \wedge \neg\mathbf{aʃi} \wedge \ldots$

Example: Northern Italian

* ⋊ **a s o** l **a** ⋉ ⋊ **a** z **o** l **a** ⋉

Another example: Intervocalic voicing is SL-3

- Captured by forbidding voiceless consonants between vowels
- **Description:** don't have $\mathrm{V}[-\mathrm{voice}]\mathrm{V}$
- **Suppose**:
  - $[-\mathrm{voice}] = \{\mathsf{s},\mathsf{ʃ}\}$
  - $\mathrm{V} = \{\mathsf{a},\mathsf{i},\mathsf{o},\mathsf{u}\}$
- **Compiled out**: $\neg\mathbf{asa} \wedge \neg\mathbf{a\mathsf{ʃ}a} \wedge \neg\mathbf{asi} \wedge \neg\mathbf{a\mathsf{ʃ}i} \wedge \ldots$

Example: Northern Italian

* ⋊ **a s o** l **a** ⋉ ⋊ **a** z **o** l **a** ⋉ ⋊ **a** + **s o** c **i a** l **e** ⋉

Another example: Intervocalic voicing is SL-3

- Captured by forbidding voiceless consonants between vowels
- **Description:** don't have $\mathrm{V}[-\mathrm{voice}]\mathrm{V}$
- **Suppose**:
  - $[-\mathrm{voice}] = \{\mathsf{s},\mathsf{ʃ}\}$
  - $\mathrm{V} = \{\mathsf{a},\mathsf{i},\mathsf{o},\mathsf{u}\}$
- **Compiled out**: $\neg\mathbf{asa} \wedge \neg\mathbf{a\mathsf{ʃ}a} \wedge \neg\mathbf{asi} \wedge \neg\mathbf{a\mathsf{ʃ}i} \wedge \ldots$

Example: Northern Italian

* ⋊ **a s o** l **a** ⋉ ⋊ **a** z **o** l **a** ⋉ ⋊ **a** + **s o** c **i a** l **e** ⋉

Another example: Intervocalic voicing is SL-3

- Captured by forbidding voiceless consonants between vowels
- **Description:** don't have $\mathrm{V}[-\mathrm{voice}]\mathrm{V}$
- **Suppose**:
  - $[-\mathrm{voice}] = \{\mathsf{s},\mathsf{ʃ}\}$
  - $\mathrm{V} = \{\mathsf{a},\mathsf{i},\mathsf{o},\mathsf{u}\}$
- **Compiled out**: $\neg\mathbf{asa} \wedge \neg\mathbf{aʃa} \wedge \neg\mathbf{asi} \wedge \neg\mathbf{aʃi} \wedge \ldots$

Example: Northern Italian

$*\rtimes$ **a s o** l **a** $\ltimes$ $\quad\rtimes$ **a** z **o** l **a** $\ltimes$ $\quad\rtimes$ **a** + **s o** c **i a** l **e** $\ltimes$

Another example: Intervocalic voicing is SL-3

- Captured by forbidding voiceless consonants between vowels
- **Description:** don't have $\mathrm{V}[-\mathrm{voice}]\mathrm{V}$
- **Suppose**:
  - $[-\mathrm{voice}] = \{\mathsf{s},\mathsf{ʃ}\}$
  - $\mathrm{V} = \{\mathsf{a},\mathsf{i},\mathsf{o},\mathsf{u}\}$
- **Compiled out**: $\neg\mathbf{asa} \wedge \neg\mathbf{a\mathsf{ʃ}a} \wedge \neg\mathbf{asi} \wedge \neg\mathbf{a\mathsf{ʃ}i} \wedge \ldots$

Example: Northern Italian

$*\rtimes$ **a s o** l **a** $\ltimes$ $\quad$ $\rtimes$ **a** z **o** l **a** $\ltimes$ $\quad$ $\rtimes$ **a** + **s o** c **i a** l **e** $\ltimes$

Another example: Intervocalic voicing is SL-3

- Captured by forbidding voiceless consonants between vowels
- **Description:** don't have $\mathrm{V}[-\mathrm{voice}]\mathrm{V}$
- **Suppose**:
  - $[-\mathrm{voice}] = \{\mathsf{s},\mathsf{ʃ}\}$
  - $\mathrm{V} = \{\mathsf{a},\mathsf{i},\mathsf{o},\mathsf{u}\}$
- **Compiled out**: $\neg\mathbf{asa} \wedge \neg\mathbf{a\mathsf{ʃ}a} \wedge \neg\mathbf{asi} \wedge \neg\mathbf{a\mathsf{ʃ}i} \wedge \ldots$

Example: Northern Italian

* ⋊ **a s o** l **a** ⋉ ⋊ **a** z **o** l **a** ⋉ ⋊ **a** + **s o** c **i a** l **e** ⋉

Another example: Intervocalic voicing is SL-3

- Captured by forbidding voiceless consonants between vowels
- **Description:** don't have $\mathrm{V}[-\mathrm{voice}]\mathrm{V}$
- **Suppose**:
  - $[-\mathrm{voice}] = \{\mathsf{s},\mathsf{ʃ}\}$
  - $\mathrm{V} = \{\mathsf{a},\mathsf{i},\mathsf{o},\mathsf{u}\}$
- **Compiled out**: ¬**asa** ∧ ¬**aʃa** ∧ ¬**asi** ∧ ¬**aʃi** ∧ . . .

Example: Northern Italian

* ⋊ **a** **s** **o** l **a** ⋉ ⋊ **a** z **o** l **a** ⋉ ⋊ **a** + **s** **o** c **i** **a** l **e** ⋉

Another example: Intervocalic voicing is SL-3

- Captured by forbidding voiceless consonants between vowels
- **Description:** don't have $\mathrm{V}[-\mathrm{voice}]\mathrm{V}$
- **Suppose**:
  - $[-\mathrm{voice}] = \{\mathsf{s},\mathsf{ʃ}\}$
  - $\mathrm{V} = \{\mathsf{a},\mathsf{i},\mathsf{o},\mathsf{u}\}$
- **Compiled out**: $\neg\mathbf{asa} \wedge \neg\mathbf{a\mathsf{ʃ}a} \wedge \neg\mathbf{asi} \wedge \neg\mathbf{a\mathsf{ʃ}i} \wedge \ldots$

Example: Northern Italian

$*\rtimes$ **a s o** l **a** $\ltimes$ $\quad\rtimes$ **a** z **o** l **a** $\ltimes$ $\quad\rtimes$ **a** + **s o** c **i a** l **e** $\ltimes$

Another example: Intervocalic voicing is SL-3

- Captured by forbidding voiceless consonants between vowels
- **Description:** don't have $\mathrm{V}[-\mathrm{voice}]\mathrm{V}$
- **Suppose**:
  - $[-\mathrm{voice}] = \{\mathsf{s},\mathsf{ʃ}\}$
  - $\mathrm{V} = \{\mathsf{a},\mathsf{i},\mathsf{o},\mathsf{u}\}$
- **Compiled out**: $\neg\mathbf{asa} \wedge \neg\mathbf{a\mathsf{ʃ}a} \wedge \neg\mathbf{asi} \wedge \neg\mathbf{a\mathsf{ʃ}i} \wedge \ldots$

Example: Northern Italian

$*\rtimes$ **a s o** l **a** $\ltimes$ $\quad\rtimes$ **a** z **o** l **a** $\ltimes$ $\quad\rtimes$ **a** + **s o** c **i a** l **e** $\ltimes$

Another example: Intervocalic voicing is SL-3

- Captured by forbidding voiceless consonants between vowels
- **Description:** don't have $\mathrm{V}[-\mathrm{voice}]\mathrm{V}$
- **Suppose**:
  - $[-\mathrm{voice}] = \{\mathsf{s},\mathsf{ʃ}\}$
  - $\mathrm{V} = \{\mathsf{a},\mathsf{i},\mathsf{o},\mathsf{u}\}$
- **Compiled out**: ¬**asa** ∧ ¬**aʃa** ∧ ¬**asi** ∧ ¬**aʃi** ∧ . . .

Example: Northern Italian

* ⋊ **a** **s** **o** l **a** ⋉ ⋊ **a** z **o** l **a** ⋉ ⋊ **a** + **s** **o** c **i** **a** l **e** ⋉

Conditional word-final devoicing is not SL

Conditional word-final devoicing is not SL-n for any $n \geq 1$.

The mathematical argument

1. Suppose ill-formed: $bababa \cdots bababa \cdots bababa\mathbf{d}$
2. Suppose well-formed: $bababa \cdots ba\mathbf{d}aba \cdots bababa\mathbf{d}$
3. Every literal of the ill-formed word also occurs in the well-formed word.
4. If we forbid any of those literals, we incorrectly forbid the well-formed string.
5. If we permit all of those literals, we incorrectly permit the ill-formed string.
6. Hence no CNL formula can forbid the former while permitting the latter.
7. Hence the pattern is not SL-n for any $n \geq 1$.

Uh, what?

- The mathematical argument seems irrelevant:
  no language has arbitrarily long words like that.

1. **Linguistic answer**: competence VS performance
2. **Computational answer**: infinity is more insightful
   (Savitch 1993)
3. **My own take**: this doesn't really matter here
   - We want to explain why word-final devoicing is so common, while conditional word-final devoicing is unattested.
   - Even if conditional word-final devoicing in German, Russian, ..., is SL-n for some very large n, that large n would distinguish it from word-final devoicing, which is SL-2.
   - And the higher the n, the harder the pattern is to learn.

SL patterns are easy to learn (for small n)

- Suppose you have to learn an SL-n pattern from **positive input only**. How could you do that?

1. Start out with the formula $\neg\phi_1 \wedge \neg\phi_2 \wedge \cdots \wedge \neg\phi_m$ that forbids all n-literals (basically: everything is forbidden).
2. When you see an n-literal in the input, remove it from the formula.

A toy example

- Only two symbols: a and b
- We have to learn an SL-2 pattern from positive input.

Input **Current formula**

$$\neg \rtimes a \wedge \neg \rtimes b \wedge \neg aa \wedge \neg ab \wedge \neg ba \wedge \neg bb \wedge \neg a \ltimes \wedge \neg b \ltimes$$

Exercise

What problem would this algorithm run into with, say, SL-6 or SL-13 patterns?

A toy example

- Only two symbols: a and b
- We have to learn an SL-2 pattern from positive input.

Input	**Current formula**
	$\neg \rtimes a \wedge \neg \rtimes b \wedge \neg aa \wedge \neg ab \wedge \neg ba \wedge \neg bb \wedge \neg a \ltimes \wedge \neg b \ltimes$
aa	$\neg \rtimes b \wedge \neg ab \wedge \neg ba \wedge \neg bb \wedge \neg b \ltimes$

Exercise

What problem would this algorithm run into with, say, SL-6 or SL-13 patterns?

A toy example

- Only two symbols: a and b
- We have to learn an SL-2 pattern from positive input.

Input	**Current formula**
	$\neg \rtimes a \wedge \neg \rtimes b \wedge \neg aa \wedge \neg ab \wedge \neg ba \wedge \neg bb \wedge \neg a \ltimes \wedge \neg b \ltimes$
aa	$\neg \rtimes b \wedge \neg ab \wedge \neg ba \wedge \neg bb \wedge \neg b \ltimes$
aaaaa	$\neg \rtimes b \wedge \neg ab \wedge \neg ba \wedge \neg bb \wedge \neg b \ltimes$

Exercise

What problem would this algorithm run into with, say, SL-6 or SL-13 patterns?

A toy example

- Only two symbols: a and b
- We have to learn an SL-2 pattern from positive input.

Input	**Current formula**
	$\neg \rtimes a \wedge \neg \rtimes b \wedge \neg aa \wedge \neg ab \wedge \neg ba \wedge \neg bb \wedge \neg a \ltimes \wedge \neg b \ltimes$
aa	$\neg \rtimes b \wedge \neg ab \wedge \neg ba \wedge \neg bb \wedge \neg b \ltimes$
aaaaa	$\neg \rtimes b \wedge \neg ab \wedge \neg ba \wedge \neg bb \wedge \neg b \ltimes$
baa	$\neg ab \wedge \neg bb \wedge \neg b \ltimes$

Exercise

What problem would this algorithm run into with, say, SL-6 or SL-13 patterns?

Do we have a winner?

SL has **many advantages**:

- Heavily restricted fragment of FO, yet covers a lot of data
  (I'd wager at least 90% of phonological phenomena are SL-2, SL-3, SL-4, or SL-5)
- Rules out many unattested phenomena, like conditional word-final devoicing
- Has a simple and efficient learning algorithm
  (for small n, which is what we find in the wild)
- Minimal resource needs
  (model checker only needs to store one formula consider at most n symbols at a time)

Do we have a winner?

SL has **many advantages**:

- Heavily restricted fragment of FO, yet covers a lot of data
  (I'd wager at least 90% of phonological phenomena are SL-2, SL-3, SL-4, or SL-5)
- Rules out many unattested phenomena, like conditional word-final devoicing
- Has a simple and efficient learning algorithm
  (for small n, which is what we find in the wild)
- Minimal resource needs
  (model checker only needs to store one formula consider at most n symbols at a time)

Unfortunately, SL is **too weak**...

SL is not enough

Culminativity (Heinz 2014)

Every phonological word has **exactly one** syllable that carries primary stress ($\acute{\sigma}$).

Culminativity is not SL-n for any n.

1. Suppose ill-formed: $\sigma\sigma\sigma\cdots\sigma\sigma\sigma\cdots\sigma\sigma\sigma$
2. Suppose well-formed: $\sigma\sigma\sigma\cdots\sigma\acute{\sigma}\sigma\cdots\sigma\sigma\sigma$
3. Every literal of the ill-formed word also occurs in the well-formed word.
4. If we forbid any of those literals, we incorrectly forbid the well-formed string.
5. If we permit all of those literals, we incorrectly permit the ill-formed string.
6. Hence no CNL formula can forbid the former while permitting the latter.
7. Hence the pattern is not SL-n for any $n \geq 1$.

Another Problem: Samala Sibilant Harmony

- If multiple sibilants occur in the same word, they must all be [+anterior] (**s**,**z**) or [−anterior] (**ʃ**,**ʒ**).
- In other words: Don't mix **purple** and **teal**.

*sʃ	*sʒ	*zʃ	*zʒ
*ʃs	*ʒs	*ʃz	*ʒz

- **But:** Sibilants can be arbitrarily far away from each other!

Example: Samala (Applegate 1972)

* ⋊ h a s x i n t i l a w a ʃ ⋉

⋊ h a ʃ x i n t i l a w a ʃ ⋉

Another Problem: Samala Sibilant Harmony

- If multiple sibilants occur in the same word, they must all be [+anterior] (**s**,**z**) or [−anterior] (**ʃ**,**ʒ**).
- In other words: Don't mix **purple** and **teal**.

*sʃ *sʒ *zʃ *zʒ
*ʃs *ʒs *ʃz *ʒz

- **But:** Sibilants can be arbitrarily far away from each other!

Example: Samala (Applegate 1972)

* ⋊ h a **s** x i n t i l a w a **ʃ** ⋉

⋊ h a **ʃ** x i n t i l a w a **ʃ** ⋉

Another Problem: Samala Sibilant Harmony

- If multiple sibilants occur in the same word, they must all be [+anterior] (s,z) or [−anterior] (ʃ,ʒ).
- In other words: Don't mix **purple** and **teal**.

*sʃ *sʒ *zʃ *zʒ
*ʃs *ʒs *ʃz *ʒz

- **But:** Sibilants can be arbitrarily far away from each other!

Example: Samala (Applegate 1972)

* ⋊ h a s x i n t i l a w a ʃ ⋉

⋊ h a ʃ x i n t i l a w a ʃ ⋉

Another Problem: Samala Sibilant Harmony

- If multiple sibilants occur in the same word, they must all be [+anterior] (**s**,**z**) or [−anterior] (**ʃ**,**ʒ**).
- In other words: Don't mix **purple** and **teal**.

*sʃ	*sʒ	*zʃ	*zʒ
*ʃs	*ʒs	*ʃz	*ʒz

- **But:** Sibilants can be arbitrarily far away from each other!

Example: Samala (Applegate 1972)

* ⋊ h a **s** x i n t i l a w a **ʃ** ⋉

⋊ h a **ʃ** x i n t i l a w a **ʃ** ⋉

Another Problem: Samala Sibilant Harmony

- If multiple sibilants occur in the same word, they must all be [+anterior] (**s**,**z**) or [−anterior] (**ʃ**,**ʒ**).
- In other words: Don't mix **purple** and **teal**.

*sʃ *sʒ *zʃ *zʒ
*ʃs *ʒs *ʃz *ʒz

- **But:** Sibilants can be arbitrarily far away from each other!

Example: Samala (Applegate 1972)

* ⋊ h a **s** x i n t i l a w a **ʃ** ⋉

⋊ h a **ʃ** x i n t i l a w a **ʃ** ⋉

* ⋊ **s** t a j a n o w o n w a **ʃ** ⋉

Another Problem: Samala Sibilant Harmony

- If multiple sibilants occur in the same word, they must all be [+anterior] (**s**,**z**) or [−anterior] (**ʃ**,**ʒ**).
- In other words: Don't mix **purple** and **teal**.

*sʃ	*sʒ	*zʃ	*zʒ
*ʃs	*ʒs	*ʃz	*ʒz

- **But:** Sibilants can be arbitrarily far away from each other!

Example: Samala (Applegate 1972)

*⋊ h a **s** x i n t i l a w a **ʃ** ⋉

⋊ h a **ʃ** x i n t i l a w a **ʃ** ⋉

*⋊ **s** t a j a n o w o n w a **ʃ** ⋉

Summary: The big picture

- **Observation**: traditional computational classes don't provide a tight fit for language
- **Goal**: find subregular classes that have enough power for attested phenomena while ruling out unattested ones
- **Motivation**
  1. explain typology
  2. insights into required cognitive resources
  3. learning algorithms
  4. later on: parallels between phonology, syntax, semantics

Summary: SL

- SL-$n \equiv$ CNL-$n \equiv$ heavily restricted FO
- handles local dependencies within n segments
- efficient learning algorithm
- minimal cognitive resources
- **But:** fails for long-distance dependencies
  $\Rightarrow$ SL is not enough, **we need more power**

References

Applegate, Richard B. 1972. *Ineseño Chumash grammar*. Doctoral Dissertation, University of California, Berkeley.

Heinz, Jeffrey. 2014. Culminativity times harmony equals unbounded stress. In *Word stress: Theoretical and typological issues*, ed. Harry van der Hulst, 255–275. Cambridge, UK: Cambridge University Press.

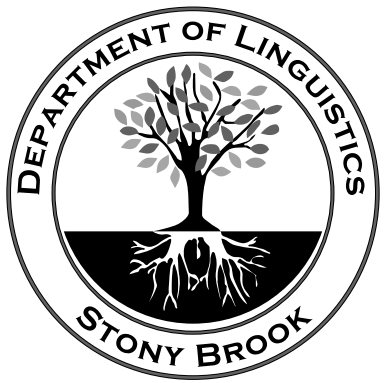
Kaplan, Ronald M., and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics* 20:331–378. URL `http://www.aclweb.org/anthology/J94-3001.pdf`.

Savitch, Walter J. 1993. Why it might pay to assume that languages are infinite. *Annals of Mathematics and Artificial Intelligence* 8:17–25.

The Computational Nature of Language

Part 3: Is syntax FO?

Thomas Graf



Stony Brook University
mail@thomasgraf.net
https://thomasgraf.net

ESSLLI 2021
July 26–30, 2021

Get the slides at:
https://esslli2021.thomasgraf.net

Outline

The received point of view

- **Our goal:** show that syntax is subregular
- **Our problem:** syntax isn't even regular

regular $<$ context-free $<$ mildly context-sensitive $<$ $\cdots$

Phonology

Syntax

The received point of view

- **Our goal:** show that syntax is subregular
- **Our problem:** syntax isn't even regular

regular $<$ context-free $<$ mildly context-sensitive $<$ $\cdots$

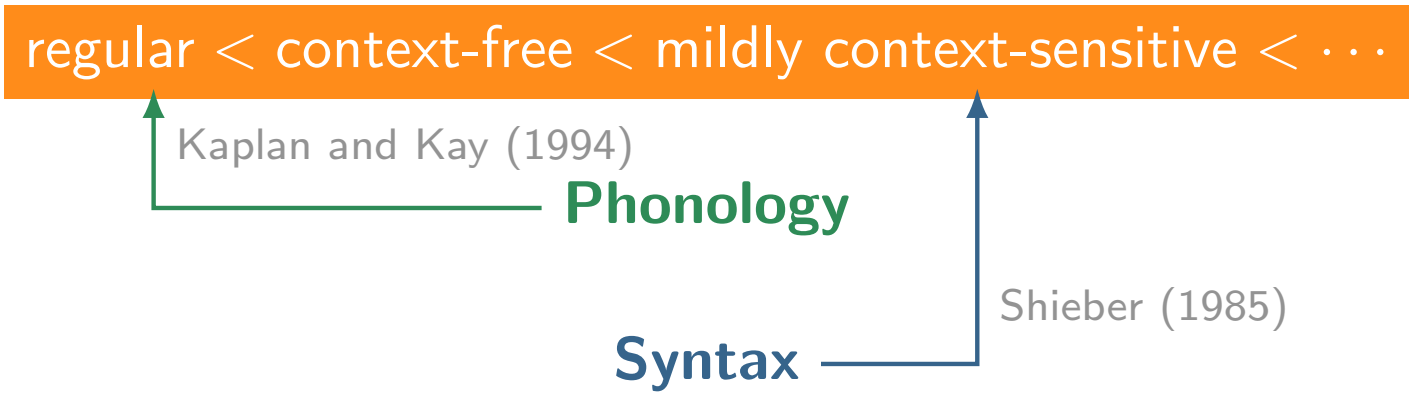
Kaplan and Kay (1994)

Phonology

Syntax

The received point of view

- **Our goal:** show that syntax is subregular
- **Our problem:** syntax isn't even regular



What does it mean (not) to be regular?

- Regularity can be defined in many equivalent ways.
- A string set L is regular iff:

1. it is recognized by a finite-state automaton.
2. it can be described by a regular expression.
3. it is generated by a right-linear grammar.
4. it is definable in monadic second-order logic (MSO).
5. it has a cylindrifaction that is SL-2.
6. its Myhill-Nerode congruence relation has finite index.

- Very cool. But:

What does it mean (not) to be regular?

- Regularity can be defined in many equivalent ways.
- A string set L is regular iff:

1. it is recognized by a finite-state automaton.
2. it can be described by a regular expression.
3. it is generated by a right-linear grammar.
4. it is definable in monadic second-order logic (MSO).
5. it has a cylindrifaction that is SL-2.
6. its Myhill-Nerode congruence relation has finite index.

- Very cool. But: what

What does it mean (not) to be regular?

- Regularity can be defined in many equivalent ways.
- A string set L is regular iff:

1. it is recognized by a finite-state automaton.
2. it can be described by a regular expression.
3. it is generated by a right-linear grammar.
4. it is definable in monadic second-order logic (MSO).
5. it has a cylindrifaction that is SL-2.
6. its Myhill-Nerode congruence relation has finite index.

- Very cool. But: what does

What does it mean (not) to be regular?

- Regularity can be defined in many equivalent ways.
- A string set L is regular iff:

1. it is recognized by a finite-state automaton.
2. it can be described by a regular expression.
3. it is generated by a right-linear grammar.
4. it is definable in monadic second-order logic (MSO).
5. it has a cylindrifaction that is SL-2.
6. its Myhill-Nerode congruence relation has finite index.

- Very cool. But: what does it

What does it mean (not) to be regular?

- Regularity can be defined in many equivalent ways.
- A string set L is regular iff:

1. it is recognized by a finite-state automaton.
2. it can be described by a regular expression.
3. it is generated by a right-linear grammar.
4. it is definable in monadic second-order logic (MSO).
5. it has a cylindrifaction that is SL-2.
6. its Myhill-Nerode congruence relation has finite index.

- Very cool. But: what does it **mean?**

What does it mean (not) to be regular?

- Regularity can be defined in many equivalent ways.
- A string set L is regular iff:

1. it is recognized by a finite-state automaton.
2. it can be described by a regular expression.
3. it is generated by a right-linear grammar.
4. it is definable in monadic second-order logic (MSO).
5. it has a cylindrifaction that is SL-2.
6. its Myhill-Nerode congruence relation has finite index.

- Very cool. But: what does it **mean?!!!1!11!**

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε	

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε	ε

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε	$\varepsilon,\ ab$

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε	$\varepsilon,\ ab,\ abab$

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε	$\varepsilon,\ ab,\ abab,\ \ldots$

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε	$\varepsilon,\ ab,\ abab,\ \ldots$
a	

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε	$\varepsilon,\ ab,\ abab,\ \ldots$
a	b

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε	$\varepsilon,\ ab,\ abab,\ \ldots$
a	$b,\ bab$

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε	$\varepsilon,\ ab,\ abab,\ \ldots$
a	$b,\ bab,\ babab$

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε	$\varepsilon,\ ab,\ abab,\ \ldots$
a	$b,\ bab,\ babab,\ \ldots$

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε	$\varepsilon,\ ab,\ abab,\ \ldots$
a	$b,\ bab,\ babab,\ \ldots$
b	

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε	$\varepsilon,\ ab,\ abab,\ \ldots$
a	$b,\ bab,\ babab,\ \ldots$
b	none

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε	$\varepsilon,\ ab,\ abab,\ \ldots$
a	$b,\ bab,\ babab,\ \ldots$
b	none
aa	

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε	$\varepsilon,\ ab,\ abab,\ \ldots$
a	$b,\ bab,\ babab,\ \ldots$
b	none
aa	none

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε	$\varepsilon,\ ab,\ abab,\ \ldots$
a	$b,\ bab,\ babab,\ \ldots$
$b,\ aa$	none

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε	$\varepsilon,\ ab,\ abab,\ \ldots$
a	$b,\ bab,\ babab,\ \ldots$
$b,\ aa$	none
ab	

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε	$\varepsilon,\ ab,\ abab,\ \ldots$
a	$b,\ bab,\ babab,\ \ldots$
$b,\ aa$	none
ab	ε

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε	$\varepsilon,\ ab,\ abab,\ \ldots$
a	$b,\ bab,\ babab,\ \ldots$
$b,\ aa$	none
ab	$\varepsilon,\ ab$

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε	$\varepsilon,\ ab,\ abab,\ \ldots$
a	$b,\ bab,\ babab,\ \ldots$
$b,\ aa$	none
ab	$\varepsilon,\ ab,\ abab$

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε	$\varepsilon,\ ab,\ abab,\ \ldots$
a	$b,\ bab,\ babab,\ \ldots$
$b,\ aa$	none
ab	$\varepsilon,\ ab,\ abab,\ \ldots$

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε, ab	ε, ab, $abab$, $\ldots$
a	b, bab, $babab$, $\ldots$
b, aa	none

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε, ab	ε, ab, $abab$, $\ldots$
a	b, bab, $babab$, $\ldots$
b, aa	none
ba	

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε, ab	ε, ab, $abab$, $\ldots$
a	b, bab, $babab$, $\ldots$
b, aa	none
ba	none

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε, ab	ε, ab, $abab$, $\ldots$
a	b, bab, $babab$, $\ldots$
b, aa, ba	none

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε, ab	ε, ab, $abab$, $\ldots$
a	b, bab, $babab$, $\ldots$
b, aa, ba	none
bb	

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε, ab	ε, ab, $abab$, ...
a	b, bab, $babab$, ...
b, aa, ba	none
bb	none

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε, ab	ε, ab, $abab$, $\ldots$
a	b, bab, $babab$, $\ldots$
b, aa, ba, bb	none

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε, ab	ε, ab, $abab$, $\ldots$
a	b, bab, $babab$, $\ldots$
b, aa, ba, bb	none
aaa	

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε, ab	ε, ab, $abab$, $\ldots$
a	b, bab, $babab$, $\ldots$
b, aa, ba, bb	none
aaa	none

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε, ab	ε, ab, $abab$, $\ldots$
a	b, bab, $babab$, $\ldots$
b, aa, ba, bb, aaa	none

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε, ab	ε, ab, $abab$, $\ldots$
a	b, bab, $babab$, $\ldots$
b, aa, ba, bb, aaa	none
aab	

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε, ab	ε, ab, $abab$, $\ldots$
a	b, bab, $babab$, $\ldots$
b, aa, ba, bb, aaa	none
aab	none

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
$\varepsilon,\ ab$	$\varepsilon,\ ab,\ abab,\ \ldots$
a	$b,\ bab,\ babab,\ \ldots$
$b,\ aa,\ ba,\ bb,\ aaa,\ aab$	none

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε, ab	ε, ab, $abab$, $\ldots$
a	b, bab, $babab$, $\ldots$
b, aa, ba, bb, aaa, aab	none
aba	

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
$\varepsilon,\ ab$	$\varepsilon,\ ab,\ abab,\ \ldots$
a	$b,\ bab,\ babab,\ \ldots$
$b,\ aa,\ ba,\ bb,\ aaa,\ aab$	none
aba	b

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
$\varepsilon,\ ab$	$\varepsilon,\ ab,\ abab,\ \ldots$
a	$b,\ bab,\ babab,\ \ldots$
$b,\ aa,\ ba,\ bb,\ aaa,\ aab$	none
aba	$b,\ bab$

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
$\varepsilon,\ ab$	$\varepsilon,\ ab,\ abab,\ \ldots$
a	$b,\ bab,\ babab,\ \ldots$
$b,\ aa,\ ba,\ bb,\ aaa,\ aab$	none
aba	$b,\ bab,\ babab$

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
$\varepsilon,\ ab$	$\varepsilon,\ ab,\ abab,\ \ldots$
a	$b,\ bab,\ babab,\ \ldots$
$b,\ aa,\ ba,\ bb,\ aaa,\ aab$	none
aba	$b,\ bab,\ babab,\ \ldots$

Myhill-Nerode theorem (simpler than it looks)

Theorem (slightly paraphrased)

A string set L is regular iff
the Myhill-Nerode congruence table for L has finitely many rows.

Example: Constructing a Myhill-Nerode congruence table

- Consider the string set $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.
- Now let us look at all possible strings over a and b, and group them by **their good continuations**.

Strings	**Good continuations wrt** $(ab)^*$
ε, ab	ε, ab, $abab$, $\ldots$
a, aba	b, bab, $babab$, $\ldots$
b, aa, ba, bb, aaa, aab	none

Showing that $a^n b^n$ is not regular

- The Myhill-Nerode theorem's primary use is show that something is **not regular**.
- Consider $a^n b^n = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$.
  Its table keeps growing indefinitely.

Example

Strings	**Good continuations wrt** $a^n b^n$

Showing that $a^n b^n$ is not regular

- The Myhill-Nerode theorem's primary use is show that something is **not regular**.
- Consider $a^n b^n = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$.
  Its table keeps growing indefinitely.

Example

Strings	**Good continuations wrt** $a^n b^n$
ε	

Showing that $a^n b^n$ is not regular

- The Myhill-Nerode theorem's primary use is show that something is **not regular**.
- Consider $a^n b^n = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$.
  Its table keeps growing indefinitely.

Example

Strings	**Good continuations wrt** $a^n b^n$
ε	$\varepsilon, ab, aaabb, aaabbb$

Showing that $a^n b^n$ is not regular

- The Myhill-Nerode theorem's primary use is show that something is **not regular**.
- Consider $a^n b^n = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$.
  Its table keeps growing indefinitely.

