A computational Minimalist program for syntax and sentence processing

Thomas Graf

Stony Brook University
mail@thomasgraf.net

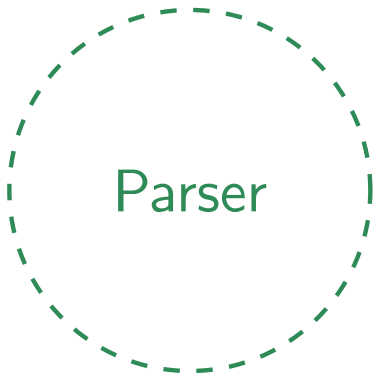
GCLing Colloquium, April 8, 2021

A better title (Fodor & Fernandez 2015)

Toward a unified theory of language knowledge and use

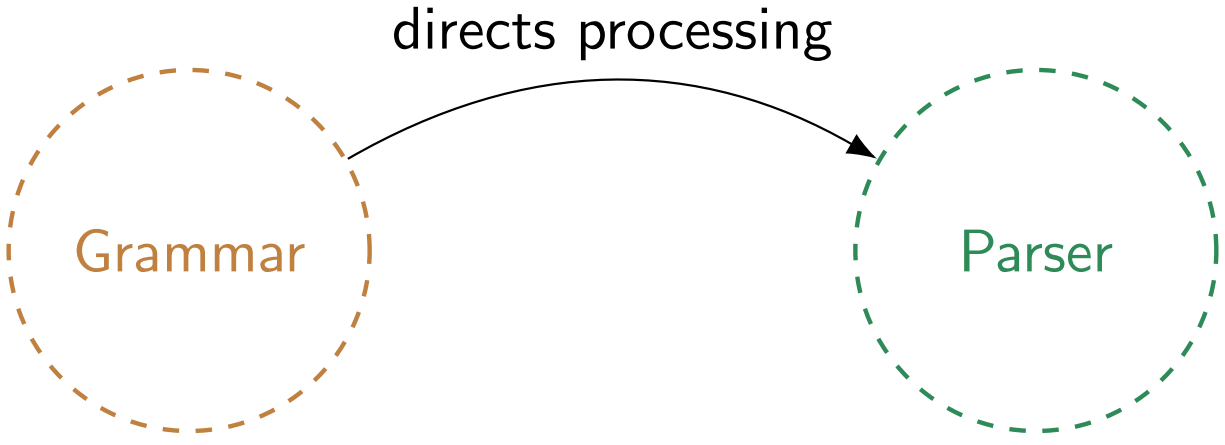
A better title (Fodor & Fernandez 2015)

Toward a unified theory of language knowledge and use



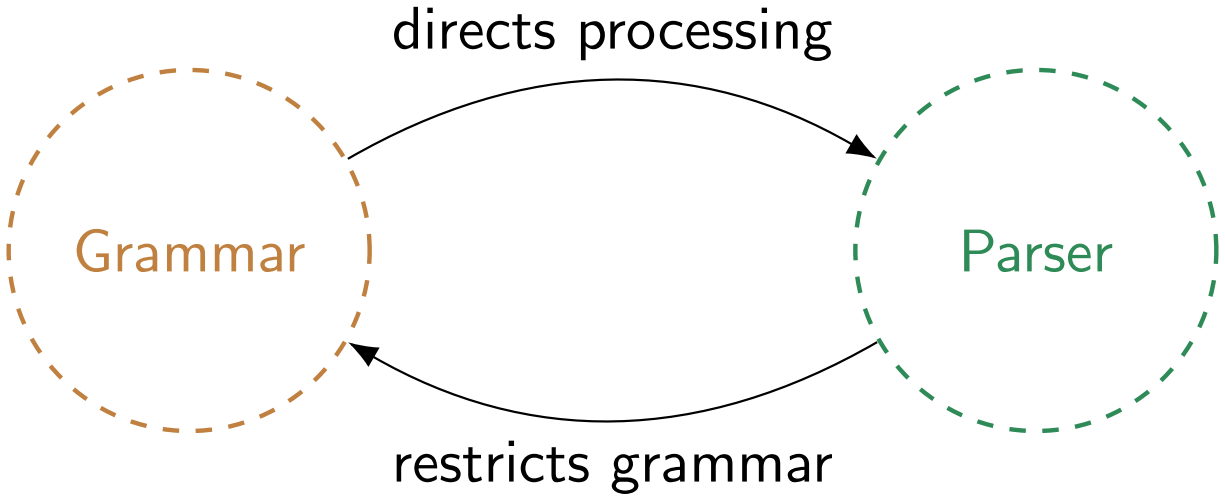
A better title (Fodor & Fernandez 2015)

Toward a unified theory of language knowledge and use



A better title (Fodor & Fernandez 2015)

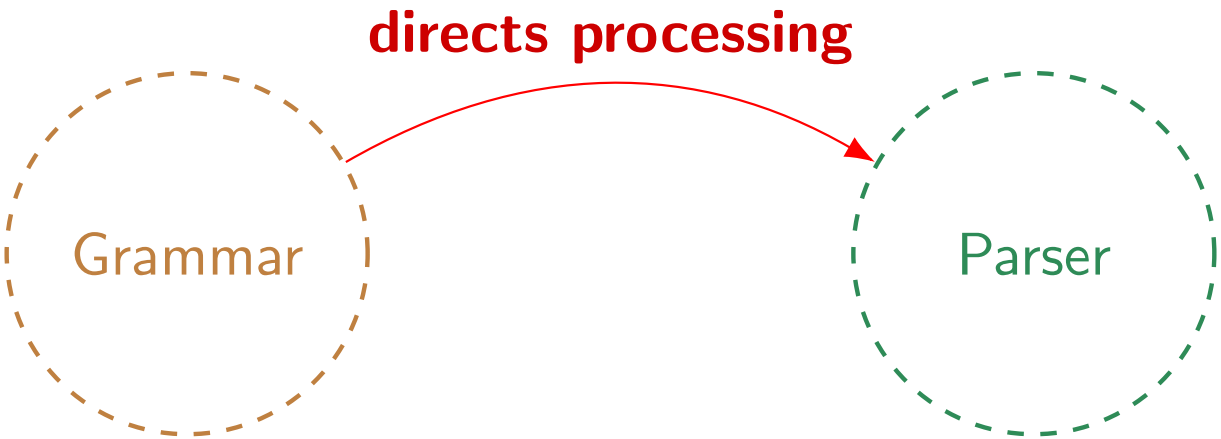
Toward a unified theory of language knowledge and use



Take-home message

- Computation is the **glue between knowledge and use**.
- Syntactic structure explains many processing differences.
- Processing requirements explain many syntactic constraints.
- Concretely:
  1. **top-down parser for Minimalist grammars**
  2. **head-sensing tree automata**

Outline


directs processing
Grammar
Parser

An overly simplistic approach to processing...

Syntactic processing is influenced by many factors:

- lexical frequency
- structural frequency
- lexical ambiguity
- structural ambiguity
- semantics
- and much more

A much simpler question

How hard is it to build the target structure
if we already know what this structure looks like?

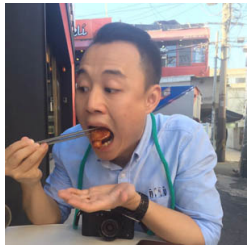
. . . works remarkably well

Structural complexity explains numerous processing preferences

- right-embedding < center-embedding
  (Greg Kobele, John Hale, Sabrina Gerth 2013)
- crossing dependencies < nested dependencies
  (Kobele et al. 2013)
- post-nominal subject RC < post-nominal object RC
  (Graf et al. 2017)
- pre-nominal subject RC < pre-nominal object RC
  (Graf et al. 2017)
- [SC [RC]] < [RC [SC]]
  (Graf et al. 2017)
- processing of stacked RCs in English and Chinese
  (Zhang 2017)

. . . way better than I ever expected

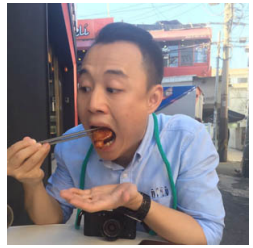
- heavy NP shift preferences
  (Liu 2018)
- attachment preferences in English and Korean
  (Lee 2019)
- structural priming effects
  (De Santo 2020)
- quantifier scope preferences
  (Pasternak and Graf 2020)




. . . way better than I ever expected

- heavy NP shift preferences
  (Liu 2018)
- attachment preferences in English and Korean
  (Lee 2019)
- structural priming effects
  (De Santo 2020)
- quantifier scope preferences
  (Pasternak and Graf 2020)

But how does it work?

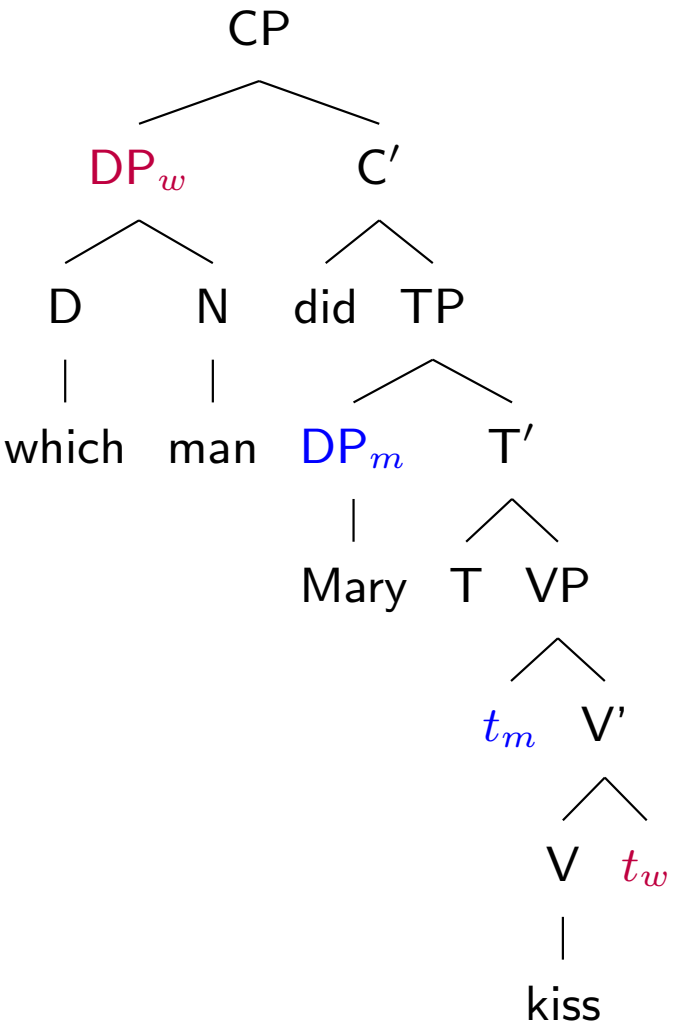


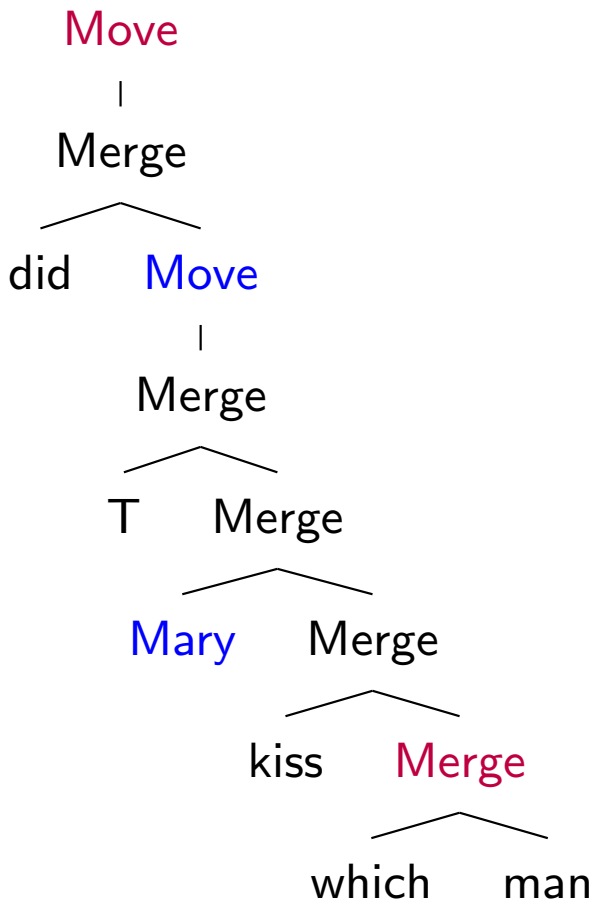

Three components

1. **Syntactic model: MGs** (Stabler 1997, 2011)
   - closely modeled after Minimalist syntax
   - allows us to use sophisticated analyses
2. **Parsing model: Recursive descent**
   - builds trees top-down, left-to-right
   - the main complication is what "left-to-right" means in syntax
3. **Cognitive linking hypothesis: Memory usage**
   - There are many ways to measure memory.
   - We mostly use **MaxTen** (longest time a node stays in memory) and **SumSize** ($\sim$ combined length of movement dependencies)

MGs: Derivation trees, not phrase structure trees

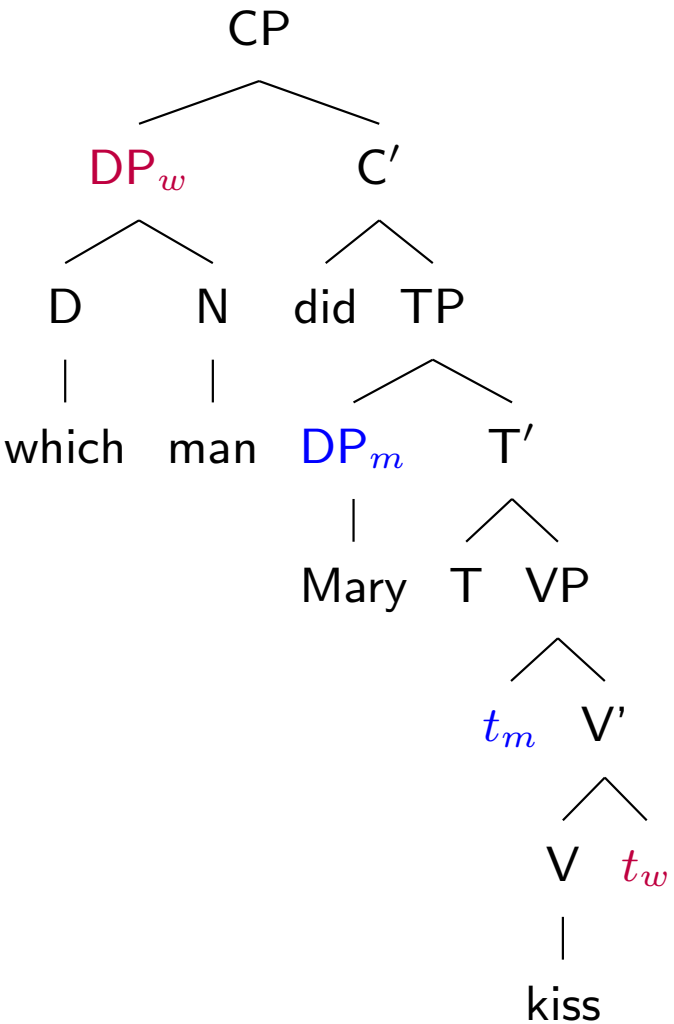
- **MGs**: syntactic structure = derivation tree
- **Advantage**: allows us to use CFG techniques

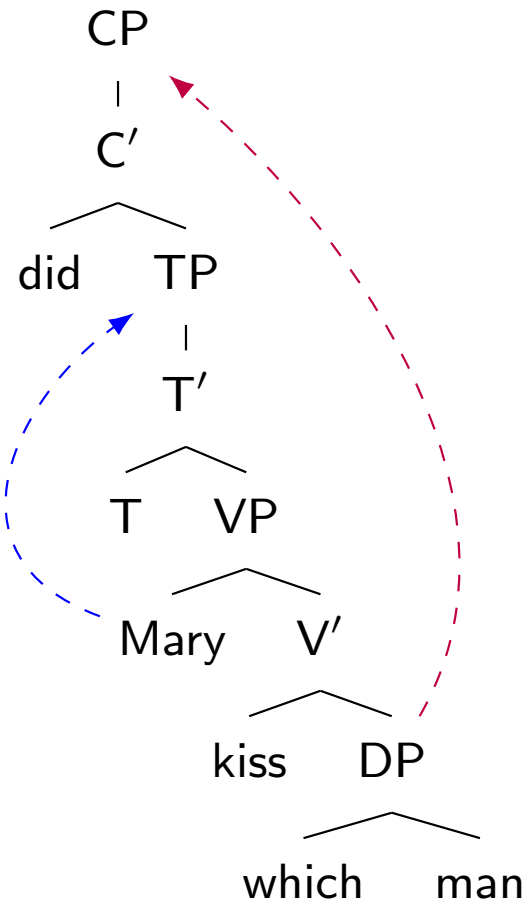




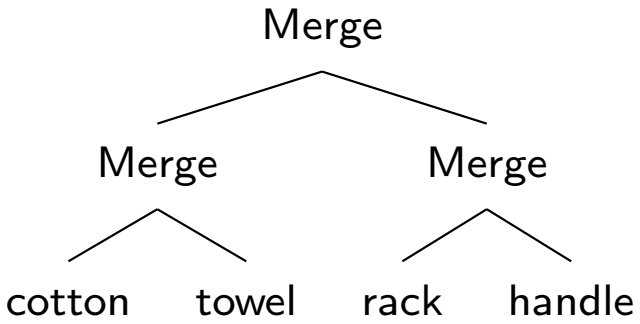
MGs: Derivation trees, not phrase structure trees

- **MGs**: syntactic structure = derivation tree
- **Advantage**: allows us to use CFG techniques

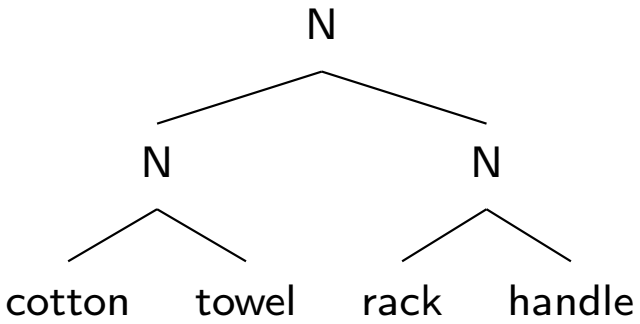




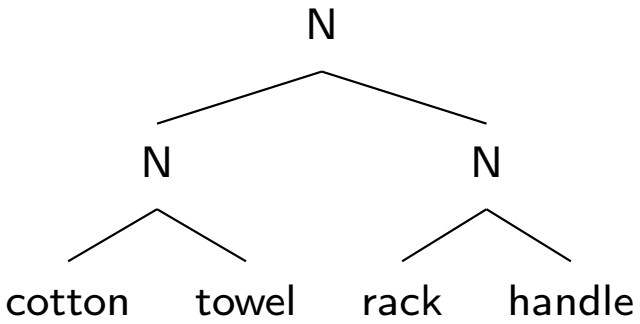
Recursive descent parsing without movement



Recursive descent parsing without movement

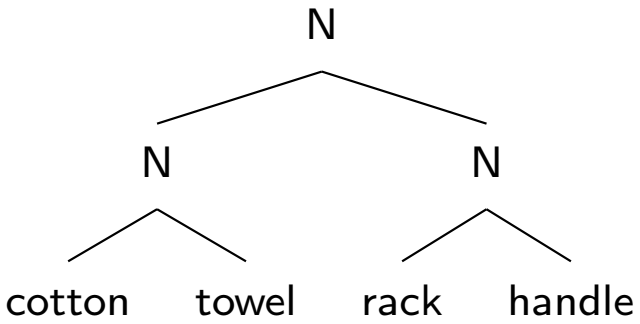


Recursive descent parsing without movement



● cotton towel rack handle

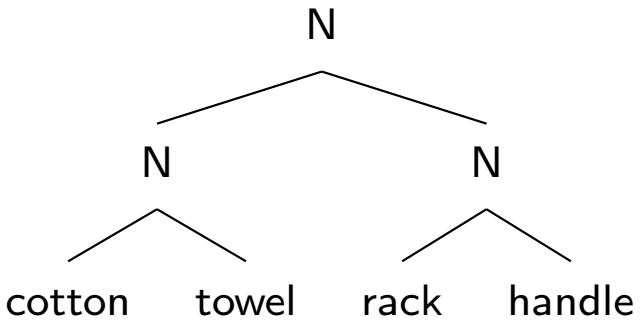
Recursive descent parsing without movement

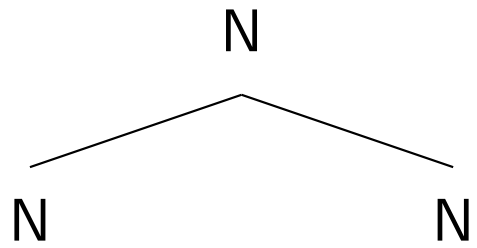


N

- cotton towel rack handle

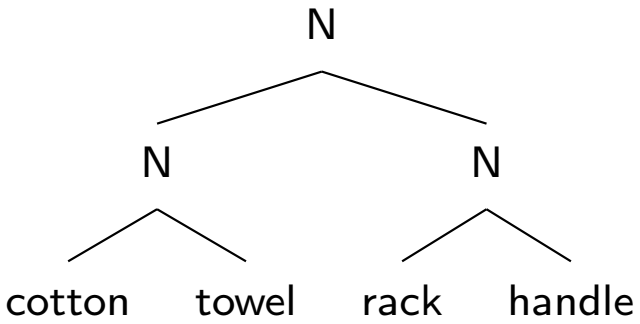
Recursive descent parsing without movement

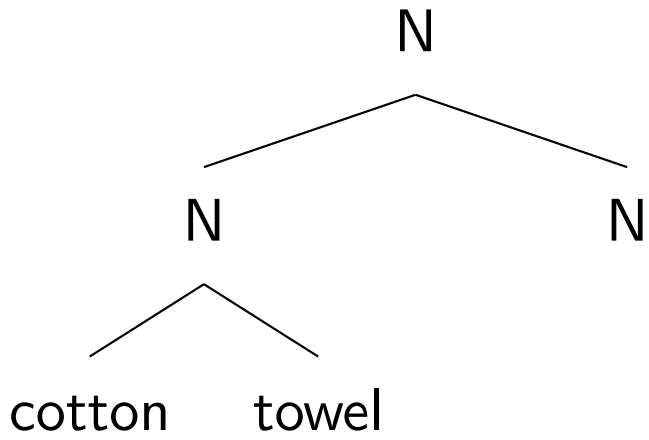




- cotton towel rack handle

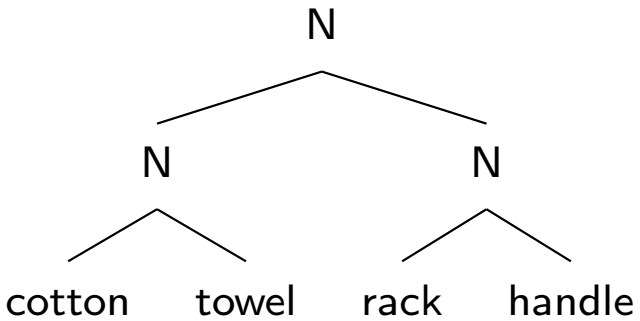
Recursive descent parsing without movement

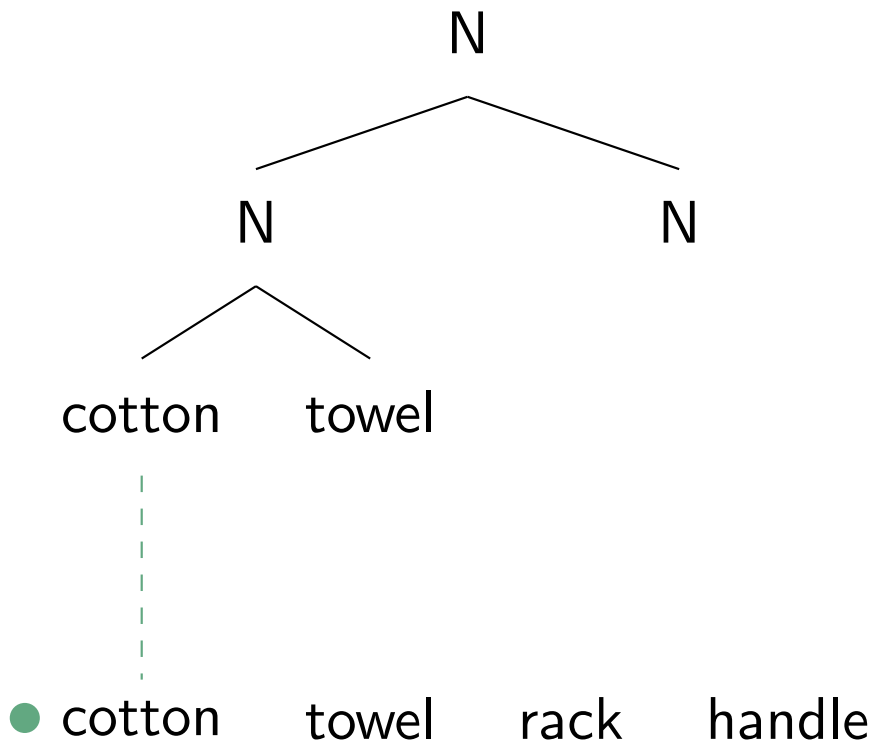




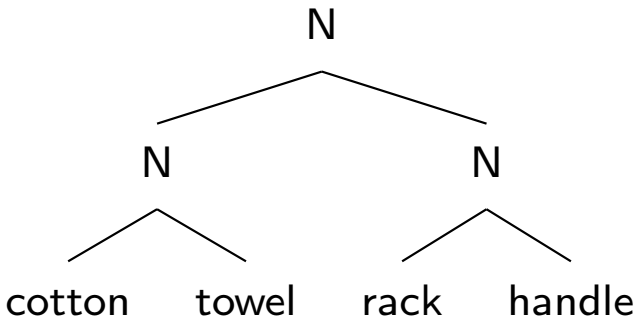
- cotton towel rack handle

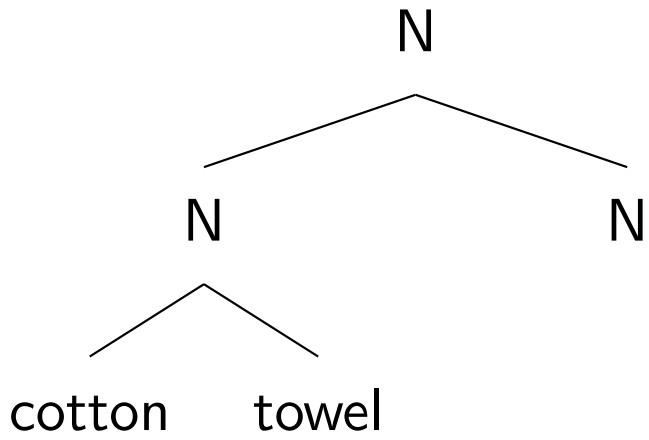
Recursive descent parsing without movement





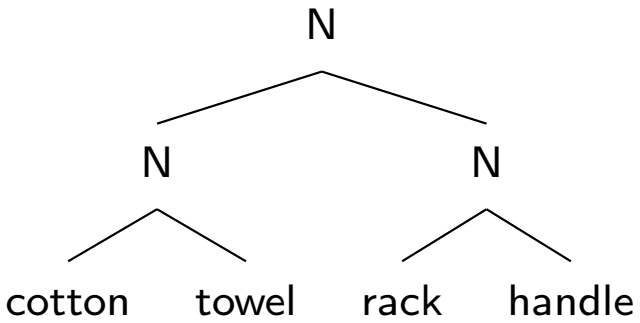
Recursive descent parsing without movement

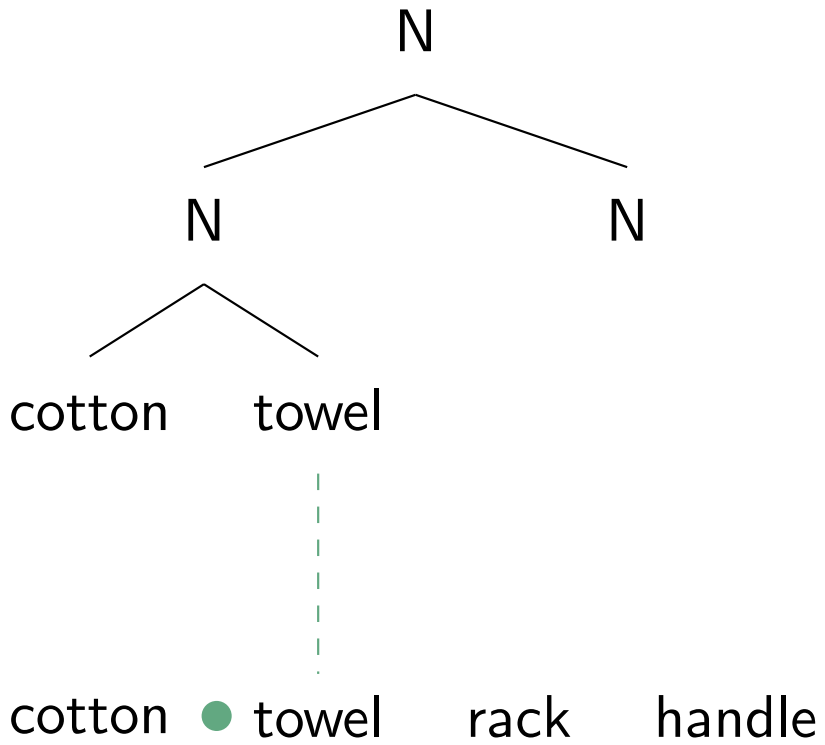




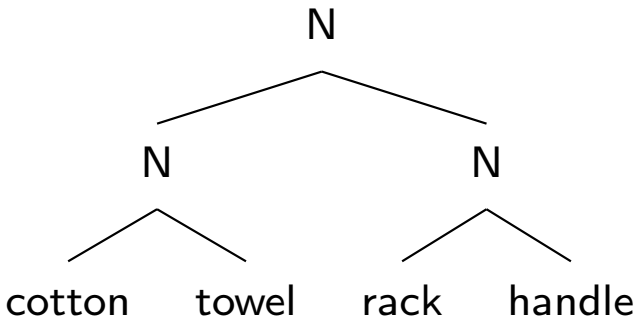
cotton ● towel rack handle

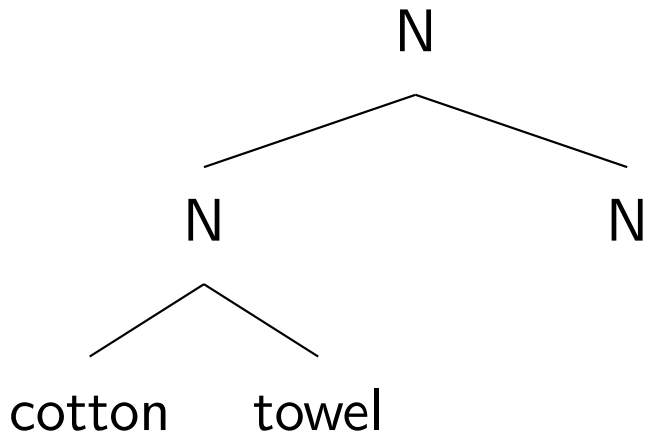
Recursive descent parsing without movement





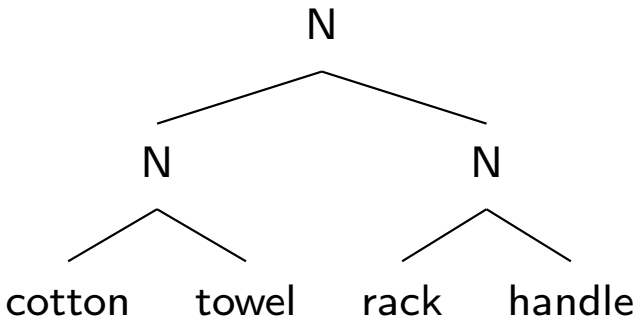
Recursive descent parsing without movement

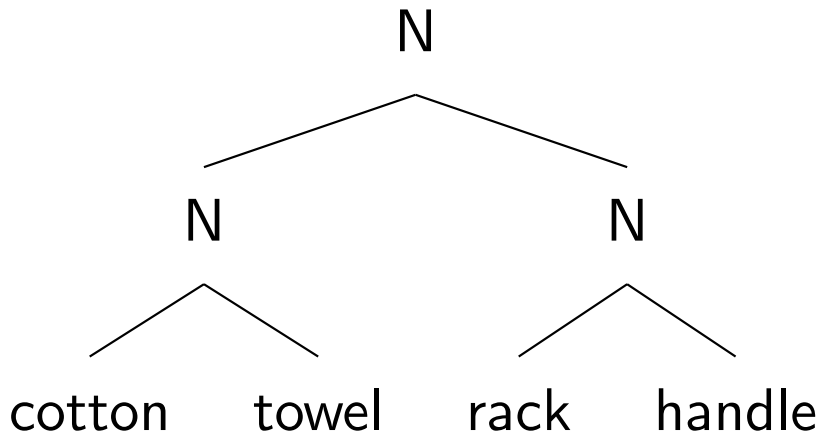




cotton towel • rack handle

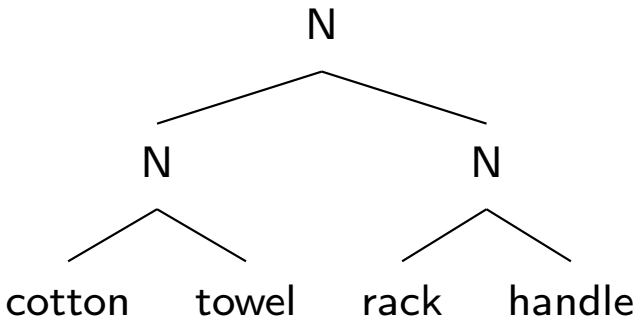
Recursive descent parsing without movement

