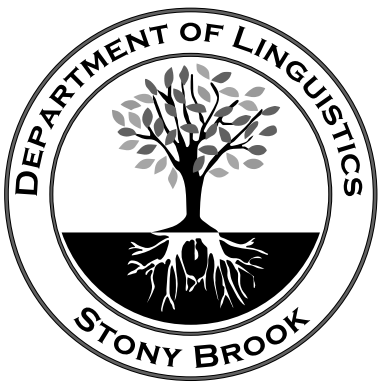
Subregular Linguistics for. . . well, Linguists

(Part 1: String Land)

Thomas Graf



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

MIT Mini Course
May 5 & 6, 2021

The Big Linguistic Questions

- What are the laws that govern each structural level?
- **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?

The Opportunistic Program for Lazy Researchers Like Myself

- 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 each structural level?
- **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?

The Opportunistic Program for Lazy Researchers Like Myself

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

A Mathematical Distinctness Theorem

- From a computational perspective, there is a split between "P-side" and "S-side".

$$\text{regular} < \text{context-free} < \text{mildly context-sensitive} < \cdots$$

Phonology

Morphology

Syntax

- Matches linguistic practice (despite attempts at unification, e.g. Government Phonology, DM, OT syntax)
- A unified Theory of Everything is not on the linguistic horizon.

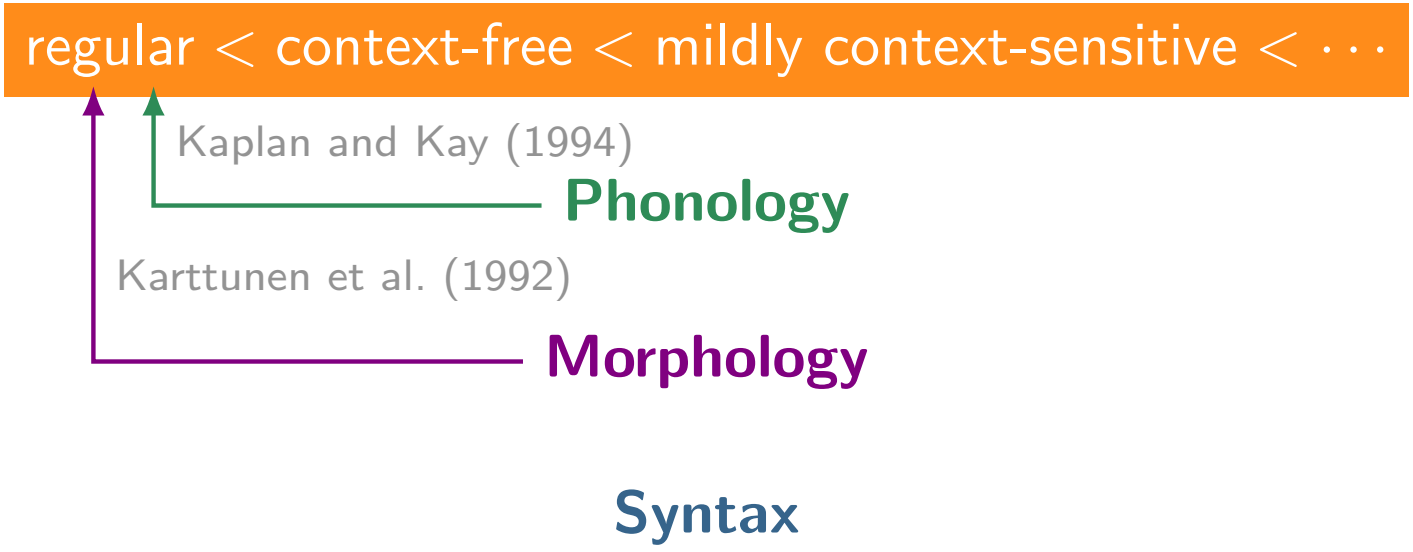
A Mathematical Distinctness Theorem

- From a computational perspective, there is a split between "P-side" and "S-side".

- Matches linguistic practice (despite attempts at unification, e.g. Government Phonology, DM, OT syntax)
- A unified Theory of Everything is not on the linguistic horizon.

A Mathematical Distinctness Theorem

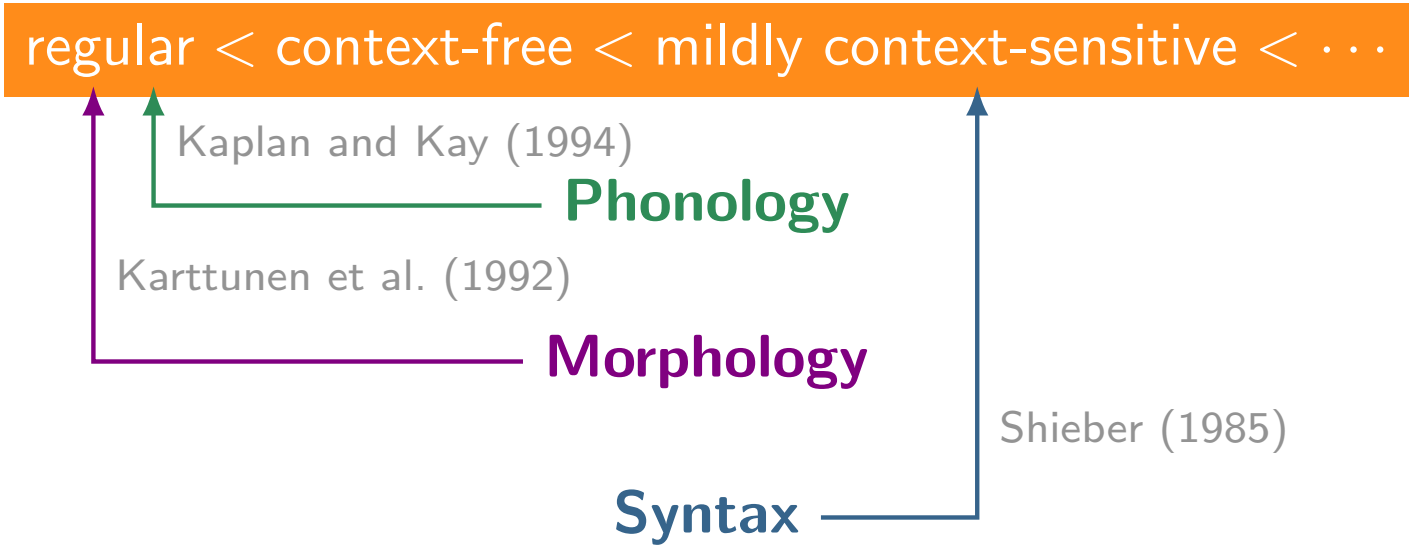
- From a computational perspective, there is a split between "P-side" and "S-side".



- Matches linguistic practice (despite attempts at unification, e.g. Government Phonology, DM, OT syntax)
- A unified Theory of Everything is not on the linguistic horizon.

A Mathematical Distinctness Theorem

- From a computational perspective, there is a split between "P-side" and "S-side".



- Matches linguistic practice (despite attempts at unification, e.g. Government Phonology, DM, OT syntax)
- A unified Theory of Everything is not on the linguistic horizon.

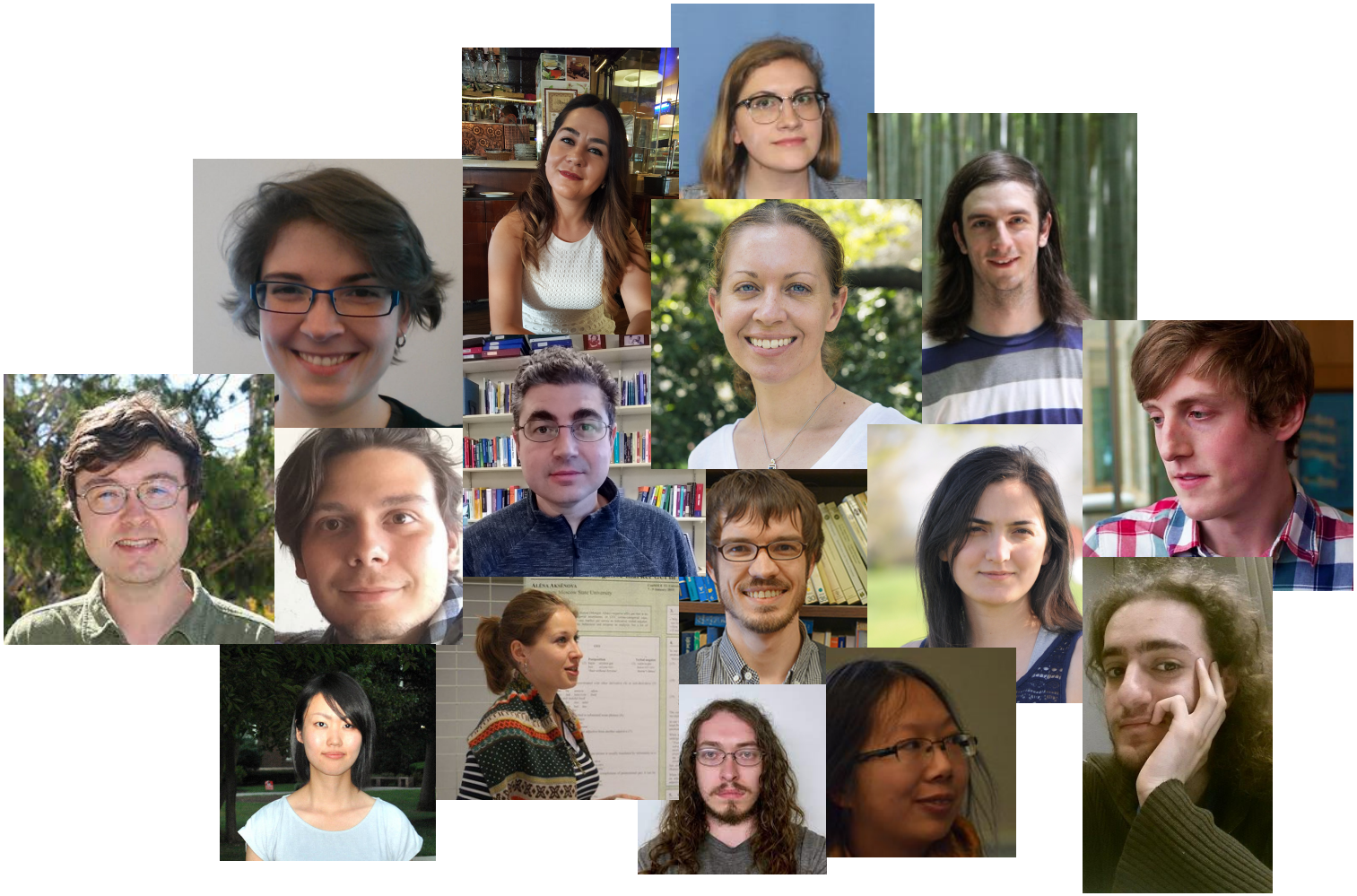
Cognitive Parallelism Hypothesis

- The postulated split is misleading.
- If we probe deeper, we find that
  - different modules are remarkably similar,
  - their dependencies are weaker than regular
    $\Rightarrow$ **subregular**
- Cognitive parallelism is **empirically fertile**.

Take-Home Messages

1. Phonology and syntax show surprising subregular parallels.
   (Morphology and morphosemantics, too. . .)
2. Like every good theory, subregularity yields
   new generalizations and data.
3. You can easily add subregular ideas to your own research!

A Growing Community



A Growing Community

Outline

SL & TSL: (T)ier-Based Strictly Local

- There are a variety of subregular classes to choose from.
- SL and TSL are among the weaker ones.
- They work well empirically.

(Tier-Based) Strictly Local Dependencies

- All patterns described by markedness constraints that are
  - inviolable,
  - locally bounded,
  - formalized as n-grams.
- Non-local dependencies are **local over tiers**.
  (Goldsmith 1976; but there is another. . .)
- **Linguistic core idea:**
  Dependencies are local over the right structure.

SL & TSL: (T)ier-Based Strictly Local

- There are a variety of subregular classes to choose from.
- SL and TSL are among the weaker ones.
- They work well empirically.

(Tier-Based) Strictly Local Dependencies

- All patterns described by markedness constraints that are
  - inviolable,
  - locally bounded,
  - formalized as n-grams.
- Non-local dependencies are **local over tiers**.
  (Goldsmith 1976; but there is another. . .)
- **Linguistic core idea:**
  Dependencies are local over the right structure.

Example: Word-Final Devoicing is SL-2

- Captured by forbidding voiced segments at the end of a word
- **German**: Don't have **z$** or **v$** or **d$** (where **$** = word edge).

Example: German

*$ r a d $

*z$

*v$

*d$

Example: Word-Final Devoicing is SL-2

- Captured by forbidding voiced segments at the end of a word
- **German**: Don't have **z$** or **v$** or **d$** (where **$** = word edge).

Example: German

	*z$
*$ r a d $	*v$
	*d$

Example: Word-Final Devoicing is SL-2

- Captured by forbidding voiced segments at the end of a word
- **German**: Don't have **z$** or **v$** or **d$** (where **$** = word edge).

Example: German

*$ r a d $

*z$

*v$

*d$

Example: Word-Final Devoicing is SL-2

- Captured by forbidding voiced segments at the end of a word
- **German**: Don't have **z$** or **v$** or **d$** (where **$** = word edge).

Example: German

* **$** r a **d** **$**

* **z$**

* **v$**

* **d$**

Example: Word-Final Devoicing is SL-2

- Captured by forbidding voiced segments at the end of a word
- **German**: Don't have **z$** or **v$** or **d$** (where **$** = word edge).

Example: German

* **$** r a **d** **$**

* **z$**

* **v$**

* **d$**

Example: Word-Final Devoicing is SL-2

- Captured by forbidding voiced segments at the end of a word
- **German**: Don't have **z$** or **v$** or **d$** (where **$** = word edge).

Example: German

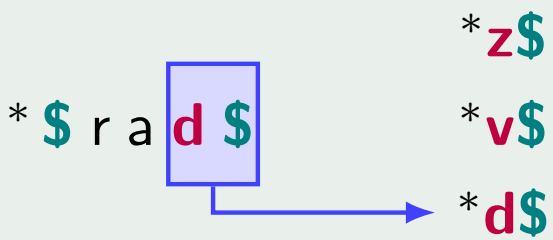
*$ r a d $

*z$
*v$
*d$

Example: Word-Final Devoicing is SL-2

- Captured by forbidding voiced segments at the end of a word
- **German**: Don't have **z$** or **v$** or **d$** (where **$** = word edge).

Example: German



Example: Word-Final Devoicing is SL-2

- Captured by forbidding voiced segments at the end of a word
- **German**: Don't have **z$** or **v$** or **d$** (where **$** = word edge).

Example: German

	*z$	
*$ r a d $	*v$	$ r a t $
	*d$	

Example: Word-Final Devoicing is SL-2

- Captured by forbidding voiced segments at the end of a word
- **German**: Don't have **z$** or **v$** or **d$** (where **$** = word edge).

Example: German

*$ r a d $

*z$
*v$
*d$

$ r a t $

Example: Word-Final Devoicing is SL-2

- Captured by forbidding voiced segments at the end of a word
- **German**: Don't have **z$** or **v$** or **d$** (where **$** = word edge).

Example: German

* **$** r a **d** **$**

* **z$**
* **v$**
* **d$**

$** r a t **$

Example: Word-Final Devoicing is SL-2

- Captured by forbidding voiced segments at the end of a word
- **German**: Don't have **z$** or **v$** or **d$** (where **$** = word edge).

Example: German

*$ r a d $ *z$ *v$ *d$ $ r a t $

Example: Word-Final Devoicing is SL-2

- Captured by forbidding voiced segments at the end of a word
- **German**: Don't have **z$** or **v$** or **d$** (where **$** = word edge).

Example: German

*$ r a d $

*z$
*v$
*d$

$ r a t $

Example: Intervocalic Voicing is SL-3

- Captured by forbidding voiceless segments 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**: don't have **asa**, **aʃa**, **asi**, **aʃi**, . . .

Example: Northern Italian

* $ a s o l a $

Example: Intervocalic Voicing is SL-3

- Captured by forbidding voiceless segments 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**: don't have **asa**, **aʃa**, **asi**, **aʃi**, . . .

Example: Northern Italian

* $ **a** **s** **o** l **a** $

Example: Intervocalic Voicing is SL-3

- Captured by forbidding voiceless segments 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**: don't have **asa**, **aʃa**, **asi**, **aʃi**, . . .

Example: Northern Italian

*$ **a** **s** **o** l **a** $

Example: Intervocalic Voicing is SL-3

- Captured by forbidding voiceless segments 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**: don't have **asa**, **aʃa**, **asi**, **aʃi**, . . .

Example: Northern Italian

* $ **a s o** l **a** $

Example: Intervocalic Voicing is SL-3

- Captured by forbidding voiceless segments 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**: don't have **asa**, **aʃa**, **asi**, **aʃi**, . . .

Example: Northern Italian

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $

Example: Intervocalic Voicing is SL-3

- Captured by forbidding voiceless segments 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**: don't have **asa**, **aʃa**, **asi**, **aʃi**, . . .

Example: Northern Italian

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $

Example: Intervocalic Voicing is SL-3

- Captured by forbidding voiceless segments 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**: don't have **asa**, **aʃa**, **asi**, **aʃi**, . . .

Example: Northern Italian

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $

Example: Intervocalic Voicing is SL-3

- Captured by forbidding voiceless segments 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**: don't have **asa**, **aʃa**, **asi**, **aʃi**, . . .

Example: Northern Italian

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $

Example: Intervocalic Voicing is SL-3

- Captured by forbidding voiceless segments 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**: don't have **asa**, **aʃa**, **asi**, **aʃi**, . . .

Example: Northern Italian

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $

Example: Intervocalic Voicing is SL-3

- Captured by forbidding voiceless segments 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**: don't have **asa**, **aʃa**, **asi**, **aʃi**, . . .

Example: Northern Italian

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $

Example: Intervocalic Voicing is SL-3

- Captured by forbidding voiceless segments 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**: don't have **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** $

Example: Intervocalic Voicing is SL-3

- Captured by forbidding voiceless segments 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**: don't have **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** $

Example: Intervocalic Voicing is SL-3

- Captured by forbidding voiceless segments 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**: don't have **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** $

Example: Intervocalic Voicing is SL-3

- Captured by forbidding voiceless segments 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**: don't have **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** $

Example: Intervocalic Voicing is SL-3

- Captured by forbidding voiceless segments 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**: don't have **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** $

Example: Intervocalic Voicing is SL-3

- Captured by forbidding voiceless segments 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**: don't have **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** $

Example: Intervocalic Voicing is SL-3

- Captured by forbidding voiceless segments 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**: don't have **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** $

Example: Intervocalic Voicing is SL-3

- Captured by forbidding voiceless segments 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**: don't have **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** $

Example: Intervocalic Voicing is SL-3

- Captured by forbidding voiceless segments 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**: don't have **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** $

Test Your Might

Exercise

Suppose that a language with CV as the only syllable template exhibits vowel harmony without any neutral vowels or any blockers.

1. What is the complexity of vowel harmony in this language?
2. What if CVC syllables are also possible?

Exercise

Can you give an example of an SL string dependency in (morpho)syntax?

More SL: 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

More SL: 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

Every and *No* are SL-1

- Every quantifier defines a set of licit **strings over Y and N**.

every	string must not contain N (*N ⇒ SL-1)
no	string must not contain Y (*Y ⇒ SL-1)

- But what about other quantifiers?

some	string contains at least one Y
not all	string contains at least one N
three	string contains at least three Y
most	string contains more Y than N
even number	string contains $2n$ Y, $n \geq 0$

- These languages are **not SL**!

The Limits of SL

Example: *some* is not SL

1. The quantifier language for *some* must
   - include every string of the form NN$\cdots$Y$\cdots$NN,
   - exclude every string of the form NN$\cdots$N$\cdots$NN.
2. But every **n**-gram of some illicit string is also an **n**-gram of some well-formed string.
3. Forbidding any **n**-gram of an illicit string means incorrectly ruling out some well-formed strings.
4. Hence the quantifier language of *some* cannot be SL.

Exercise

Show that the quantifier languages of *most* and *an even number* are not SL either.

The Limits of SL

Example: *some* is not SL

1. The quantifier language for *some* must
   - include every string of the form NN$\cdots$Y$\cdots$NN,
   - exclude every string of the form NN$\cdots$N$\cdots$NN.
2. But every **n**-gram of some illicit string is also an **n**-gram of some well-formed string.
3. Forbidding any **n**-gram of an illicit string means incorrectly ruling out some well-formed strings.
4. Hence the quantifier language of *some* cannot be SL.

Exercise

Show that the quantifier languages of *most* and *an even number* are not SL either.

Some in Phonology

- The problem with *some* goes beyond semantics.

Culminativity

Every phonological word has **exactly one** syllable that carries primary stress.

Culminativity (Rephrased)

Every phonological word has

- **at most one** syllable that carries primary stress, and
- **some** syllable that carries primary stress.

- Culminativity cannot be SL.

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** $

Making Long-Distance Dependencies Local

- Let's take a clue from phonology: create locality with **tiers**. (Heinz et al. 2011)

Jeff Heinz

Example: Samala Revisited

1. Project sibilant tier
2. *sʃ, *s3, *zʃ, *z3, *ʃs, *3s, *ʃz, *3z

*$ 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 $

Making Long-Distance Dependencies Local

- Let's take a clue from phonology: create locality with **tiers**. (Heinz et al. 2011)

Jeff Heinz

Example: Samala Revisited

1. Project sibilant tier
2. *sʃ, *sʒ, *zʃ, *zʒ, *ʃs, *ʒs, *ʃz, *ʒz

*$ 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 $

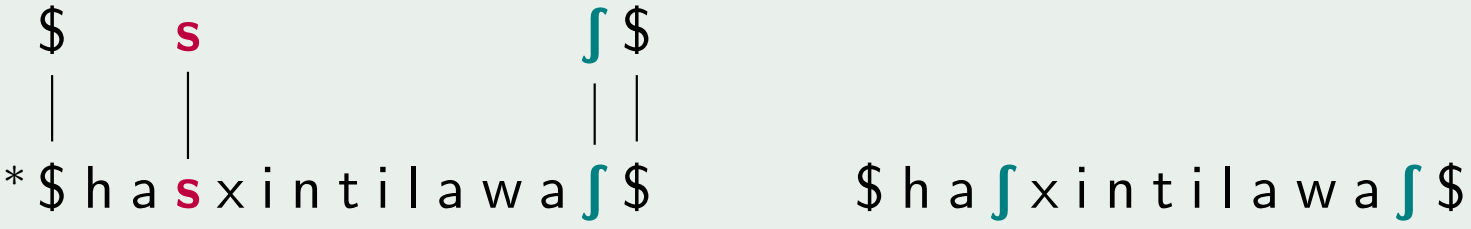
Making Long-Distance Dependencies Local

- Let's take a clue from phonology: create locality with **tiers**. (Heinz et al. 2011)

Jeff Heinz

Example: Samala Revisited

1. Project sibilant tier
2. *sʃ, *sʒ, *zʃ, *zʒ, *ʃs, *ʒs, *ʃz, *ʒz



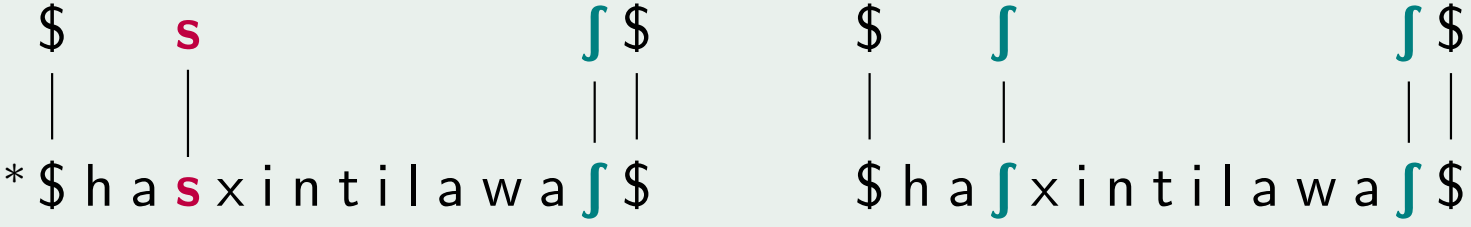
Making Long-Distance Dependencies Local

- Let's take a clue from phonology: create locality with **tiers**. (Heinz et al. 2011)

Jeff Heinz

Example: Samala Revisited

1. Project sibilant tier
2. *sʃ, *sʒ, *zʃ, *zʒ, *ʃs, *ʒs, *ʃz, *ʒz



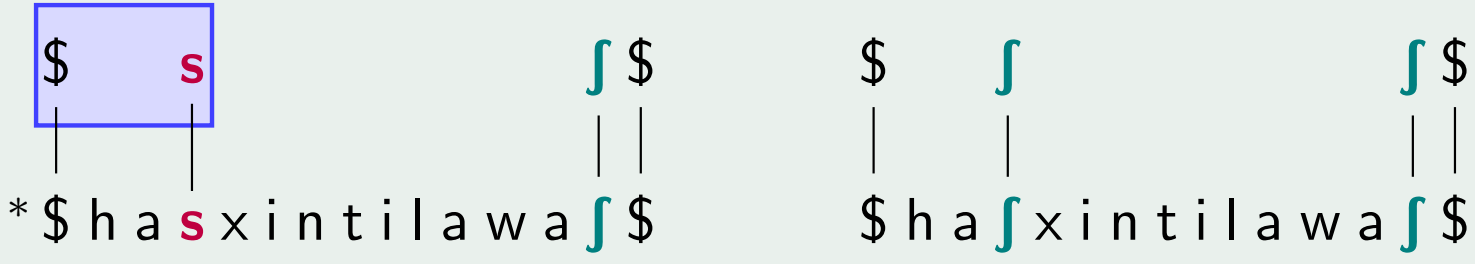
Making Long-Distance Dependencies Local

- Let's take a clue from phonology: create locality with **tiers**. (Heinz et al. 2011)

Jeff Heinz

Example: Samala Revisited

1. Project sibilant tier
2. *sʃ, *s3, *zʃ, *z3, *ʃs, *3s, *ʃz, *3z



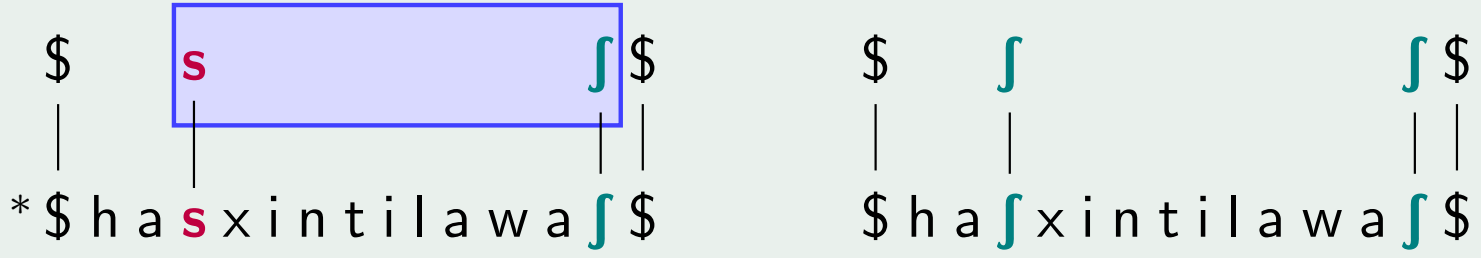
Making Long-Distance Dependencies Local

- Let's take a clue from phonology: create locality with **tiers**. (Heinz et al. 2011)

Jeff Heinz

Example: Samala Revisited

1. Project sibilant tier
2. *sʃ, *sʒ, *zʃ, *zʒ, *ʃs, *ʒs, *ʃz, *ʒz



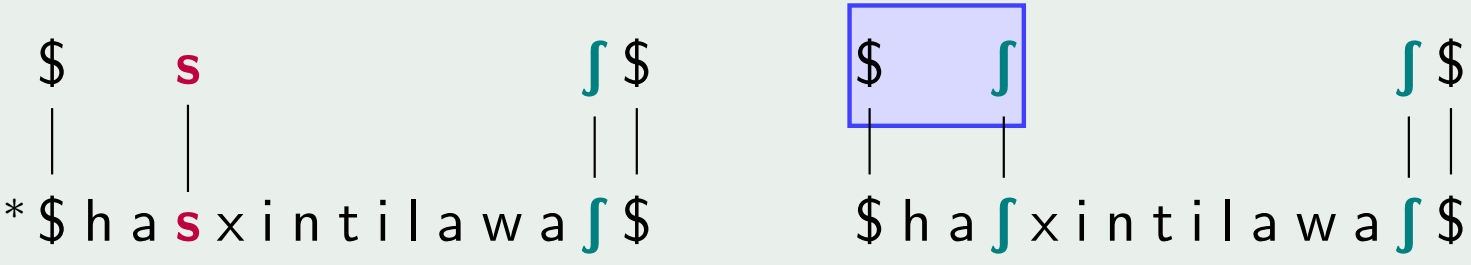
Making Long-Distance Dependencies Local

- Let's take a clue from phonology: create locality with **tiers**. (Heinz et al. 2011)

Jeff Heinz

Example: Samala Revisited

1. Project sibilant tier
2. *sʃ, *sʒ, *zʃ, *zʒ, *ʃs, *ʒs, *ʃz, *ʒz



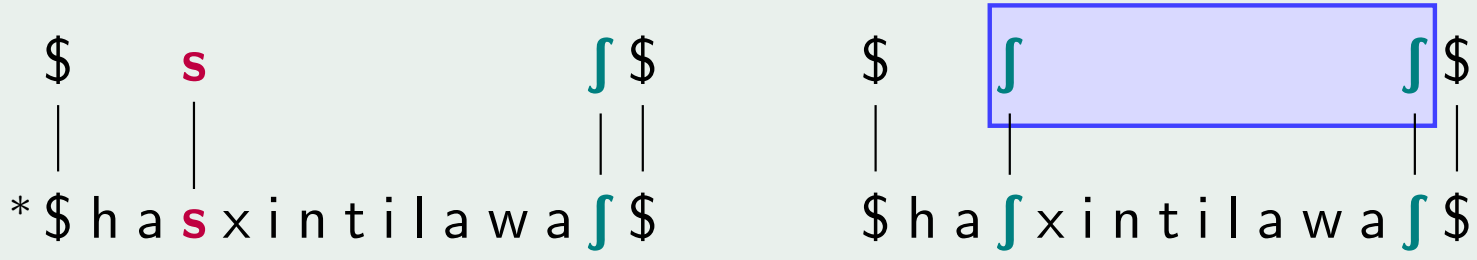
Making Long-Distance Dependencies Local

- Let's take a clue from phonology: create locality with **tiers**. (Heinz et al. 2011)

Jeff Heinz

Example: Samala Revisited

1. Project sibilant tier
2. *sʃ, *sʒ, *zʃ, *zʒ, *ʃs, *ʒs, *ʃz, *ʒz



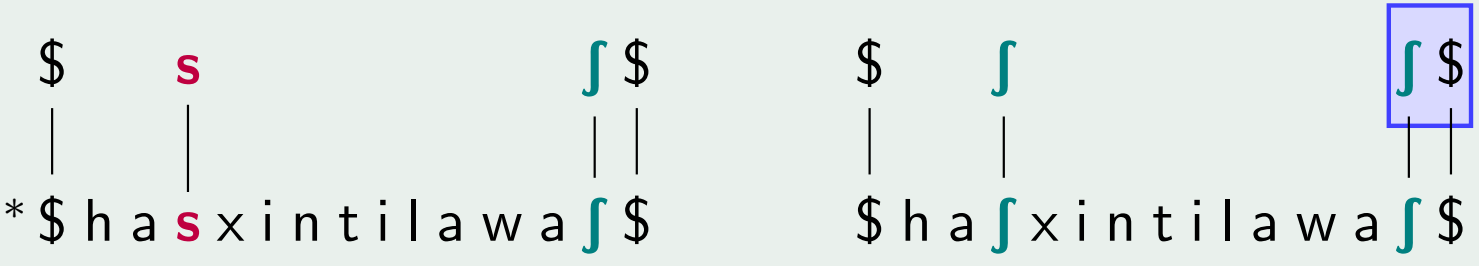
Making Long-Distance Dependencies Local

- Let's take a clue from phonology: create locality with **tiers**. (Heinz et al. 2011)

Jeff Heinz

Example: Samala Revisited

1. Project sibilant tier
2. *sʃ, *sʒ, *zʃ, *zʒ, *ʃs, *ʒs, *ʃz, *ʒz



Another TSL Boon: Blocking

- TSL can also handle blocking effects.
- **Slovenian sibilant harmony with blocking**
  1. **[−ant]** . . . **[+ant]** is illicit,
  2. unless **t** or **d** intervenes.
- **TSL-2 account**
  1. project all **[−ant]**, **[+ant]**, **t**, and **d**
  2. don't have **[−ant]** **[+ant]**

Example: Slovenian (Jurgec 2011; McMullin 2016)

* $ **s** p i **ʃ** $ $ **z** i **d** a **ʃ** $

Another TSL Boon: Blocking

- TSL can also handle blocking effects.
- **Slovenian sibilant harmony with blocking**
  1. **[−ant]** . . . **[+ant]** is illicit,
  2. unless **t** or **d** intervenes.
- **TSL-2 account**
  1. project all **[−ant]**, **[+ant]**, **t**, and **d**
  2. don't have **[−ant]** **[+ant]**

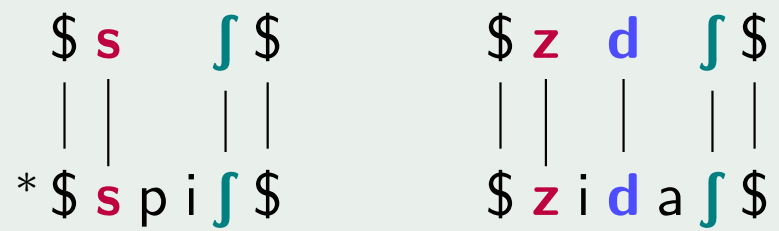
Example: Slovenian (Jurgec 2011; McMullin 2016)

$ s ʃ $ | | | | * $ s p i ʃ $ $ z i d a ʃ $

Another TSL Boon: Blocking

- TSL can also handle blocking effects.
- **Slovenian sibilant harmony with blocking**
  1. **[−ant]** . . . **[+ant]** is illicit,
  2. unless **t** or **d** intervenes.
- **TSL-2 account**
  1. project all **[−ant]**, **[+ant]**, **t**, and **d**
  2. don't have **[−ant]** **[+ant]**

Example: Slovenian (Jurgec 2011; McMullin 2016)



Some revisited

- TSL is a much better fit for quantifier languages.

Example: *Some* with Tier Projection

1. Project **Y**-tier
2. *$$ (tier must not be empty)

*$ N N N N N N N N N N $ $ N N Y N N N N N N Y $

Exercise

Suppose we also forbid YY on the Y-tier.

1. What quantifier does the resulting string set describe?
2. Do we perhaps need that somewhere else?

Some revisited

- TSL is a much better fit for quantifier languages.

Example: *Some* with Tier Projection

1. Project **Y**-tier
2. *$$ (tier must not be empty)

*$ N N N N N N N N N N $ $ N N Y N N N N N N Y $

Exercise

Suppose we also forbid **YY** on the **Y**-tier.

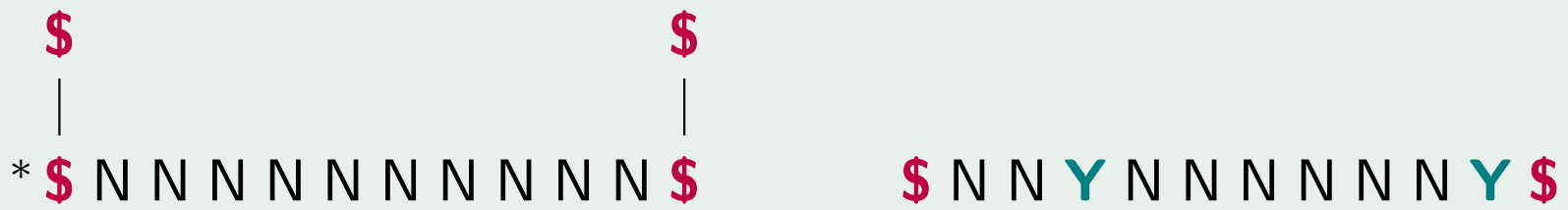
1. What quantifier does the resulting string set describe?
2. Do we perhaps need that somewhere else?

Some revisited

- TSL is a much better fit for quantifier languages.

Example: *Some* with Tier Projection

1. Project **Y**-tier
2. ***$$** (tier must not be empty)



Exercise

Suppose we also forbid **YY** on the **Y**-tier.

1. What quantifier does the resulting string set describe?
2. Do we perhaps need that somewhere else?

Some revisited

- TSL is a much better fit for quantifier languages.

Example: *Some* with Tier Projection

1. Project **Y**-tier
2. *$$ (tier must not be empty)

Exercise

Suppose we also forbid **YY** on the **Y**-tier.

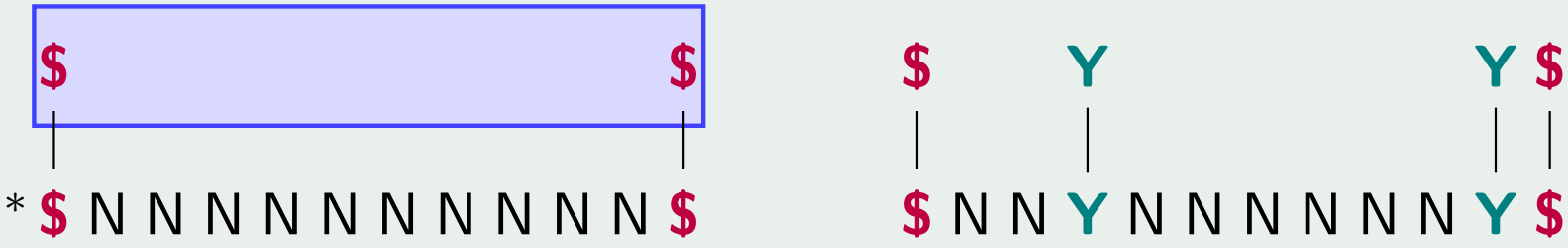
1. What quantifier does the resulting string set describe?
2. Do we perhaps need that somewhere else?

Some revisited

- TSL is a much better fit for quantifier languages.

Example: *Some* with Tier Projection

1. Project **Y**-tier
2. ***$$** (tier must not be empty)



Exercise

Suppose we also forbid **YY** on the **Y**-tier.

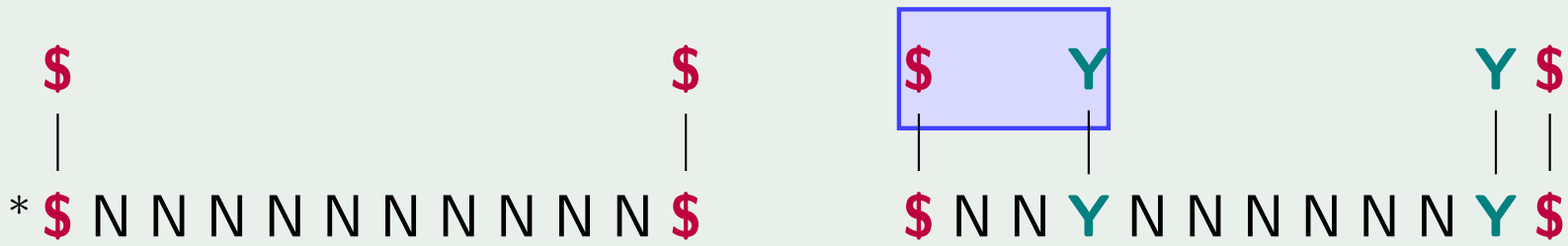
1. What quantifier does the resulting string set describe?
2. Do we perhaps need that somewhere else?

Some revisited

- TSL is a much better fit for quantifier languages.