Example

Strings	**Good continuations wrt** $a^n b^n$
ε	$\varepsilon, ab, aaabb, aaabbb$
a	

Showing that $a^n b^n$ is not regular

- The Myhill-Nerode theorem's primary use is show that something is **not regular**.
- Consider $a^n b^n = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$.
  Its table keeps growing indefinitely.

Example

Strings	**Good continuations wrt** $a^n b^n$
ε	$\varepsilon, ab, aaabb, aaabbb$
a	$b, abb, aabbb, \ldots$

Showing that $a^n b^n$ is not regular

- The Myhill-Nerode theorem's primary use is show that something is **not regular**.
- Consider $a^n b^n = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$.
  Its table keeps growing indefinitely.

Example

Strings	**Good continuations wrt** $a^n b^n$
ε	$\varepsilon, ab, aaabb, aaabbb$
a	$b, abb, aabbb, \ldots$
b	

Showing that $a^n b^n$ is not regular

- The Myhill-Nerode theorem's primary use is show that something is **not regular**.
- Consider $a^n b^n = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$.
  Its table keeps growing indefinitely.

Example

Strings	**Good continuations wrt** $a^n b^n$
ε	$\varepsilon, ab, aaabb, aaabbb$
a	$b, abb, aabbb, \ldots$
b	none

Showing that $a^n b^n$ is not regular

- The Myhill-Nerode theorem's primary use is show that something is **not regular**.
- Consider $a^n b^n = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$.
  Its table keeps growing indefinitely.

Example

Strings	**Good continuations wrt** $a^n b^n$
ε	$\varepsilon, ab, aaabb, aaabbb$
a	$b, abb, aabbb, \ldots$
b	none
aa	

Showing that $a^n b^n$ is not regular

- The Myhill-Nerode theorem's primary use is show that something is **not regular**.
- Consider $a^n b^n = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$.
  Its table keeps growing indefinitely.

Example

Strings	**Good continuations wrt** $a^n b^n$
ε	$\varepsilon, ab, aaabb, aaabbb$
a	$b, abb, aabbb, \ldots$
b	none
aa	$bb, abbb, aabbbb, \ldots$

Showing that $a^n b^n$ is not regular

- The Myhill-Nerode theorem's primary use is show that something is **not regular**.
- Consider $a^n b^n = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$.
  Its table keeps growing indefinitely.

Example

Strings	**Good continuations wrt** $a^n b^n$
ε	$\varepsilon, ab, aaabb, aaabbb$
a	$b, abb, aabbb, \ldots$
b	none
aa	$bb, abbb, aabbbb, \ldots$
ab	

Showing that $a^n b^n$ is not regular

- The Myhill-Nerode theorem's primary use is show that something is **not regular**.
- Consider $a^n b^n = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$.
  Its table keeps growing indefinitely.

Example

Strings	**Good continuations wrt** $a^n b^n$
ε	$\varepsilon, ab, aaabb, aaabbb$
a	$b, abb, aabbb, \ldots$
b	none
aa	$bb, abbb, aabbbb, \ldots$
ab	ε

Showing that $a^n b^n$ is not regular

- The Myhill-Nerode theorem's primary use is show that something is **not regular**.
- Consider $a^n b^n = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$.
  Its table keeps growing indefinitely.

Example

Strings	**Good continuations wrt** $a^n b^n$
ε	$\varepsilon, ab, aaabb, aaabbb$
a	$b, abb, aabbb, \ldots$
$b,\ ba$	none
aa	$bb, abbb, aabbbb, \ldots$
ab	ε

Showing that $a^n b^n$ is not regular

- The Myhill-Nerode theorem's primary use is show that something is **not regular**.
- Consider $a^n b^n = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$.
  Its table keeps growing indefinitely.

Example

Strings	**Good continuations wrt** $a^n b^n$
ε	$\varepsilon, ab, aaabb, aaabbb$
a	$b, abb, aabbb, \ldots$
$b,\ ba,\ bb$	none
aa	$bb, abbb, aabbbb, \ldots$
ab	ε

Showing that $a^n b^n$ is not regular

- The Myhill-Nerode theorem's primary use is show that something is **not regular**.
- Consider $a^n b^n = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$.
  Its table keeps growing indefinitely.

Example

Strings	**Good continuations wrt** $a^n b^n$
ε	$\varepsilon, ab, aaabb, aaabbb$
a	$b, abb, aabbb, \ldots$
$b,\ ba,\ bb$	none
aa	$bb, abbb, aabbbb, \ldots$
ab	ε
aaa	

Showing that $a^n b^n$ is not regular

- The Myhill-Nerode theorem's primary use is show that something is **not regular**.
- Consider $a^n b^n = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$.
  Its table keeps growing indefinitely.

Example

Strings	**Good continuations wrt** $a^n b^n$
ε	$\varepsilon, ab, aaabb, aaabbb$
a	$b, abb, aabbb, \ldots$
$b,\ ba,\ bb$	none
aa	$bb, abbb, aabbbb, \ldots$
ab	ε
aaa	$bbb, abbbb, aabbbbb, \ldots$

Showing that $a^n b^n$ is not regular

- The Myhill-Nerode theorem's primary use is show that something is **not regular**.
- Consider $a^n b^n = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$.
  Its table keeps growing indefinitely.

Example

Strings	**Good continuations wrt** $a^n b^n$
ε	$\varepsilon, ab, aaabb, aaabbb$
a	$b, abb, aabbb, \ldots$
$b,\ ba,\ bb$	none
aa	$bb, abbb, aabbbb, \ldots$
ab	ε
aaa	$bbb, abbbb, aabbbbb, \ldots$
a^i	

Showing that $a^n b^n$ is not regular

- The Myhill-Nerode theorem's primary use is show that something is **not regular**.
- Consider $a^n b^n = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$.
  Its table keeps growing indefinitely.

Example

Strings	**Good continuations wrt** $a^n b^n$
ε	$\varepsilon, ab, aaabb, aaabbb$
a	$b, abb, aabbb, \ldots$
$b,\ ba,\ bb$	none
aa	$bb, abbb, aabbbb, \ldots$
ab	ε
aaa	$bbb, abbbb, aabbbbb, \ldots$
a^i	$a^j\, b^{i+j}$

Applying it to language

- The Myhill-Nerode theorem entails that a string set cannot be regular if it contains at least one of the following:

1. unbounded nested dependencies
2. unbounded crossing dependencies

- We find both in language.

Unbounded nested dependencies in English

1. **The cheese was rotten**.
2. **The cheese the mouse ate was rotten**.
3. **The cheese the mouse the cat chased ate was rotten**.
4. **The cheese the mouse the cat the dog barked at chased ate was rotten**.

Translation to formal string set

1. replace all nouns with a
2. replace all verbs with b
3. ignore the rest
4. et voila: $a^n b^n$

Note: relabelings and deletions preserve regularity, so if the construction were regular we wouldn't get a context-free string set at the end

Unbounded crossing dependencies in English

1. **John slept**.
2. **John** and **Mary slept** and **worked**, respectively.
3. **John** and **Mary** and **Peter slept** and **worked** and **partied**, respectively.
4. **John** and **Mary** and **Peter** and **Sue slept** and **worked** and **partied** and **enjoyed ESSLLI**, respectively.

Remark

- There's several confounds with this construction.
- But there's more robust patterns in other languages, like Dutch and Swiss German.
  (Huybregts 1984; Shieber 1985)

Subregularity dead in the water?

- There are at least some languages whose syntax is not regular.
- But then syntax cannot be subregular, either.
- Time to pack up, I guess...

Subregularity dead in the water?

- There are at least some languages whose syntax is not regular.
- But then syntax cannot be subregular, either.
- Time to pack up, I guess...

No, there's another way!

- These claims are about syntax as a set of strings.
- But is that actually what syntax is about?

Linguistics to the rescue

- Decades of linguistic research tell us that we should not think of sentences as this:

  Thomas loves his trees.

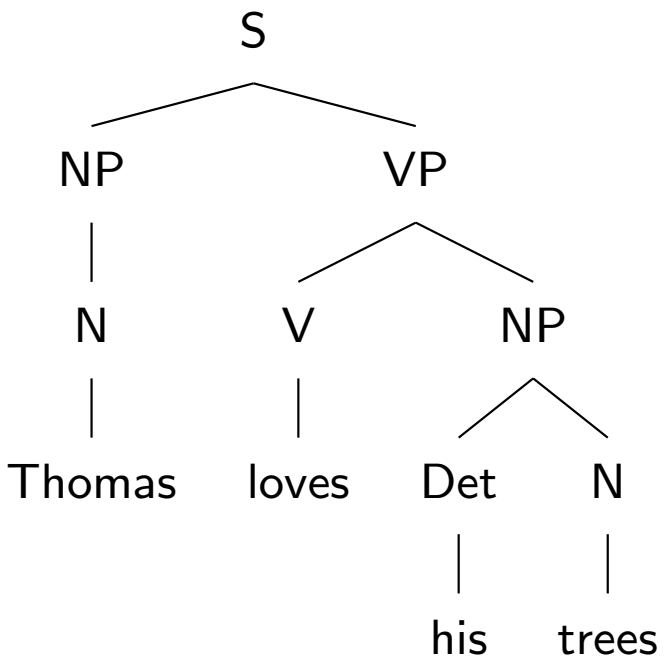
- We should think of them as something like this:

Linguistics to the rescue

- Decades of linguistic research tell us that we should not think of sentences as this:

  Thomas loves his trees.

- We should think of them as something like this:

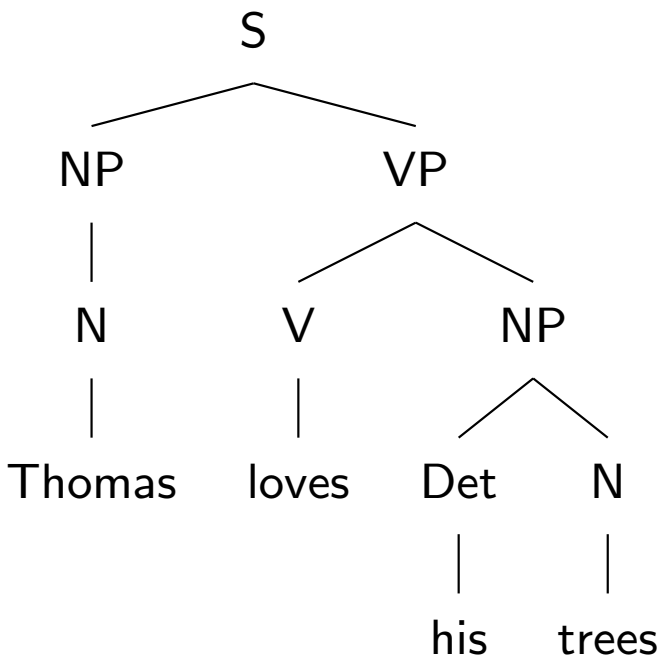


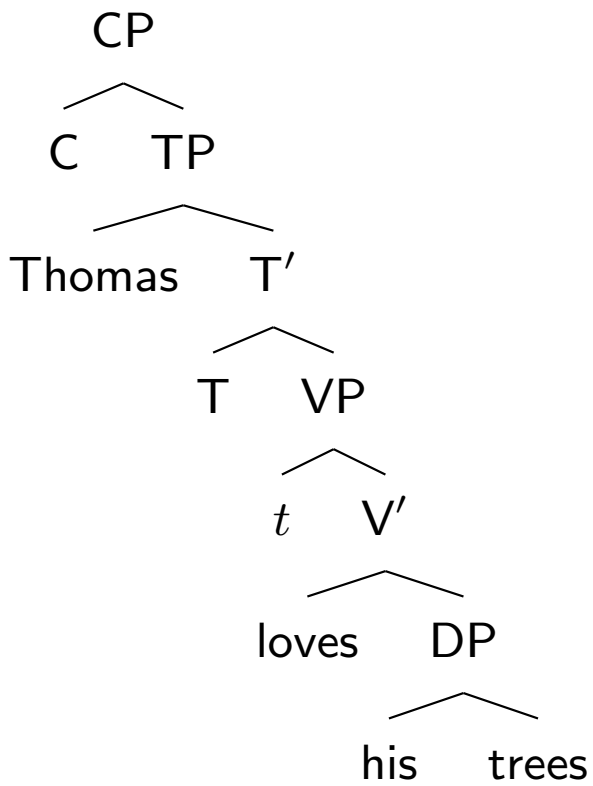
Linguistics to the rescue

- Decades of linguistic research tell us that we should not think of sentences as this:

  Thomas loves his trees.

- We should think of them as something like this:



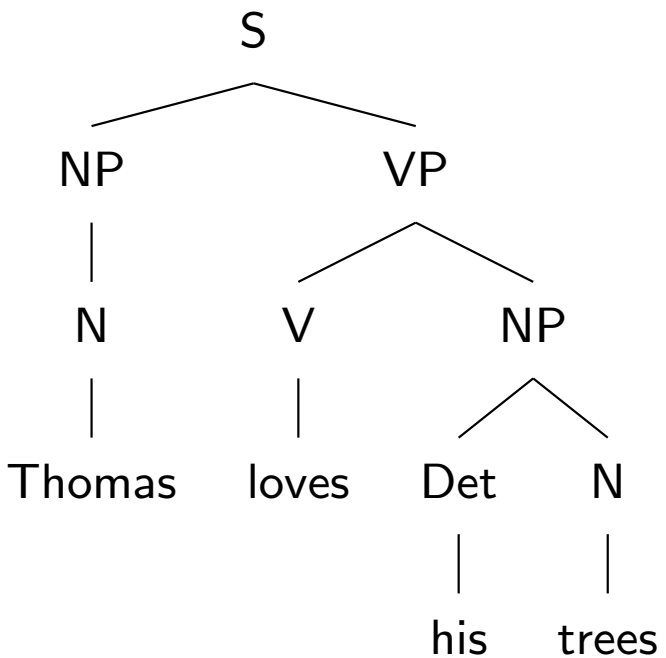


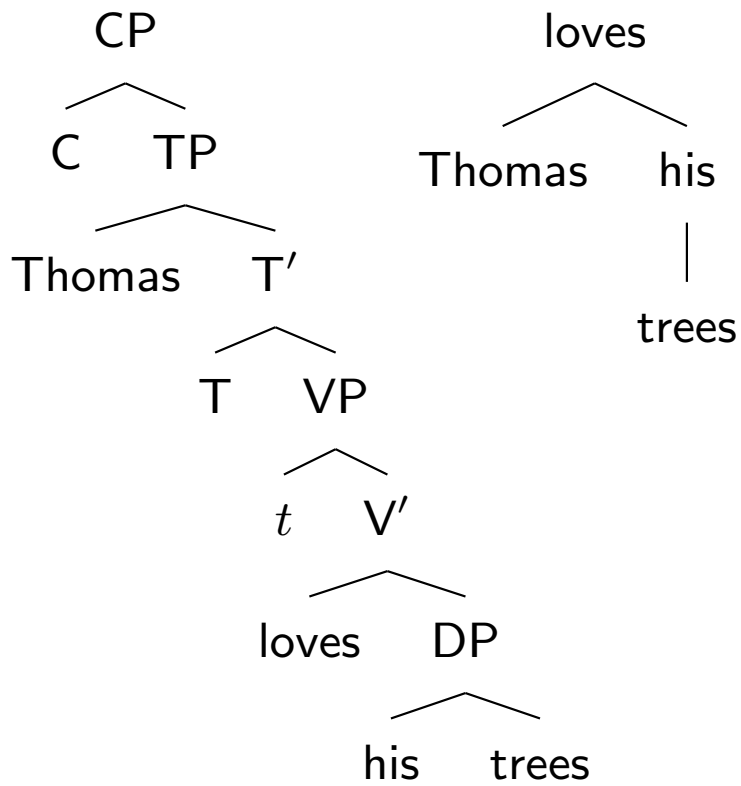
Linguistics to the rescue

- Decades of linguistic research tell us that we should not think of sentences as this:

  Thomas loves his trees.

- We should think of them as something like this:

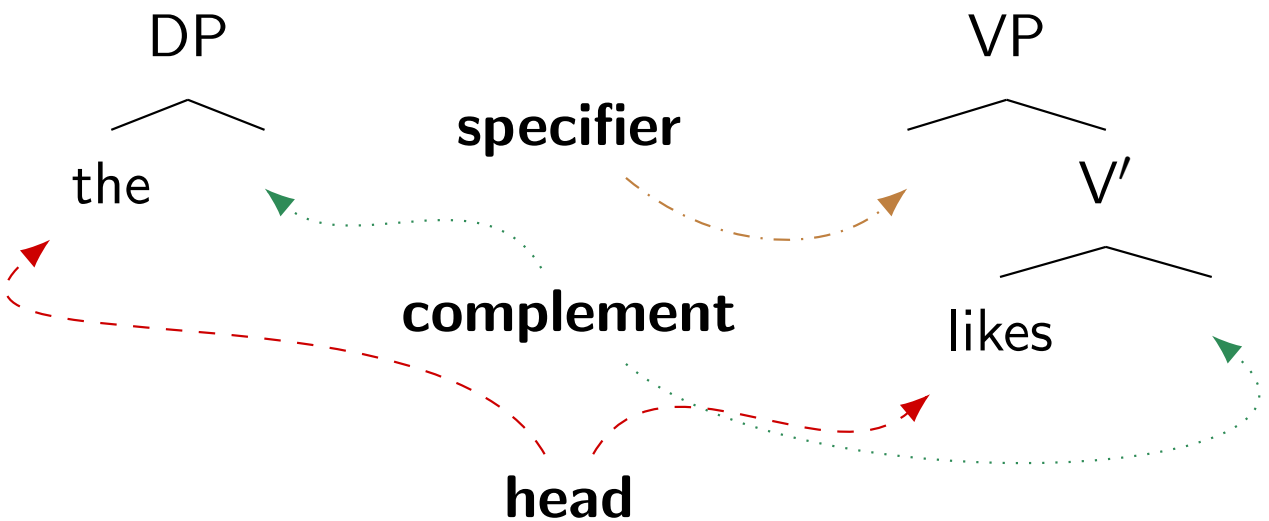




A quick primer on syntactic trees

- There's many competing proposals, we'll keep it general.
- Words combine into **phrases**:
  1. verb phrase (VP)
  2. noun phrase (NP)
  3. determiner phrase (DP)
  4. adjective phrase (AP)
  5. preposition phrase (PP)

  ⋮
- Every phrase has a **head**, which **selects** up to 2 **arguments**.



Category and selector features

- Heads only select arguments of specific categories.
  For example, *the* only selects NPs.
- We can express this with features.

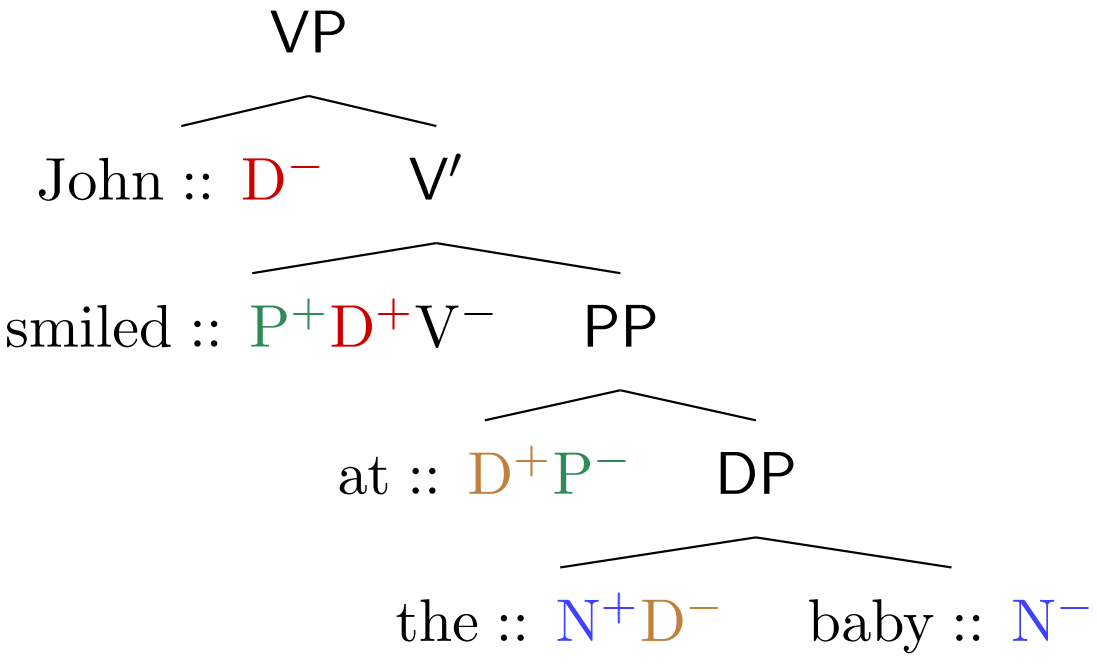
Category features	V^-, N^-, D^-, ...
Selector features	V^+, N^+, D^+, ...

Example

Lexical item	**Prose description**
the :: $\mathrm{N}^+\mathrm{D}^-$	*the* selects an NP to form a DP
smiles :: $\mathrm{P}^+\mathrm{D}^+\mathrm{V}^-$	*smiles* selects PP, then DP, and forms VP
smiles :: $\mathrm{D}^+\mathrm{V}^-$	*smiles* selects a DP and forms a VP
dog :: N^-	*dog* selects nothing and forms an NP

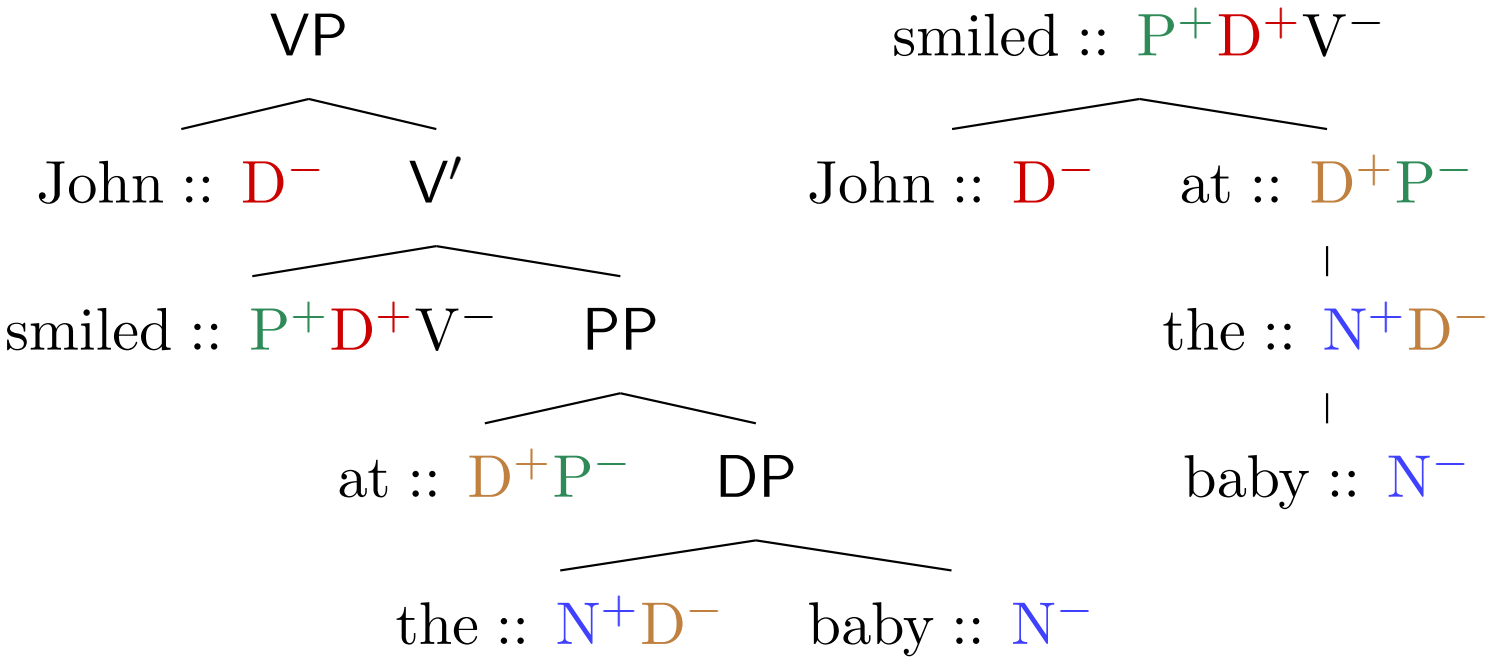
Syntax as chemistry

- Features tell us how words can combine.
- Features are like electrons that join words into **sentence molecules**.



Syntax as chemistry

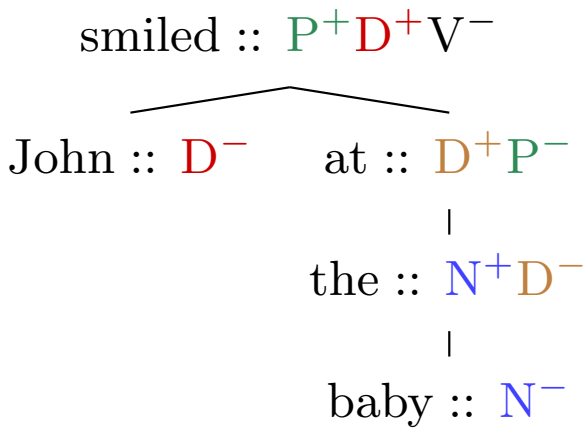
- Features tell us how words can combine.
- Features are like electrons that join words into **sentence molecules**.



Selection as a constraint on dependency trees

Selection as a constraint

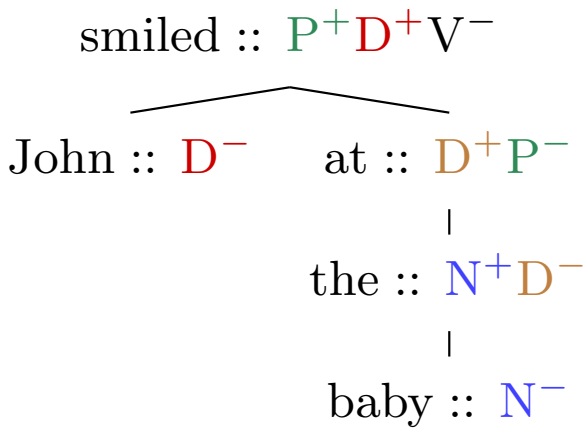
The **n**-th selection feature on a head must match
the category feature on the **n**-th daughter from the right.



Selection as a constraint on dependency trees

Selection as a constraint

The **n**-th selection feature on a head must match
the category feature on the **n**-th daughter from the right.



FO implementation

$\mathbf{x} \lhd_{\mathbf{i}} \mathbf{y}$ **y** is **x**'s **i**-th daughter from the right

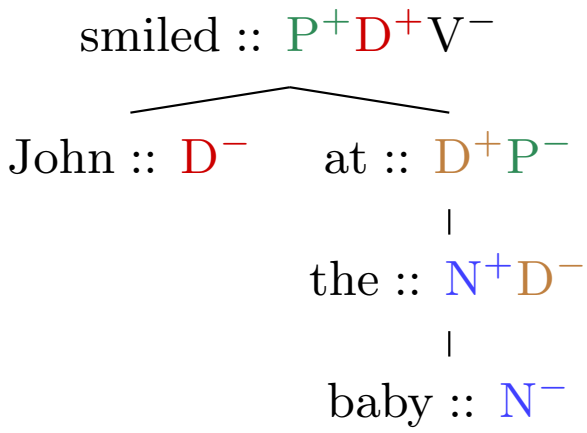
$\mathrm{F}^{-}(\mathbf{x})$ **x** has category feature F^{-}

$\mathrm{F}^{+}\mathbf{i}(\mathbf{x})$ **x**'s **i**-th selector feature is F^{+}

Selection as a constraint on dependency trees

Selection as a constraint

The **n**-th selection feature on a head must match
the category feature on the **n**-th daughter from the right.



FO implementation

$\mathbf{x} \lhd_{\mathbf{i}} \mathbf{y}$ — **y** is **x**'s **i**-th daughter from the right

$\mathrm{F}^{-}(\mathbf{x})$ — **x** has category feature F^{-}

$\mathrm{F}^{+}\mathbf{i}(\mathbf{x})$ — **x**'s **i**-th selector feature is F^{+}

$$\forall \mathbf{x}\Big[$$

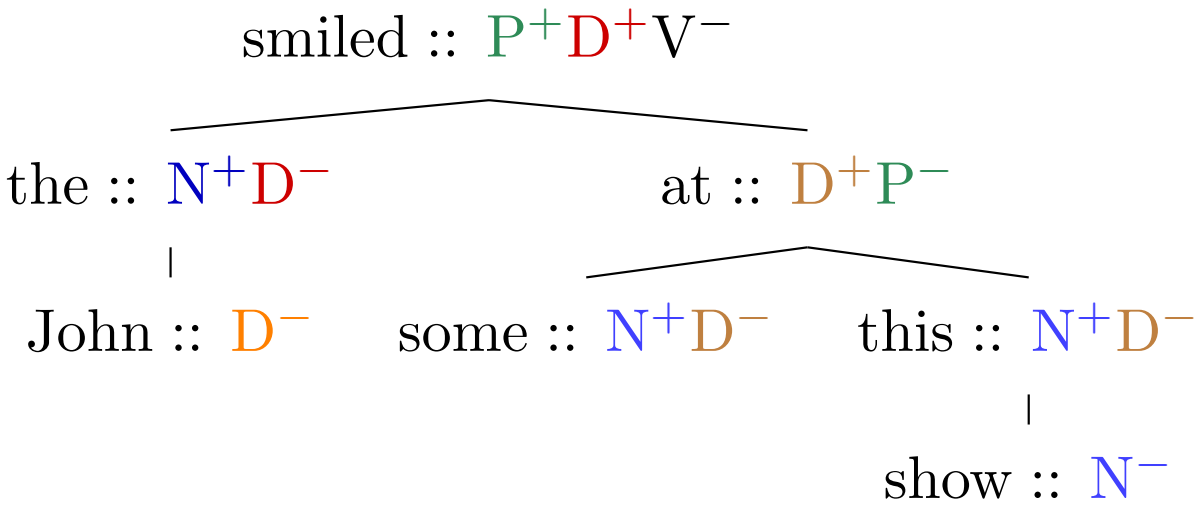
$$(\mathbf{F}^{+}\mathbf{1}(\mathbf{x}) \rightarrow \exists \mathbf{y}[\mathbf{x} \lhd_{\mathbf{1}} \mathbf{y} \wedge \mathbf{F}^{-}(\mathbf{y})]) \wedge (\mathbf{G}^{+}\mathbf{1}(\mathbf{x}) \rightarrow \exists \mathbf{y}[\mathbf{x} \lhd_{\mathbf{1}} \mathbf{y} \wedge \mathbf{G}^{-}(\mathbf{y})]) \wedge \cdots$$

$$(\mathbf{F}^{+}\mathbf{2}(\mathbf{x}) \rightarrow \exists \mathbf{y}[\mathbf{x} \lhd_{\mathbf{2}} \mathbf{y} \wedge \mathbf{F}^{-}(\mathbf{y})]) \wedge (\mathbf{G}^{+}\mathbf{2}(\mathbf{x}) \rightarrow \exists \mathbf{y}[\mathbf{x} \lhd_{\mathbf{2}} \mathbf{y} \wedge \mathbf{G}^{-}(\mathbf{y})]) \wedge \cdots$$

$$\Big]$$

An ill-formed dependency tree

The John smiled at some this show



Three problems

1. the argument of *the* has the wrong category
2. *at* has more arguments than selector features
3. *some* has fewer arguments than selector features

We're back in business. . . sorta

- FO is enough to define well-formed dependency trees with feature-based selection.
- But that's not quite enough for syntax.

