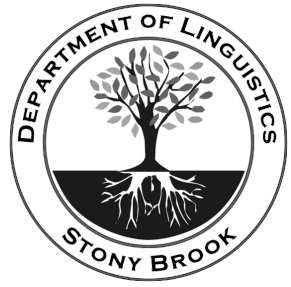


Sensing Tree Automata as a Model of Syntactic Dependencies

Thomas Graf & Aniello De Santo

Stony Brook University
aniello.desanto@stonybrook.edu
https://aniellodesanto.github.io/

MOL
Toronto, July 18-19, 2019

The Talk in One Minute

The research program

- a tight upper bound to the complexity of natural language dependencies?

In this talk

- Sensing tree automata as a uniform upper bound
- MG dependency trees

Spoilers

- A (linguistically) natural perspective!
- Empirically attested restrictions on movement
- Head-argument relations
- C-command and licensing conditions

Outline

Computational Theories of Language

The subregular program

Can we provide tight complexity characterizations for natural language?

- Particularly successful in phonology
  (Heinz et al. 2011; Chandlee 2014; Jardine 2016; McMullin 2016; Graf 2017; Graf and Mayer 2018)
- Some results for syntax
  - regular tree languages
    (Michaelis 2004; Kobele et al. 2007; Graf 2012)
  - subregular operations? (Graf 2012, 2018)
  - subregular dependencies? (Vu 2018; Vu et al. 2019)
  - subregular constraints? (Shafiei and Graf 2019)

Computational Theories of Language

The subregular program

Can we provide tight complexity characterizations for natural language?

- Particularly successful in phonology
  (Heinz et al. 2011; Chandlee 2014; Jardine 2016; McMullin 2016; Graf 2017; Graf and Mayer 2018)
- Some results for syntax
  - regular tree languages
    (Michaelis 2004; Kobele et al. 2007; Graf 2012)
  - subregular operations? (Graf 2012, 2018)
  - subregular dependencies? (Vu 2018; Vu et al. 2019)
  - subregular constraints? (Shafiei and Graf 2019)

Computational Theories of Language

The subregular program

Can we provide tight complexity characterizations for natural language?

- Particularly successful in phonology
  (Heinz et al. 2011; Chandlee 2014; Jardine 2016; McMullin 2016; Graf 2017; Graf and Mayer 2018)
- Some results for syntax
  - regular tree languages
    (Michaelis 2004; Kobele et al. 2007; Graf 2012)
  - subregular operations? (Graf 2012, 2018)
  - subregular dependencies? (Vu 2018; Vu et al. 2019)
  - subregular constraints? (Shafiei and Graf 2019)

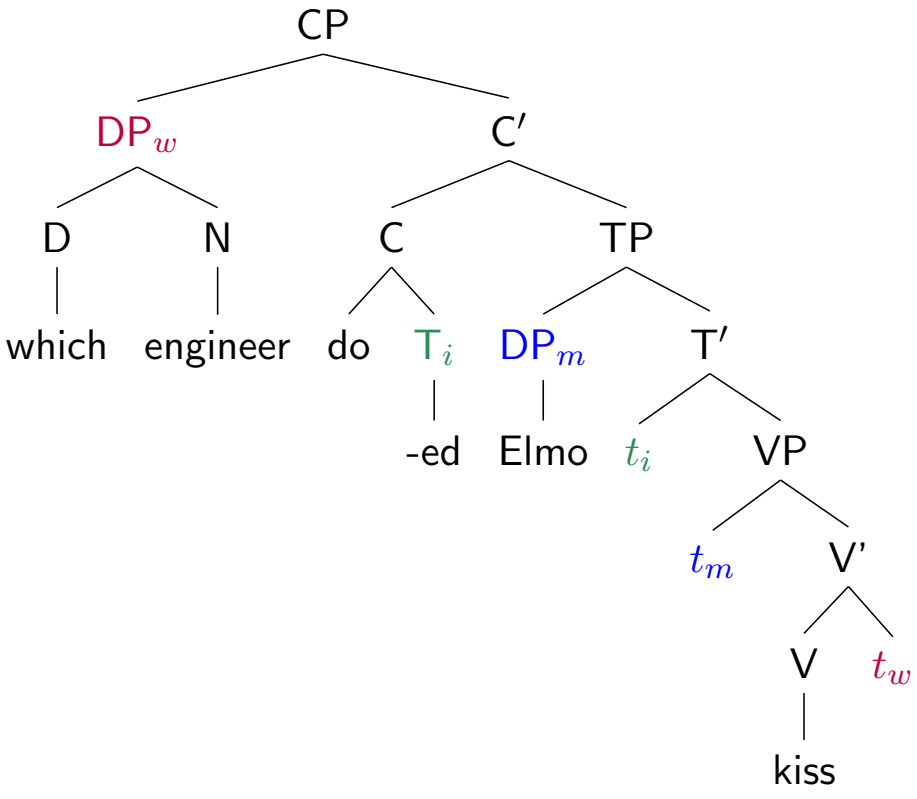
Can we gain a unified perspective for syntax?