Example: *Some* with Tier Projection

1. Project **Y**-tier
2. ***$$** (tier must not be empty)



Exercise

Suppose we also forbid **YY** on the **Y**-tier.

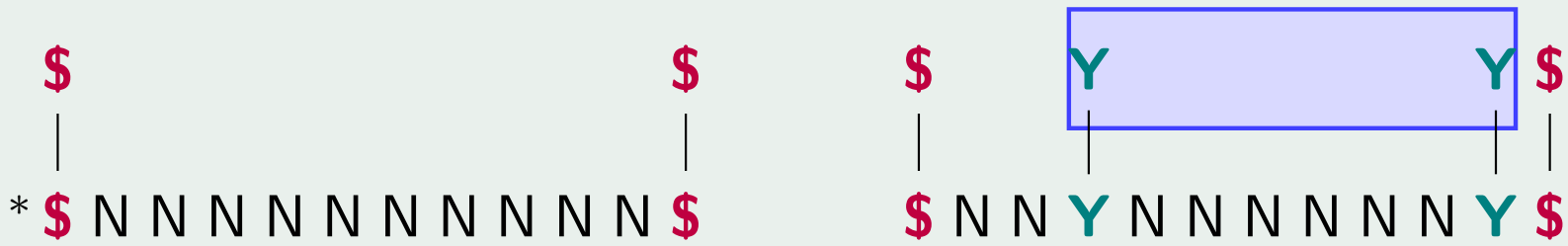
1. What quantifier does the resulting string set describe?
2. Do we perhaps need that somewhere else?

Some revisited

- TSL is a much better fit for quantifier languages.

Example: *Some* with Tier Projection

1. Project **Y**-tier
2. *$$ (tier must not be empty)



Exercise

Suppose we also forbid **YY** on the **Y**-tier.

1. What quantifier does the resulting string set describe?
2. Do we perhaps need that somewhere else?

Some revisited

- TSL is a much better fit for quantifier languages.

Example: *Some* with Tier Projection

1. Project **Y**-tier
2. *$$ (tier must not be empty)

Exercise

Suppose we also forbid **YY** on the **Y**-tier.

1. What quantifier does the resulting string set describe?
2. Do we perhaps need that somewhere else?

Overview of Quantifier Languages

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

Quantifier	**TSL?**	**Tier**	**Lex.** (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 simplex** in any language.

(Paperno 2011)

Quantifier	TSL?	Tier	Lex.
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 simplex** in any language.

(Paperno 2011)

Quantifier	TSL?	Tier	Lex.
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 simplex** in any language.

(Paperno 2011)

Quantifier	TSL?	Tier	Lex.
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		???

TSL

Overview of Quantifier Languages

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

(Paperno 2011)

Quantifier	TSL?	Tier	Lex.	
every	yes	none	yes	Lexical
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	TSL
all but one	yes	N	no	
even number	no		no	
prime number	no		no	
infinitely many	no		no	
most	no		???	

The Case of *most*

There is good semantic evidence that "most" is internally complex and hence **not lexically simplex**. (Hackl 2009)

Quantifier	TSL?	Tier	Lex.
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

(Paperno 2011)

Lexical

TSL

SL and TSL for Phonology and Semantics

- Linguistically natural
- Correct and efficient learning algorithm (Jardine and McMullin 2017)
- Low resource demands ⇒ cognitively plausible
- Captures wide range of phonotactic dependencies
- Excludes many unattested patterns

Example: First-Last Harmony

- Harmony only holds between initial and final segments
- Linguistically plausible, yet unattested

$ h a s x i n t i l a w a ʃ $ *$ s t a j a n o w o n w a ʃ $

SL and TSL for Phonology and Semantics

- Linguistically natural
- Correct and efficient learning algorithm (Jardine and McMullin 2017)
- Low resource demands ⇒ cognitively plausible
- Captures wide range of phonotactic dependencies
- Excludes many unattested patterns

Example: First-Last Harmony

- Harmony only holds between initial and final segments
- Linguistically plausible, yet unattested

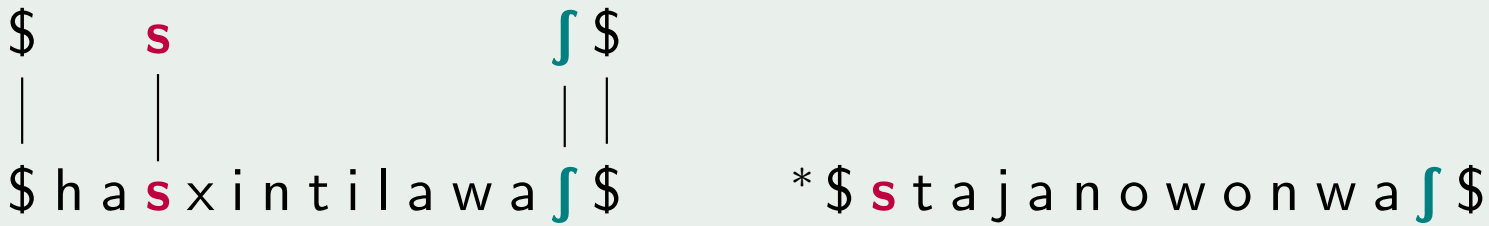
$ h a s x i n t i l a w a ʃ $ *$ s t a j a n o w o n w a ʃ $

SL and TSL for Phonology and Semantics

- Linguistically natural
- Correct and efficient learning algorithm (Jardine and McMullin 2017)
- Low resource demands ⇒ cognitively plausible
- Captures wide range of phonotactic dependencies
- Excludes many unattested patterns

Example: First-Last Harmony

- Harmony only holds between initial and final segments
- Linguistically plausible, yet unattested

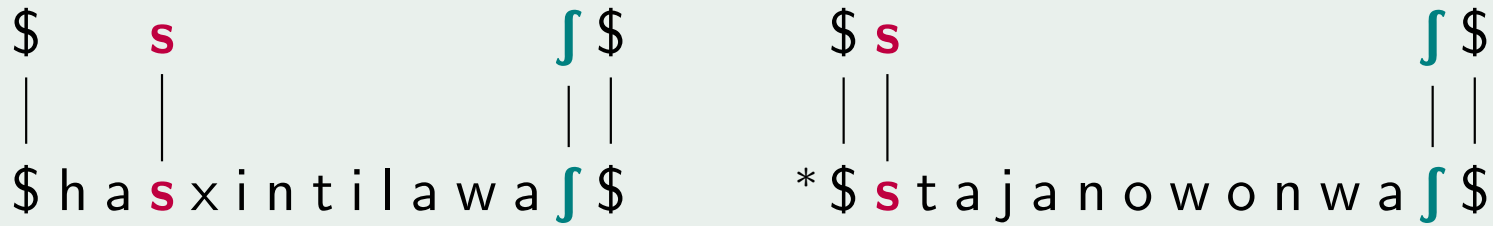


SL and TSL for Phonology and Semantics

- Linguistically natural
- Correct and efficient learning algorithm (Jardine and McMullin 2017)
- Low resource demands ⇒ cognitively plausible
- Captures wide range of phonotactic dependencies
- Excludes many unattested patterns

Example: First-Last Harmony

- Harmony only holds between initial and final segments
- Linguistically plausible, yet unattested



Supercharging the Tier Projection

- The TSL tier projection only considers individual symbols.
- But sometimes we need the context, too.

I[nput TSL]	consider surrounding symbols in input string
O[utput TSL]	consider previous symbols in output tier
IOTSL	consider both

What may Project?

Tier projection controlled by

1. label of segment
2. local context
3. symbols already on tier

What may Project?

Tier projection controlled by

1. label of segment
2. local context
3. symbols already on tier

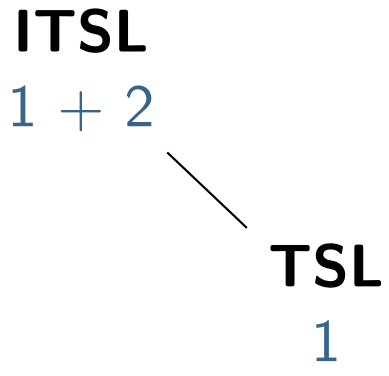
TSL

1

What may Project?

Tier projection controlled by

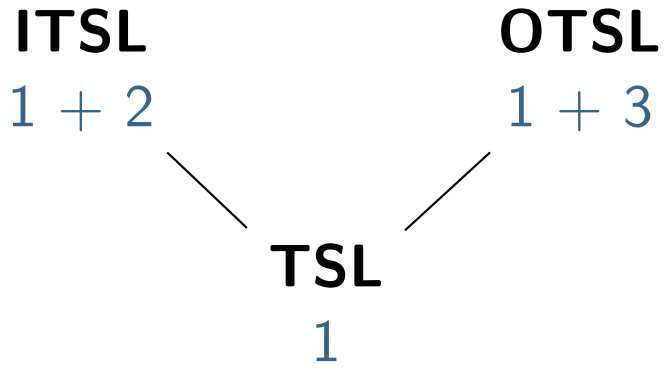
1. label of segment
2. local context
3. symbols already on tier



What may Project?

Tier projection controlled by

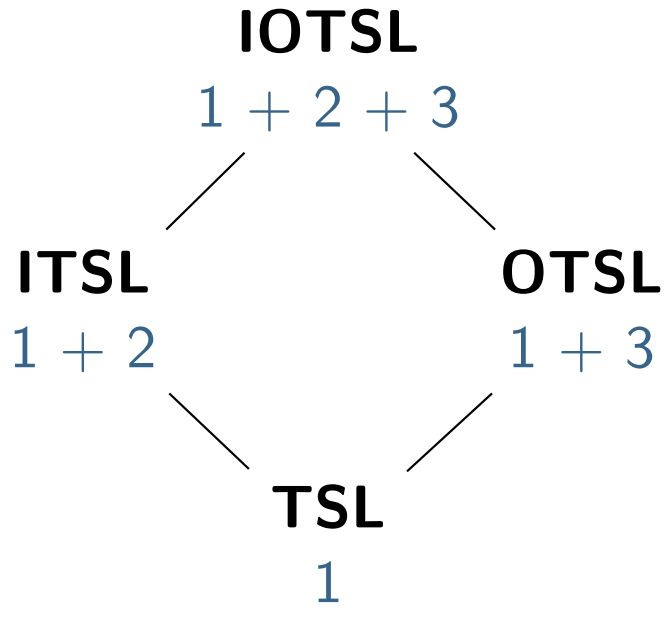
1. label of segment
2. local context
3. symbols already on tier



What may Project?

Tier projection controlled by

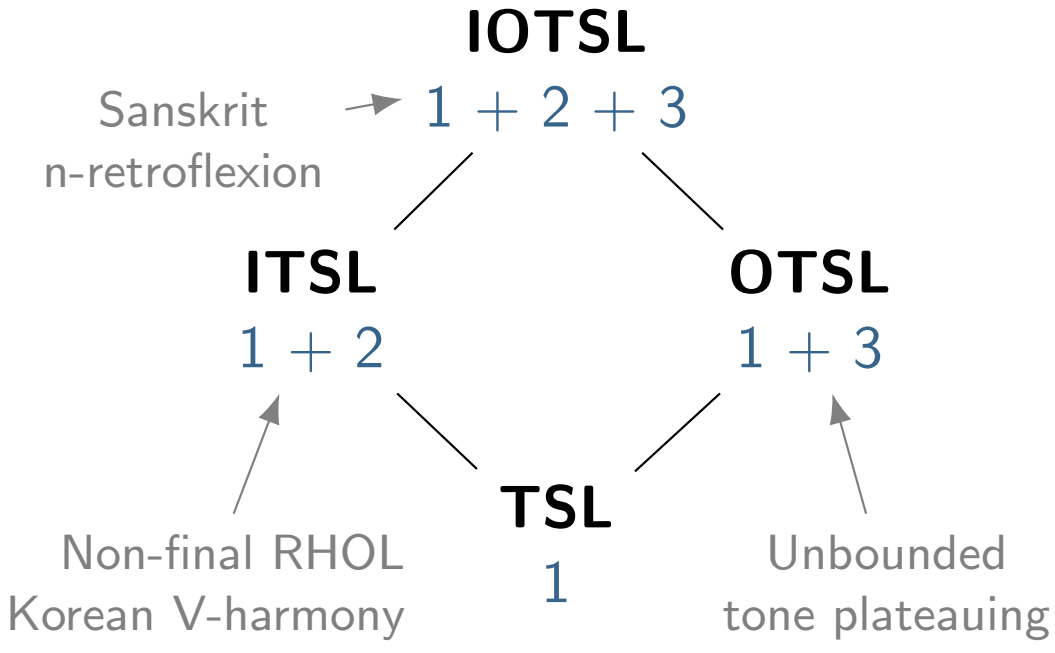
1. label of segment
2. local context
3. symbols already on tier



What may Project?

Tier projection controlled by

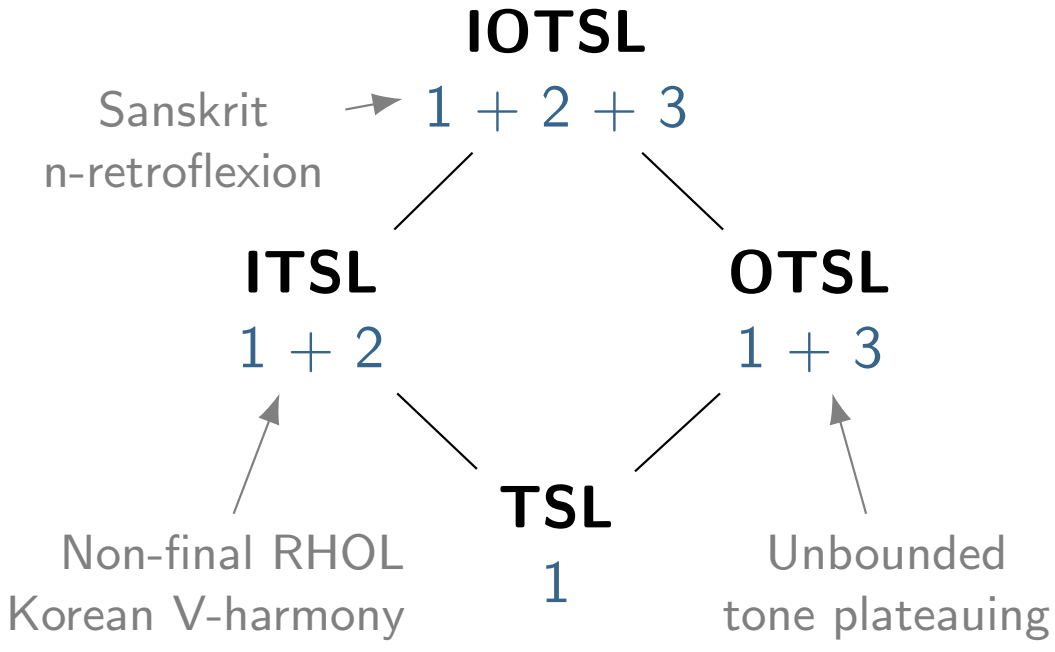
1. label of segment
2. local context
3. symbols already on tier



What may Project?

Tier projection controlled by

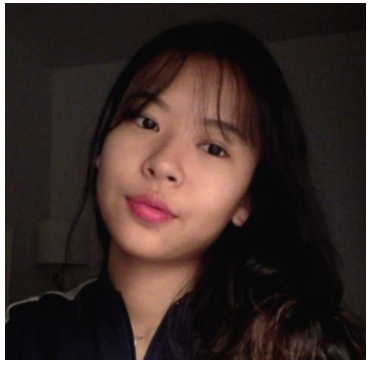
1. label of segment
2. local context
3. symbols already on tier



Hyunah Baek

Aniello De Santo

Connor Mayer



Suji Yang

ITSL: Korean Vowel Harmony

High dark vowels are neutral unless they are initial, in which case they behave like mid dark vowels.

Example: Korean Vowel Harmony with CV-Syllables

1. Project bright vowels (**B**), mid dark (**M**), and initial high dark (**H**)
2. ***BM**, ***MB**, ***HB**

*$ C **H** C **H** C **B** $ $ C **B** C **H** C **B** $

Exercise

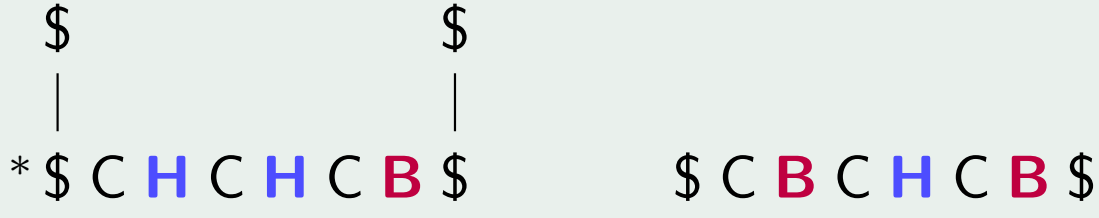
Why is Korean vowel harmony not TSL?

ITSL: Korean Vowel Harmony

High dark vowels are neutral unless they are initial, in which case they behave like mid dark vowels.

Example: Korean Vowel Harmony with CV-Syllables

1. Project bright vowels (**B**), mid dark (**M**), and initial high dark (**H**)
2. *__BM__, *__MB__, *__HB__



Exercise

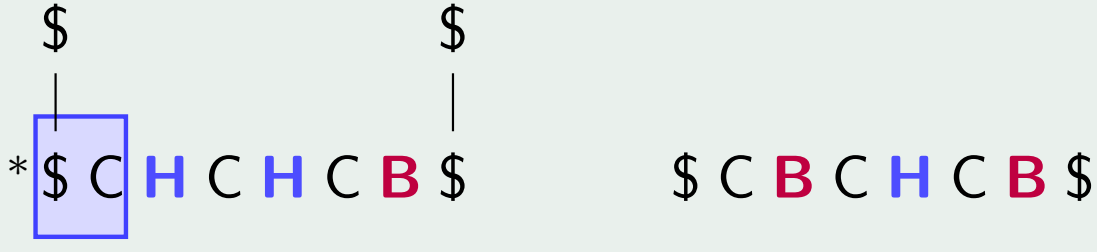
Why is Korean vowel harmony not TSL?

ITSL: Korean Vowel Harmony

High dark vowels are neutral unless they are initial, in which case they behave like mid dark vowels.

Example: Korean Vowel Harmony with CV-Syllables

1. Project bright vowels (**B**), mid dark (**M**), and initial high dark (**H**)
2. ***BM**, ***MB**, ***HB**



Exercise

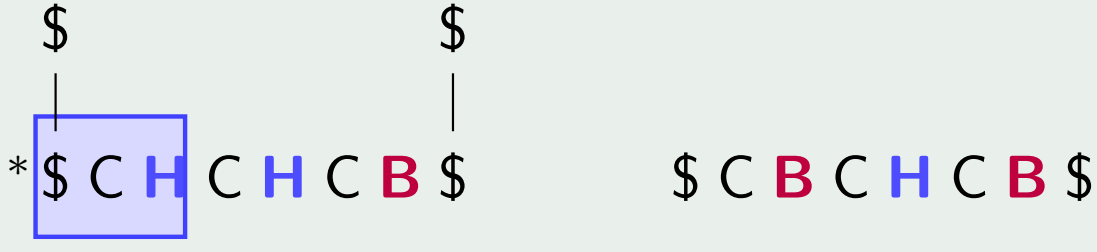
Why is Korean vowel harmony not TSL?

ITSL: Korean Vowel Harmony

High dark vowels are neutral unless they are initial, in which case they behave like mid dark vowels.

Example: Korean Vowel Harmony with CV-Syllables

1. Project bright vowels (**B**), mid dark (**M**), and initial high dark (**H**)
2. ***BM**, ***MB**, ***HB**



Exercise

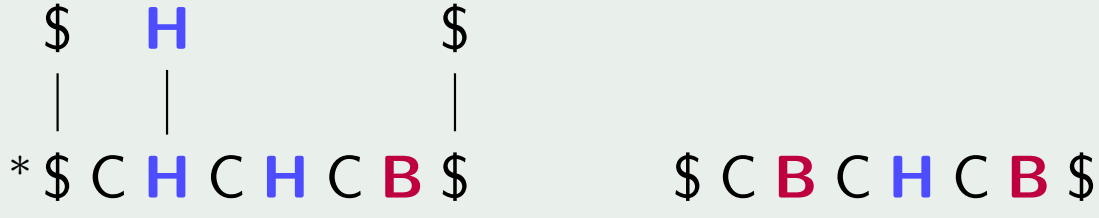
Why is Korean vowel harmony not TSL?

ITSL: Korean Vowel Harmony

High dark vowels are neutral unless they are initial, in which case they behave like mid dark vowels.

Example: Korean Vowel Harmony with CV-Syllables

1. Project bright vowels (**B**), mid dark (**M**), and initial high dark (**H**)
2. *__BM__, *__MB__, *__HB__



Exercise

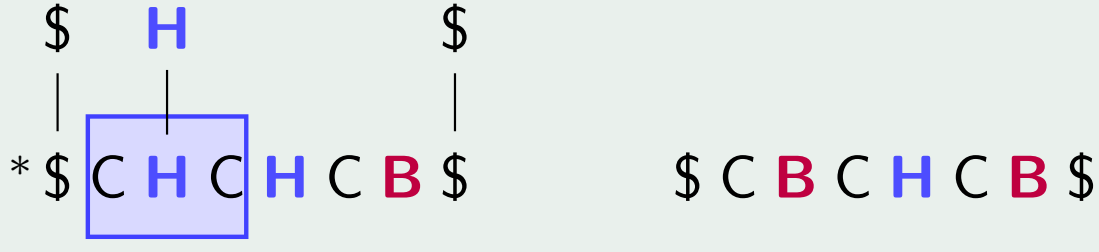
Why is Korean vowel harmony not TSL?

ITSL: Korean Vowel Harmony

High dark vowels are neutral unless they are initial, in which case they behave like mid dark vowels.

Example: Korean Vowel Harmony with CV-Syllables

1. Project bright vowels (**B**), mid dark (**M**), and initial high dark (**H**)
2. ***BM**, ***MB**, ***HB**



Exercise

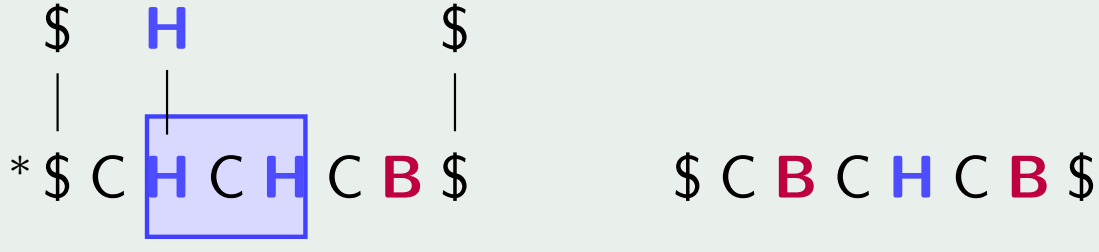
Why is Korean vowel harmony not TSL?

ITSL: Korean Vowel Harmony

High dark vowels are neutral unless they are initial, in which case they behave like mid dark vowels.

Example: Korean Vowel Harmony with CV-Syllables

1. Project bright vowels (**B**), mid dark (**M**), and initial high dark (**H**)
2. ***BM**, ***MB**, ***HB**



Exercise

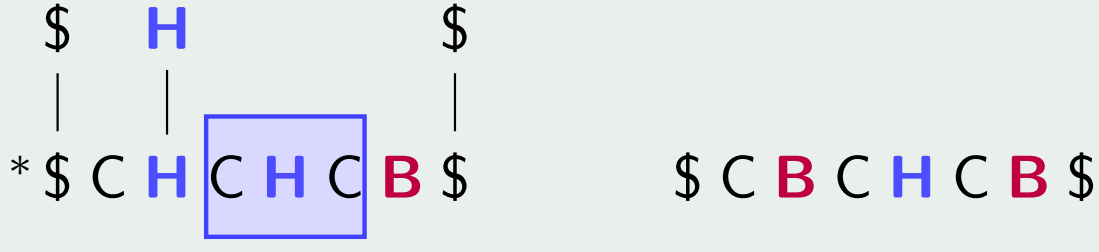
Why is Korean vowel harmony not TSL?

ITSL: Korean Vowel Harmony

High dark vowels are neutral unless they are initial, in which case they behave like mid dark vowels.

Example: Korean Vowel Harmony with CV-Syllables

1. Project bright vowels (**B**), mid dark (**M**), and initial high dark (**H**)
2. ***BM**, ***MB**, ***HB**



Exercise

Why is Korean vowel harmony not TSL?

ITSL: Korean Vowel Harmony

High dark vowels are neutral unless they are initial, in which case they behave like mid dark vowels.

Example: Korean Vowel Harmony with CV-Syllables

1. Project bright vowels (**B**), mid dark (**M**), and initial high dark (**H**)
2. ***BM**, ***MB**, ***HB**

Exercise

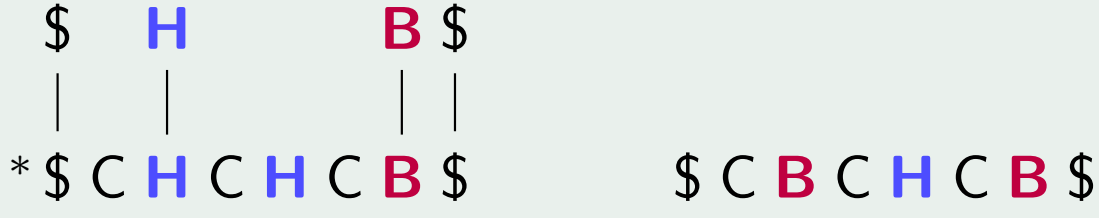
Why is Korean vowel harmony not TSL?

ITSL: Korean Vowel Harmony

High dark vowels are neutral unless they are initial, in which case they behave like mid dark vowels.

Example: Korean Vowel Harmony with CV-Syllables

1. Project bright vowels (**B**), mid dark (**M**), and initial high dark (**H**)
2. *__BM__, *__MB__, *__HB__



Exercise

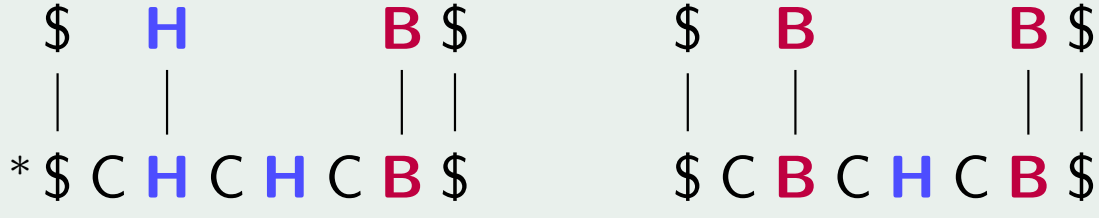
Why is Korean vowel harmony not TSL?

ITSL: Korean Vowel Harmony

High dark vowels are neutral unless they are initial,
in which case they behave like mid dark vowels.

Example: Korean Vowel Harmony with CV-Syllables

1. Project bright vowels (**B**), mid dark (**M**), and initial high dark (**H**)
2. *__BM__, *__MB__, *__HB__



Exercise

Why is Korean vowel harmony not TSL?

OTSL: Unbounded Tone Plateauing

No low tone (**L**) may occur anywhere between two high tones (**H**).

Example: Unbounded Tone Plateauing is OTSL

1. Project **H**, project **L** only if the previous tier symbol is **H**
2. *__HLH__

*$ L **H** **H** **L** L L L **H** **L** $ $ L **H** **H** **L** L L L L L $

Exercise

Why is unbounded tone plateauing not TSL?

Exercise

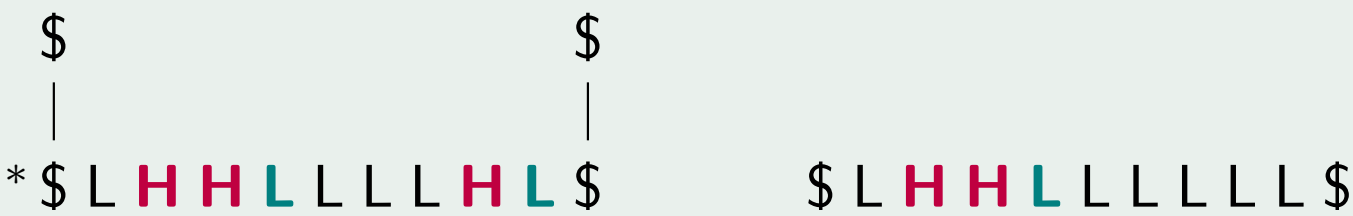
Show that unbounded tone plateauing is also ITSL.

OTSL: Unbounded Tone Plateauing

No low tone (**L**) may occur anywhere between two high tones (**H**).

Example: Unbounded Tone Plateauing is OTSL

1. Project **H**, project **L** only if the previous tier symbol is **H**
2. *__HLH__



Exercise

Why is unbounded tone plateauing not TSL?

Exercise

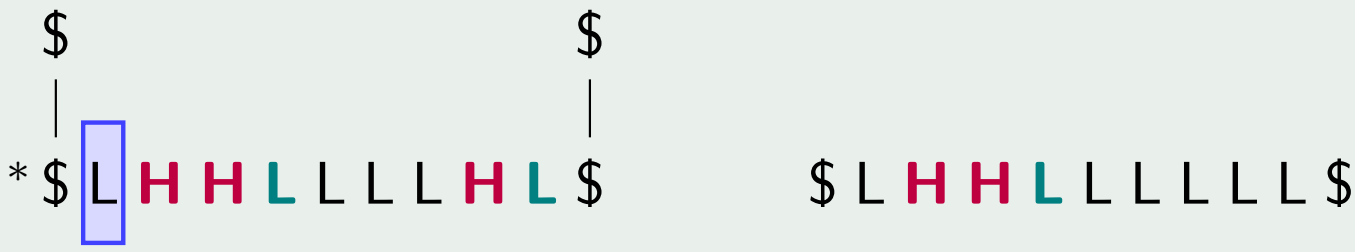
Show that unbounded tone plateauing is also ITSL.

OTSL: Unbounded Tone Plateauing

No low tone (**L**) may occur anywhere between two high tones (**H**).

Example: Unbounded Tone Plateauing is OTSL

1. Project **H**, project **L** only if the previous tier symbol is **H**
2. *__HLH__



Exercise

Why is unbounded tone plateauing not TSL?

Exercise

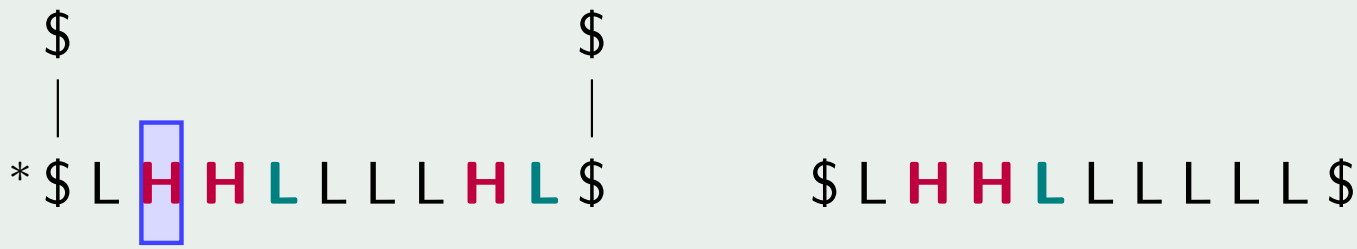
Show that unbounded tone plateauing is also ITSL.

OTSL: Unbounded Tone Plateauing

No low tone (**L**) may occur anywhere between two high tones (**H**).

Example: Unbounded Tone Plateauing is OTSL

1. Project **H**, project **L** only if the previous tier symbol is **H**
2. *__HLH__



Exercise

Why is unbounded tone plateauing not TSL?

Exercise

Show that unbounded tone plateauing is also ITSL.

OTSL: Unbounded Tone Plateauing

No low tone (**L**) may occur anywhere between two high tones (**H**).

Example: Unbounded Tone Plateauing is OTSL

1. Project **H**, project **L** only if the previous tier symbol is **H**
2. *__HLH__

Exercise

Why is unbounded tone plateauing not TSL?

Exercise

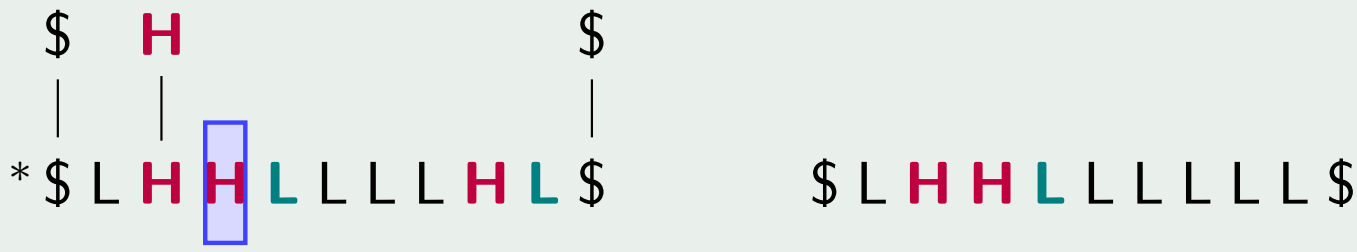
Show that unbounded tone plateauing is also ITSL.

OTSL: Unbounded Tone Plateauing

No low tone (**L**) may occur anywhere between two high tones (**H**).

Example: Unbounded Tone Plateauing is OTSL

1. Project **H**, project **L** only if the previous tier symbol is **H**
2. *__HLH__



Exercise

Why is unbounded tone plateauing not TSL?

Exercise

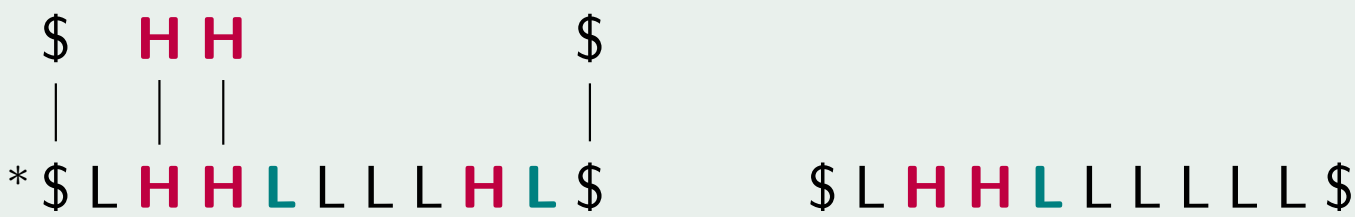
Show that unbounded tone plateauing is also ITSL.

OTSL: Unbounded Tone Plateauing

No low tone (**L**) may occur anywhere between two high tones (**H**).

Example: Unbounded Tone Plateauing is OTSL

1. Project **H**, project **L** only if the previous tier symbol is **H**
2. *__HLH__



Exercise

Why is unbounded tone plateauing not TSL?

Exercise

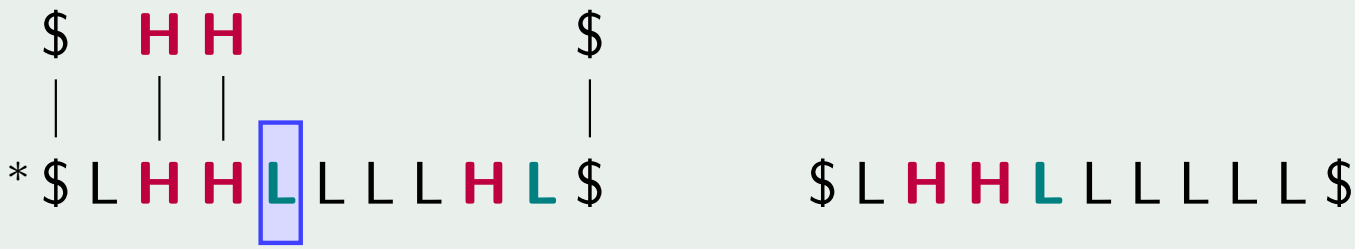
Show that unbounded tone plateauing is also ITSL.

OTSL: Unbounded Tone Plateauing

No low tone (**L**) may occur anywhere between two high tones (**H**).

Example: Unbounded Tone Plateauing is OTSL

1. Project **H**, project **L** only if the previous tier symbol is **H**
2. *__HLH__



Exercise

Why is unbounded tone plateauing not TSL?

Exercise

Show that unbounded tone plateauing is also ITSL.

OTSL: Unbounded Tone Plateauing

No low tone (**L**) may occur anywhere between two high tones (**H**).

Example: Unbounded Tone Plateauing is OTSL

1. Project **H**, project **L** only if the previous tier symbol is **H**
2. *__HLH__

Exercise

Why is unbounded tone plateauing not TSL?

Exercise

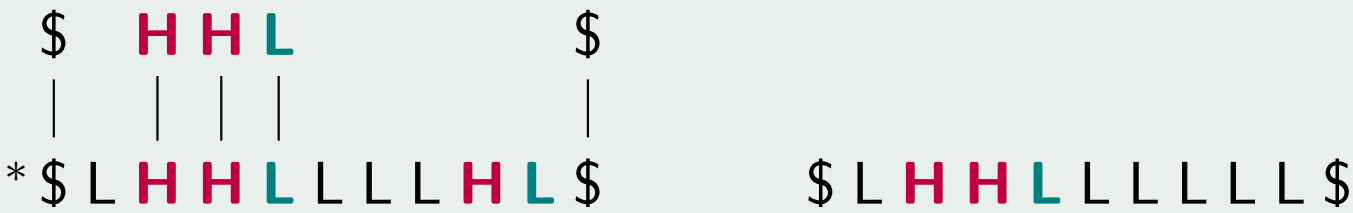
Show that unbounded tone plateauing is also ITSL.

OTSL: Unbounded Tone Plateauing

No low tone (**L**) may occur anywhere between two high tones (**H**).

Example: Unbounded Tone Plateauing is OTSL

1. Project **H**, project **L** only if the previous tier symbol is **H**
2. *__HLH__



Exercise

Why is unbounded tone plateauing not TSL?

Exercise

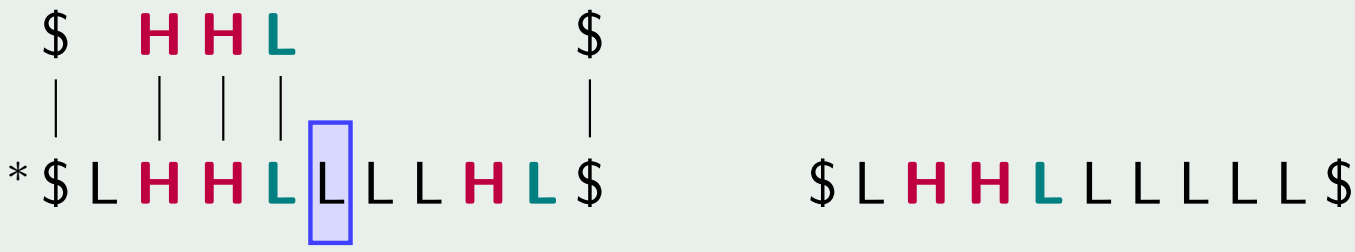
Show that unbounded tone plateauing is also ITSL.

OTSL: Unbounded Tone Plateauing

No low tone (**L**) may occur anywhere between two high tones (**H**).

Example: Unbounded Tone Plateauing is OTSL

1. Project **H**, project **L** only if the previous tier symbol is **H**
2. *__HLH__



Exercise

Why is unbounded tone plateauing not TSL?

Exercise

Show that unbounded tone plateauing is also ITSL.

OTSL: Unbounded Tone Plateauing

No low tone (**L**) may occur anywhere between two high tones (**H**).

Example: Unbounded Tone Plateauing is OTSL

1. Project **H**, project **L** only if the previous tier symbol is **H**
2. *__HLH__

Exercise

Why is unbounded tone plateauing not TSL?