The formal argument (sketched)

- Every FO-definable set of trees is a regular tree set.
- Every regular tree set yields a context-free string set.
- Wrt strings, syntax is at least mildly context-sensitive.
- context-free $<$ mildly context-sensitive

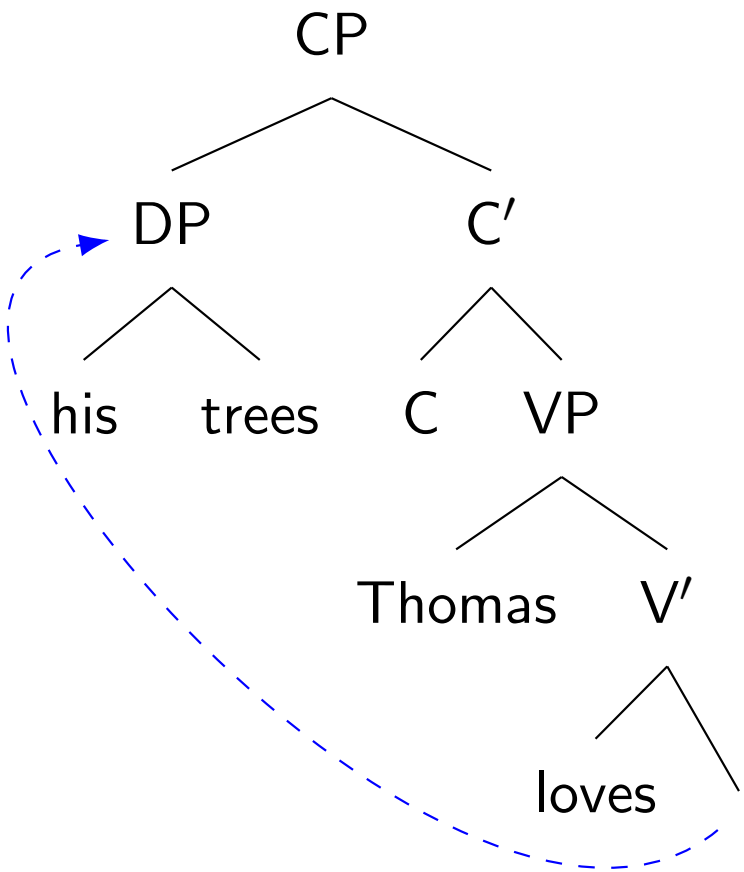
- One solution is to adopt the linguistic proposal of **Movement**.

Movement

1. Thomas loves his trees.
2. His trees, Thomas loves [his trees].

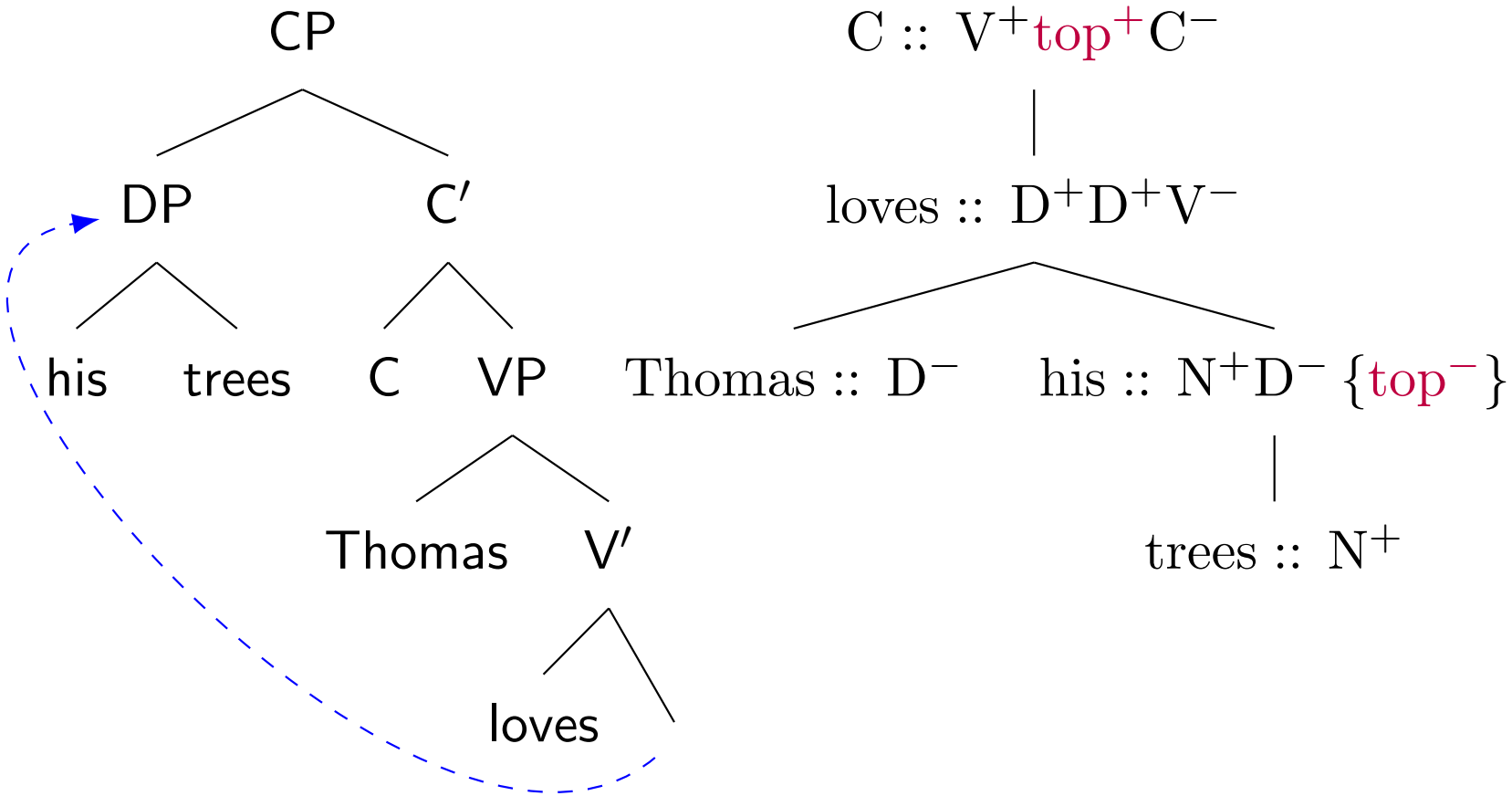
Movement

1. Thomas loves his trees.
2. His trees, Thomas loves [his trees].



Movement

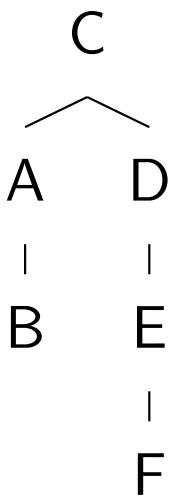
1. Thomas loves his trees.
2. His trees, Thomas loves [his trees].



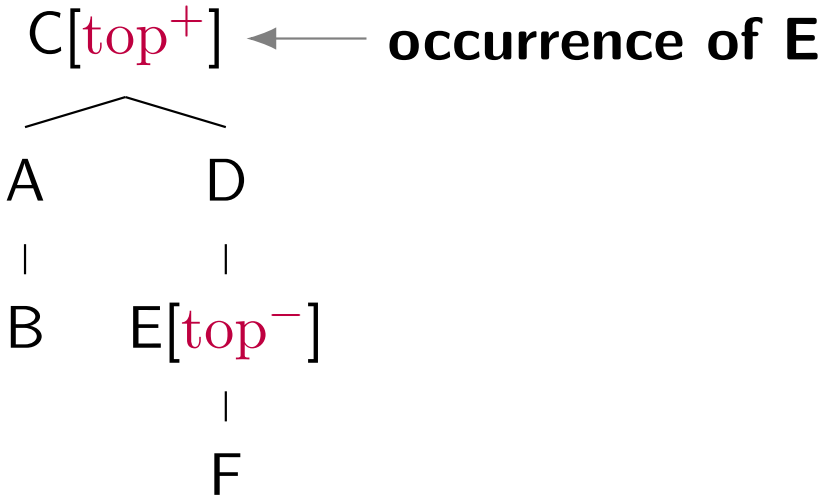
Movement in dependency trees

- Encoded via **licensor features** f^+ and **licensee features** f^-.
- Each head has at most one licensor feature.
- A head may have multiple licensee features, but they're unordered with respect to each other.
- Movement changes the linearization of dependency trees.

ABCDEF



EFABCD



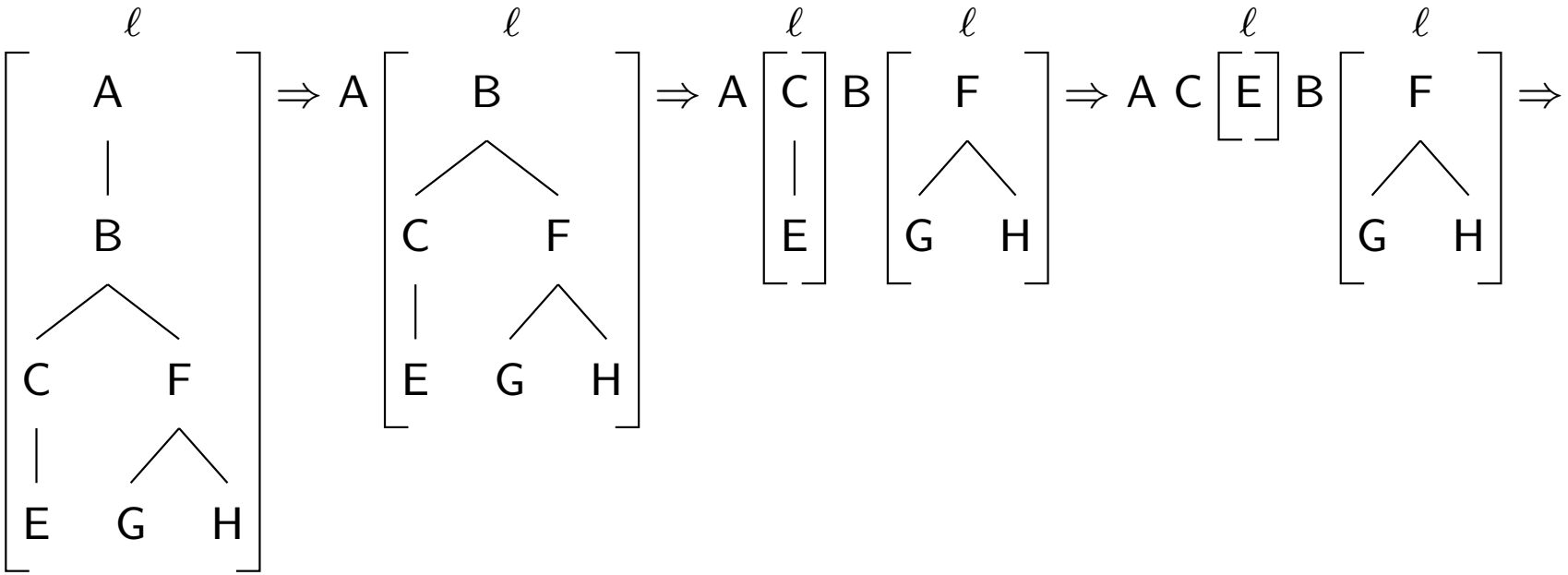
So what is the linearization rule?

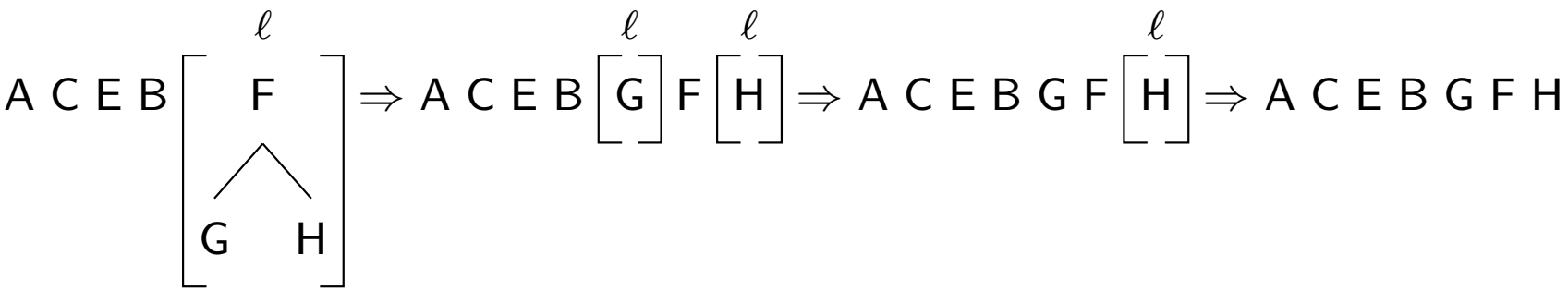
$$\ell(\mathbf{H}(\mathbf{A}_2, \mathbf{A}_1)) = \begin{cases} \varepsilon & \text{if } \mathbf{H} \text{ is a mover} \\ \mu(\mathbf{M})\ell(\mathbf{A}_2)\mathbf{H}\ell(\mathbf{A}_1) & \text{if } \mathbf{H} \text{ is the highest occurrence of } \mathbf{M} \\ \ell(\mathbf{A}_2)\mathbf{H}\ell(\mathbf{A}_1) & \text{otherwise} \end{cases}$$

$$\mu(\mathbf{H}(\mathbf{A}_2, \mathbf{A}_1)) = \ell(\mathbf{A}_2)\mathbf{H}\ell(\mathbf{A}_1)$$

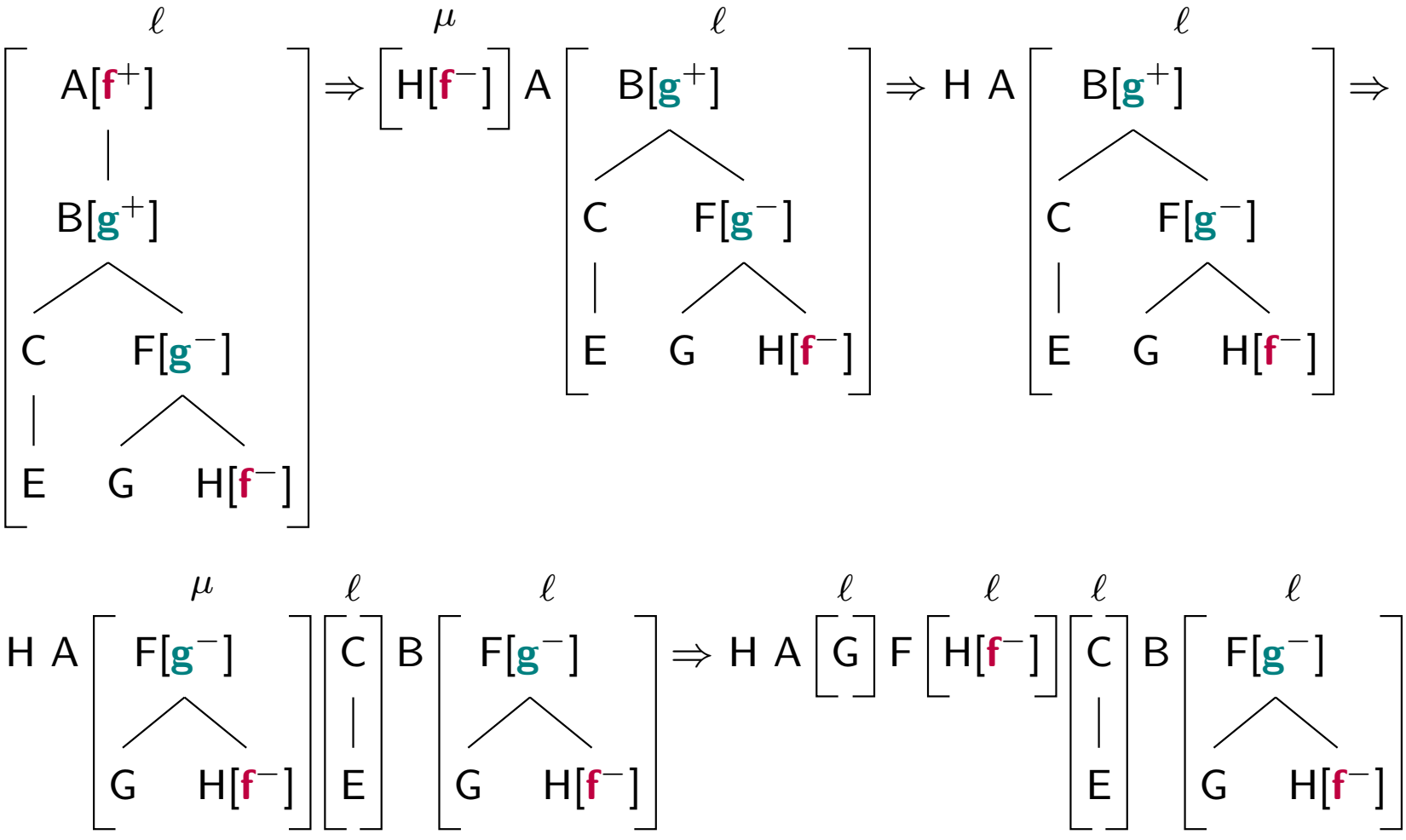
- This is FO-definable, but not for the faint of heart.
- I'll upload a separate document with the FO formulas.
- Let's focus on regulating movement features.

Detailed linearization example without movement

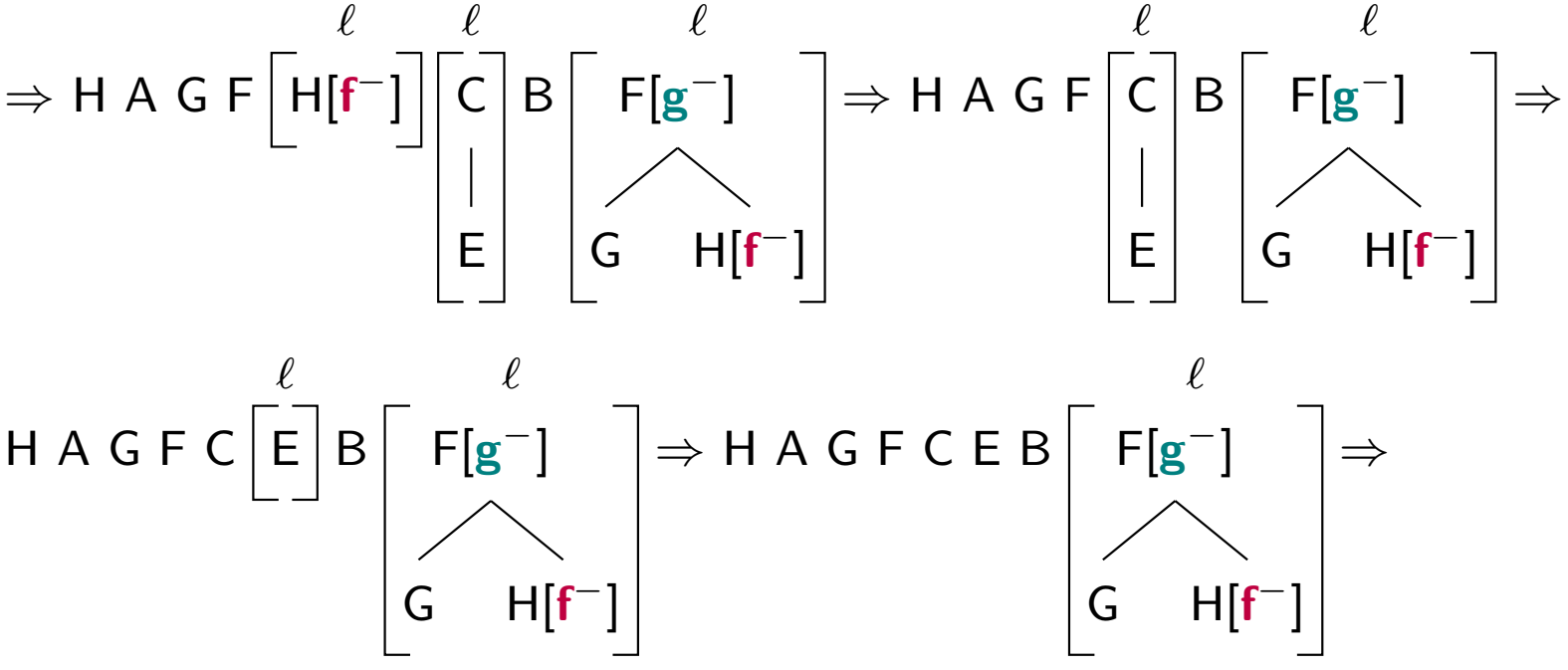




Detailed linearization example with movement

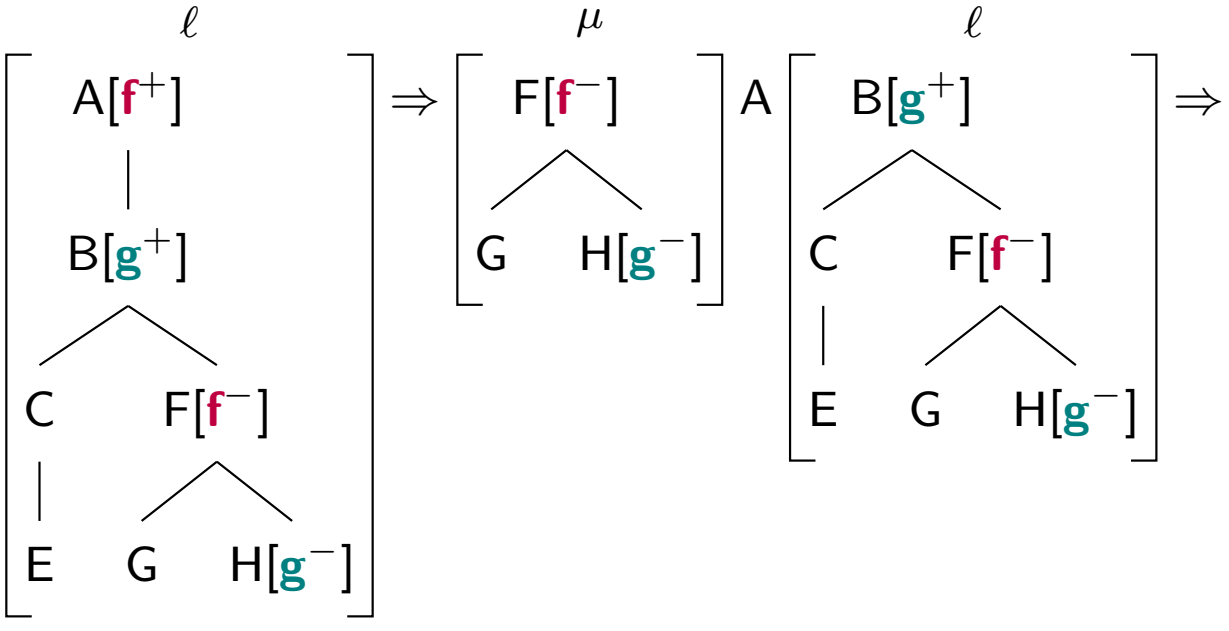


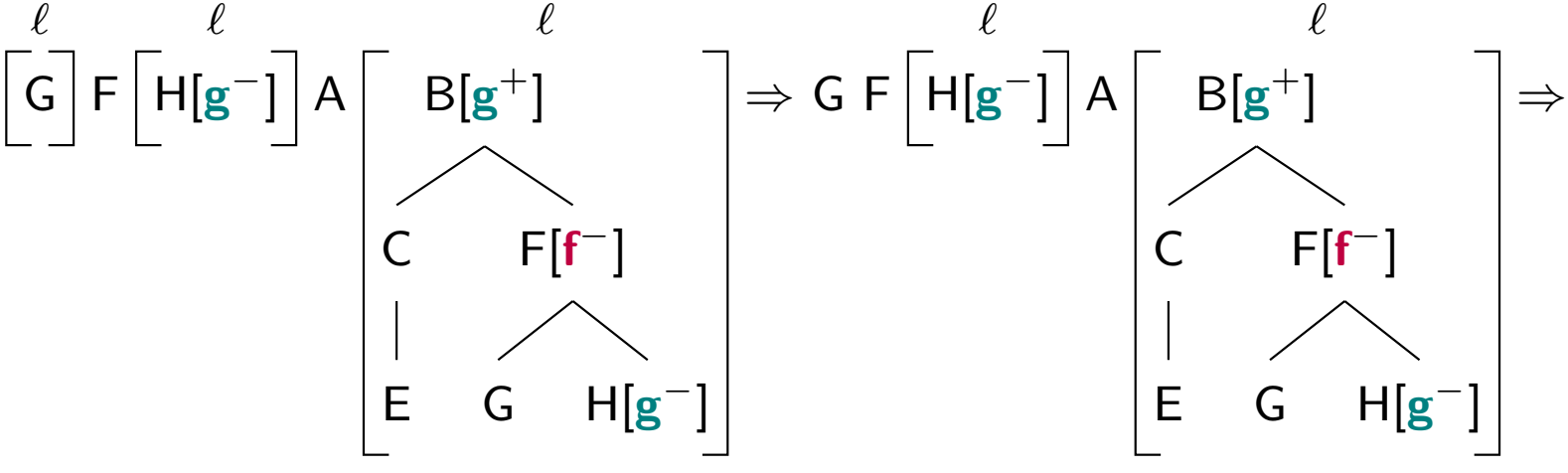
Detailed linearization example with movement [cont.]