Syntax?

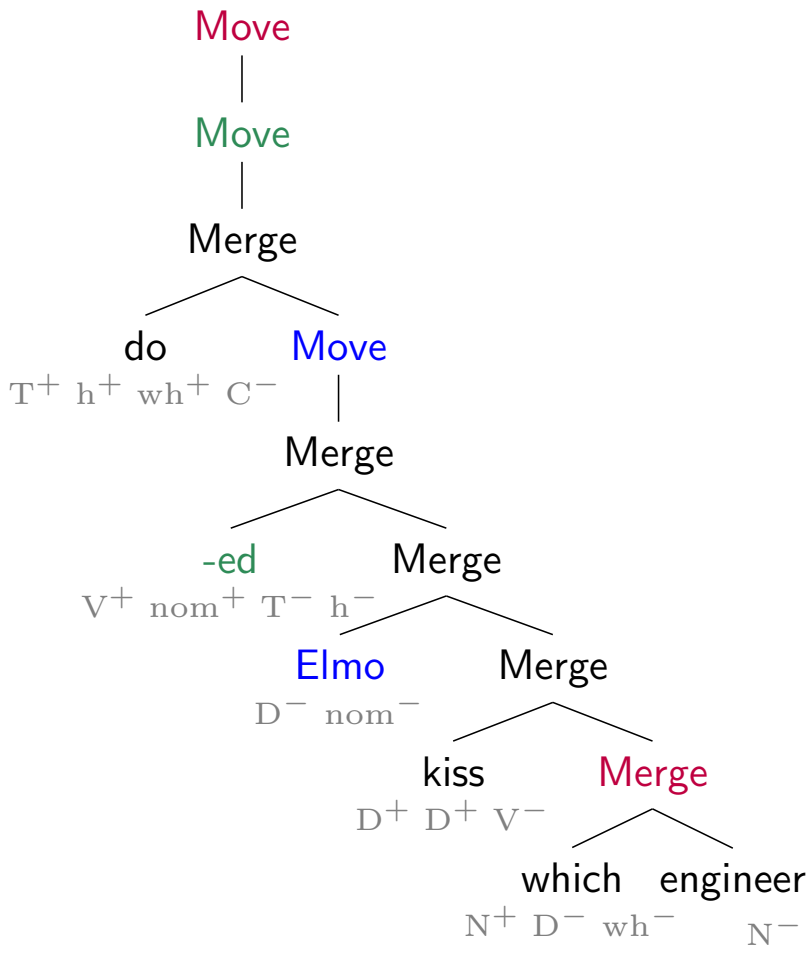
We need a formal model of syntactic structures.

- Minimalist grammars (MGs) are a formalization of Minimalist syntax. (Stabler 1997, 2011)
- Operations:
  - **Merge**
    category feature N^-, D^-, ...
    selector feature N^+, D^+, ...
  - **Move**
    licensee feature wh^-, nom^-, ...
    licensor feature wh^+, nom^+, ...
- Adopt Chomsky-Borer hypothesis:
  Grammar is just a finite list of feature-annotated lexical items
- The set of derivation trees is a regular tree language.
  (Michaelis 2004; Kobele et al. 2007; Graf 2012)