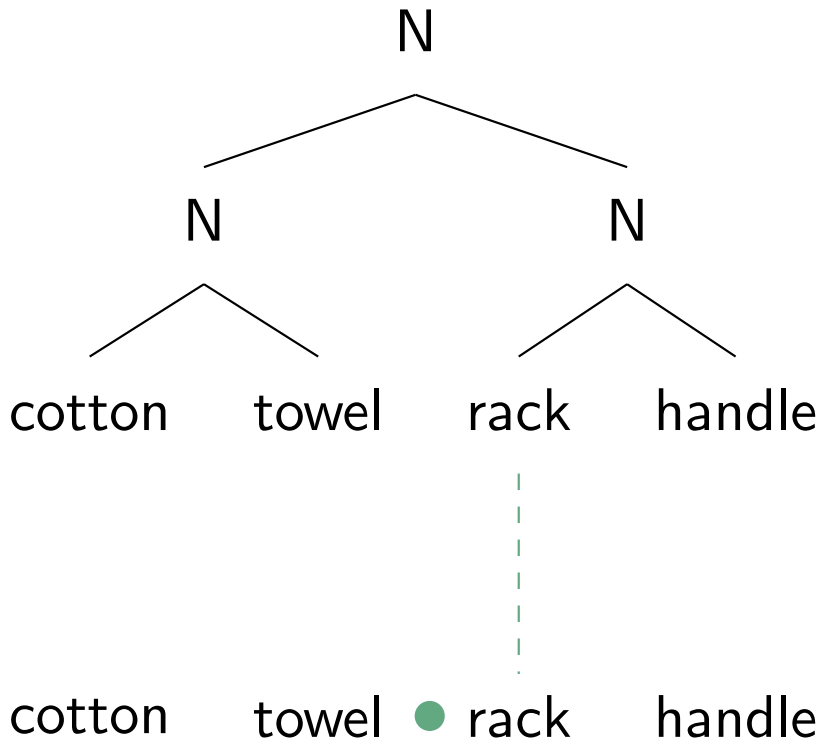




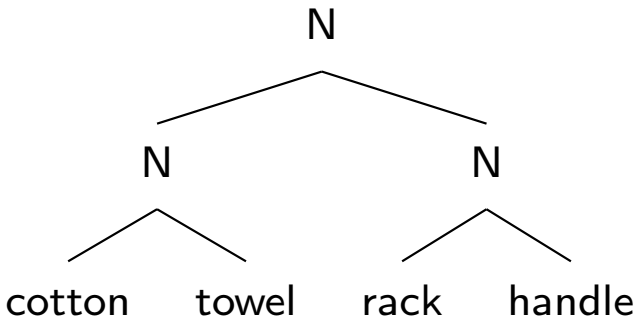
cotton towel • rack handle

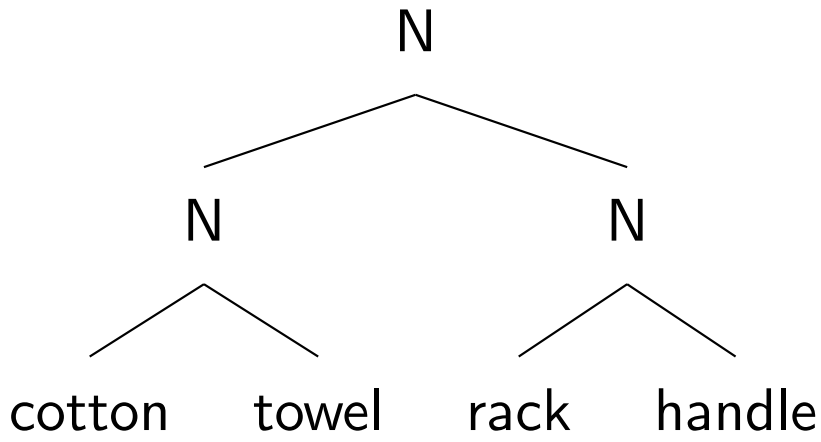
Recursive descent parsing without movement





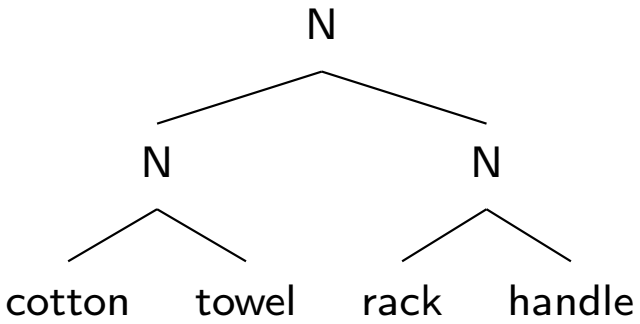
Recursive descent parsing without movement

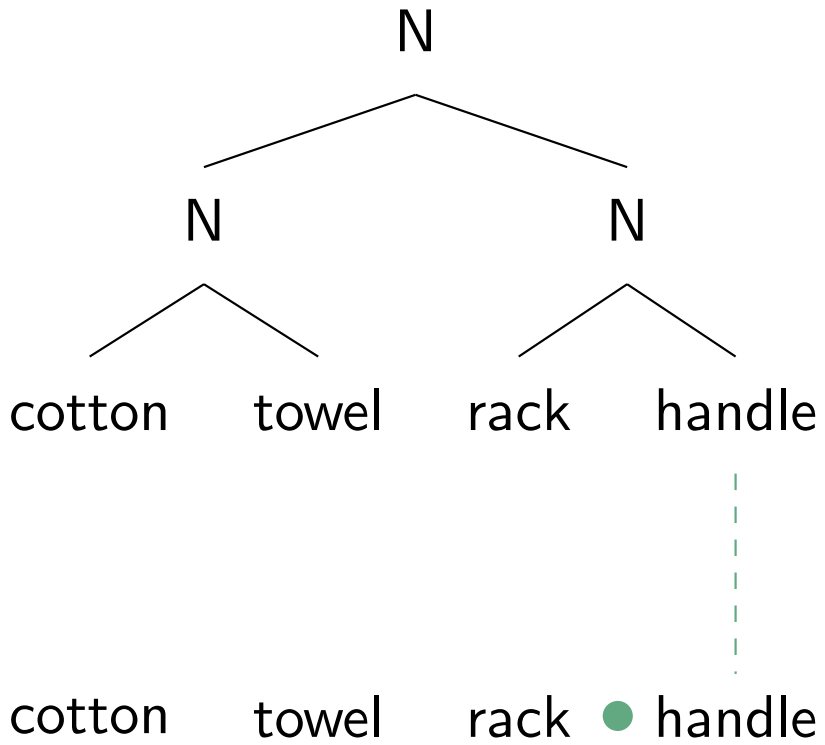




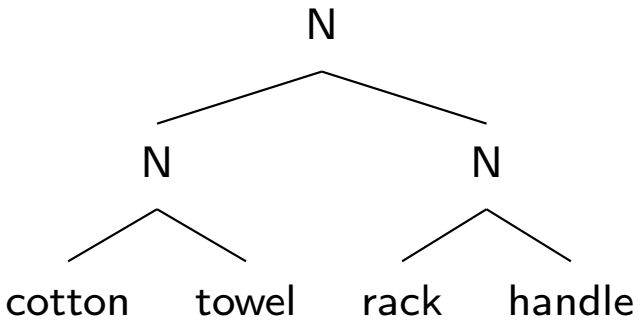
cotton towel rack ● handle

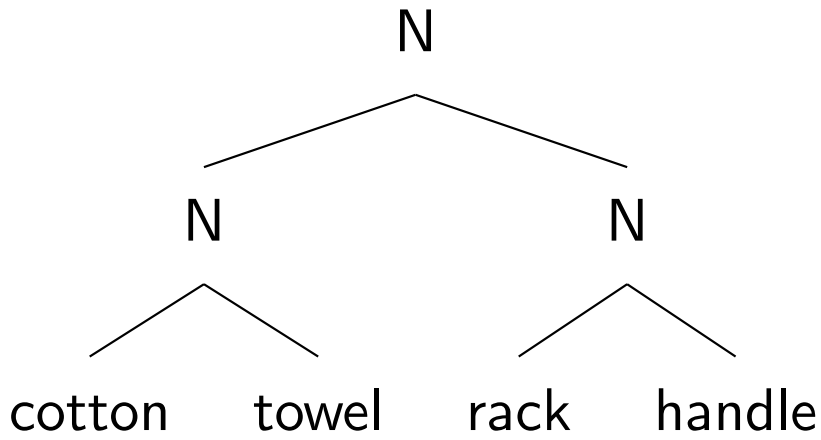
Recursive descent parsing without movement





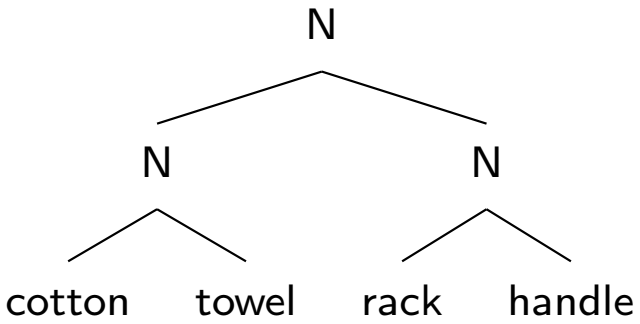
Recursive descent parsing without movement





cotton towel rack handle

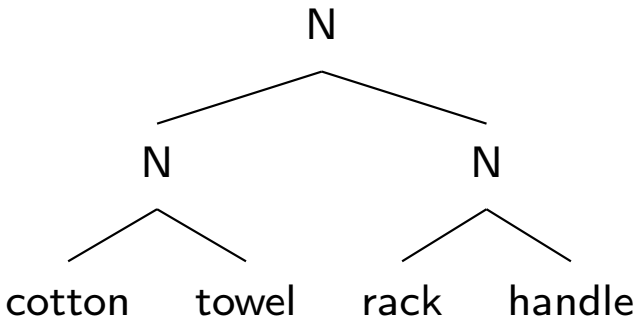
Recursive descent parsing without movement



N

- cotton towel rack handle

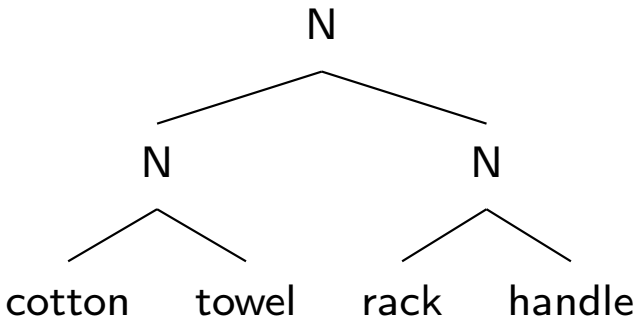
Recursive descent parsing without movement

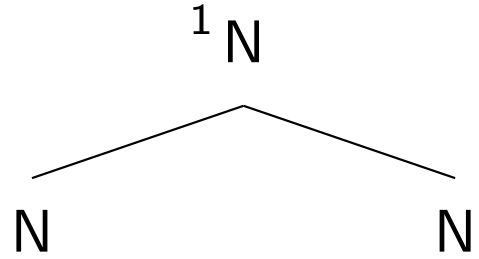


$^{1}\,\mathrm{N}$

● cotton towel rack handle

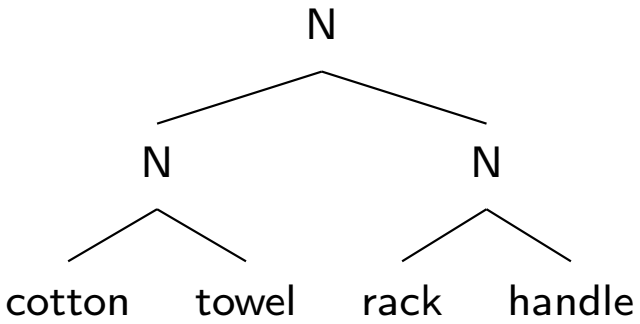
Recursive descent parsing without movement

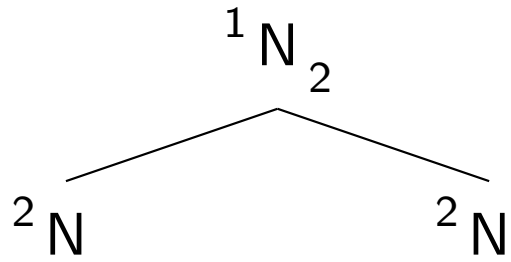




- cotton towel rack handle

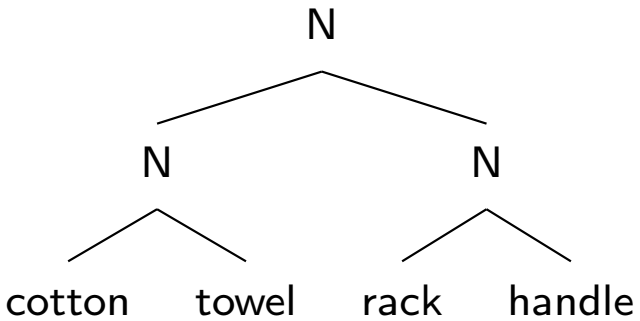
Recursive descent parsing without movement

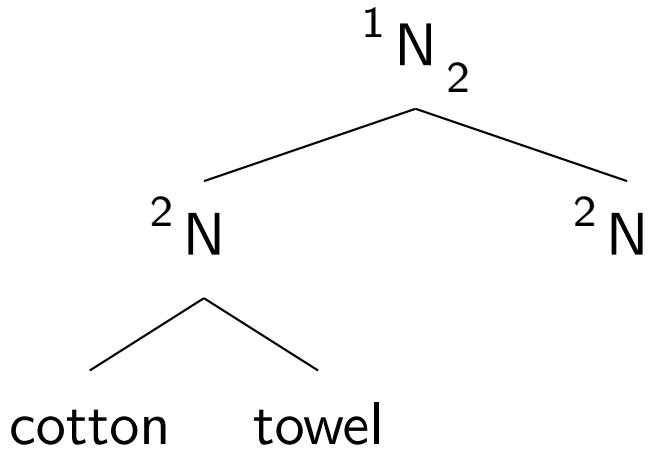




- cotton towel rack handle

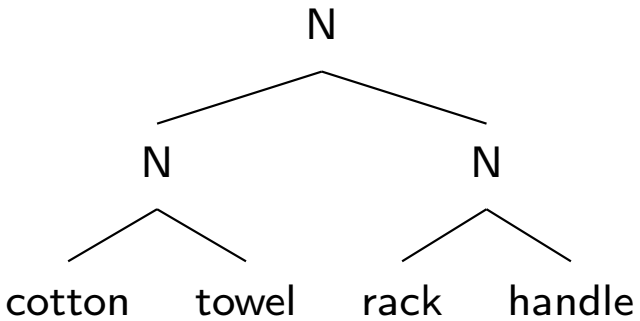
Recursive descent parsing without movement

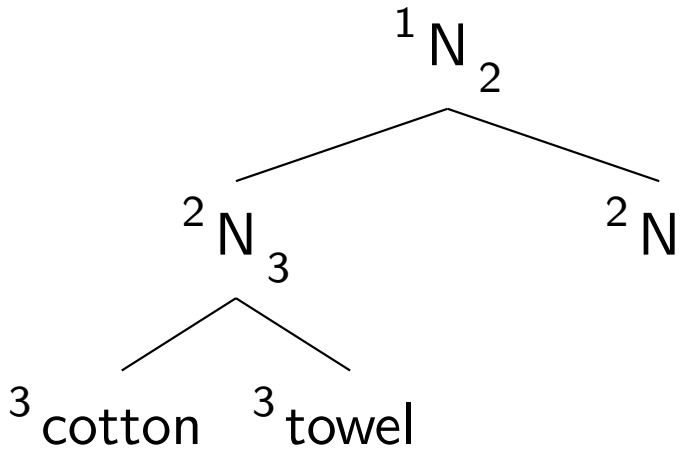




- cotton towel rack handle

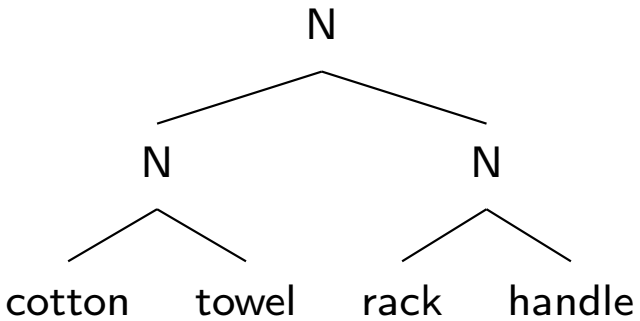
Recursive descent parsing without movement

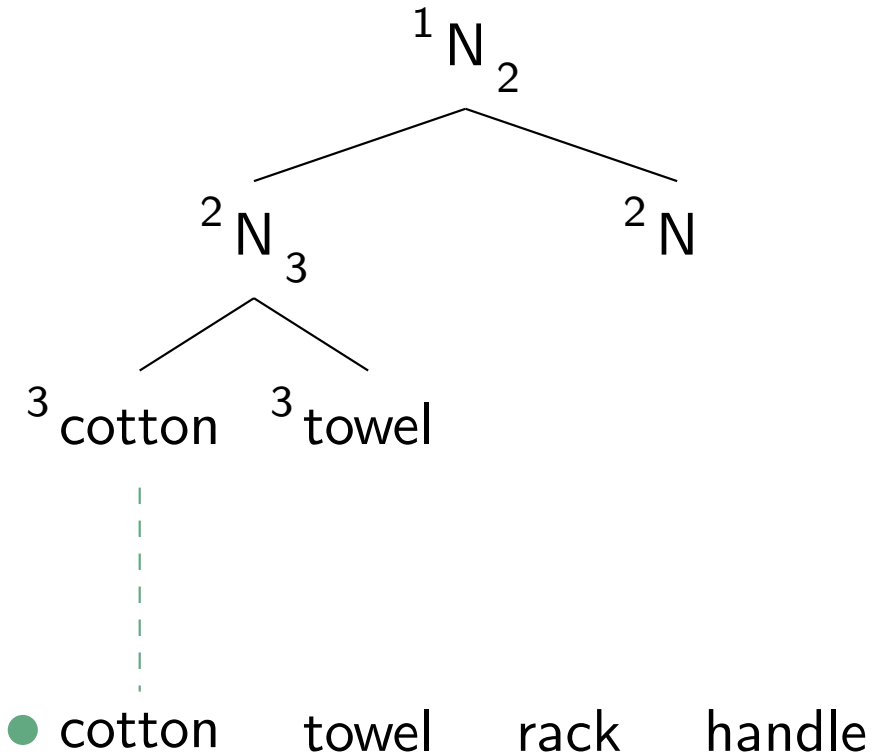




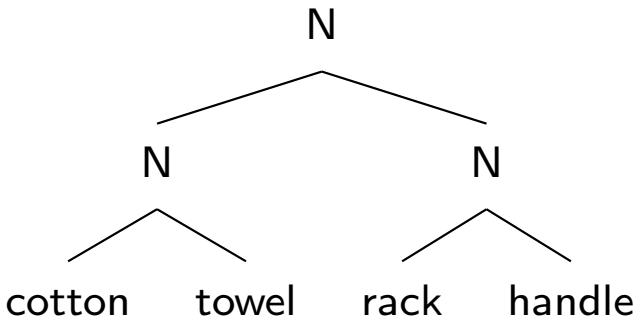
● cotton towel rack handle

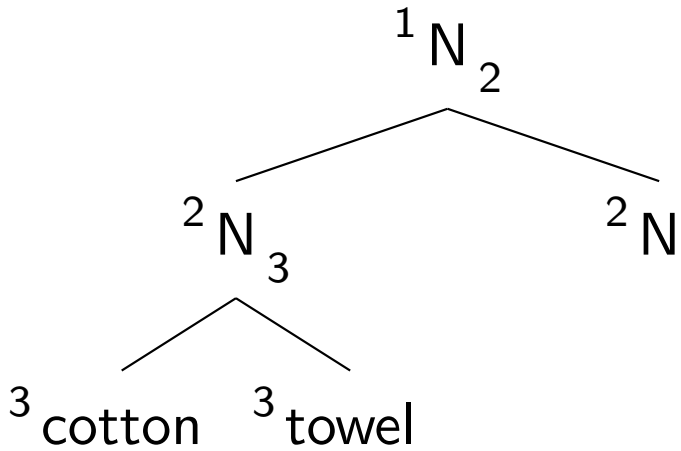
Recursive descent parsing without movement





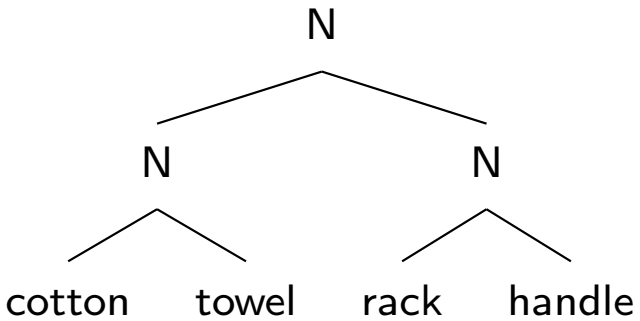
Recursive descent parsing without movement

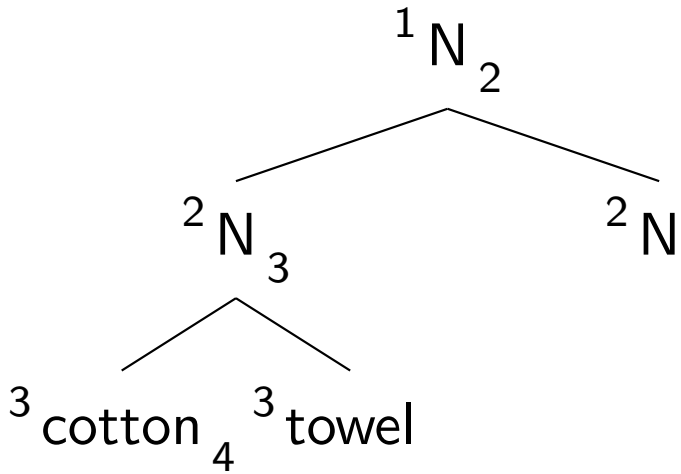




cotton ● towel rack handle

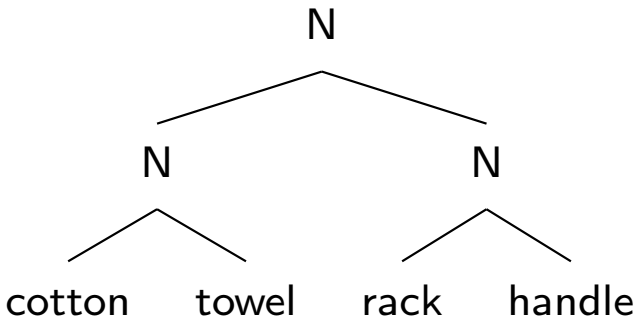
Recursive descent parsing without movement

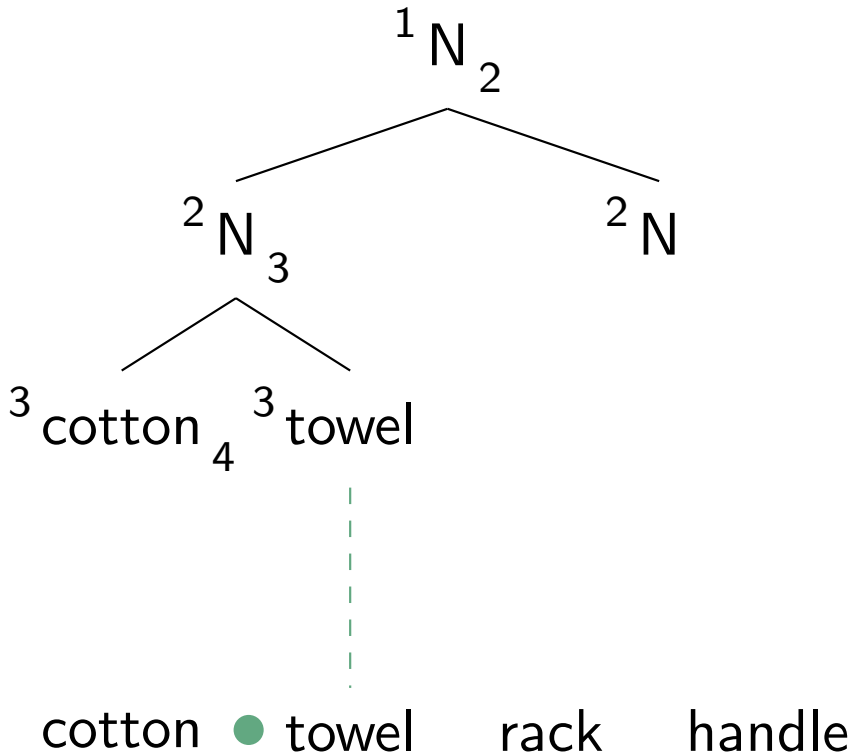




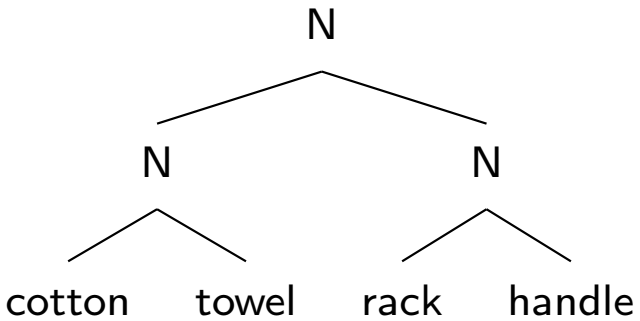
cotton ● towel rack handle

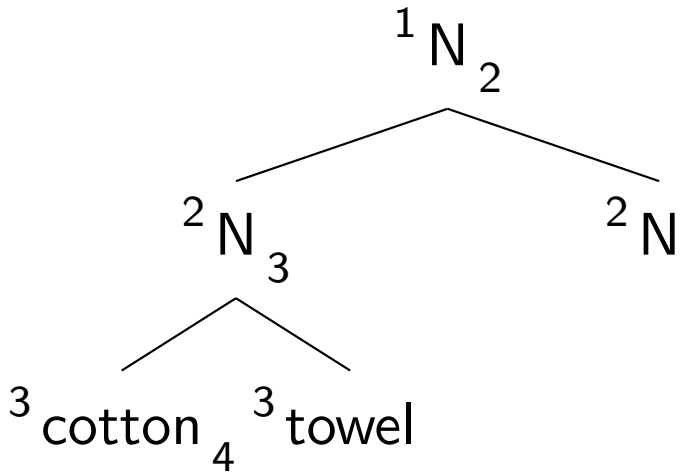
Recursive descent parsing without movement





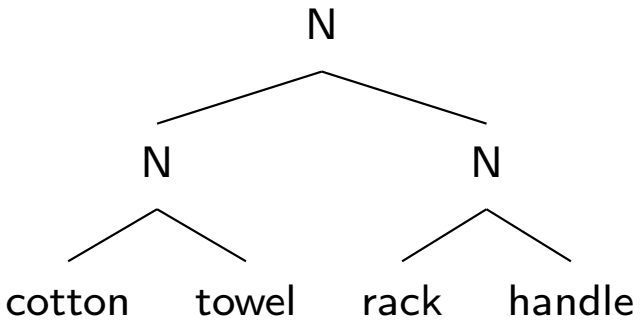
Recursive descent parsing without movement

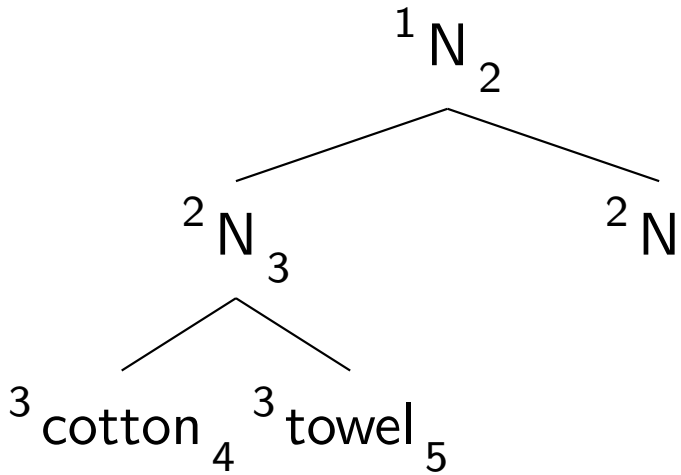




cotton towel ● rack handle

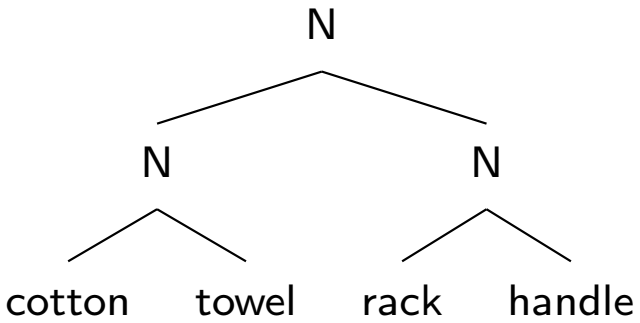
Recursive descent parsing without movement

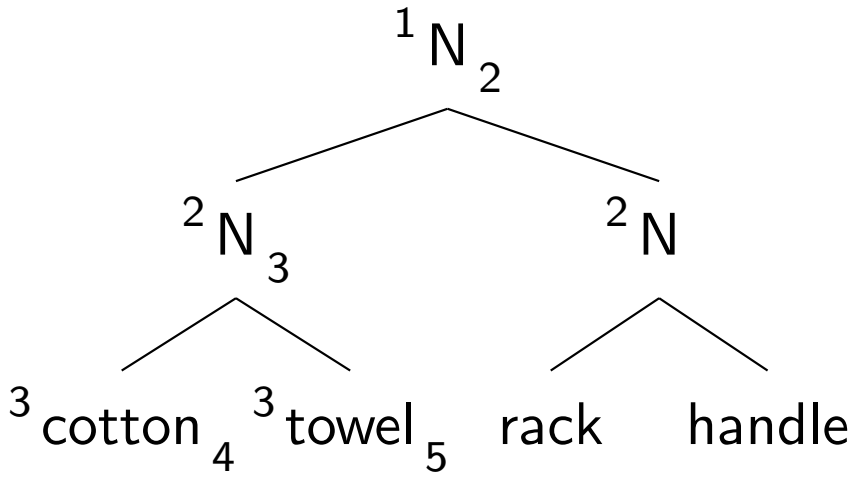




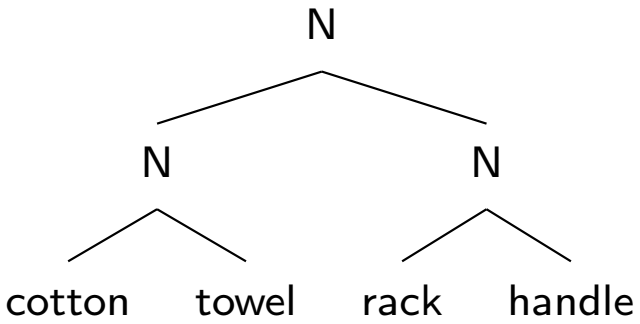
cotton towel rack handle

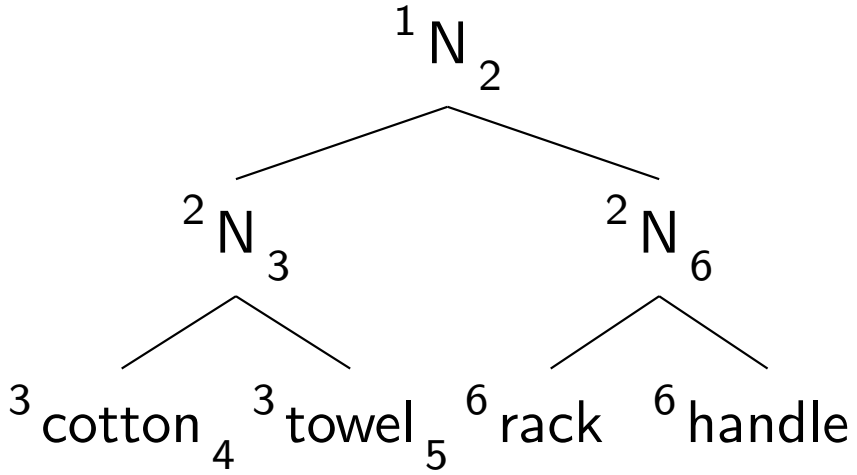
Recursive descent parsing without movement



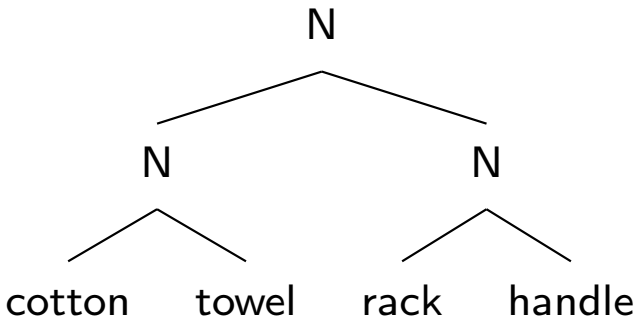


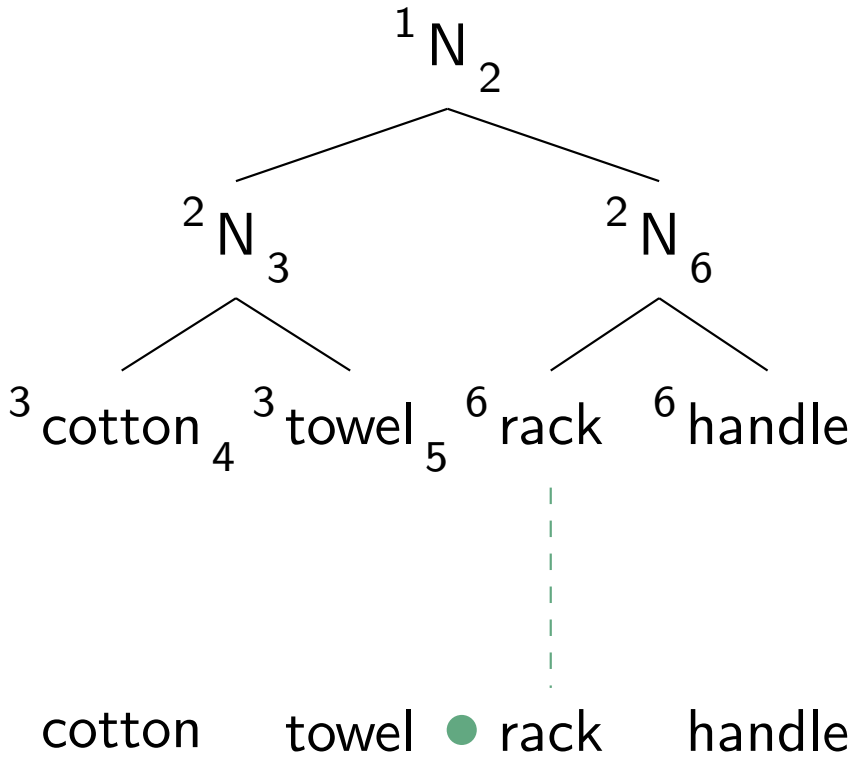
Recursive descent parsing without movement