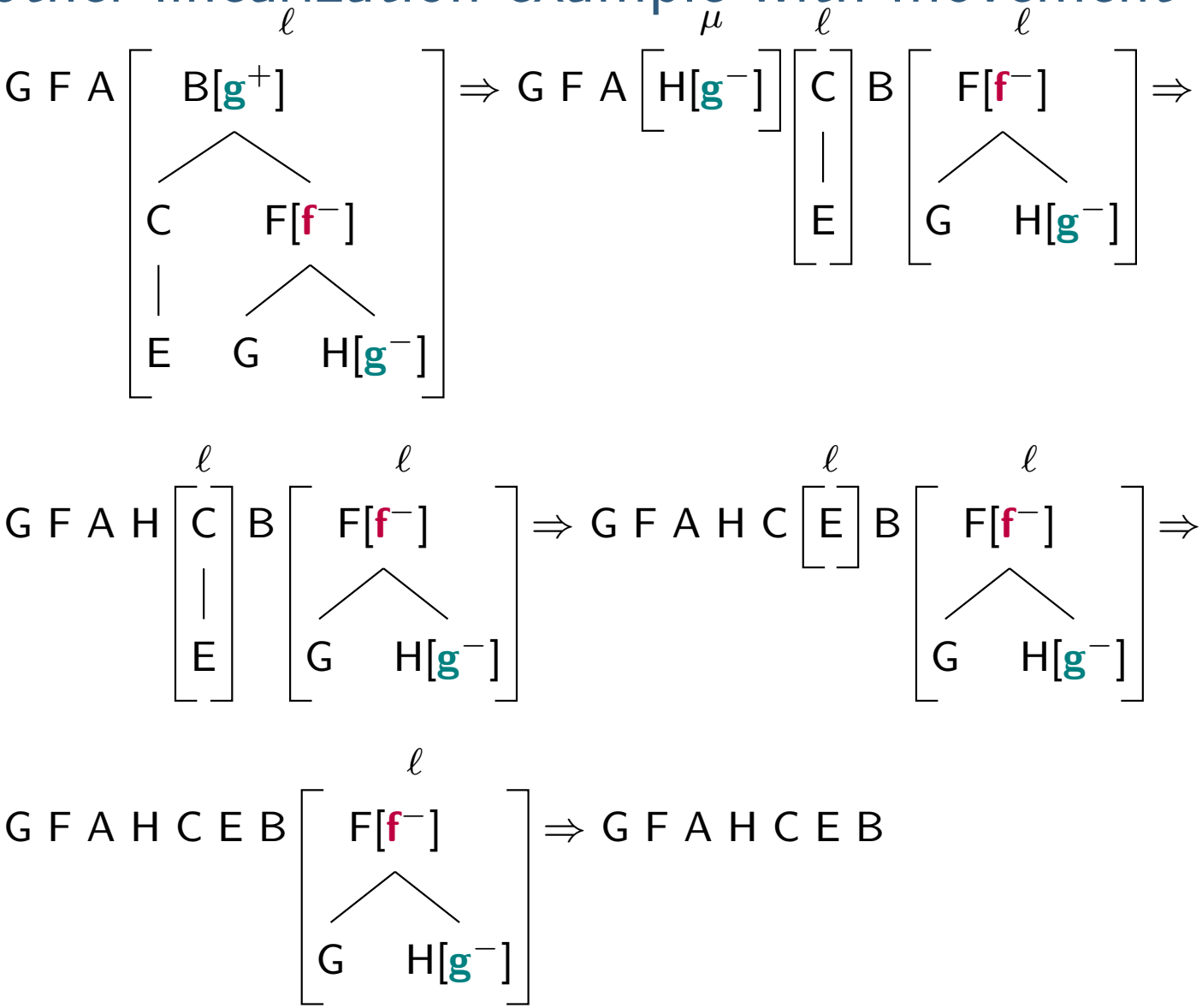
H A G F C E B

Another linearization example with movement

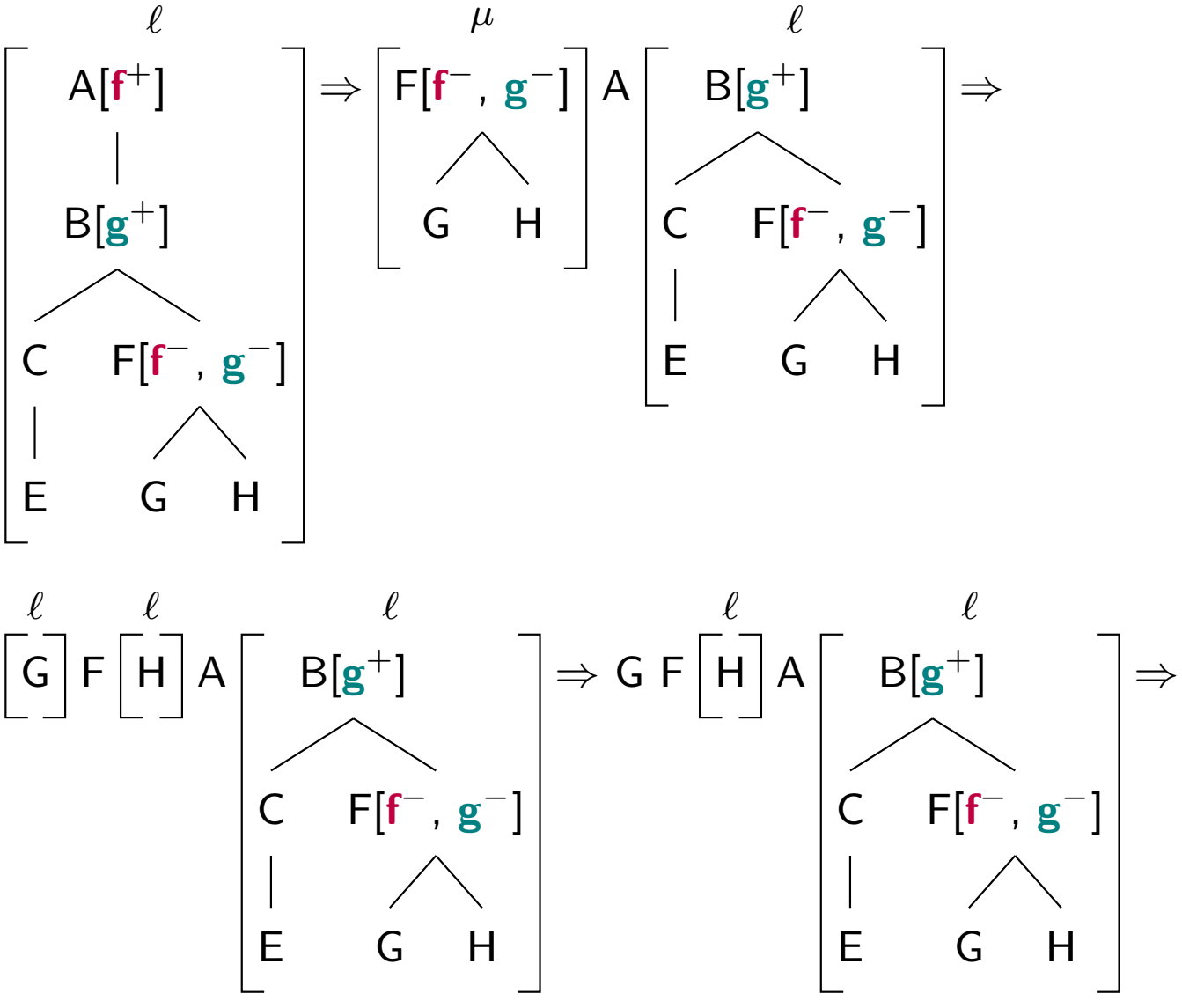




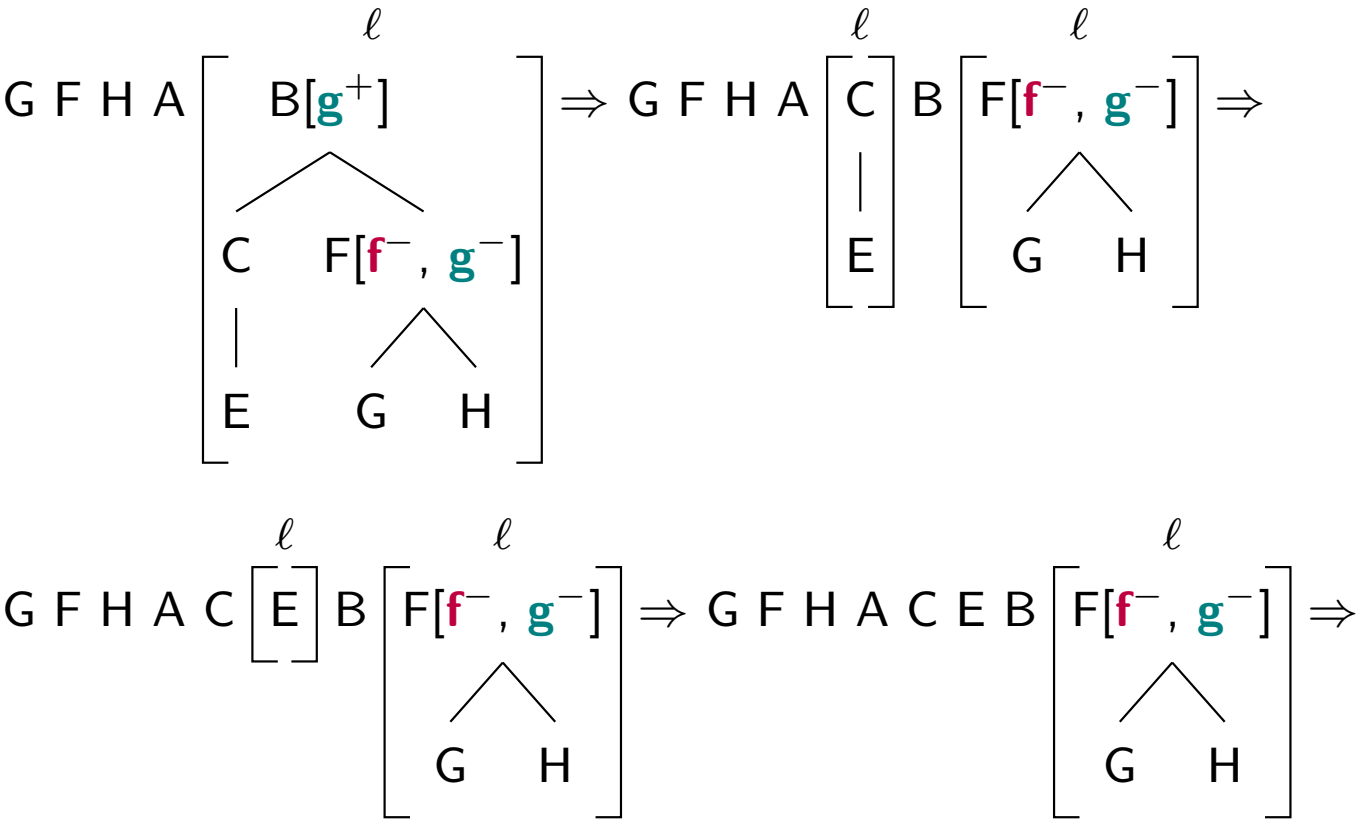
Another linearization example with movement [cont.]



A third linearization example with movement



A third linearization example with movement [cont.]



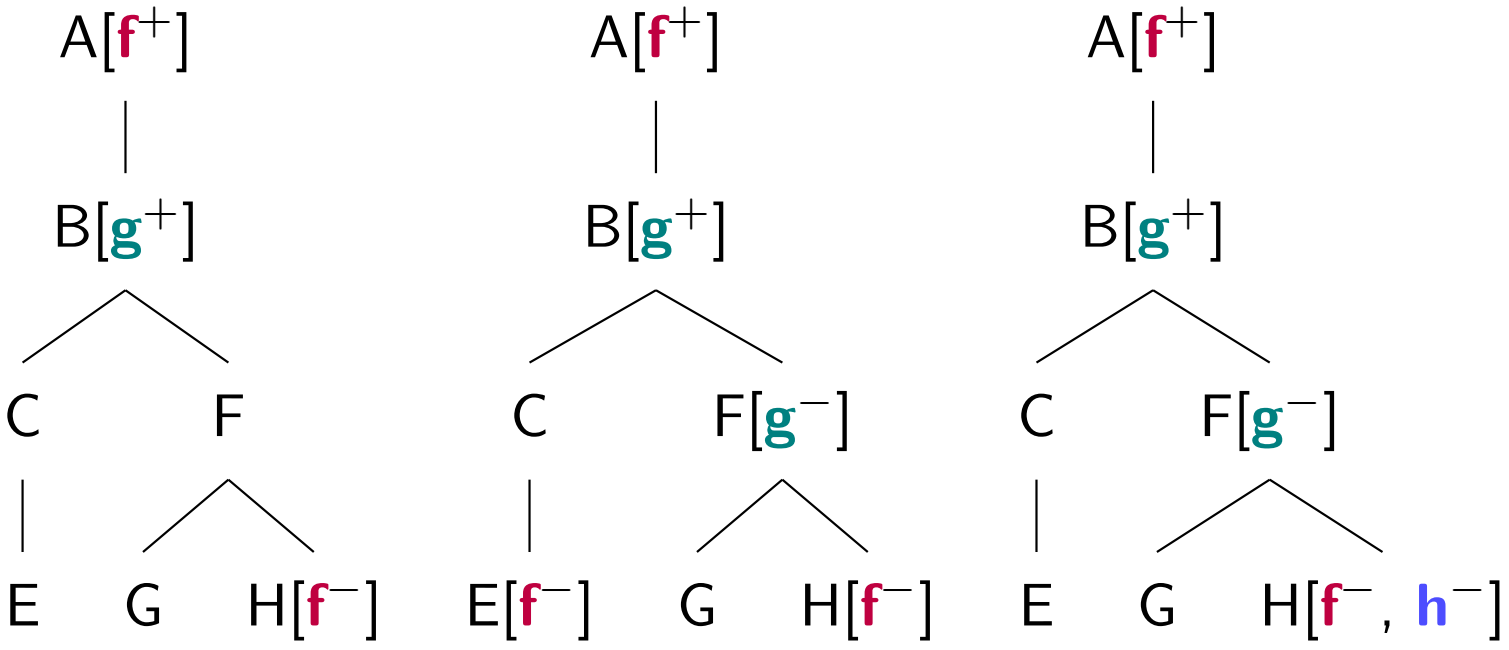
Two movement constraints

Definition (Occurrence)

If **x** carries licensee feature $\mathbf{f}^-$, then **y** is an **occurrence** of **x** iff it is the closest node above **x** that carries a matching $\mathbf{f}^+$.

1. **Every landing site needs a mover**
   If a lexical item has a licensor feature, it must be an occurrence for exactly one lexical item.
2. **Every movement step needs a landing site**
   If a lexical item has **n** licensee features, it must have **n** occurrences.

Examples of ill-formed trees



What's wrong

left B is not an occurrence

middle A is not an occurrence for two nodes

right H is missing a second occurrence

FO definition of the constraints

- $x \lhd^+ y$ x properly dominates y ("y is somewhere below x")
- $f^+(x)$ x carries licensor feature f^+
- $f^-(x)$ x carries licensee feature f^-
- $n(x)$ x carries n licensee features
- $\text{f-occ}(x,y) \Leftrightarrow x \lhd^+ y \wedge f^+(x) \wedge f^-(y) \wedge \neg\exists z[x \lhd^+ y \wedge z \lhd^+ y \wedge f^+(z)]$
- $\text{occ}(x,y) \Leftrightarrow \text{f-occ}(x,y) \vee \text{g-occ}(x,y) \vee \cdots$

1. $\forall x[(f^+(x) \vee g^+(x) \vee \cdots) \rightarrow \exists y[\text{occ}(x,y)]]$
2. $\forall x[$
   $(1(x) \rightarrow \exists y_1[\text{occ}(x,y_1)]) \wedge$
   $(2(x) \rightarrow \exists y_1 \exists y_2[\text{occ}(x,y_1) \wedge \text{occ}(x,y_2)] \wedge$
   $\cdots]$

What do we have?

- We have an FO-definable model of syntax based on trees.
  1. FO constraints on dependency trees
  2. FO formula defines linearization of tree
- It uses
  1. selector and category features for head-argument relations
  2. licensor and licensee features to change linear order.
- Every lexical item has
  1. at most two selector features
  2. at most one licensor feature
  3. exactly one category feature
  4. a finite number of licensee features
- But how do we know this is empirically adequate?

A small surprise

Ed Stabler

- Our model is actually a notational variant of Minimalist grammars (MGs). (Stabler 1997, 2011)
- modeled after Minimalist syntax
- They have very good empirical coverage: mildly context-sensitive
- They have tons of extensions, all of which are FO-definable! sideward movement, lowering movement, head movement, clustering movement, Late Merge, reference-set computation, ...

Syntax is FO over trees

- MGs are a very powerful model of syntax, with numerous extensions, yet MGs are FO-definable.
- Syntax is **FO over trees, not strings**.
  - FO constraints on trees
  - FO formula for linearization
- Subregular syntax is not dead in the water after all.

A glimpse of subregularity: islands

- Syntax being FO means it is subregular, but can we find weaker classes?
- How do we apply SL or TSL to trees?
- **One option:** decompose tree to **path language**

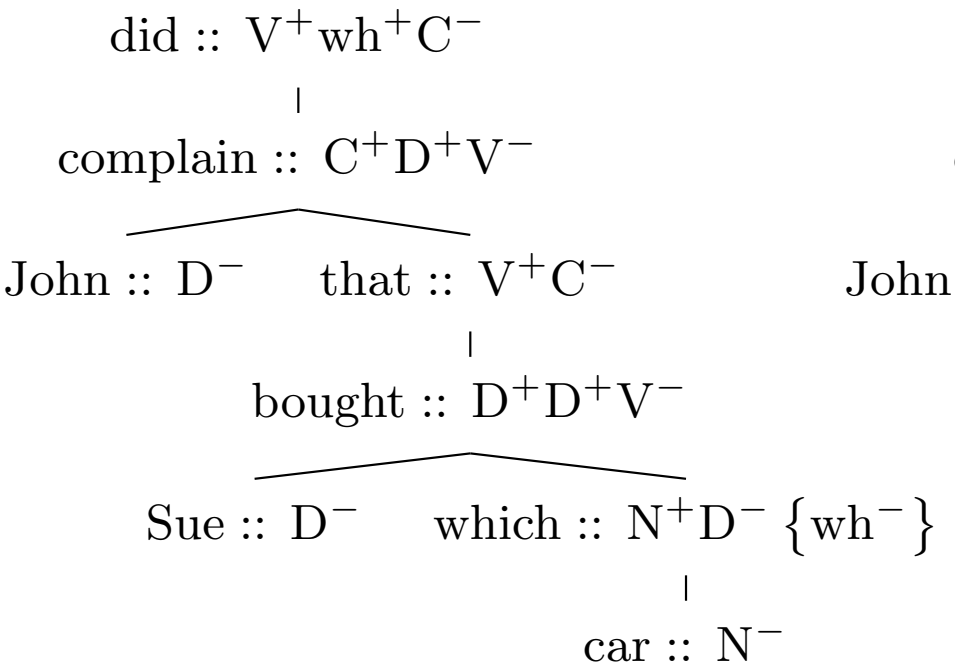
Example

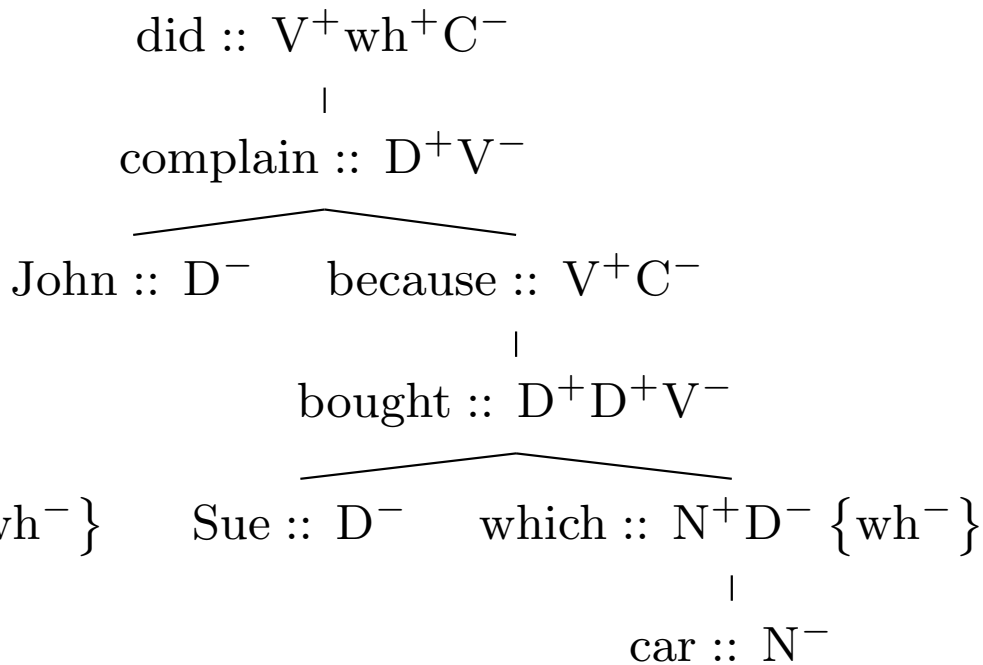
$$\left\{\begin{array}{c} C, \\ AC, \\ BAC, \\ DC, \\ EDC, \\ FEDC \end{array}\right\}$$

Application: Island effects

- Movement is not unrestricted, some phrases cannot be extracted from $\Rightarrow$ **islands**

1. Which car did John complain **that** Sue bought [which car].
2. *Which car did John complain **because** Sue bought [which car].





Comparing the paths of *which*

1. **which**[wh^-] bought **that** John complain **did**[wh^+]
2. **which**[wh^-] bought **because** John complain **did**[wh^+]

A simple movement constraint

For each movement type f (wh, top, . . .):
If H is the head of an island, then it may not occur between a node carrying f^- and its occurrence with the matching f^+.

Equivalent TSL constraint on path language

1. Project every node carrying f^- or f^+, and every head **H** of an island (e.g. *because*).
2. $^*\mathrm{f}^-$ **H**

Summary

- **Today's focus**: paving the way to subregular syntax
- Syntax is not regular over strings.
- But syntax is about **trees**!
- Over trees, FO is an empirically robust upper bound on syntax.
- We can bring in SL and TSL as constraints on path languages. (ongoing research)
- But what we really want to do is generalize them to trees.

References

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. URL `http://www.aclweb.org/anthology/J94-3001.pdf`.

Shieber, Stuart M. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy* 8:333–345. URL `http://dx.doi.org/10.1007/BF00630917`.

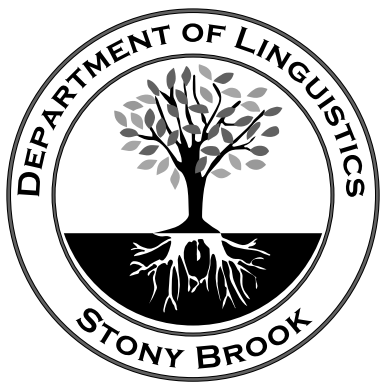
Stabler, Edward P. 1997. Derivational Minimalism. In *Logical aspects of computational linguistics*, ed. Christian Retoré, volume 1328 of *Lecture Notes in Computer Science*, 68–95. Berlin: Springer. URL `https://doi.org/10.1007/BFb0052152`.

Stabler, Edward P. 2011. Computational perspectives on Minimalism. In *Oxford handbook of linguistic Minimalism*, ed. Cedric Boeckx, 617–643. Oxford: Oxford University Press.

The Computational Nature of Language

Part 4: Subregular syntax (finally)

Thomas Graf



Stony Brook University
mail@thomasgraf.net
https://thomasgraf.net

ESSLLI 2021
July 26–30, 2021

Get the slides at:
https://esslli2021.thomasgraf.net

Outline

Path languages once more

- Last time we saw that each tree defines a **path language**, which is a set of strings.
- This way, we can use string-based models for syntax.
- **Disclaimer:** active area of research right now (Graf and Shafiei 2019; Shafiei and Graf 2020)
- Let's look at one more example. . .

Wh-agreement as a TSL path constraint

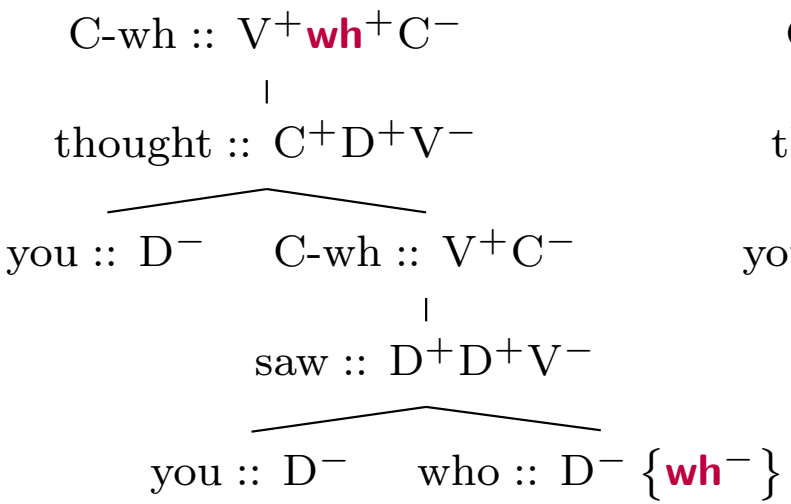
- In Irish (and Chamorro, Hausa, . . .), complementizers change their form if a wh-phrase moves across them.

(1) Cé a^{L}/*go mheas tú **a^{L}/*go** chonaic tú?
who C-wh/C thought you C-wh/C saw you
'Who did you think that you saw?'

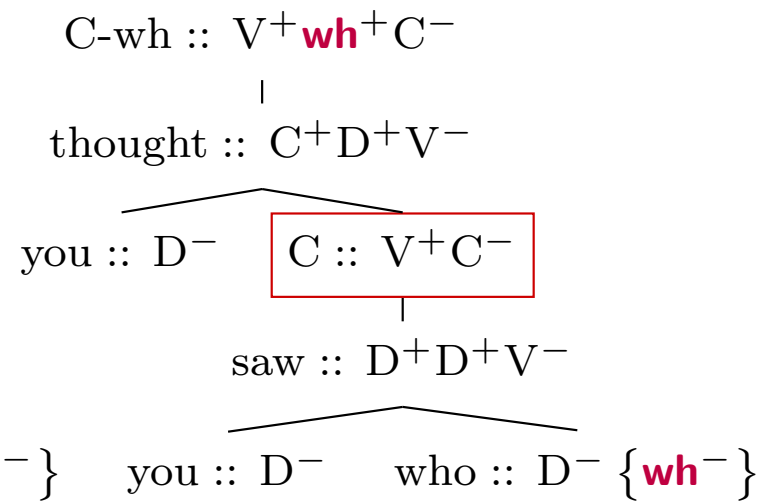
- This can be modeled as a TSL condition on paths.
- It's actually very similar to **sibilant harmony**.

TSL analysis

Good

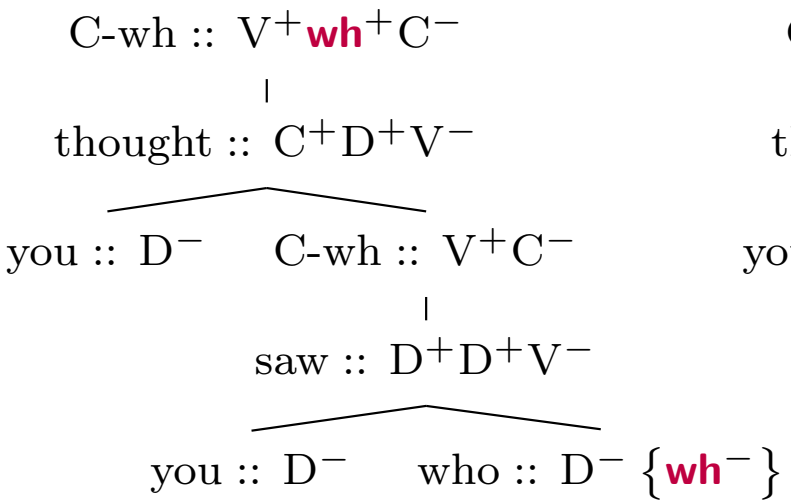


Bad

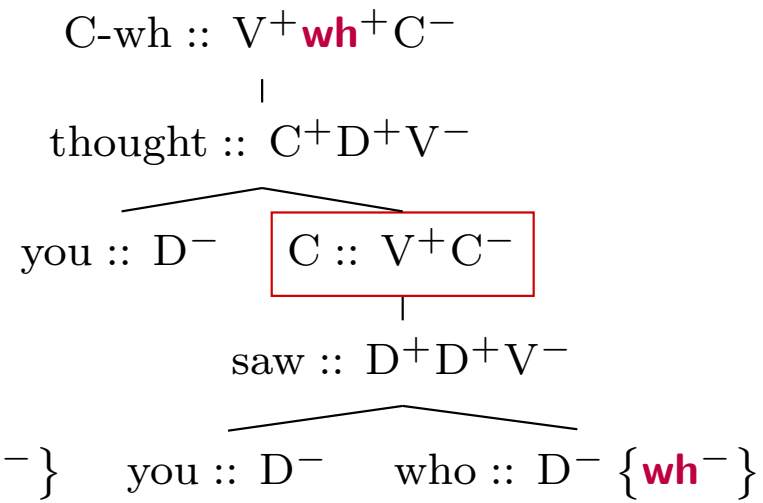


TSL analysis

Good



Bad



⋊ who[wh−] C-wh C-wh[wh+] ⋉
⋊ who[wh−] saw C-wh thought C-wh[wh+] ⋉

*wh− C, C-wh C

⋊ who[wh−] C C-wh[wh+] ⋉
⋊ who[wh−] saw C thought C-wh[wh+] ⋉

Outline

Head-argument relations are SL-2

- **Selection is SL-2 over trees** because we only need to look at

1. the selecting head and
2. its daughters

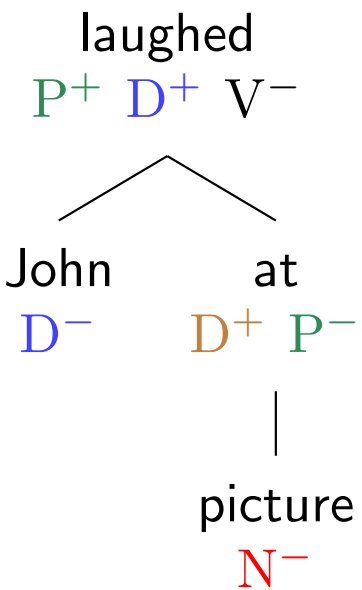
- This means selection is maximally simple!

Head-argument relations are SL-2

- **Selection is SL-2 over trees** because we only need to look at

1. the selecting head and
2. its daughters

- This means selection is maximally simple!

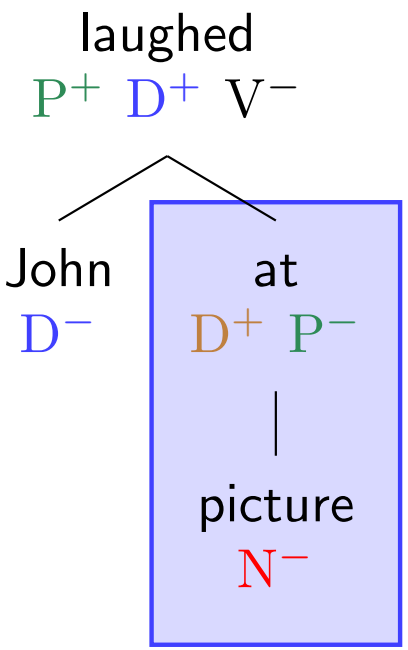


Head-argument relations are SL-2

- **Selection is SL-2 over trees** because we only need to look at

1. the selecting head and
2. its daughters

- This means selection is maximally simple!

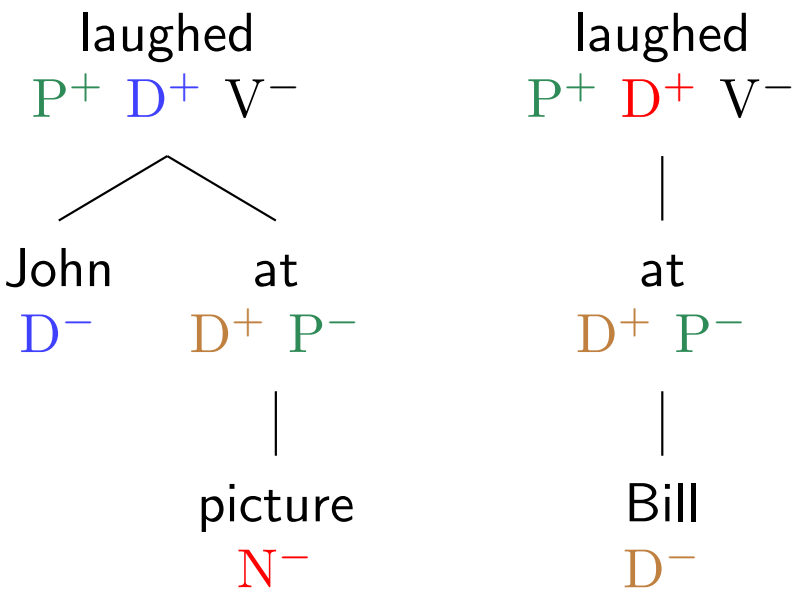


Head-argument relations are SL-2

- **Selection is SL-2 over trees** because we only need to look at

1. the selecting head and
2. its daughters

- This means selection is maximally simple!

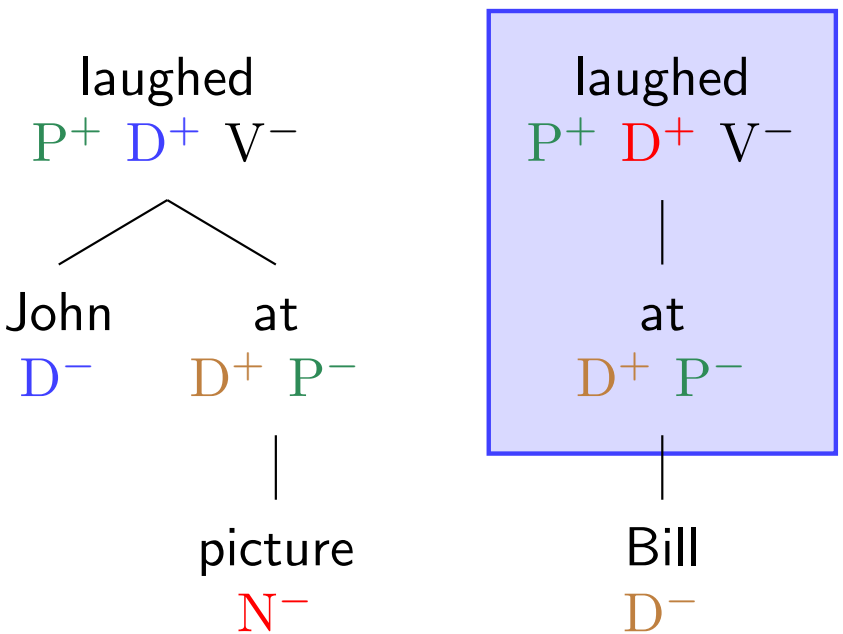


Head-argument relations are SL-2

- **Selection is SL-2 over trees** because we only need to look at

1. the selecting head and
2. its daughters

- This means selection is maximally simple!

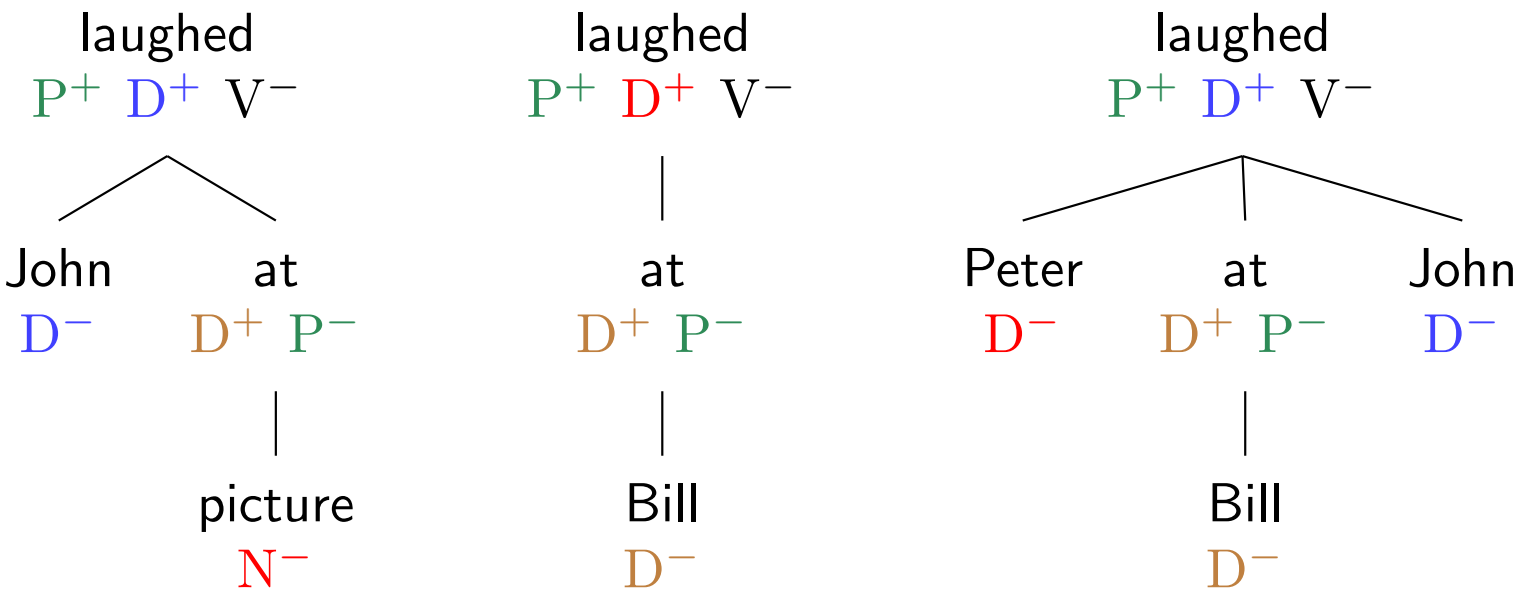


Head-argument relations are SL-2

- **Selection is SL-2 over trees** because we only need to look at

1. the selecting head and
2. its daughters

- This means selection is maximally simple!