Exercise

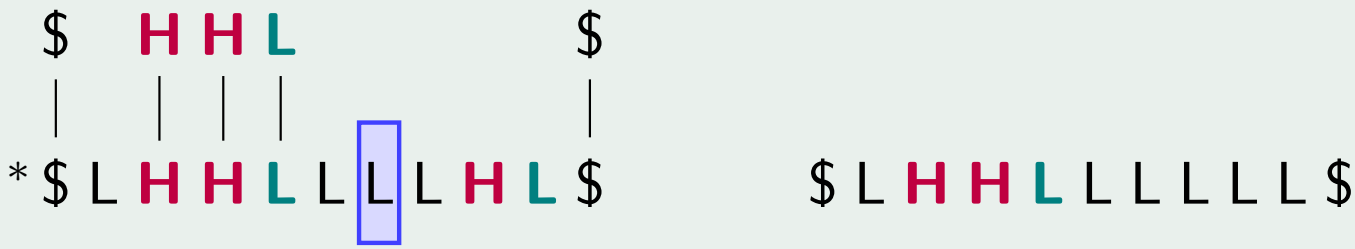
Show that unbounded tone plateauing is also ITSL.

OTSL: Unbounded Tone Plateauing

No low tone (**L**) may occur anywhere between two high tones (**H**).

Example: Unbounded Tone Plateauing is OTSL

1. Project **H**, project **L** only if the previous tier symbol is **H**
2. *__HLH__



Exercise

Why is unbounded tone plateauing not TSL?

Exercise

Show that unbounded tone plateauing is also ITSL.

OTSL: Unbounded Tone Plateauing

No low tone (**L**) may occur anywhere between two high tones (**H**).

Example: Unbounded Tone Plateauing is OTSL

1. Project **H**, project **L** only if the previous tier symbol is **H**
2. *__HLH__

Exercise

Why is unbounded tone plateauing not TSL?

Exercise

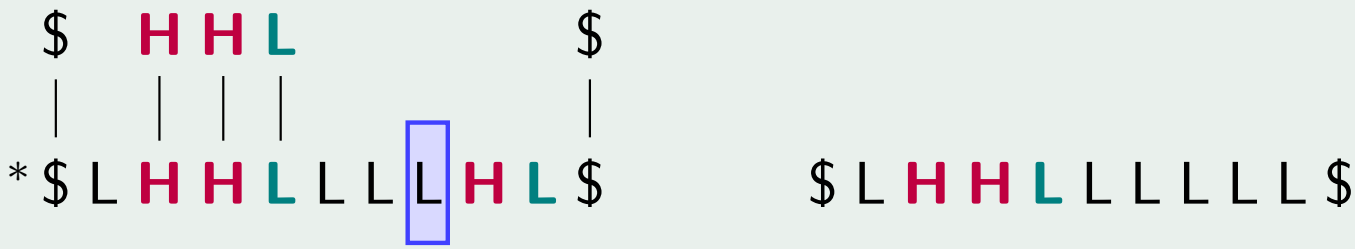
Show that unbounded tone plateauing is also ITSL.

OTSL: Unbounded Tone Plateauing

No low tone (**L**) may occur anywhere between two high tones (**H**).

Example: Unbounded Tone Plateauing is OTSL

1. Project **H**, project **L** only if the previous tier symbol is **H**
2. *__HLH__



Exercise

Why is unbounded tone plateauing not TSL?

Exercise

Show that unbounded tone plateauing is also ITSL.

OTSL: Unbounded Tone Plateauing

No low tone (**L**) may occur anywhere between two high tones (**H**).

Example: Unbounded Tone Plateauing is OTSL

1. Project **H**, project **L** only if the previous tier symbol is **H**
2. *__HLH__

Exercise

Why is unbounded tone plateauing not TSL?

Exercise

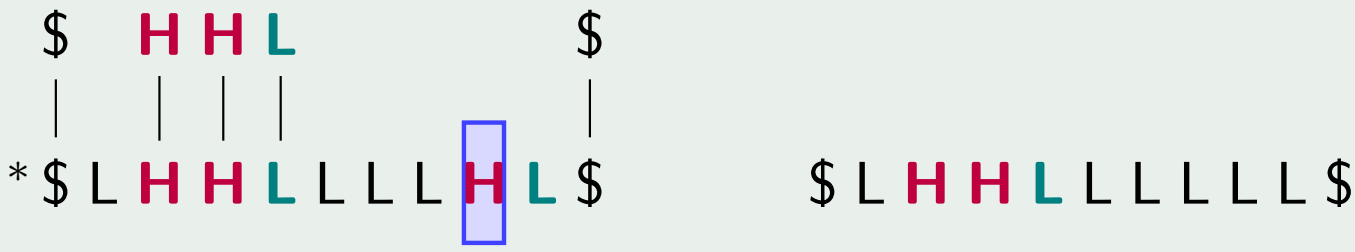
Show that unbounded tone plateauing is also ITSL.

OTSL: Unbounded Tone Plateauing

No low tone (**L**) may occur anywhere between two high tones (**H**).

Example: Unbounded Tone Plateauing is OTSL

1. Project **H**, project **L** only if the previous tier symbol is **H**
2. *__HLH__



Exercise

Why is unbounded tone plateauing not TSL?

Exercise

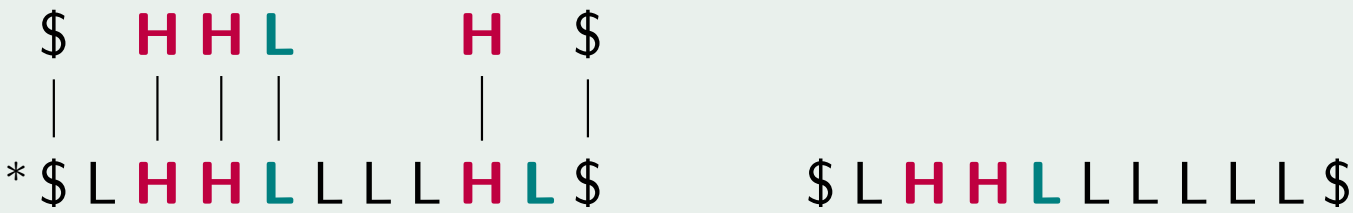
Show that unbounded tone plateauing is also ITSL.

OTSL: Unbounded Tone Plateauing

No low tone (**L**) may occur anywhere between two high tones (**H**).

Example: Unbounded Tone Plateauing is OTSL

1. Project **H**, project **L** only if the previous tier symbol is **H**
2. *__HLH__



Exercise

Why is unbounded tone plateauing not TSL?

Exercise

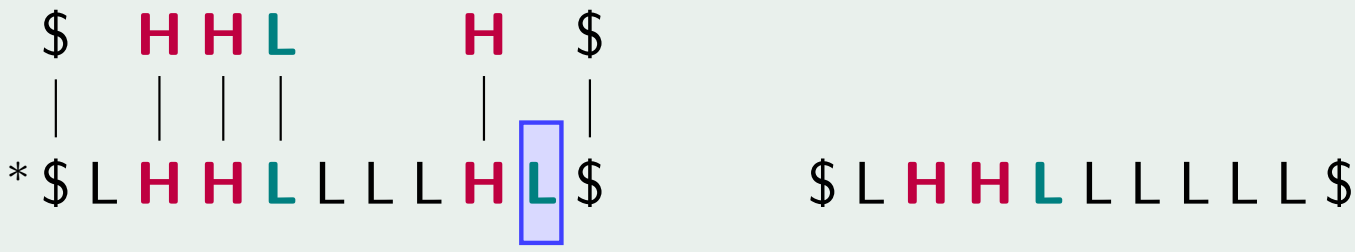
Show that unbounded tone plateauing is also ITSL.

OTSL: Unbounded Tone Plateauing

No low tone (**L**) may occur anywhere between two high tones (**H**).

Example: Unbounded Tone Plateauing is OTSL

1. Project **H**, project **L** only if the previous tier symbol is **H**
2. *__HLH__



Exercise

Why is unbounded tone plateauing not TSL?

Exercise

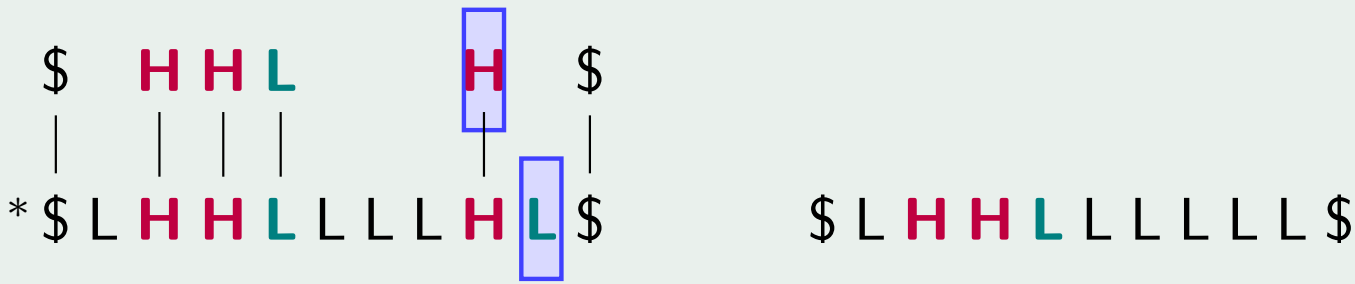
Show that unbounded tone plateauing is also ITSL.

OTSL: Unbounded Tone Plateauing

No low tone (**L**) may occur anywhere between two high tones (**H**).

Example: Unbounded Tone Plateauing is OTSL

1. Project **H**, project **L** only if the previous tier symbol is **H**
2. *__HLH__



Exercise

Why is unbounded tone plateauing not TSL?

Exercise

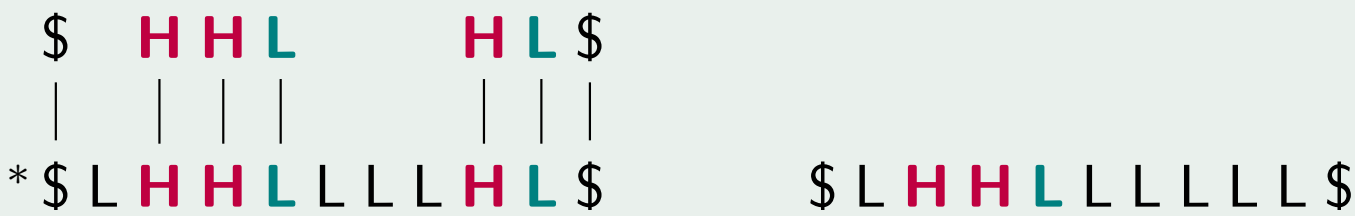
Show that unbounded tone plateauing is also ITSL.

OTSL: Unbounded Tone Plateauing

No low tone (**L**) may occur anywhere between two high tones (**H**).

Example: Unbounded Tone Plateauing is OTSL

1. Project **H**, project **L** only if the previous tier symbol is **H**
2. *__HLH__



Exercise

Why is unbounded tone plateauing not TSL?

Exercise

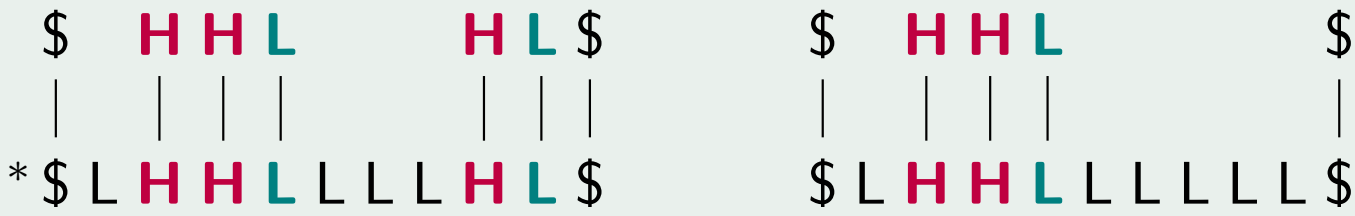
Show that unbounded tone plateauing is also ITSL.

OTSL: Unbounded Tone Plateauing

No low tone (**L**) may occur anywhere between two high tones (**H**).

Example: Unbounded Tone Plateauing is OTSL

1. Project **H**, project **L** only if the previous tier symbol is **H**
2. *__HLH__



Exercise

Why is unbounded tone plateauing not TSL?

Exercise

Show that unbounded tone plateauing is also ITSL.

The Full Monty: IOTSL

- IOTSL uses a tier projection that considers both the structural context and the previous **n** tier symbols.
- Only a few phenomena in phonology seem to require IOTSL:
  - Sanskrit n-retroflexion (aka *nati*) (Graf and Mayer 2018)
  - Uyghur backness harmony (Mayer and Major 2018)
- **Spoiler:** IOTSL shows up quite a bit in syntax.

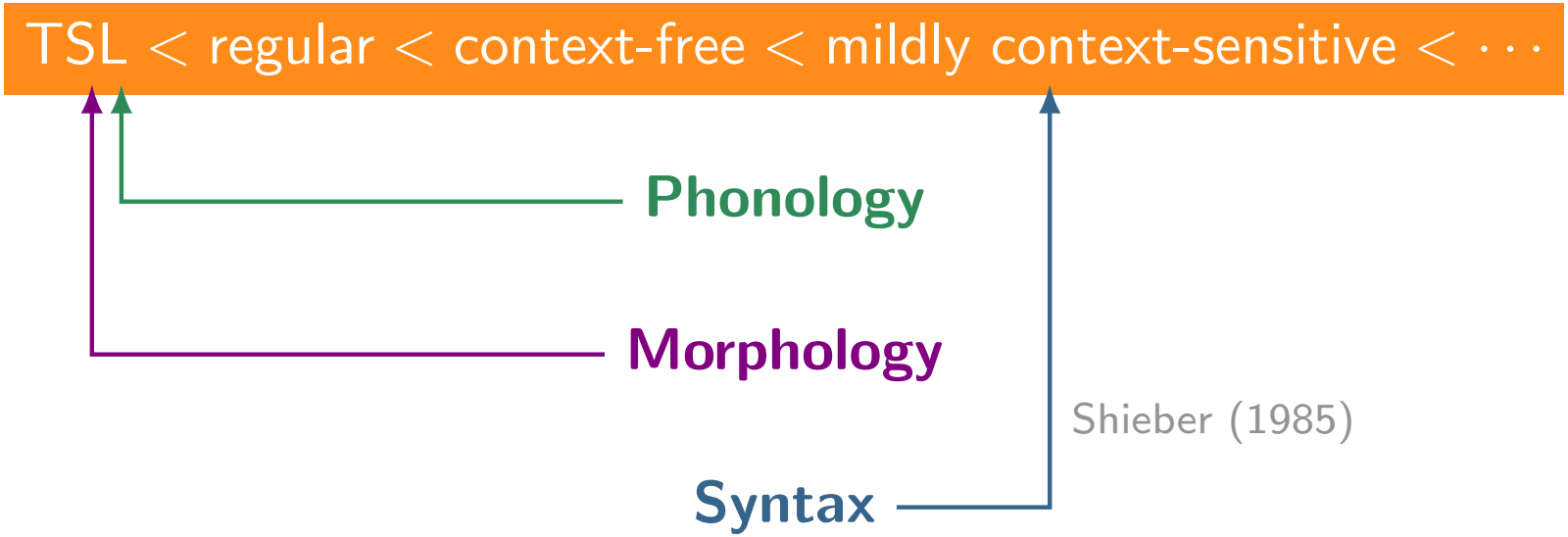
Interim Summary

SL local dependencies

TSL local dependencies over tiers (ITSL, OTSL, IOTSL use more powerful tier projections); can also handle blocking

- We can model various phenomena over strings and classify them by their complexity.
- This is not a pointless mathematical exercise, it is **useful and insightful**:
  - separate attested from unattested phenomena,
  - explain expressivity boundary of lexicalized quantifiers,
  - learning algorithms,
  - guide data gathering.
- But imho, things get really interesting once we apply these ideas to syntax...

Could Syntax Also be Subregular?



- Syntax seems even more like an outlier. . .
- Don't look at surface strings!
  What about **syntactic dependencies**?

Nazila Shafiei

c-Strings

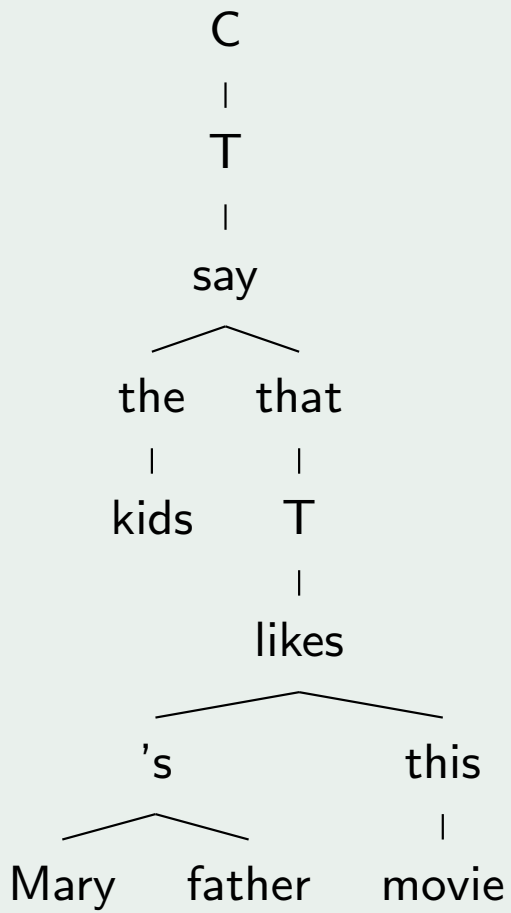
Command-Strings

The **c[ommand]-string** of a node **n** contains

- **n** itself and
- every node that c-commands **n**.

- easily computed from dependency trees
- C-command constraints seem to be largely **IOTSL over c-strings**.

Example



c-Strings

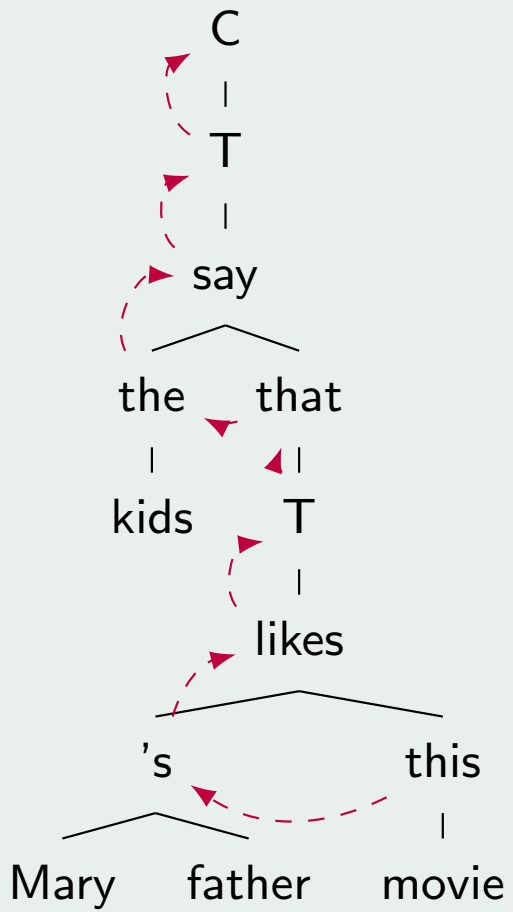
Command-Strings

The **c[ommand]-string** of a node **n** contains

- **n** itself and
- every node that c-commands **n**.

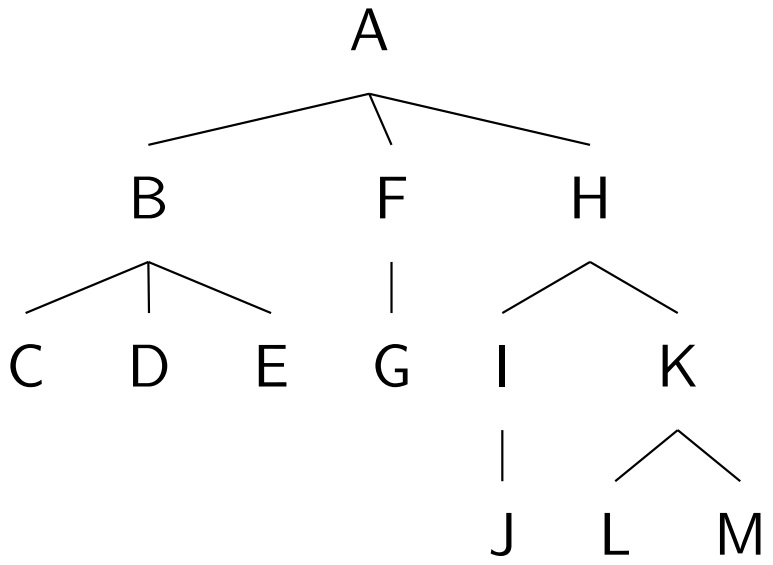
- easily computed from dependency trees
- C-command constraints seem to be largely **IOTSL over c-strings**.

Example



this 's likes T that the say T C

Exercise: Computing c-strings



Exercise

Compute the command string of

1. E
2. G
3. M
4. A

Principle A

Principle A (as a distributional constraint)

Every reflexive must be c-commanded by a DP in the same TP.

Equivalent c-String Constraint

Verify for each node: If its c-string starts with a reflexive, then at least one D must occur before the first T.

TSL Strategy for Principle A

1. Always project first symbol (ITSL)
2. Project D/T if previous symbol is Refl (OTSL)
3. Constraint: *Refl T (bigram)

Principle A

Principle A (as a distributional constraint)

Every reflexive must be c-commanded by a DP in the same TP.

Equivalent c-String Constraint

Verify for each node: If its c-string starts with a reflexive, then at least one D must occur before the first T.

TSL Strategy for Principle A

1. Always project first symbol (ITSL)
2. Project D/T if previous symbol is Refl (OTSL)
3. Constraint: *Refl T (bigram)

Principle A

Principle A (as a distributional constraint)

Every reflexive must be c-commanded by a DP in the same TP.

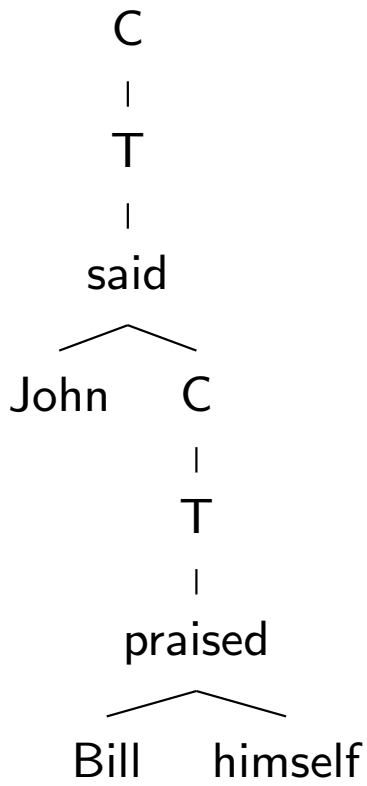
Equivalent c-String Constraint

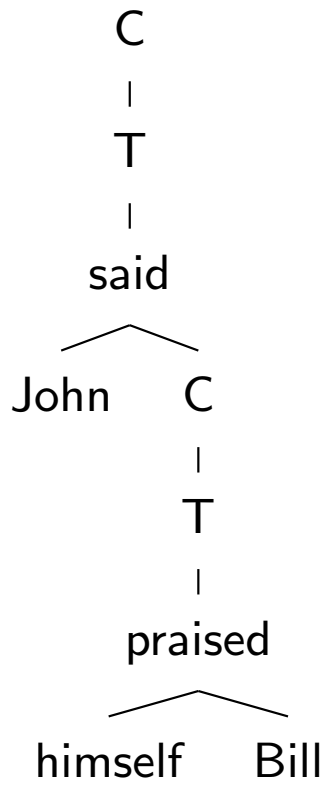
Verify for each node: If its c-string starts with a reflexive, then at least one D must occur before the first T.

TSL Strategy for Principle A

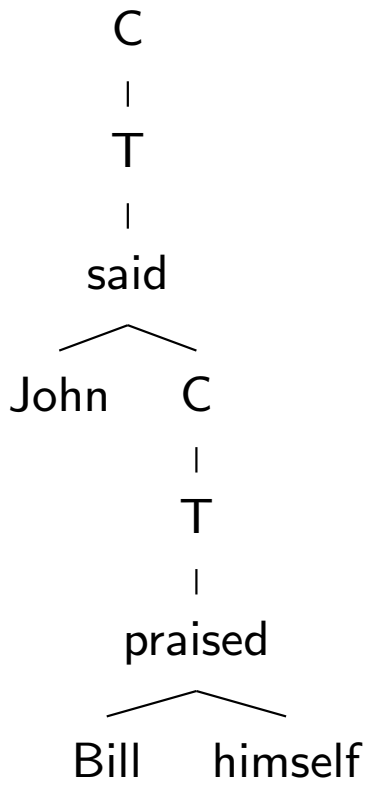
1. Always project first symbol (ITSL)
2. Project D/T if previous symbol is Refl (OTSL)
3. Constraint: *Refl T (bigram)

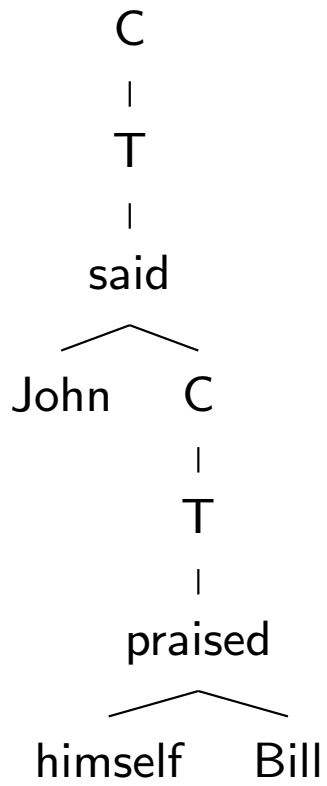
Example of Principle A as a TSL Constraint

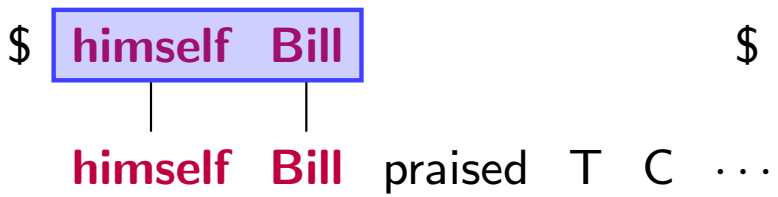




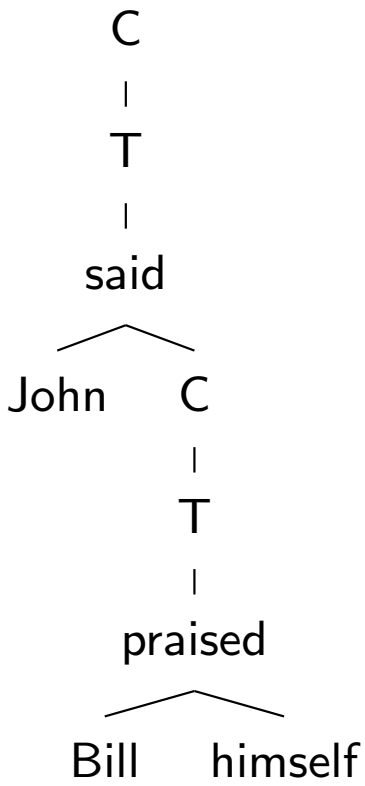
Example of Principle A as a TSL Constraint

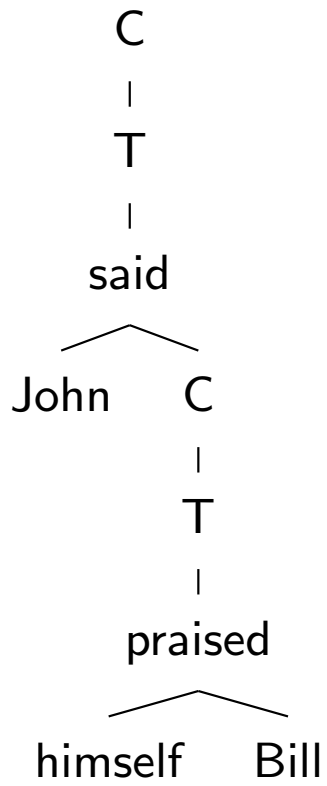


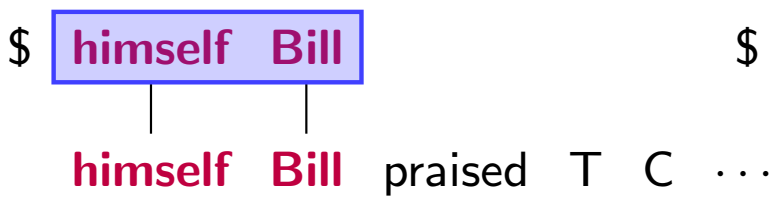


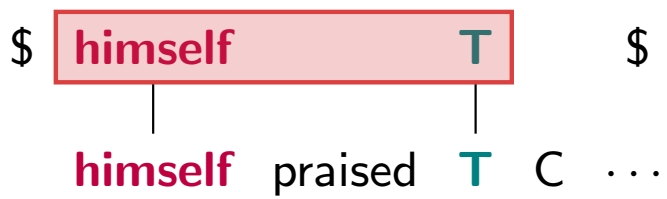


Example of Principle A as a TSL Constraint









Another Example: Swedish *SE*

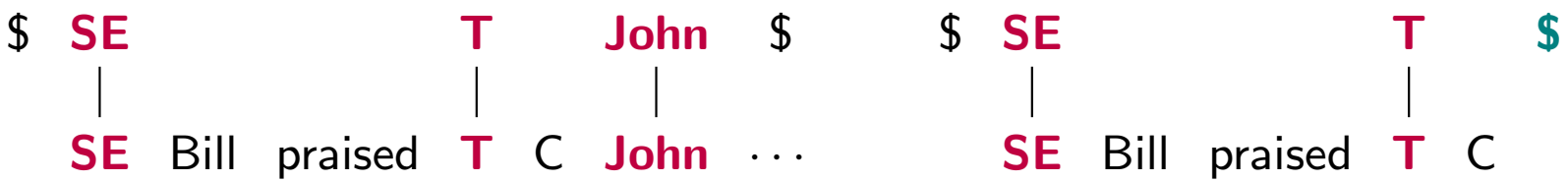
- Swedish *SE*-reflexive must be non-locally bound.

(2) a. John said Bill praised SE.

b. * Bill praised SE.

TSL Strategy for *SE*

1. Always project first symbol (ITSL)
2. Project T if previous symbol is SE (OTSL)
3. Project D if previous symbol is T (OTSL)
4. Constraint: ***SE T $** (trigram)



Another Example: Swedish *SE*

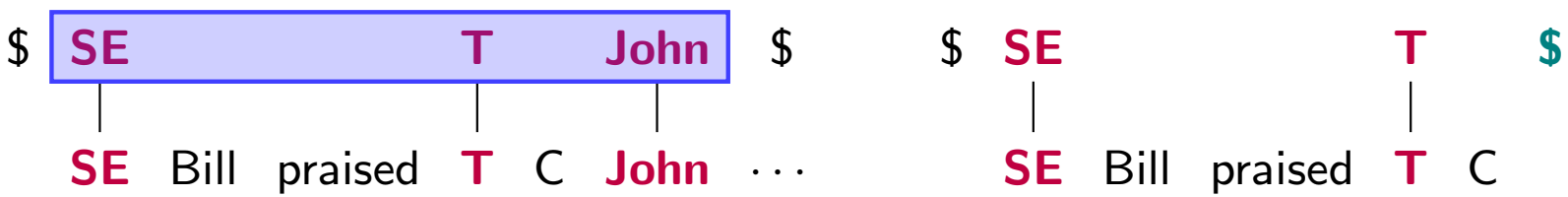
- Swedish *SE*-reflexive must be non-locally bound.

(2) a. John said Bill praised SE.

b. * Bill praised SE.

TSL Strategy for *SE*

1. Always project first symbol (ITSL)
2. Project T if previous symbol is SE (OTSL)
3. Project D if previous symbol is T (OTSL)
4. Constraint: ***SE T $** (trigram)



Another Example: Swedish *SE*

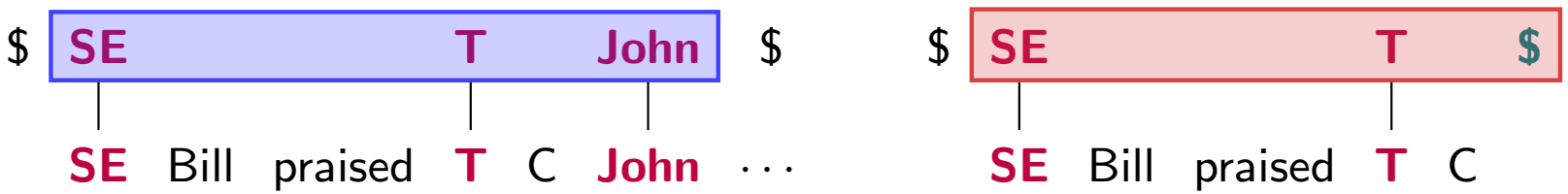
- Swedish *SE*-reflexive must be non-locally bound.

(2) a. John said Bill praised SE.

b. * Bill praised SE.

TSL Strategy for *SE*

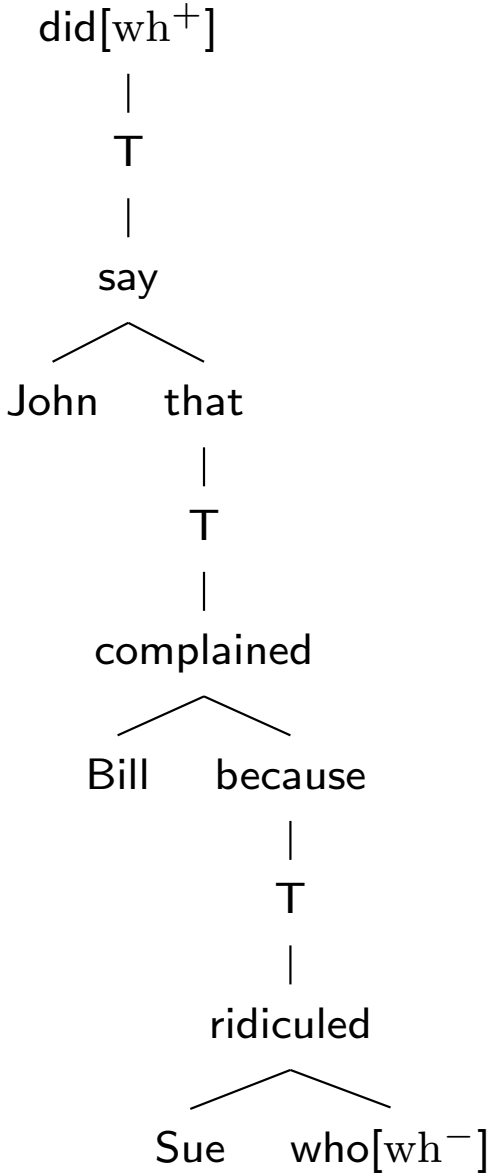
1. Always project first symbol (ITSL)
2. Project T if previous symbol is SE (OTSL)
3. Project D if previous symbol is T (OTSL)
4. Constraint: ***SE T $** (trigram)



Hey, You Got Your Phonology in My Syntax

- Binding actually looks a lot like long-distance consonant dissimilation. (Ikawa and Jardine 2020)
- And two phenomena look remarkably like unbounded tone plateauing:
  - island effects
  - wh-agreement in complementizers.

Path Constraints on Movement



- **C-string of who**
  who$[\mathrm{wh}^-]$ $\cdots$ because $\cdots$ that $\cdots$ did$[\mathrm{wh}^+]$
- **Adjunct island constraint**
  No adjunct head may occur within an interval spanned by f^- and f^+.
- **Complementizer agreement**
  No complementizer that lacks wh-agreement may occur within an interval spanned by wh^- and wh^+.

Wrapping up String Land

- **Cognitive Parallelism**
  - Phonology and syntax are surprisingly similar.
  - Some aspects of syntax can be studied via strings.
  - SL and TSL play a central role in both.
- **Specific Phenomena**
  - unbounded tone plateauing $\approx$ islands, wh-agreement
  - lexically simplex quantifiers are TSL

Join the Program

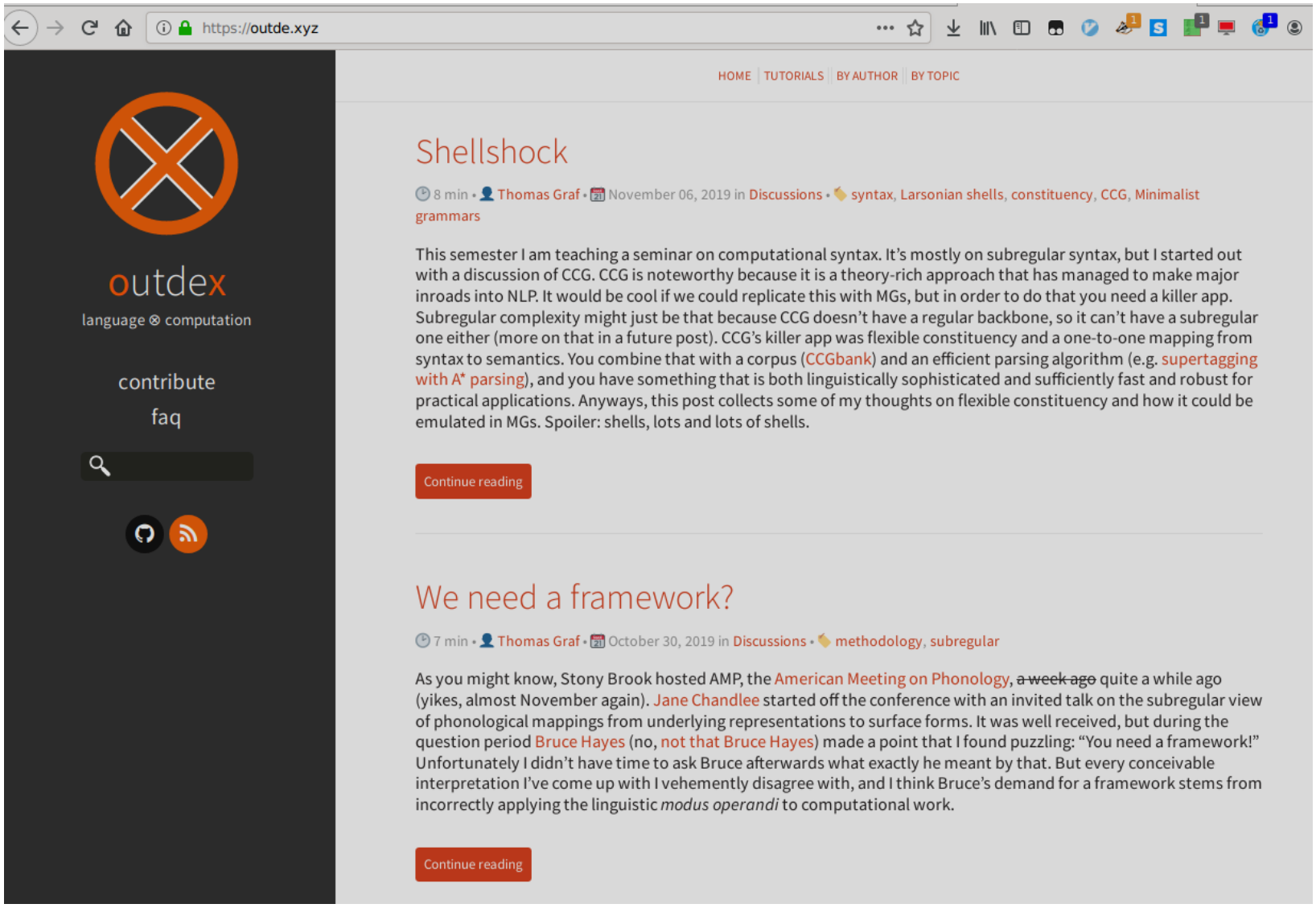
Everybody has something to contribute!