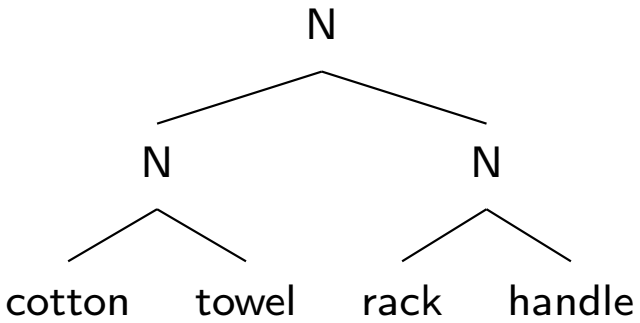


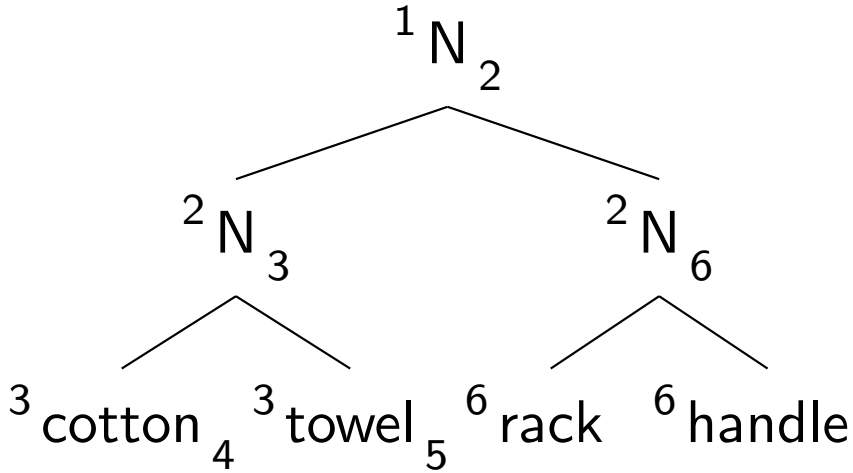
Recursive descent parsing without movement





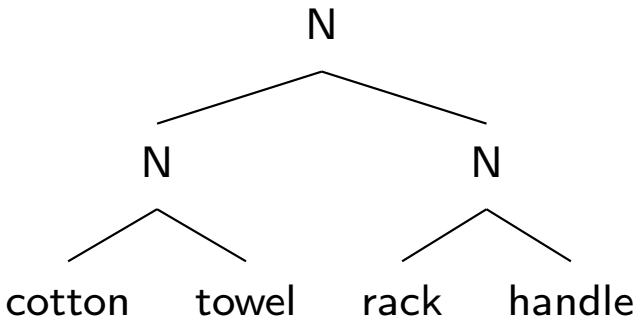
Recursive descent parsing without movement

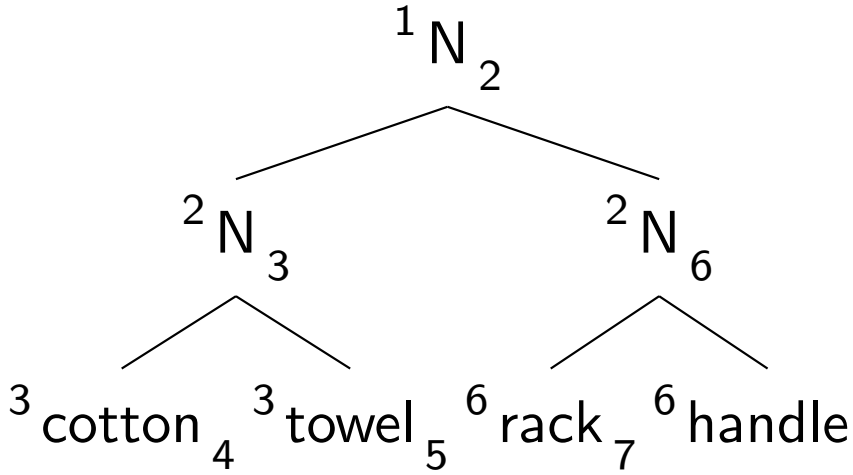




cotton towel rack handle

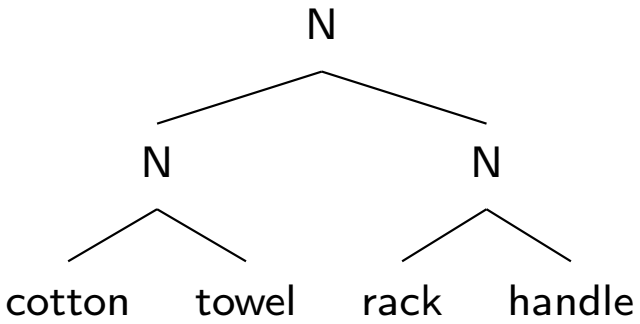
Recursive descent parsing without movement

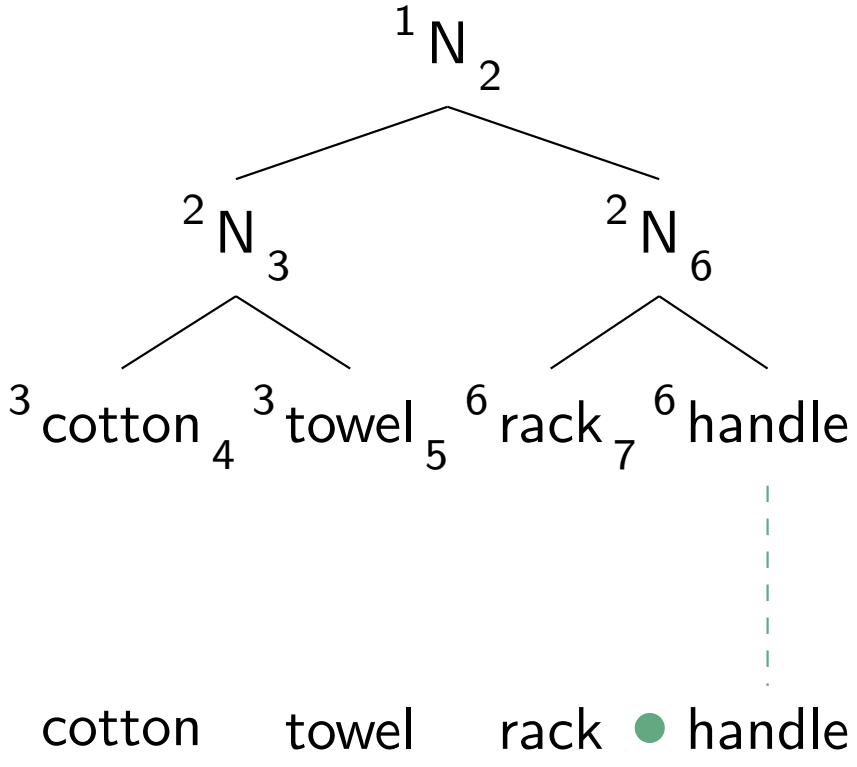




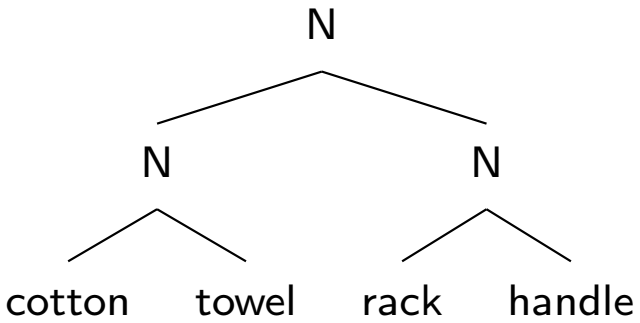
cotton towel rack handle

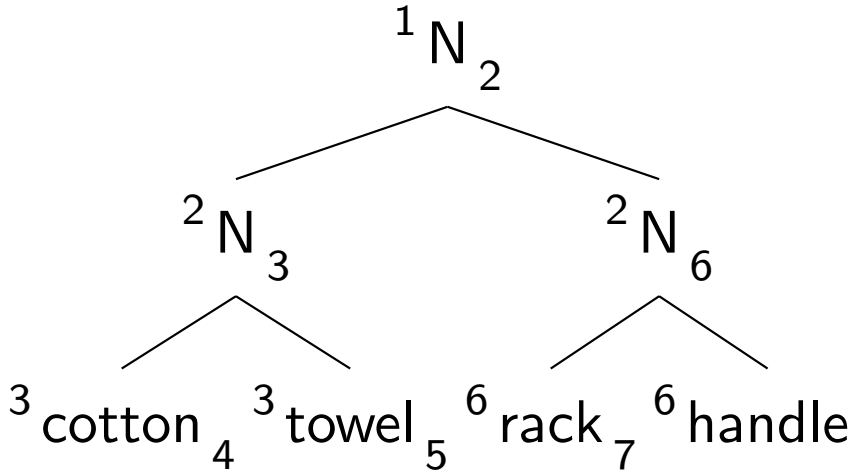
Recursive descent parsing without movement





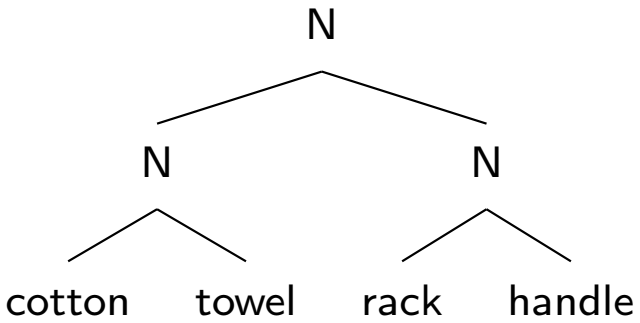
Recursive descent parsing without movement

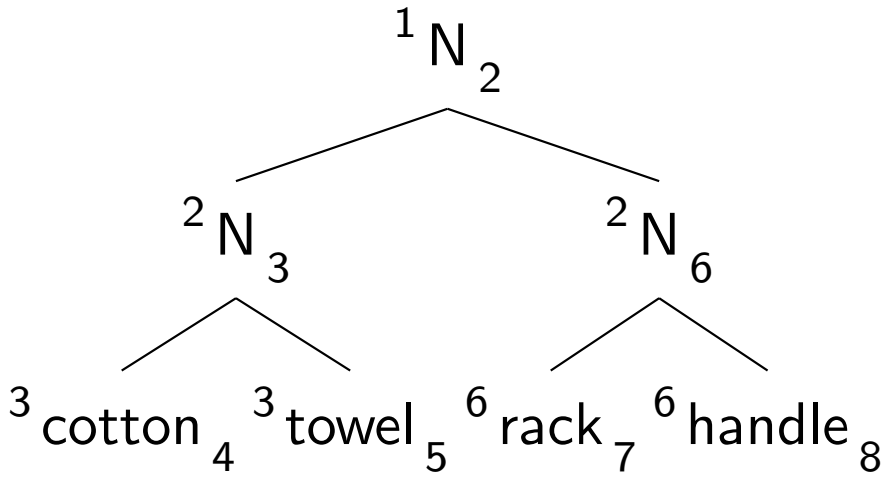




cotton towel rack handle

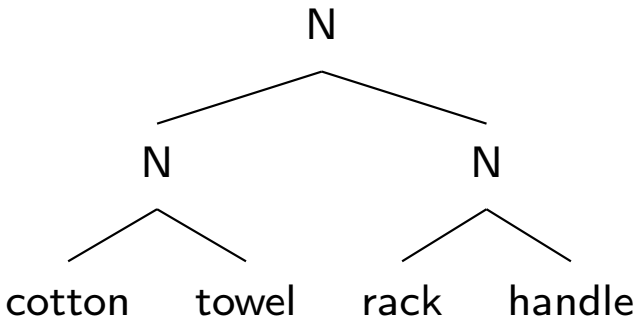
Recursive descent parsing without movement

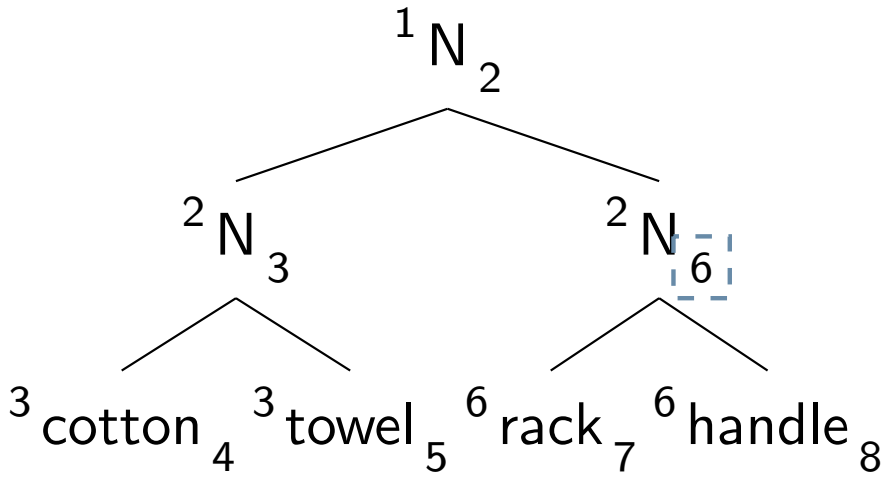




cotton towel rack handle

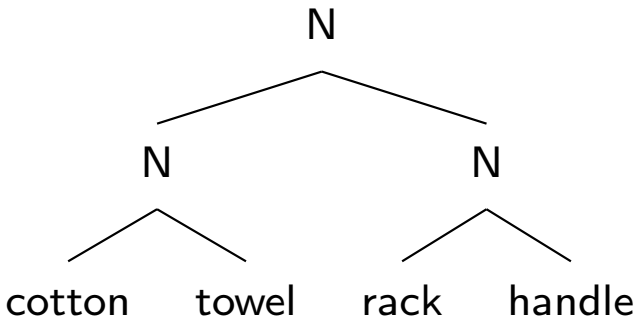
Recursive descent parsing without movement

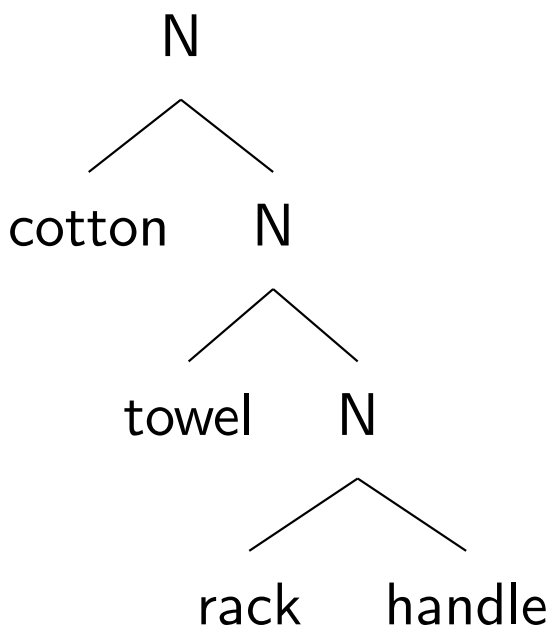


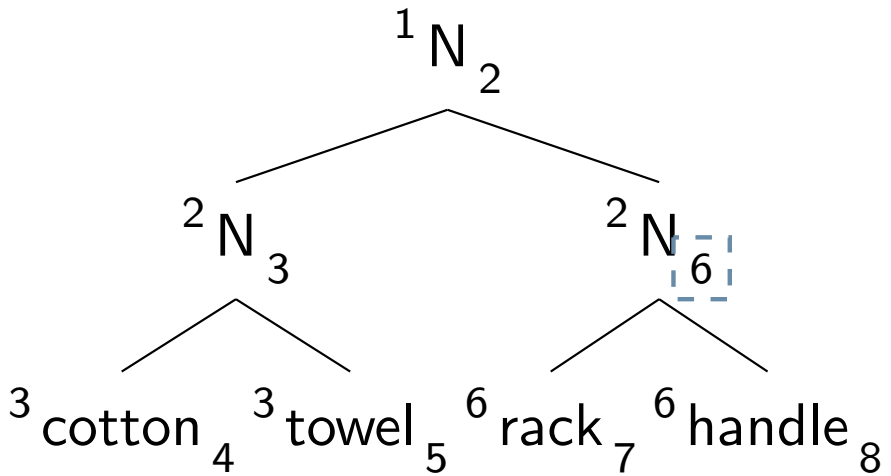


cotton towel rack handle

Recursive descent parsing without movement

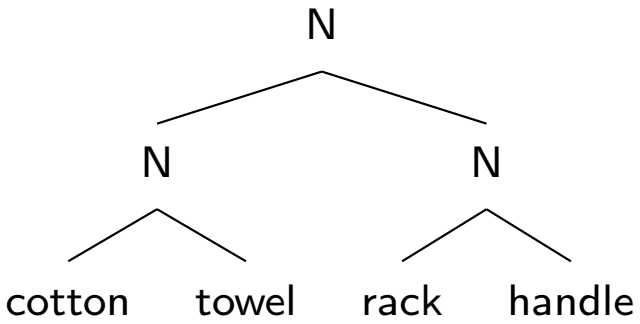


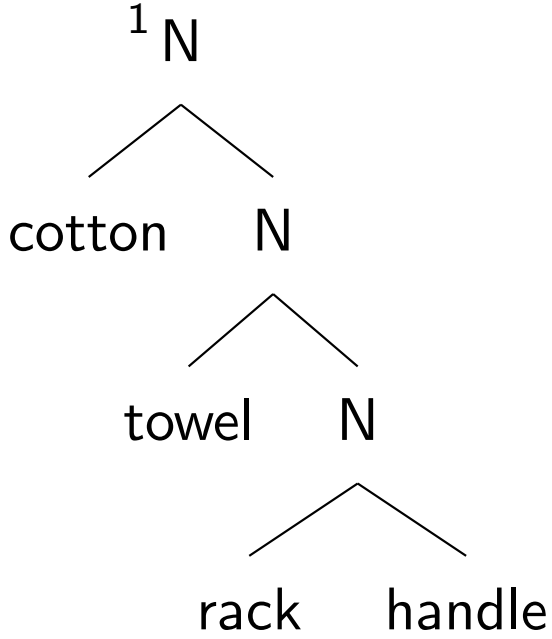


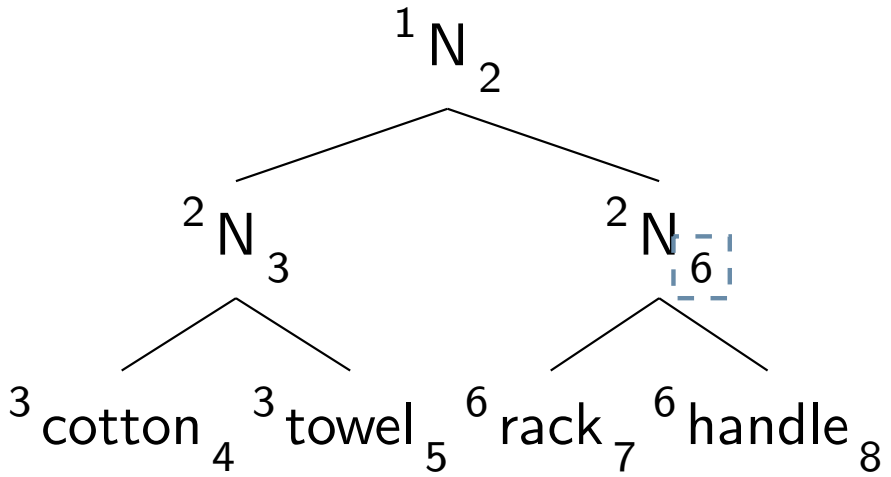


cotton towel rack handle

Recursive descent parsing without movement

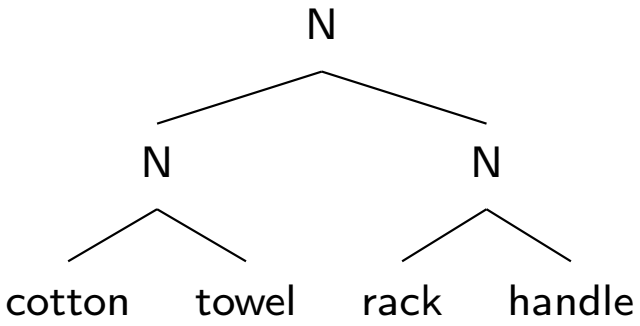


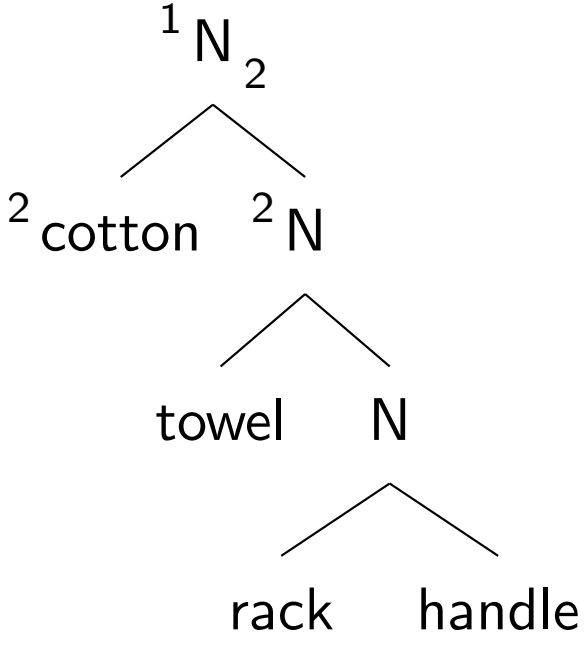


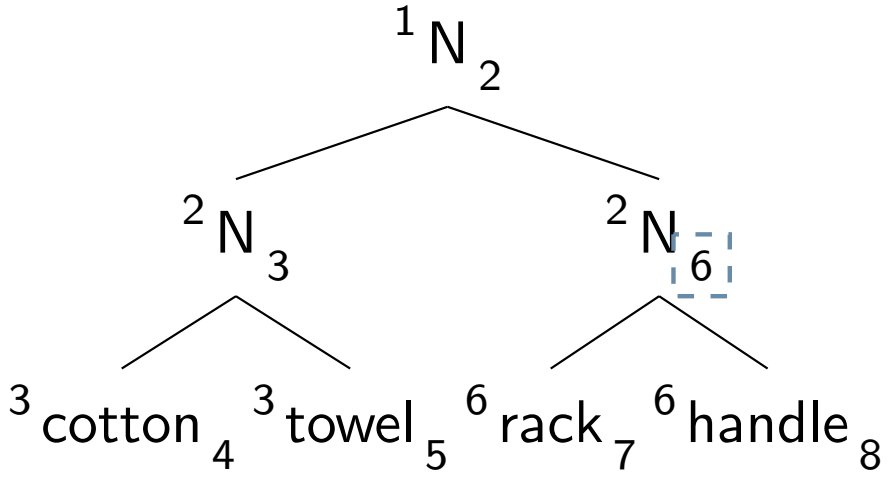


cotton towel rack handle

Recursive descent parsing without movement

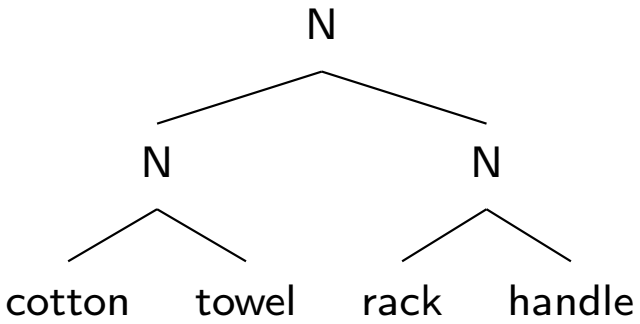


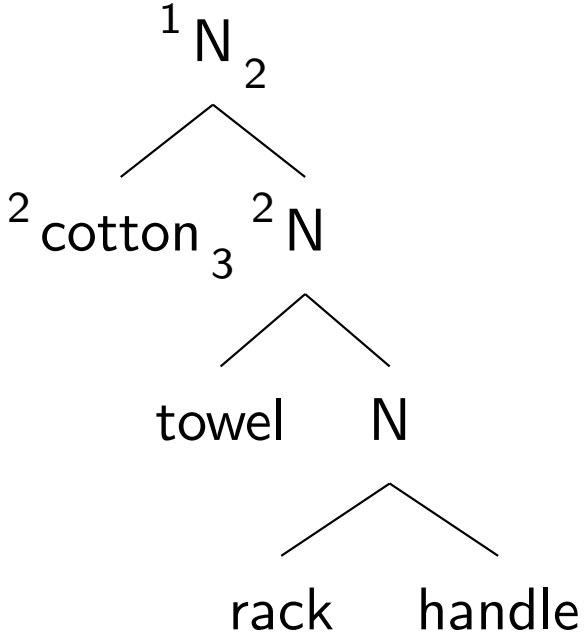


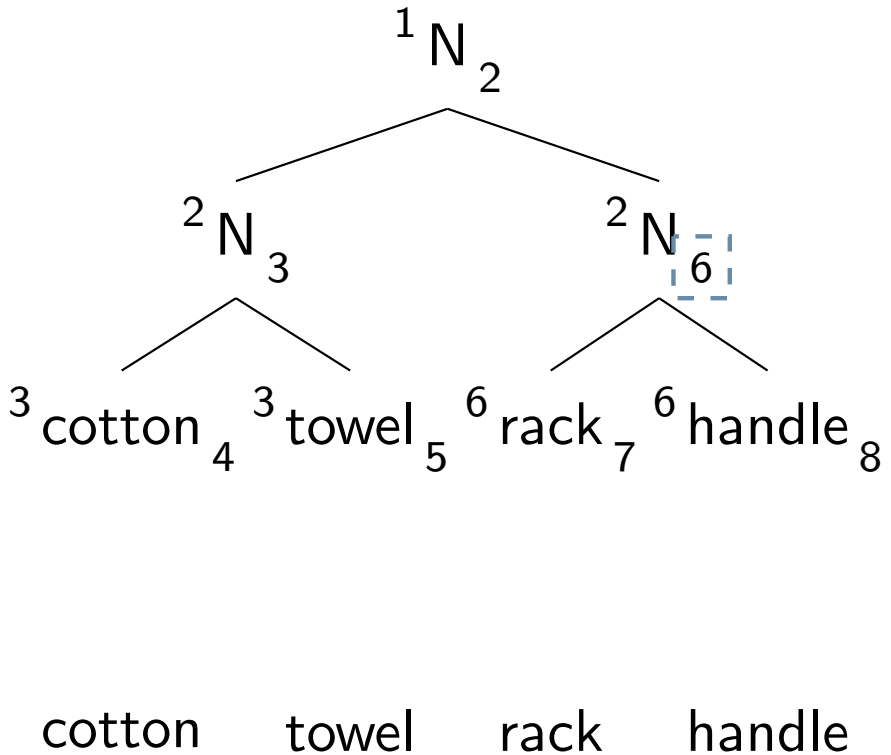


cotton towel rack handle

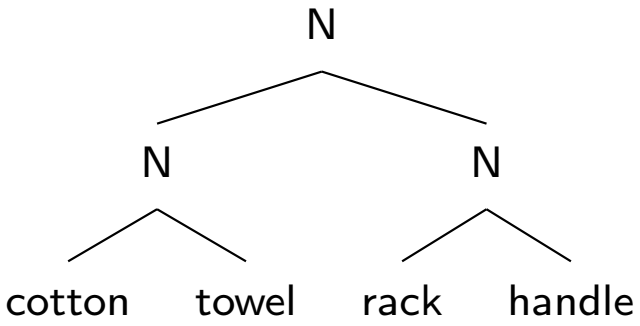
Recursive descent parsing without movement