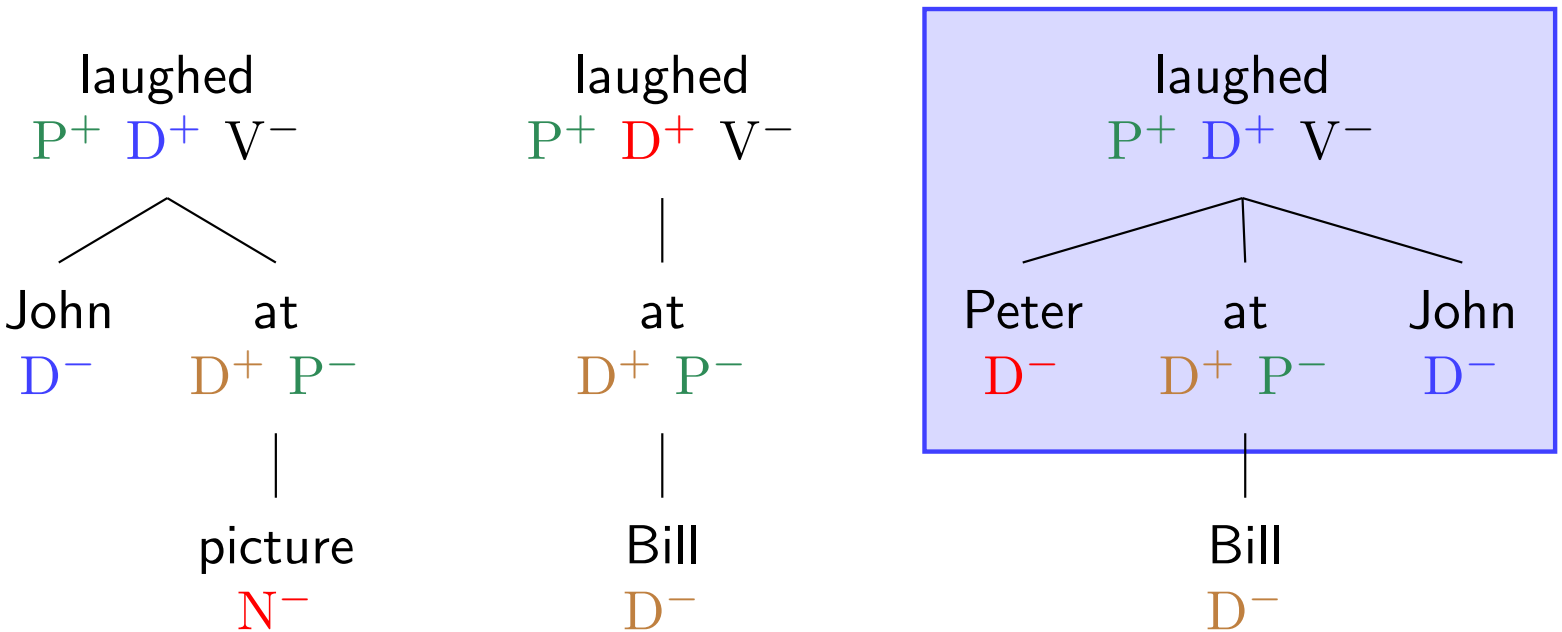


Head-argument relations are SL-2

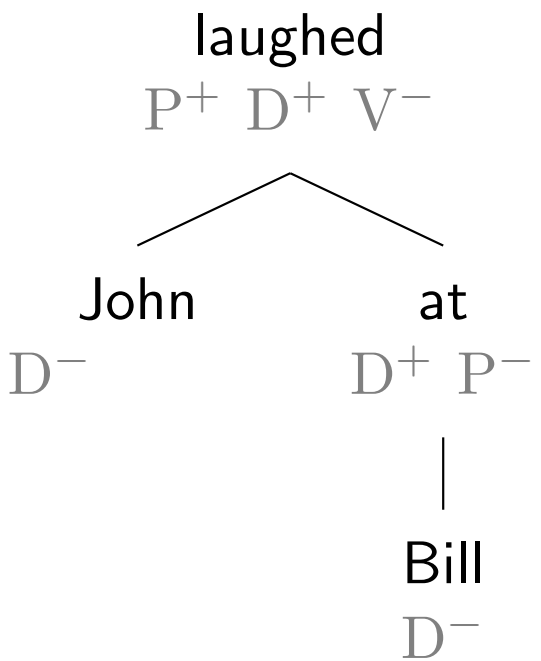
- **Selection is SL-2 over trees** because we only need to look at

1. the selecting head and
2. its daughters

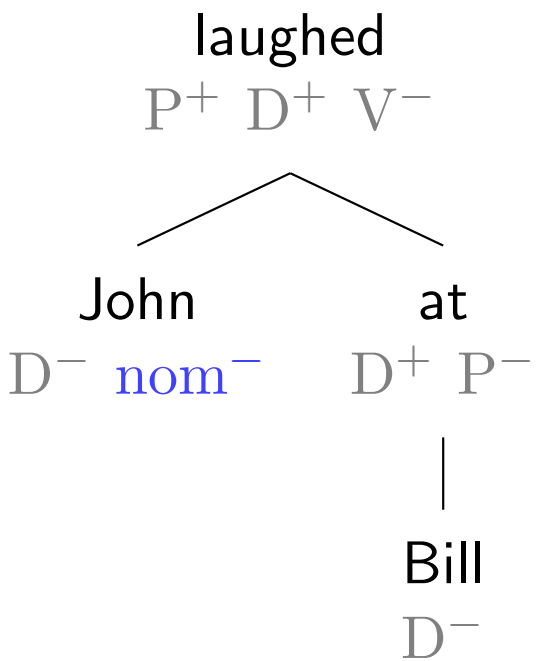
- This means selection is maximally simple!