- Do you have data that contradicts current predictions?
- In-depth analysis of specific phenomena
- Grammar fragments

Follow-Up Projects

1. Temporal semantics (Fernando 2015)
2. Weighted and probabilistic versions (Mayer 2021)
3. SL-analysis of *that*-trace filter
4. C-string syntax (in particular movement-feeding/bleeding)

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



Learning More About String Land

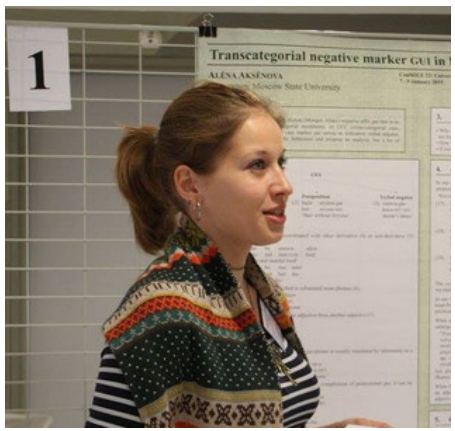
1. **Survey papers:** Pullum and Rogers (2006); Heinz (2011a,b, 2018); Rogers and Pullum (2011); Chandlee and Heinz (2016)
2. **TSL and its extensions:** Heinz et al. (2011); McMullin (2016); Baek (2018); De Santo (2017); De Santo and Graf (2019); Graf (2017); Graf and Mayer (2018); Mayer and Major (2018); 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 (2018); 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)

Acknowledgments

This work was supported by the National Science Foundation under Grant No. BCS-1845344.

Appendix

TSL Morphology



Alëna Aksënova

Sophie Moradi

- Joint work with Alëna Aksënova and Sophie Moradi.
- It seems that **morphotactics is also TSL**.
  (Aksënova et al. 2016)

Example: Unbounded *the day after*-Prefixation in German

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

Example

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

TSL Description

Tier: **über**, stem boundary +

Constraint	**Bigrams**
über must be prefix	*+ **über**

Example: Unbounded *the day after*-Prefixation in German

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

Example

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

TSL Description

Tier: **über**, stem boundary +

Constraint	**Bigrams**
über must be prefix	*+ **über**

Example: Unbounded *the day after*-Prefixation in German

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

Example

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

TSL Description

Tier: **über**, stem boundary +

Constraint	**Bigrams**
über must be prefix	*+ **über**

$ **über** **über** + + **über** $
| | | | | | | |
$ **über** **über** + *morgen* + **über** $

Example: Bounded *the day after*-Circumfixation in Ilocano

- Ilocano has a circumfix **ka- -an**.
- This prefix can be combined once with *bigát* 'tomorrow'.

Example

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

TSL Description

Tier: *ka*, *an*, stem boundary +

Constraint	**Bigrams**
ka must be prefix	*+ **ka**
an must be suffix	***an** +
ka before **an**	***an ka**
no iteration	***ka ka**, ***an an**
no lonely affix	***ka** ++ $, *$++ **an**

Example: Bounded *the day after*-Circumfixation in Ilocano

- Ilocano has a circumfix **ka- -an**.
- This prefix can be combined once with *bigát* 'tomorrow'.

Example

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

TSL Description

Tier: *ka*, *an*, stem boundary +

Constraint	**Bigrams**
ka must be prefix	*+ **ka**
an must be suffix	***an** +
ka before **an**	***an ka**
no iteration	***ka ka**, ***an an**
no lonely affix	***ka** ++ $, *$++ **an**

Example: Bounded *the day after*-Circumfixation in Ilocano

- Ilocano has a circumfix **ka- -an**.
- This prefix can be combined once with *bigát* 'tomorrow'.

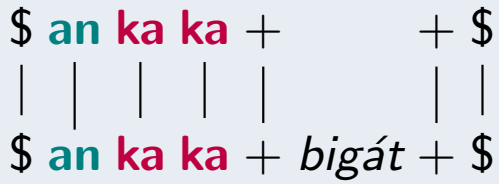
Example

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

TSL Description

Tier: *ka*, *an*, stem boundary +

Constraint	**Bigrams**
ka must be prefix	*+ **ka**
an must be suffix	***an** +
ka before **an**	***an ka**
no iteration	***ka ka**, ***an an**
no lonely affix	***ka** ++ $, *$++ **an**



Typological Gap: No Unbounded Circumfixation

- There seems to be no language with an affix that is
  - freely iterable like German **über**, and
  - a circumfix like **ka- -an** in Ilocano.
- Why this gap? Because the **result would not be TSL**!

Explanation

- The pattern would be $\mathbf{ka}^{\mathbf{n}}$+*bigát*+$\mathbf{an}^{\mathbf{n}}$.
- TSL cannot memorize exact numbers.
- All affixes would have to be visible in the same search window.
- But the window's size is bounded, while the pattern is not.

Typological Gap: No Unbounded Circumfixation

- There seems to be no language with an affix that is
  - freely iterable like German **über**, and
  - a circumfix like **ka- -an** in Ilocano.
- Why this gap? Because the **result would not be TSL**!

Explanation

- The pattern would be $\mathbf{ka}^{\mathbf{n}}$+*bigát*+$\mathbf{an}^{\mathbf{n}}$.
- TSL cannot memorize exact numbers.
- All affixes would have to be visible in the same search window.
- But the window's size is bounded, while the pattern is not.

References I

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`.

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

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.

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`.

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.

References II

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.

Fernando, Tim. 2015. The semantics of tense and aspect: a finite-state perspective. In *The handbook of contemporary semantic theory, second edition*, ed. Shalom Lappin and Chris Fox, 203–236. Wiley.

Goldsmith, John. 1976. *Autosegmental phonology*. Doctoral Dissertation, MIT.

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

Graf, Thomas. 2018. 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 III

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

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.

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

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`.

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.

References IV

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.

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.

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.

References V

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.

Jurgec, Peter. 2011. *Feature spreading 2.0: A unified theory of assimilation*. Doctoral Dissertation, University of Tromsø.

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.

Karttunen, Lauri, Ronald M. Kaplan, and Annie Zaenen. 1992. Two-level morphology with composition. In *COLING'92*, 141–148. URL http://www.aclweb.org/anthology/C92-1025.

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.

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

References VI

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.

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

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

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

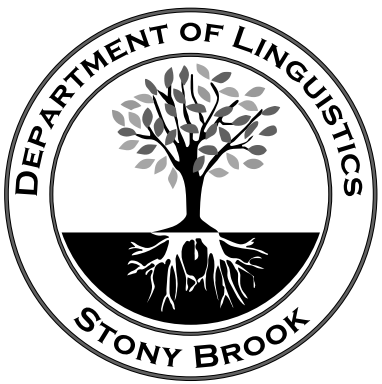
Shafiei, Nazila, and Thomas Graf. 2020. The subregular complexity of syntactic islands. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2020*, 272–281.

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`.

Subregular Linguistics for. . . well, Linguists

(Part 2: Tree Land)

Thomas Graf



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

MIT Mini Course
May 5 & 6, 2021

Outline

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

checks string for illicit substrings

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

checks string for illicit substrings

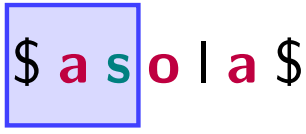
* $ **a s o** l **a** $

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

checks string for illicit substrings



SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

checks string for illicit substrings

* $ **a s o** l **a** $

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $ $ **a** + **s** **o** c **i** **a** l **e** $

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

checks string for illicit substrings

* $ a s o l a $ $ a z o l a $ $ a + s o c i a l e $

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $ $ **a** + **s** **o** c **i** **a** l **e** $

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $ $ **a** + **s** **o** c **i** **a** l **e** $

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $ $ **a** + **s** **o** c **i** **a** l **e** $

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $ $ **a** + **s** **o** c **i** **a** l **e** $

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $ $ **a** + **s** **o** c **i** **a** l **e** $

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $ $ **a** + **s** **o** c **i** **a** l **e** $

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $ $ **a** + **s** **o** c **i** **a** l **e** $

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $ $ **a** + **s** **o** c **i** **a** l **e** $

SL-**n** over trees

sliding window of **depth n**

checks tree for illicit subtrees

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

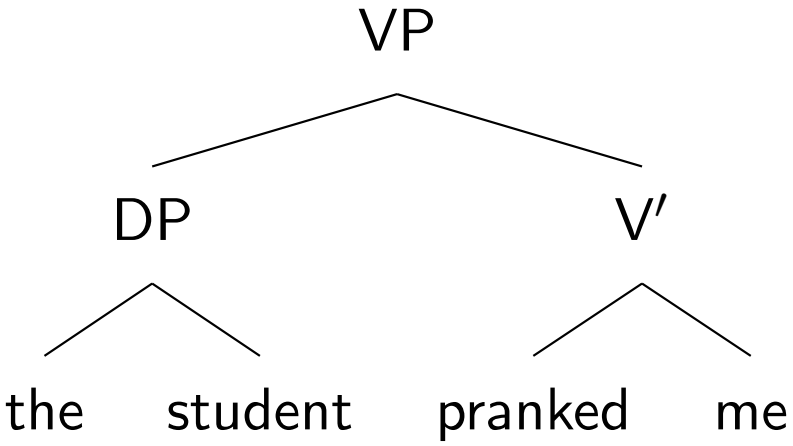
checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $ $ **a** + **s** **o** c **i** **a** l **e** $

SL-**n** over trees

sliding window of **depth n**

checks tree for illicit subtrees



SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

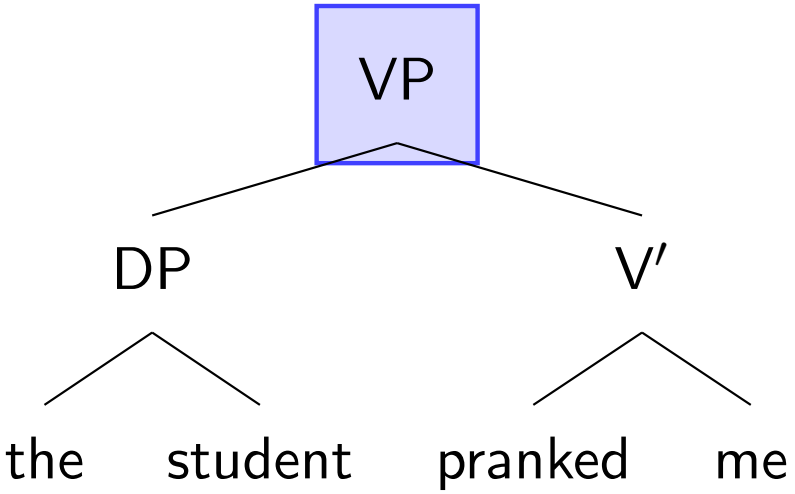
checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $ $ **a** + **s** **o** c **i** **a** l **e** $

SL-**n** over trees

sliding window of **depth n**

checks tree for illicit subtrees



SL-1

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

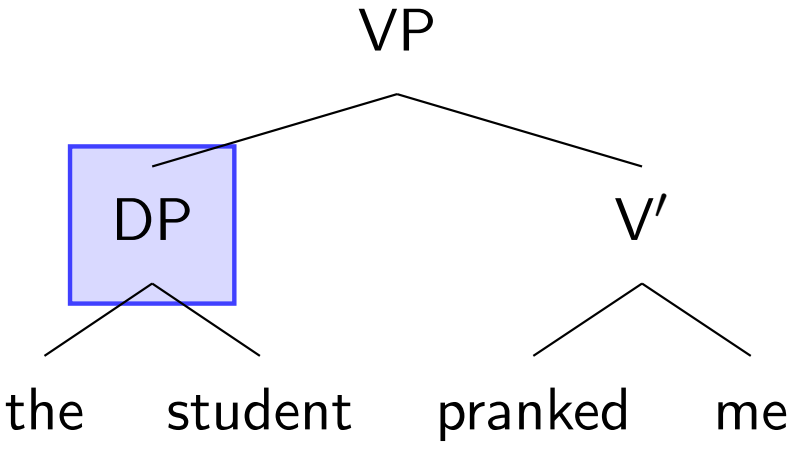
checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $ $ **a** + **s** **o** c **i** **a** l **e** $

SL-**n** over trees

sliding window of **depth n**

checks tree for illicit subtrees



SL-1

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

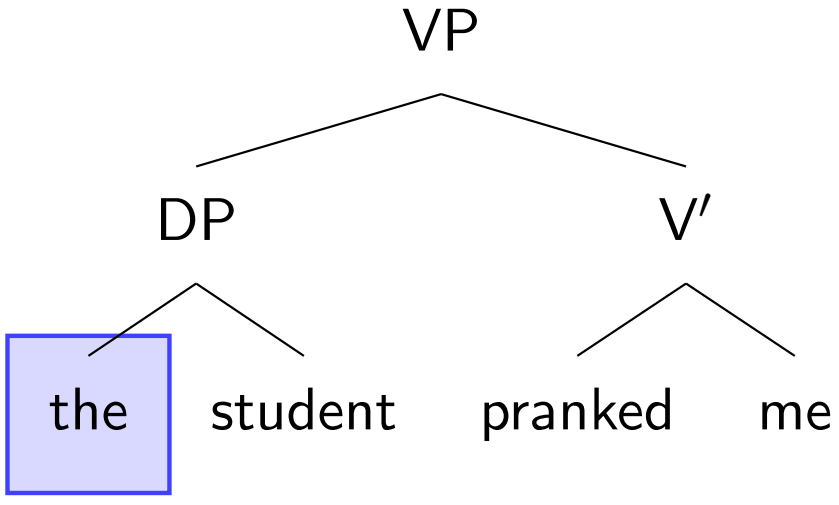
checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $ $ **a** + **s** **o** c **i** **a** l **e** $

SL-**n** over trees

sliding window of **depth n**

checks tree for illicit subtrees



SL-1

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

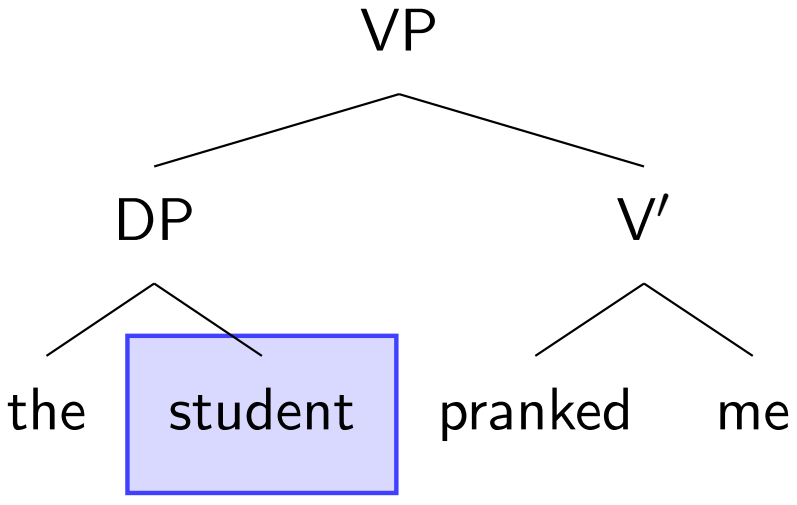
checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $ $ **a** + **s** **o** c **i** **a** l **e** $

SL-**n** over trees

sliding window of **depth n**

checks tree for illicit subtrees



SL-1

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

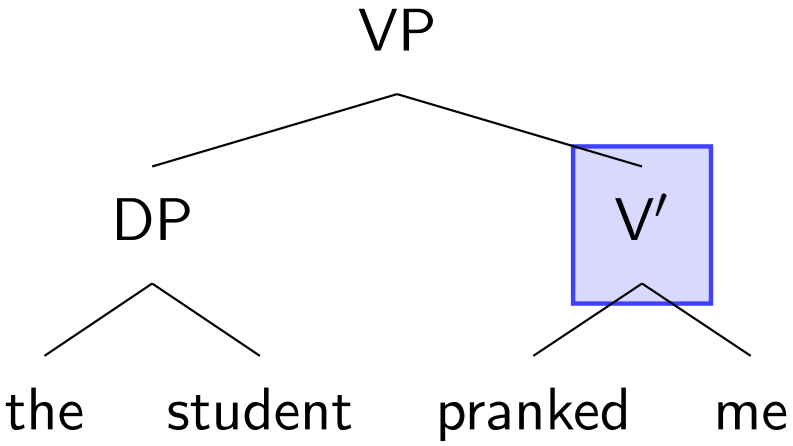
checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $ $ **a** + **s** **o** c **i** **a** l **e** $

SL-**n** over trees

sliding window of **depth n**

checks tree for illicit subtrees



SL-1

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

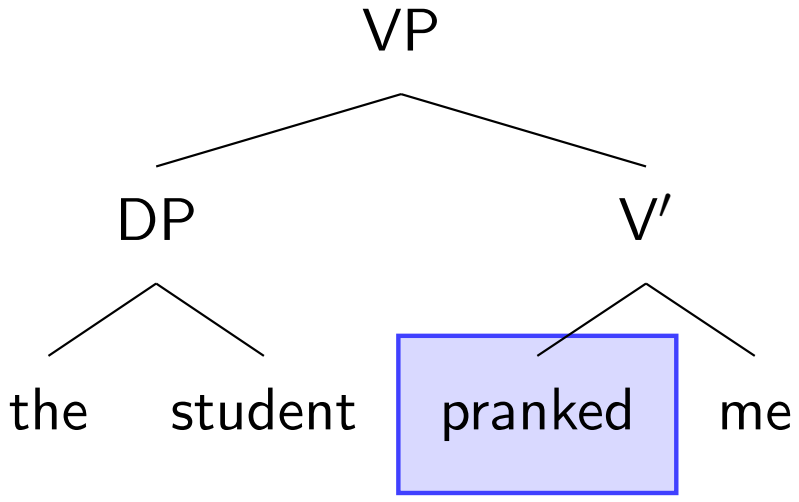
checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $ $ **a** + **s** **o** c **i** **a** l **e** $

SL-**n** over trees

sliding window of **depth n**

checks tree for illicit subtrees



SL-1

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

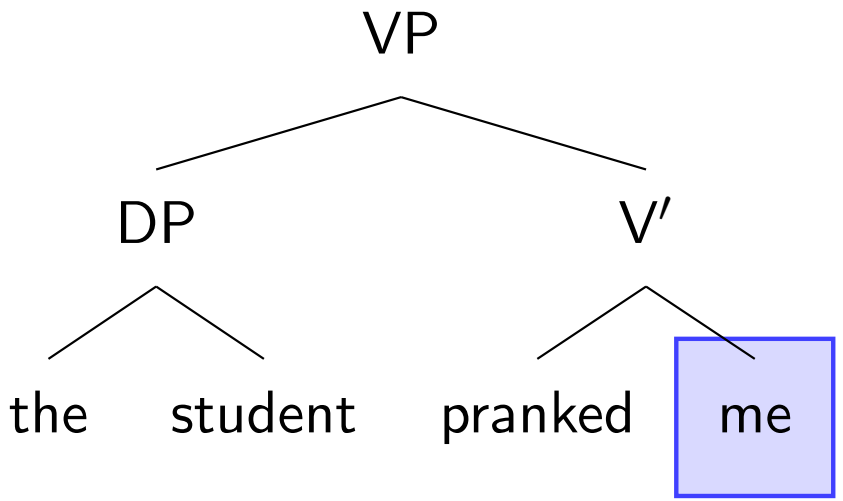
checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $ $ **a** + **s** **o** c **i** **a** l **e** $

SL-**n** over trees

sliding window of **depth n**

checks tree for illicit subtrees



SL-1

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

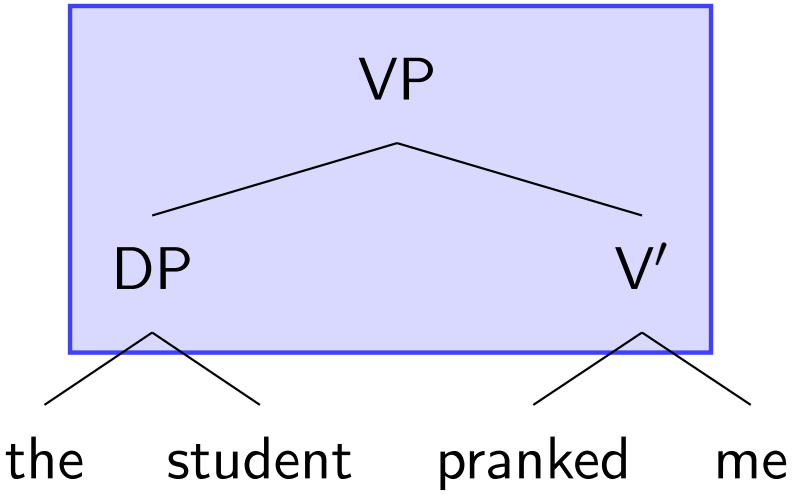
checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $ $ **a** + **s** **o** c **i** **a** l **e** $

SL-**n** over trees

sliding window of **depth n**

checks tree for illicit subtrees



SL-2

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

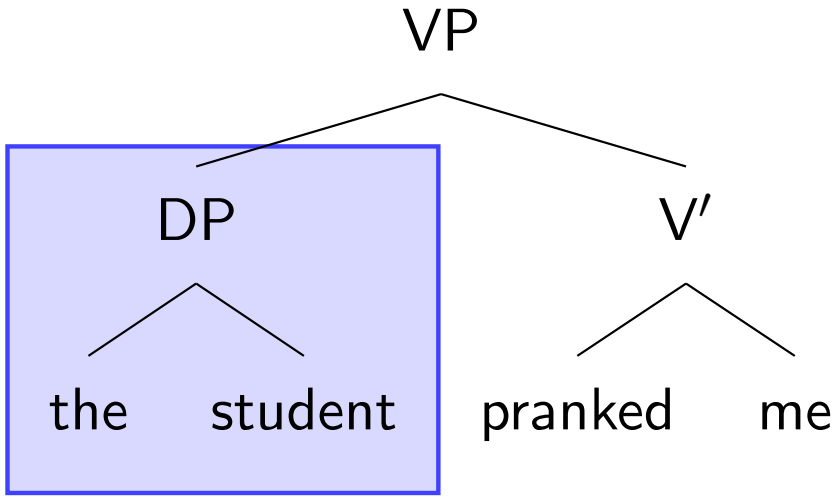
checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $ $ **a** + **s** **o** c **i** **a** l **e** $

SL-**n** over trees

sliding window of **depth n**

checks tree for illicit subtrees



SL-2

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

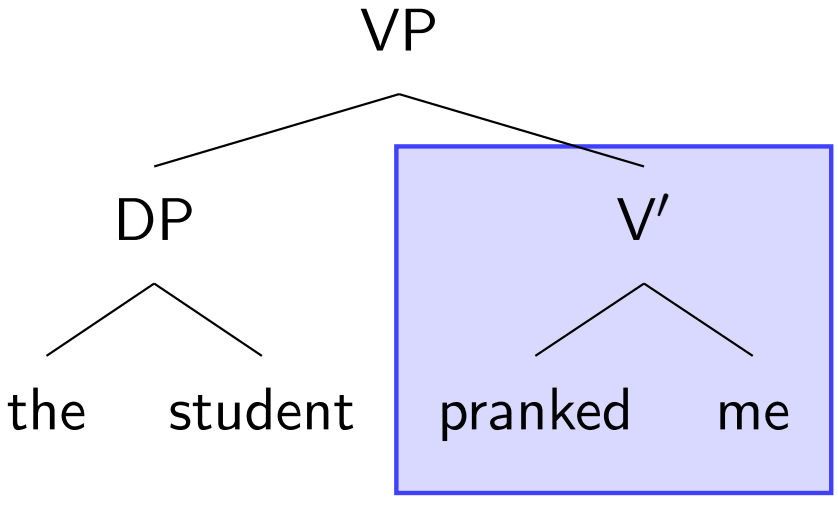
checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $ $ **a** + **s** **o** c **i** **a** l **e** $

SL-**n** over trees

sliding window of **depth n**

checks tree for illicit subtrees



SL-2

SL Over Strings and Trees

SL-**n** over strings

sliding window of **width n**

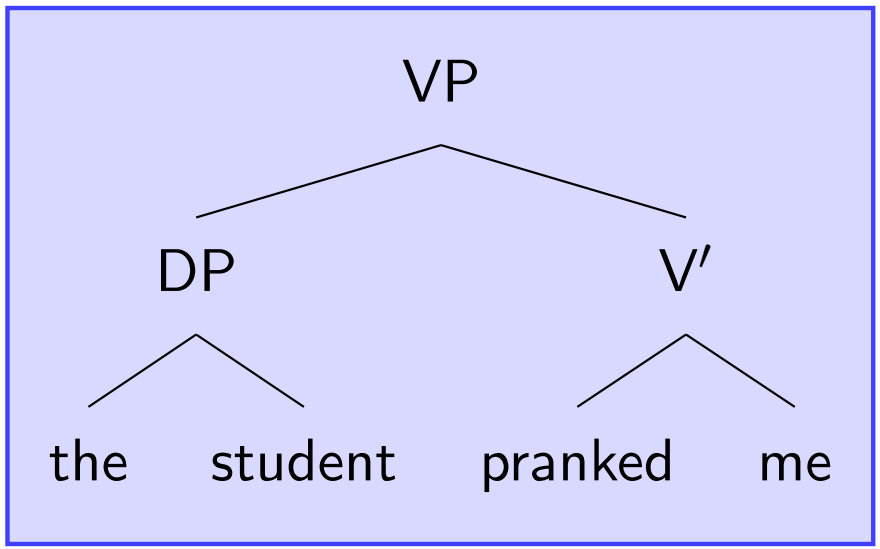
checks string for illicit substrings

* $ **a** **s** **o** l **a** $ $ **a** z **o** l **a** $ $ **a** + **s** **o** c **i** **a** l **e** $

SL-**n** over trees

sliding window of **depth n**

checks tree for illicit subtrees

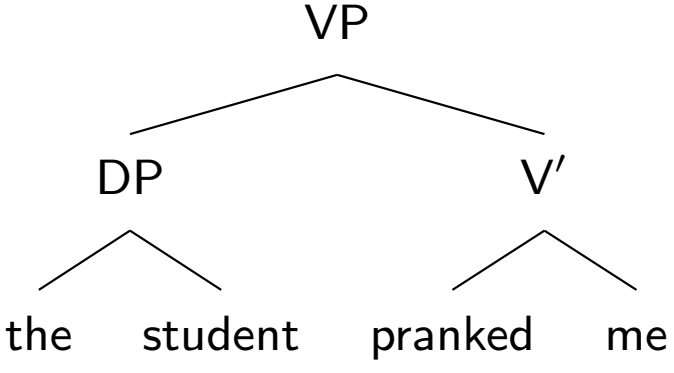


SL-3

Feature-Driven Merge

- What is the computational complexity of Merge?
- We have to look at the computations carried out by syntax, i.e. the **derivation!**

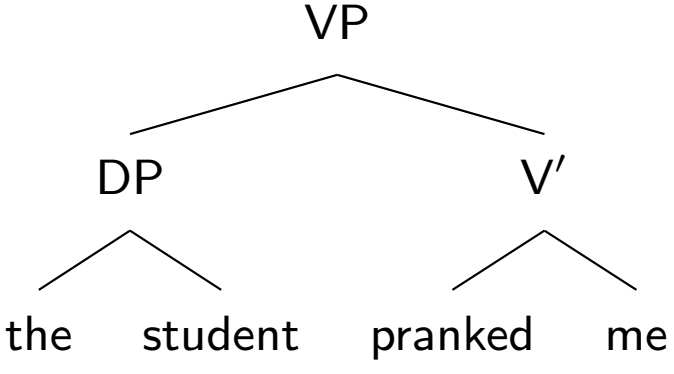
Bare Phrase Structure Tree



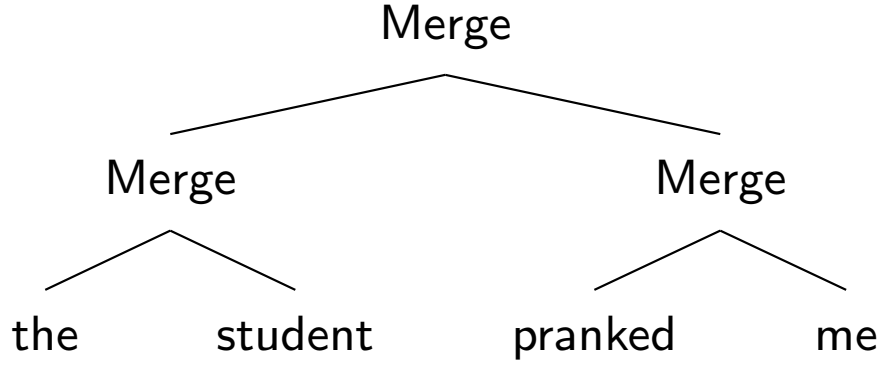
Feature-Driven Merge

- What is the computational complexity of Merge?
- We have to look at the computations carried out by syntax, i.e. the **derivation!**

Bare Phrase Structure Tree



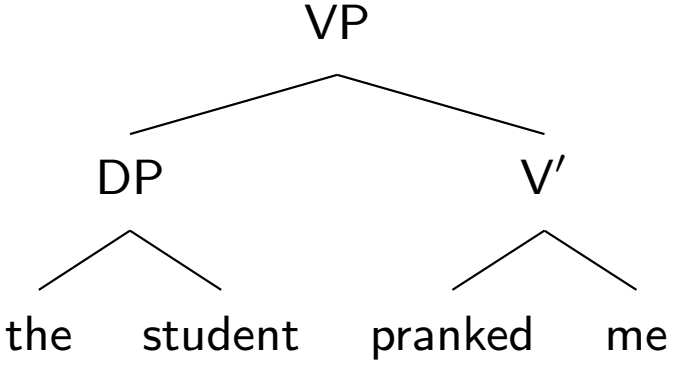
Derivation Tree



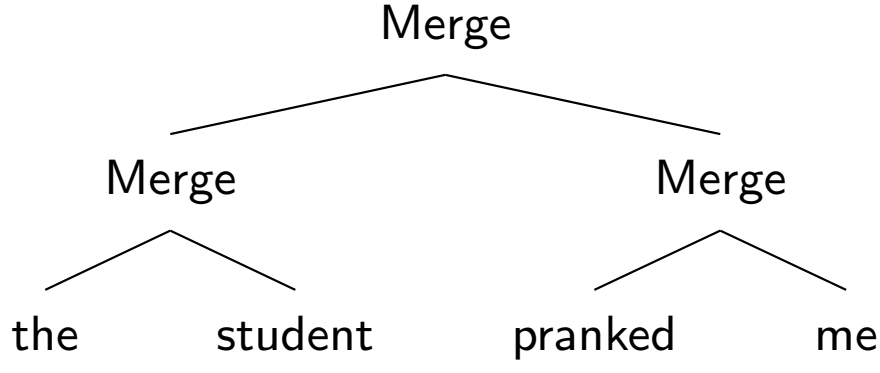
Feature-Driven Merge

- What is the computational complexity of Merge?
- We have to look at the computations carried out by syntax, i.e. the **derivation!**

Bare Phrase Structure Tree



Derivation Tree

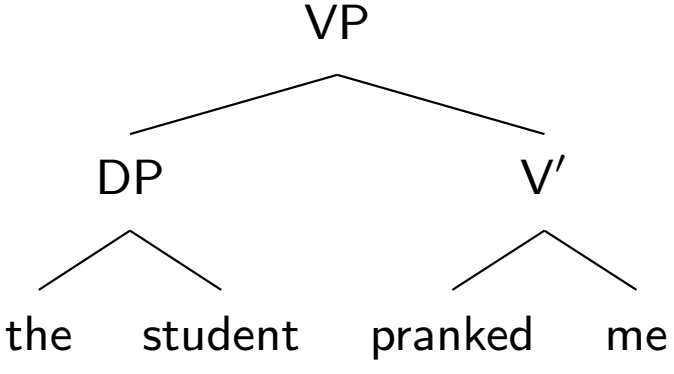


- **Assumption:** Merge is feature-triggered
  (or: interfaces use feature-like information to limit Merge)

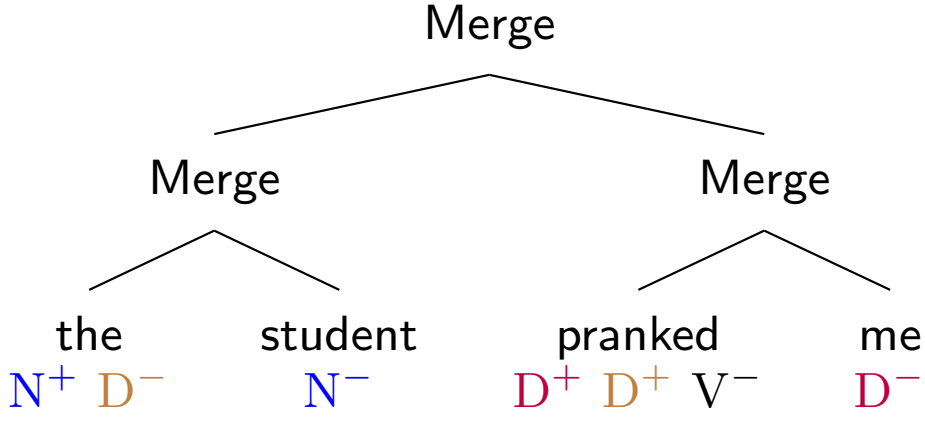
Feature-Driven Merge

- What is the computational complexity of Merge?
- We have to look at the computations carried out by syntax, i.e. the **derivation!**

Bare Phrase Structure Tree



Derivation Tree

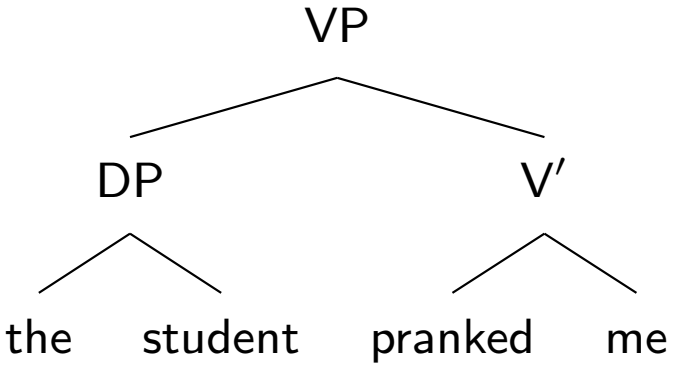


- **Assumption:** Merge is feature-triggered
  (or: interfaces use feature-like information to limit Merge)

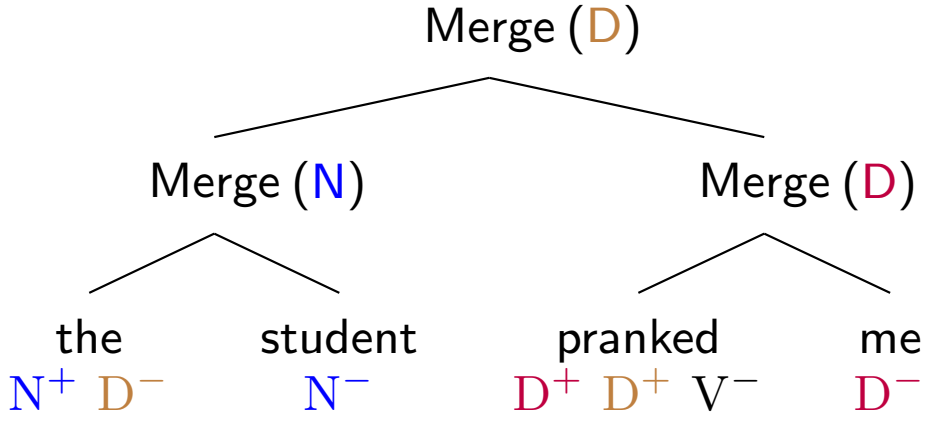
Feature-Driven Merge

- What is the computational complexity of Merge?
- We have to look at the computations carried out by syntax, i.e. the **derivation!**

Bare Phrase Structure Tree



Derivation Tree



- **Assumption:** Merge is feature-triggered
  (or: interfaces use feature-like information to limit Merge)

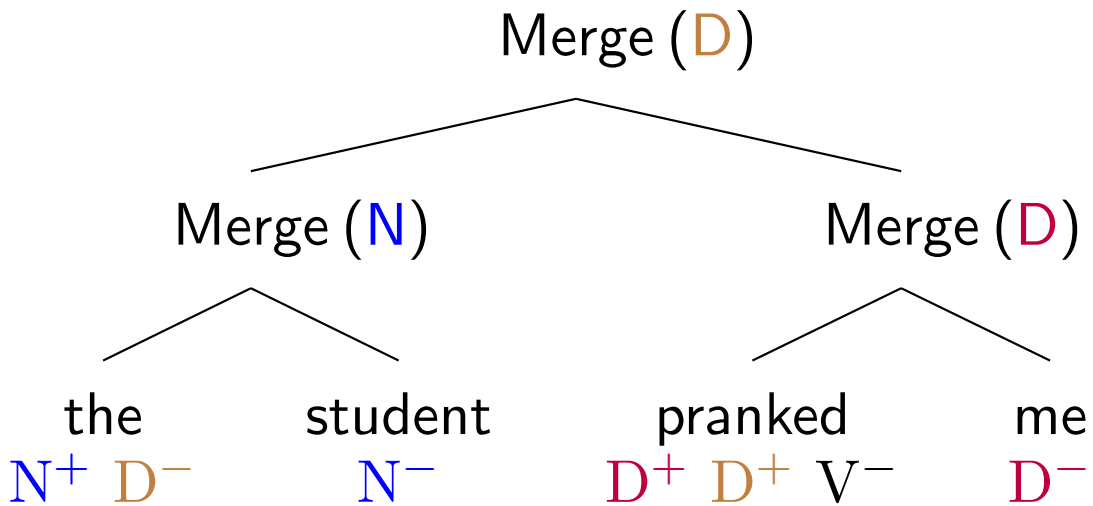
Remark: The Central Role of Derivation Trees

- Derivation trees are rarely considered in generative syntax.
  (but see Epstein et al. 1998)
- Satisfy Chomsky's structural desiderata:
  - no linear order
  - label-free
  - extension condition
  - inclusiveness condition
- Contain all information to produce phrase structure trees
  $\Rightarrow$ **central data structure** of syntax

Merge as an SL-Dependency over Derivation Trees

Feature Calculus for Merge

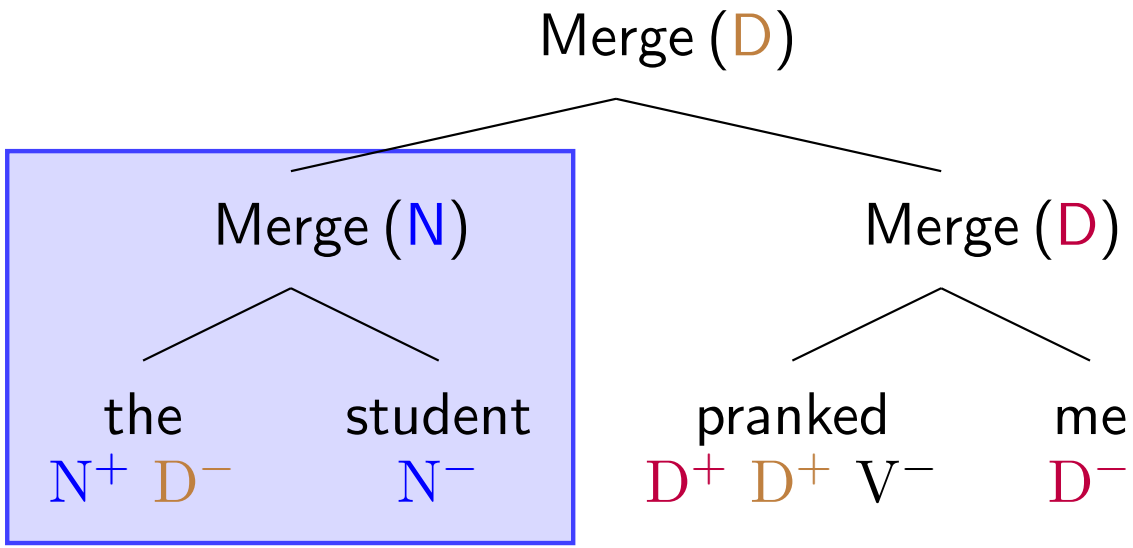
- The selector features (X^{+}) of the head have to match the category features (X^{-}) of the arguments.
- 1-to-1 match between selector features and category features.