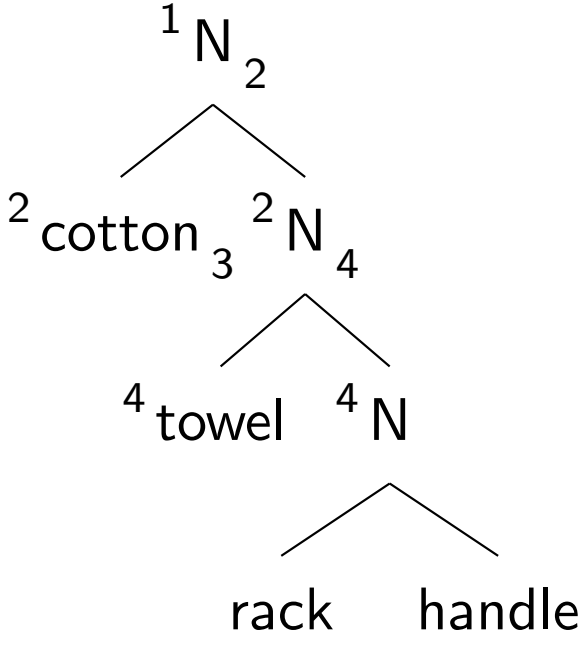


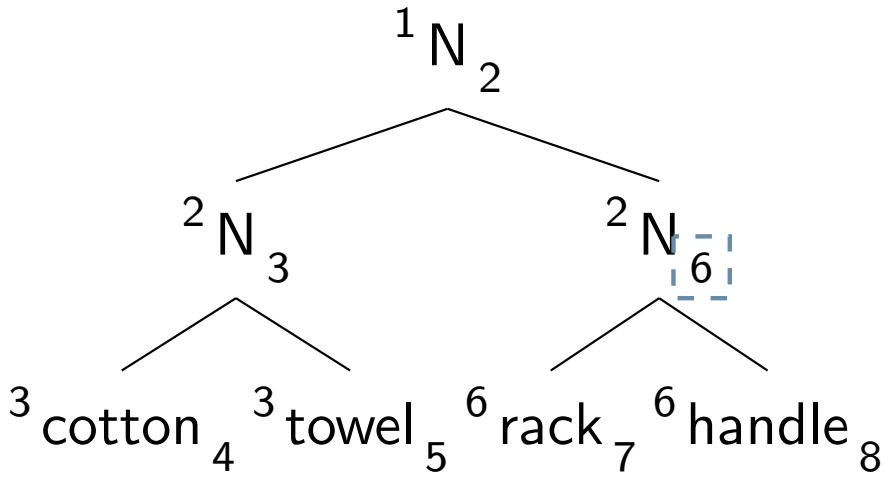




Recursive descent parsing without movement

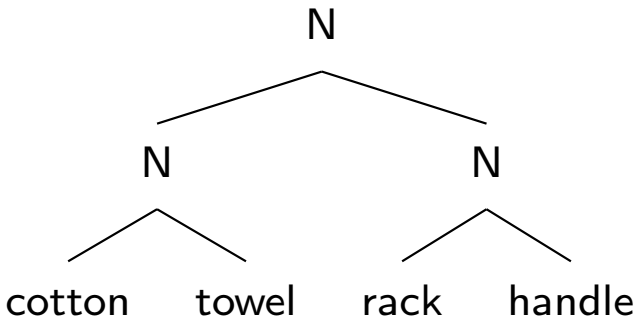


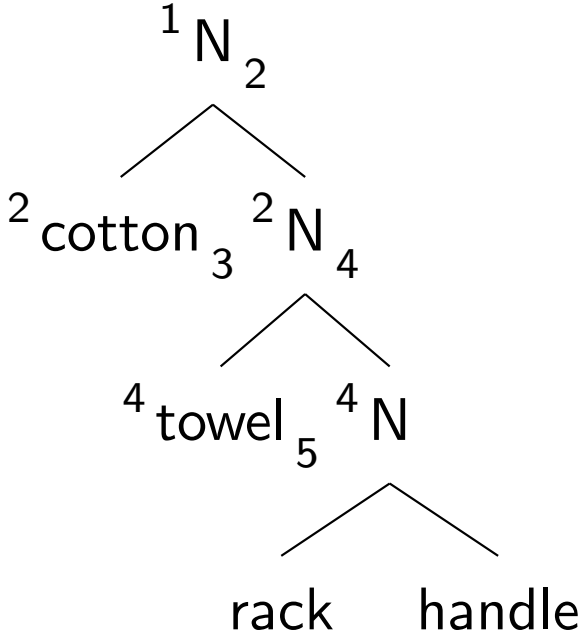


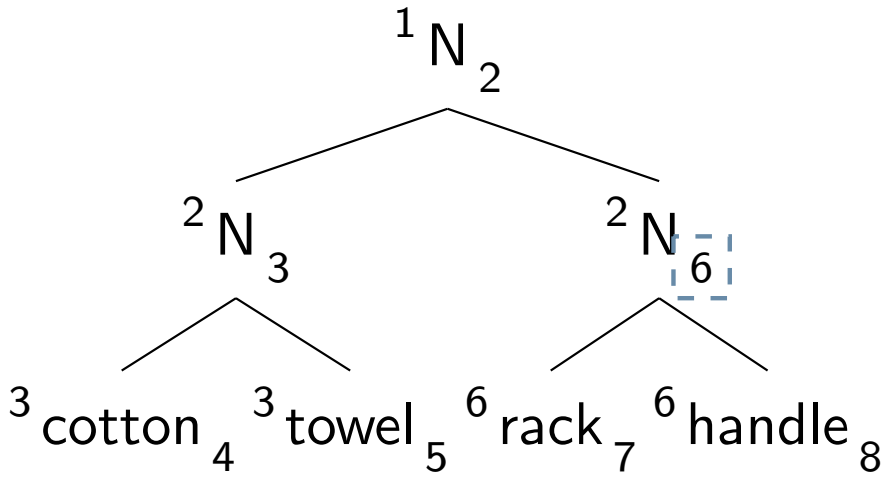


cotton towel rack handle

Recursive descent parsing without movement

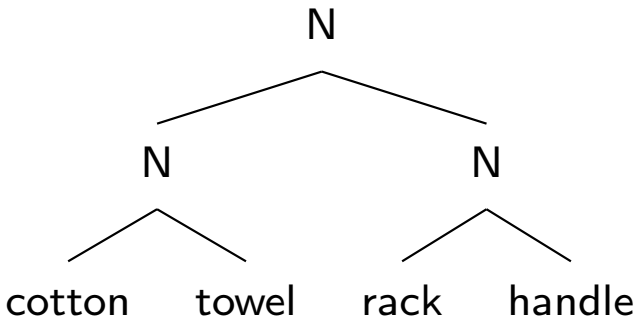


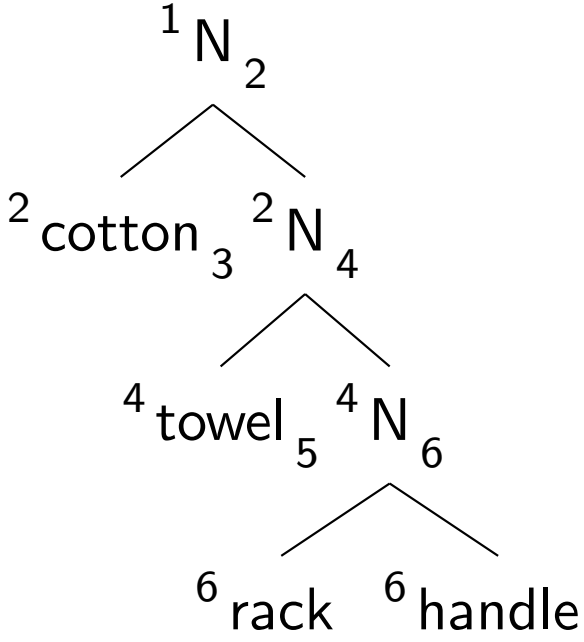


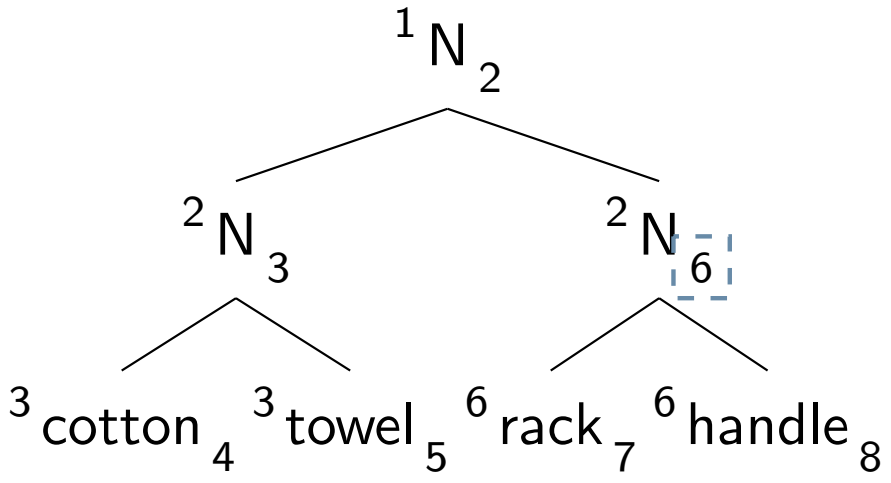


cotton towel rack handle

Recursive descent parsing without movement

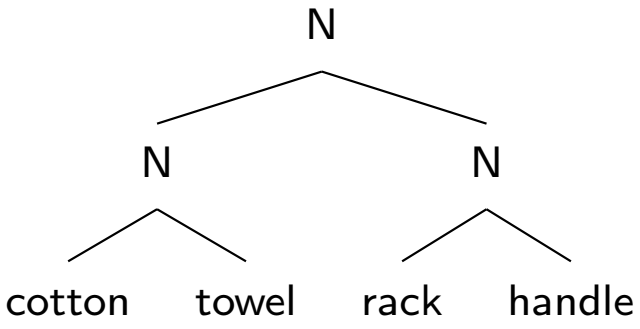


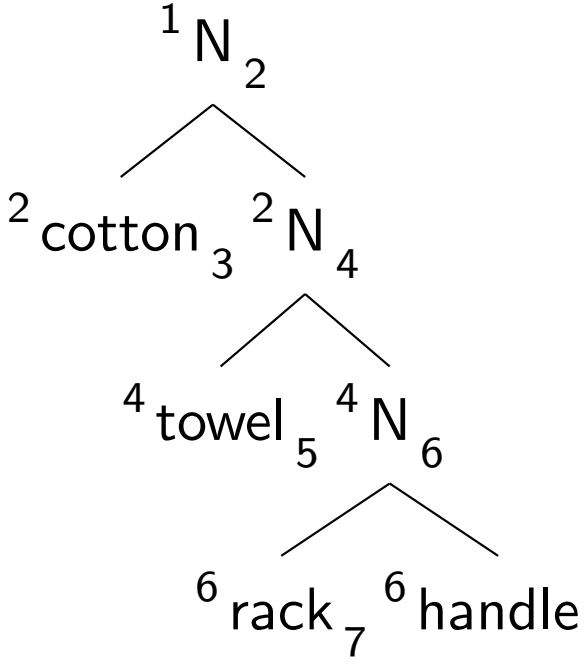


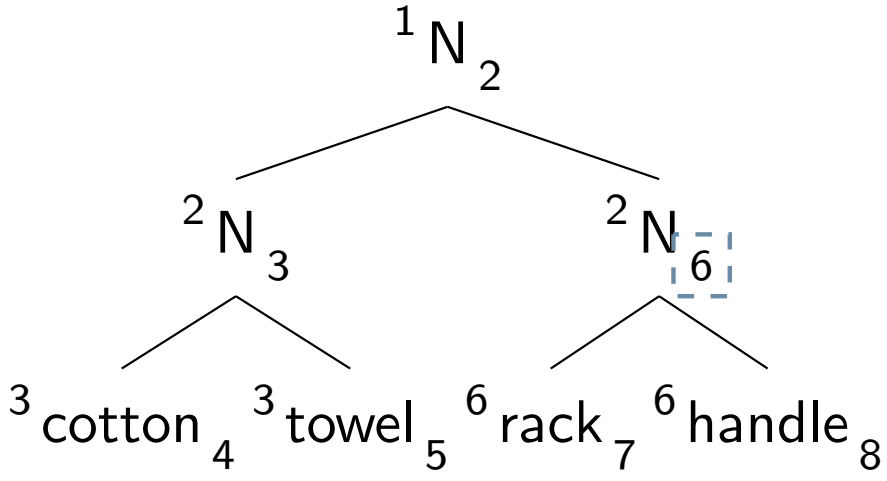


cotton towel rack handle

Recursive descent parsing without movement

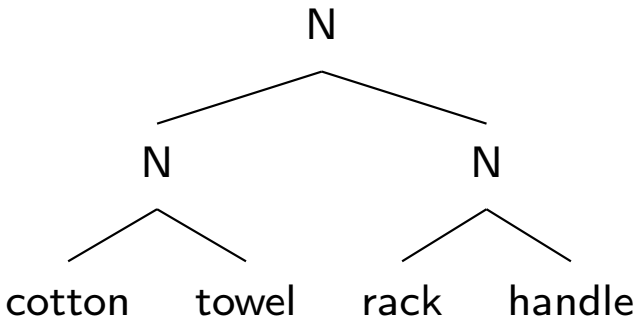


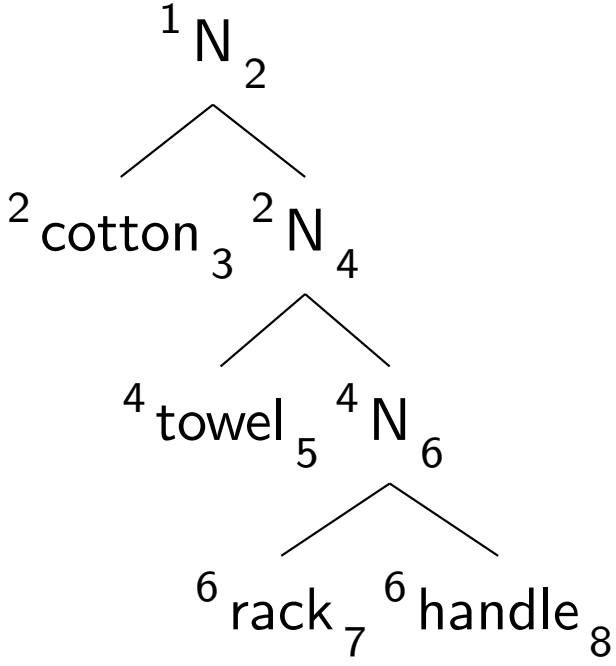


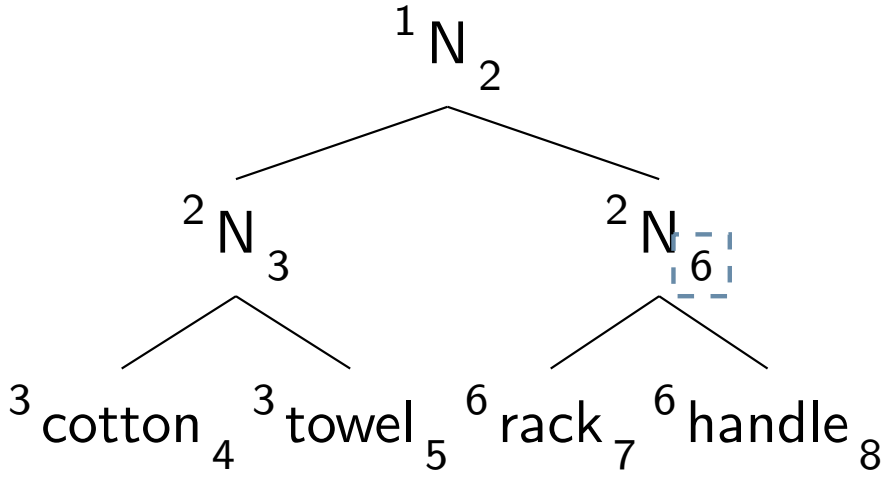


cotton towel rack handle

Recursive descent parsing without movement

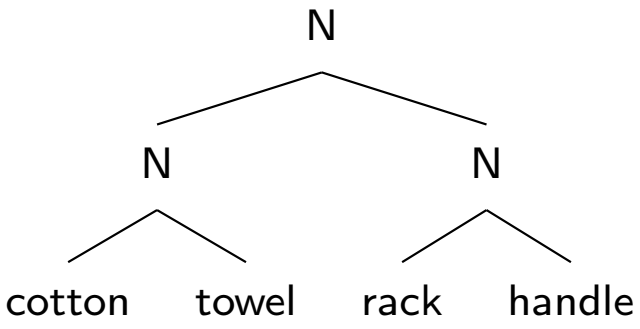


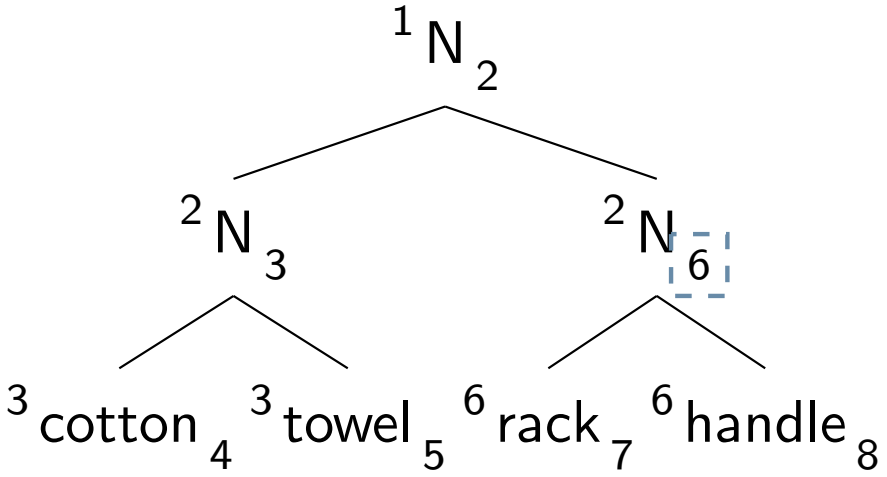


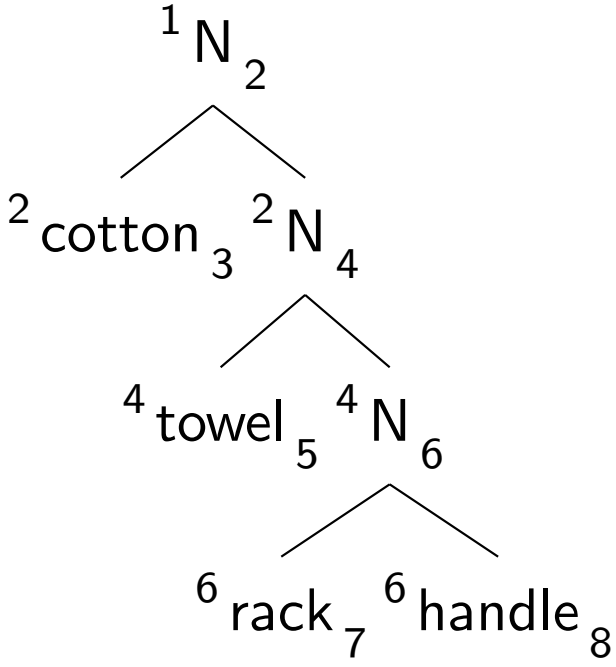


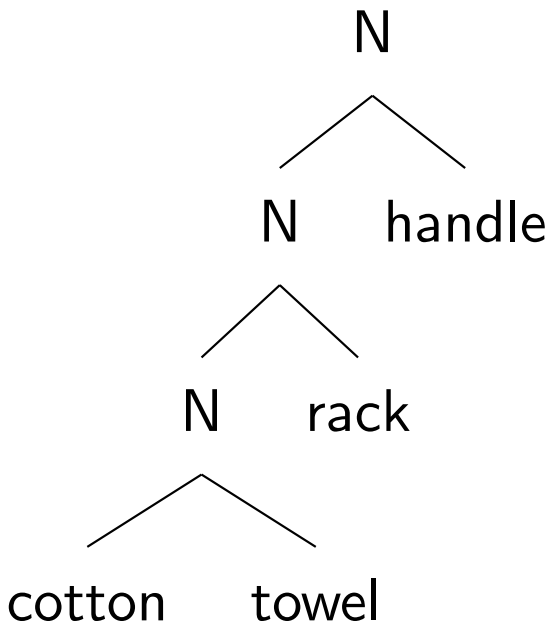
cotton towel rack handle

Recursive descent parsing without movement

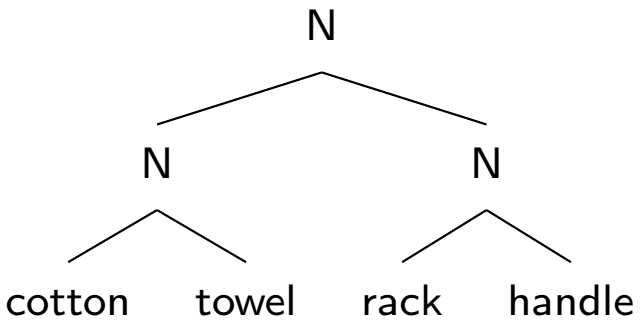


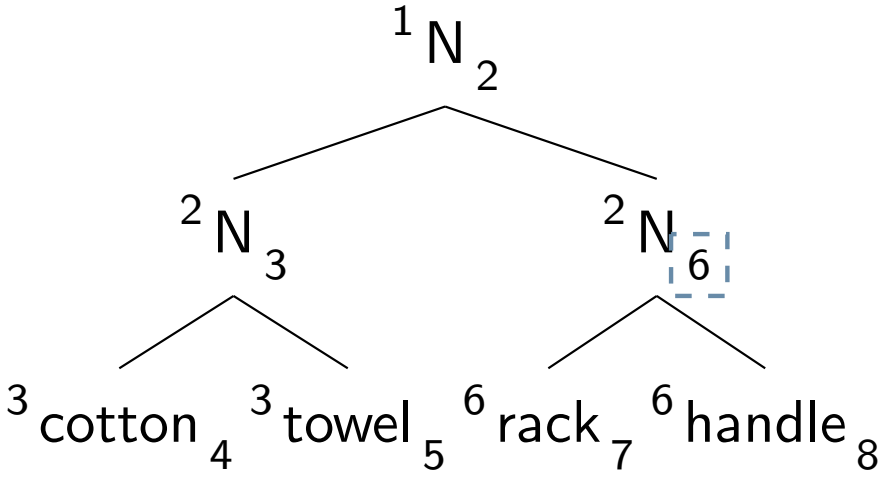


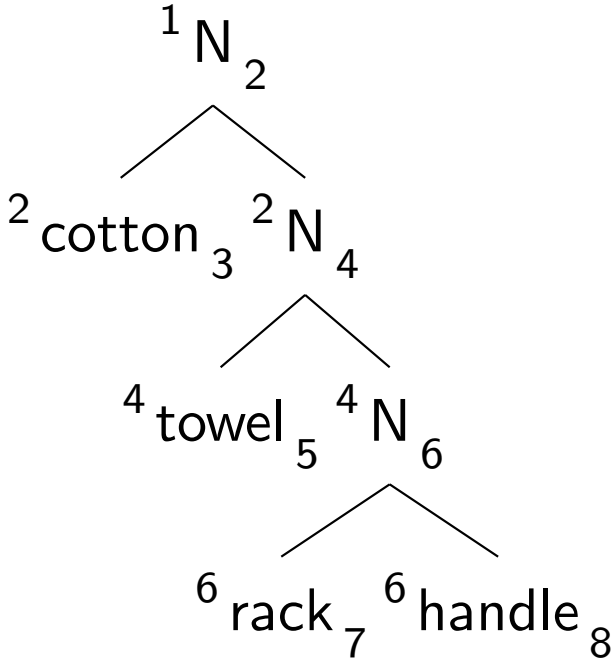


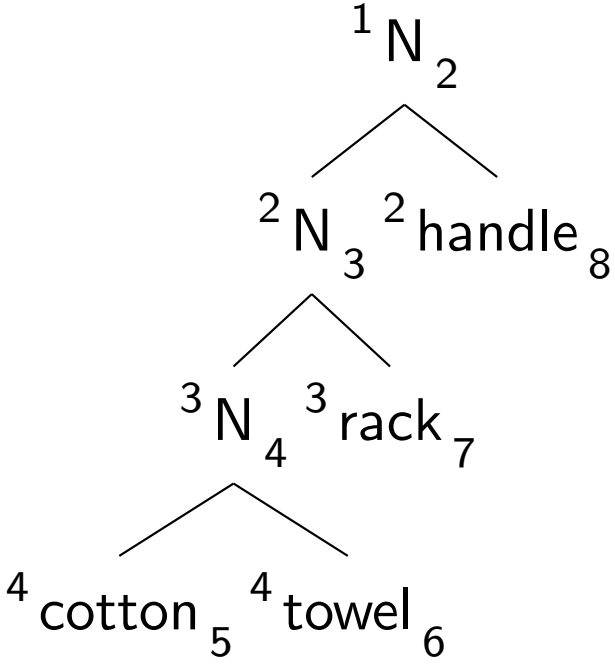


Recursive descent parsing without movement

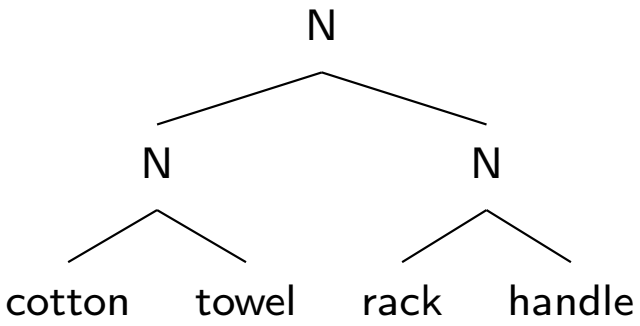


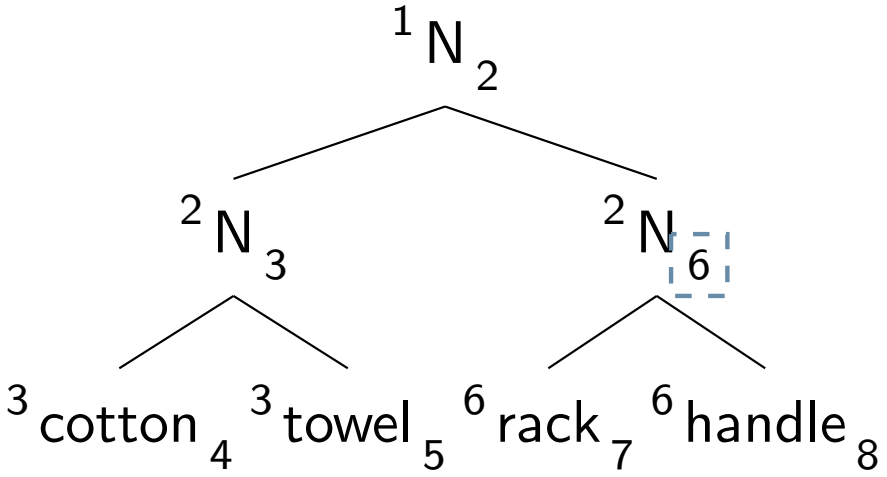


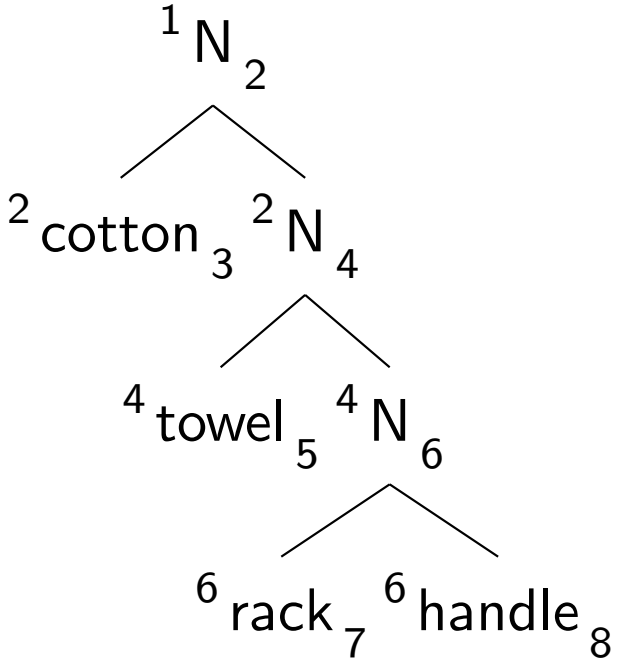


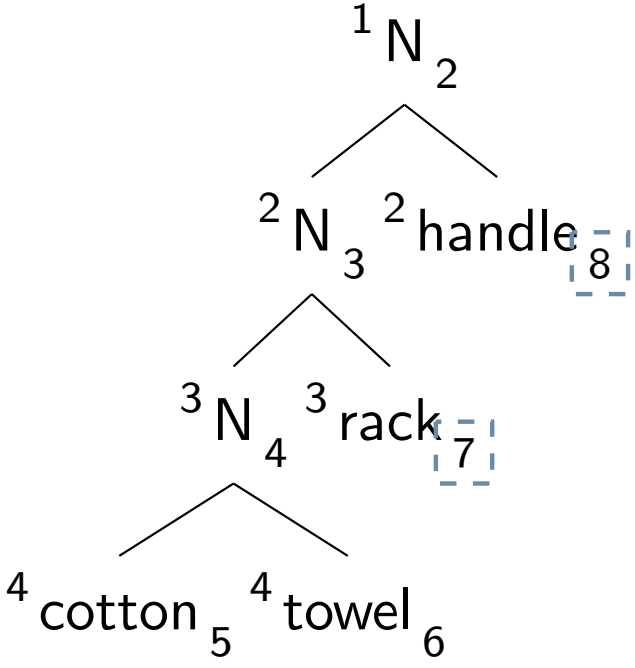


Recursive descent parsing without movement







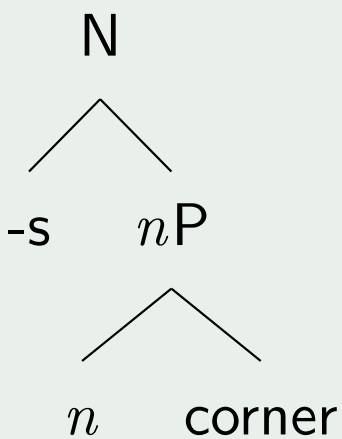


Caution: Linear order ≠ linear order

- Recursive descent parser always builds structure left-to-right.
- What matters is the **order in the input string**, not in the tree!

Example

- Suppose number is an extended projection of nouns (cf. Alec Marantz's blog): [$_n$ corner]-s

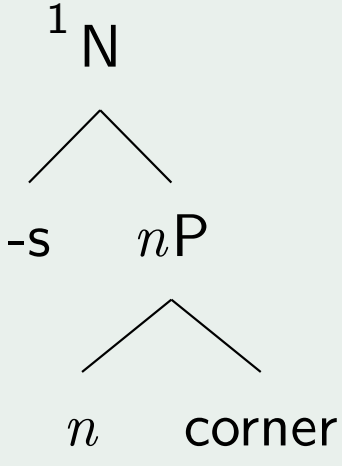


Caution: Linear order ≠ linear order

- Recursive descent parser always builds structure left-to-right.
- What matters is the **order in the input string**, not in the tree!

Example

- Suppose number is an extended projection of nouns (cf. Alec Marantz's blog): [$_n$ corner]-s

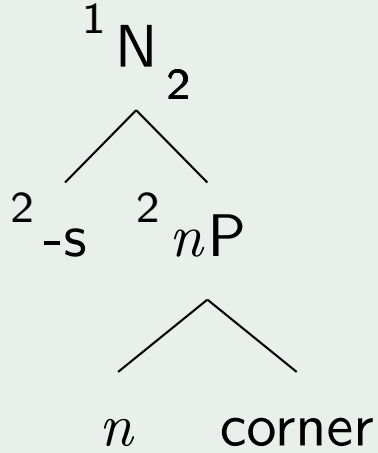


Caution: Linear order ≠ linear order

- Recursive descent parser always builds structure left-to-right.
- What matters is the **order in the input string**, not in the tree!

Example

- Suppose number is an extended projection of nouns (cf. Alec Marantz's blog): [$_n$ corner]-s

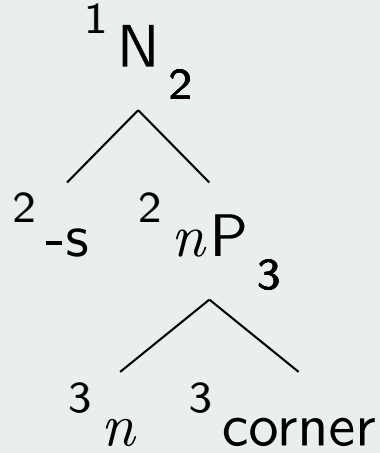


Caution: Linear order ≠ linear order

- Recursive descent parser always builds structure left-to-right.
- What matters is the **order in the input string**, not in the tree!

Example

- Suppose number is an extended projection of nouns (cf. Alec Marantz's blog): [$_n$ corner]-s

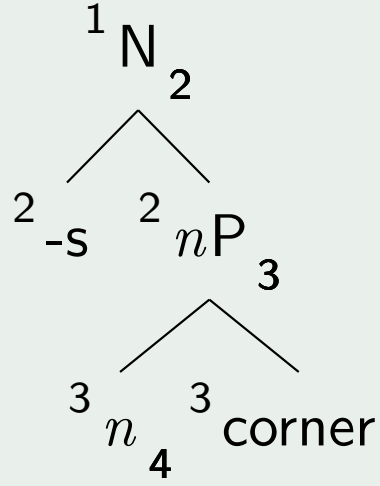


Caution: Linear order ≠ linear order

- Recursive descent parser always builds structure left-to-right.
- What matters is the **order in the input string**, not in the tree!

Example

- Suppose number is an extended projection of nouns (cf. Alec Marantz's blog): [$_n$ corner]-s

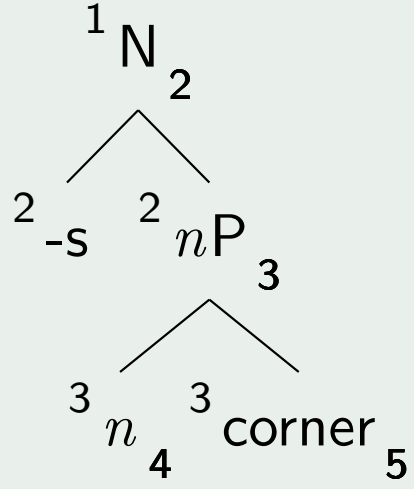


Caution: Linear order ≠ linear order

- Recursive descent parser always builds structure left-to-right.
- What matters is the **order in the input string**, not in the tree!

Example

- Suppose number is an extended projection of nouns (cf. Alec Marantz's blog): $[_n$ corner]-s

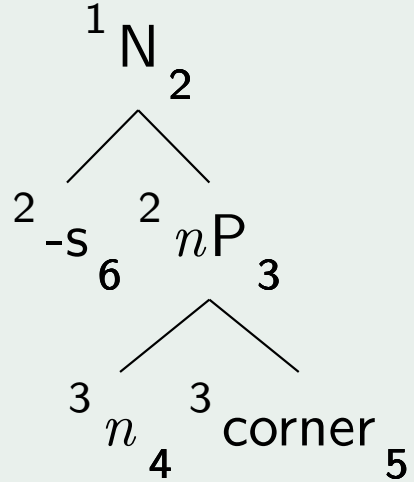


Caution: Linear order ≠ linear order

- Recursive descent parser always builds structure left-to-right.
- What matters is the **order in the input string**, not in the tree!

Example

- Suppose number is an extended projection of nouns (cf. Alec Marantz's blog): $[_n$ corner]-s

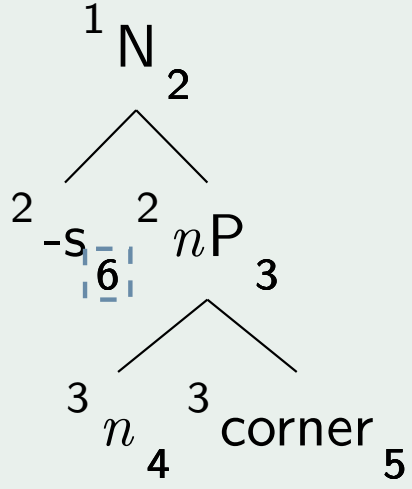


Caution: Linear order ≠ linear order

- Recursive descent parser always builds structure left-to-right.
- What matters is the **order in the input string**, not in the tree!

Example

- Suppose number is an extended projection of nouns (cf. Alec Marantz's blog): $[_n$ corner]-s

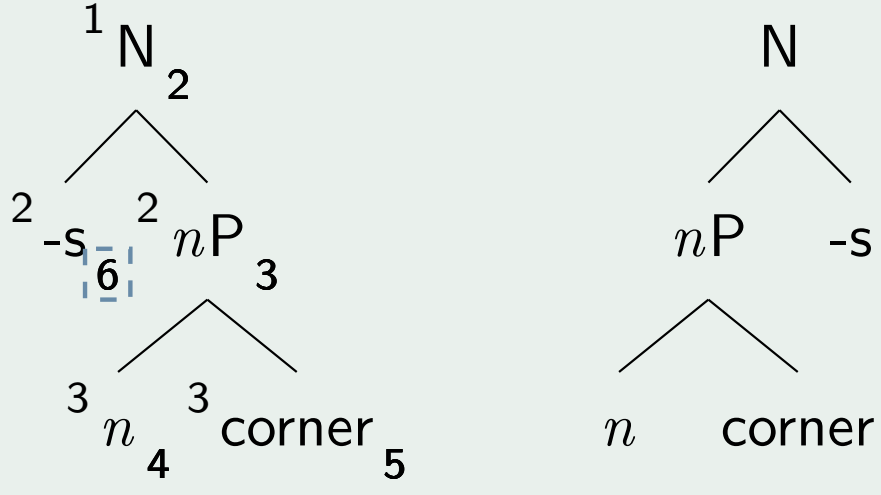


Caution: Linear order ≠ linear order

- Recursive descent parser always builds structure left-to-right.
- What matters is the **order in the input string**, not in the tree!

Example

- Suppose number is an extended projection of nouns (cf. Alec Marantz's blog): [$_n$ corner]-s

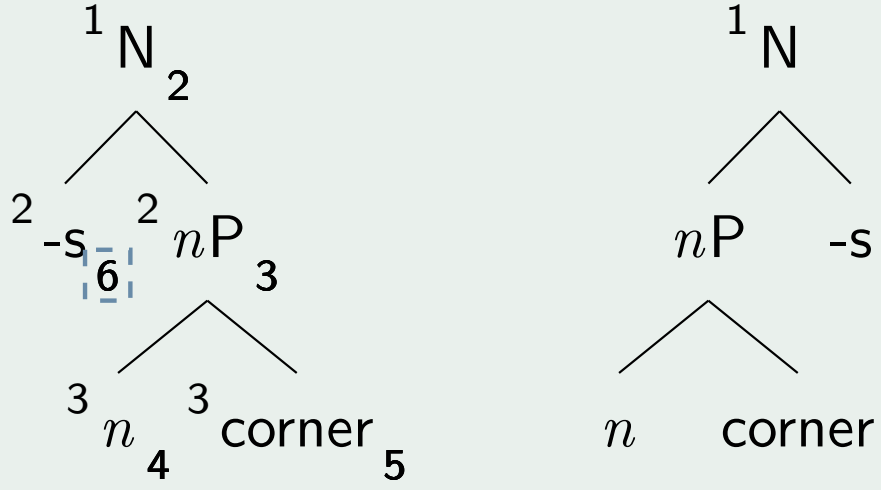


Caution: Linear order ≠ linear order

- Recursive descent parser always builds structure left-to-right.
- What matters is the **order in the input string**, not in the tree!

Example

- Suppose number is an extended projection of nouns (cf. Alec Marantz's blog): [$_n$ corner]-s

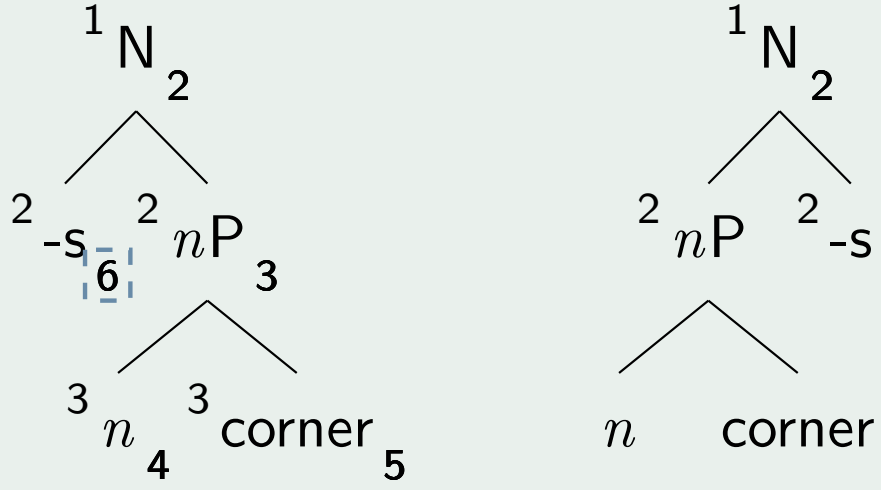


Caution: Linear order ≠ linear order

- Recursive descent parser always builds structure left-to-right.
- What matters is the **order in the input string**, not in the tree!

Example

- Suppose number is an extended projection of nouns (cf. Alec Marantz's blog): [$_n$ corner]-s

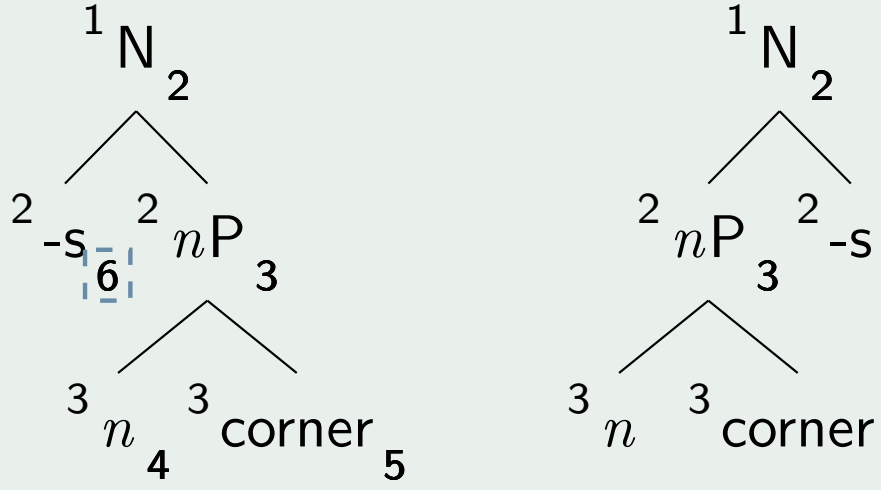


Caution: Linear order ≠ linear order

- Recursive descent parser always builds structure left-to-right.
- What matters is the **order in the input string**, not in the tree!