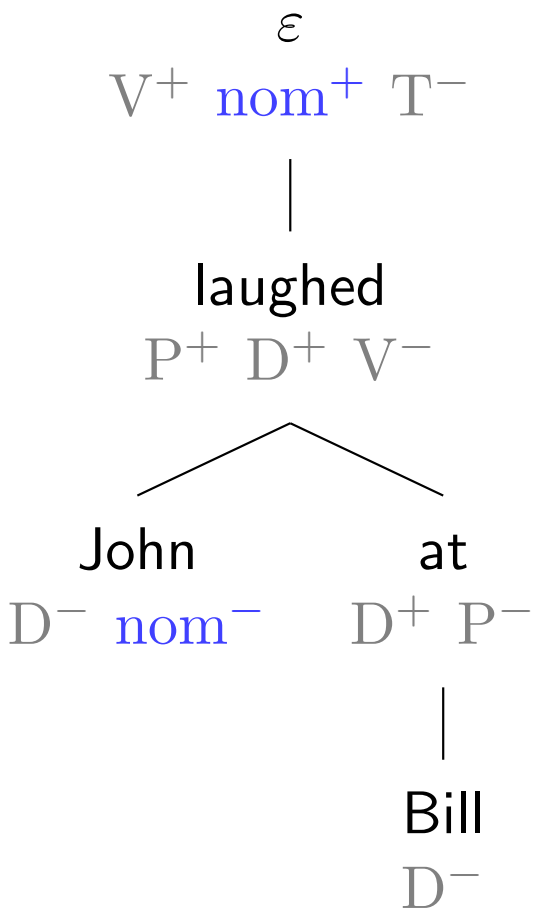
Movement is local over tree tiers



Movement is local over tree tiers

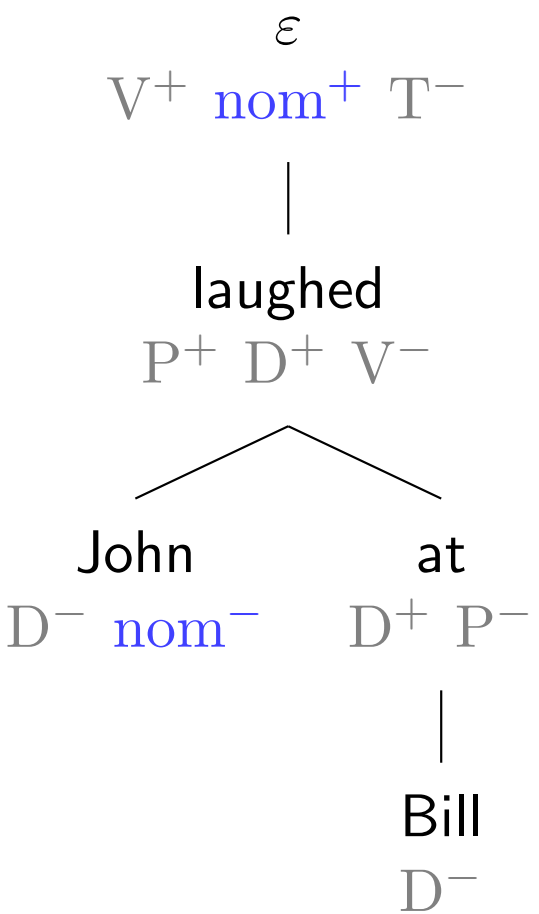


Movement is local over tree tiers

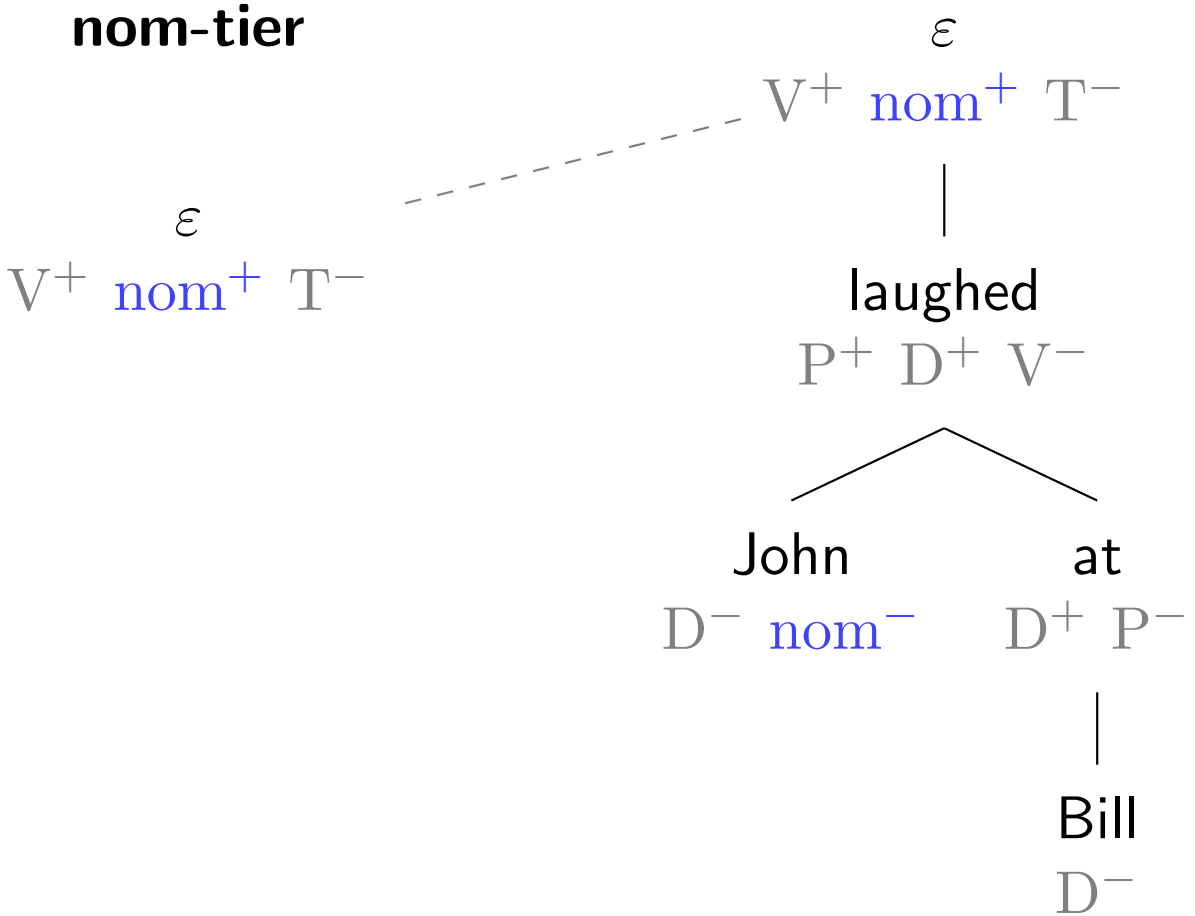


Movement is local over tree tiers

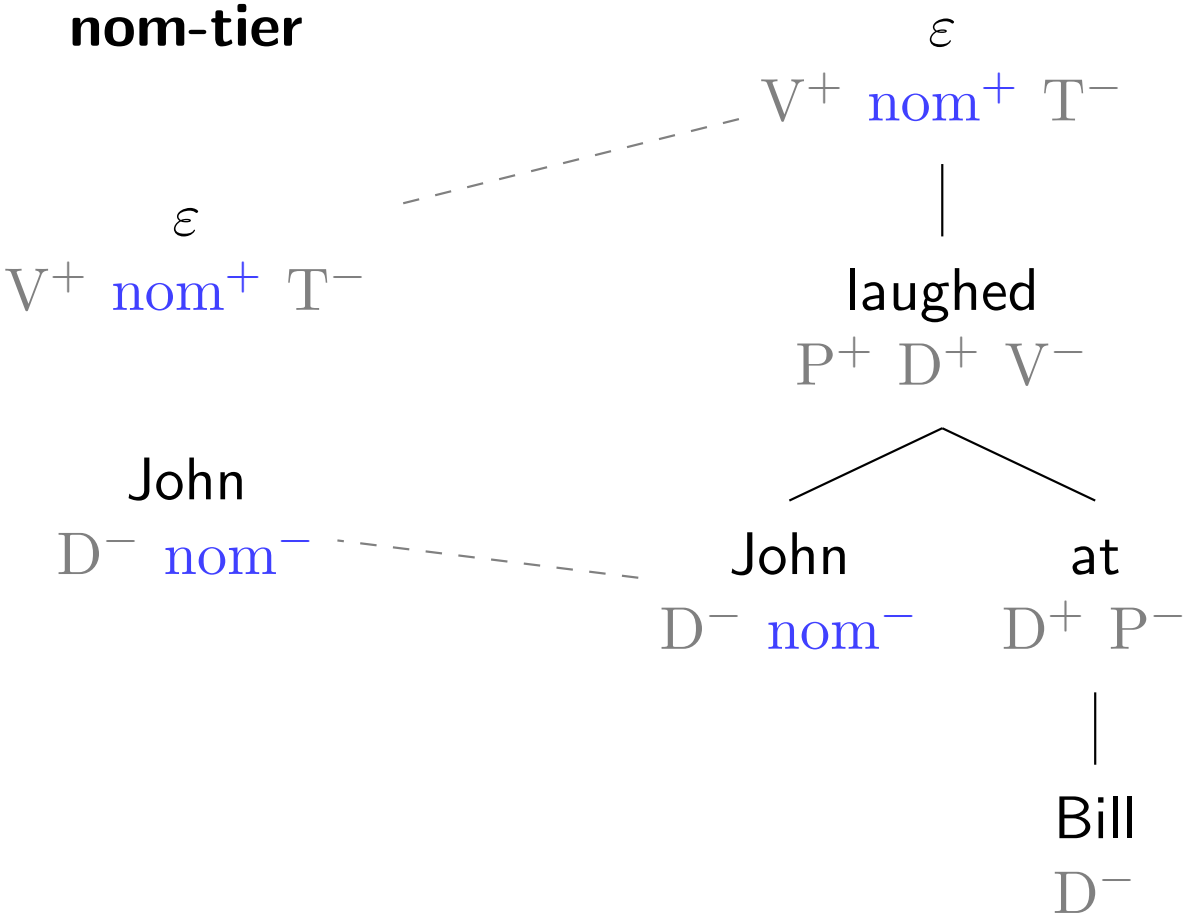
nom-tier



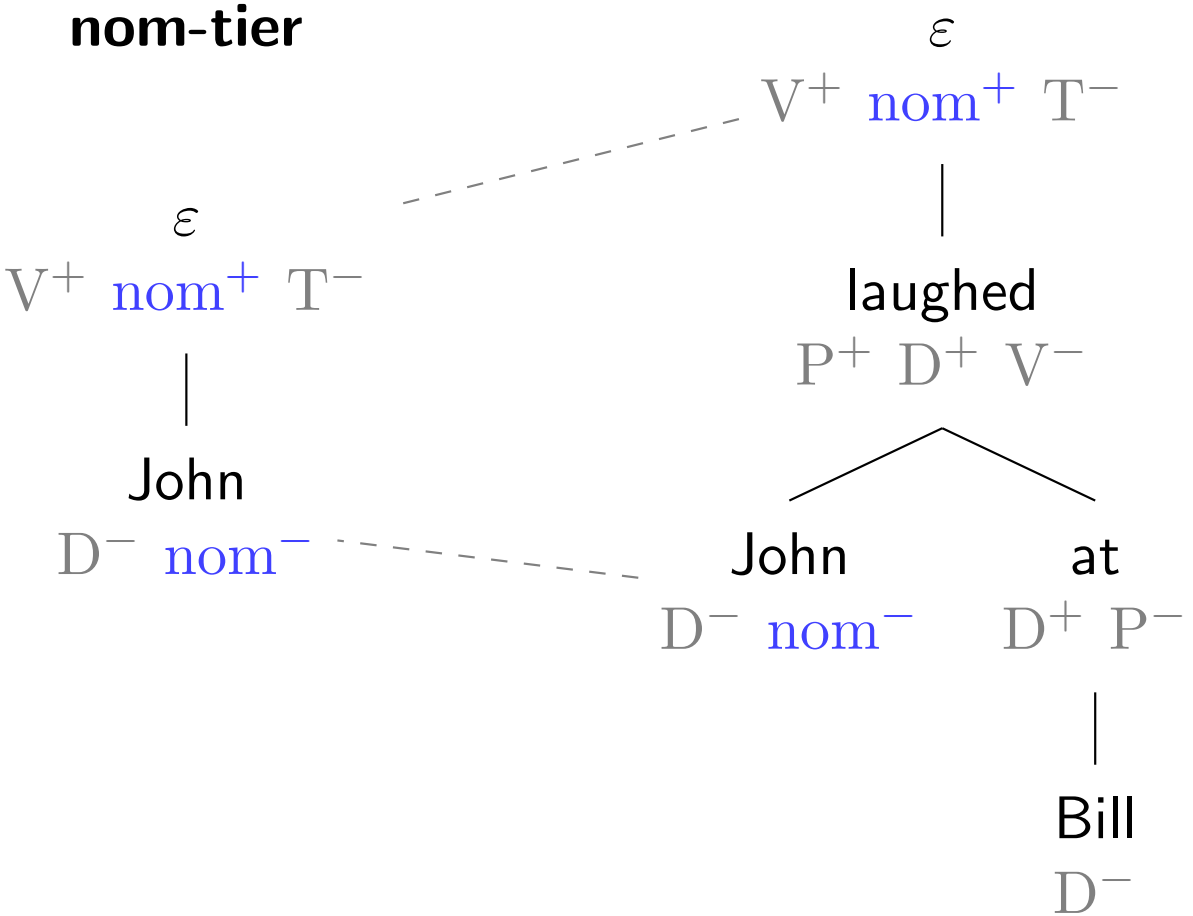
Movement is local over tree tiers



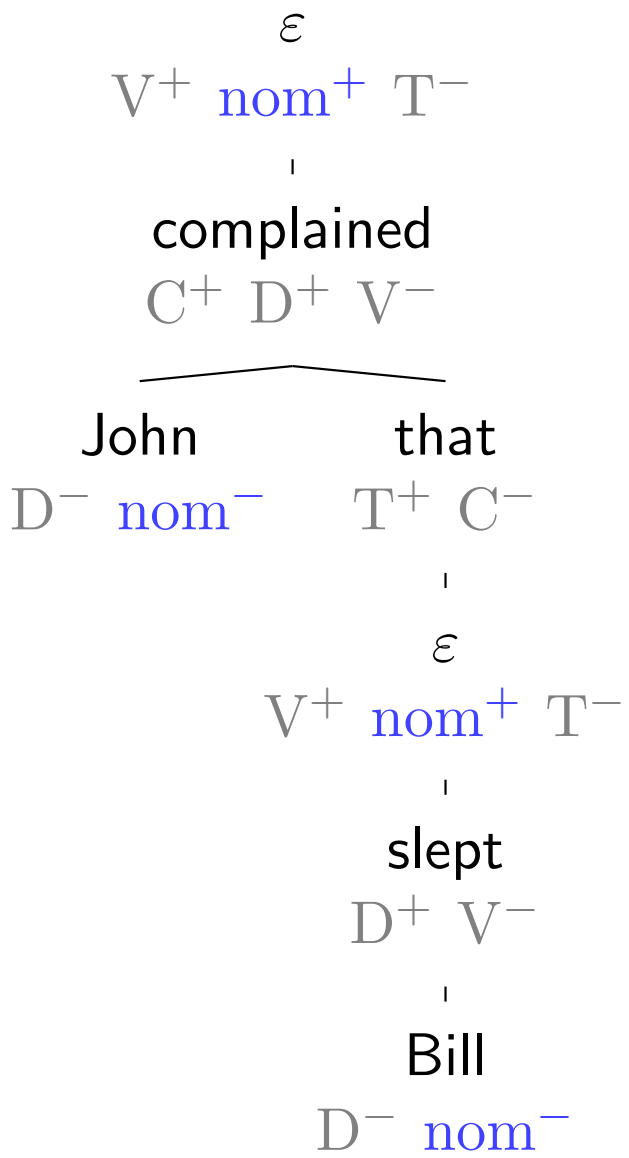
Movement is local over tree tiers



Movement is local over tree tiers

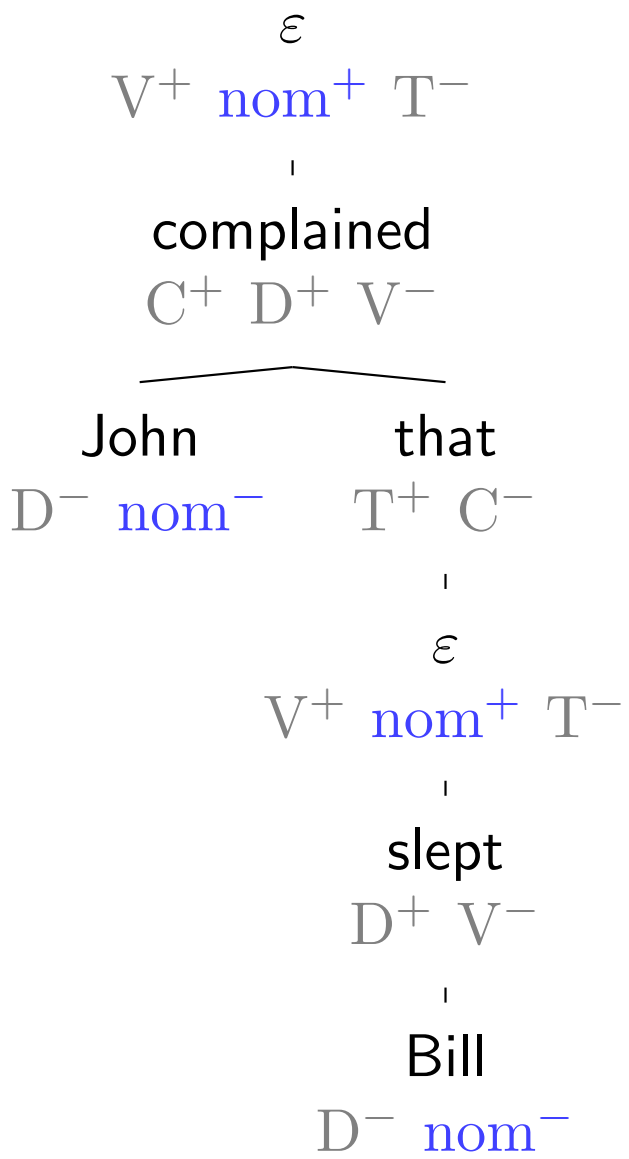


Tier with multiple movers



Tier with multiple movers

nom-tier

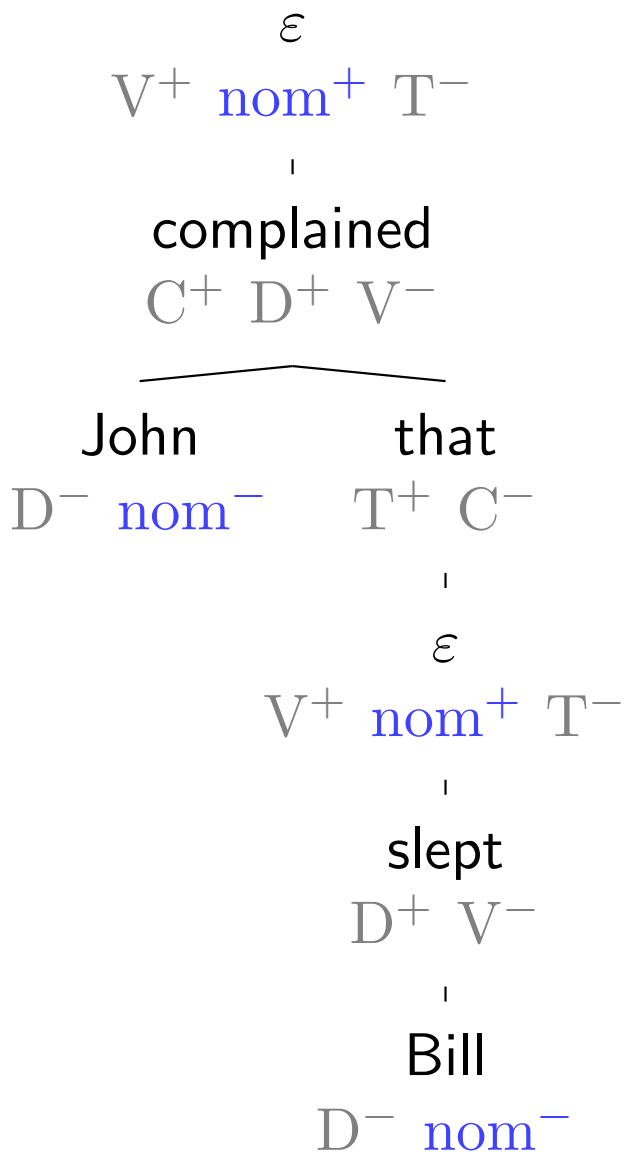


Tier with multiple movers

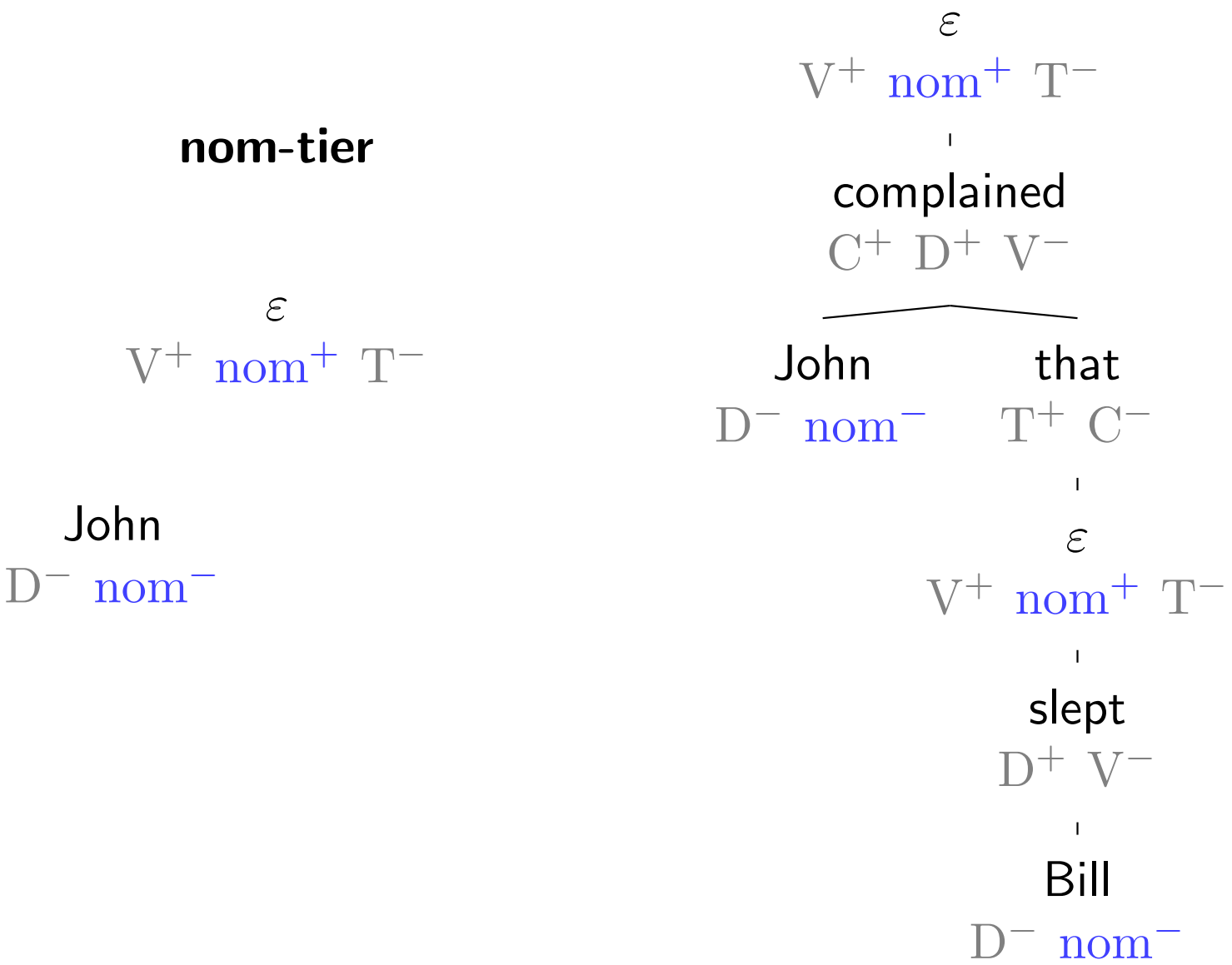
nom-tier

ε

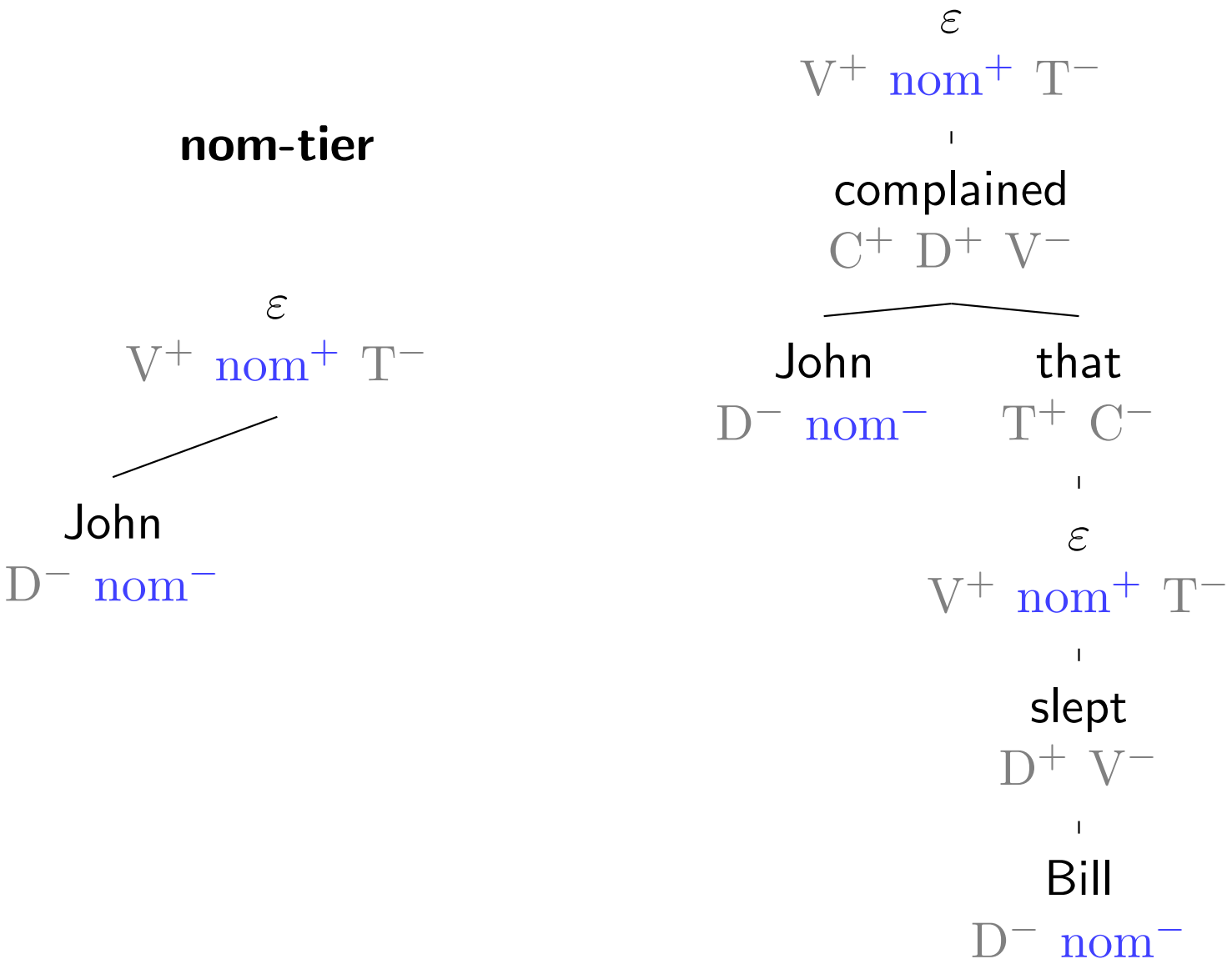
V^+ nom^+ T^-



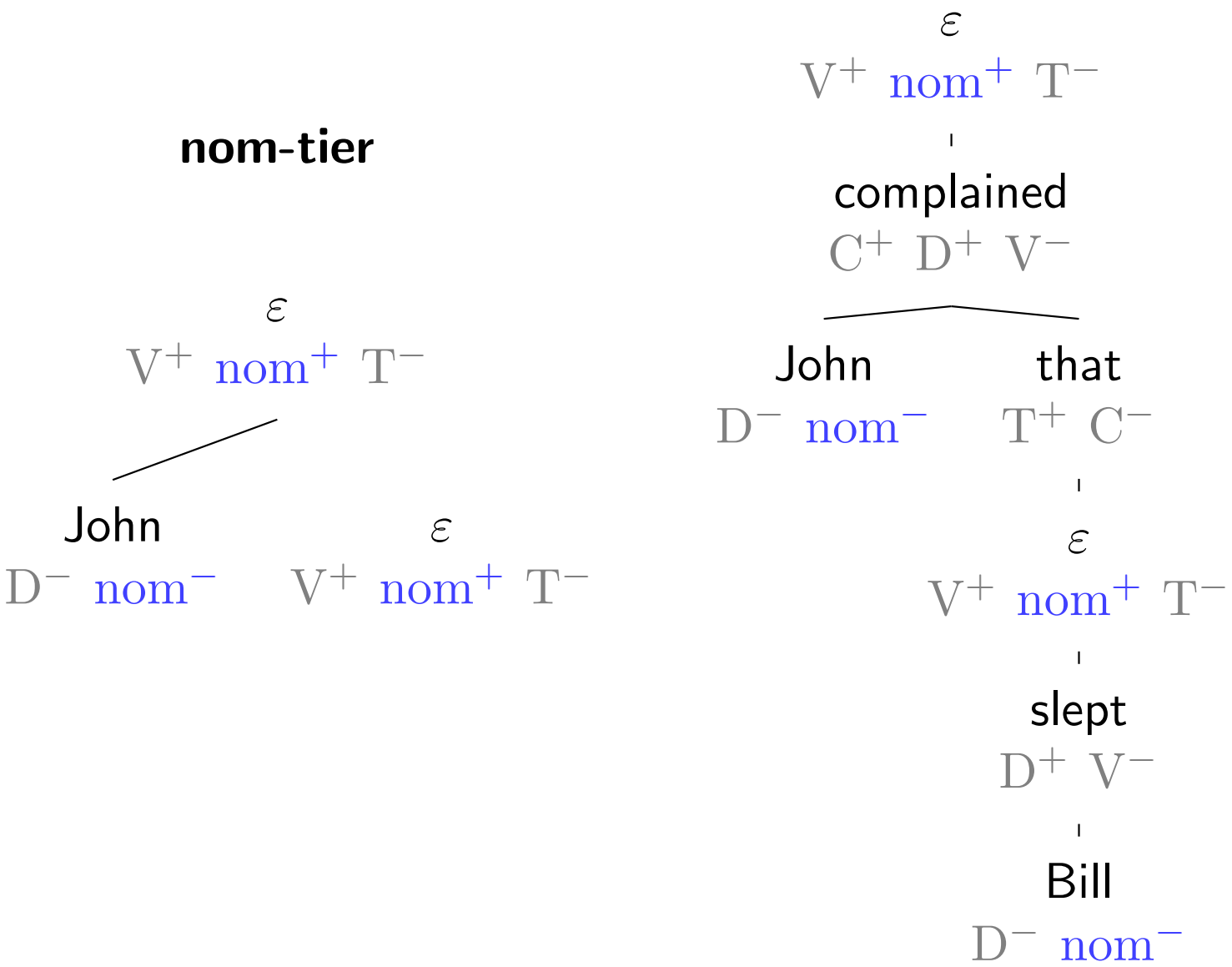
Tier with multiple movers



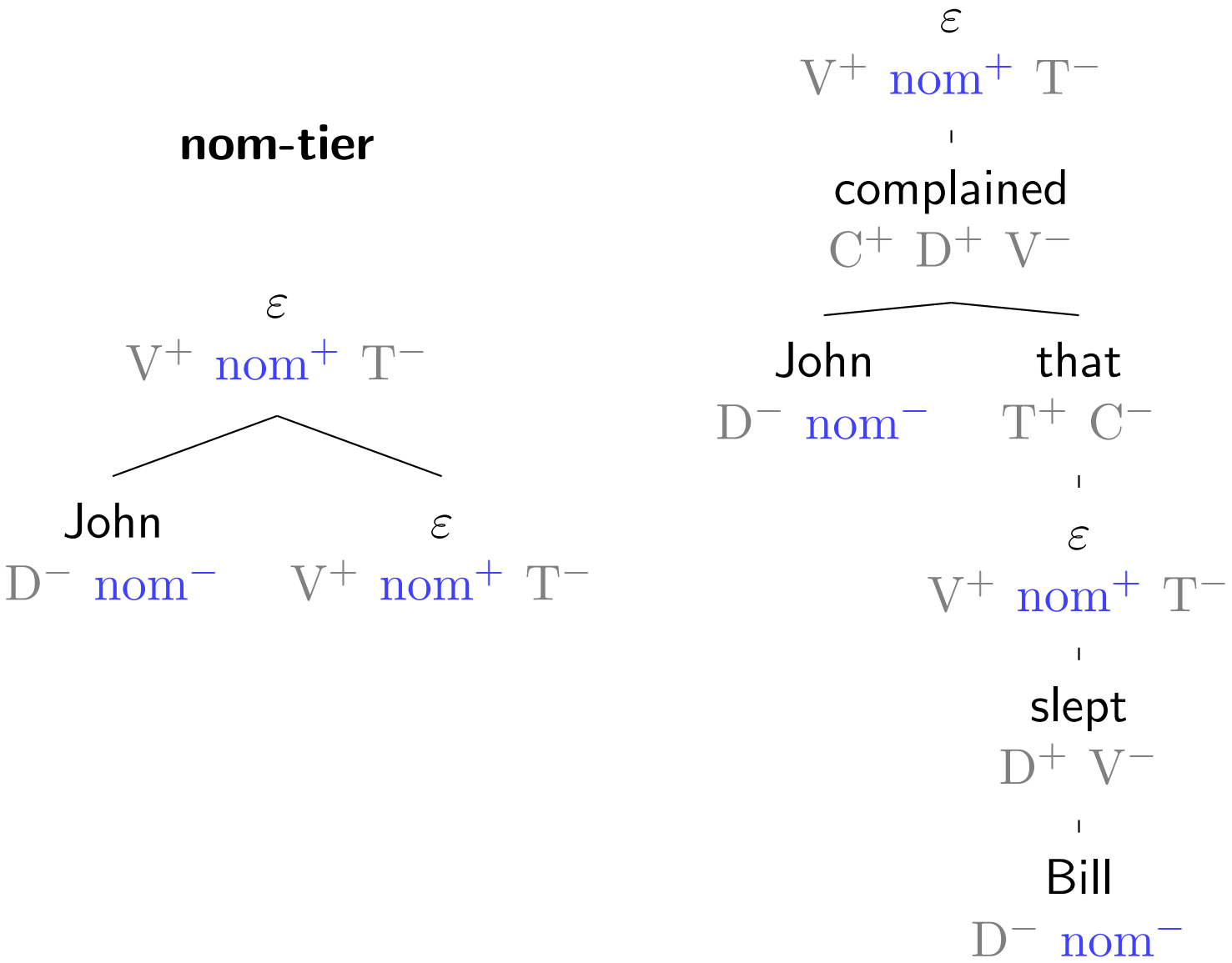
Tier with multiple movers



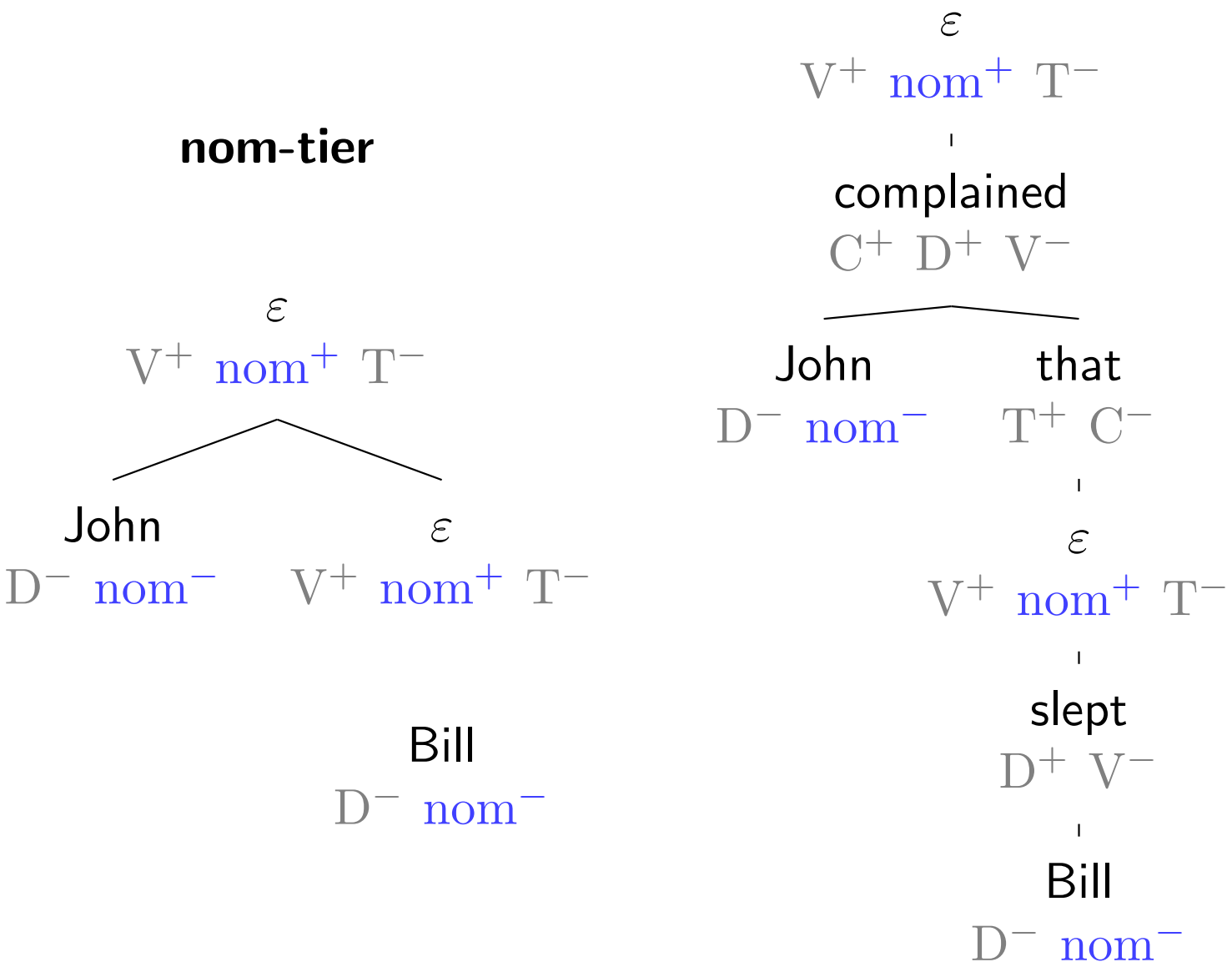
Tier with multiple movers



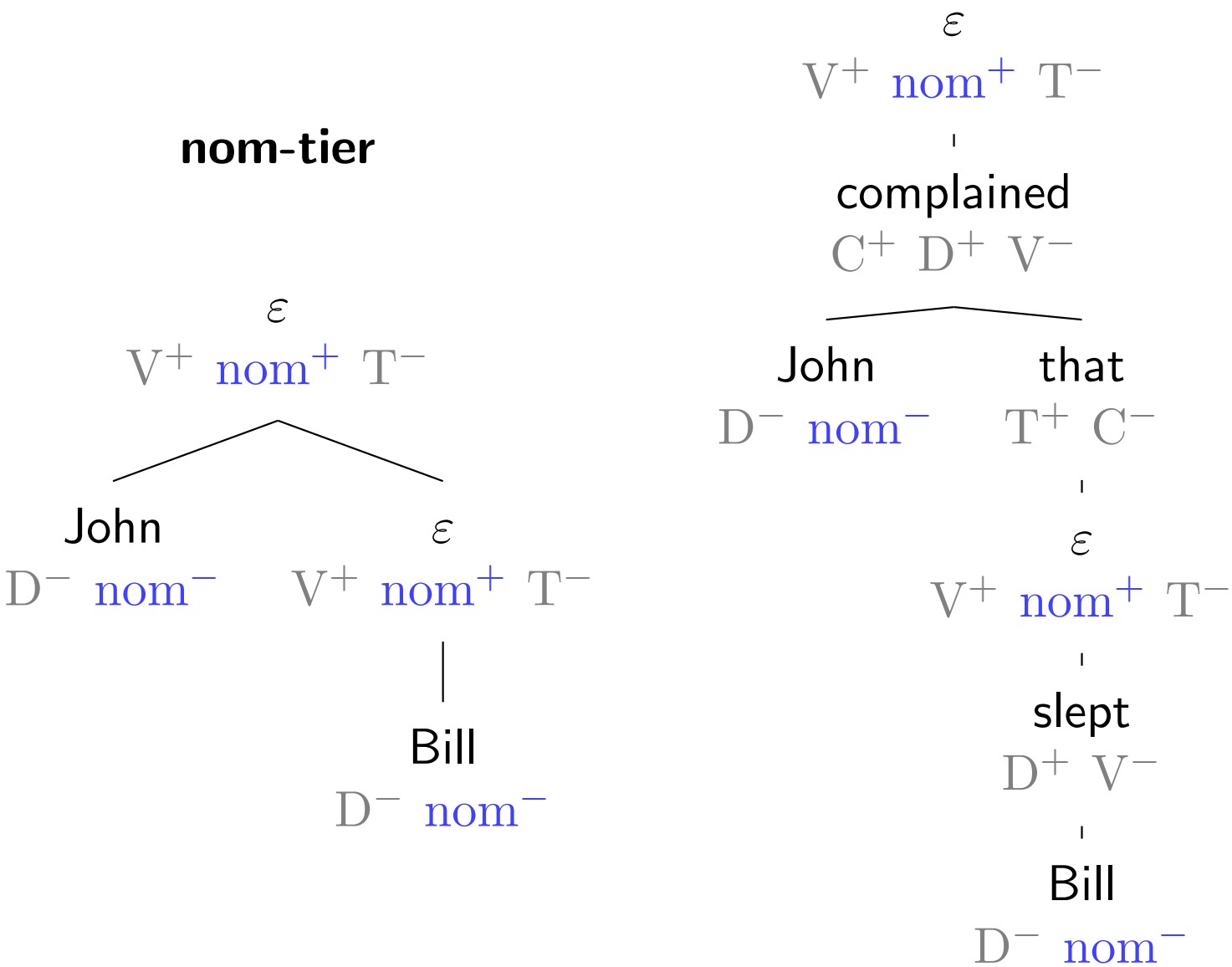
Tier with multiple movers



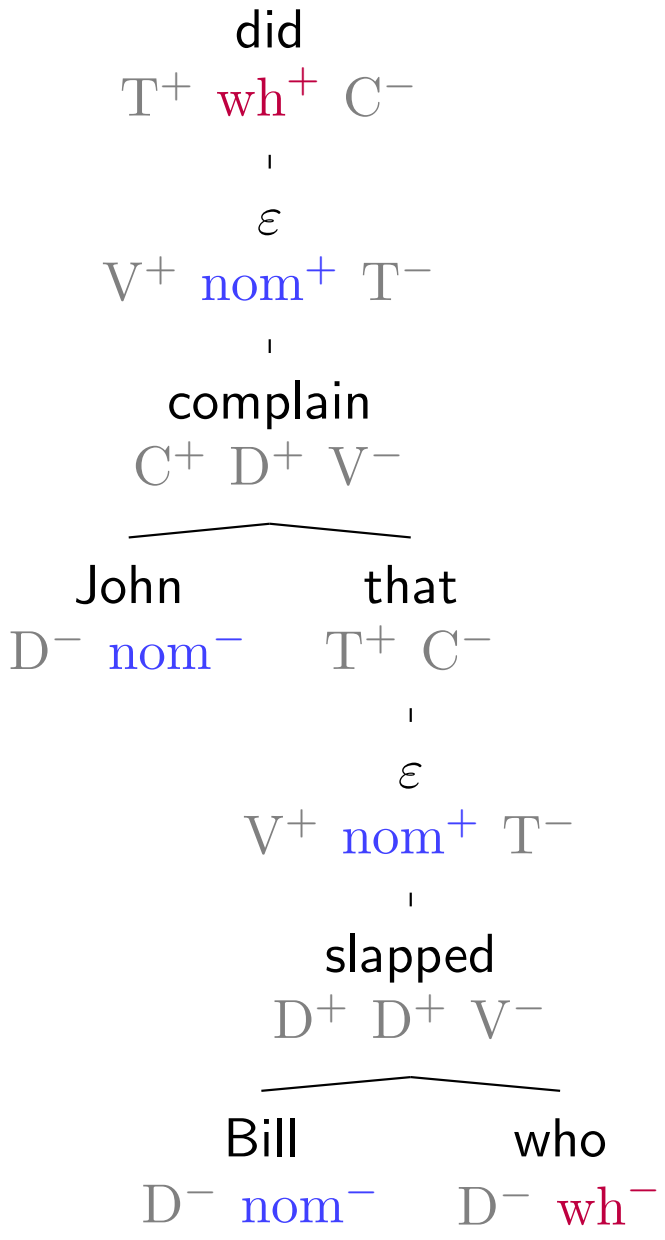
Tier with multiple movers



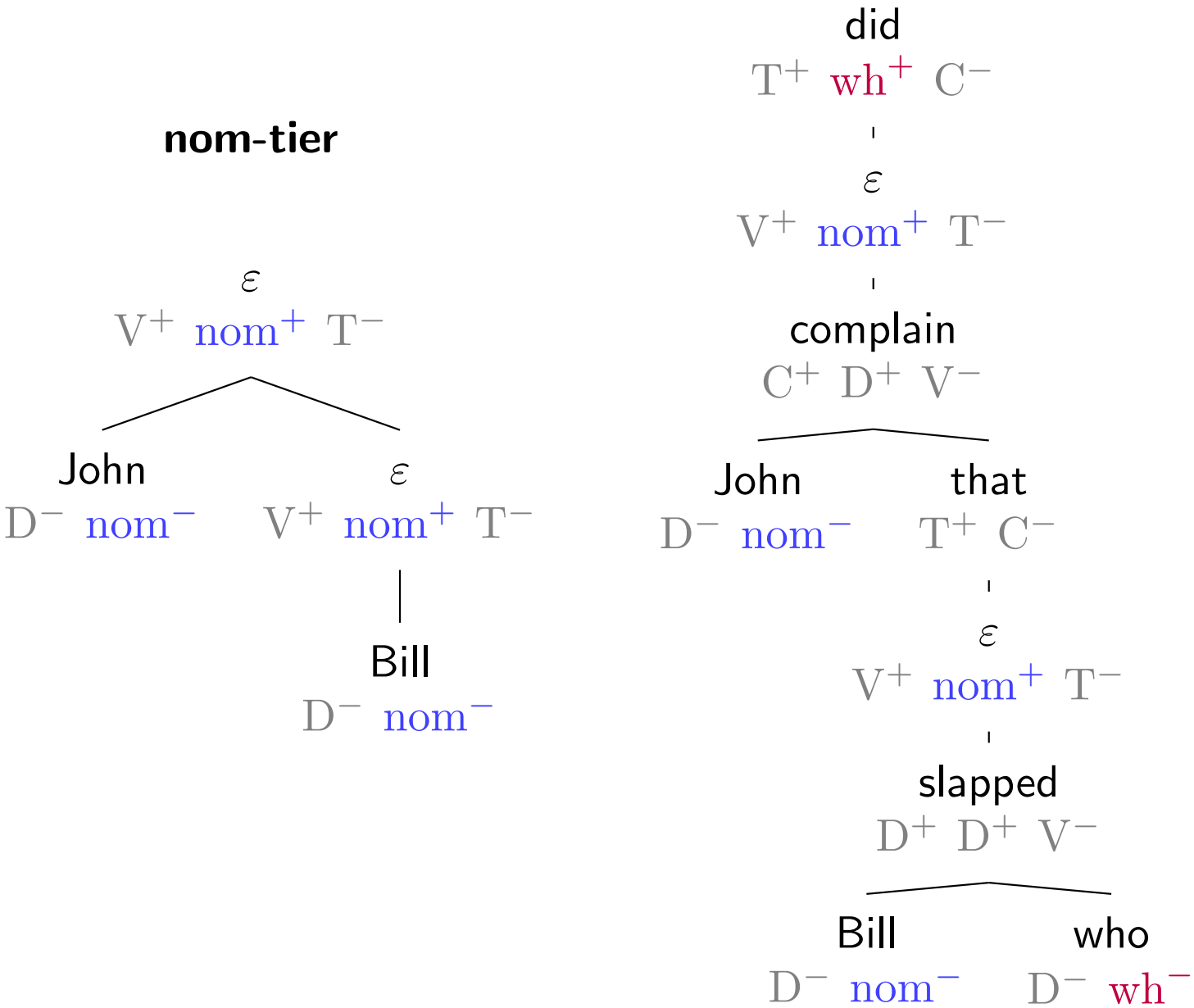
Tier with multiple movers



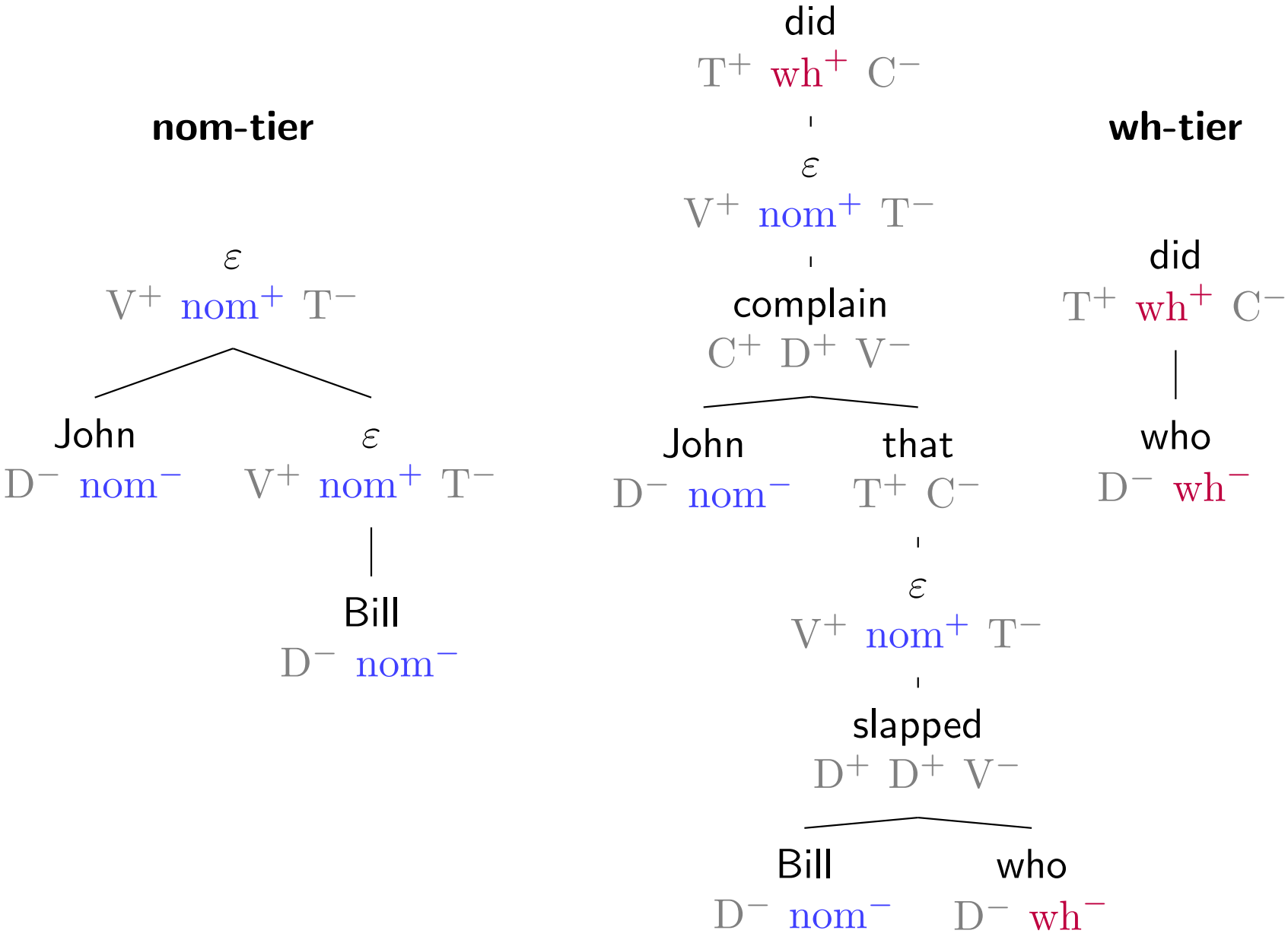
Separate tier for each movement type



Separate tier for each movement type



Separate tier for each movement type



Move is TSL-2

- We now know how to construct movement tiers.
- Licit movement only creates tiers of a specific shape.
- **Move is TSL-2 over trees**:
  1. Every f^{-} must have an f^{+} mother.
  2. Every f^{+} has exactly one f^{-} among its daughters.

Cognitive parallelism

	Phonology	Syntax
SL	local dependencies	Selection
TSL	non-local dependencies	Move

Move is TSL-2

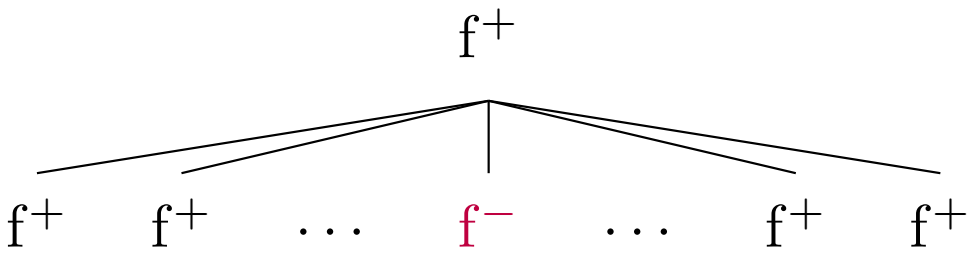
- We now know how to construct movement tiers.
- Licit movement only creates tiers of a specific shape.
- **Move is TSL-2 over trees**:
  1. Every f^- must have an f^+ mother.
  2. Every f^+ has exactly one f^- among its daughters.

Cognitive parallelism

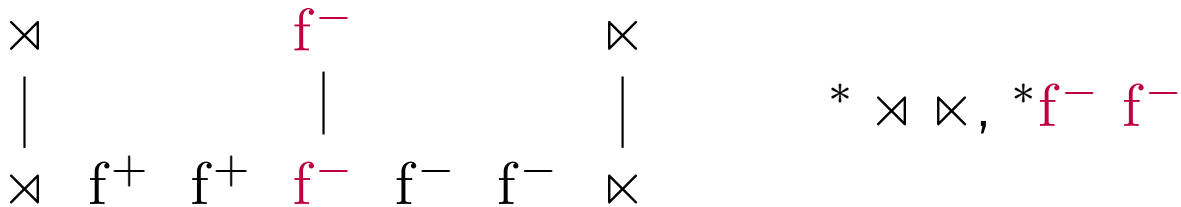
	Phonology	**Syntax**
SL	local dependencies	Selection
TSL	non-local dependencies	Move

Even more parallelism: How to check tier daughters

- Suppose we have the following configuration on a tier:



- How hard is it to verify that there is exactly one f^-?
  It's TSL over strings!