MG Syntax: Derivation Trees

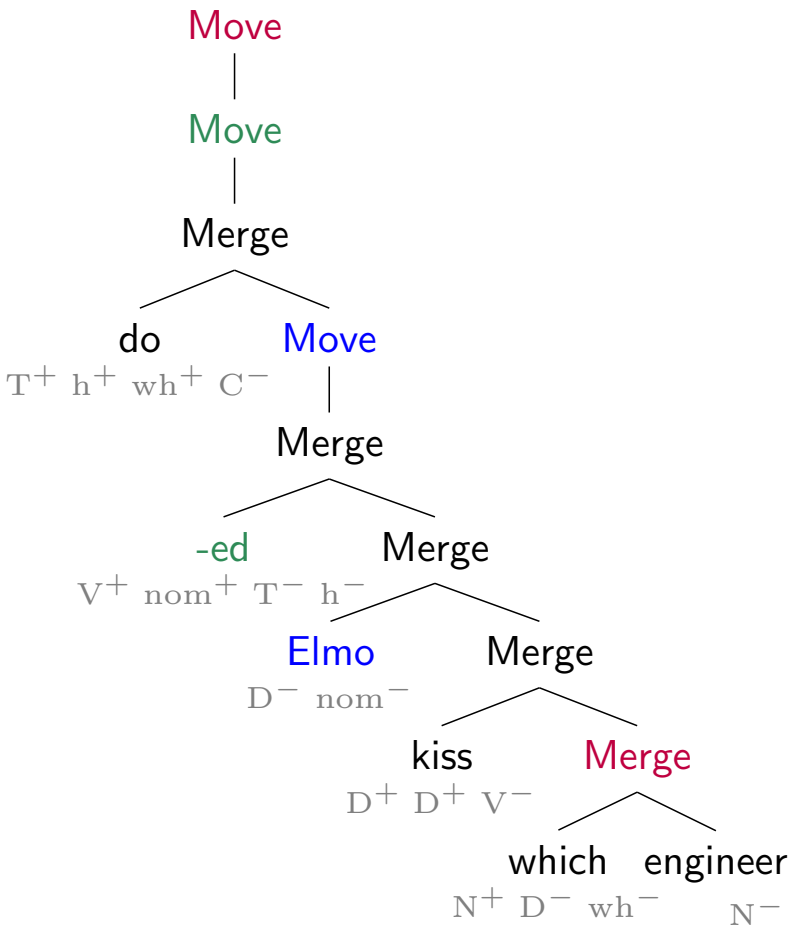


Phrase Structure Tree

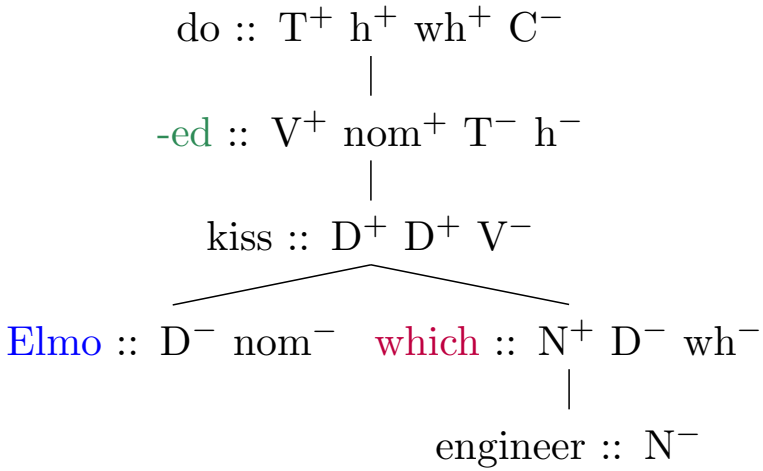


Derivation Tree

MG Syntax: Dependency Trees



Derivation Tree



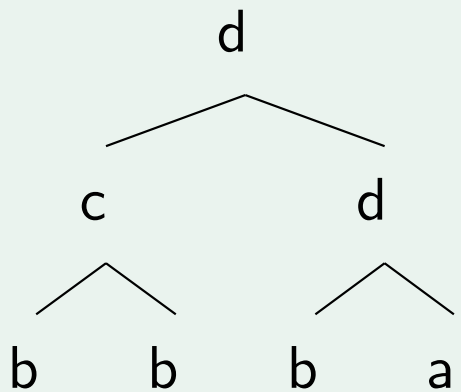
Dependency Tree

Sensing Tree Automata (STA)

Sensing Tree Automaton (Martens 2006)

Deterministic top-down tree automaton with finite look-ahead of 1.

Example: a *c-commanded* by c



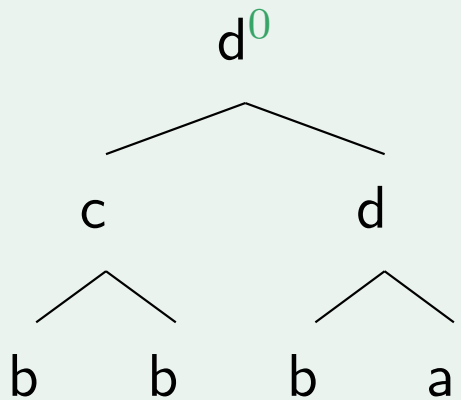
- $0(b) \to b$; $1(b) \to b$
- $1(a) \to a$

Sensing Tree Automata (STA)

Sensing Tree Automaton (Martens 2006)

Deterministic top-down tree automaton with finite look-ahead of 1.