Example

- Suppose number is an extended projection of nouns (cf. Alec Marantz's blog): $[_n$ corner]-s

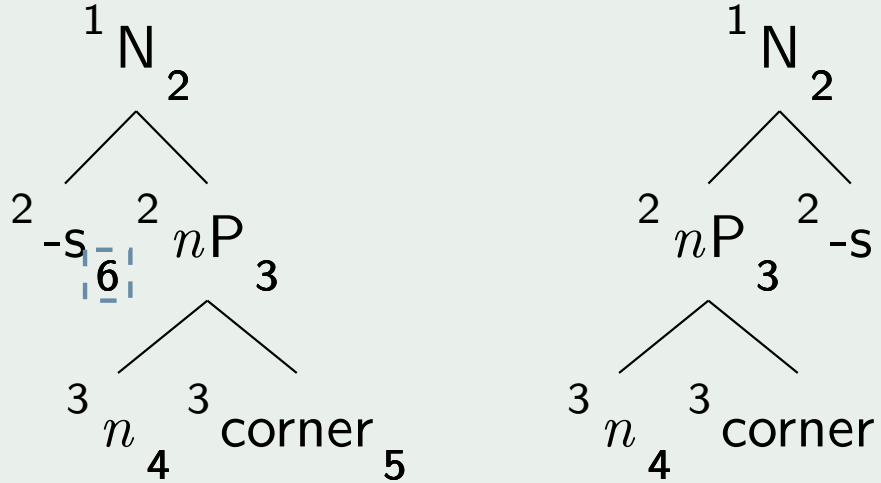


Caution: Linear order ≠ linear order

- Recursive descent parser always builds structure left-to-right.
- What matters is the **order in the input string**, not in the tree!

Example

- Suppose number is an extended projection of nouns (cf. Alec Marantz's blog): $[_n$ corner]-s

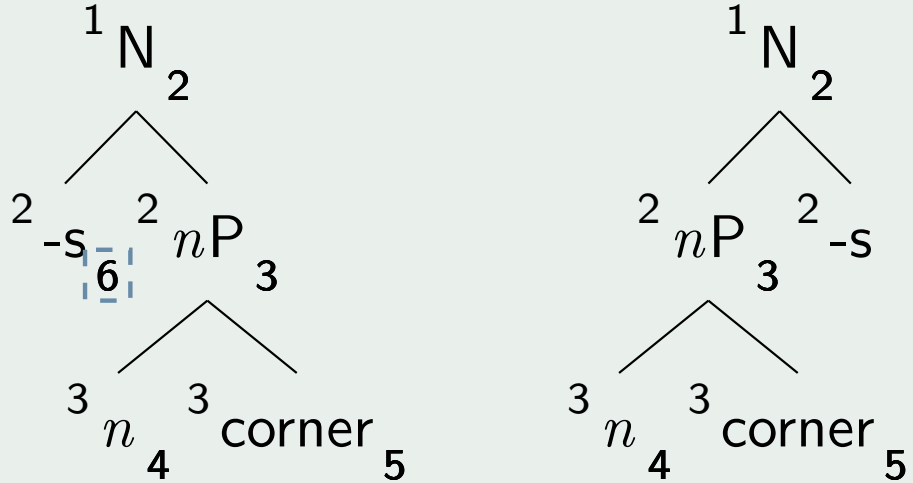


Caution: Linear order ≠ linear order

- Recursive descent parser always builds structure left-to-right.
- What matters is the **order in the input string**, not in the tree!

Example

- Suppose number is an extended projection of nouns (cf. Alec Marantz's blog): $[_n$ corner]-s

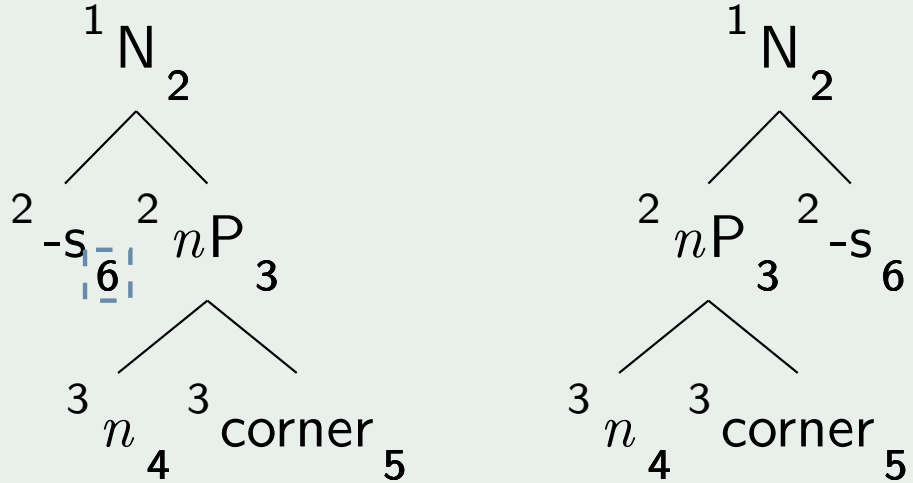


Caution: Linear order ≠ linear order

- Recursive descent parser always builds structure left-to-right.
- What matters is the **order in the input string**, not in the tree!

Example

- Suppose number is an extended projection of nouns (cf. Alec Marantz's blog): [$_n$ corner]-s

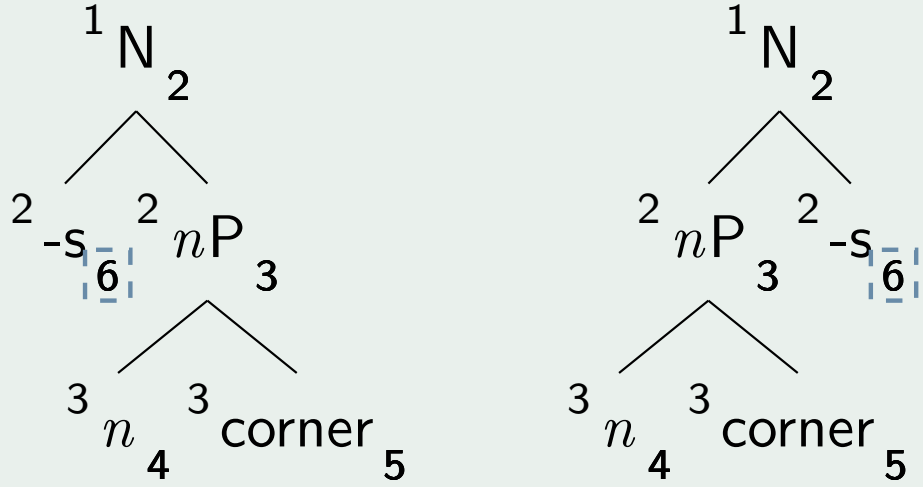


Caution: Linear order ≠ linear order

- Recursive descent parser always builds structure left-to-right.
- What matters is the **order in the input string**, not in the tree!

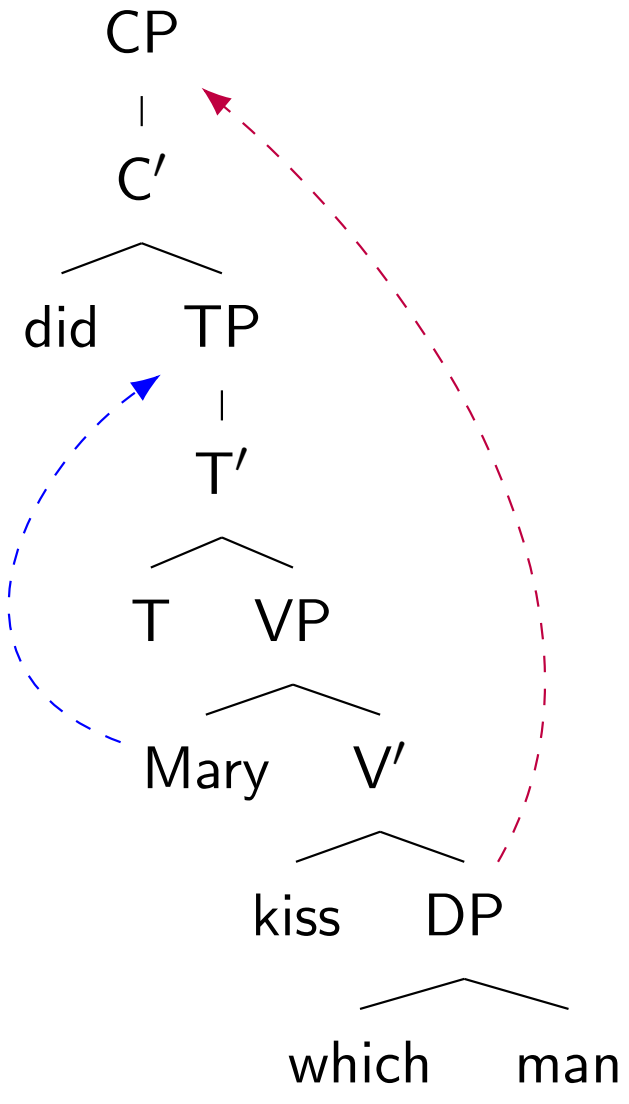
Example

- Suppose number is an extended projection of nouns (cf. Alec Marantz's blog): [$_n$ corner]-s



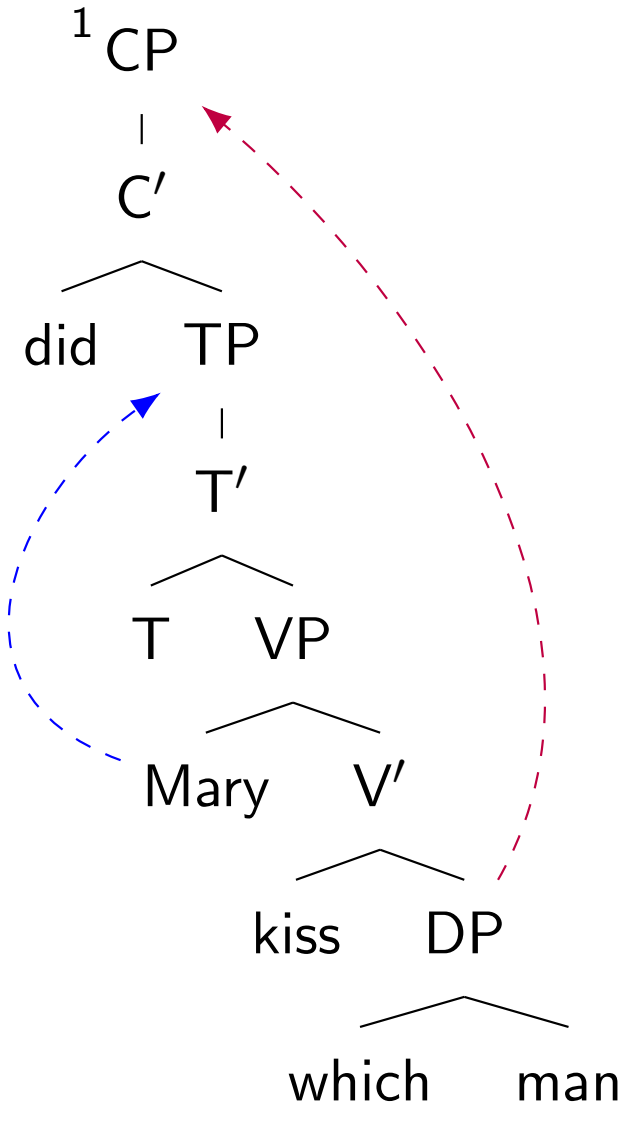
Handling movement

- Movement is simple. Just track its effect on linear order!



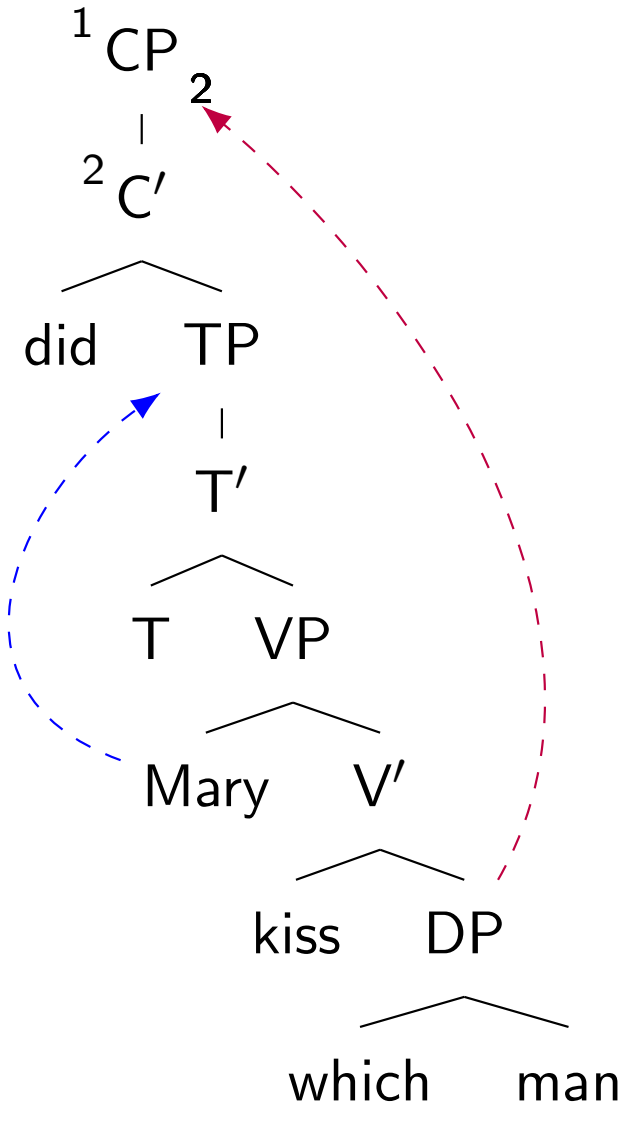
Handling movement

- Movement is simple. Just track its effect on linear order!



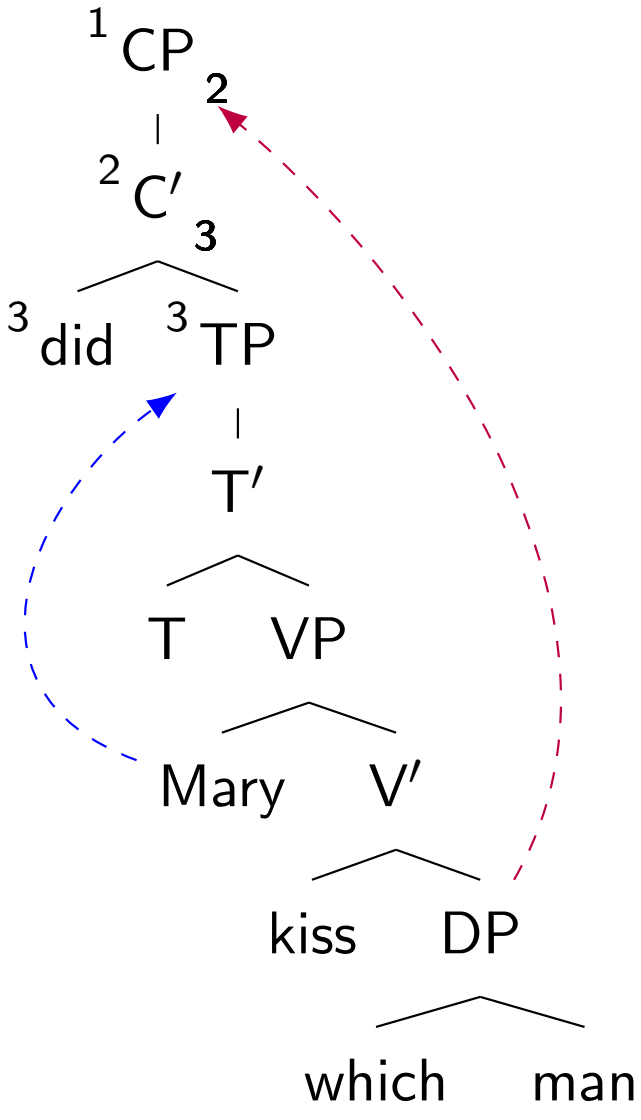
Handling movement

- Movement is simple. Just track its effect on linear order!



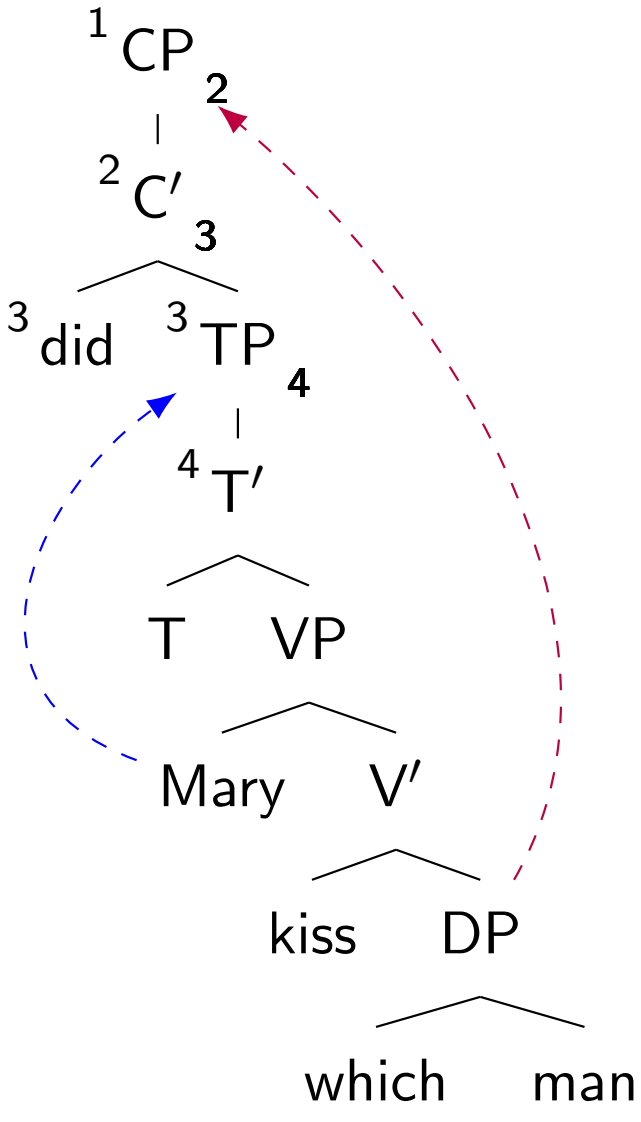
Handling movement

- Movement is simple. Just track its effect on linear order!



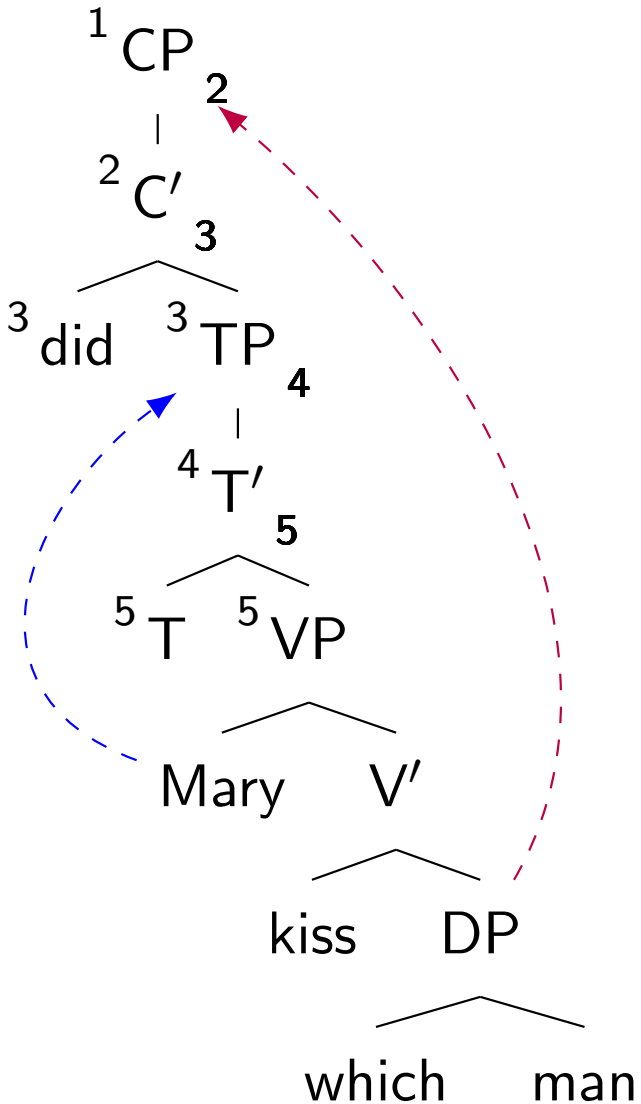
Handling movement

- Movement is simple. Just track its effect on linear order!



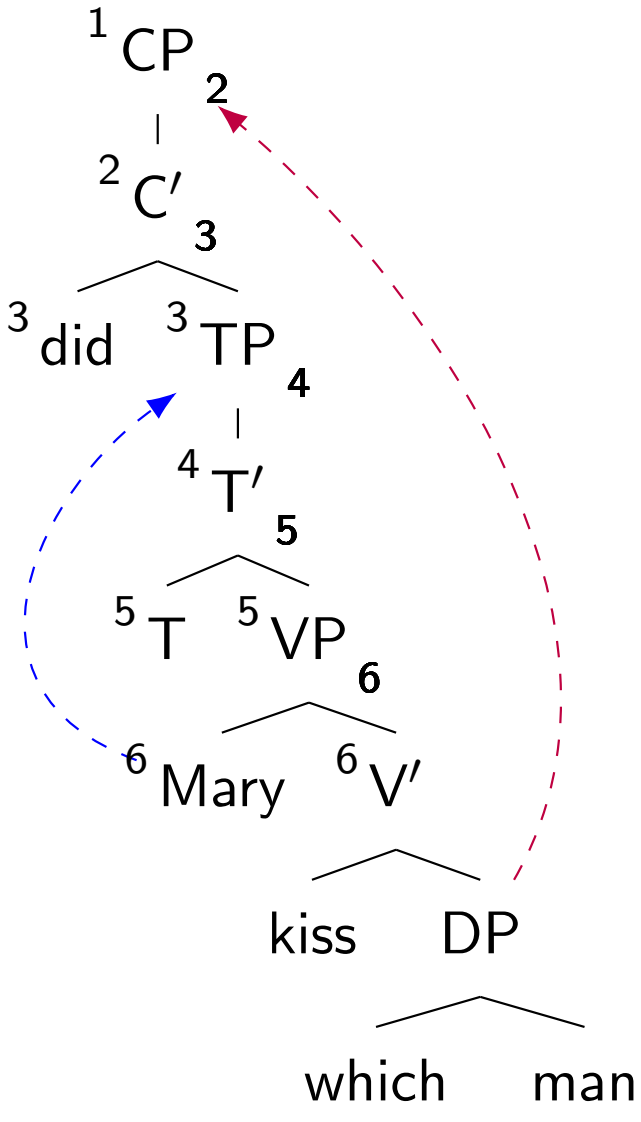
Handling movement

- Movement is simple. Just track its effect on linear order!



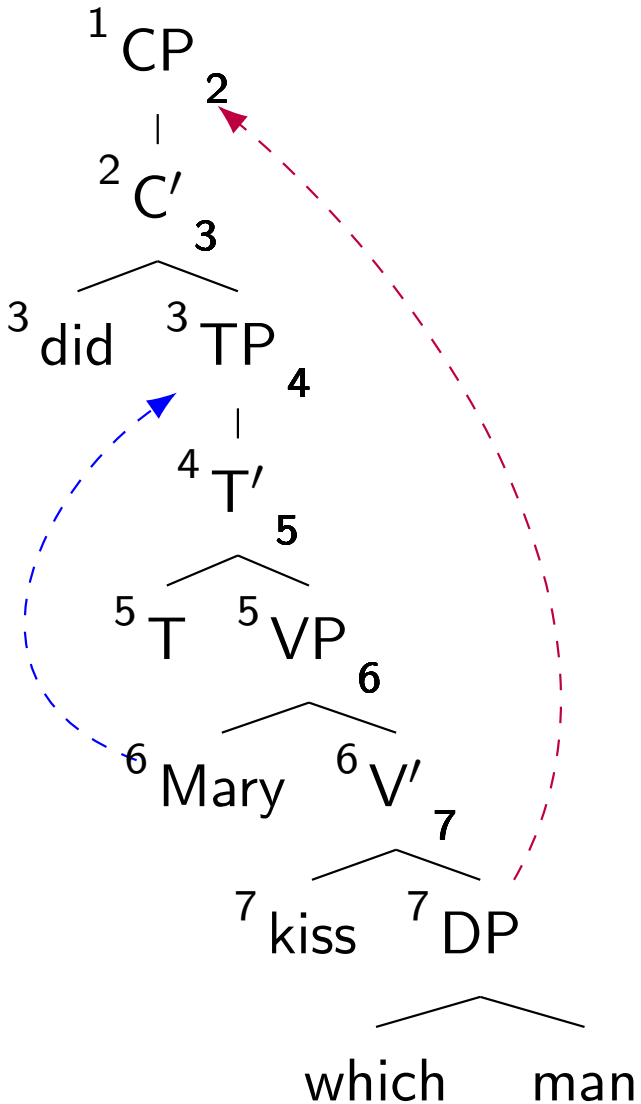
Handling movement

- Movement is simple. Just track its effect on linear order!



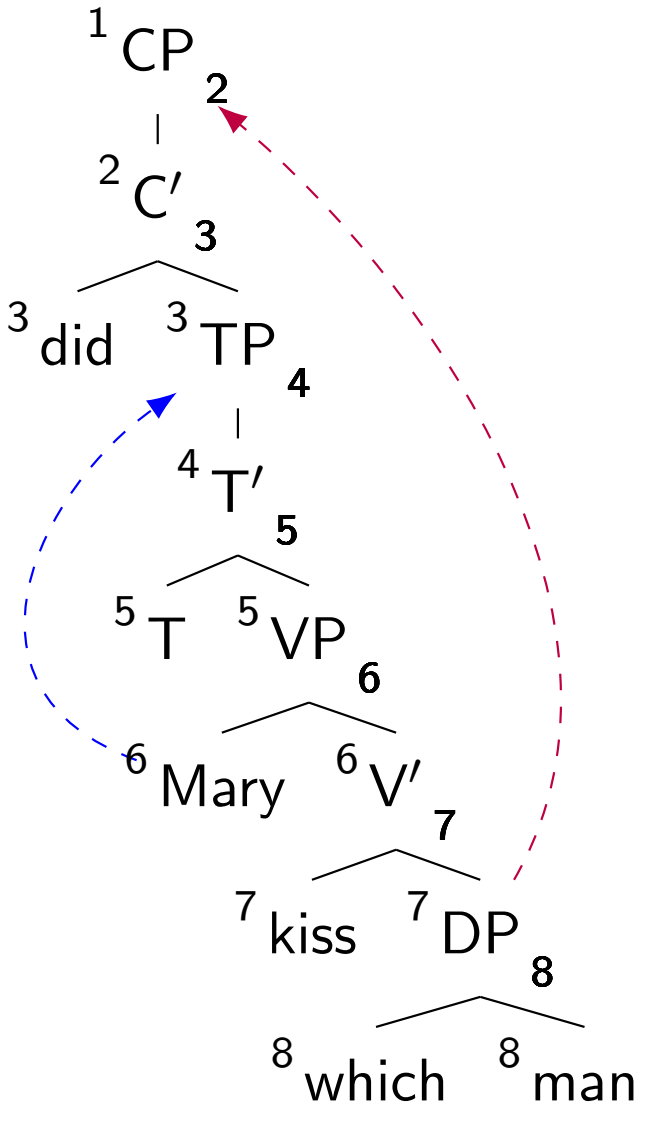
Handling movement

- Movement is simple. Just track its effect on linear order!



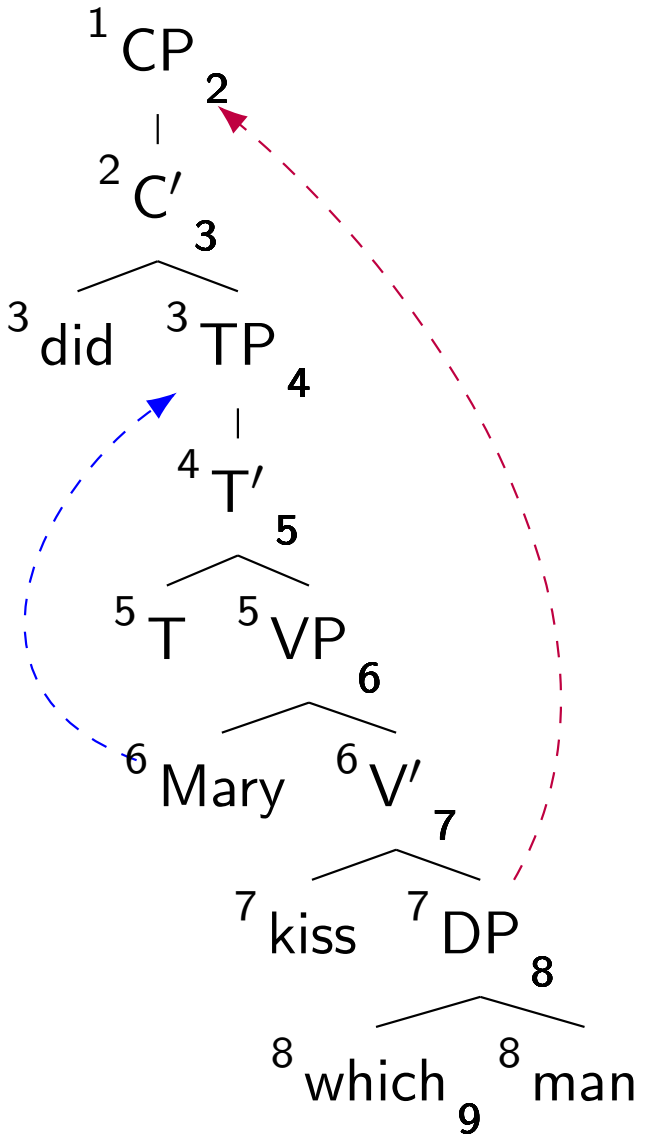
Handling movement

- Movement is simple. Just track its effect on linear order!



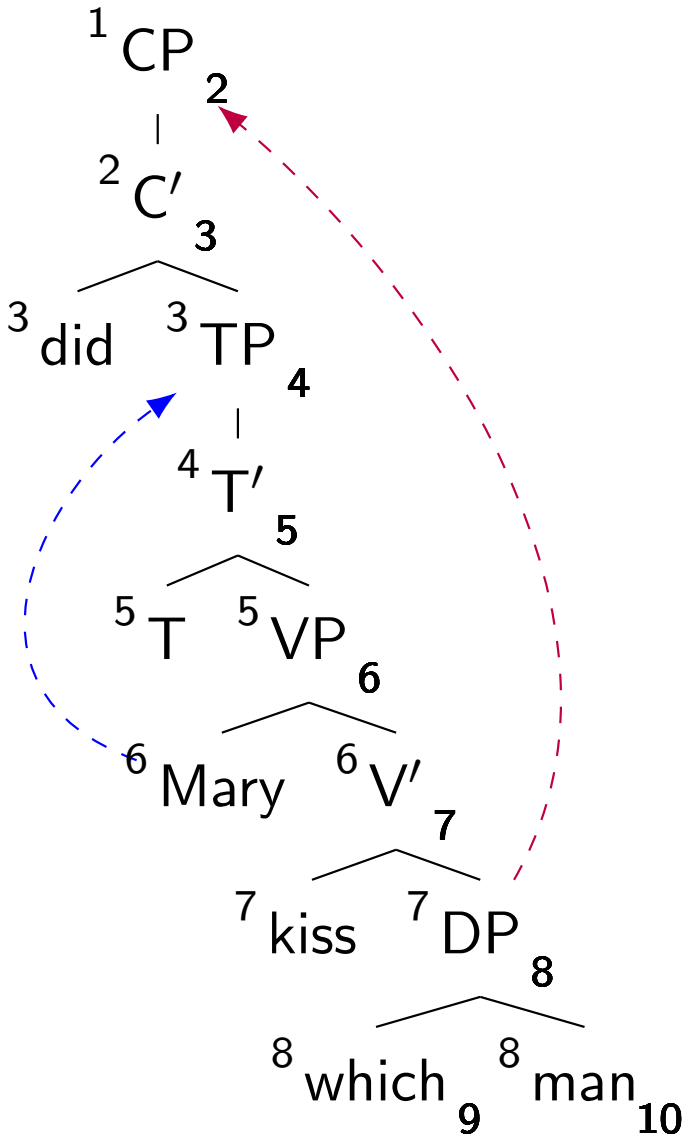
Handling movement

- Movement is simple. Just track its effect on linear order!



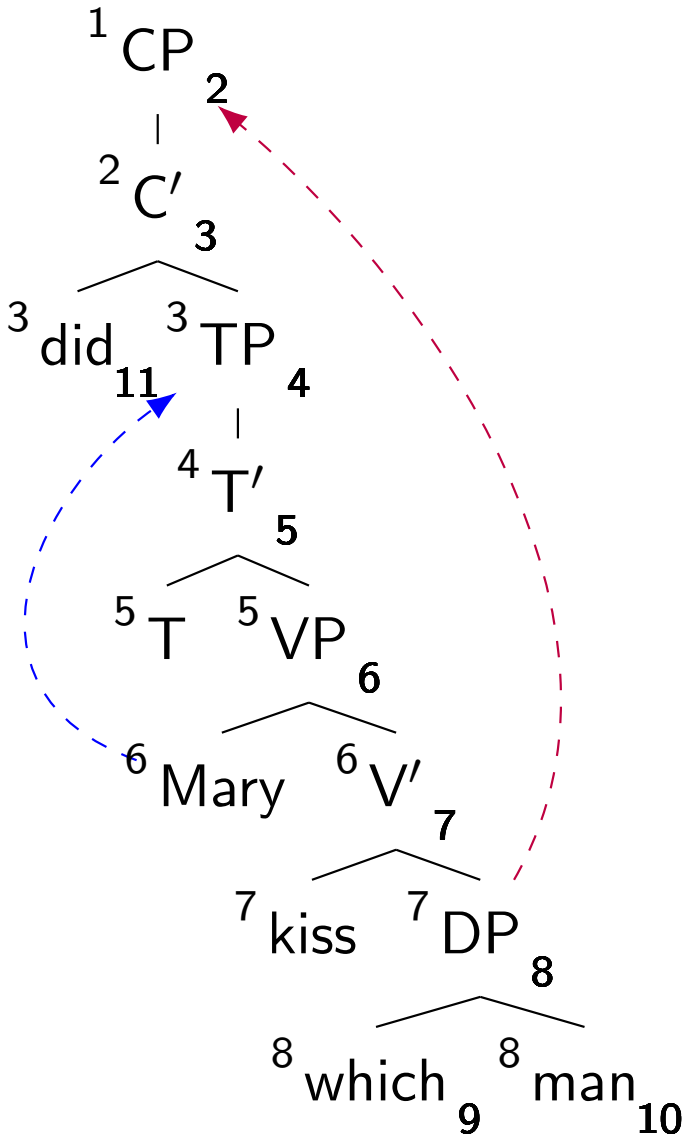
Handling movement

- Movement is simple. Just track its effect on linear order!