Merge as an SL-Dependency over Derivation Trees

Feature Calculus for Merge

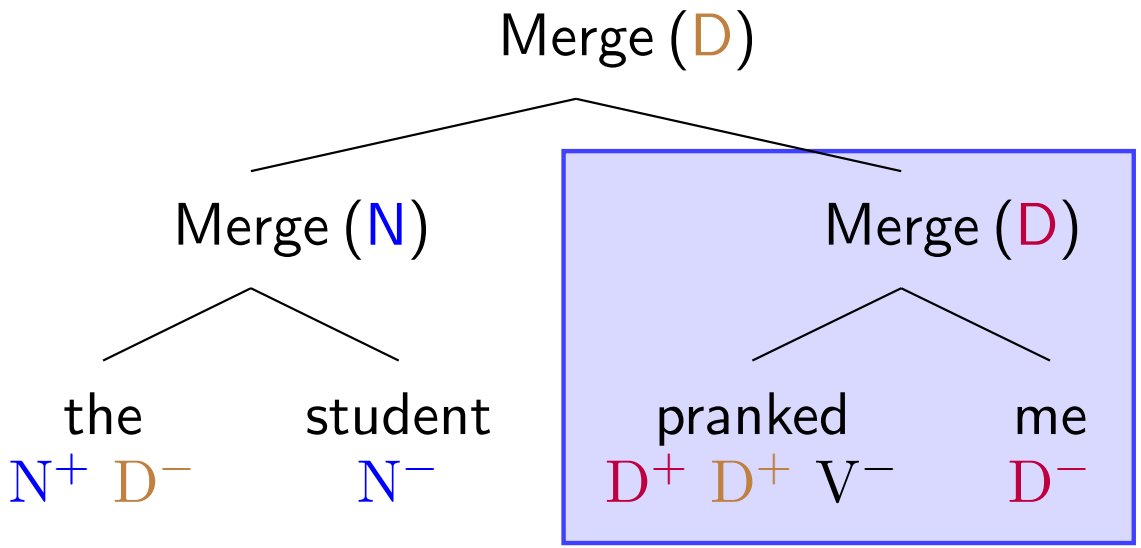
- The selector features (X^{+}) of the head have to match the category features (X^{-}) of the arguments.
- 1-to-1 match between selector features and category features.



Merge as an SL-Dependency over Derivation Trees

Feature Calculus for Merge

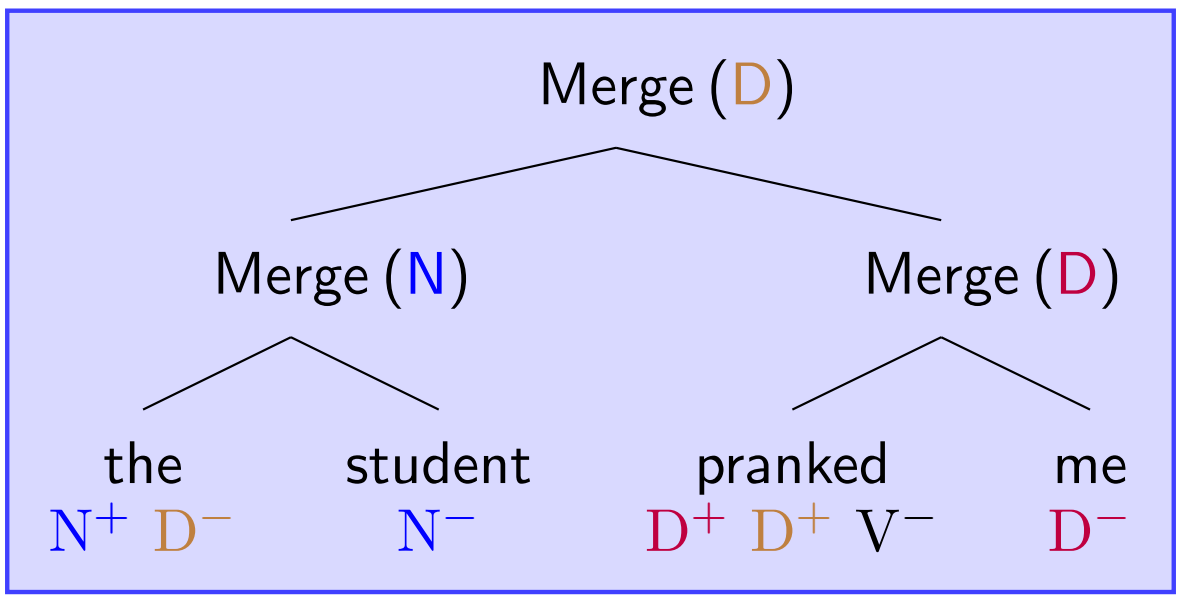
- The selector features (X^{+}) of the head have to match the category features (X^{-}) of the arguments.
- 1-to-1 match between selector features and category features.



Merge as an SL-Dependency over Derivation Trees

Feature Calculus for Merge

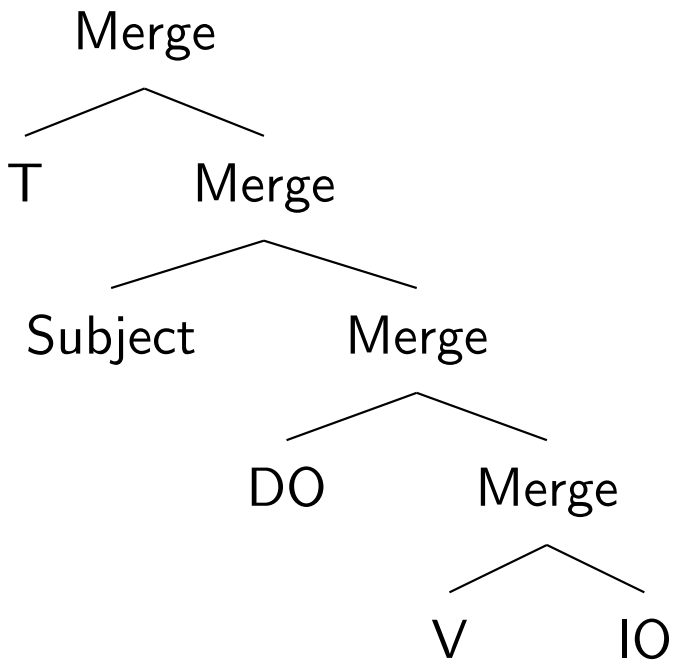
- The selector features (X^{+}) of the head have to match the category features (X^{-}) of the arguments.
- 1-to-1 match between selector features and category features.



SL-Merge Lacks Succinctness

- A large sliding window contains lots of irrelevant material
  $\Rightarrow$ needlessly bloated list of forbidden subtrees
- Small grammars should be cognitively preferable to large ones.
- Factorization helps, but not enough.

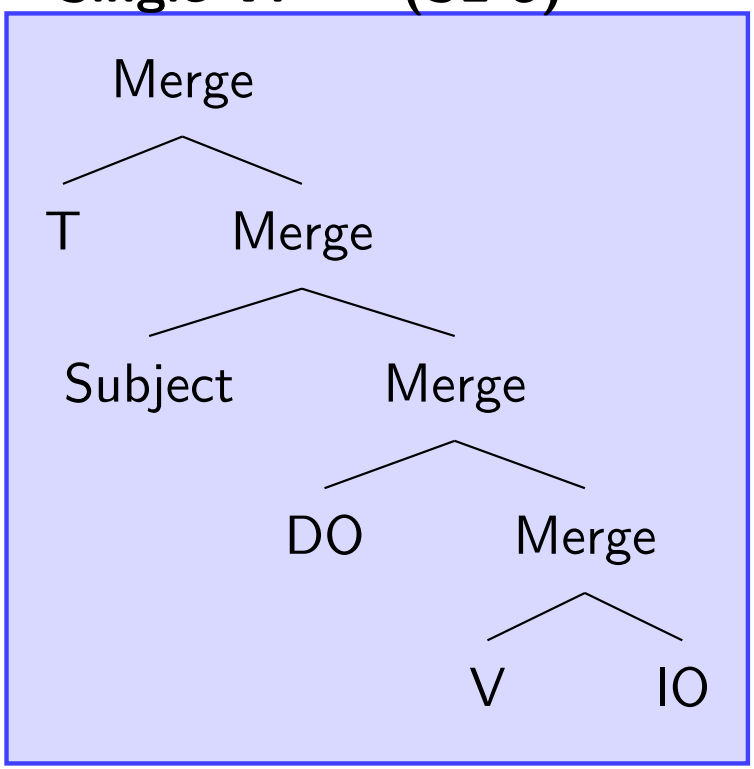
Single VP



SL-Merge Lacks Succinctness

- A large sliding window contains lots of irrelevant material
  $\Rightarrow$ needlessly bloated list of forbidden subtrees
- Small grammars should be cognitively preferable to large ones.
- Factorization helps, but not enough.

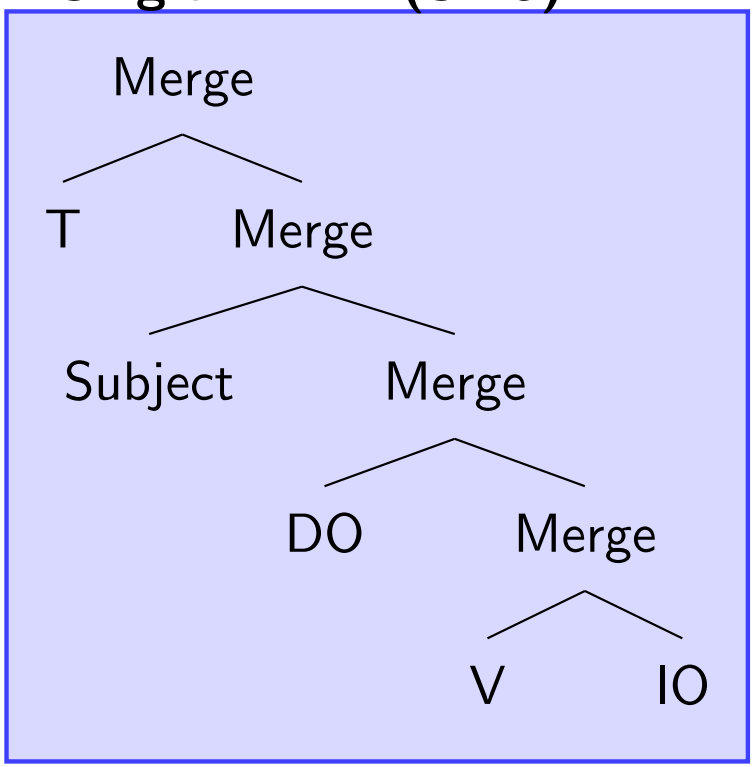
Single VP (SL-5)



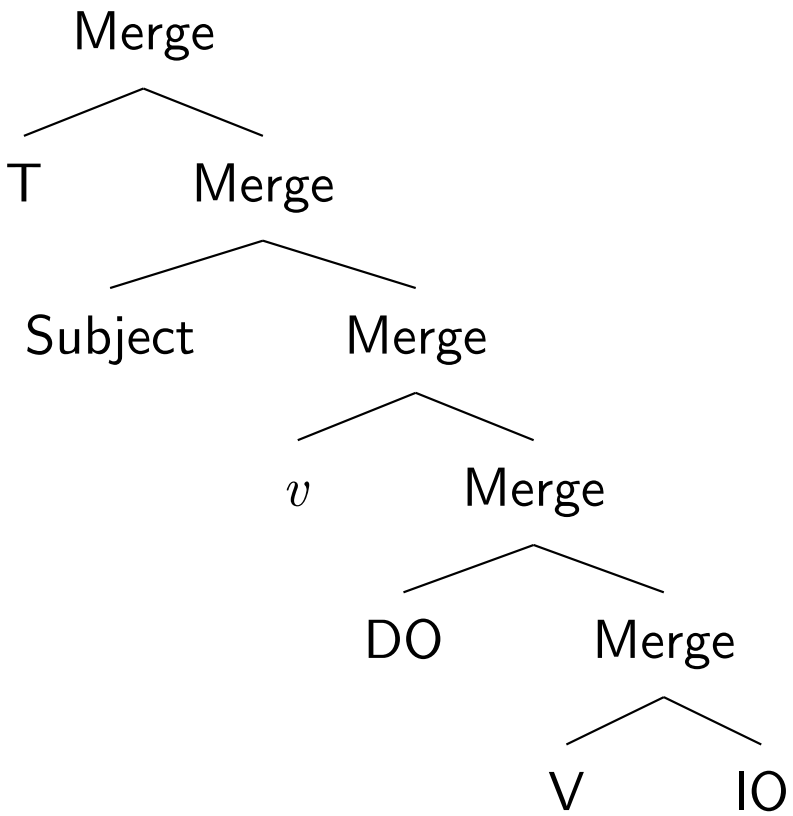
SL-Merge Lacks Succinctness

- A large sliding window contains lots of irrelevant material
  $\Rightarrow$ needlessly bloated list of forbidden subtrees
- Small grammars should be cognitively preferable to large ones.
- Factorization helps, but not enough.

Single VP (SL-5)



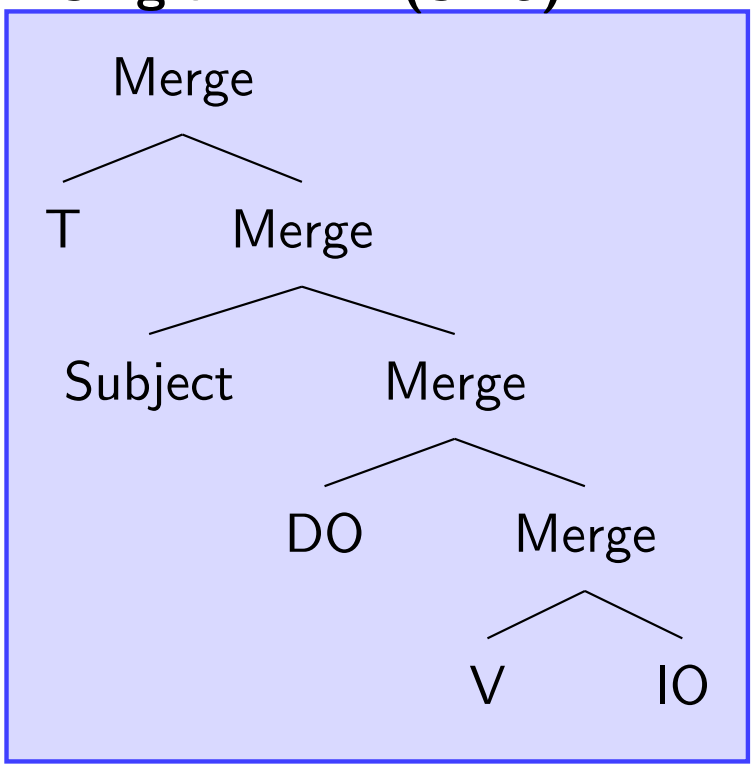
VP-Shells



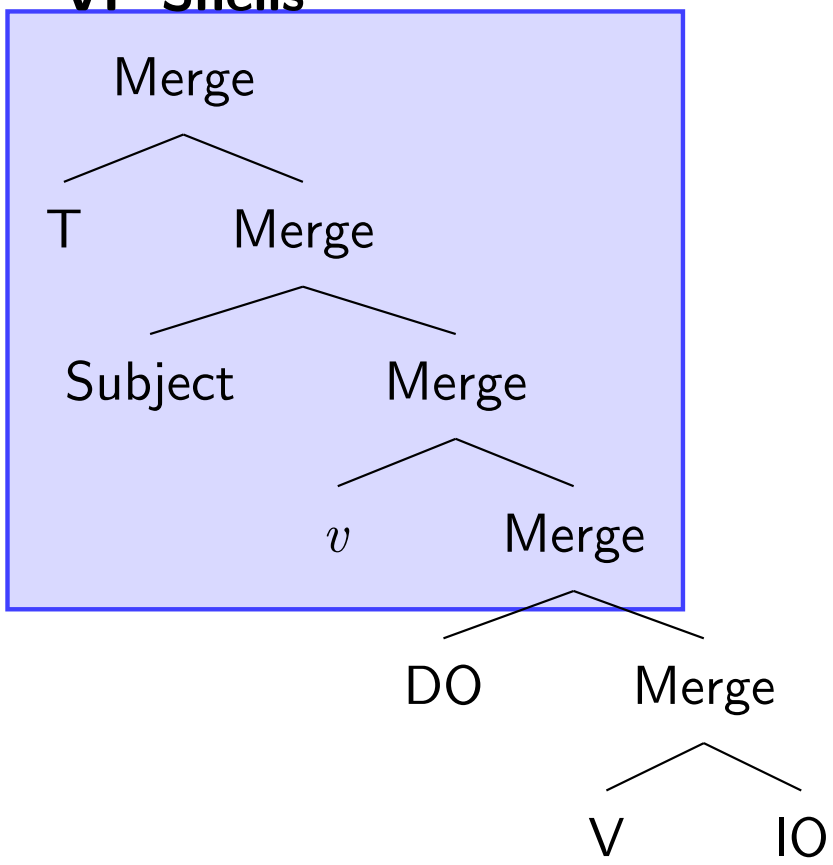
SL-Merge Lacks Succinctness

- A large sliding window contains lots of irrelevant material
  $\Rightarrow$ needlessly bloated list of forbidden subtrees
- Small grammars should be cognitively preferable to large ones.
- Factorization helps, but not enough.

Single VP (SL-5)



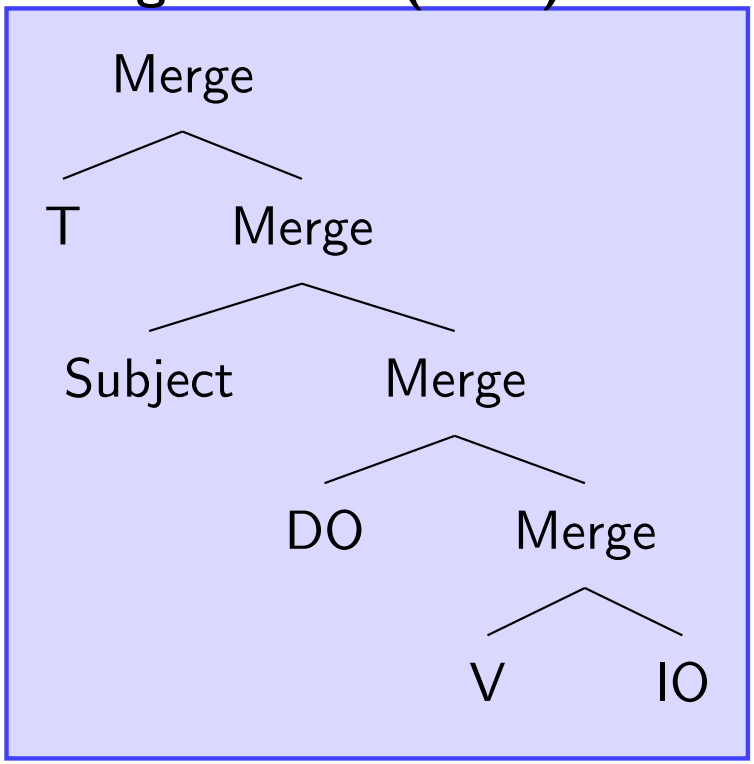
VP-Shells



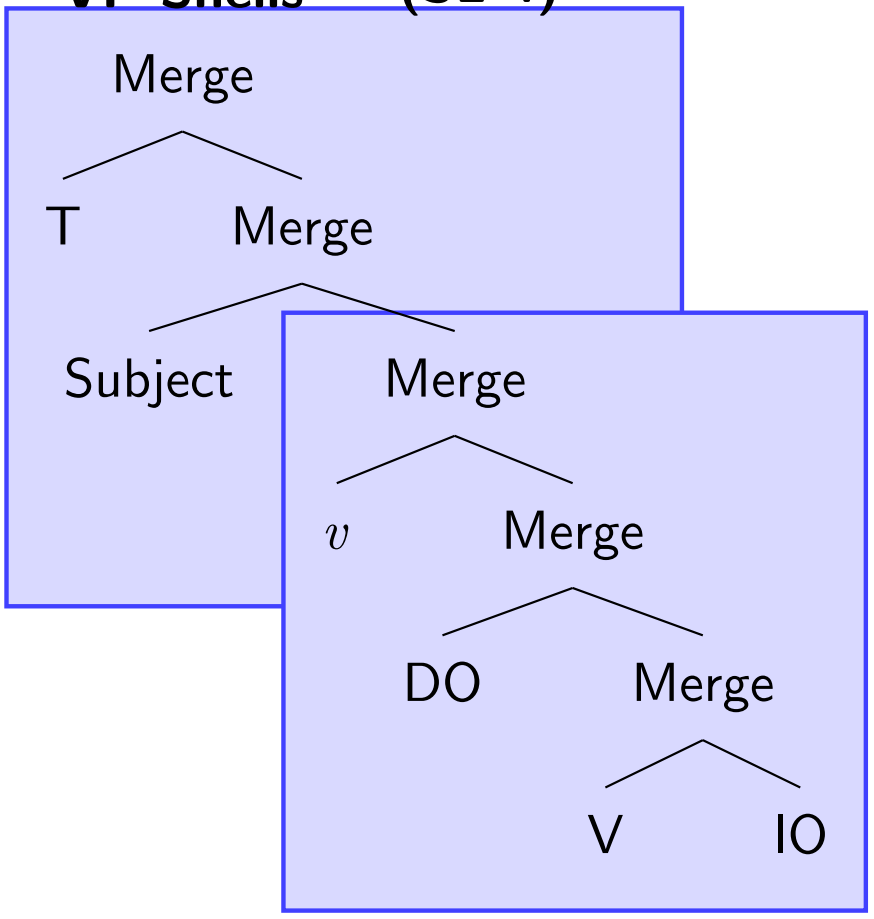
SL-Merge Lacks Succinctness

- A large sliding window contains lots of irrelevant material
  $\Rightarrow$ needlessly bloated list of forbidden subtrees
- Small grammars should be cognitively preferable to large ones.
- Factorization helps, but not enough.

Single VP (SL-5)



VP-Shells (SL-4)



Reminder: Samala Sibilant Harmony

- We ran into a similar succinctness problem with SL in string land.

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 $

Reminder: Samala Sibilant Harmony

- We ran into a similar succinctness problem with SL in string land.

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 $

Reminder: Samala Sibilant Harmony

- We ran into a similar succinctness problem with SL in string land.

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 $

Reminder: Samala Sibilant Harmony

- We ran into a similar succinctness problem with SL in string land.

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 $

Reminder: Samala Sibilant Harmony

- We ran into a similar succinctness problem with SL in string land.

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 $

Reminder: Samala Sibilant Harmony

- We ran into a similar succinctness problem with SL in string land.

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 $

Reminder: Samala Sibilant Harmony [cont.]

- Our solution then was to move **from SL to TSL**.

Example: Samala Revisited

1. Project sibilant tier
2. *sʃ, *sʒ, *zʃ, *zʒ, *ʃs, *ʒs, *ʃz, *ʒz

*$ 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 $

- We can do the same over trees!

Reminder: Samala Sibilant Harmony [cont.]

- Our solution then was to move **from SL to TSL**.

Example: Samala Revisited

1. Project sibilant tier
2. *sʃ, *sʒ, *zʃ, *zʒ, *ʃs, *ʒs, *ʃz, *ʒz

*$ 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 $

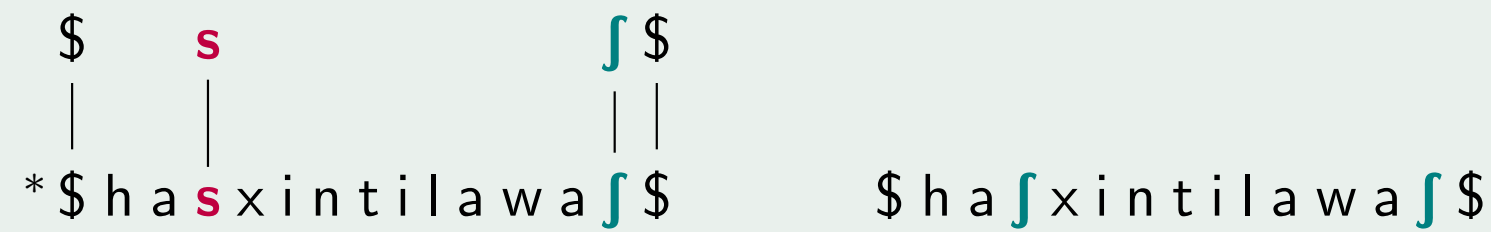
- We can do the same over trees!

Reminder: Samala Sibilant Harmony [cont.]

- Our solution then was to move **from SL to TSL**.

Example: Samala Revisited

1. Project sibilant tier
2. *sʃ, *sʒ, *zʃ, *zʒ, *ʃs, *ʒs, *ʃz, *ʒz



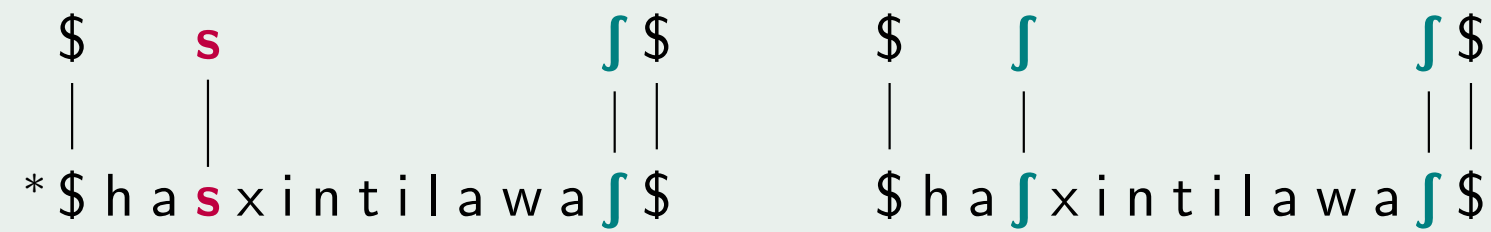
- We can do the same over trees!

Reminder: Samala Sibilant Harmony [cont.]

- Our solution then was to move **from SL to TSL**.

Example: Samala Revisited

1. Project sibilant tier
2. *sʃ, *sʒ, *zʃ, *zʒ, *ʃs, *ʒs, *ʃz, *ʒz



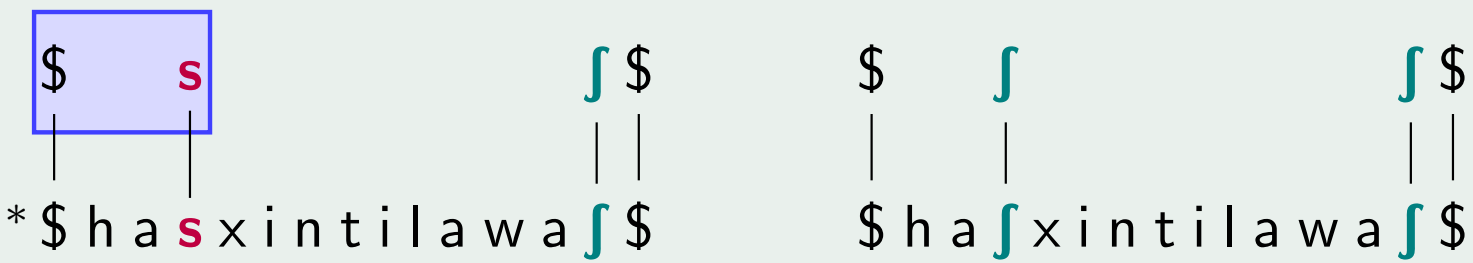
- We can do the same over trees!

Reminder: Samala Sibilant Harmony [cont.]

- Our solution then was to move **from SL to TSL**.

Example: Samala Revisited

1. Project sibilant tier
2. *sʃ, *sʒ, *zʃ, *zʒ, *ʃs, *ʒs, *ʃz, *ʒz



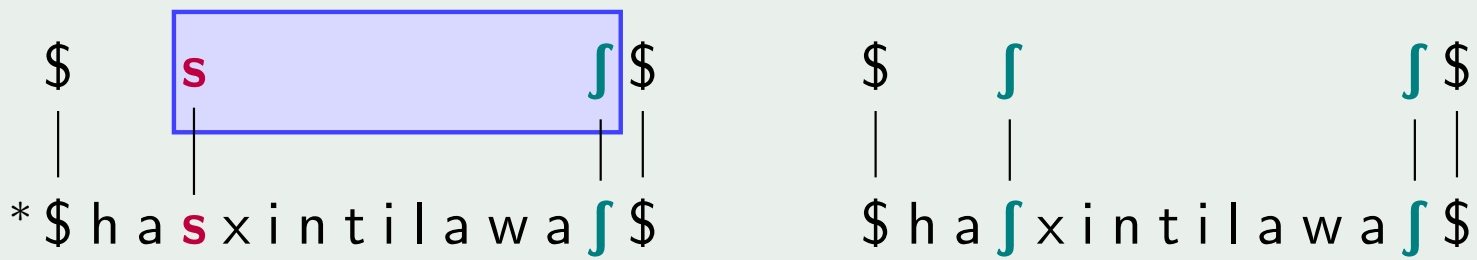
- We can do the same over trees!

Reminder: Samala Sibilant Harmony [cont.]

- Our solution then was to move **from SL to TSL**.

Example: Samala Revisited

1. Project sibilant tier
2. *sʃ, *sʒ, *zʃ, *zʒ, *ʃs, *ʒs, *ʃz, *ʒz



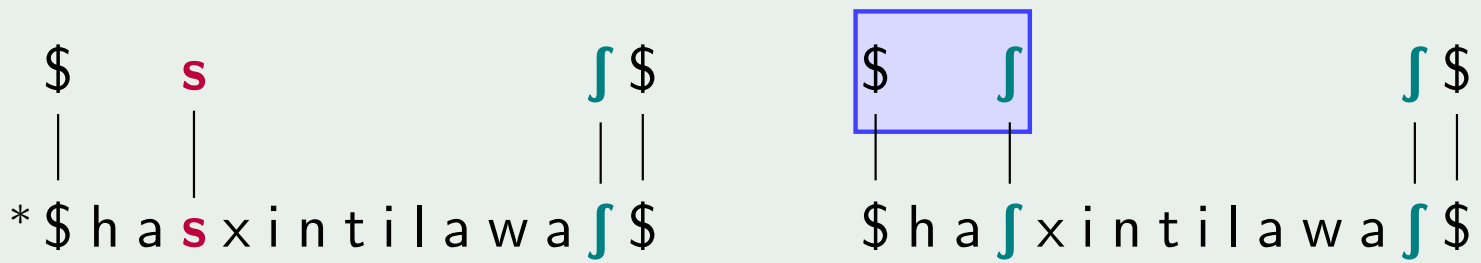
- We can do the same over trees!

Reminder: Samala Sibilant Harmony [cont.]

- Our solution then was to move **from SL to TSL**.

Example: Samala Revisited

1. Project sibilant tier
2. *sʃ, *sʒ, *zʃ, *zʒ, *ʃs, *ʒs, *ʃz, *ʒz



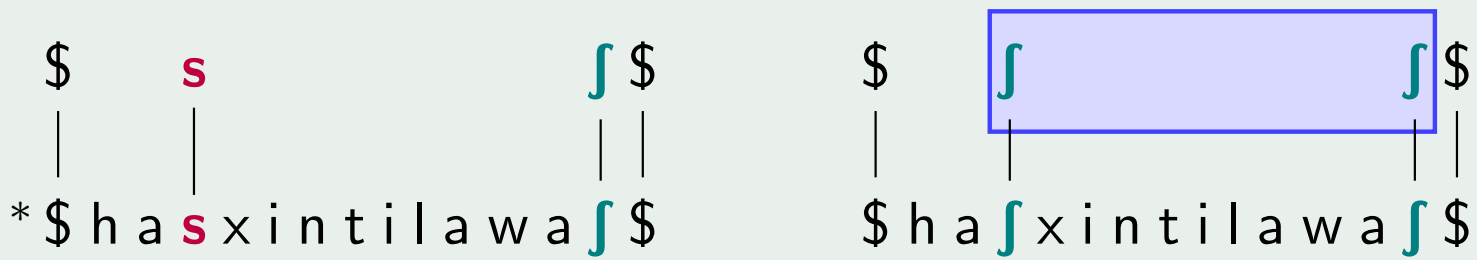
- We can do the same over trees!

Reminder: Samala Sibilant Harmony [cont.]

- Our solution then was to move **from SL to TSL**.

Example: Samala Revisited

1. Project sibilant tier
2. *sʃ, *sʒ, *zʃ, *zʒ, *ʃs, *ʒs, *ʃz, *ʒz



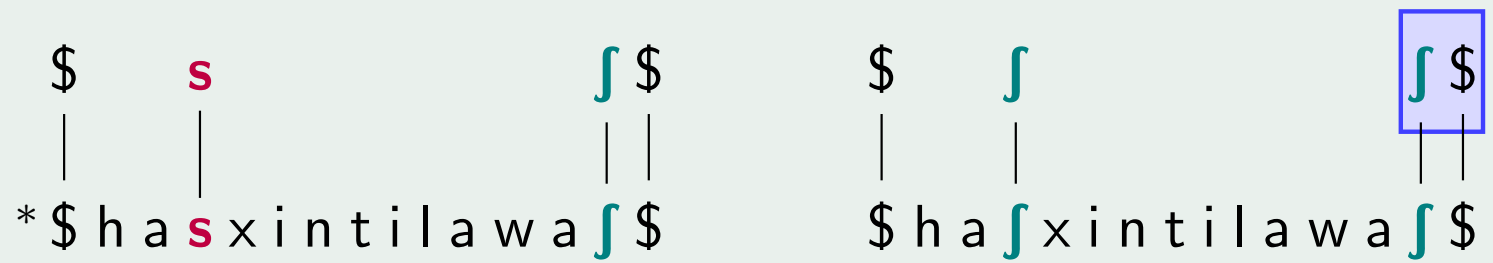
- We can do the same over trees!

Reminder: Samala Sibilant Harmony [cont.]

- Our solution then was to move **from SL to TSL**.

Example: Samala Revisited

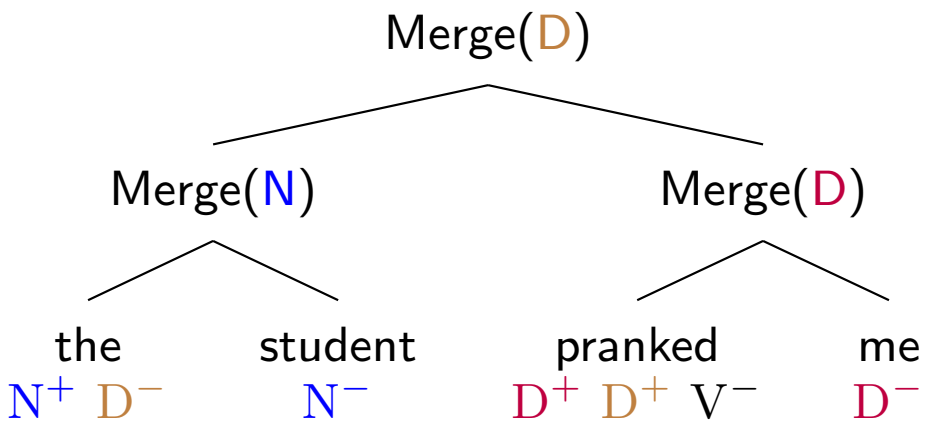
1. Project sibilant tier
2. *sʃ, *sʒ, *zʃ, *zʒ, *ʃs, *ʒs, *ʃz, *ʒz



- We can do the same over trees!

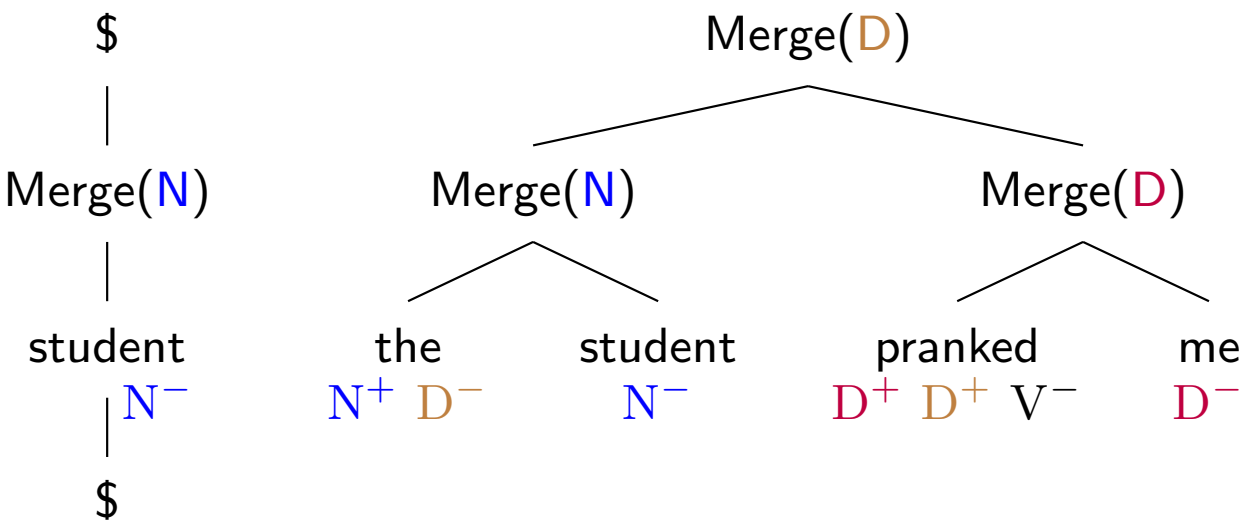
Category Tiers for Merge

- Project tree tier for each category X:
  1. Project every lexical item with X^-.
  2. Project every Merge(X).
- Every X^- has a Merge node as its mother.
- Every Merge node has exactly one X^- among its daughters.
  (cf. culminativity!)