- It's **culminativity all over again!**

Tree SL/TSL are CNL with changed literals

1. ab as an SL string literal:

$$\exists x_1 \exists x_2 [x_1 \triangleleft x_2 \wedge a(x_1) \wedge b(x_2)]$$

2. ab as a TSL string literal:

$$\exists x_1 \exists x_2 [x_1 \triangleleft_T x_2 \wedge a(x_1) \wedge b(x_2)]$$

3. a as an SL tree literal:

$$\begin{array}{l} a \\ | \\ b \end{array}$$

$$\exists x_1 \exists x_2 [x_1 \triangleleft_1 x_2 \wedge a(x_1) \wedge b(x_2) \wedge \neg \exists y [x_1 \triangleleft_2 y]]$$

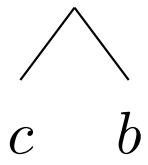
4. a as a TSL tree literal:

$$\begin{array}{l} a \\ | \\ b \end{array}$$

$$\exists x_1 \exists x_2 [x_1 \triangleleft_T x_2 \wedge a(x_1) \wedge b(x_2) \wedge \neg \exists y [\neg (y = x_2) \wedge x_1 \triangleleft_T y]]$$

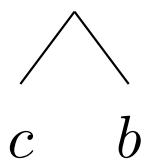
More tree literal examples

SL



$$\exists \mathbf{x}_1 \exists \mathbf{x}_2 \exists \mathbf{x}_3 [\mathbf{x}_1 \lhd_1 \mathbf{x}_2 \wedge \mathbf{x}_1 \lhd_2 \mathbf{x}_3 \wedge a(\mathbf{x}_1) \wedge b(\mathbf{x}_2) \wedge c(\mathbf{x}_3)]$$

TSL



$$\exists \mathbf{x}_1 \exists \mathbf{x}_2 \exists \mathbf{x}_3 [\mathbf{x}_1 \lhd_T \mathbf{x}_2 \wedge \mathbf{x}_1 \lhd_T \mathbf{x}_3 \wedge a(\mathbf{x}_1) \wedge b(\mathbf{x}_2) \wedge c(\mathbf{x}_3) \wedge$$

$$\neg \exists \mathbf{y} [\neg (\mathbf{y} = \mathbf{x}_2 \vee \mathbf{y} = \mathbf{x}_3) \wedge \mathbf{x}_1 \lhd_T \mathbf{y}]]$$

Defining $\lhd_T$

1. **Over strings**

$$\mathbf{x} \lhd_{\mathbf{T}} \mathbf{y} \Leftrightarrow \mathbf{x} \prec \mathbf{y} \wedge \mathbf{T}(\mathbf{x}) \wedge \mathbf{T}(\mathbf{y}) \wedge \neg\exists \mathbf{z}[\mathbf{x} \prec \mathbf{z} \wedge \mathbf{z} \prec \mathbf{y} \wedge \mathbf{T}(\mathbf{z})]$$

2. **Over trees**

$$\mathbf{x} \lhd_{\mathbf{T}} \mathbf{y} \Leftrightarrow \mathbf{x} \lhd^{+} \mathbf{y} \wedge \mathbf{T}(\mathbf{x}) \wedge \mathbf{T}(\mathbf{y}) \wedge \neg\exists \mathbf{z}[\mathbf{x} \lhd^{+} \mathbf{z} \wedge \mathbf{z} \lhd^{+} \mathbf{y} \wedge \mathbf{T}(\mathbf{z})]$$

Note

- If I had thought ahead a bit more, I would have used $\lhd^{+}$ as the symbol for precedence in string land, too.

All the parallelisms

1. Selection $\equiv$ SL-2 phenomena in phonology
   e.g. word-final devoicing
2. Move $\equiv$ TSL-2 phenomena in phonology
   e.g. Samala sibilant harmony
3. 1-to-1 matching of f^+ and f^- $\equiv$ culminativity in phonology
4. SL and TSL literals almost the same over strings and trees, except
   1. $\lhd_1$ and $\lhd_2$ instead of $\lhd$
   2. tree tier literals have to explicitly limit number of daughters
5. Definition of $\lhd_T$ over strings and over trees exactly the same

Islands come for free

Two fundamental questions of syntax

- Why do islands exist?
- Why do island exceptions exist?

A computational argument

1. Movement requires the power of TSL-2.
2. TSL-2 can model islands as blocking effects.
3. The cognitive ability for movement entails the cognitive ability for islands.

And cognitive parallelism again

- islands $\equiv$ Slovenian sibilant harmony

Islands examples

(2) * Which car did John complain [**because** Bill damaged t].

(3) * Which car did John deny [the **fact** that Bill damaged t].

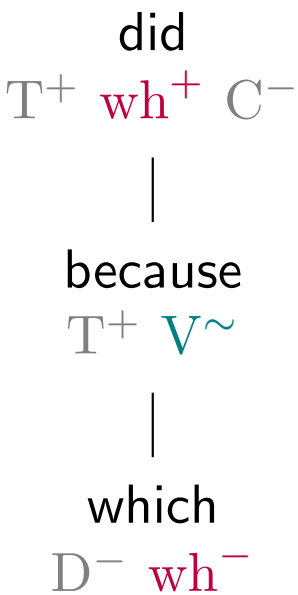
(4) Which car did John drive Mary crazy [**while** trying to fix t].

Islands examples

(2) * Which car did John complain [**because** Bill damaged t].

(3) * Which car did John deny [the **fact** that Bill damaged t].

(4) Which car did John drive Mary crazy [**while** trying to fix t].

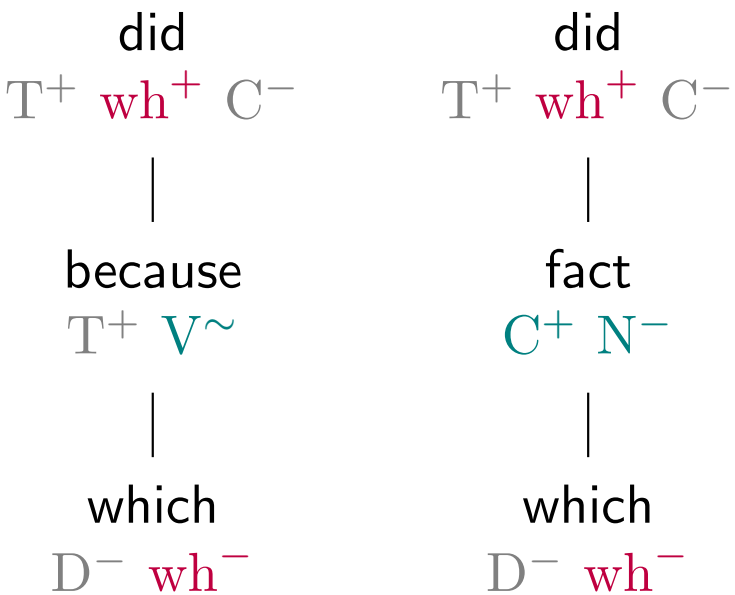


Islands examples

(2) * Which car did John complain [**because** Bill damaged t].

(3) * Which car did John deny [the **fact** that Bill damaged t].

(4) Which car did John drive Mary crazy [**while** trying to fix t].

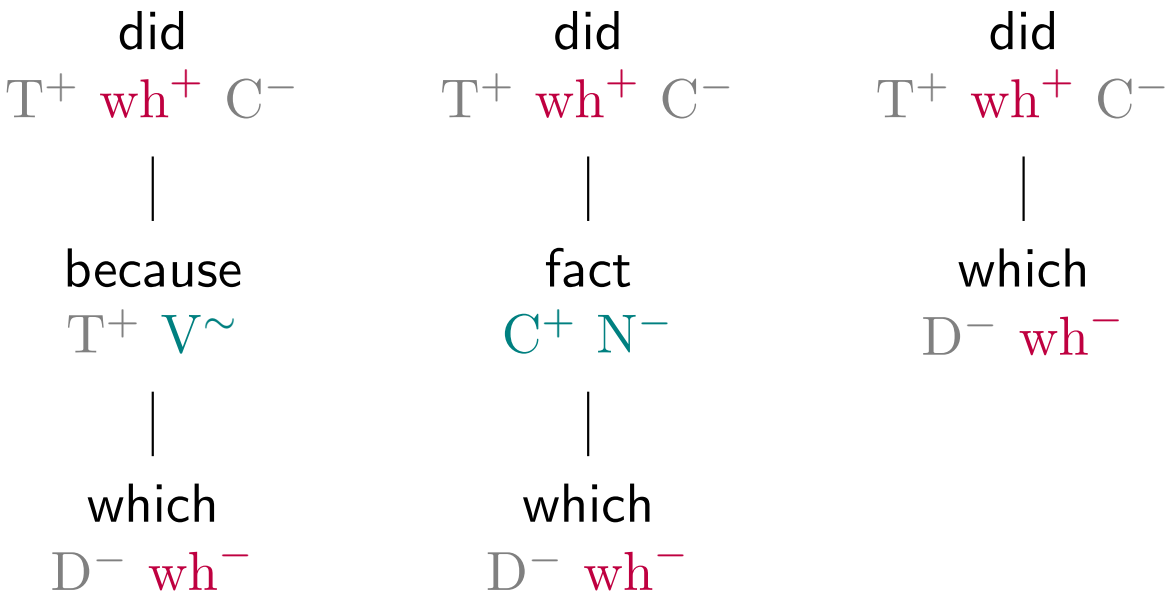


Islands examples

(2) * Which car did John complain [**because** Bill damaged t].

(3) * Which car did John deny [the **fact** that Bill damaged t].

(4) Which car did John drive Mary crazy [**while** trying to fix t].



Impossible islands

- Islands arise when a blocker is projected onto a tier.
- Tier projection only considers lexical item itself, not its structural context
- TSL-2 theory of islands hence rules out:
  - **Gang-up islands**
    "A mover can escape n islands, but not more than that."
  - **Configurational islands**
    "An adjunct is an island iff it is inside an embedded clause."
  - **Cowardly islands**
    "An adjunct is an island iff there are at least two adjuncts in the clause."
  - **Rationed islands**
    "Only one adjunct per clause can be an island."
  - **Discerning islands**
    "Adjuncts only block movers that contain an adjective."

Multiple wh-movement comes for free

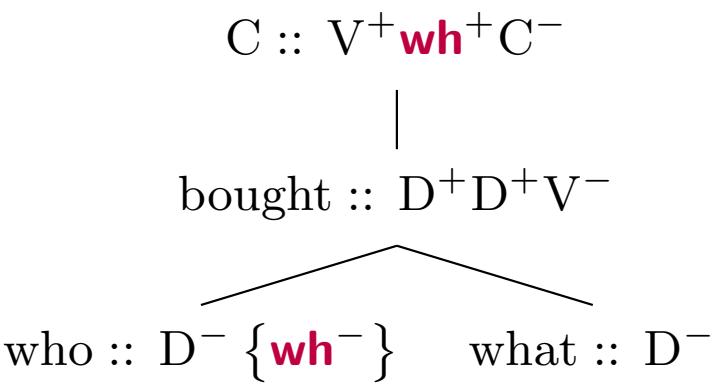
- Some languages (e.g. Bulgarian) allow multiple wh-phrases to move to the front.

1. Who bought what?
2. Who what bought?

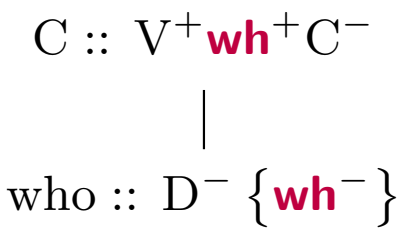
- This is very peculiar given standard syntactic assumptions.
- But from our TSL perspective,
  multiple wh-movement is expected to exist.

Comparing normal to multiple wh-movement

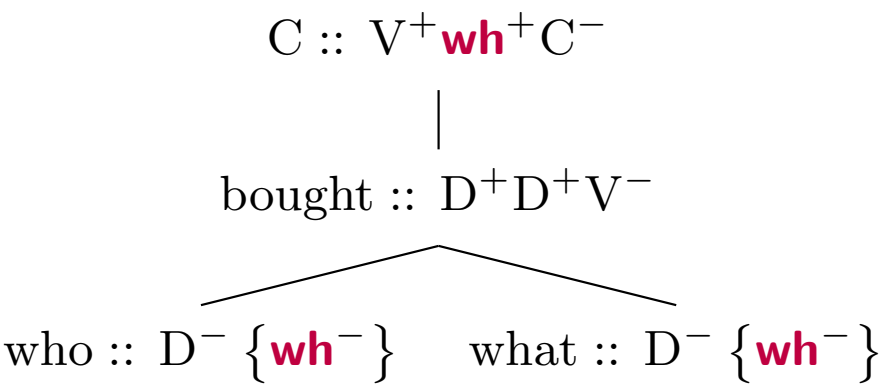
Simple wh-movement



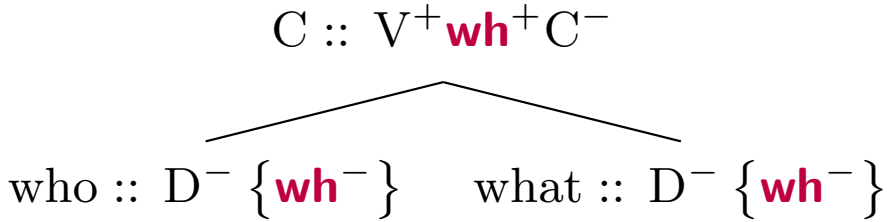
wh-tier



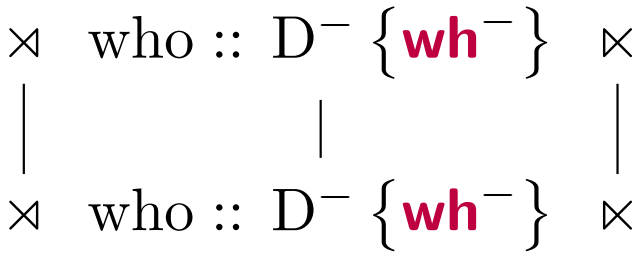
Multiple wh-movement



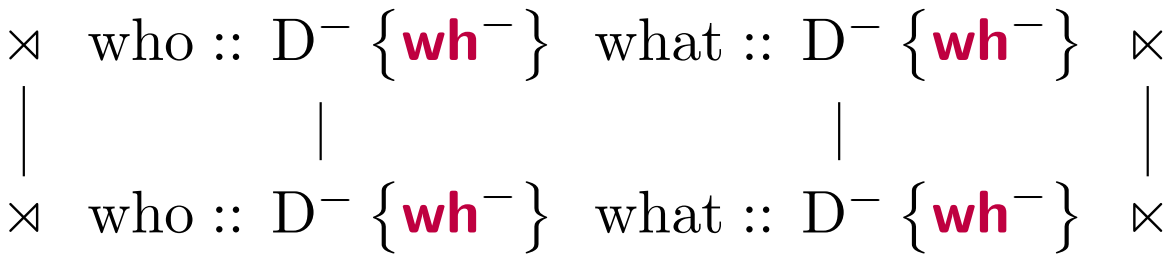
wh-tier



Comparing the wh-tier daughter strings



$* \rtimes \ltimes$, $*\mathrm{wh}^- \ \mathrm{wh}^-$



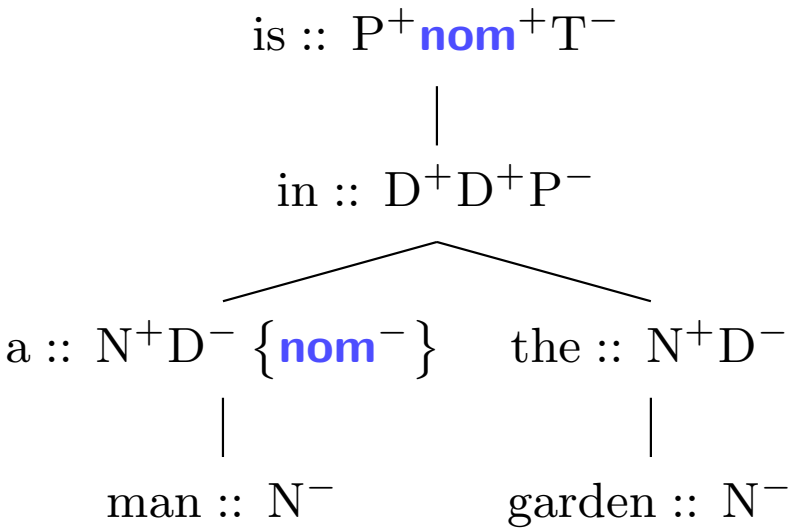
- We can drop $*\mathrm{wh}^- \ \mathrm{wh}^-$ to permit multiple wh-movement.
- Computationally, there is no reason this should never happen.
- Multiple wh-movement is **computationally natural**.

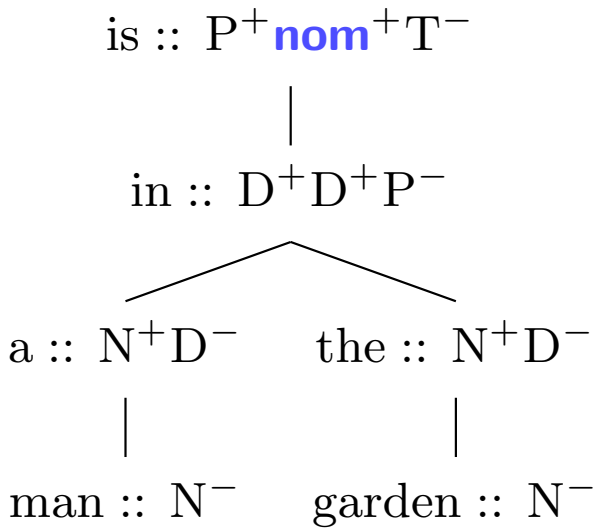
Optional movement comes for free

- But by the same logic, we could have dropped $\rtimes\ltimes$ instead
  $\Rightarrow$ **optional movement**
- This might be what's going on with **expletive constructions**.

1. A man is in the garden.
2. There is a man in the garden.

- Expletive *there* may be the spell-out of an unmatched nom^+.





Summary

- SL and TSL are surprisingly easy to apply to syntax.
  Getting to FO was the hard part. . .
- These classes **capture the core of syntax**

  SL selection
  TSL movement
- The perspective is very natural.
- It immediately derives other aspects of syntax:
  1. existence of islands
  2. non-existence of unattested islands
  3. multiple wh-movement
  4. optional movement

A personal note

- It is amazing how well SL and TSL fit syntax
  even though they were originally designed for phonology.
- I was the first to notice the TSL nature of movement,
  and I work on this on a daily basis, and it still blows my mind.
- Cognitive parallelism may really be on to something:
  1. Phonolgy and syntax do the same things, computationally.
  2. But they do it over different types of structures,
     and that's why they look so different at the surface.

References

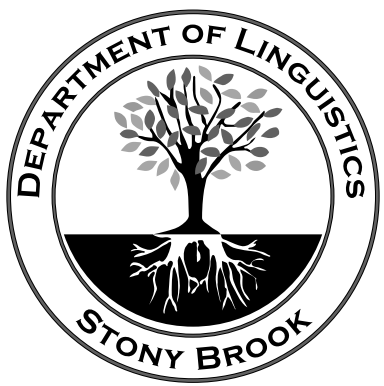
Graf, Thomas, and Nazila Shafiei. 2019. C-command dependencies as TSL string constraints. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2019*, ed. Gaja Jarosz, Max Nelson, Brendan O'Connor, and Joe Pater, 205–215.

Shafiei, Nazila, and Thomas Graf. 2020. The subregular complexity of syntactic islands. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2020*, 272–281.

The Computational Nature of Language

Part 5: Subregular morphosemantics

Thomas Graf



Stony Brook University
mail@thomasgraf.net
https://thomasgraf.net

ESSLLI 2021
July 26–30, 2021

Get the slides at:
https://esslli2021.thomasgraf.net

Outline

Subregular morphosemantics?

The importance of SL and TSL seems to extend
even into the intersection of **semantics and morphology**.

Case Study: Generalized Quantifiers

A generalized quantifier may have a lexically simplex realization only if its quantifier language is TSL.

Generalized quantifiers?

1. every
2. no
3. some
4. not all
5. three
6. all but one
7. an even number
8. a prime number
9. an infinite number

Quantifier languages (van Benthem 1986)

(1) a. Every student cheated.
b. No student cheated.
c. Some student cheated.
d. Three students cheated.

students	John	Mary	Sue
cheated	yes	no	yes
string	Y	N	Y

- (1a): **False**, because the string contains a N
- (1b): **False**, because the string contains a Y
- (1c): **True**, because the string contains a Y
- (1d): **False**, because the string does not contain three Ys

Quantifier languages (van Benthem 1986)

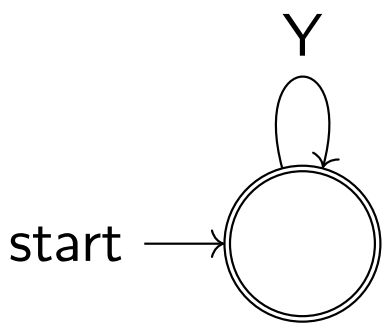
(1) a. Every student cheated.
b. No student cheated.
c. Some student cheated.
d. Three students cheated.

students	John	Mary	Sue
cheated	yes	no	yes
string	Y	N	Y

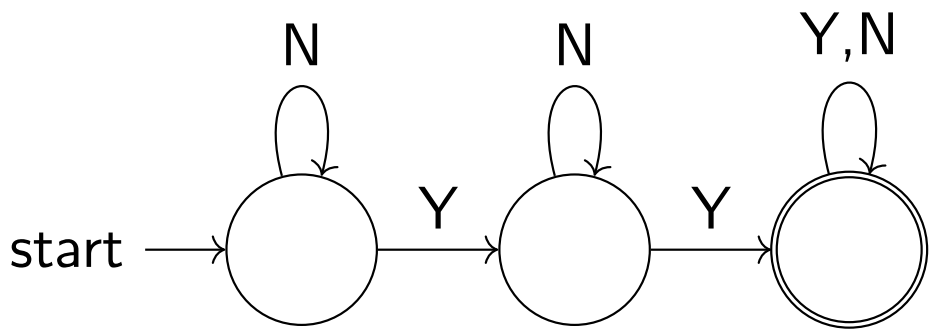
- (1a): **False**, because the string contains a N
- (1b): **False**, because the string contains a Y
- (1c): **True**, because the string contains a Y
- (1d): **False**, because the string does not contain three Ys

Semantic automata

- van Benthem (1986) uses these string languages to equate (some) quantifiers with **finite-state automata** (FSA).
- **General idea:** the FSA for quantifier **Q** only generates those strings over Y and N that describe a situation where **Q** gives us a true statement.



every



at least two

- A language can be generated by an FSA iff it is **regular**.

Could they be subregular?

- If some of these languages are regular, then they might actually be subregular.
- In fact, many of the string languages are easy to define in FO:

every	$\forall x[\mathsf{Y}(x)]$
no	$\forall x[\mathsf{N}(x)]$
some	$\exists x[\mathsf{Y}(x)]$
not all	$\exists x[\mathsf{N}(x)]$
at least two	$\exists x \exists y[\neg(x = y) \wedge \mathsf{Y}(x) \wedge \mathsf{Y}(y)]$
at most one	$\neg\exists x \exists y[\neg(x = y) \wedge \mathsf{Y}(x) \wedge \mathsf{Y}(y)]$
all but one	$\exists x[\mathsf{N}(x) \wedge \forall y[\neg(x = y) \rightarrow \mathsf{Y}(y)]]$

- Some of these string languages are clearly SL:

every	$^*0 \Rightarrow$ SL-1
no	$^*1 \Rightarrow$ SL-1

- Some others are TSL...

TSL descriptions for quantifier languages

Quantifier	Constraint	Literals	Tier
every	$\lvert \mathsf{N} \rvert = 0$	$^*\mathsf{N}$	none
no	$\lvert \mathsf{Y} \rvert = 0$	$^*\mathsf{Y}$	none
some	$\lvert \mathsf{Y} \rvert \geq 1$	$^*\$\$$	Y
at least **n**	$\lvert \mathsf{Y} \rvert \geq \mathbf{n}$	$^*\$1^{\mathbf{m}}\$$ ($\mathbf{m} < \mathbf{n}$)	Y
at most **n**	$\lvert \mathsf{Y} \rvert \leq \mathbf{n}$	$^*\mathsf{Y}^{\mathbf{n}+1}$	Y

Example

some	$^*\$\$$	**True**
at least 2	$^*\$\$$, $^*\$\mathsf{Y}\$$	**True**
at least 3	$^*\$\$$, $^*\$\mathsf{Y}\$$, $^*\$\mathsf{YY}\$$	**False**
at most 2	$^*\mathsf{YYY}$	**True**

TSL descriptions for quantifier languages

Quantifier	Constraint	Literals	Tier
every	$\lvert \mathsf{N} \rvert = 0$	$^*\mathsf{N}$	none
no	$\lvert \mathsf{Y} \rvert = 0$	$^*\mathsf{Y}$	none
some	$\lvert \mathsf{Y} \rvert \geq 1$	$^*\$\$$	Y
at least **n**	$\lvert \mathsf{Y} \rvert \geq \mathbf{n}$	$^*\$1^{\mathbf{m}}\$$ ($\mathbf{m} < \mathbf{n}$)	Y
at most **n**	$\lvert \mathsf{Y} \rvert \leq \mathbf{n}$	$^*\mathsf{Y}^{\mathbf{n}+1}$	Y

Example

$ Y Y $
| | | | |
$ Y N Y $

some	$^*\$\$$	**True**
at least 2	$^*\$\$$, $^*\$\mathsf{Y}\$$	**True**
at least 3	$^*\$\$$, $^*\$\mathsf{Y}\$$, $^*\$\mathsf{YY}\$$	**False**
at most 2	$^*\mathsf{YYY}$	**True**

TSL descriptions for quantifier languages

Quantifier	Constraint	Literals	Tier
every	$\lvert \mathsf{N} \rvert = 0$	$^*\mathsf{N}$	none
no	$\lvert \mathsf{Y} \rvert = 0$	$^*\mathsf{Y}$	none
some	$\lvert \mathsf{Y} \rvert \geq 1$	$^*\$\$$	Y
at least **n**	$\lvert \mathsf{Y} \rvert \geq \mathbf{n}$	$^*\$1^{\mathbf{m}}\$$ ($\mathbf{m} < \mathbf{n}$)	Y
at most **n**	$\lvert \mathsf{Y} \rvert \leq \mathbf{n}$	$^*\mathsf{Y}^{\mathbf{n}+1}$	Y

Example

$ Y Y $
| | | | |
$ Y N Y $

some	$^*\$\$$	**True**
at least 2	$^*\$\$$, $^*\$\mathsf{Y}\$$	**True**
at least 3	$^*\$\$$, $^*\$\mathsf{Y}\$$, $^*\$\mathsf{YY}\$$	**False**
at most 2	$^*\mathsf{YYY}$	**True**

Overview of quantifier languages

If a quantifier language is **not TSL**,
then its quantifier **cannot be lexically simple** in any language.

Quantifier	**TSL?**	**Tier**	**Simple** (Paperno 2011)
every	yes	none	yes
no	yes	none	yes
some	yes	Y	yes
(at least) two	yes	Y	yes
(at most) two	yes	Y	yes
not all	yes	N	no
all but one	yes	N	no
even number	no		no
prime number	no		no
infinitely many	no		no
most	no		???

Overview of quantifier languages

If a quantifier language is **not TSL**,
then its quantifier **cannot be lexically simple** in any language.

Quantifier	TSL?	Tier	Simple (Paperno 2011)
every	yes	none	yes
no	yes	none	yes
some	yes	Y	yes
(at least) two	yes	Y	yes
(at most) two	yes	Y	yes
not all	yes	N	no
all but one	yes	N	no
even number	no		no
prime number	no		no
infinitely many	no		no
most	no		???

Overview of quantifier languages

If a quantifier language is **not TSL**,
then its quantifier **cannot be lexically simple** in any language.

Quantifier	TSL?	Tier	Simple (Paperno 2011)
every	yes	none	yes
no	yes	none	yes
some	yes	Y	yes
(at least) two	yes	Y	yes
(at most) two	yes	Y	yes
not all	yes	N	no
all but one	yes	N	no
even number	no		no
prime number	no		no
infinitely many	no		no
most	no		???

Overview of quantifier languages

If a quantifier language is **not TSL**,
then its quantifier **cannot be lexically simple** in any language.

Quantifier	TSL?	Tier	Simple (Paperno 2011)
every	yes	none	yes
no	yes	none	yes
some	yes	Y	yes
(at least) two	yes	Y	yes
(at most) two	yes	Y	yes
not all	yes	N	no
all but one	yes	N	no
even number	no		no
prime number	no		no
infinitely many	no		no
most	no		???

The case of *most*

Semantic evidence suggests that "most" might be internally complex and hence **not lexically simplex**. (Hackl 2009)

Quantifier	TSL?	Tier	Simple
every	yes	none	yes
no	yes	none	yes
some	yes	Y	yes
(at least) two	yes	Y	yes
(at most) two	yes	Y	yes
not all	yes	N	no
all but one	yes	N	no
even number	no		no
prime number	no		no
infinitely many	no		no
most	no		no

Why no N on tier?

- Why don't languages have access to both options, projecting Y and projecting N?
- It might be **monotonicity**!

Definition

- Let **A** and **B** be two sets ordered by $\leq_{\mathbf{A}}$ and $\leq_{\mathbf{B}}$, respectively.
- A function **f** from **A** to **B** is **monotonic**ally increasing iff the following holds for all **x** and **y** in **A**:

$$\mathbf{x} \leq_{\mathbf{A}} \mathbf{y} \rightarrow \mathbf{f}(\mathbf{x}) \leq_{\mathbf{B}} \mathbf{f}(\mathbf{y})$$

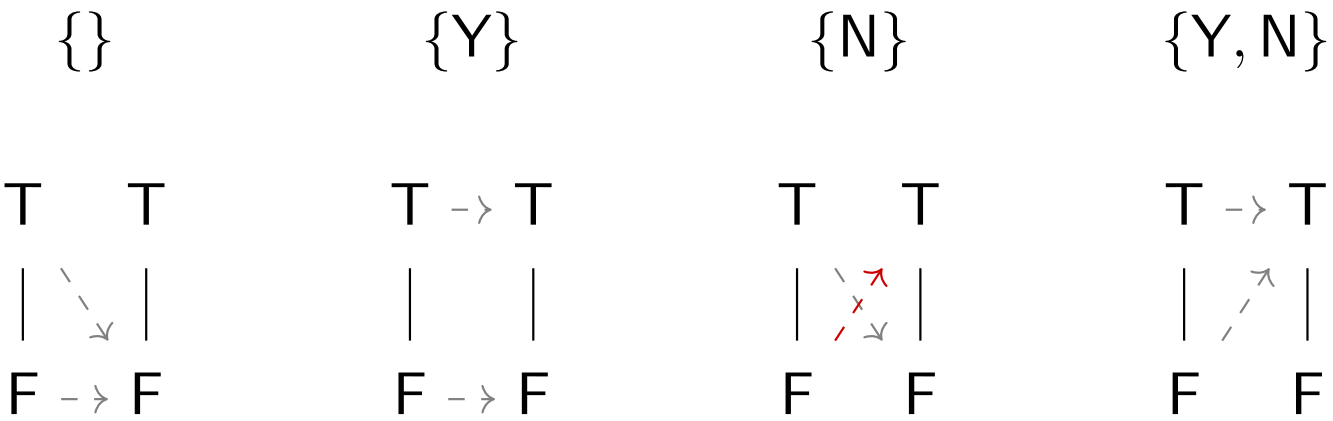
Applying monotonicity to projections

- The choice of tier symbols to project is a function into $\{\text{True}, \text{False}\}$:

Tier symbols	**Project Y?**	**Project N?**
$\{\}$	False	False
$\{\text{Y}\}$	True	False
$\{\text{N}\}$	False	True
$\{\text{Y, N}\}$	True	True

Applying monotonicity to projections

- We may identify Y and N with True and False since they represent set membership.
- This means the choice of tier symbols for a quantifier language is a map from {True, False} into {True, False}.