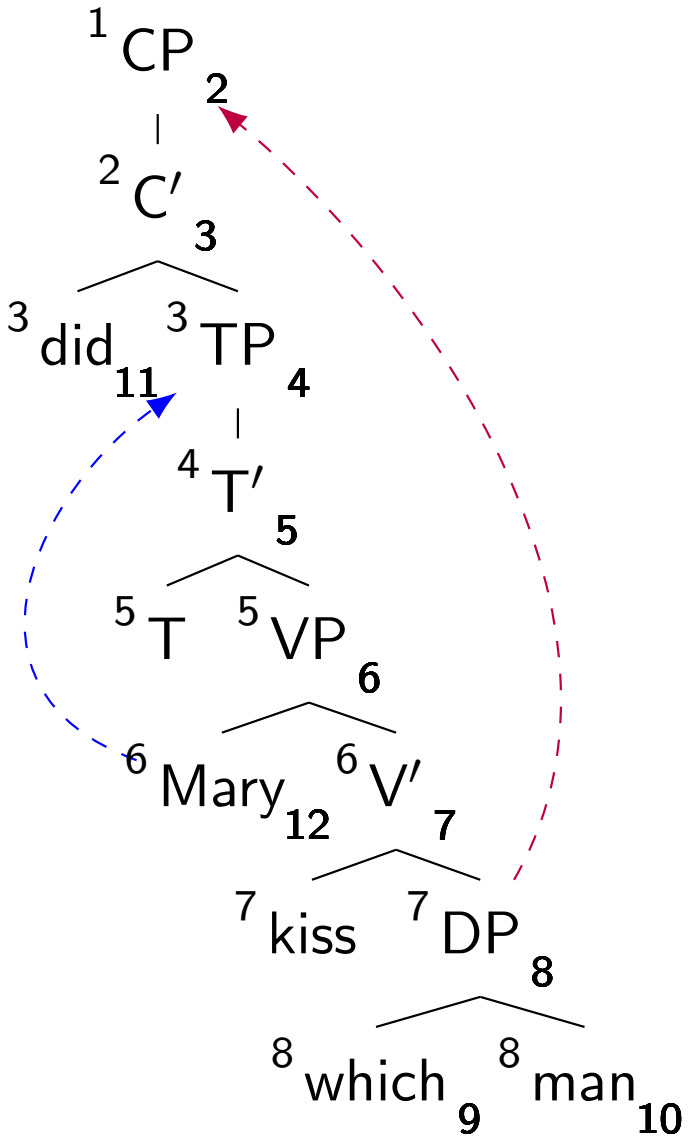
Handling movement

- Movement is simple. Just track its effect on linear order!



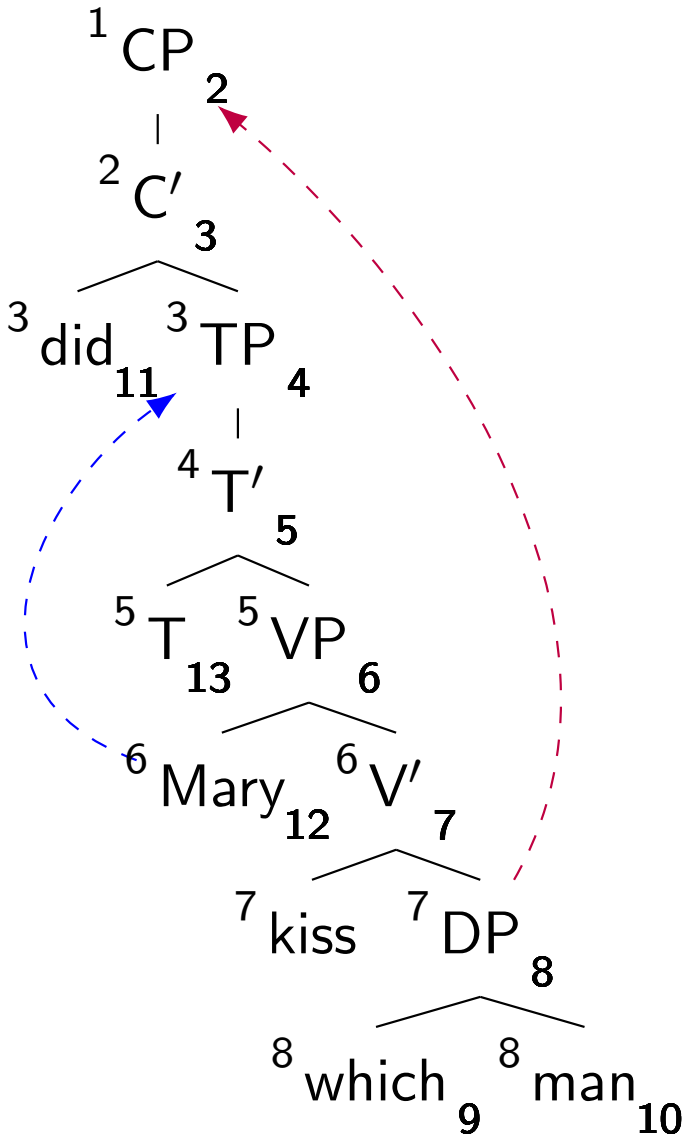
Handling movement

- Movement is simple. Just track its effect on linear order!



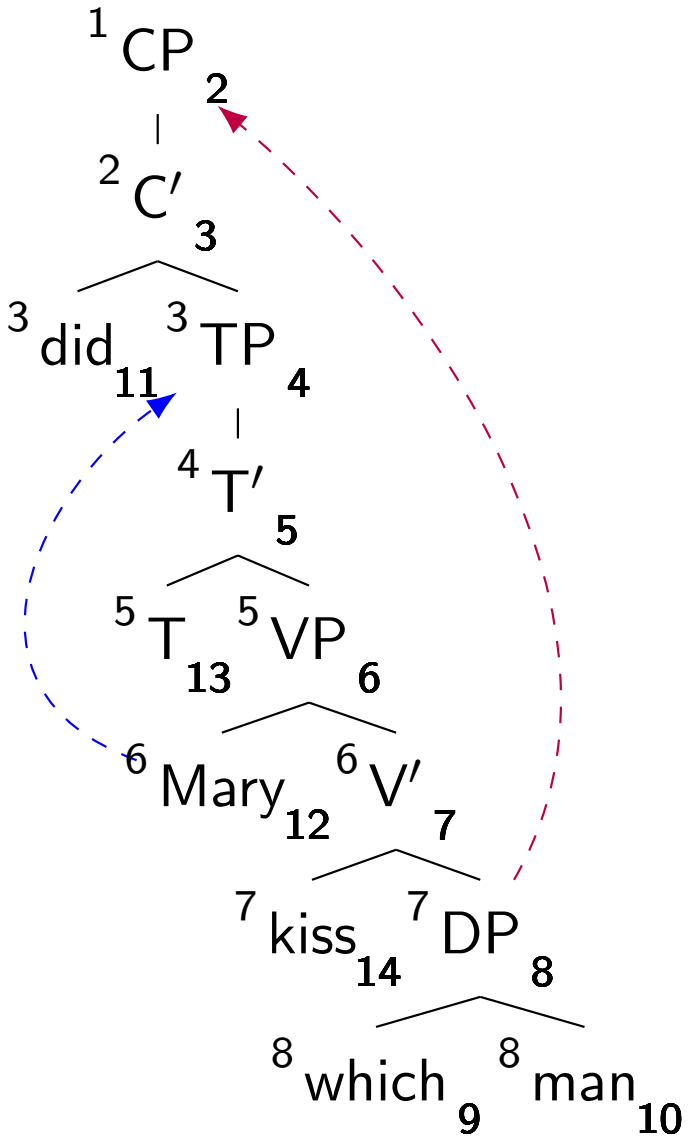
Handling movement

- Movement is simple. Just track its effect on linear order!



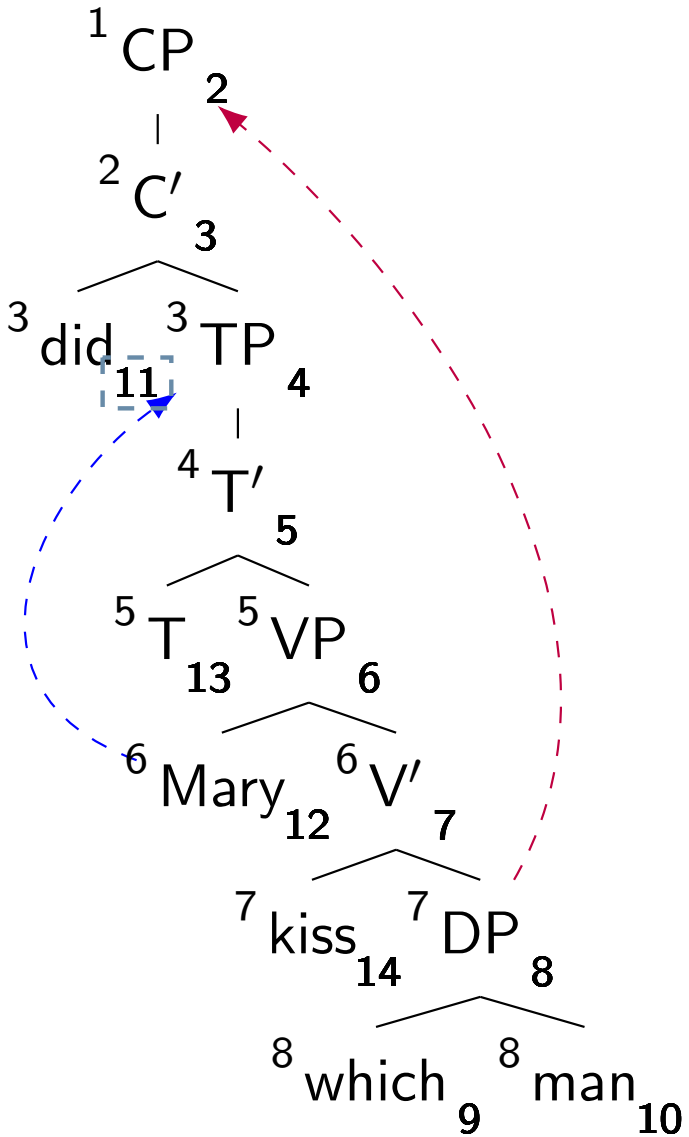
Handling movement

- Movement is simple. Just track its effect on linear order!



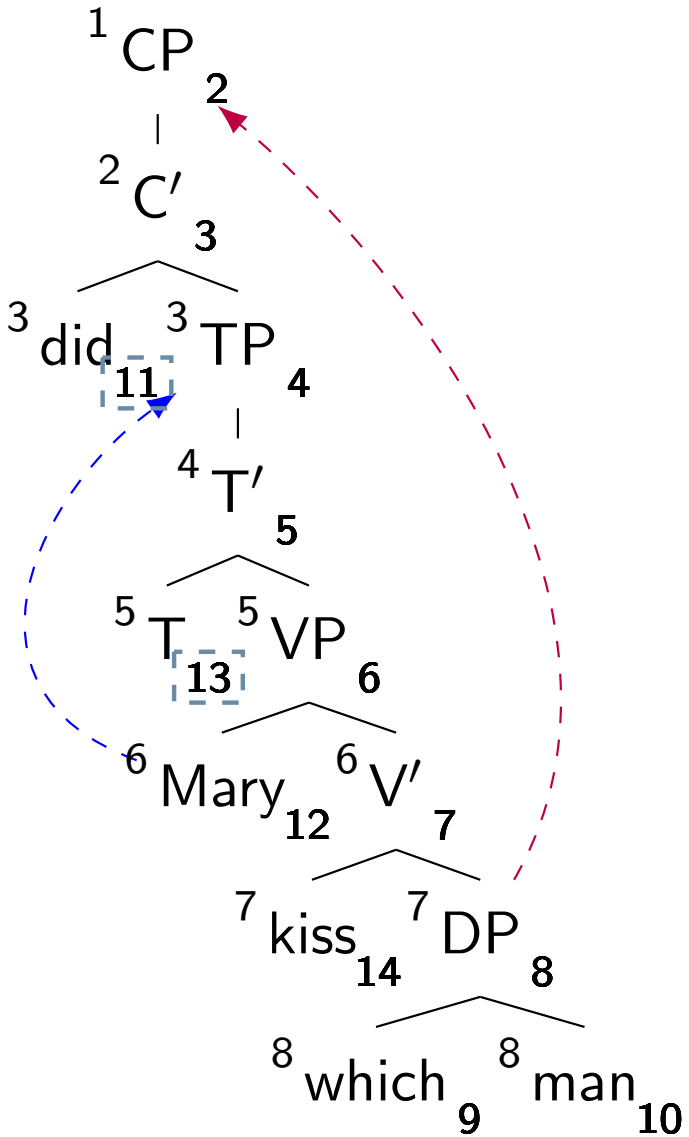
Handling movement

- Movement is simple. Just track its effect on linear order!



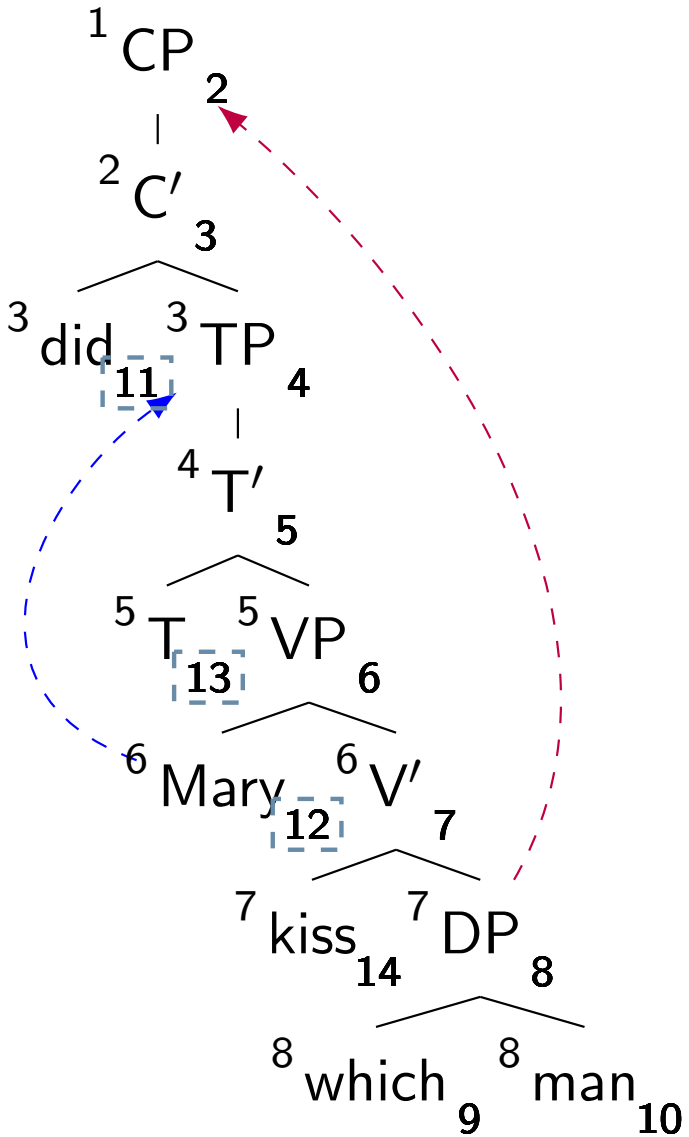
Handling movement

- Movement is simple. Just track its effect on linear order!



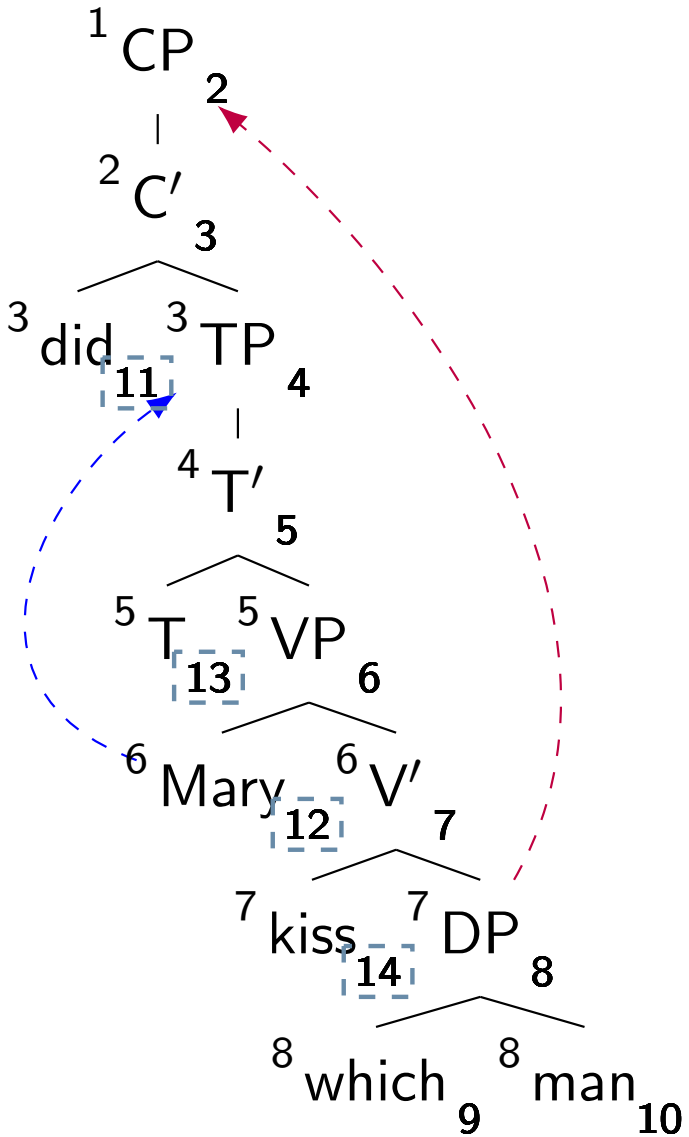
Handling movement

- Movement is simple. Just track its effect on linear order!



Handling movement

- Movement is simple. Just track its effect on linear order!



General methodology

1. Pick a processing contrast observed in the literature
2. Pick Minimalist analysis from literature
   (often multiple competing options, then we try all)
3. Construct MG derivation trees according to analysis.
4. Annotate derivation trees.
5. Compute memory usage metrics:

Tenure	|node's index − node's oudex|
MaxTen	largest tenure in derivation
Size	|mover's index − outdex of mover's landing site|
SumSize	sum of size values

6. Preferred structure should have lower values

SRC $<$ ORC: The basics

- SRCs are processed faster than ORCs in languages with post-nominal RCs:
  English, French, German, ...
  (Mecklinger et al. 1995; Gibson 2000; Gordon et al. 2006)

(1) **SRC $<$ ORC in English**

a. The reporter$_i$ [$_{\text{RC}}$ who t_i attacked the senator] admitted the error.

b. The reporter$_i$ [$_{\text{RC}}$ who the senator attacked t_i] admitted the error.

- This can be explained in terms of string distance between the head noun and the gap.

SRC < ORC: The challenge

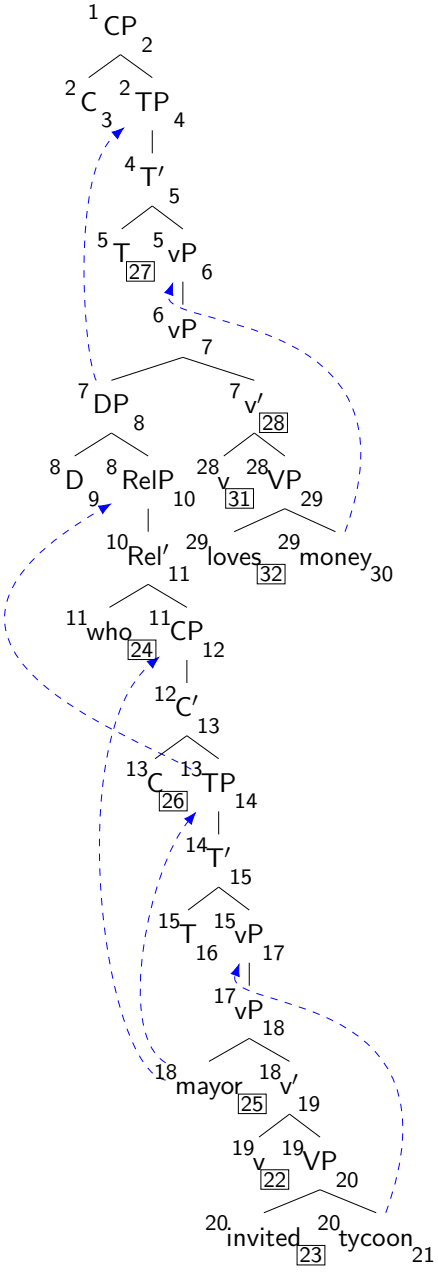
- The SRC-advantage also holds with pre-nominal RCs: Japanese, Korean, Chinese
  (Gibson and Wu 2013; Miyamoto and Nakamura 2013)

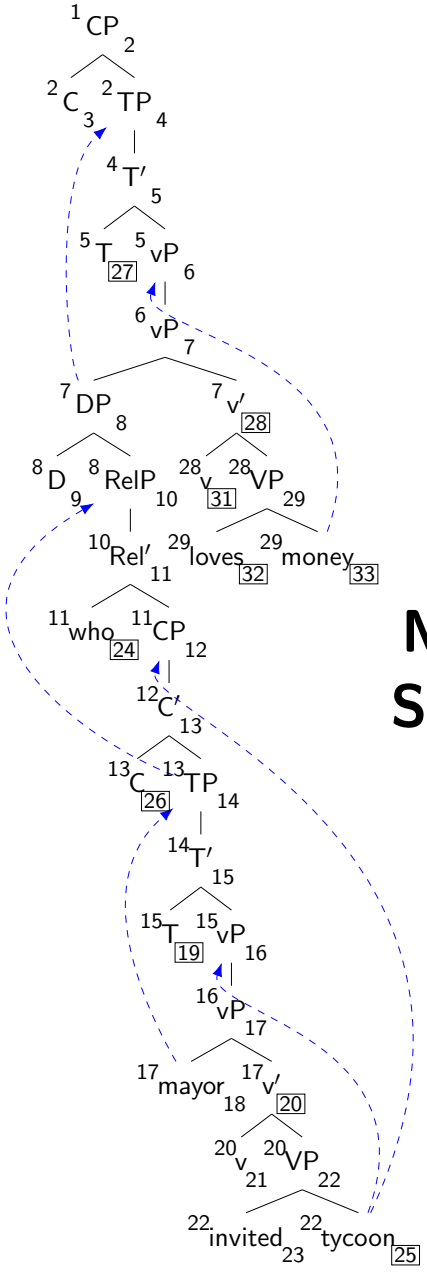
(2) **SRC < ORC in Japanese**

a. [$_{\text{RC}}$ t_i giin-ga hinanshita] kisha$_i$-ga
senator-ACC attacked reporter-NOM
ayamari-o mitometa
error-ACC admitted

b. [$_{\text{RC}}$ giin-ga t_i hinanshita] kisha$_i$-ga
senator-NOM attacked reporter-NOM
ayamari-o mitometa
error-ACC admitted

- String distance fails here.
- But a structural account succeeds.





	SRC	ORC
MaxTen	22	22
SumSize	38	42

Taking stock

- The MG processing model ties use to knowledge:
  processing complexity $\equiv$ structural complexity
- The linking hypothesis is maximally simple:
  1. top-down parsing
  2. simple memory usage metrics
- Every syntactic analysis is also a processing model.
- Doing syntax $\equiv$ doing psycholinguistics

For future reference: Two recent entry points

Cyclic scope and processing difficulty in a Minimalist parser*

Robert Pasternak
Leibniz-Center for General Linguistics (ZAS)
mail@robertpasternak.com

Thomas Graf
Stony Brook University
mail@thomasgraf.net

Structure and Memory:

A Computational Model of Storage, Gradience, and Priming

by

Aniello De Santo

Doctor of Philosophy

in

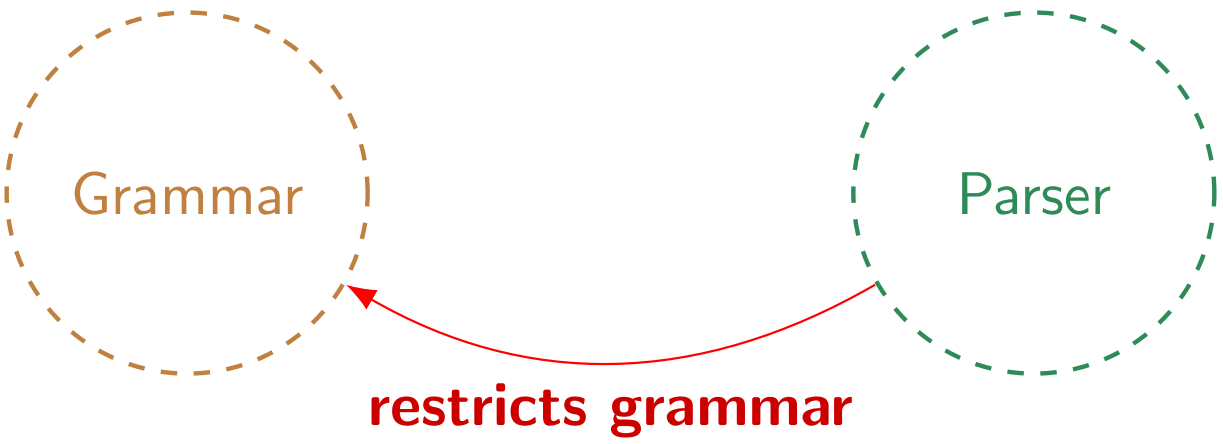
Linguistics

Stony Brook University

2020

aniellodesanto.github.io

The mid-talk turn-around



Third-factor considerations

- Minimalist assumption: syntax limited by external factors (Chomsky 2005)
- If the parser is indeed top-down,
  syntax should be easy to parse top-down.
- But what does "easy" mean?

Roadmap to a definition of "easy"

1. Syntactic computation as **tree automata**
2. Identify an efficient class of top-down tree automata

Syntactic computation as automata

- Syntactic computations can be encoded as **tree automata**. (cf. Morawietz and Cornell 1999; Kobele et al. 2007)
- Each node in the tree is assigned a **state**.
- The tree is well-formed iff the state assignment is valid.
- For top-down efficiency, we want tree automata that are:

finite-state the set of available states is finite

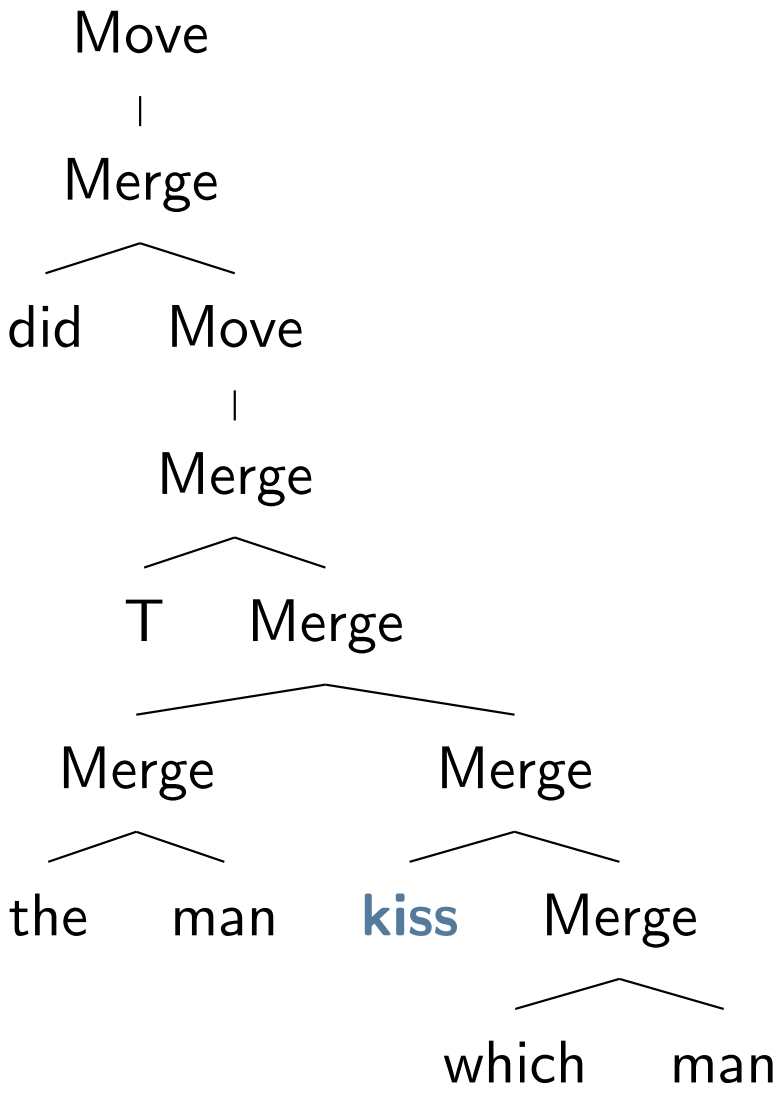
deterministic when deciding what state to assign to a node n, there is only one option

- But this limits what is computable!

limited automaton $\equiv$ limited syntactic computation

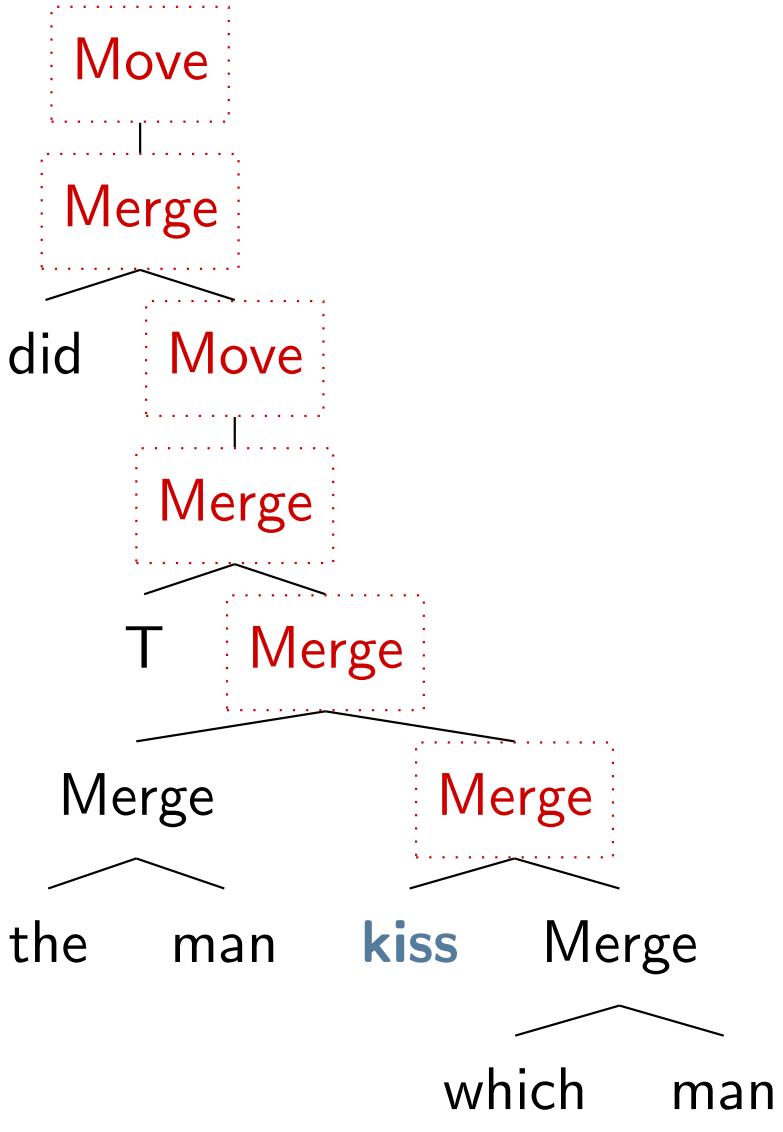
Standard deterministic tree automata are too weak

- Deterministic top-down tree automata are myopic.
- They can't even handle c-selection.
- What state is assigned to a node n depends only on
  1. the state of the mother of n, and
  2. the label of the mother of n.
- **Corollary:** state of n depends only on ancestors of n
- We can visualize this as the **s[yntactic]-horizon** of n.



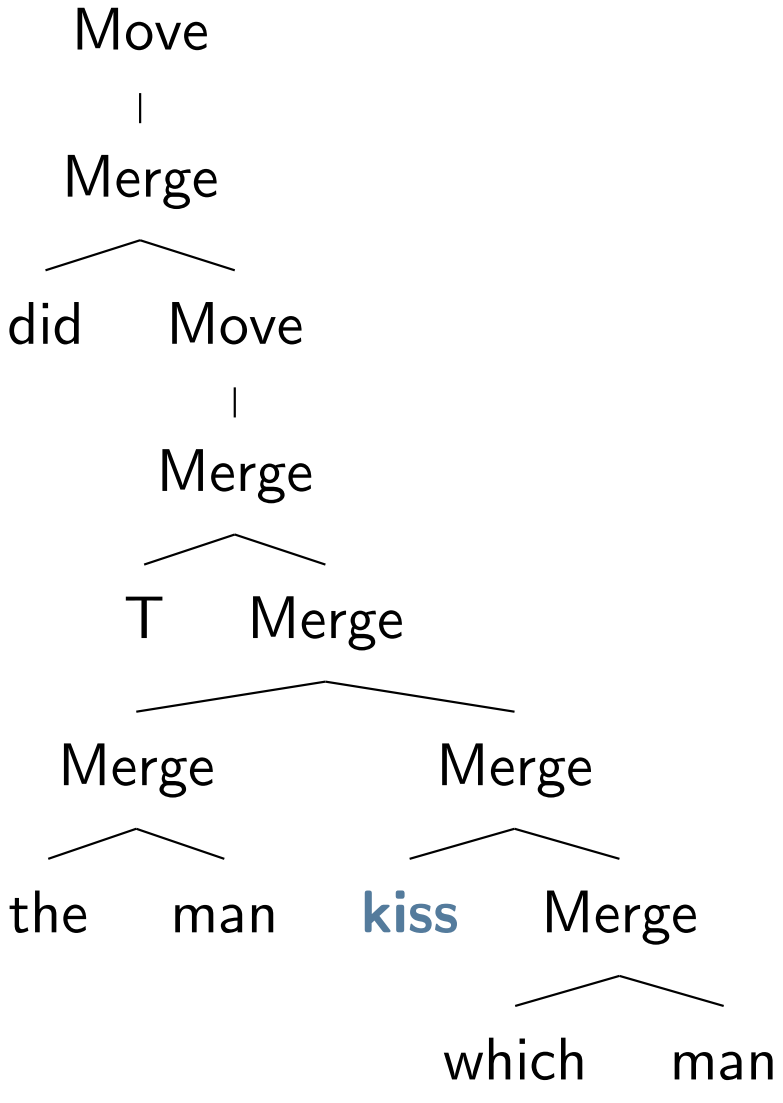
Standard deterministic tree automata are too weak

- Deterministic top-down tree automata are myopic.
- They can't even handle c-selection.
- What state is assigned to a node n depends only on
  1. the state of the mother of n, and
  2. the label of the mother of n.
- **Corollary:** state of n depends only on ancestors of n
- We can visualize this as the **s[yntactic]-horizon** of n.