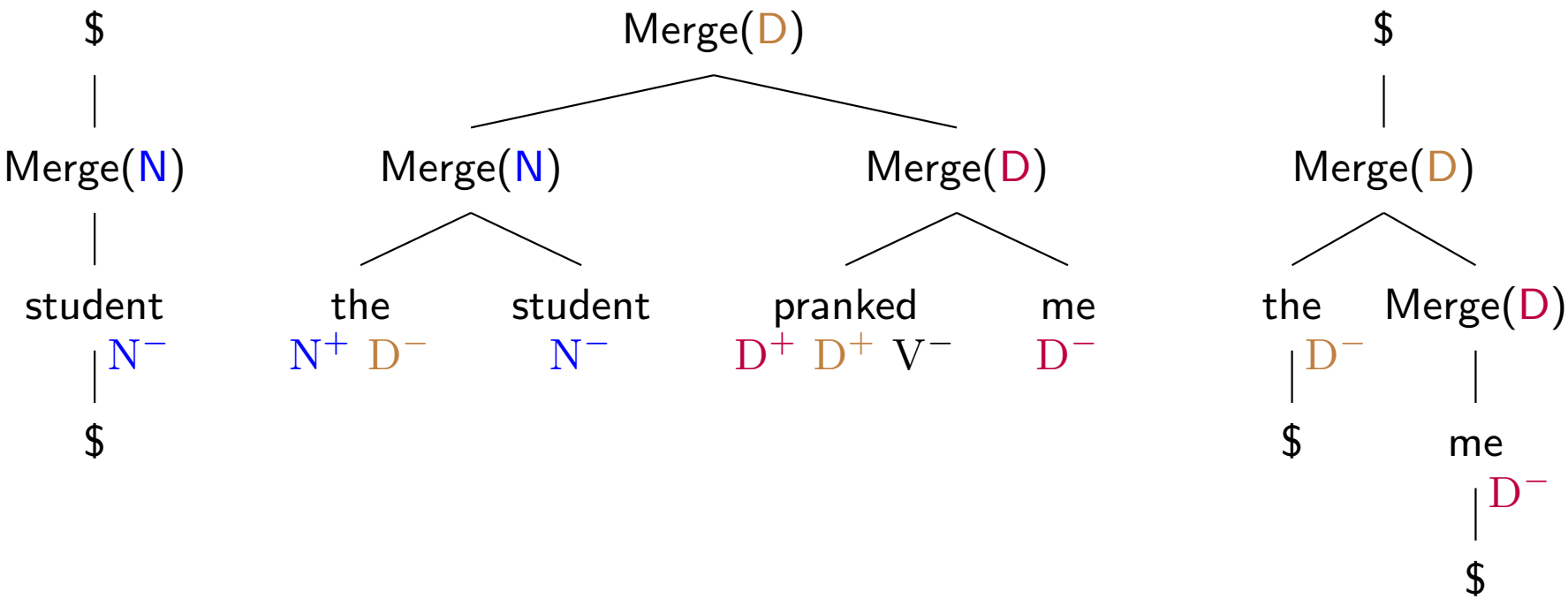
Category Tiers for Merge

- Project tree tier for each category X:
  1. Project every lexical item with X^-.
  2. Project every Merge(X).
- Every X^- has a Merge node as its mother.
- Every Merge node has exactly one X^- among its daughters.
  (cf. culminativity!)

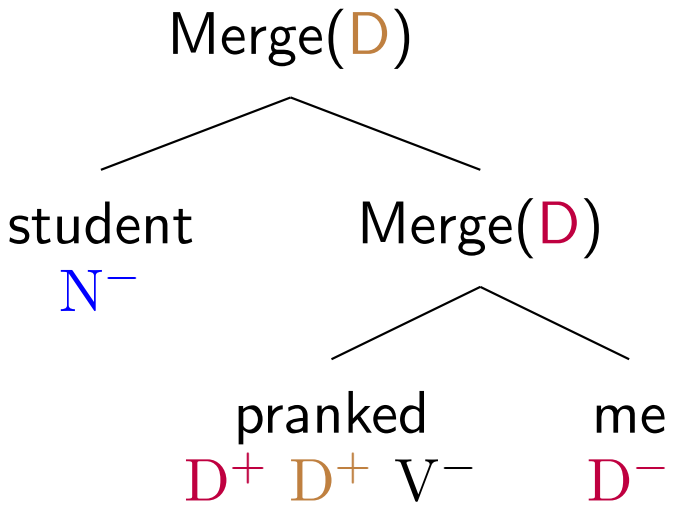


Category Tiers for Merge

- Project tree tier for each category X:
  1. Project every lexical item with X^-.
  2. Project every Merge(X).
- Every X^- has a Merge node as its mother.
- Every Merge node has exactly one X^- among its daughters.
  (cf. culminativity!)

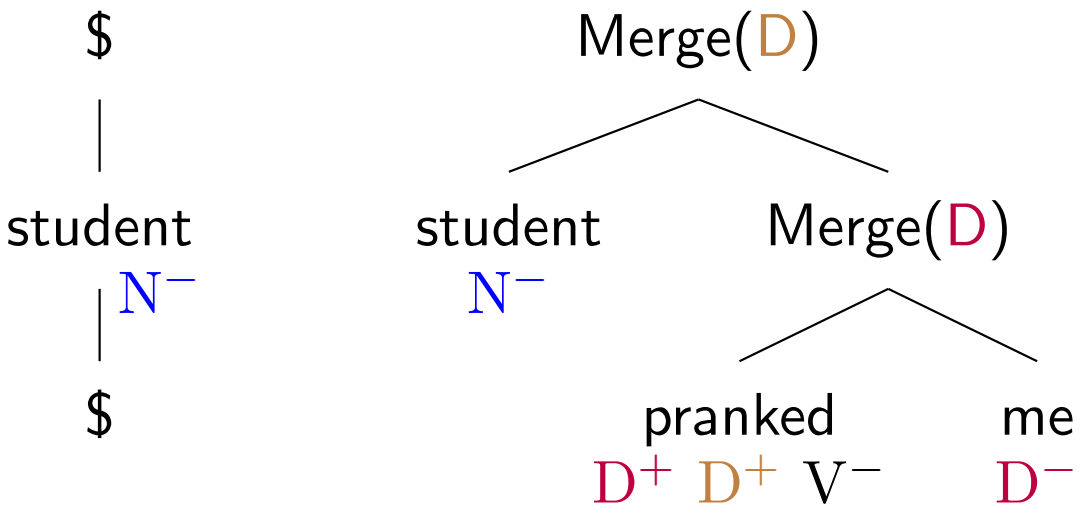


Illicit Merge Yields Ill-Formed Tiers



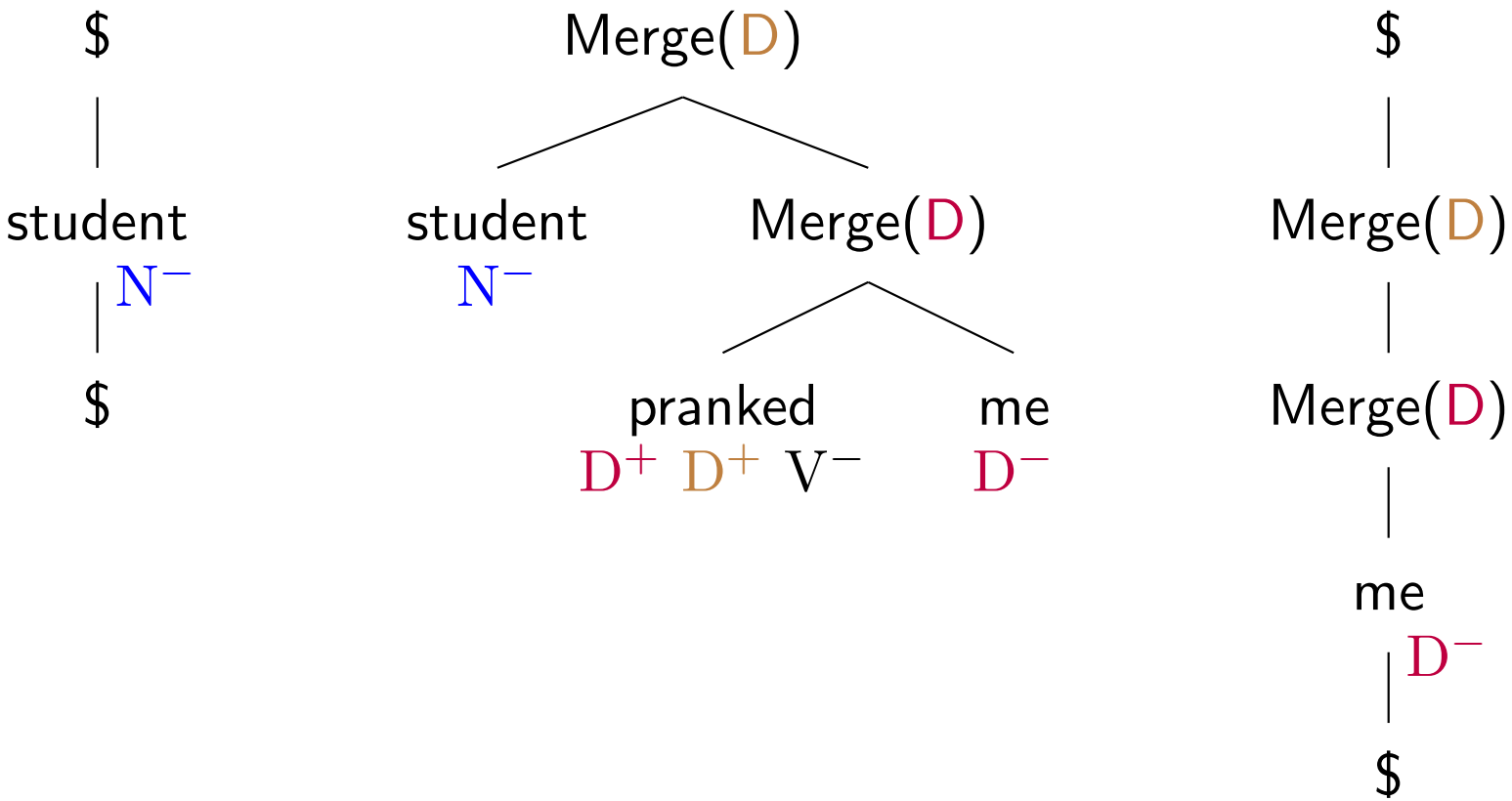
- Because we only need to inspect mother-daughter configurations on each tier, **Merge is TSL-2**.

Illicit Merge Yields Ill-Formed Tiers



- Because we only need to inspect mother-daughter configurations on each tier, **Merge is TSL-2**.

Illicit Merge Yields Ill-Formed Tiers



- Because we only need to inspect mother-daughter configurations on each tier, **Merge is TSL-2**.

Remark: TSL or ITSL?

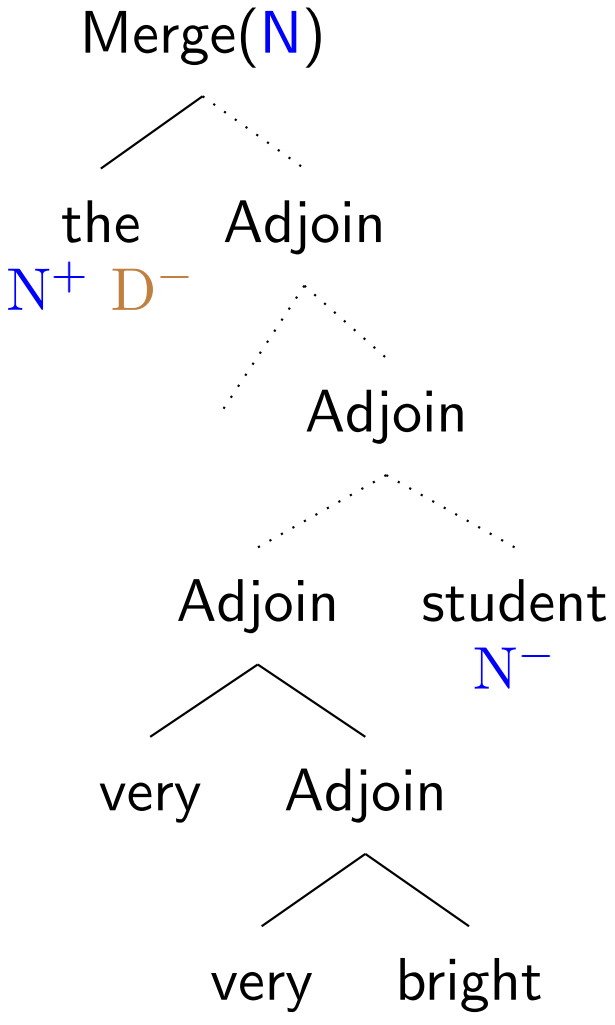
ITSL tier projection considers node label and local context

- Whether Merge is TSL or ITSL depends on whether we assume that interior nodes are labeled
  - Merge, or
  - Merge(**X**).
- The former is preferable as it does not "fossilize" computation into the representation.

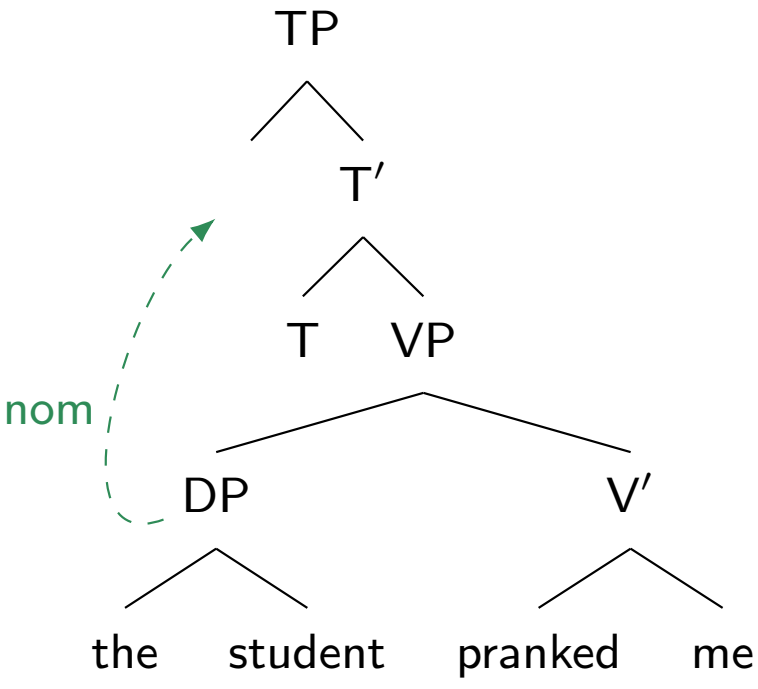
TSL-Merge Enables Adjunction

- Because adjunction is
  - recursive, and
  - iterable

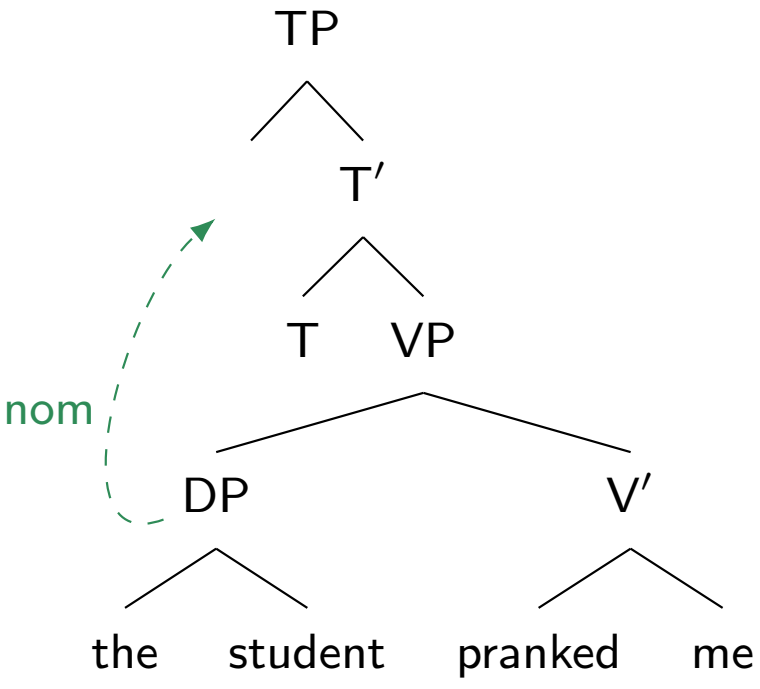
  Merge cannot be SL
  in grammars with adjunction.
- **Corollary**
  Adjunction requires TSL Merge.

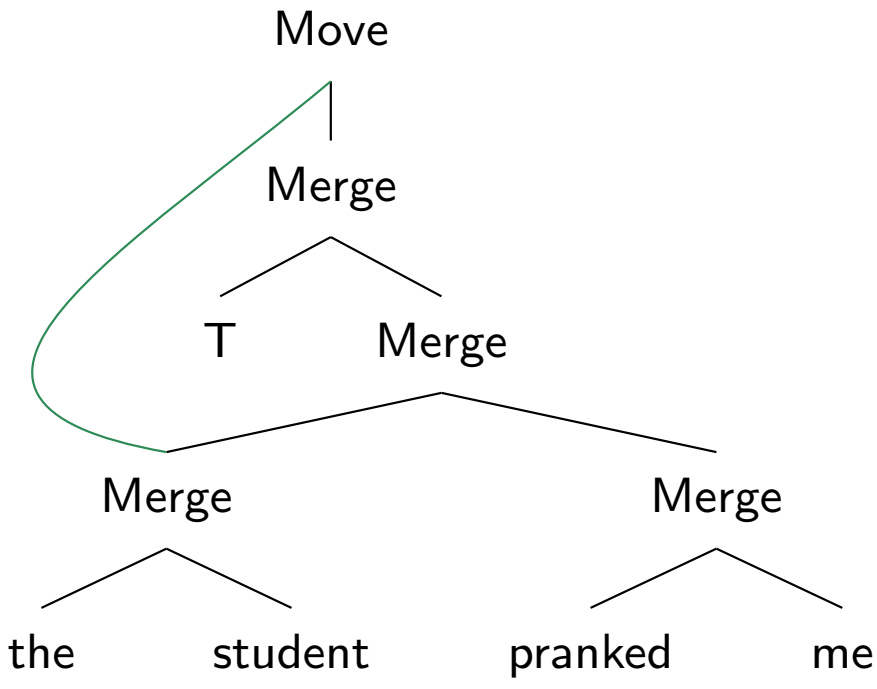


Feature-Driven Move

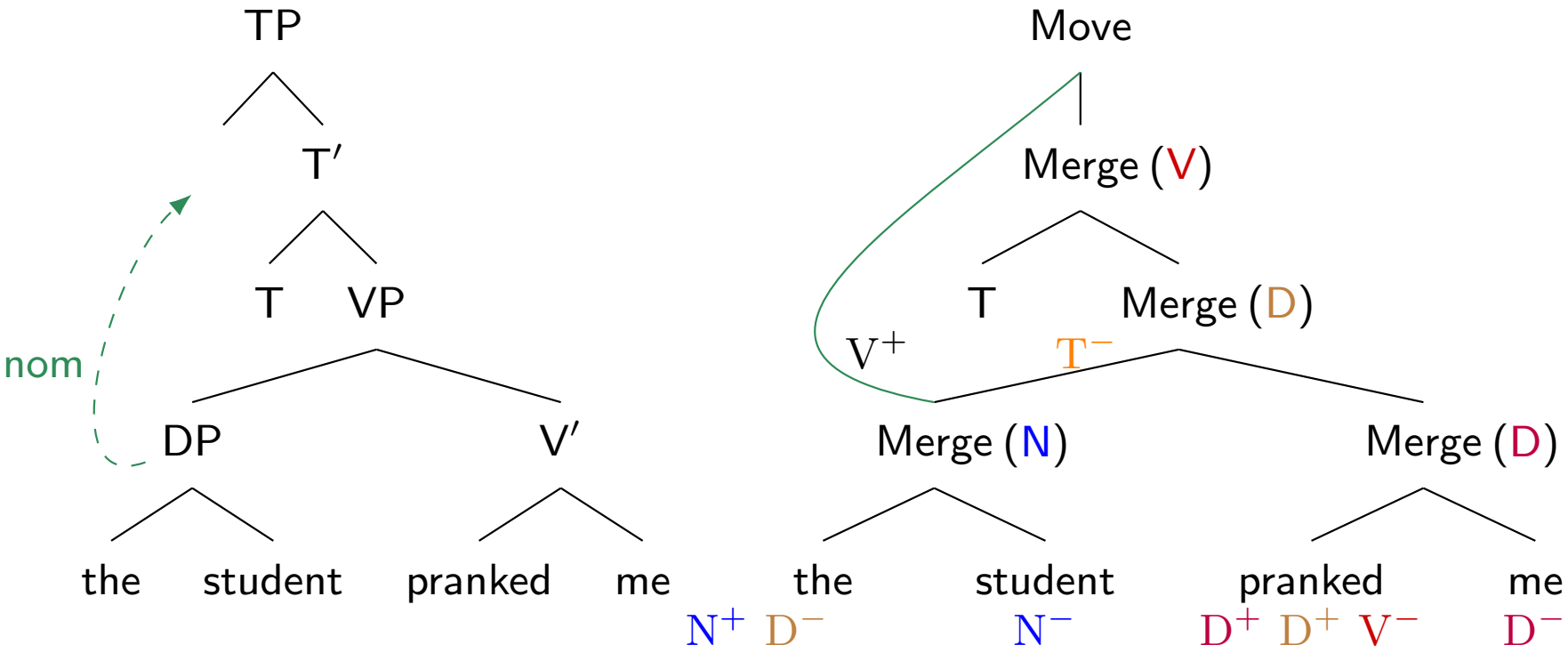


Feature-Driven Move

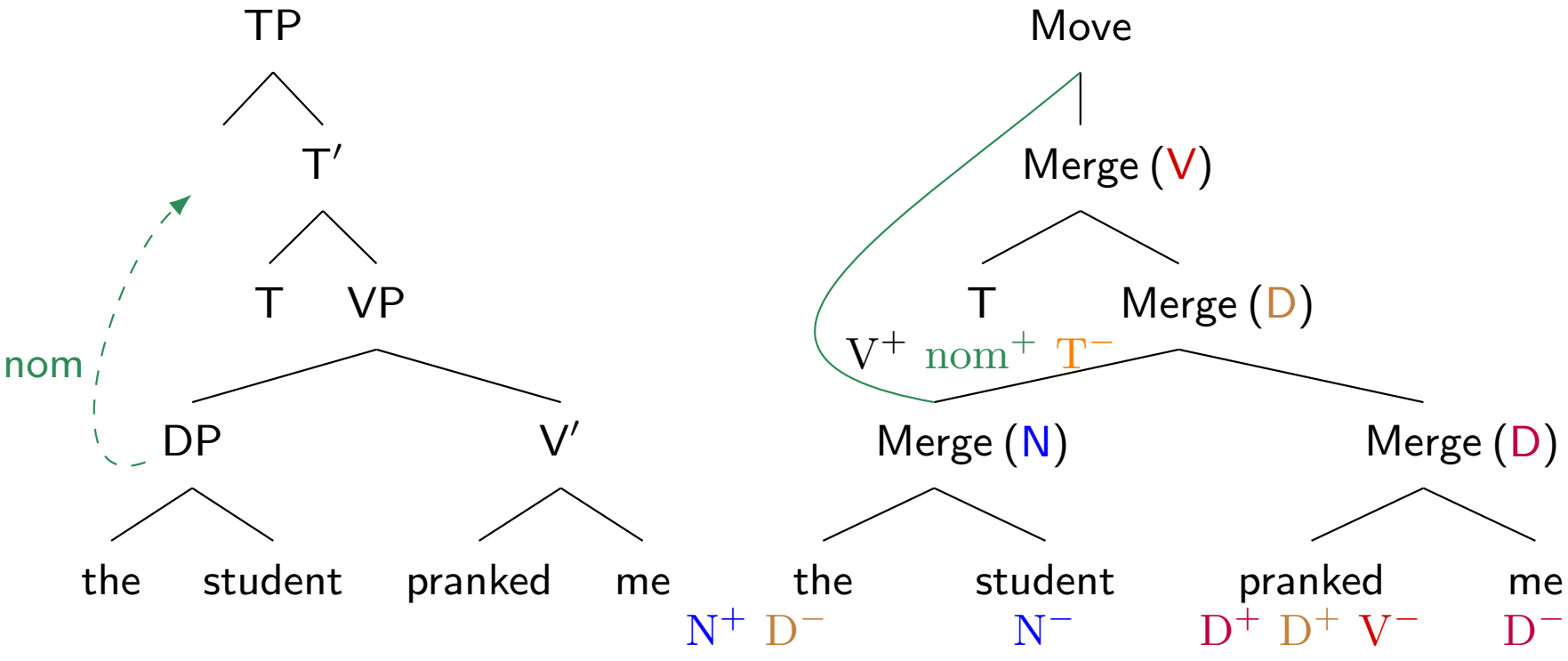




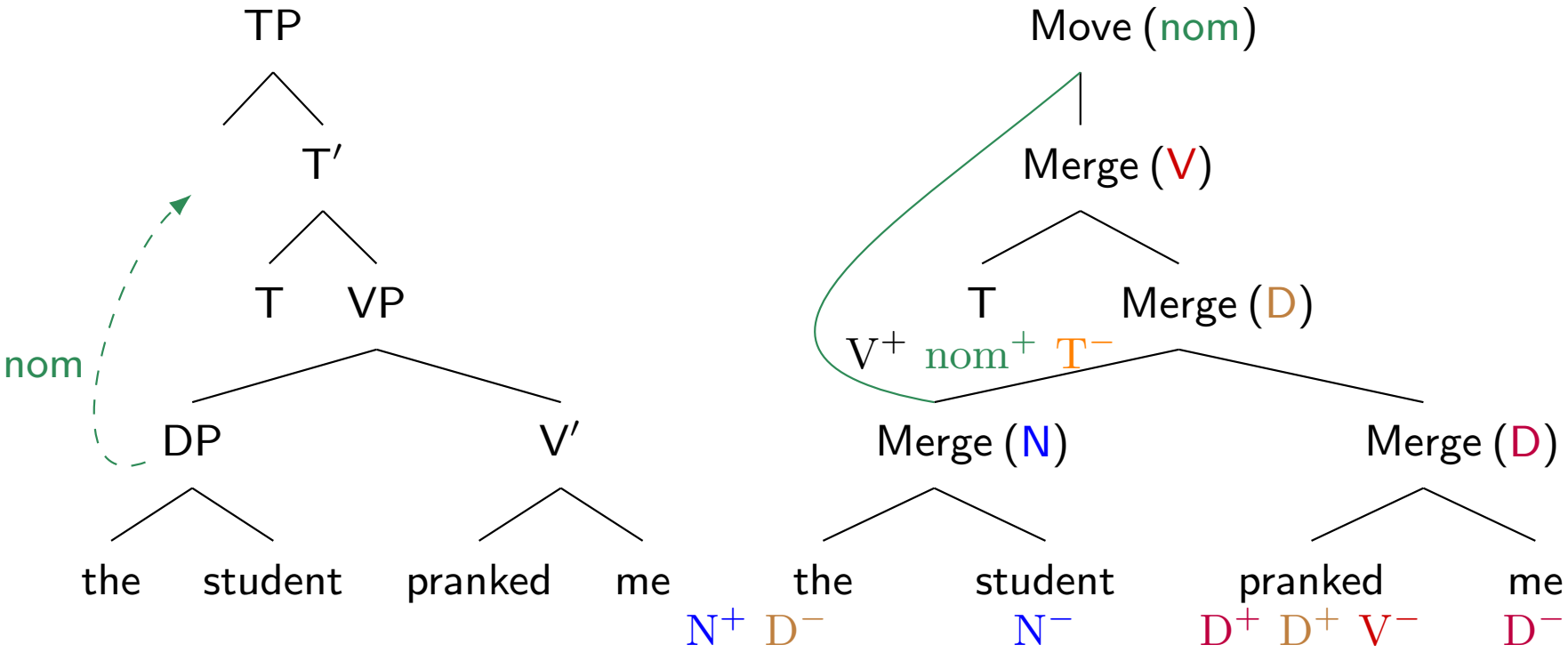
Feature-Driven Move



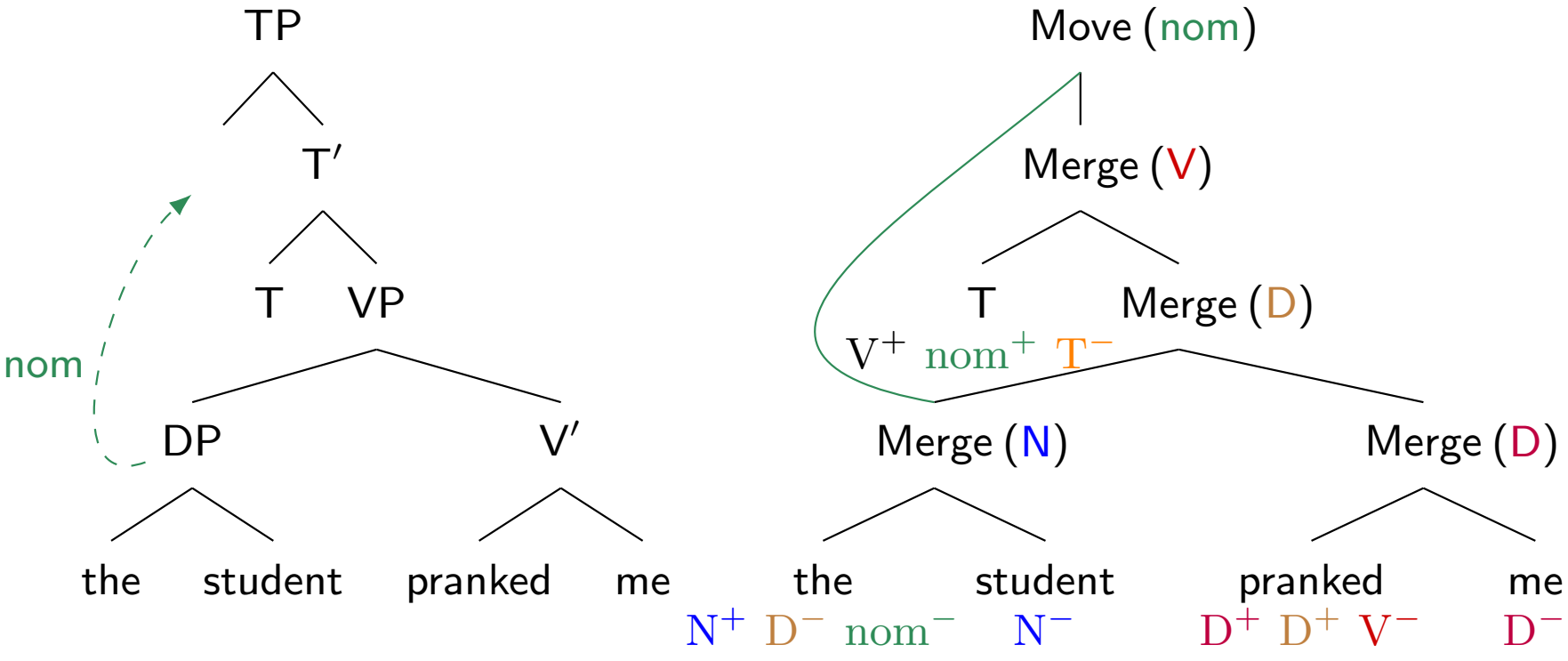
Feature-Driven Move



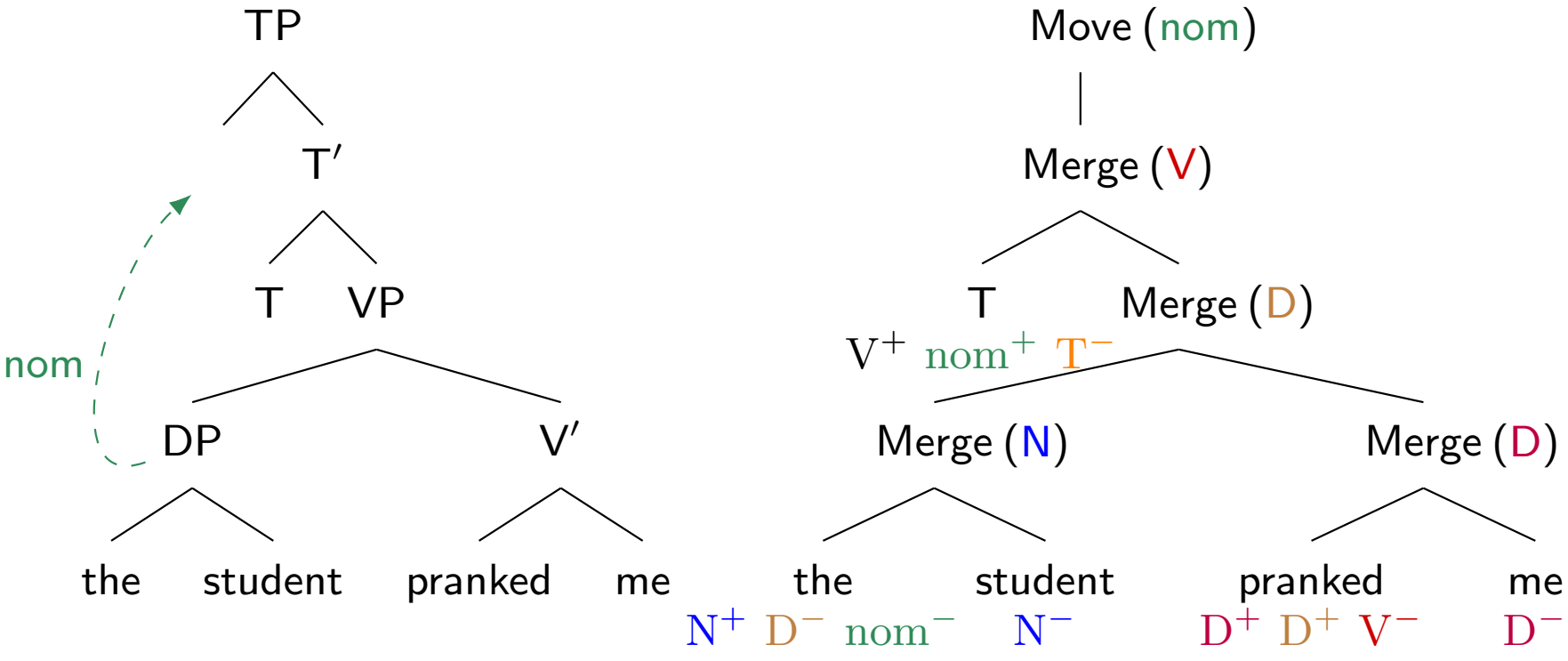
Feature-Driven Move



Feature-Driven Move

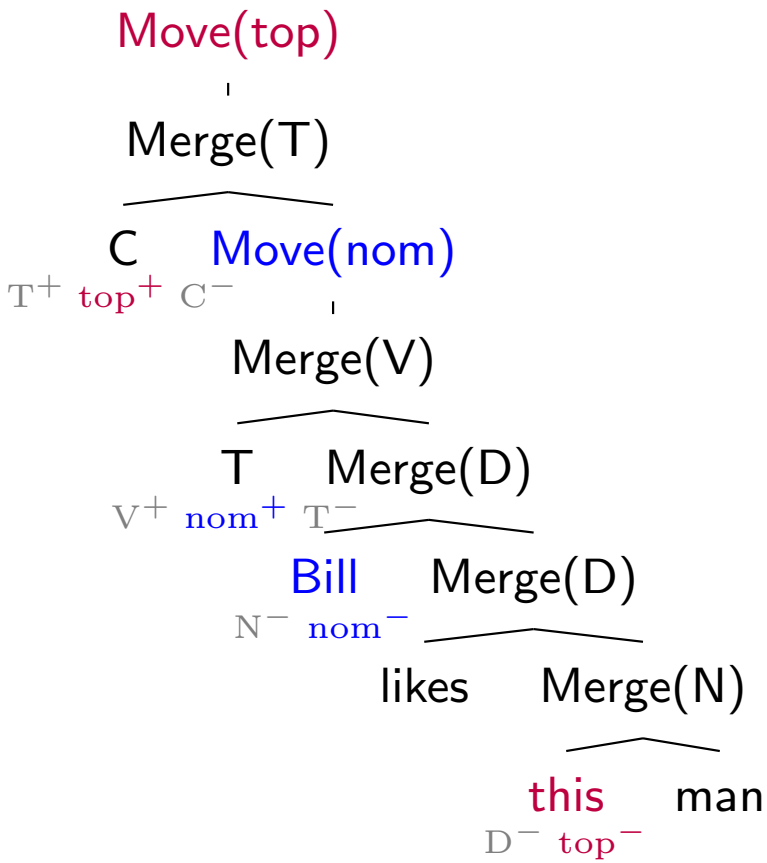


Feature-Driven Move



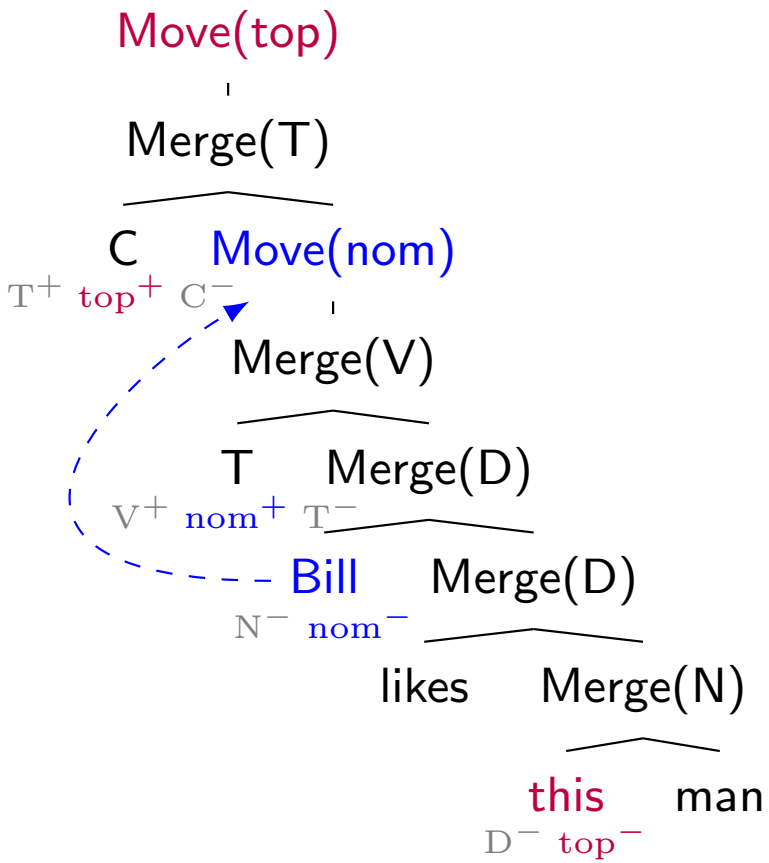
Move as TSL Merge

- Project tree tier for each movement type x.
  1. Project every lexical item with x^-.
  2. Project every Move(x).
- Every x^- has a Move node as its mother.
- Every Move node has exactly one x^- among its daughters.



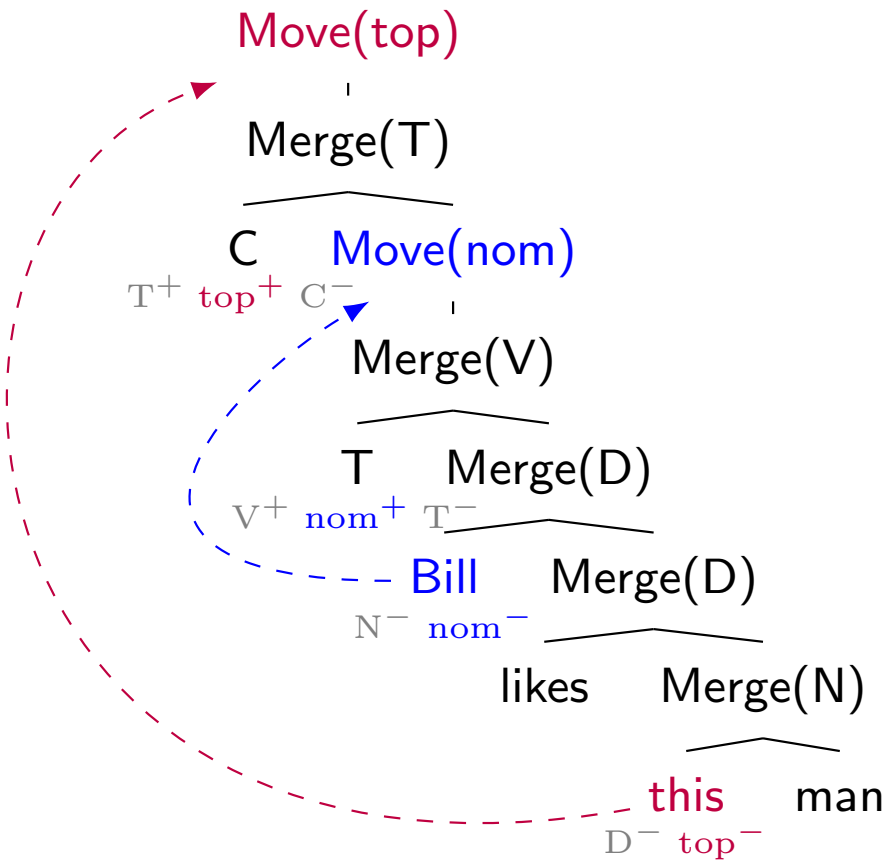
Move as TSL Merge

- Project tree tier for each movement type x.
  1. Project every lexical item with x^-.
  2. Project every Move(x).
- Every x^- has a Move node as its mother.
- Every Move node has exactly one x^- among its daughters.