- The tier projection for **{N} sticks out**:
  it is the only one that is not monotically increasing.

What about larger numerals?

- The largest lexically simplex numerals top out around 12. (*twelve* VS *thirteen*)
- Isn't TSL-12 too big a window?
- In general, yes, but since we only allow Y on the tier, there's at most 13 different literals to store.
- Since the description would still be very short, the increased window size might have marginal cost relative to the real-world payoff.
- Quite generally, though, numerals are a bit weird.

Alternative: An SL typology

- The story so far builds on the semantic automata literature, which assumes that all string orders have to be permitted.

Example

- If YNY is well-formed, then so must be
  1. YYN
  2. NYY

- If we drop this requirement, we can tell **a pure SL-2 story**.

A simple start

- **Start:** *every* and *no* are still SL-1.
  Let's put them aside.
- **For SL-2, suppose:**
  1. Every string must start with Y ($^{*}\rtimes$N).
  2. No "deadends" where the string cannot continue
     Extension condition
  3. We may not put restrictions on the end of the string.
     Because the string may be infinite and thus have no end
- This leaves us with **very few options**.

A remark: from negative to positive

- For the following, it is more intuitive to talk about allowed combinations, not the forbidden ones.
- Since there's only finitely many SL-2 literals over Y and N, we get the forbidden ones by subtracting the allowed ones from the set of all possible literals.

Converting allowed SL-2 to forbidden SL-2

Allowed	**Forbidden**
⋊ ⋉	
⋊ Y	
Y N	
N Y	
N ⋉	

A remark: from negative to positive

- For the following, it is more intuitive to talk about allowed combinations, not the forbidden ones.
- Since there's only finitely many SL-2 literals over Y and N, we get the forbidden ones by subtracting the allowed ones from the set of all possible literals.

Converting allowed SL-2 to forbidden SL-2

Allowed	**Forbidden**
$\rtimes\ltimes$	
$\rtimes$Y	$^*\rtimes$N
Y N	
N Y	
N $\ltimes$	

A remark: from negative to positive

- For the following, it is more intuitive to talk about allowed combinations, not the forbidden ones.
- Since there's only finitely many SL-2 literals over Y and N, we get the forbidden ones by subtracting the allowed ones from the set of all possible literals.

Converting allowed SL-2 to forbidden SL-2

Allowed	**Forbidden**
⋊⋉	
⋊Y	*⋊N
Y N	*YY
N Y	
N ⋉	

A remark: from negative to positive

- For the following, it is more intuitive to talk about allowed combinations, not the forbidden ones.
- Since there's only finitely many SL-2 literals over Y and N, we get the forbidden ones by subtracting the allowed ones from the set of all possible literals.

Converting allowed SL-2 to forbidden SL-2

Allowed	**Forbidden**
⋊⋉	
⋊Y	*⋊N
Y N	*YY
N Y	*NN
N ⋉	

A remark: from negative to positive

- For the following, it is more intuitive to talk about allowed combinations, not the forbidden ones.
- Since there's only finitely many SL-2 literals over Y and N, we get the forbidden ones by subtracting the allowed ones from the set of all possible literals.

Converting allowed SL-2 to forbidden SL-2

Allowed	**Forbidden**
⋊⋉	
⋊Y	*⋊N
Y N	*YY
N Y	*NN
N ⋉	*Y⋉

The possible SL-2 languages (must allow $\rtimes$Y)

Allowed	Language	Quantifier
nothing	none	none
YY	Y^+	every (with existential import)
YN	none	none
NY	none	none
NN	none	none
YY, YN	none	none
YY, NY	Y^+	every (with existential import)
YY, NN	Y^+	every (with existential import)
YN, NY	$(YN)^+$	half
YN, NN	YN^+	(exactly) one
YY, YN, NY	$(Y^+N)^+$	most (= half or more)
YY, YN, NN	Y^+N^*	some
YY, NY, NN	Y^+	every (with existential import)
YN, NY, NN	$(YN^+)^+$	half or less
YY, YN, NY, NN	$Y\{Y,N\}^*$	some

Table with useless options removed

We can remove all options that

1. produce no string sets, or
2. allow literals that cannot occur in the string.

Allowed	**Language**	**Quantifier**
YY	Y^+	every (with existential import)
YN, NY	$(YN)^+$	half
YN, NN	YN^+	(exactly) one
YY, YN, NY	$(Y^+N)^+$	most (= half or more)
YY, YN, NN	Y^+N^*	some
YN, NY, NN	$(YN^+)^+$	half or less
YY, YN, NY, NN	$Y\{Y,N\}^*$	some

Three outliers

- Three quantifiers in this table stand out:
  1. *half* is lexically simple, but not a determiner
  2. *half or less* is never lexically simple
  3. independent arguments that *one* means *at least one*, with *exactly one* reading derived from that.
- We can rule out all of them by stipulating that YY must be allowed.

Final typology

- SL-1 quantifiers: *every* (no existential import), *no*
- SL-2 quantifiers: *every* (with existential import), *some*, *most*

Allowed	**Language**	**Quantifier**
YY	Y^+	every (with existential import)
YY, YN, NY	$(Y^+N)^+$	most (= half or more)
YY, YN, NN	Y^+N^*	some
YY, YN, NY, NN	$Y\{Y,N\}^*$	some

Pragmatics?

- Quantifiers often carry pragmatic *implicatures*:
  - inference from *some* to *not all*
  - *most* feels closer to $\frac{2}{3}$ than $\frac{1}{2}$

A cute idea to toy around with

Pragmatics enforces conditions on how often each literal must occur in the string.

Example

- *some*: $|\mathsf{YN}| \geq 1$
- *most*: $|\mathsf{YY}| \geq |\mathsf{YN}|$

Summary (case study)

- Quantifiers display principled lexical gaps across languages.
- Linguists don't have a good story.
- We have on in terms of subregular complexity.
- Once again, **SL and TSL save the day**.

Summary (general message)

- Subregular complexity can be applied to string or trees in any domain
  (even robot navigation)
- Subregular semantics hinges on our ability to **develop string/tree encodings** for:
  - quantifiers
  - tense
  - aspect
  - ⋮

Course summary: the what

subregular weaker than regular

- SL and TSL (and perhaps ITSL) as restrictive notions of computational complexity
- formalization in terms of FO fragment:
  conjunction of negative literals
  many alternatives exist, literature favors FSA definition
- varying the literals gives us
  1. string SL
  2. string TSL
  3. tree SL
  4. tree TSL
- surprising parallels across subdomains of language
  wh-agreement $\equiv$ Slovenian sibilant harmony

Things we didn't get to

- We didn't talk much about mappings.

  phonology underlying representations to surface forms

  syntax dependency tree to phrase structure tree or multi-dominance tree ($\approx$ DAG)

  Check Heinz (2018) and Chandlee and Heinz (2018)

- It's actually very easy to make SL and TSL probabilistic because they're so closely related to n-gram models. Check Mayer (2021) for probabilistic TSL

- What we did rests on tons of mathematical theorems.
  I would have liked to give more of a glimpse under the hood.

What could have been covered instead

A survey course on language and computation could choose from so many topics.

- formal language theory
  strings: (Hopcroft and Ullman 1979; Kozen 1997; Sipser 2005)
  trees: (Gécseg and Steinby 1984, 1997; Comon et al. 2008)
- finite-state methods
  (Beesley and Karttunen 2003; Karttunen and Beesley 2005)
- parsing and sentence processing
  (Kallmeyer 2010; Hale 2014; Fowlie and Koller 2017; Graf et al. 2017; Stanojević and Stabler 2018; Torr et al. 2019)

What could have been covered instead [cont.]

- formalisms: Tree Adjoining Grammar (TAG), Combinatory Categorial Grammar (CCG), Minimalist grammars (MGs)
  (Abeille and Rambow 2001; Steedman and Baldridge 2003; Stabler 2011; Graf forthcoming)
- model-theoretic syntax
  (Backofen et al. 1995; Rogers 1998, 2003; Morawietz 2003; Mönnich 2012; Graf 2013)
- linguistics for mathematically inclined students
  (Kornai 2007)
- mathematical concepts for the study of language
  (Partee et al. 1990; Keenan and Moss 2017)
- the very fabric of the universe
  (Kracht 2003)

Course summary: the why

Why, then, focus on subregularity?

- manageable learning curve
- lots of modeling work in addition to theorem proving
- easy to apply to linguistic data
- empirical coverage across many languages
- easy to add to existing research interests
- addresses fundamental linguistic issues
- completely new perspective on some phenomena
- increasingly making it into theoretical linguistics journals
- it has worked really well for my own students
  academia, Google, Apple, Amazon, Facebook, . . .

A personal story

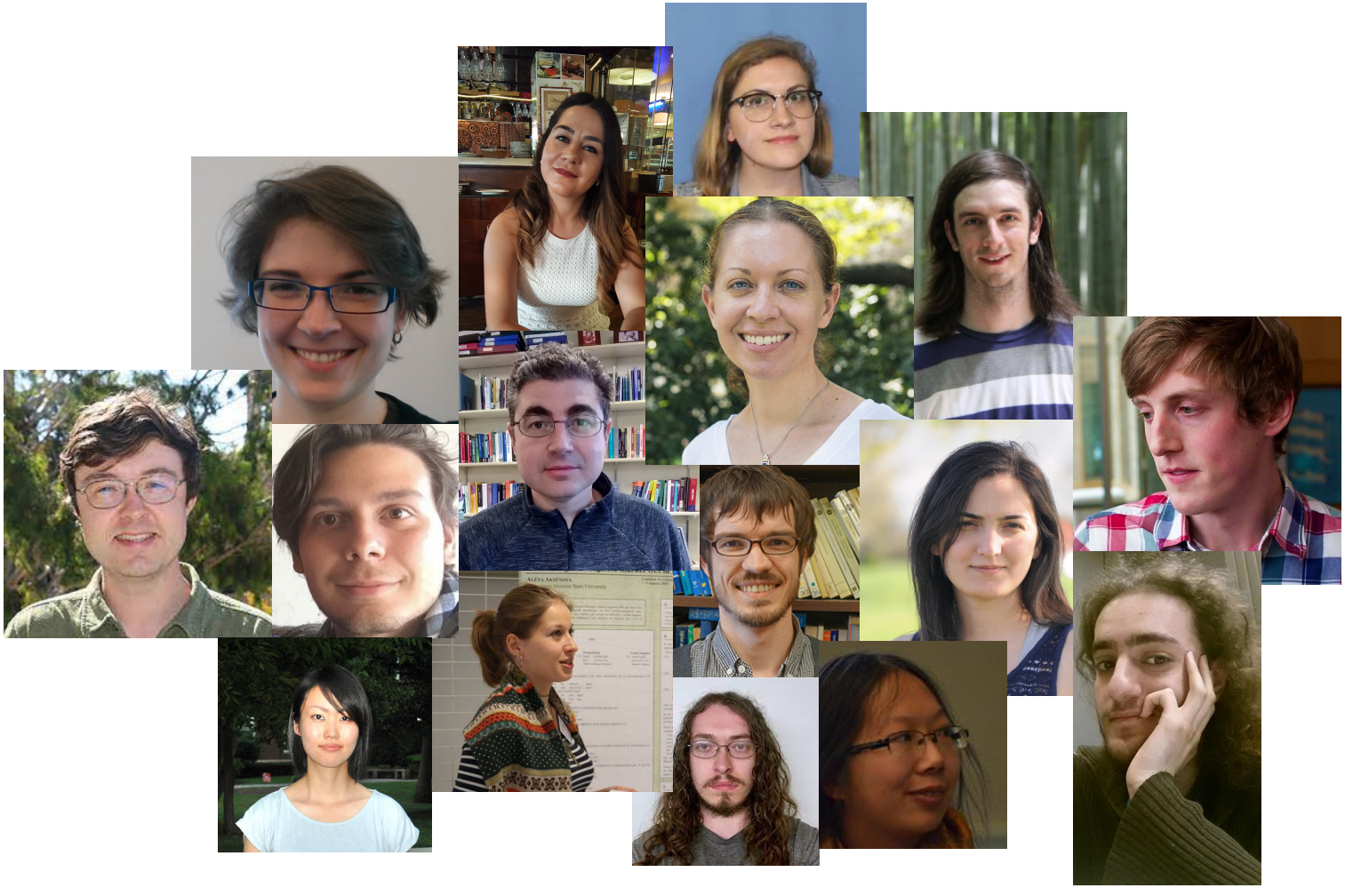
ESSLLI 2007 I sit in Jim Roger's class on logic and subregular complexity

ESSLLI 2010 I win the ESSLLI student session with a paper on the subregular complexity of Government Phonology

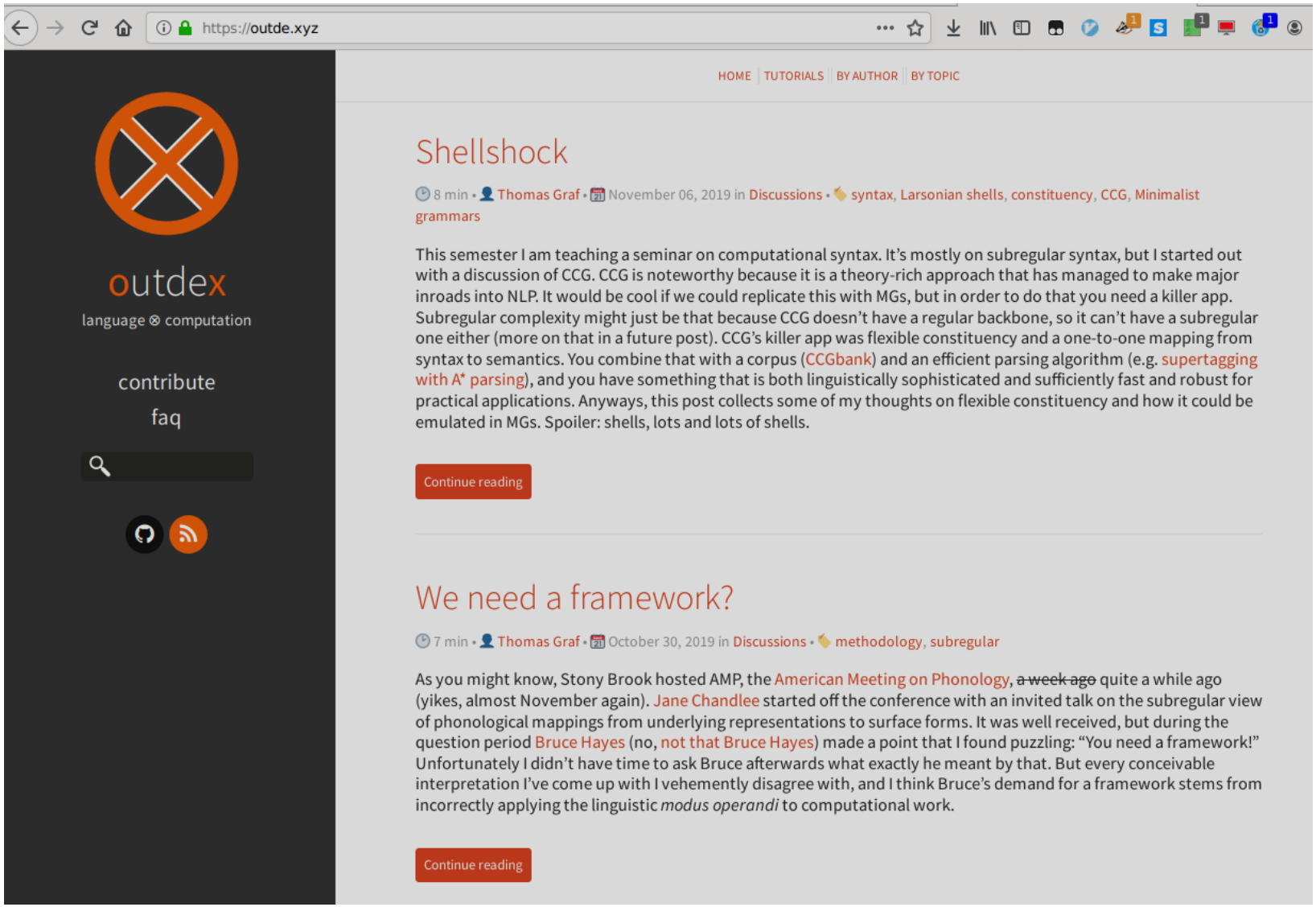
ESSLLI 2014 I receive the E.W. Beth dissertation prize for my thesis, which uses logic to analyze syntactic constraints in MGs

- ESSLLI kickstarted my voyage into computational linguistics and was a fellow traveler along the way.
- I hope that for some of you, this ESSLLI course will do the same.

A growing community (SBU, Rutgers, UCI, Ottawa, . . .)



Follow along (`https://outde.xyz`)



Learning more about string land

1. **Survey papers:** Pullum and Rogers (2006); Heinz (2011a,b, 2018); Rogers and Pullum (2011); Chandlee and Heinz (2016)
2. **TSL and its extensions:** Heinz et al. (2011); McMullin (2016); Baek (2018); De Santo (2017); De Santo and Graf (2019); Graf (2017); Graf and Mayer (2018); Mayer and Major (2018); Mayer (2021)
3. **TSL morphology:** Aksënova et al. (2016)
4. **TSL morpho-semantics:** Graf (2019)
5. **TSL string-syntax:** Graf and Shafiei (2019); Shafiei and Graf (2020)
6. **String parallels between phonology and syntax:** Graf (2018a); Ikawa and Jardine (2020)
7. **Mappings:** Courcelle and Engelfriet (2012); Chandlee (2014, 2017); Jardine (2016)
8. **Learnability:** Heinz (2010); Kasprzik and Kötzing (2010); Heinz et al. (2012); Jardine et al. (2014); Lai (2015); Jardine and Heinz (2016); Jardine and McMullin (2017)

Learning more about tree land

1. **TSL syntax:** Graf (2012, 2018b); Graf and Shafiei (2019); Graf and Kostyszyn (2021); Shafiei and Graf (2020); Vu (2018); Vu et al. (2019)
2. **SL tree mappings**: Graf (2020)
3. **Sensing tree automata (not discussed)**: Graf and De Santo (2019)

References I

Abeille, Anne, and Owen Rambow. 2001. *Tree adjoining grammars: Formalisms, linguistic analysis and processing*. Stanford: CSLI.

Aksënova, Alëna, Thomas Graf, and Sedigheh Moradi. 2016. Morphotactics as tier-based strictly local dependencies. In *Proceedings of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, 121–130. URL `https://www.aclweb.org/anthology/W/W16/W16-2019.pdf`.

Backofen, Rolf, James Rogers, and K. Vijay-Shanker. 1995. A first-order axiomatization of the theory of finite trees. *Journal of Logic, Language and Information* 4:5–39.

Baek, Hyunah. 2018. Computational representation of unbounded stress: Tiers with structural features. In *Proceedings of CLS 53*, ed. Daniel Edmiston, Marina Ermolaeva, Emre Hakgüder, Jackie Lai, Kathryn Montemurro, Brandon Rhodes, Amara Sankhagowit, and Miachel Tabatowski, 13–24.

Beesley, Kenneth R., and Lauri Karttunen. 2003. *Finite state morphology*. Stanford: CSLI.

van Benthem, Johan. 1986. Semantic automata. In *Essays in logical semantics*, 151–176. Dordrecht: Springer.

Chandlee, Jane. 2014. *Strictly local phonological processes*. Doctoral Dissertation, University of Delaware. URL `http://udspace.udel.edu/handle/19716/13374`.

References II

Chandlee, Jane. 2017. Computational locality in morphological maps. *Morphology* 27:599–641.

Chandlee, Jane, and Jeffrey Heinz. 2016. Computational phonology. Ms., Haverford College and University of Delaware.

Chandlee, Jane, and Jeffrey Heinz. 2018. Strict locality and phonological maps. *Linguistic Inquiry* 49:23–60.

Comon, H., M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. TommasiK. 2008. Tree automata: Techniques and applications. Published online: `http://www.grappa.univ-lille3.fr/tata`. URL `http://www.grappa.univ-lille3.fr/tata`, release from November 18, 2008.

Courcelle, Bruno, and Joost Engelfriet. 2012. *Graph structure and monadic second-order logic: A language-theoretic approach*. Cambridge, UK: Cambridge University Press.

De Santo, Aniello. 2017. Extending TSL languages: Conjunction as multiple tier-projection. Ms., Stony Brook University.

De Santo, Aniello, and Thomas Graf. 2019. Structure sensitive tier projection: Applications and formal properties. In *Formal Grammar*, ed. Raffaella Bernardi, Gregory Kobele, and Sylvain Pogodalla, 35–50. Berlin, Heidelberg: Springer.

References III

Fowlie, Meaghan, and Alexander Koller. 2017. Parsing minimalist languages with interpreted regular tree grammars. In *Proceedings of the 13th International Workshop on Tree Adjoining Grammars and Related Formalisms*, 11–20.

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. URL `http://dx.doi.org/10.1007/978-3-642-32024-8_14`.

Graf, Thomas. 2013. *Local and transderivational constraints in syntax and semantics*. Doctoral Dissertation, UCLA. URL `http://thomasgraf.net/doc/papers/Graf13Thesis.pdf`.

Graf, Thomas. 2017. The power of locality domains in phonology. *Phonology* 34:385–405. URL `https://dx.doi.org/10.1017/S0952675717000197`.

Graf, Thomas. 2018a. Locality domains and phonological c-command over strings. In *NELS 48: Proceedings of the Forty-Eighth Annual Meeting of the North East Linguistic Society*, volume 1, 257–270. Amherst, MA: GLSA. URL `http://ling.auf.net/lingbuzz/004080`.

References IV

Graf, Thomas. 2018b. Why movement comes for free once you have adjunction. In *Proceedings of CLS 53*, ed. Daniel Edmiston, Marina Ermolaeva, Emre Hakgüder, Jackie Lai, Kathryn Montemurro, Brandon Rhodes, Amara Sankhagowit, and Miachel Tabatowski, 117–136.

Graf, Thomas. 2019. A subregular bound on the complexity of lexical quantifiers. In *Proceedings of the 22nd Amsterdam Colloquium*, ed. Julian J. Schlöder, Dean McHugh, and Floris Roelofsen, 455–464.

Graf, Thomas. 2020. Curbing feature coding: Strictly local feature assignment. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2020*, 362–371.

Graf, Thomas. forthcoming. Computational linguistics. In *Handbook of Minimalism*, ed. Kleanthes K. Grohman and Evelina Leivada. Cambridge, MA: Cambridge University Press.

Graf, Thomas, and Aniello De Santo. 2019. Sensing tree automata as a model of syntactic dependencies. In *Proceedings of the 16th Meeting on the Mathematics of Language*, 12–26. Toronto, Canada: Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/W19-5702`.

Graf, Thomas, and Kalina Kostyszyn. 2021. Multiple wh-movement is not special: The subregular complexity of persistent features in Minimalist grammars. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2021*, 275–285.

References V

Graf, Thomas, and Connor Mayer. 2018. Sanskrit n-retroflexion is input-output tier-based strictly local. In *Proceedings of SIGMORPHON 2018*, 151–160.

Graf, Thomas, James Monette, and Chong Zhang. 2017. Relative clauses as a benchmark for Minimalist parsing. *Journal of Language Modelling* 5:57–106. URL `http://dx.doi.org/10.15398/jlm.v5i1.157`.

Graf, Thomas, and Nazila Shafiei. 2019. C-command dependencies as TSL string constraints. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2019*, ed. Gaja Jarosz, Max Nelson, Brendan O'Connor, and Joe Pater, 205–215.

Gécseg, Ferenc, and Magnus Steinby. 1984. *Tree automata*. Budapest: Academei Kaido. URL `http://arxiv.org/abs/1509.06233`.

Gécseg, Ferenc, and Magnus Steinby. 1997. Tree languages. In *Handbook of formal languages*, ed. Gregorz Rozenberg and Arto Salomaa, volume 3, 1–68. New York: Springer.

Hackl, Martin. 2009. On the grammar and processing of proportional quantifiers: Most versus more than half. *Natural Language Semantics* 17:63–98.

Hale, John. 2014. *Automaton theories of human sentence comprehension*. Stanford: CSLI.

Heinz, Jeffrey. 2010. String extension learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, 897–906. URL `http://www.aclweb.org/anthology/P10-1092.pdf`.

References VI

Heinz, Jeffrey. 2011a. Computational phonology — part I: Foundations. *Language and Linguistics Compass* 5:140–152.

Heinz, Jeffrey. 2011b. Computational phonology — part II: Grammars, learning, and the future. *Language and Linguistics Compass* 5:153–168.

Heinz, Jeffrey. 2018. The computational nature of phonological generalizations. In *Phonological typology*, ed. Larry Hyman and Frank Plank, Phonetics and Phonology, chapter 5, 126–195. Mouton De Gruyter.

Heinz, Jeffrey, Anna Kasprzik, and Timo Kötzing. 2012. Learning in the limit with lattice-structured hypothesis spaces. *Theoretical Computer Science* 457:111–127. URL `https://doi.org/10.1016/j.tcs.2012.07.017`.

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. URL `http://www.aclweb.org/anthology/P11-2011`.

Hopcroft, John E., and Jeffrey D. Ullman. 1979. *Introduction to automata theory, languages, and computation*. Reading, MA: Addison Wesley.

Ikawa, Shiori, and Adam Jardine. 2020. The computational similarity of binding and long-distance consonant dissimilation. In *Proceedings of NELS 2020*.

Jardine, Adam. 2016. Computationally, tone is different. *Phonology* 33:247–283. URL `https://doi.org/10.1017/S0952675716000129`.

References VII

Jardine, Adam, Jane Chandlee, Rémi Eryaud, and Jeffrey Heinz. 2014. Very efficient learning of structured classes of subsequential functions from positive data. In *Proceedings of the 12th International Conference on Grammatical Inference (ICGI 2014)*, JMLR Workshop Proceedings, 94–108. URL `http://www.jmlr.org/proceedings/papers/v34/jardine14a.html`.

Jardine, Adam, and Jeffrey Heinz. 2016. Learning tier-based strictly 2-local languages. *Transactions of the ACL* 4:87–98. URL `https://aclweb.org/anthology/Q/Q16/Q16-1007.pdf`.

Jardine, Adam, and Kevin McMullin. 2017. Efficient learning of tier-based strictly k-local languages. In *Proceedings of Language and Automata Theory and Applications*, Lecture Notes in Computer Science, 64–76. Berlin: Springer. URL `https://doi.org/10.1007/978-3-319-53733-7_4`.

Kallmeyer, Laura. 2010. *Parsing beyond context-free grammars*. Berlin: Springer.

Karttunen, Lauri, and Kenneth R. Beesley. 2005. Twenty-five years of finite-state morphology. In *Inquiries into words, constraints and context. Festschrift for Kimmo Koskenniemi on his 60th birthday*, ed. Antti Arppe, Lauri Carlson, Krister Lindén, Jussi Piitulainen, Mickael Suominen, Martti Vainio, Hanna Westerlund, and Anssi Yli-Jyrä, 71–83. Stanford: CSLI.

References VIII

Kasprzik, Anna, and Timo Kötzing. 2010. String extension learning using lattices. In *Language and automata theory and applications: 4th international conference, LATA 2010, Trier, Germany, May 24-28, 2010*, ed. Adrian-Horia Dediu, Henning Fernau, and Carlos Martín-Vide, 380–391. Berlin, Heidelberg: Springer. URL `http://dx.doi.org/10.1007/978-3-642-13089-2_32`.

Keenan, Edward, and Larry Moss. 2017. *Mathematical structures in language*. Stanford: CSLI.

Kornai, Andras. 2007. *Mathematical linguistics*. New York: Springer.

Kozen, Dexter C. 1997. *Automata and computability*. Springer.

Kracht, Marcus. 2003. *The mathematics of language*. Berlin: Mouton de Gruyter.

Lai, Regine. 2015. Learnable vs. unlearnable harmony patterns. *Linguistic Inquiry* 46:425–451.

Mayer, Connor. 2021. Capturing gradience in long-distance phonology using probabilistic tier-based strictly local grammars. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2021*, 39–50.

Mayer, Connor, and Travis Major. 2018. A challenge for tier-based strict locality from Uyghur backness harmony. In *Proceedings of Formal Grammar 2018*. To appear.

McMullin, Kevin. 2016. *Tier-based locality in long-distance phonotactics: Learnability and typology*. Doctoral Dissertation, University of British Columbia.

References IX

Mönnich, Uwe. 2012. A logical characterization of extended TAGs. In *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+11)*, 37–45. Paris, France. URL `http://www.aclweb.org/anthology-new/W/W12/W12-4605`.

Morawietz, Frank. 2003. *Two-step approaches to natural language formalisms*. Berlin: Walter de Gruyter.

Paperno, Denis. 2011. Learnable classes of natural language quantifiers: Two perspectives. URL `http://paperno.bol.ucla.edu/q_learning.pdf`, ms., UCLA.

Partee, Barbara, Alice ter Meulen, and Robert Wall. 1990. *Mathematical methods in linguistics*. Dordrecht: Kluwer.

Pullum, Geoffrey K., and James Rogers. 2006. Animal pattern-learning experiments: Some mathematical background. Ms., Radcliffe Institute for Advanced Study, Harvard University.

Rogers, James. 1998. *A descriptive approach to language-theoretic complexity*. Stanford: CSLI.

Rogers, James. 2003. Syntactic structures as multi-dimensional trees. *Research on Language and Computation* 1:265–305.

Rogers, James, and Geoffrey K. Pullum. 2011. Aural pattern recognition experiments and the subregular hierarchy. *Journal of Logic, Language and Information* 20:329–342.

References X

Shafiei, Nazila, and Thomas Graf. 2020. The subregular complexity of syntactic islands. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2020*, 272–281.

Sipser, Michael. 2005. *Introduction to the theory of computation*. Course Technology, second edition.

Stabler, Edward P. 2011. Computational perspectives on Minimalism. In *Oxford handbook of linguistic Minimalism*, ed. Cedric Boeckx, 617–643. Oxford: Oxford University Press.

Stanojević, Miloš, and Edward Stabler. 2018. A sound and complete left-corner parser for Minimalist grammars. In *Proceedings of the 8th Workshop on Cognitive Aspects of Computational Language Learning and Processing*, 65–74.

Steedman, Mark, and Jason Baldridge. 2003. Combinatory categorial grammar. Unpublished tutorial paper.

Torr, John, Miloš Stanojević, Mark Steedman, and Shay Cohen. 2019. Wide-coverage neural A* parsing for Minimalist grammars. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Florence, Italy: Association for Computational Linguistics.

Vu, Mai Ha. 2018. Towards a formal description of NPI-licensing patterns. In *Proceedings of the Society for Computation in Linguistics*, volume 1, 154–163.

References XI

Vu, Mai Ha, Nazila Shafiei, and Thomas Graf. 2019. Case assignment in TSL syntax: A case study. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2019*, ed. Gaja Jarosz, Max Nelson, Brendan O'Connor, and Joe Pater, 267–276.