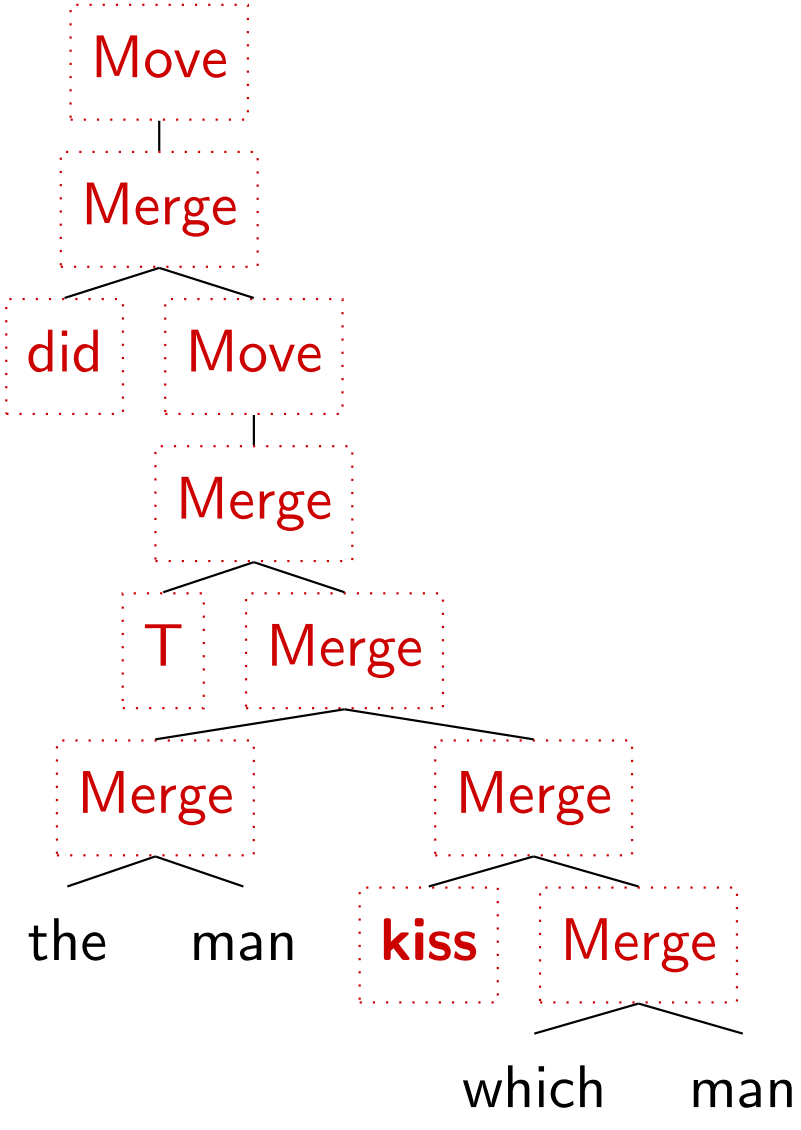
Sensing tree automata (STAs)

- STAs have a more powerful state assignment mechanism.
- What state is assigned to a node n depends only on
  1. the state of the mother of n, and
  2. the label of the mother of n, and
  3. the labels of n's siblings, and
  4. the label of n itself.
- **Result:** larger s-horizon



Sensing tree automata (STAs)

- STAs have a more powerful state assignment mechanism.
- What state is assigned to a node n depends only on
  1. the state of the mother of n, and
  2. the label of the mother of n, and
  3. the labels of n's siblings, and
  4. the label of n itself.
- **Result:** larger s-horizon



A final tweak: Head-sensing tree automata

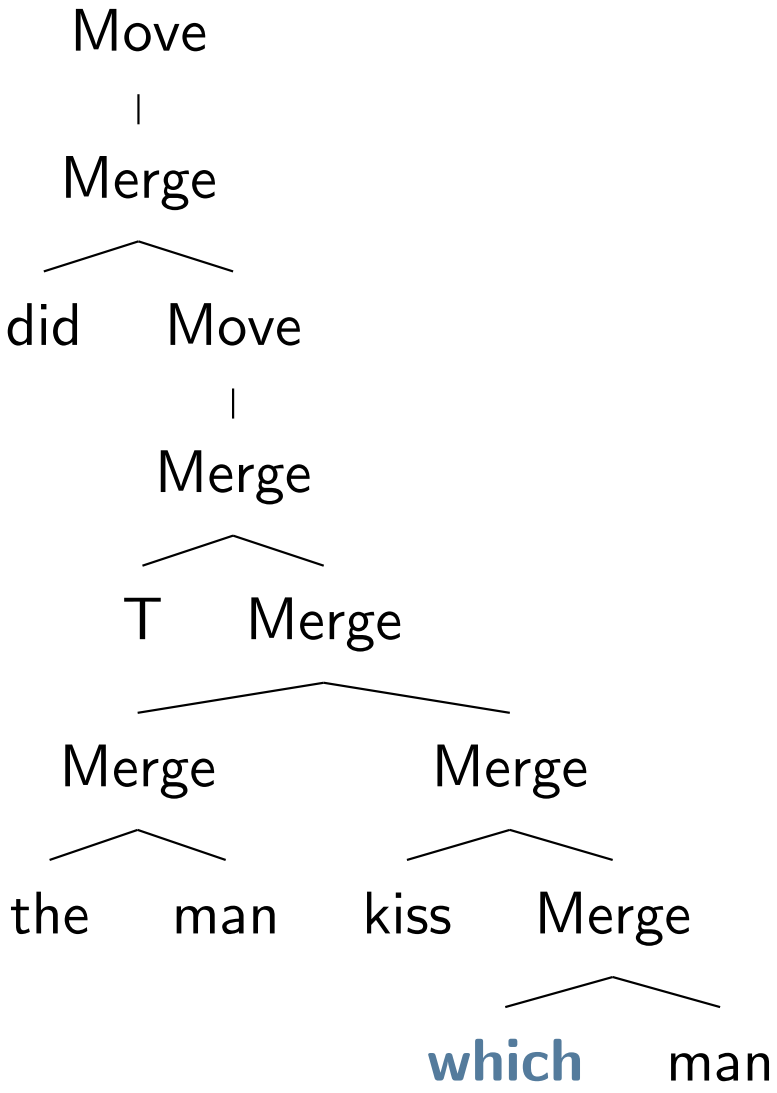
- STAs look promising, almost like c-command.
- But they're stipulative from a Minimalist perspective.
- How can we make STAs linguistically natural?
- Adapt them to Merge: "mother" and "sibling" don't matter, we want **head-argument relations**.

Head-sensing tree automata (HSTAs)

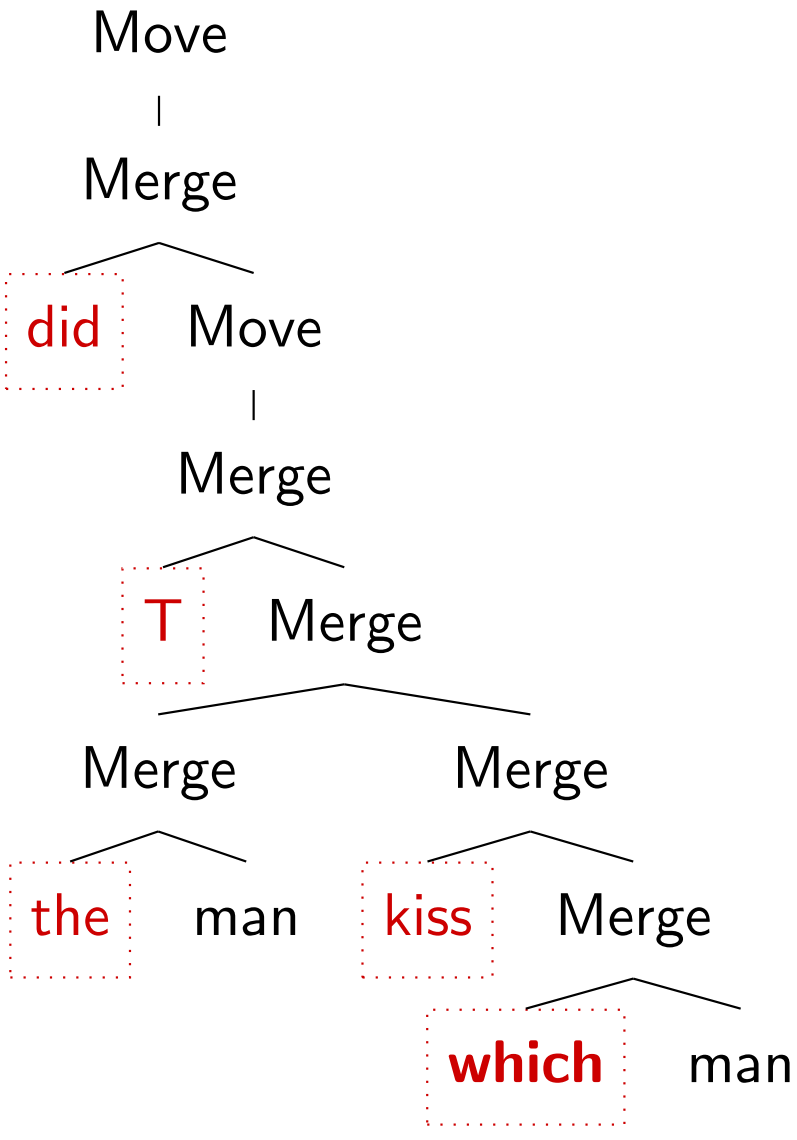
What state is assigned to a head h depends only on

1. the state of the head selecting h, and
2. the label/features of the head selecting h, and
3. the label/features of the head of each co-argument of h, and
4. the label/features of h itself.

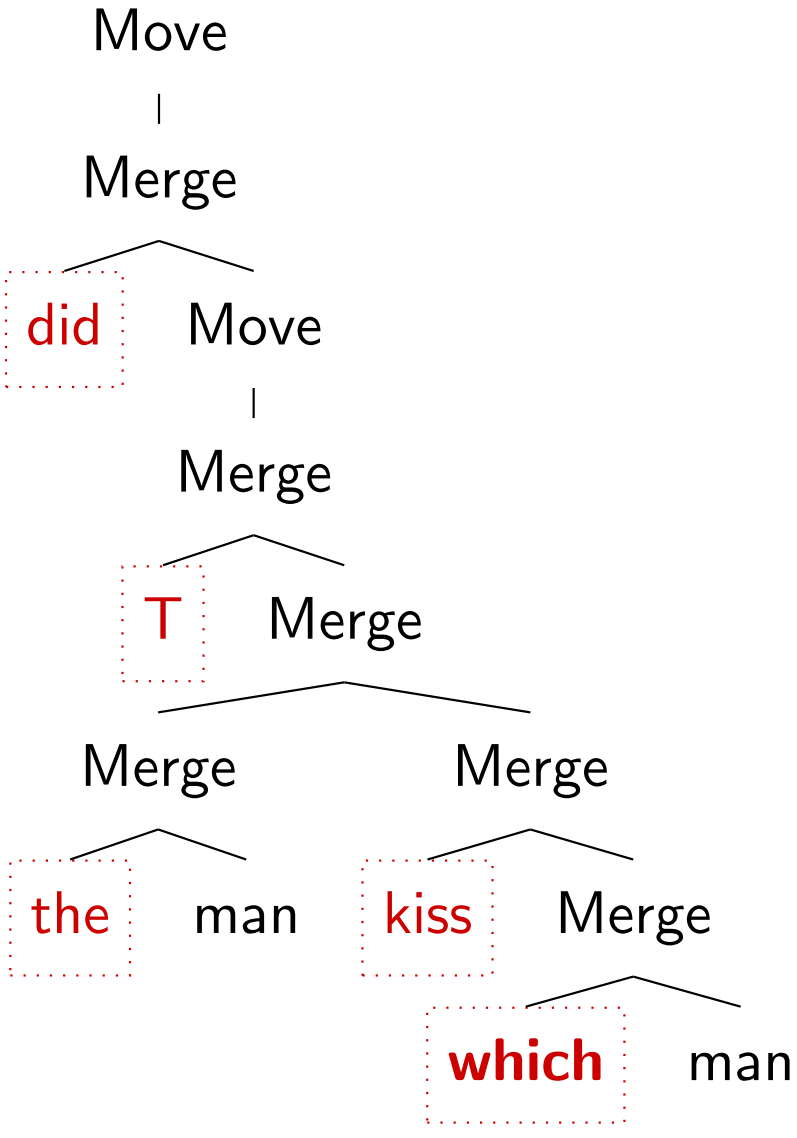
Visualizing HSTAs with dependency trees

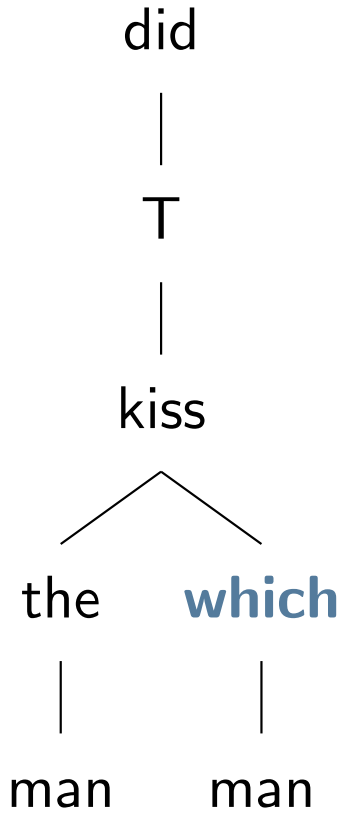


Visualizing HSTAs with dependency trees

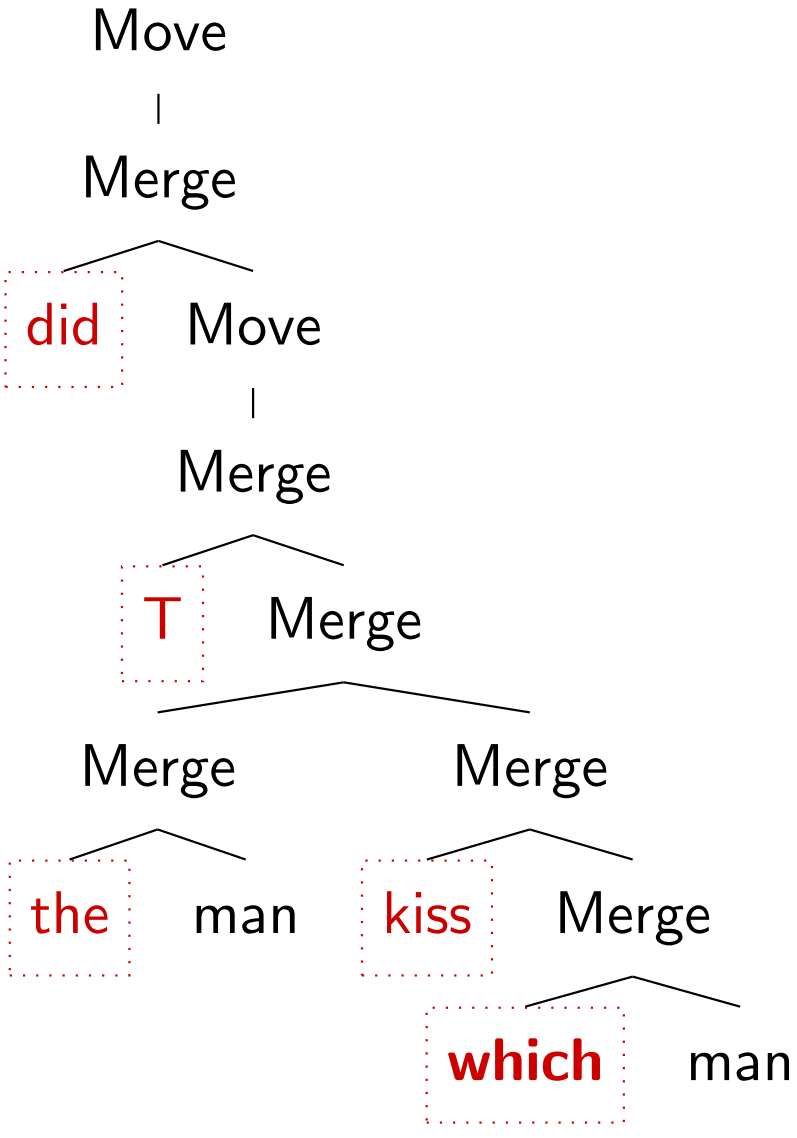


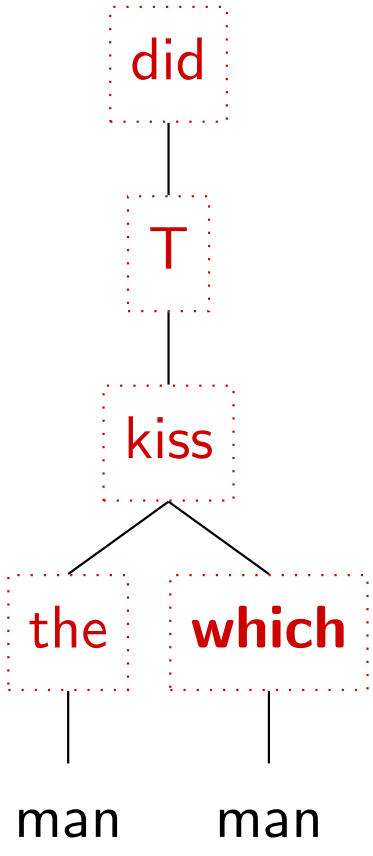
Visualizing HSTAs with dependency trees





Visualizing HSTAs with dependency trees

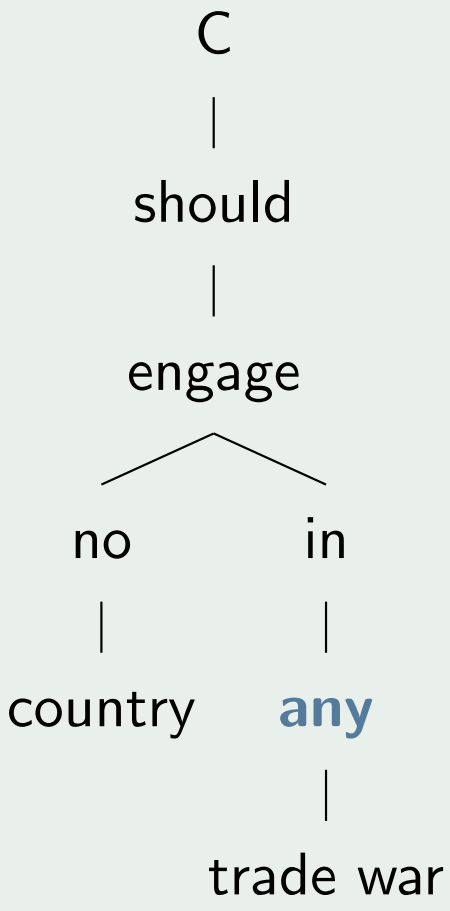




HSTAs for syntactic constraints

- HSTAs can handle almost everything in syntax.
  (Graf and De Santo 2019; Graf and Shafiei 2019)

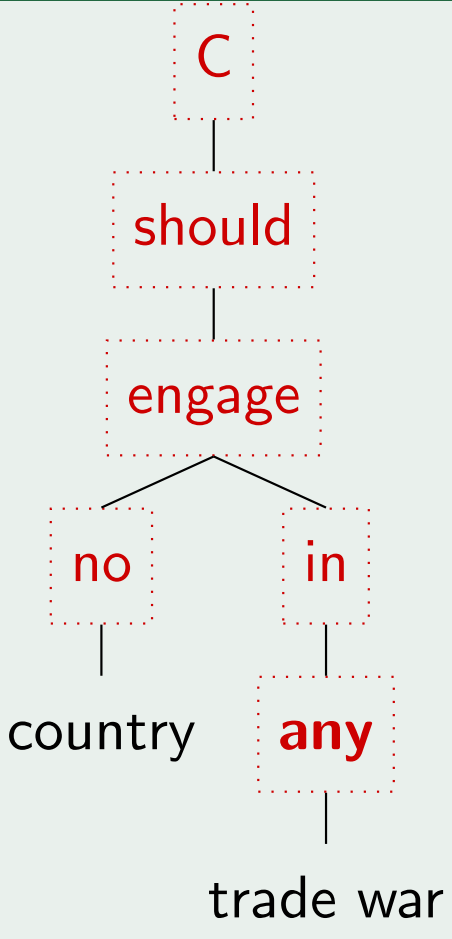
Example: NPI-licensing upward and downward



HSTAs for syntactic constraints

- HSTAs can handle almost everything in syntax.
  (Graf and De Santo 2019; Graf and Shafiei 2019)

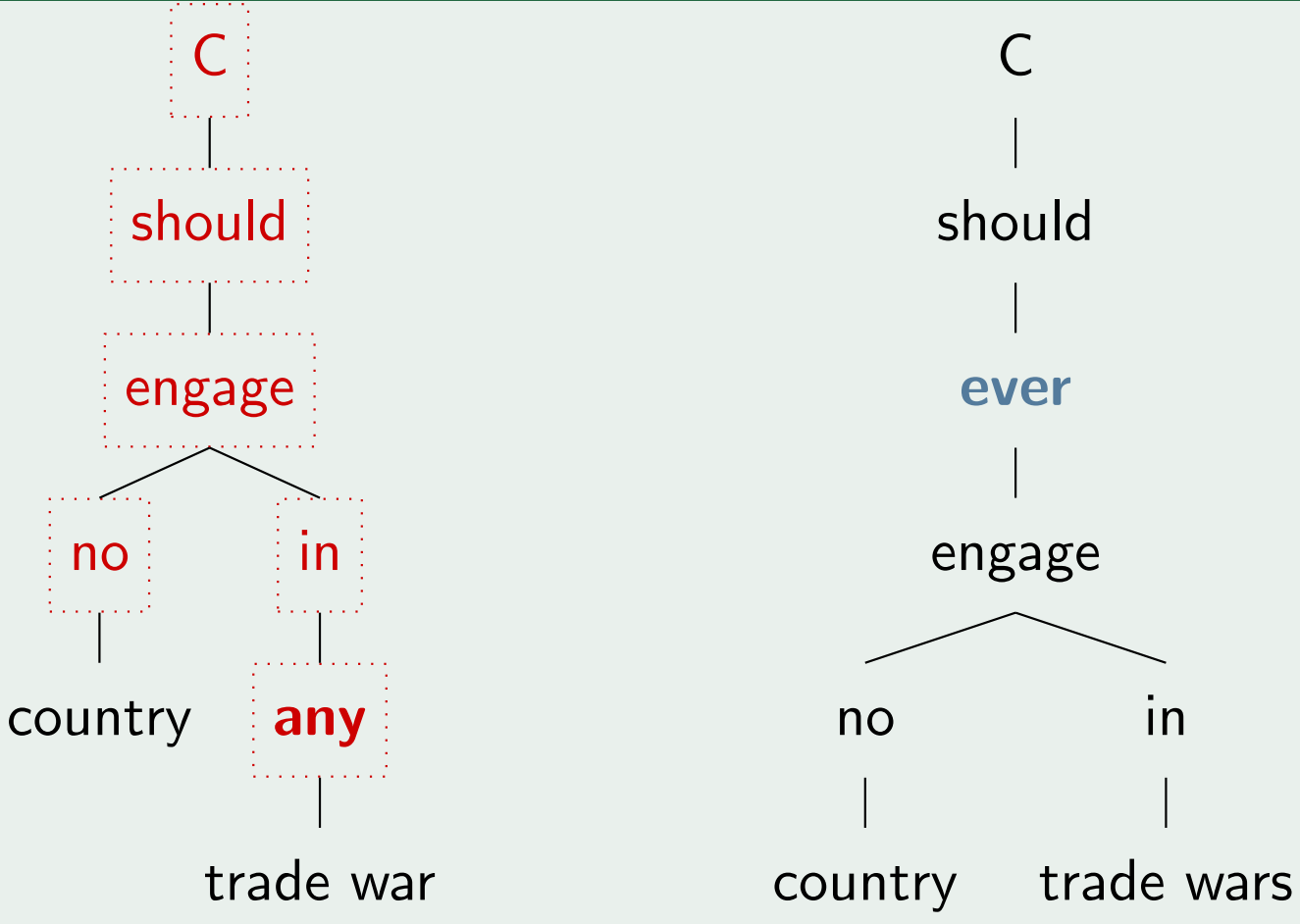
Example: NPI-licensing upward and downward



HSTAs for syntactic constraints

- HSTAs can handle almost everything in syntax.
  (Graf and De Santo 2019; Graf and Shafiei 2019)

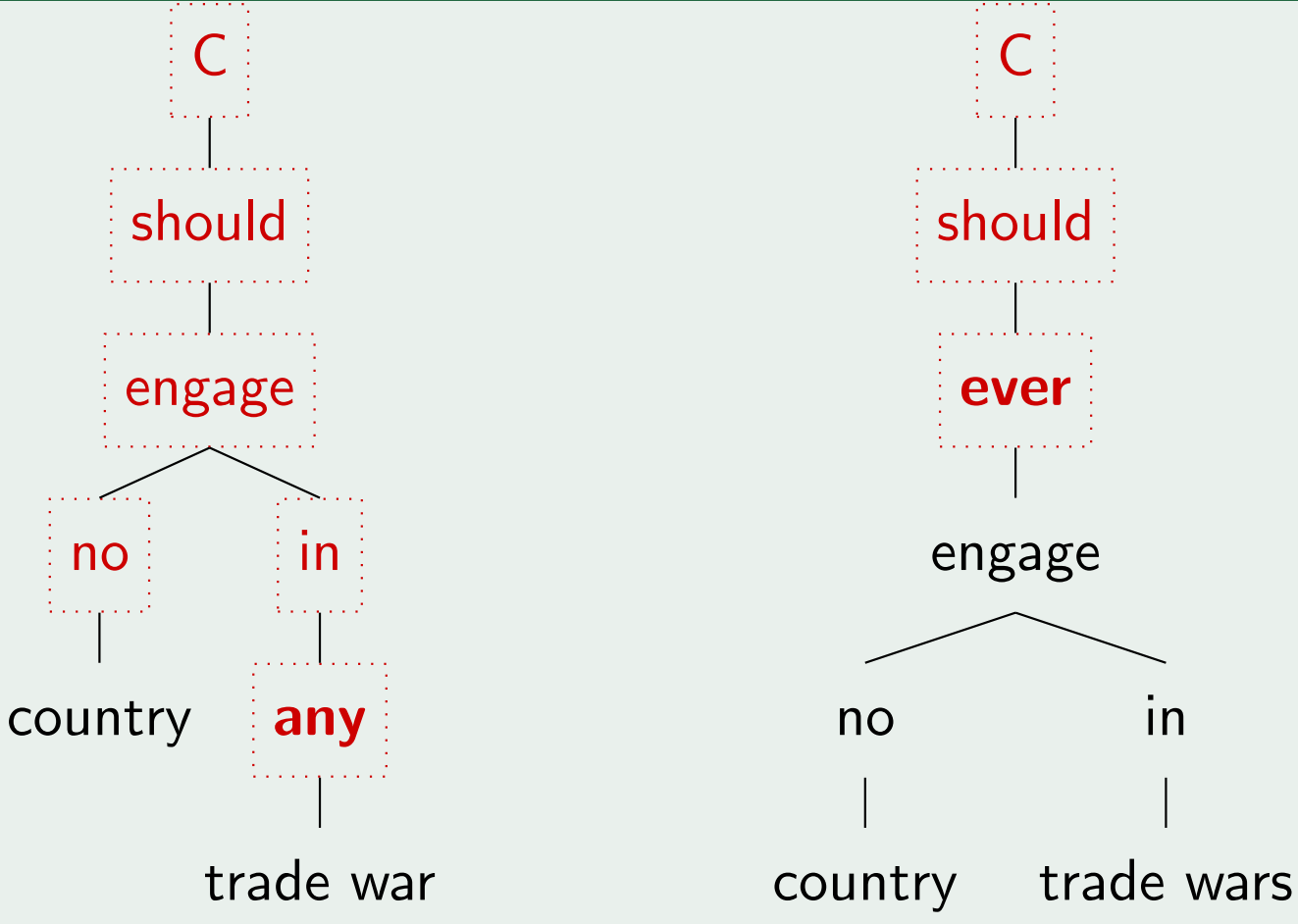
Example: NPI-licensing upward and downward



HSTAs for syntactic constraints

- HSTAs can handle almost everything in syntax.
  (Graf and De Santo 2019; Graf and Shafiei 2019)

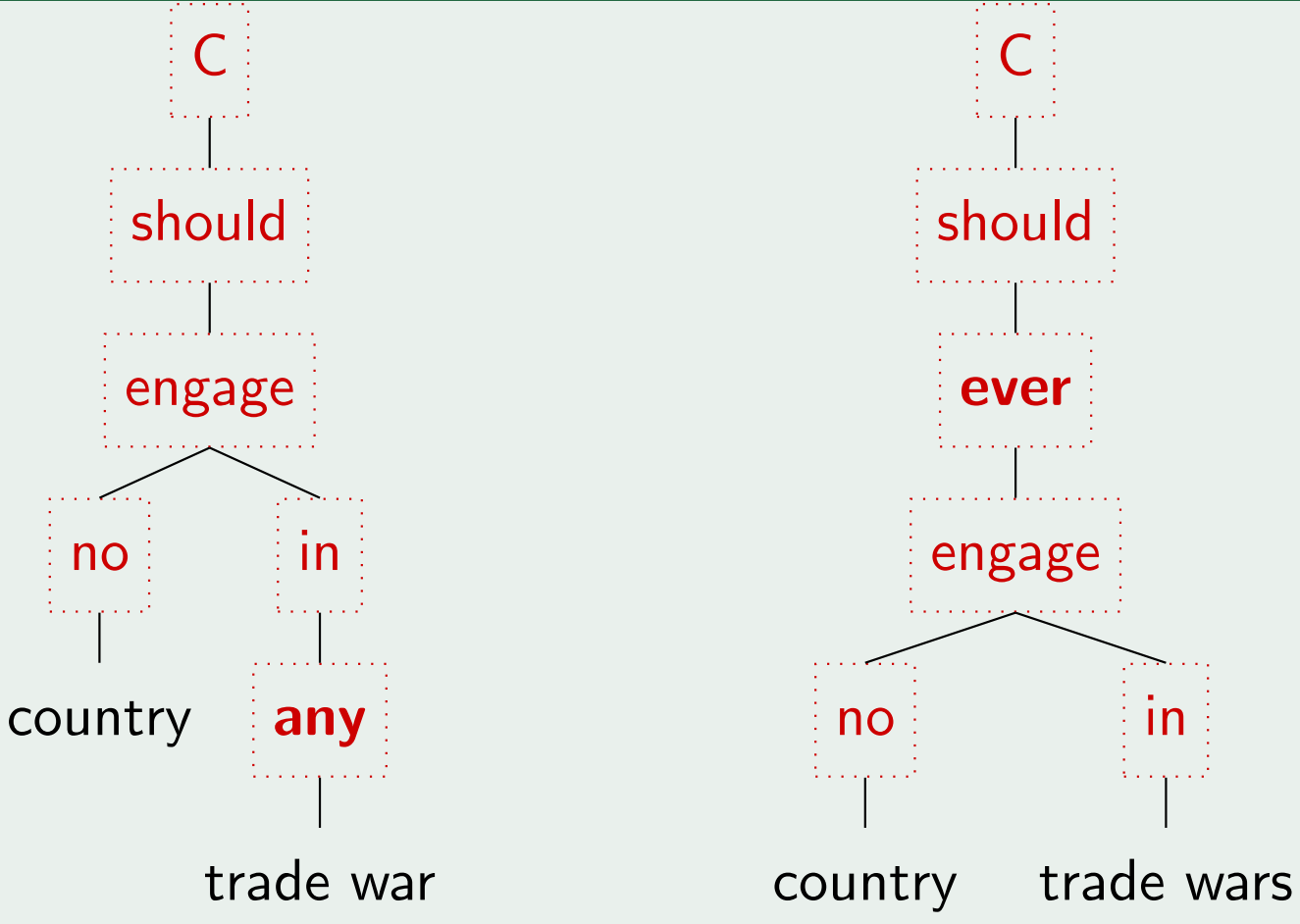
Example: NPI-licensing upward and downward



HSTAs for syntactic constraints

- HSTAs can handle almost everything in syntax.
  (Graf and De Santo 2019; Graf and Shafiei 2019)

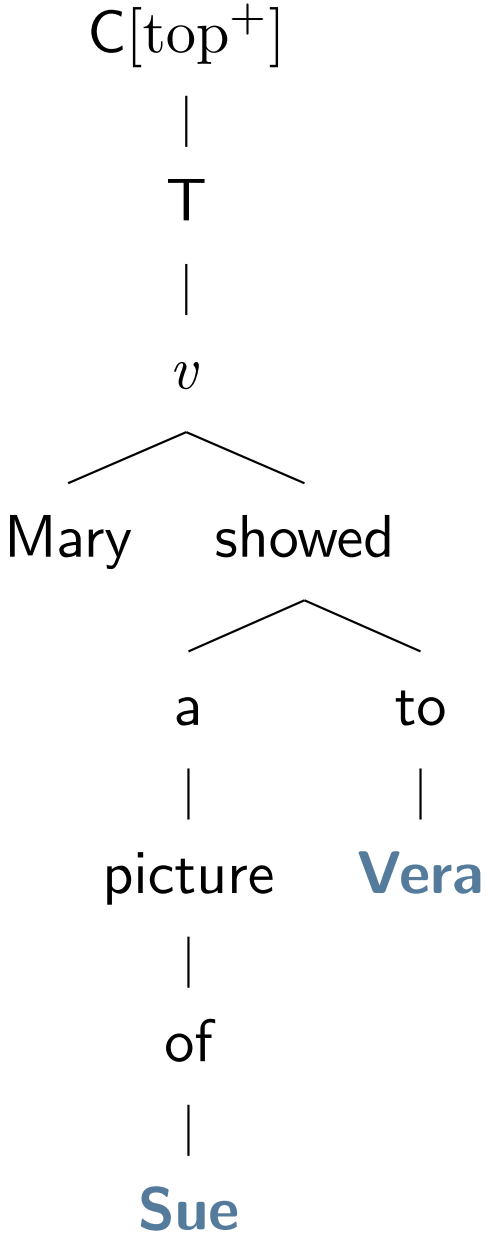
Example: NPI-licensing upward and downward



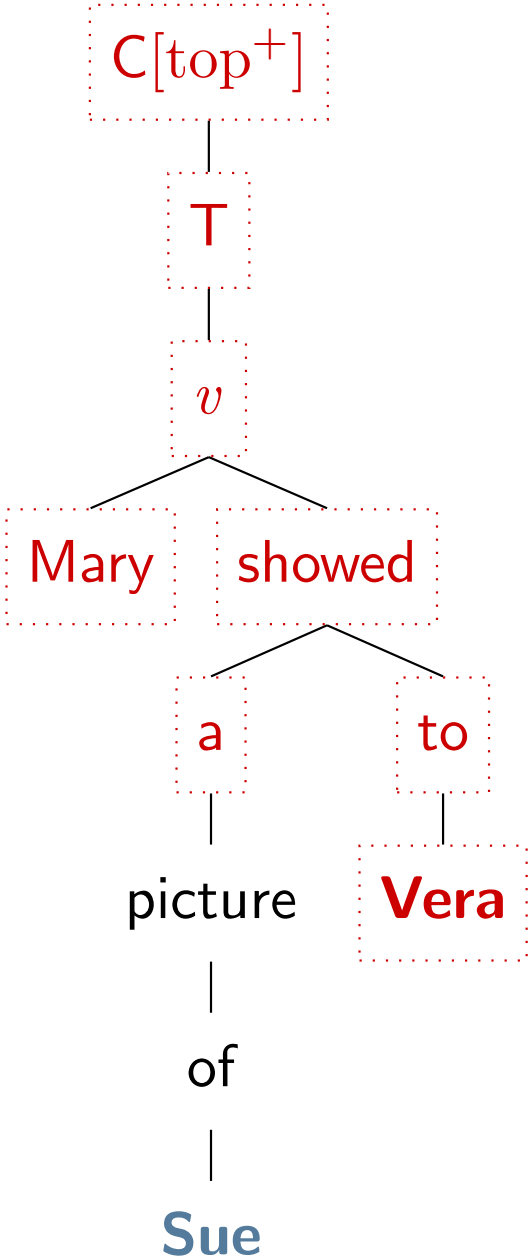
A (welcome) problem with movement

- In MGs, movement is triggered by two matching features:
  - **licensor** feature f^{+} at the target site
  - **licensee** feature f^{-} on the head of the mover
- A licensor feature must not remain unchecked.
- HSTAs fail to enforce this in some cases.