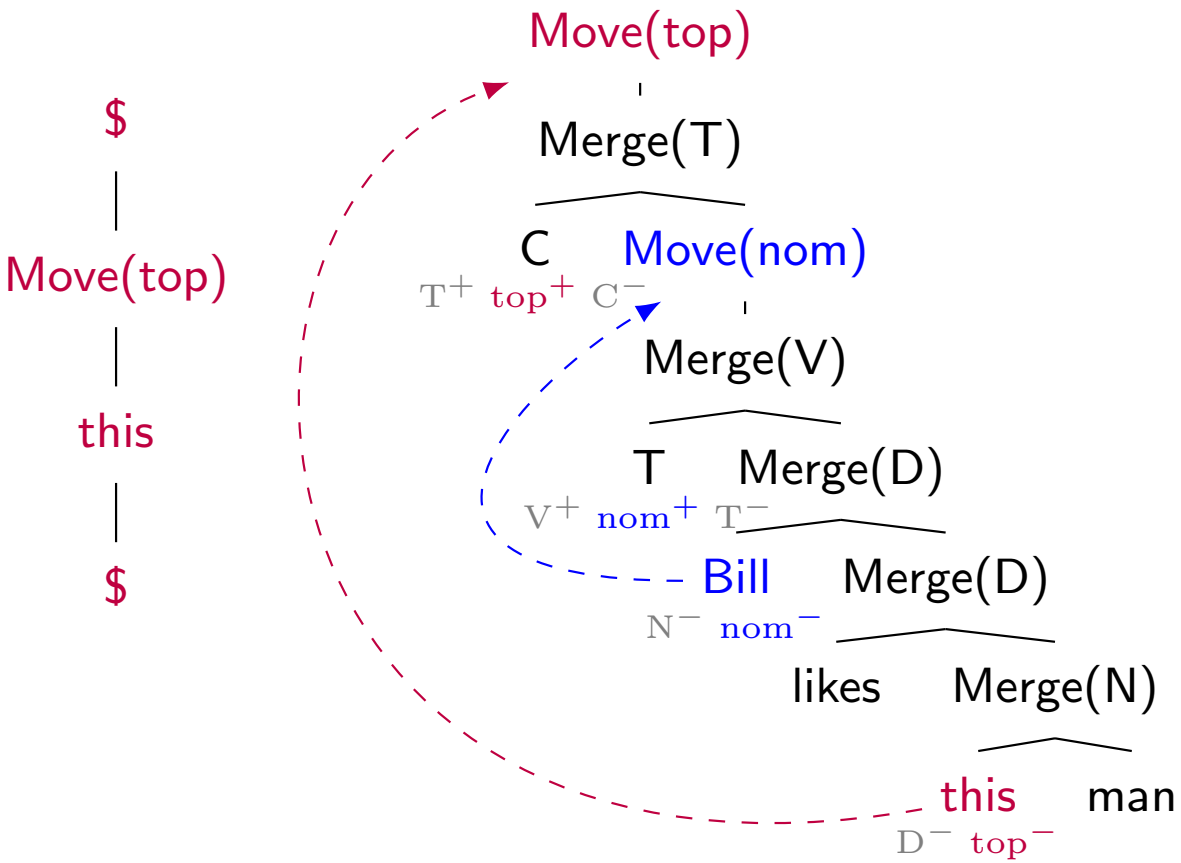
Move as TSL Merge

- Project tree tier for each movement type x.
  1. Project every lexical item with x^-.
  2. Project every Move(x).
- Every x^- has a Move node as its mother.
- Every Move node has exactly one x^- among its daughters.



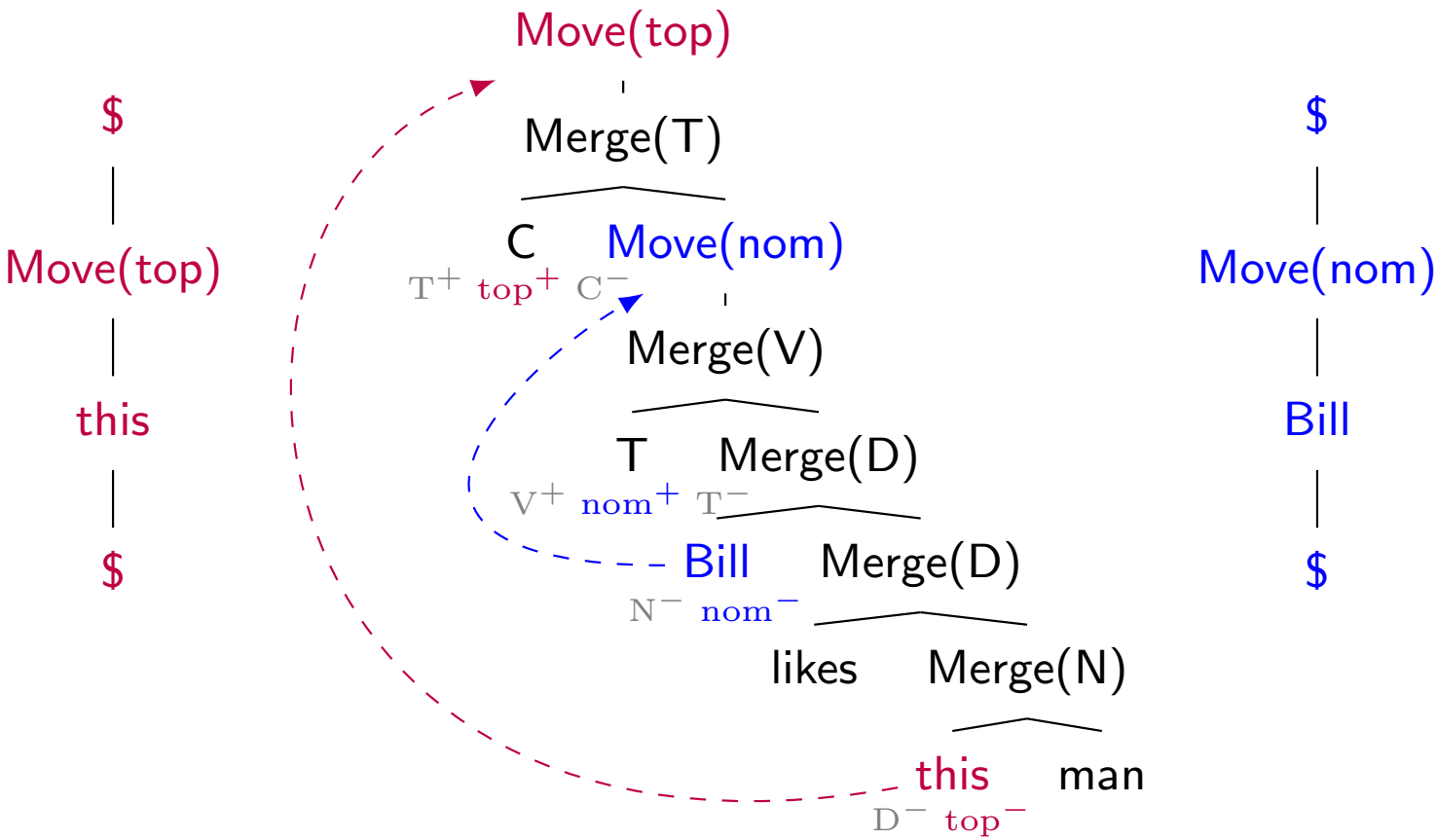
Move as TSL Merge

- Project tree tier for each movement type x.
  1. Project every lexical item with x^-.
  2. Project every Move(x).
- Every x^- has a Move node as its mother.
- Every Move node has exactly one x^- among its daughters.

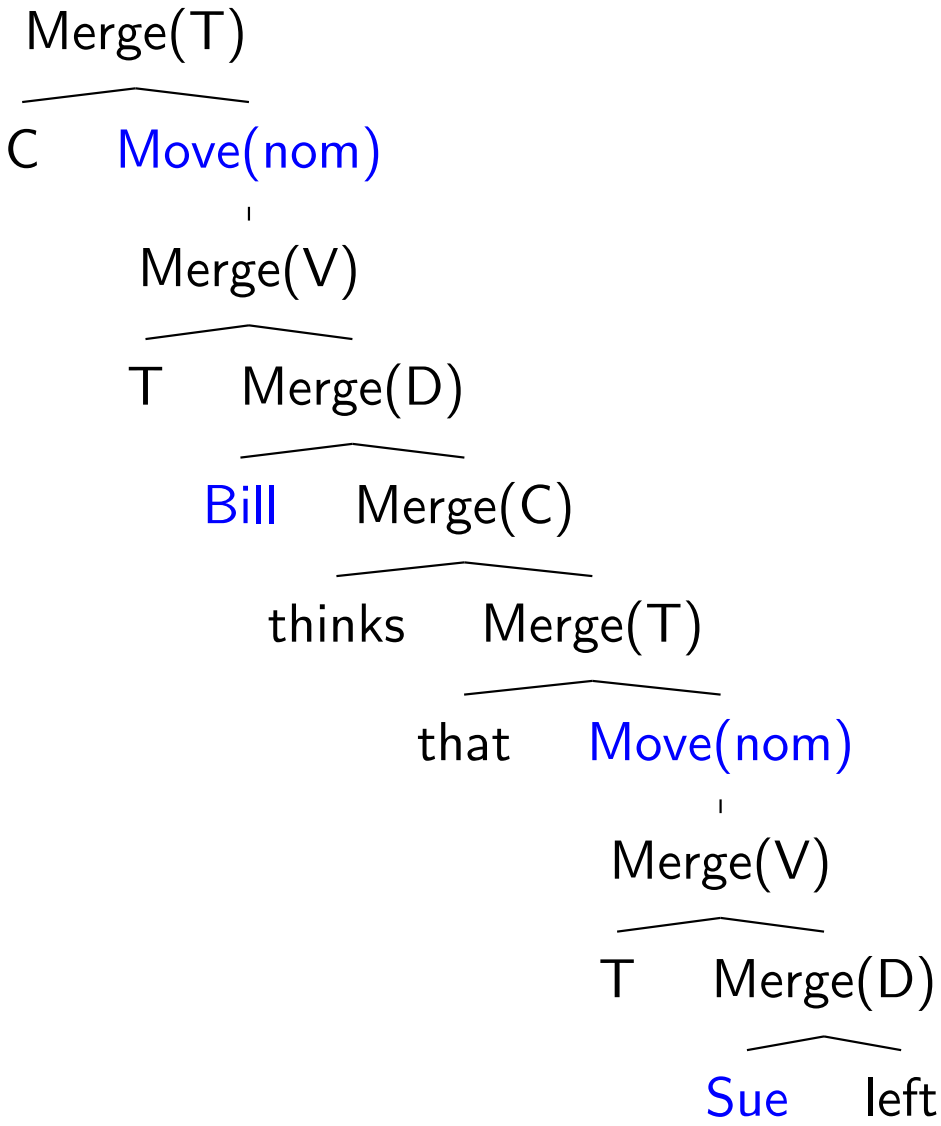


Move as TSL Merge

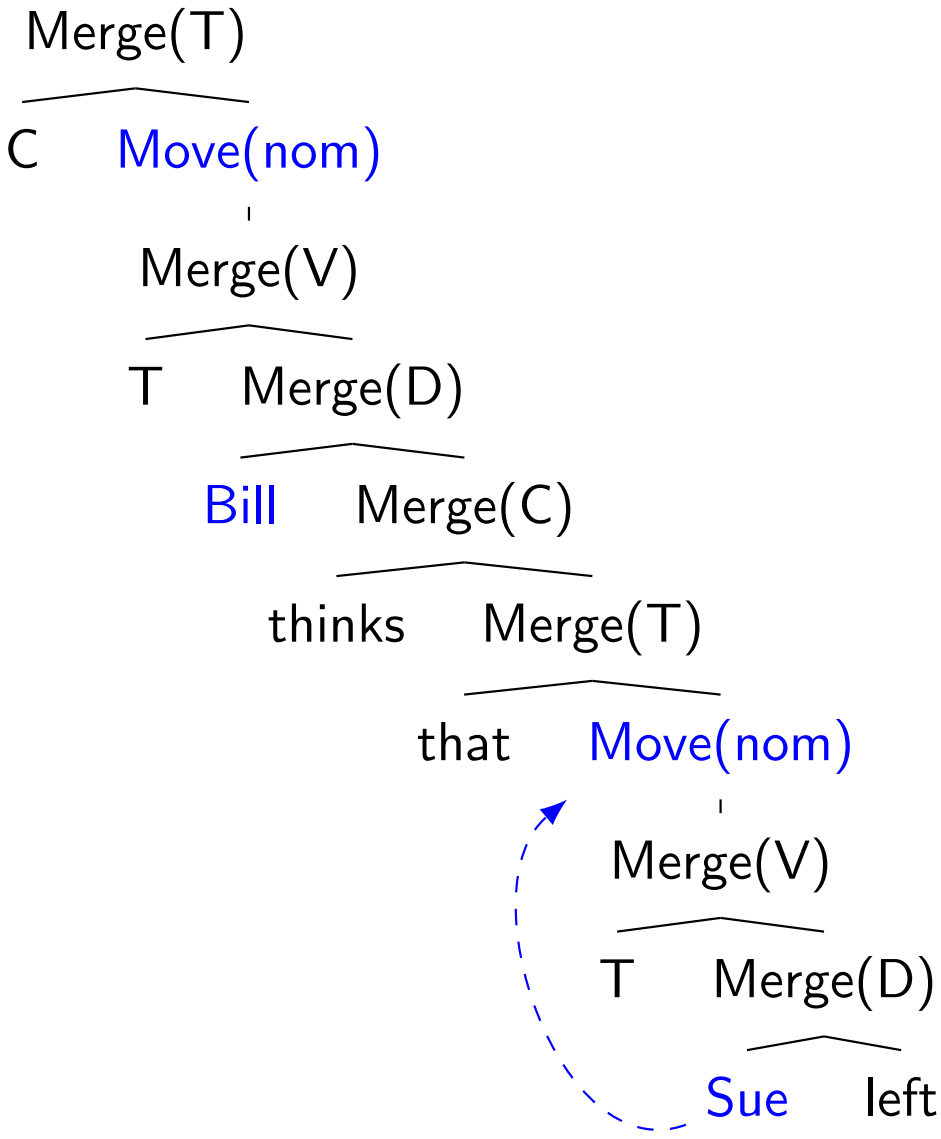
- Project tree tier for each movement type x.
  1. Project every lexical item with x^-.
  2. Project every Move(x).
- Every x^- has a Move node as its mother.
- Every Move node has exactly one x^- among its daughters.



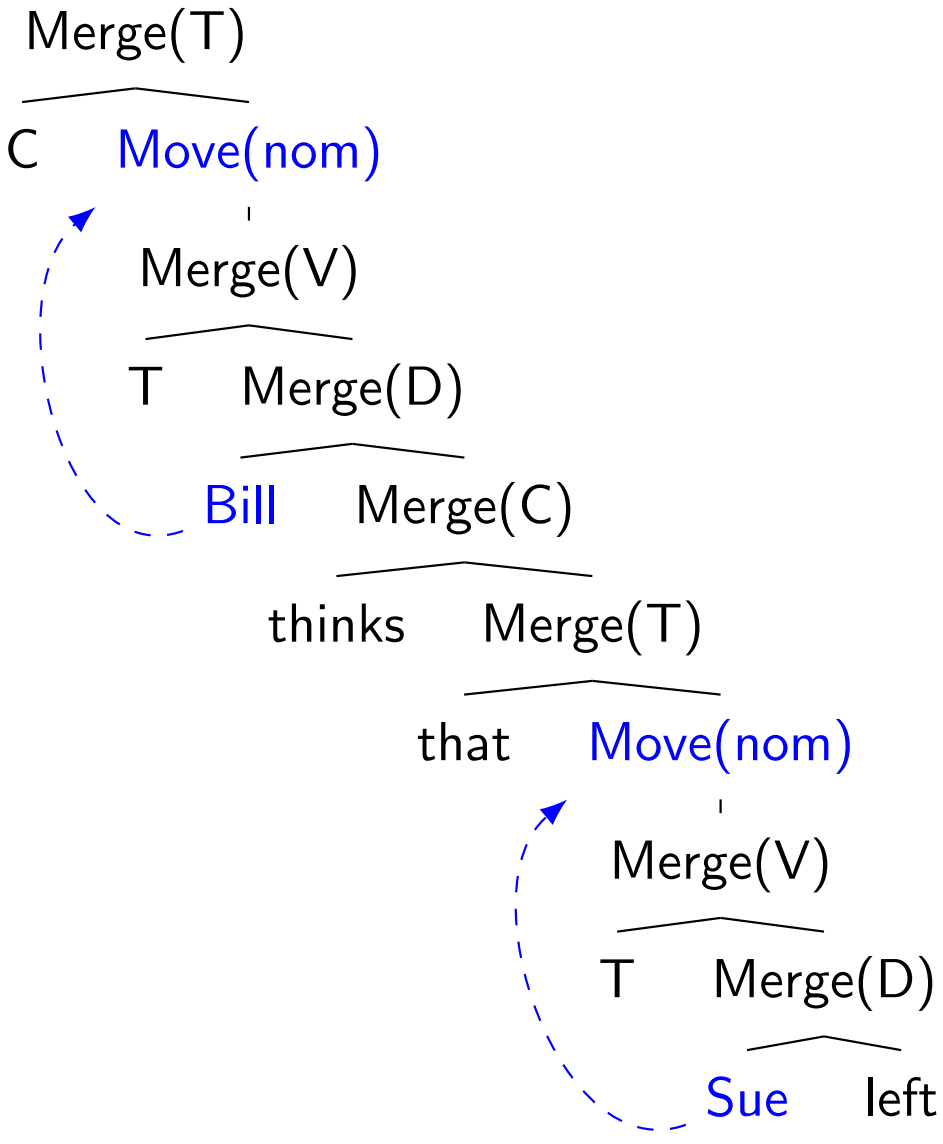
A Tier With Multiple Movers



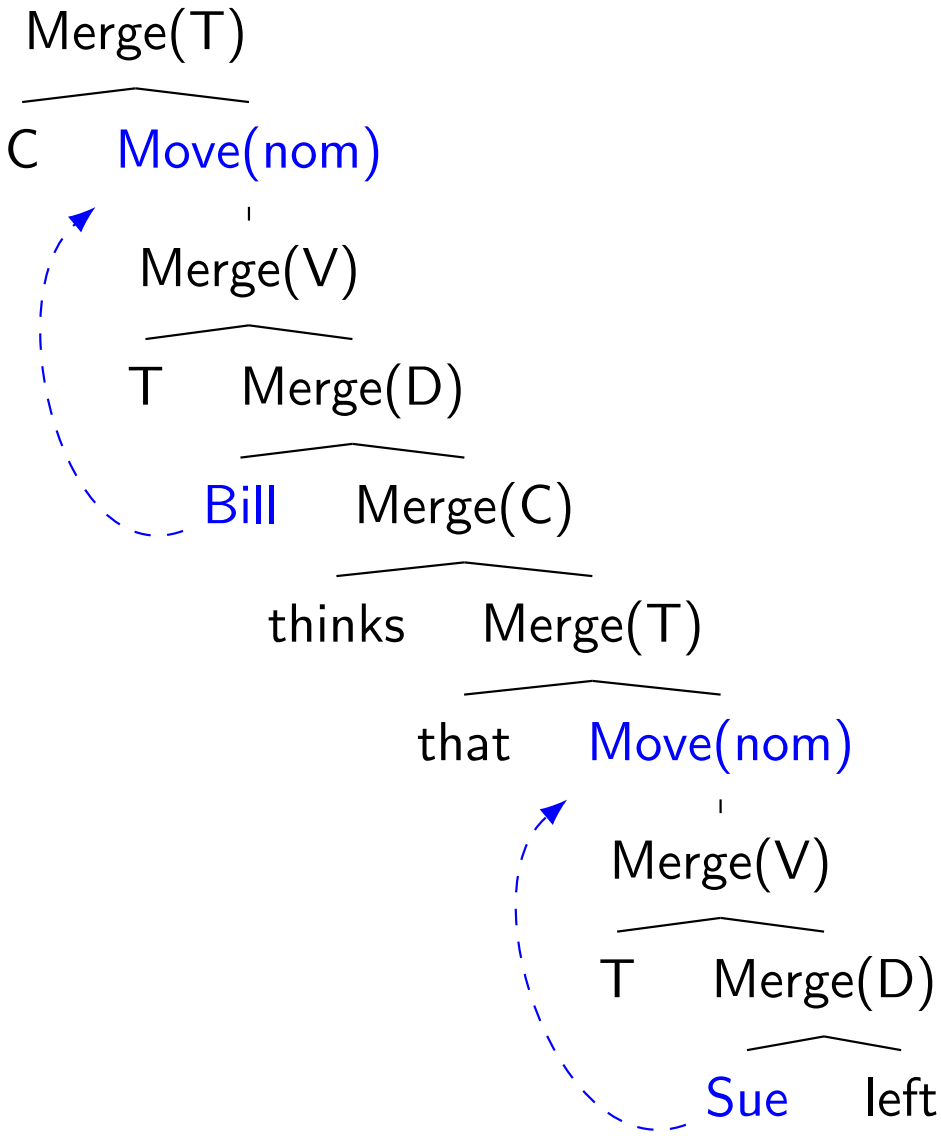
A Tier With Multiple Movers

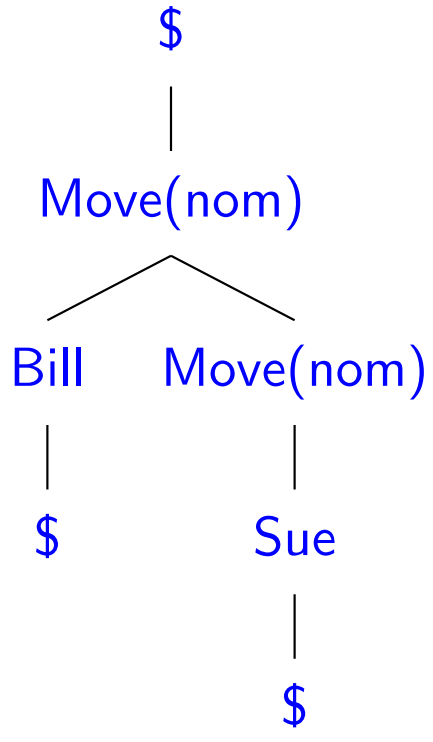


A Tier With Multiple Movers

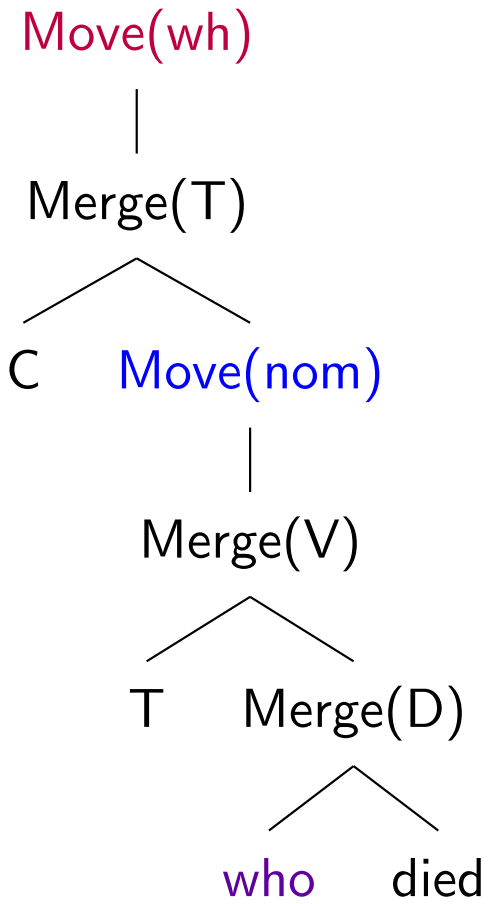


A Tier With Multiple Movers

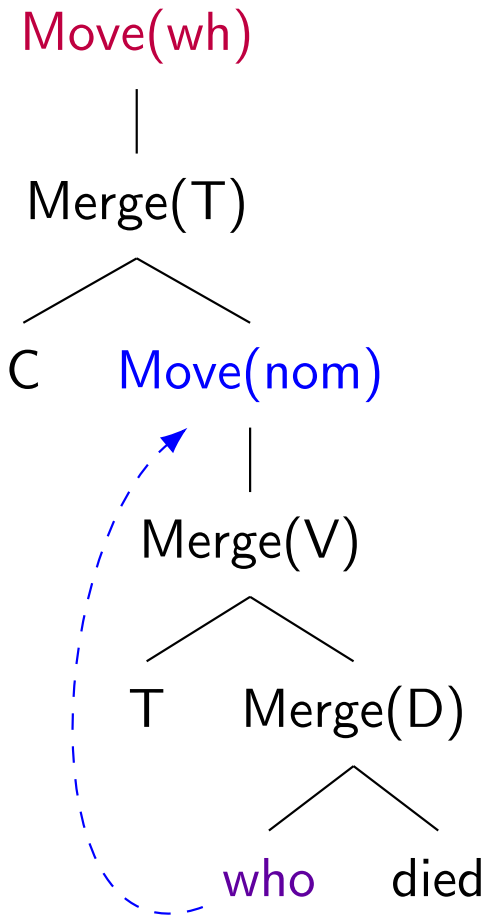




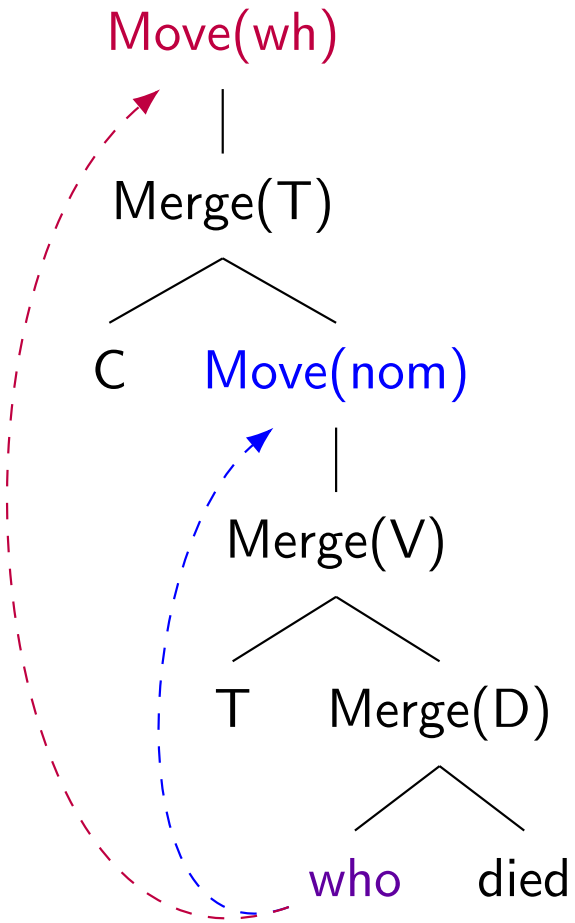
Consecutive Movement Works the Same



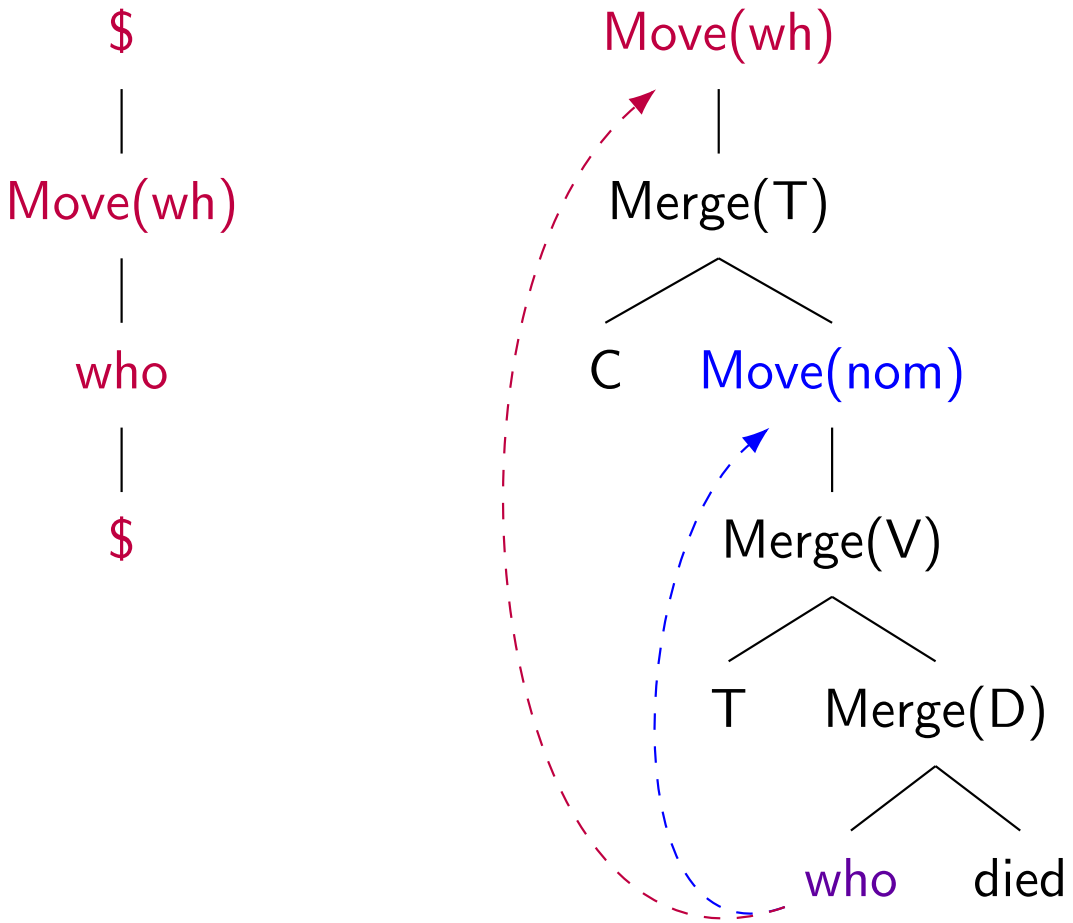
Consecutive Movement Works the Same



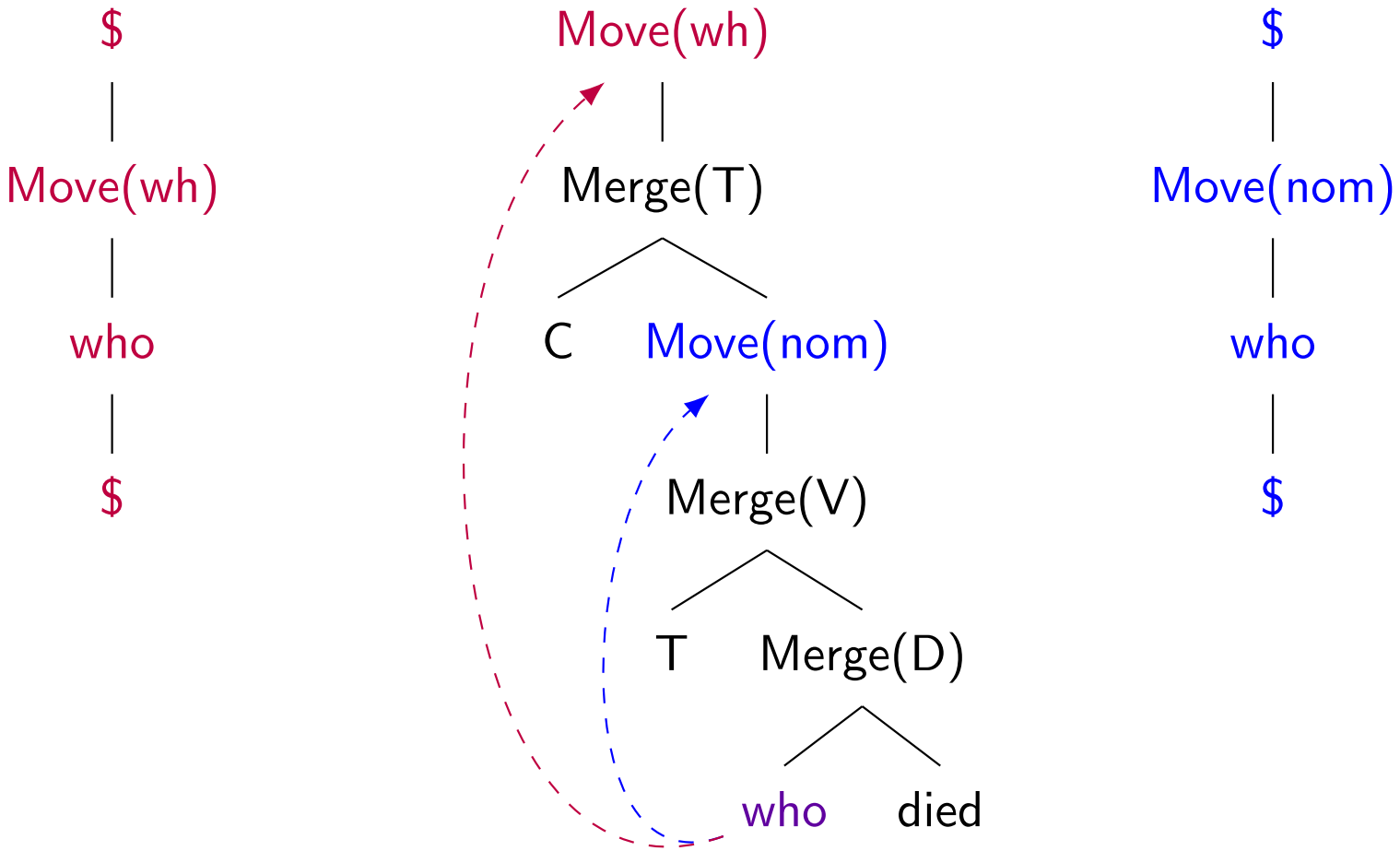
Consecutive Movement Works the Same



Consecutive Movement Works the Same



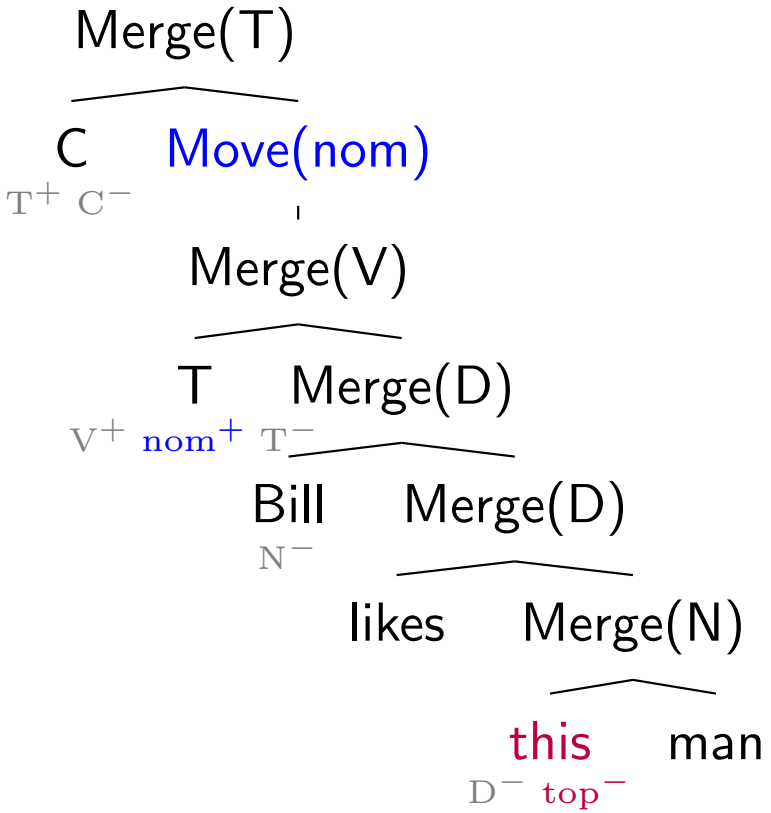
Consecutive Movement Works the Same



Blocking Simple Cases of Illicit Movement

The derivation below has

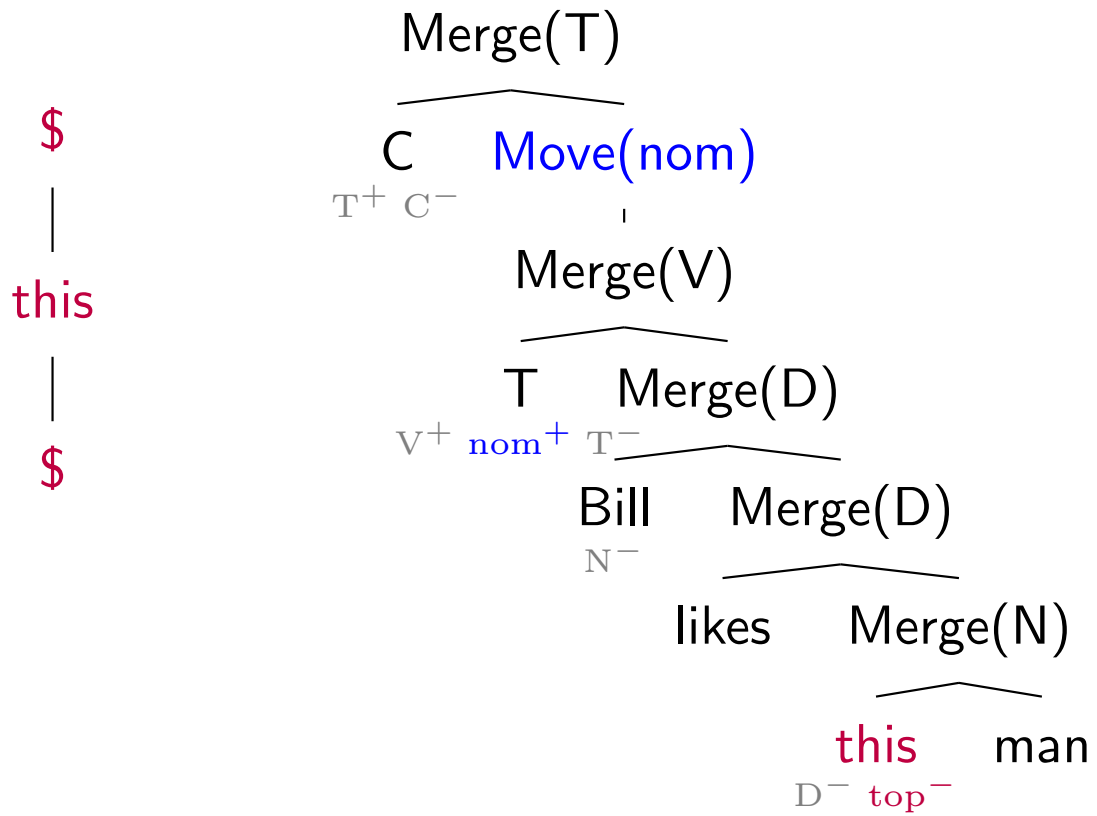
1. a nom-landing site with no mover,
2. a top-mover with no landing site.



Blocking Simple Cases of Illicit Movement

The derivation below has

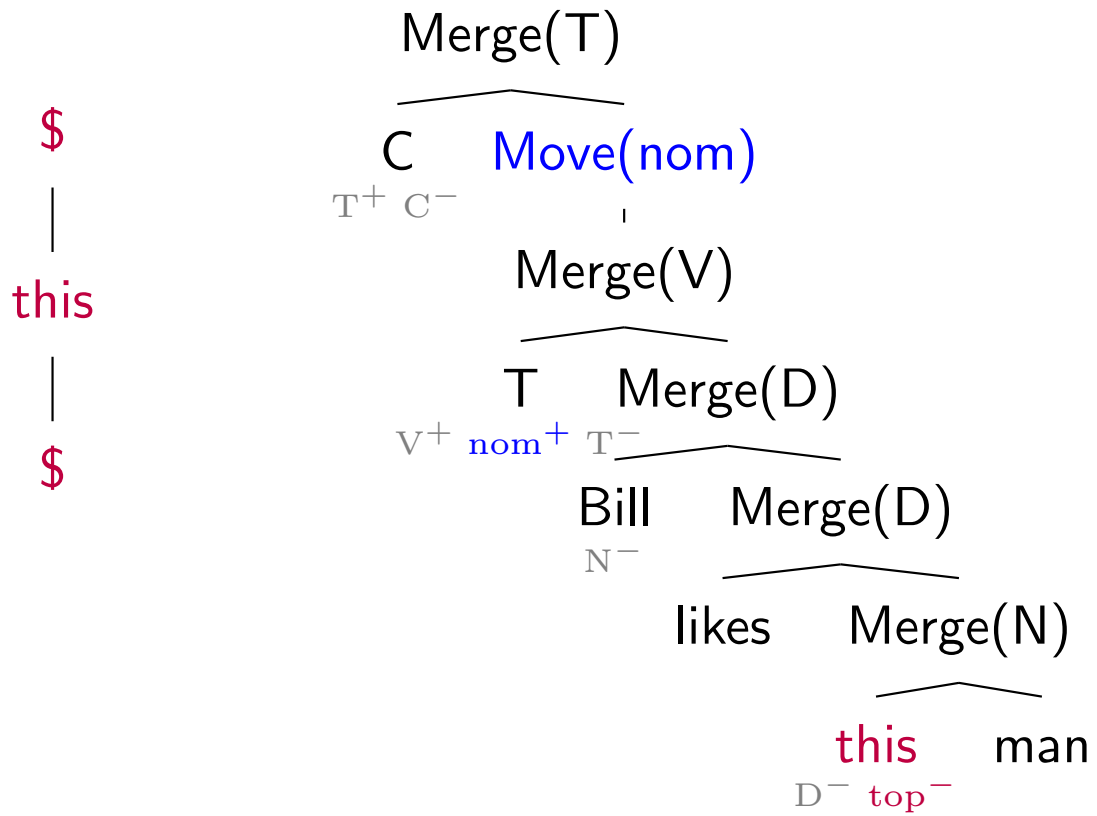
1. a nom-landing site with no mover,
2. a top-mover with no landing site.



Blocking Simple Cases of Illicit Movement

The derivation below has

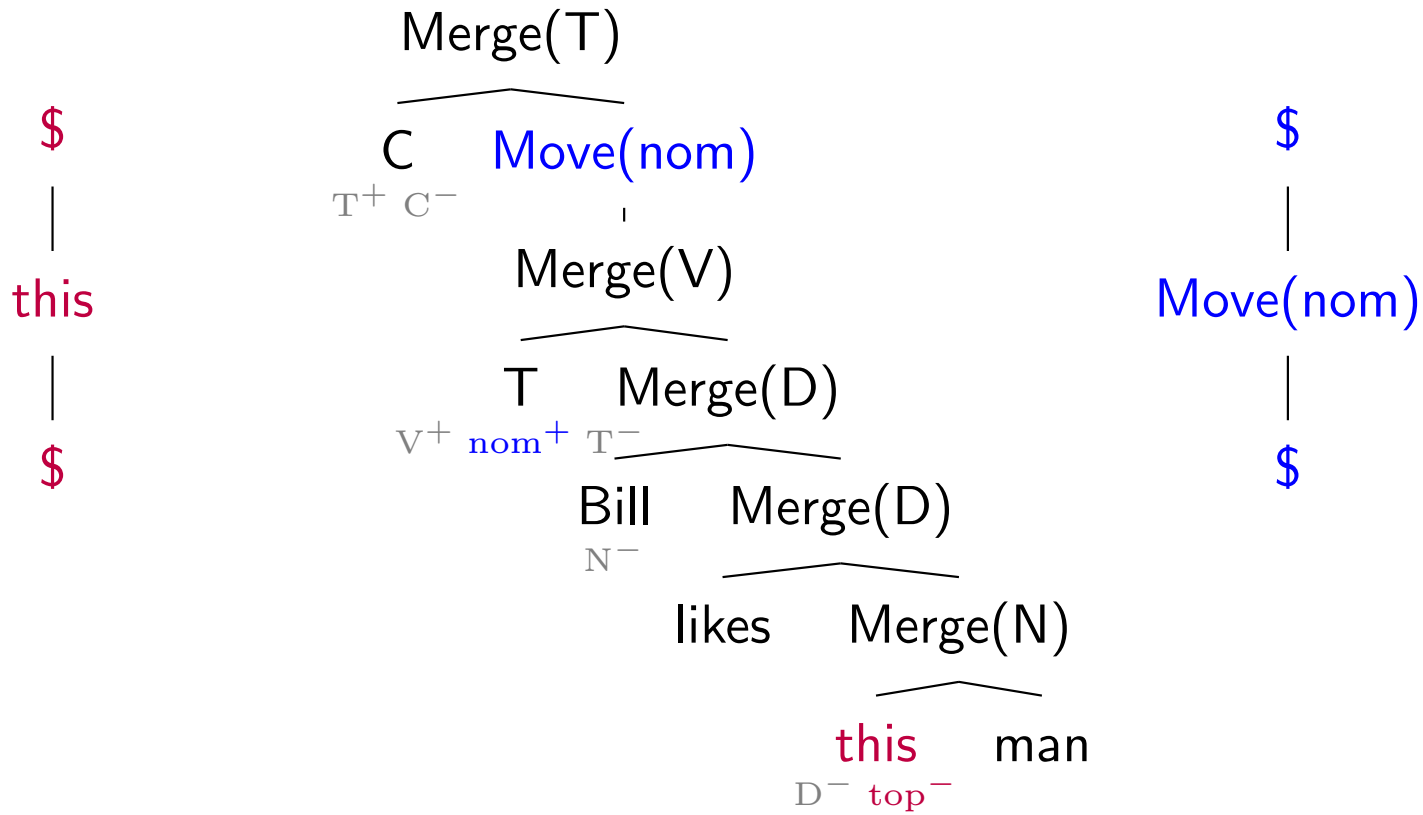
1. a nom-landing site with no mover,
2. a top-mover with no landing site.



Blocking Simple Cases of Illicit Movement

The derivation below has

1. a nom-landing site with no mover,
2. a top-mover with no landing site.



The TSL-2 Core of Merge and Move

TSL Strategy for Merge

- Project tree tier for each **category** X.
- Every X^- has a **Merge** node as its mother.
- Every **Merge** node has exactly one X^- among its daughters.

TSL Strategy for Move (Spoilers: it's exactly the same)

- Project tree tier for each **movement type** x.
- Every x^- has a **Move** node as its mother.
- Every **Move** node has exactly one x^- among its daughters.

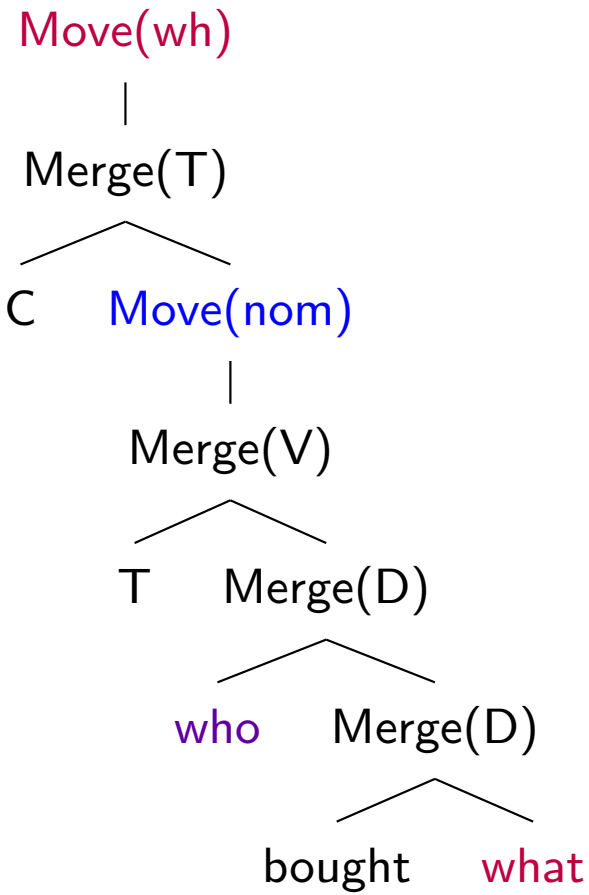
Note: constraints again **highly local**

The Minimality of Merge and Move

- TSL-2 is the weakest non-trivial TSL class.
- TSL-1 is equivalent to SL-1.
  (Tiers don't help if you only look at individual nodes)
- TSL-2 is the minimal step from local to unbounded.
- Merge and Move are **maximally non-trivially simple**.

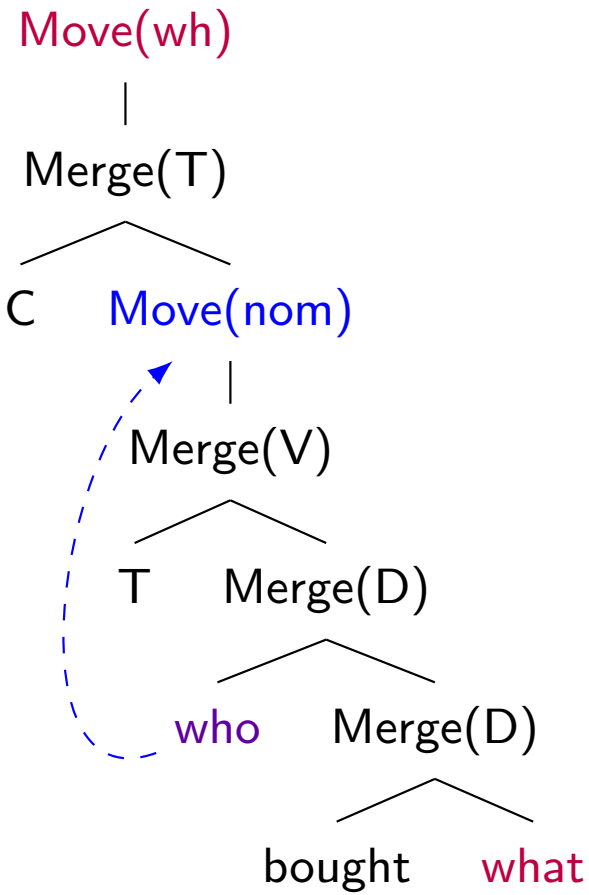
Multiple wh-Movement

- The "exactly one" part is actually two separate conditions:
  1. Every **Move** node has **at least one** x^{-}-daughter.
  2. Every **Move** node has **at most one** x^{-}-daughter.
- Dropping the second requirement gives us multiple movement.



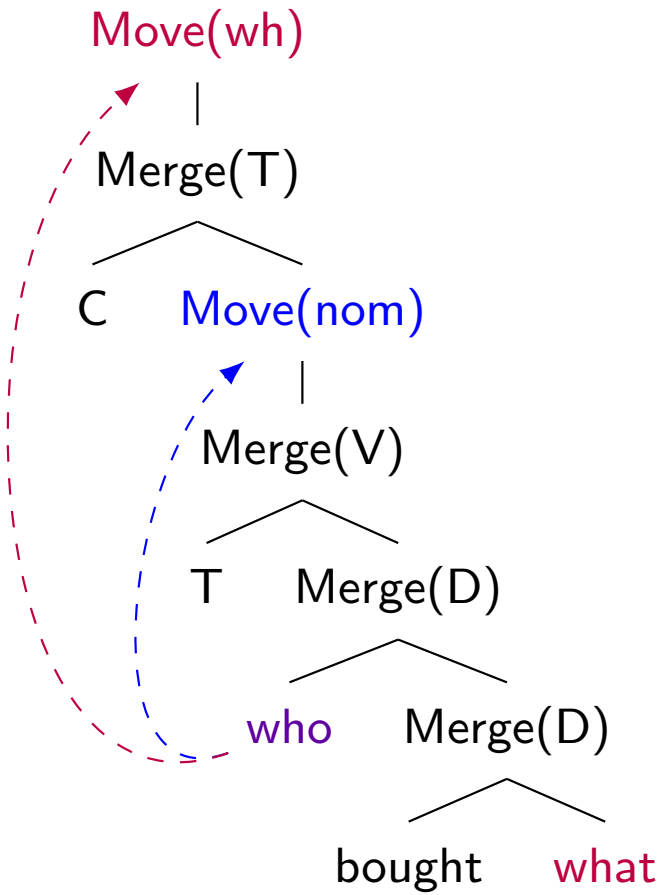
Multiple wh-Movement

- The "exactly one" part is actually two separate conditions:
  1. Every **Move** node has **at least one** x^{-}-daughter.
  2. Every **Move** node has **at most one** x^{-}-daughter.
- Dropping the second requirement gives us multiple movement.



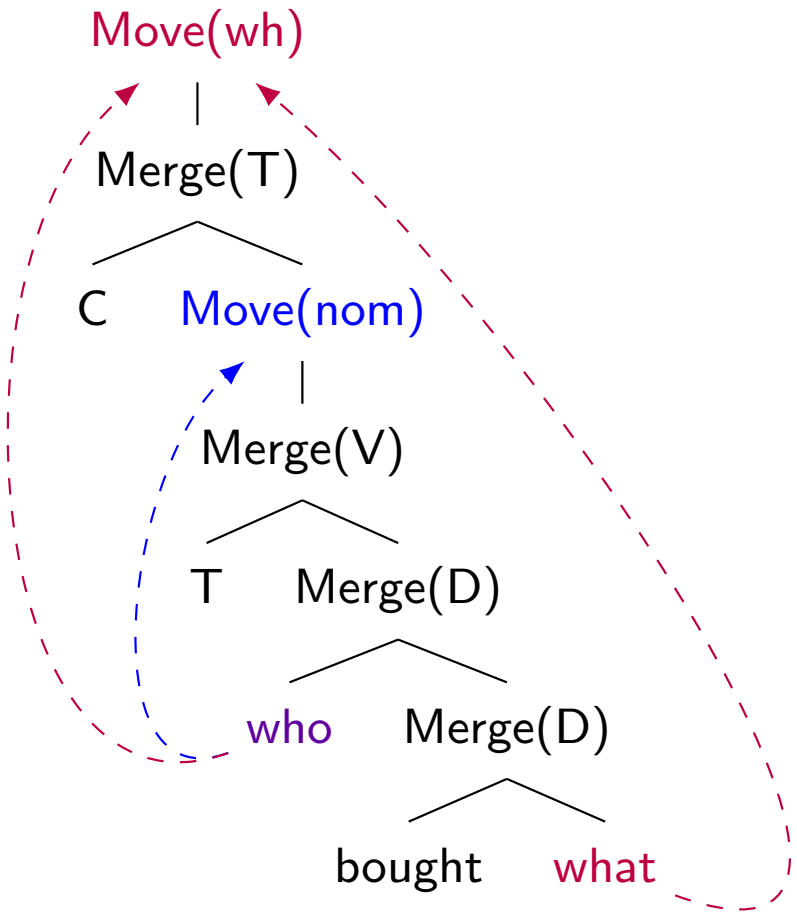
Multiple wh-Movement

- The "exactly one" part is actually two separate conditions:
  1. Every **Move** node has **at least one** x^{-}-daughter.
  2. Every **Move** node has **at most one** x^{-}-daughter.
- Dropping the second requirement gives us multiple movement.



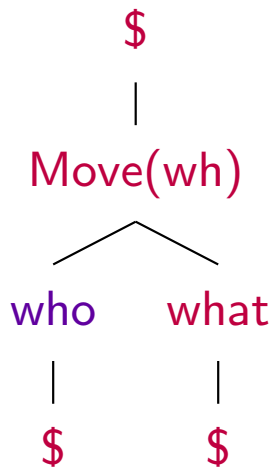
Multiple wh-Movement

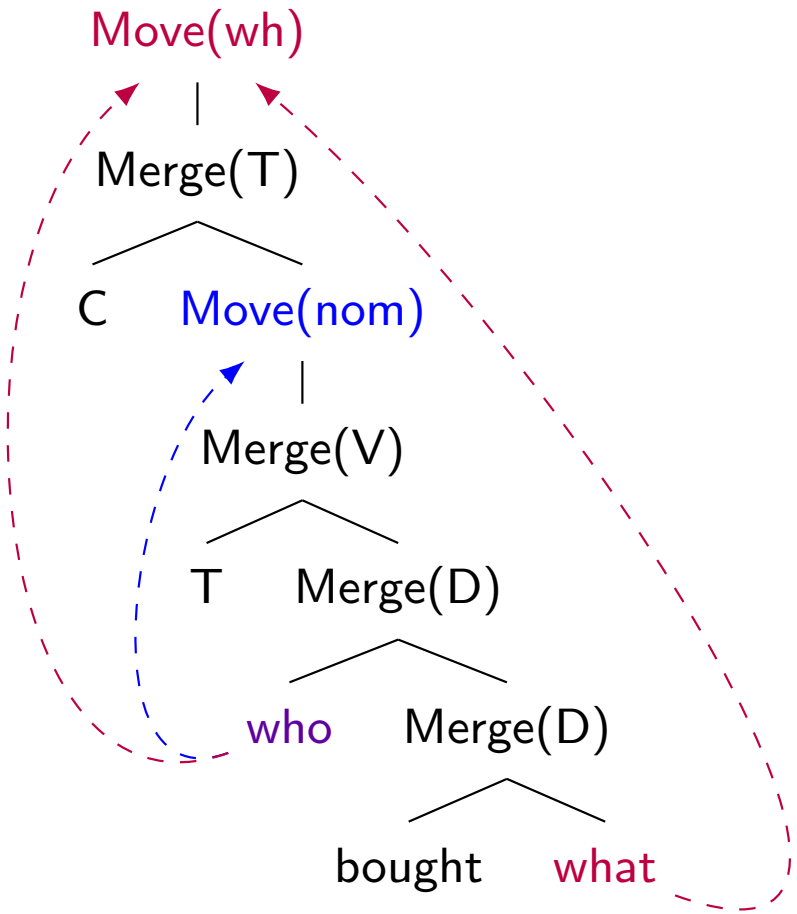
- The "exactly one" part is actually two separate conditions:
  1. Every **Move** node has **at least one** x^--daughter.
  2. Every **Move** node has **at most one** x^--daughter.
- Dropping the second requirement gives us multiple movement.



Multiple wh-Movement

- The "exactly one" part is actually two separate conditions:
  1. Every **Move** node has **at least one** x^--daughter.
  2. Every **Move** node has **at most one** x^--daughter.
- Dropping the second requirement gives us multiple movement.





What Gives?

- A system that has the means for standard movement, also has the means for multiple wh-movement.
- **Caution**: this only holds for the derivational part.

Additional Complications of Multiple wh-Movement

- How do we determine c-command/scope?
- How do we linearize the movers?
- Why isn't there multiple Merge?

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.

Reminder: Blocking is TSL

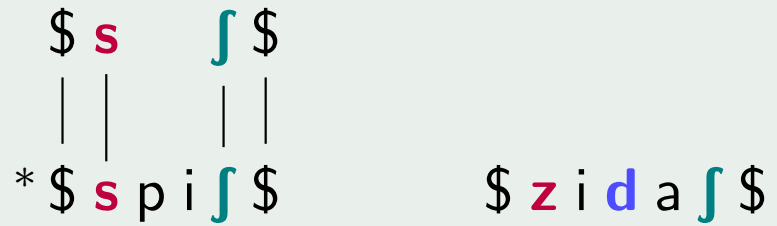
Example: Slovenian (Jurgec 2011; McMullin 2016)

* $ s p i ʃ $ $ z i d a ʃ $

- Generalizing from phonology to syntax:
  islands are pushy guys that add useless nodes to tiers.

Reminder: Blocking is TSL

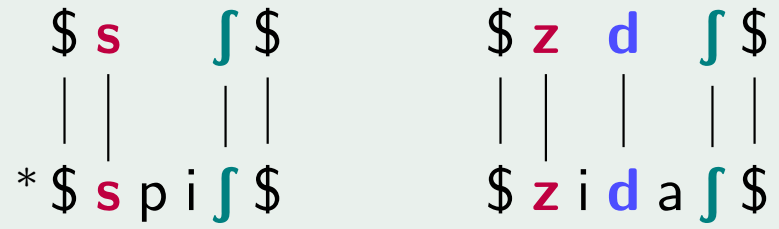
Example: Slovenian (Jurgec 2011; McMullin 2016)



- Generalizing from phonology to syntax:
  islands are pushy guys that add useless nodes to tiers.

Reminder: Blocking is TSL

Example: Slovenian (Jurgec 2011; McMullin 2016)



- Generalizing from phonology to syntax:
  islands are pushy guys that add useless nodes to tiers.

Island Examples

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

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

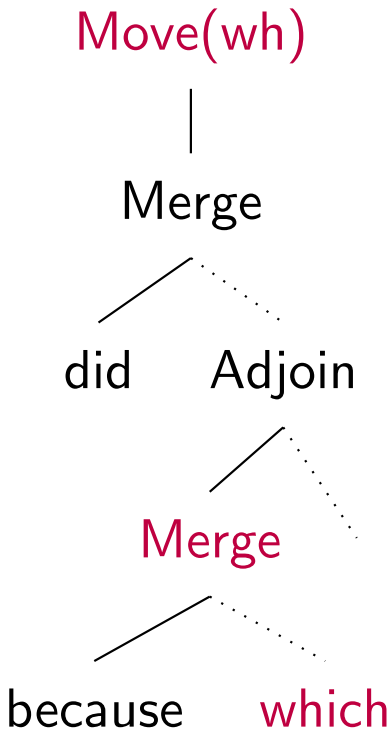
(3) Which car did John drive Mary crazy [**while** trying to fix t].

Island Examples

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

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

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

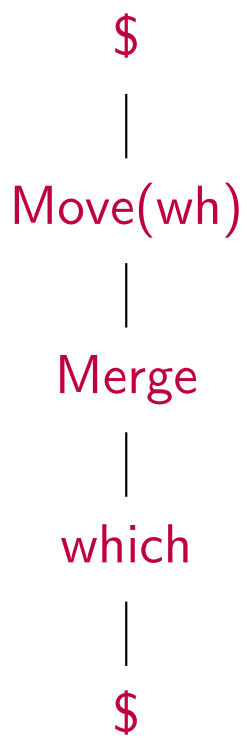


Island Examples

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

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

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

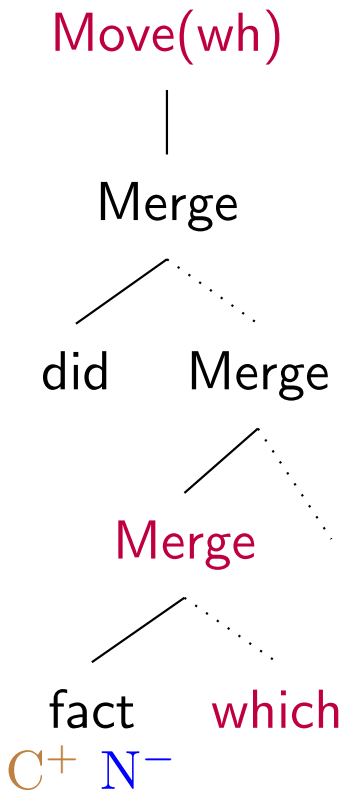


Island Examples

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

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

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

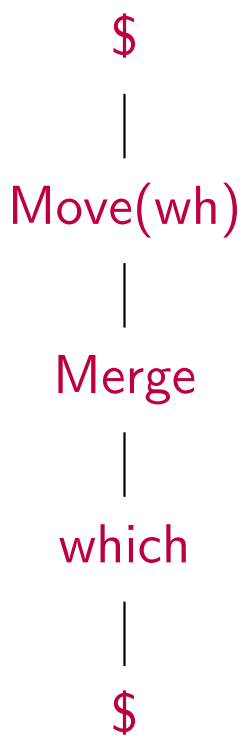


Island Examples

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

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

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

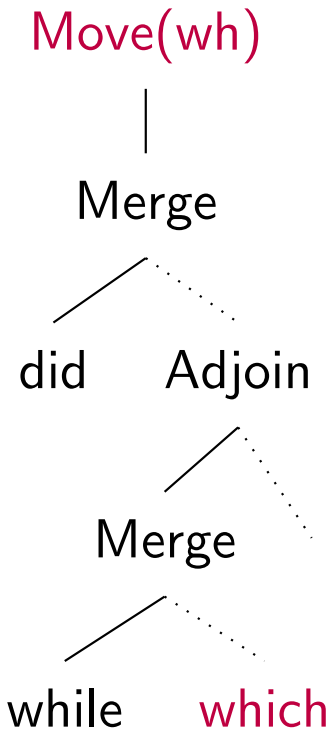


Island Examples

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

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

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

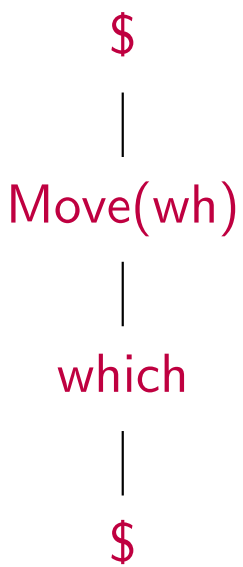


Island Examples

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

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

(3) 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 is very limited.
- 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 same clause."
  - **Rationed islands**
    "Only one adjunct per clause can be an island."
  - **Discerning islands**
    "Adjuncts only block movers that contain an adjective."

Islands are Still Mysterious, Though

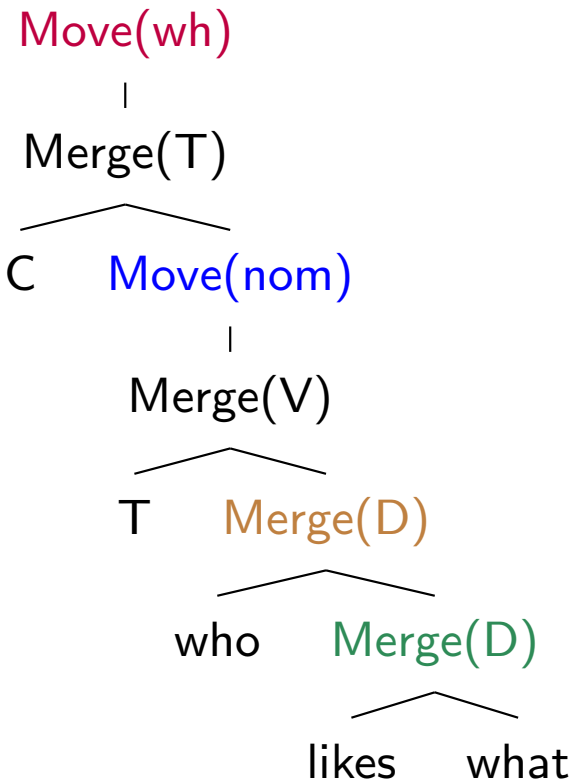
- The existence of islands is unsurprising under the TSL-view.
- What is surprising is that
  1. islands do no vary more across languages,
  2. islands form natural classes,
     (e.g. almost all adjuncts are islands, not just *because*)
  3. some things are never islands.
- There are alternative stories that address some of that.

Superiority

- Superiority effects can be analyzed as conditional islands.

Example

A wh-item that does not carry wh^- causes a Merge node to be projected onto the wh-tier.

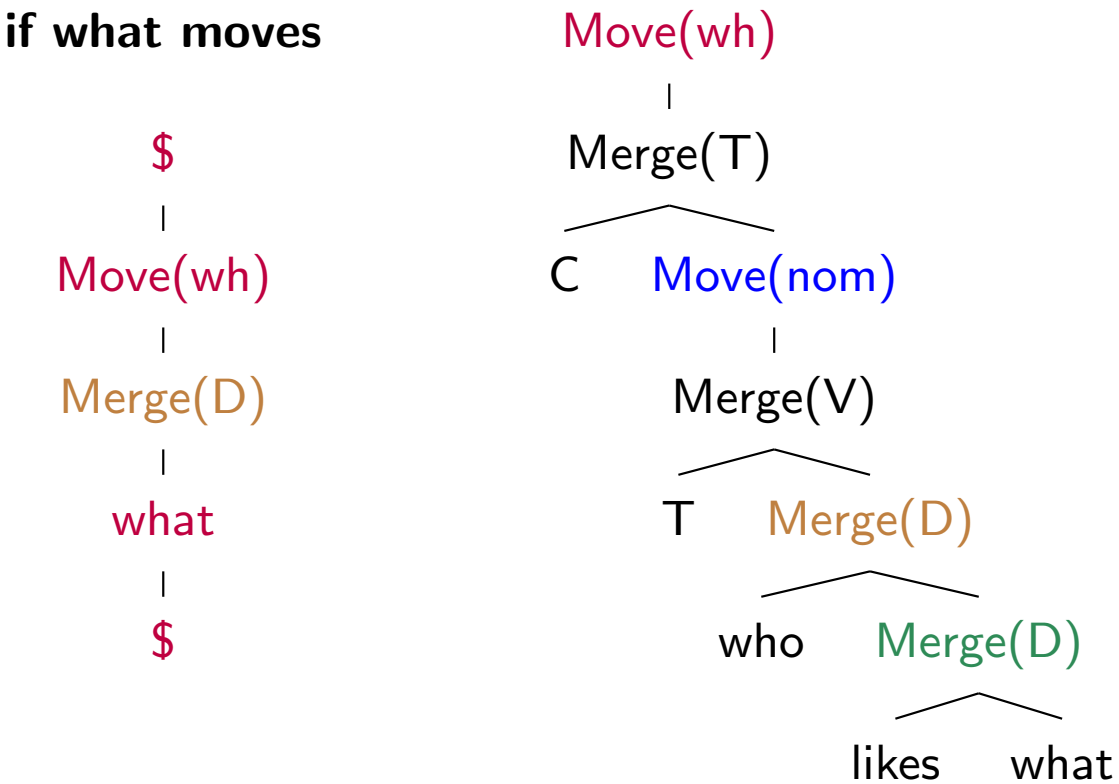


Superiority

- Superiority effects can be analyzed as conditional islands.

Example

A wh-item that does not carry wh^- causes a Merge node to be projected onto the wh-tier.

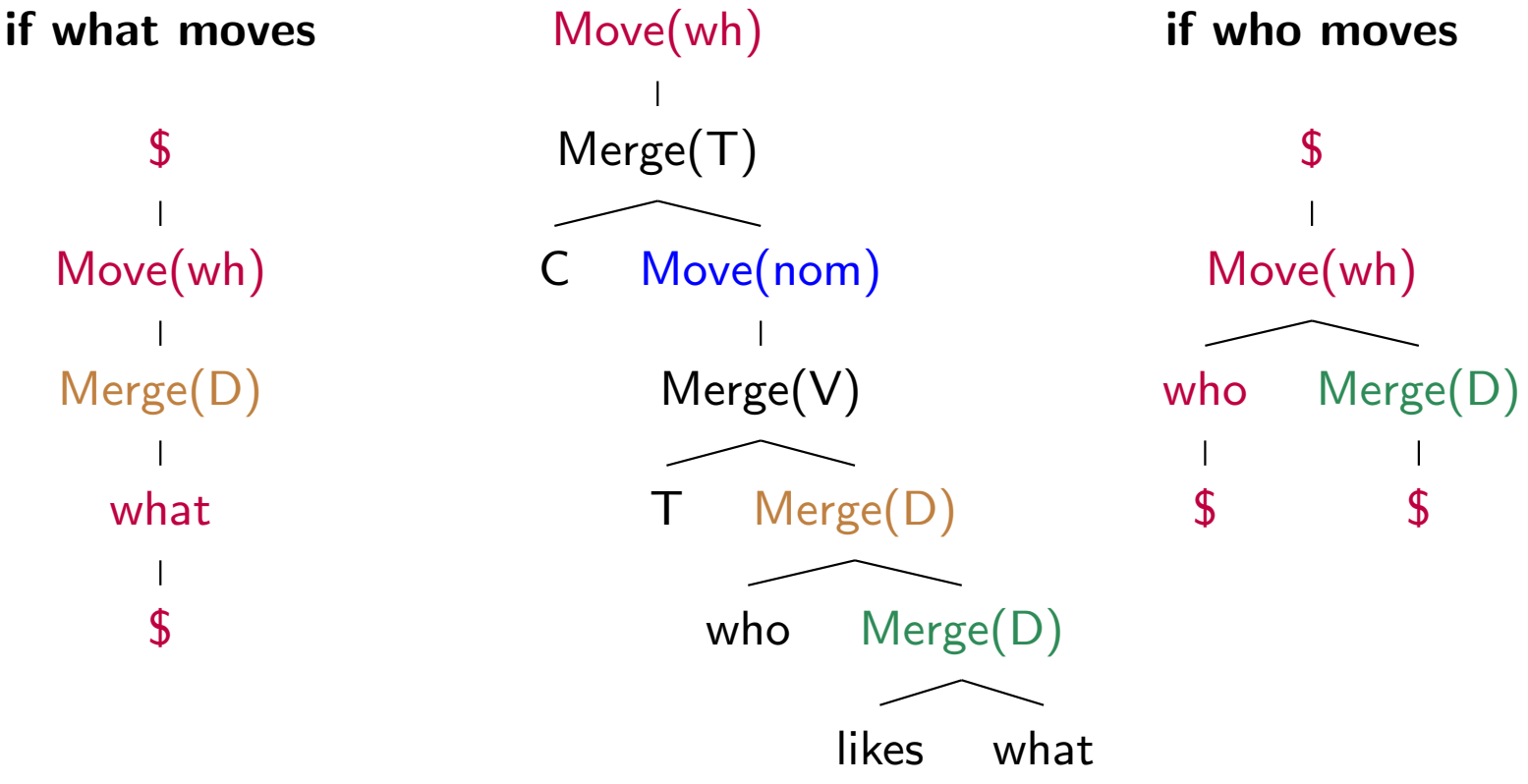


Superiority

- Superiority effects can be analyzed as conditional islands.

Example

A wh-item that does not carry wh^- causes a Merge node to be projected onto the wh-tier.



But What About Actual Movement?

- We're only handling the feature checking part of Move.
- The actual displacement is handled differently.

Two-Step Approach to Syntax

- Derivation tree is locus of syntactic dependencies.
- Derivation tree can be rewritten into
  - phrase structure tree
  - multi-dominance tree
  - LF
  - prosodic structure
  - string yield
  - much more
- Formalized as tree transduction
- Mapping to multi-dominance tree is SL with a bit of TSL

Wrapping up: Concrete Results

- SL and TSL can be generalized to trees.
- Succinctness drives us from SL Merge to TSL Merge.
- TSL Merge paves the way for
  - movement
  - adjunction
  - multiple wh-movement
  - island effects
  - superiority

Join the Program

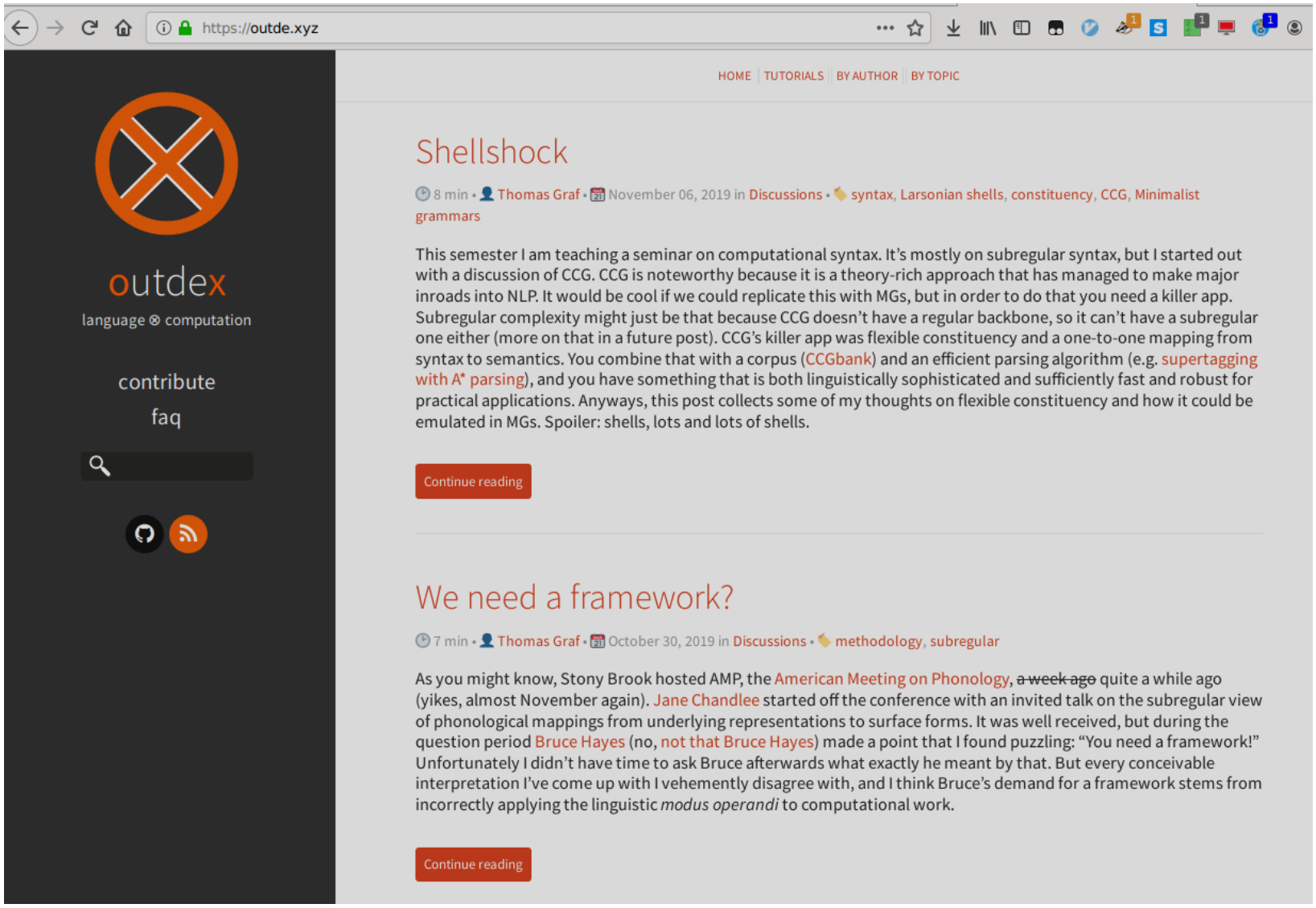
Everybody has something to contribute!

- Do you have data that contradicts current predictions?
- In-depth analysis of specific phenomena
- Grammar fragments

Follow-Up Projects

1. Agree
2. Contiguity theory
3. Parasitic gaps
4. Smuggling
5. Movement as a transformation (Graf 2020)
6. Applications of weighted/probabilistic TSL for trees

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



Learning More About Tree Land

1. **TSL syntax:** Graf (2012, 2018); 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)

Acknowledgments

This work was supported by the National Science Foundation under Grant No. BCS-1845344.

Appendix

References I

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

Epstein, Samuel D., Erich M. Groat, Ruriko Kawashima, and Hisatsugu Kitahara. 1998. *A derivational approach to syntactic relations*. Oxford: Oxford University Press.

Graf, Thomas. 2012. Locality and the complexity of Minimalist derivation tree languages. In *Formal Grammar 2010/2011*, ed. Philippe de Groot and Mark-Jan Nederhof, volume 7395 of *Lecture Notes in Computer Science*, 208–227. Heidelberg: Springer. URL `http://dx.doi.org/10.1007/978-3-642-32024-8_14`.

Graf, Thomas. 2018. 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. 2020. Curbing feature coding: Strictly local feature assignment. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2020*, 362–371.

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`.

References II

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.

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.

Jurgec, Peter. 2011. *Feature spreading 2.0: A unified theory of assimilation*. Doctoral Dissertation, University of Tromsø.

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

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.

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.

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.