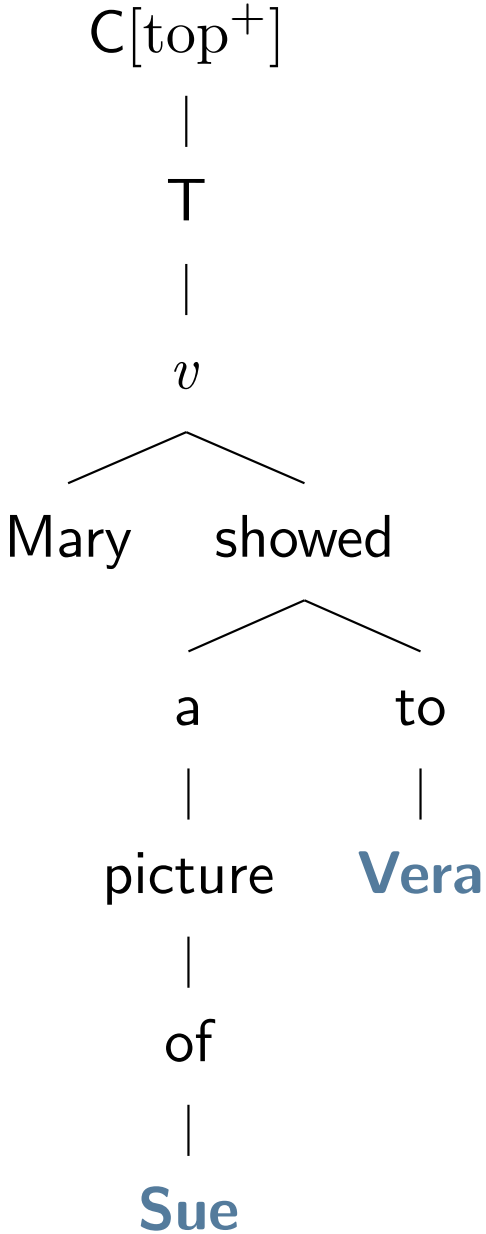
Example: HSTA does not notice missing top^-



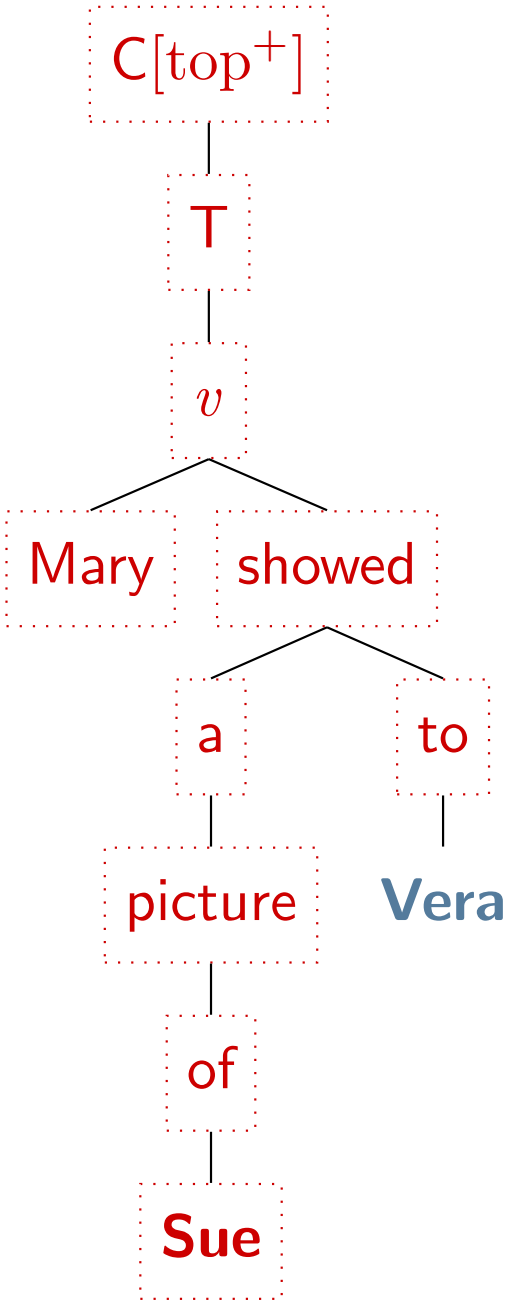
Example: HSTA does not notice missing top^-



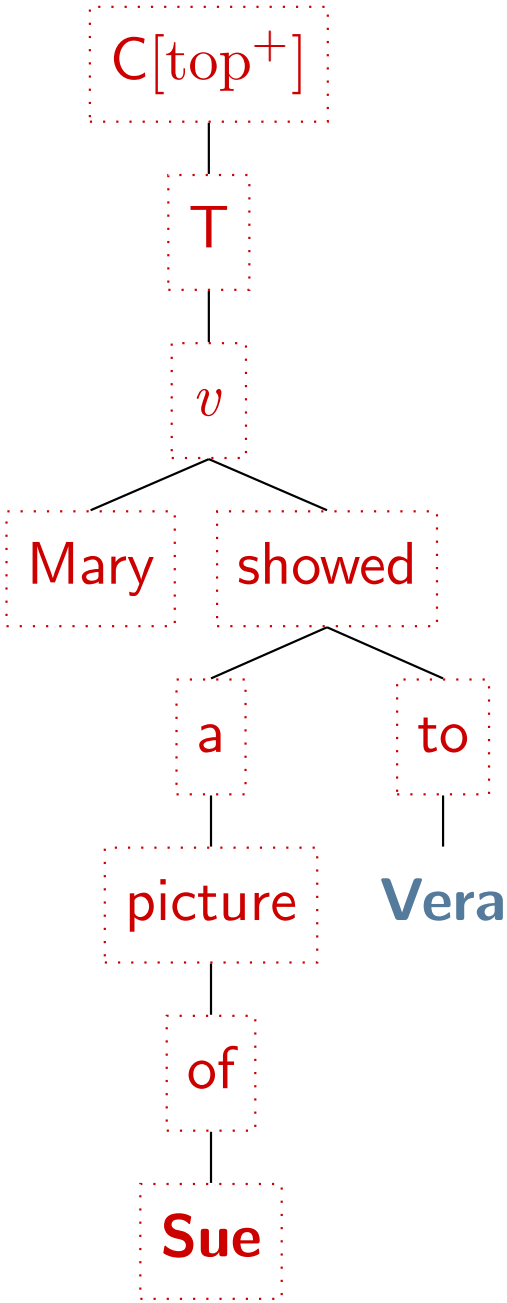
Example: HSTA does not notice missing top^-

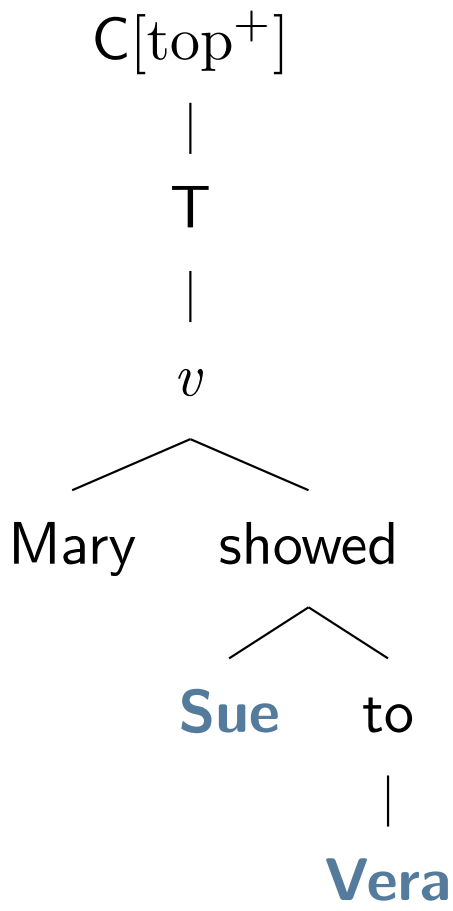


Example: HSTA does not notice missing top^-

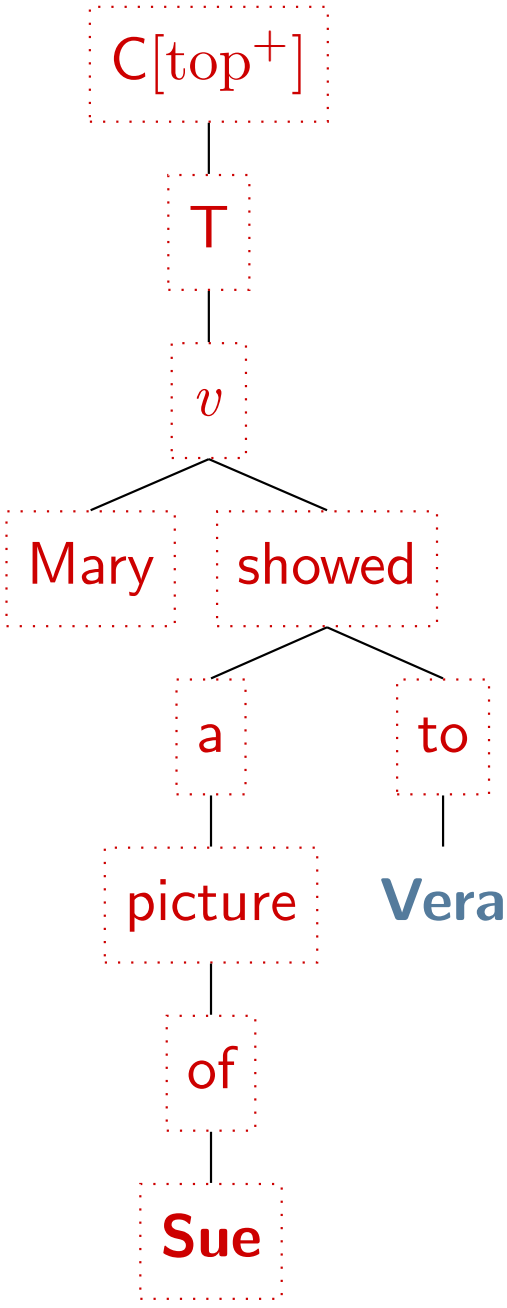


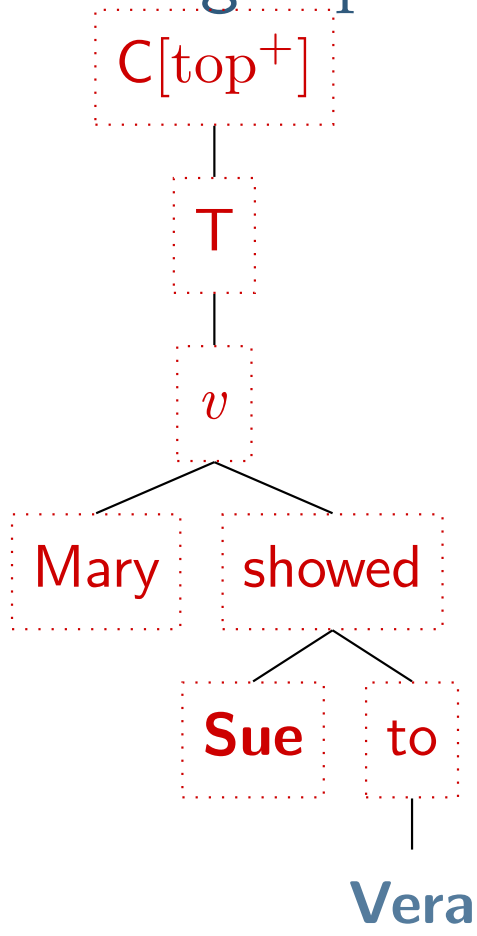
Example: HSTA does not notice missing top^-



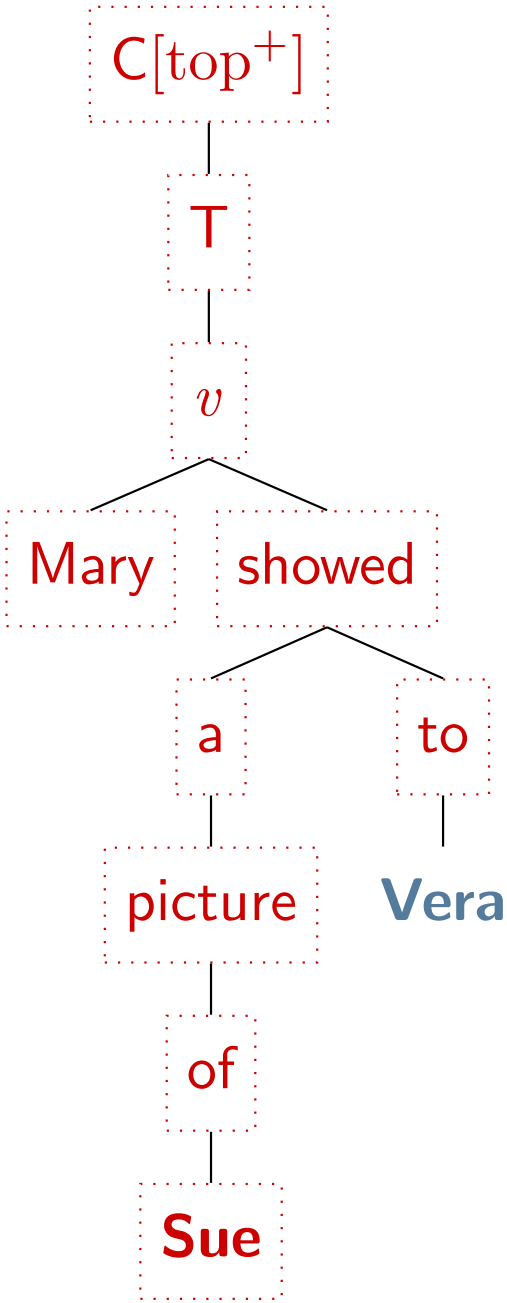


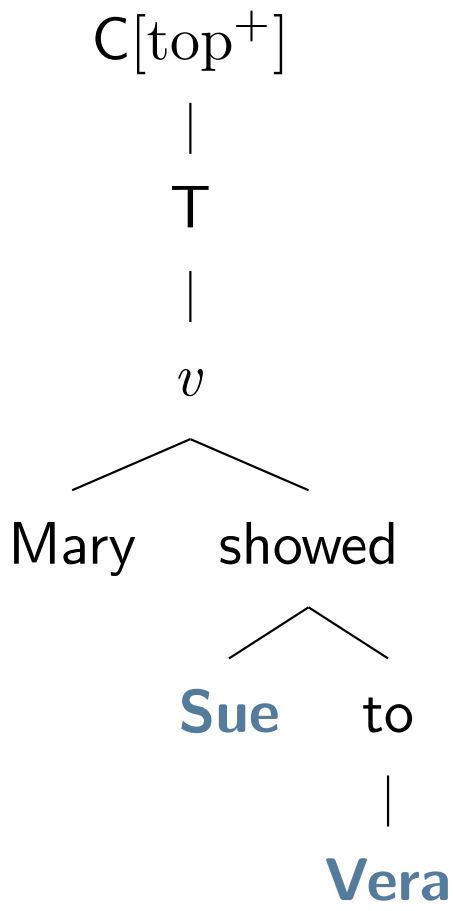
Example: HSTA does not notice missing top^-



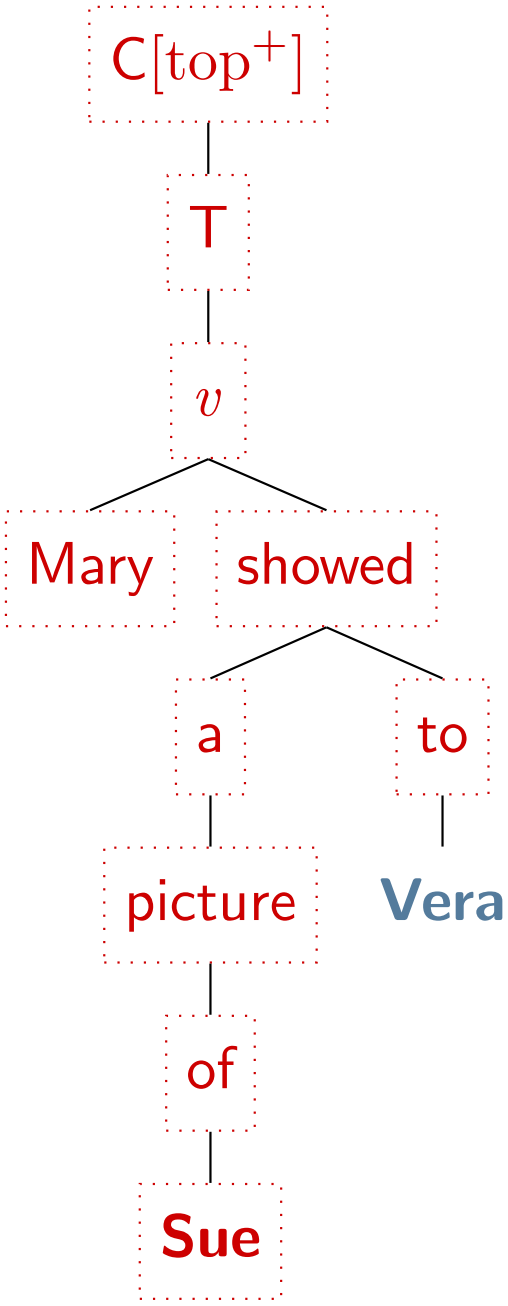


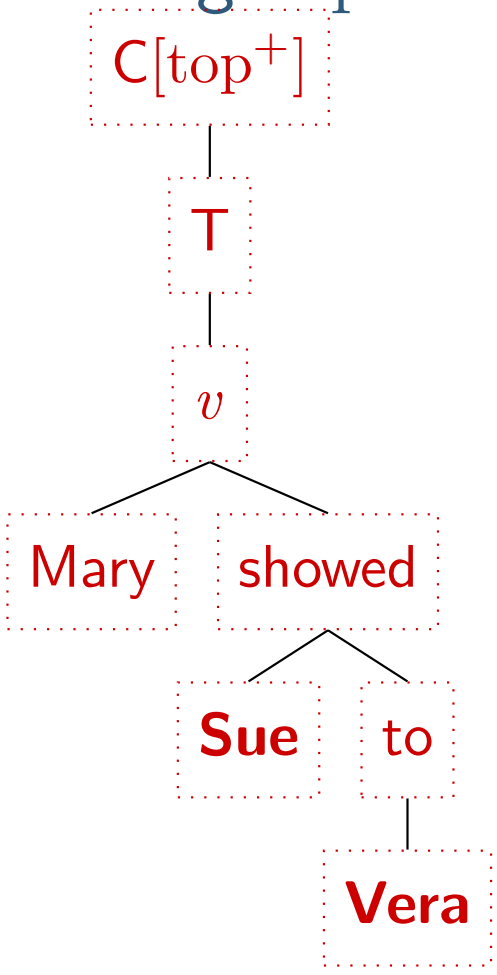
Example: HSTA does not notice missing top^-





Example: HSTA does not notice missing top^-





HSTA-movement displays island effects

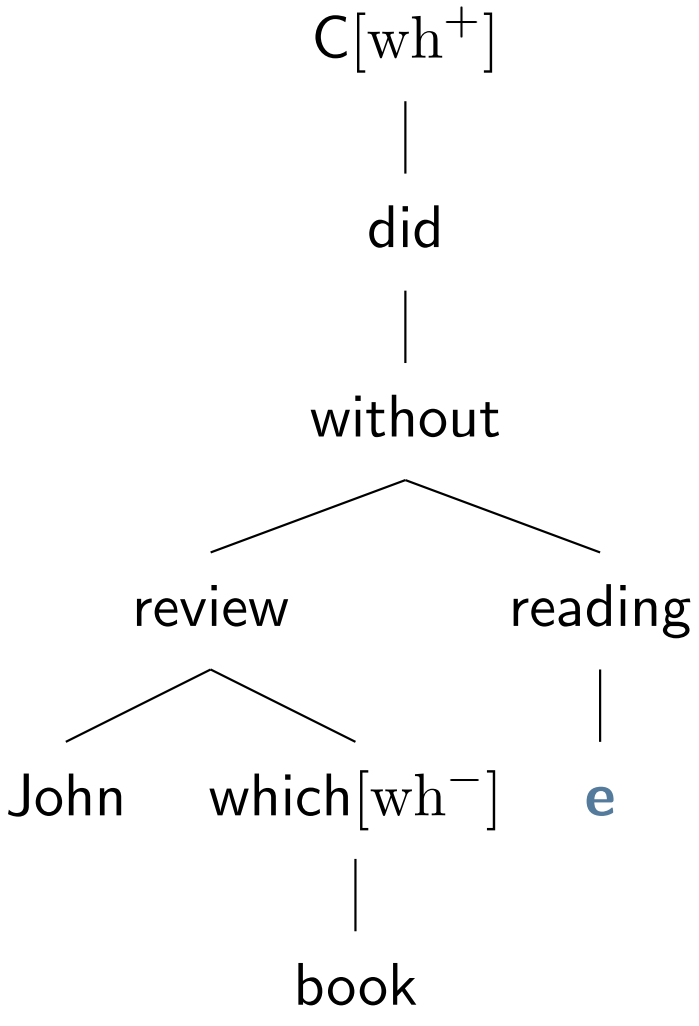
- **Observation**
  If a specifier/adjunct contains a mover, then no node outside the specifier/adjunct has that mover in its s-horizon.
- **Corollary**
  If we want HSTA-computable movement, syntax must obey:
  1. the specifier island constraint
  2. the adjunct island constraint
- Parsing considerations **derive island constraints**.

Interim summary

- **Assumption 1**
  Syntax should allow for efficient top-down parsing.
- **Assumption 2**
  efficient $\equiv$ finite-state & deterministic
- **Definition**
  HSTAs are deterministic top-down finite-state tree automata where state assignment hinges on head-argument relations.
- **Corollary**
  The computational limits of HSTAs derive island constraints.
  These are still syntactic constraints, not processing effects!

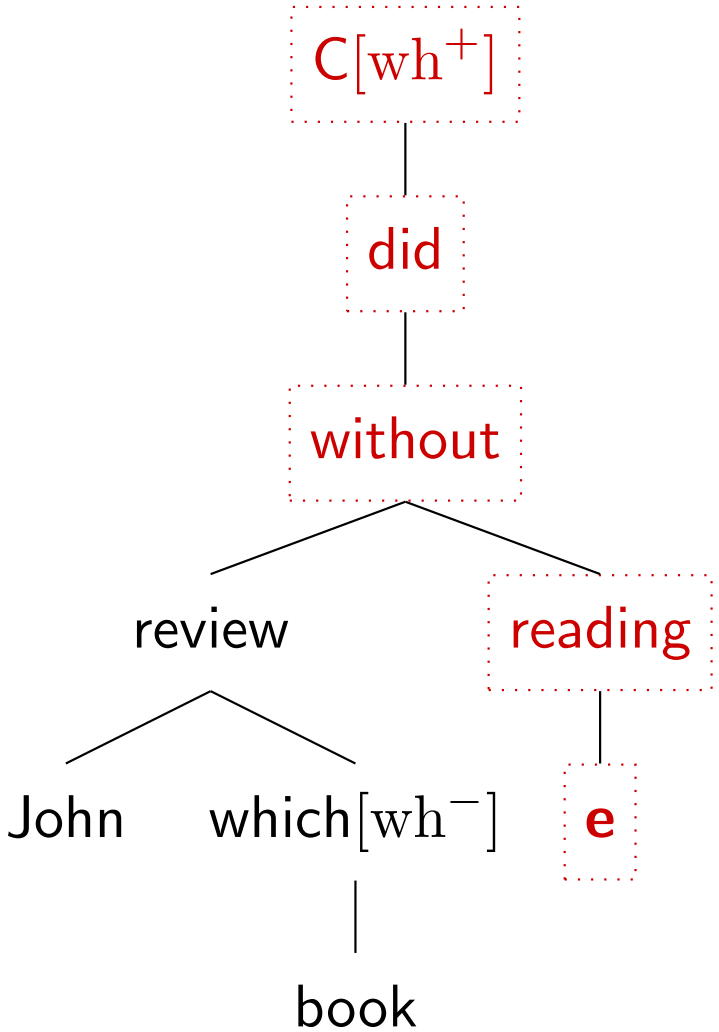
HSTAs still allow for parasitic gaps

- HSTAs can still verify PG licensing.
- If a PG's event horizon contains some f^{+} above the adjunct head, but no matching f^{-}, then the PG occurs along a movement path.
- **But:** not all movers license PGs
  $\Rightarrow$ must be inferrable from f^{+}



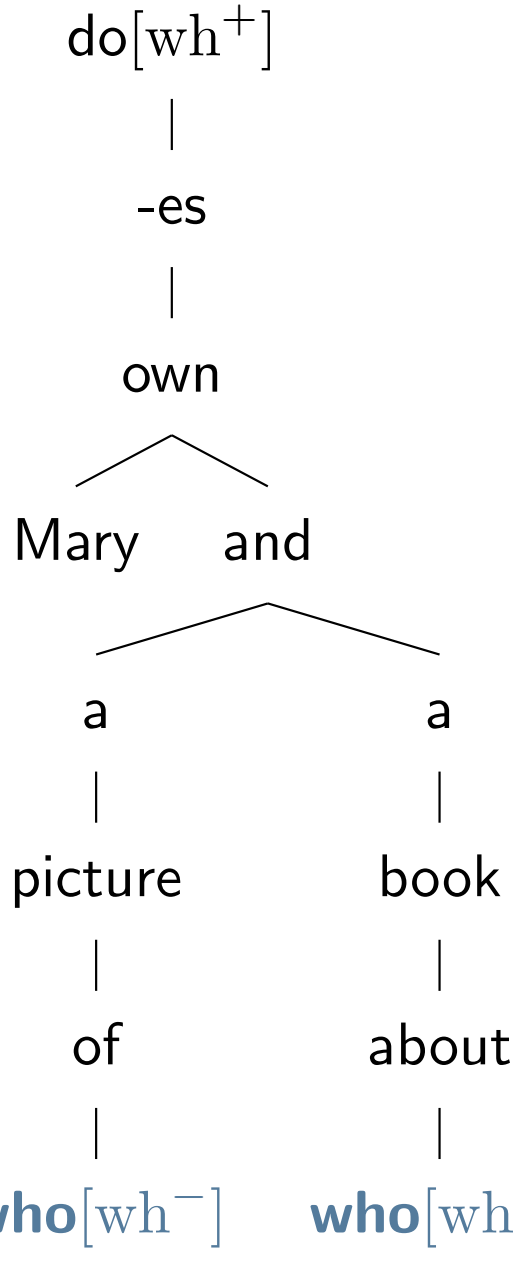
HSTAs still allow for parasitic gaps

- HSTAs can still verify PG licensing.
- If a PG's event horizon contains some f^{+} above the adjunct head, but no matching f^{-}, then the PG occurs along a movement path.
- **But:** not all movers license PGs $\Rightarrow$ must be inferrable from f^{+}



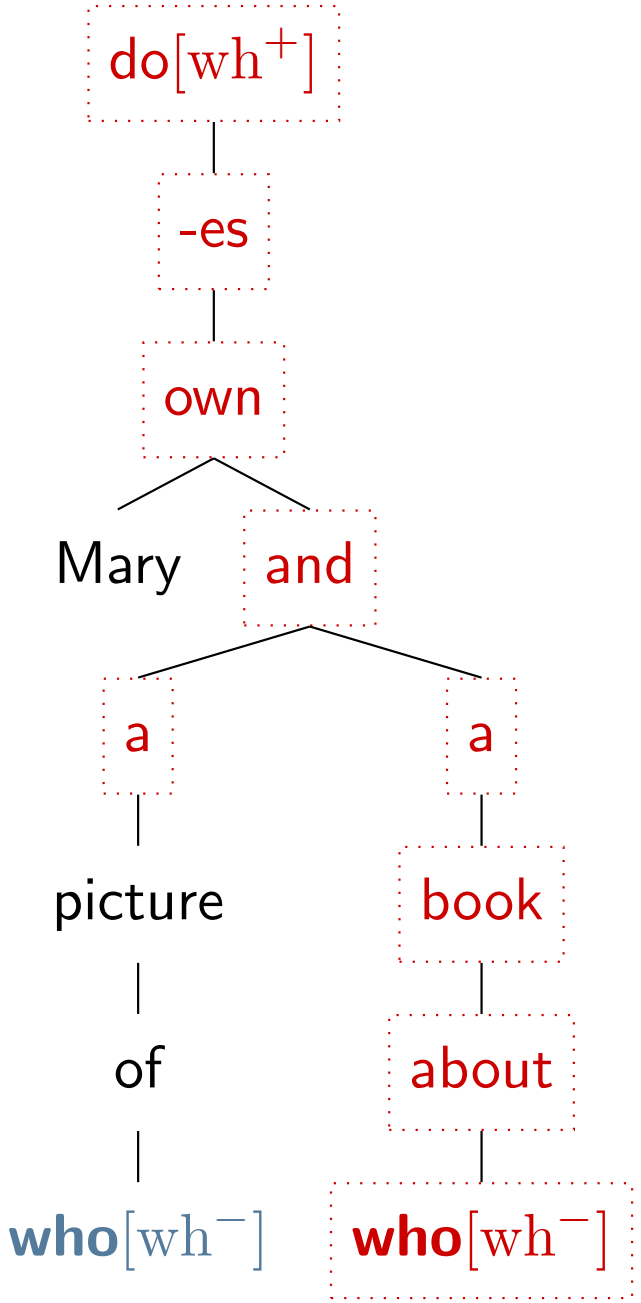
HSTAs allow for across-the-board extraction

- ATB-movement is still possible.
- Since each conjunct has to contain a mover of its own, each conjunct can be considered in isolation.



HSTAs allow for across-the-board extraction

- ATB-movement is still possible.
- Since each conjunct has to contain a mover of its own, each conjunct can be considered in isolation.



Conclusion

1. **Syntax informs sentence processing**
   A simple linking hypothesis converts syntactic analyses into processing claims.
2. **Parsing informs syntax**
   If syntax must be top-down parser-friendly, syntactic constraints must be limited:
   1. island constraints on movement
   2. primacy of c-command
3. **Open ends**
   - Truswell sentences, left-branch extraction in Japanese, Scandinavian RCs, ...
   - Left-corner parsing

Thanks

The work on HSTAs was supported by the National Science Foundation under Grant No. BCS-1845344.

References I

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

De Santo, Aniello. 2020. *Structure and memory: A computational model of storage, gradience, and priming*. Doctoral Dissertation, Stony Brook University.

Gibson, Edward. 2000. The dependency locality theory: A distance-based theory of linguistic complexity. In *Image, language, brain*, ed. Y. Miyashita, Alec Marantz, and W. O'Neil, 95–126. Cambridge, MA: MIT Press.

Gibson, Edward, and H.-H. Iris Wu. 2013. Processing Chinese relative clauses in context. *Language and Cognitive Processes* 28:125–155.

Gordon, Peter C., Randall Hendrick, Marcus Johnson, and Yoonhyoung Lee. 2006. Similarity-based interference during language comprehension: Evidence from eye tracking during reading. *Journal of Experimental Psychology: Learning, Memory, and Cognition* 32:1304.

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

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

References II

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.

Kobele, Gregory M., Sabrina Gerth, and John T. Hale. 2013. Memory resource allocation in top-down Minimalist parsing. In *Formal Grammar: 17th and 18th International Conferences, FG 2012, Opole, Poland, August 2012, Revised Selected Papers, FG 2013, Düsseldorf, Germany, August 2013*, ed. Glyn Morrill and Mark-Jan Nederhof, 32–51. Berlin, Heidelberg: Springer. URL `https://doi.org/10.1007/978-3-642-39998-5_3`.

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

Lee, So Young. 2019. A Minimalist parsing account of attachment ambiguity in English and Korean. *Journal of Cognitive Science* 3:291–329.

Liu, Lei. 2018. Minimalist parsing of heavy NP shift. In *Proceedings of the 32nd Pacific Asia Conference on Language, Information and Computation (PACLIC 32)*. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/Y18-1047`.

References III

Mecklinger, Axel, Herbert Schriefers, Karsten Steinhauer, and Angela D. Friederici. 1995. Processing relative clauses varying on syntactic and semantic dimensions: An analysis with event-related potentials. *Memory & Cognition* 23:477–494.

Miyamoto, Edson T., and Michiko Nakamura. 2013. Unmet expectations in the comprehension of relative clauses in Japanese. In *Proceedings of the 35th Annual Meeting of the Cognitive Science Society*, 3074–3079.

Morawietz, Frank, and Thomas Cornell. 1999. The MSO logic automaton connection in linguistics. In *Logical Aspects of Computational Linguistics: Second International Conference*, ed. Alain Lecomte, François Lamarche, and Guy Perrier, volume 1582, 112–131. Berlin, Heidelberg: Springer.

Pasternak, Robert, and Thomas Graf. 2020. Cyclic scope and processing difficulty in a Minimalist parser. *Glossa* To appear.

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

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

Zhang, Chong. 2017. *Stacked relatives: Their structure, processing, and computation*. Doctoral Dissertation, Stony Brook University.