Example: a *c-commanded* by c



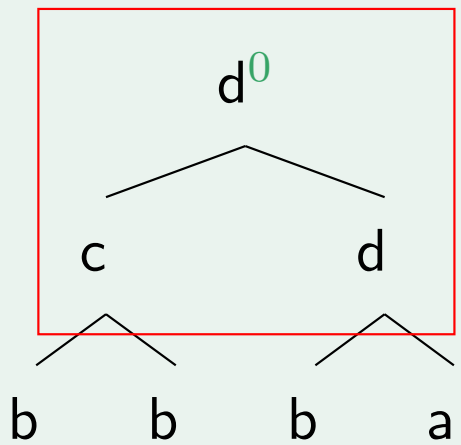
- $0(b) \to b$; $1(b) \to b$
- $1(a) \to a$

Sensing Tree Automata (STA)

Sensing Tree Automaton (Martens 2006)

Deterministic top-down tree automaton with finite look-ahead of 1.

Example: a *c-commanded* by c



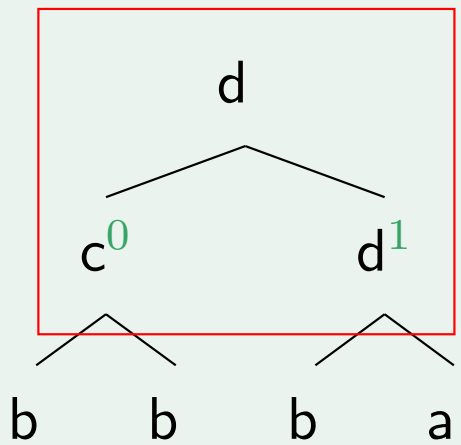
- $0(b) \to b$; $1(b) \to b$
- $1(a) \to a$

Sensing Tree Automata (STA)

Sensing Tree Automaton (Martens 2006)

Deterministic top-down tree automaton with finite look-ahead of 1.

Example: a *c-commanded* by c



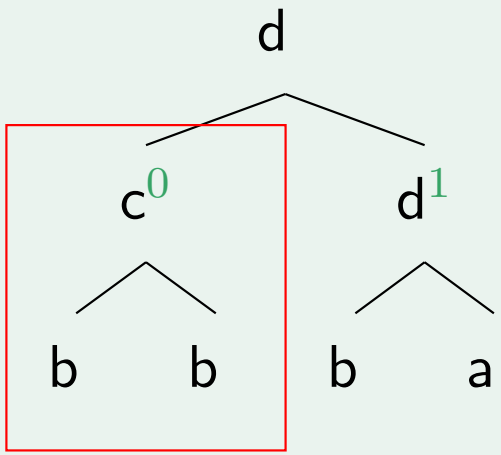
- $0(b) \to b$; $1(b) \to b$
- $1(a) \to a$

Sensing Tree Automata (STA)

Sensing Tree Automaton (Martens 2006)

Deterministic top-down tree automaton with finite look-ahead of 1.

Example: a *c-commanded* by c



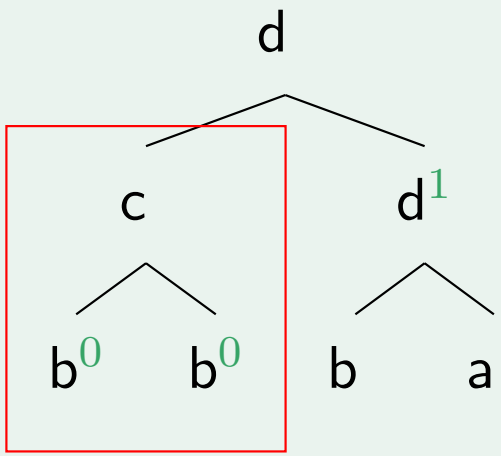
- $0(b) \to b$; $1(b) \to b$
- $1(a) \to a$

Sensing Tree Automata (STA)

Sensing Tree Automaton (Martens 2006)

Deterministic top-down tree automaton with finite look-ahead of 1.

Example: a *c-commanded* by c



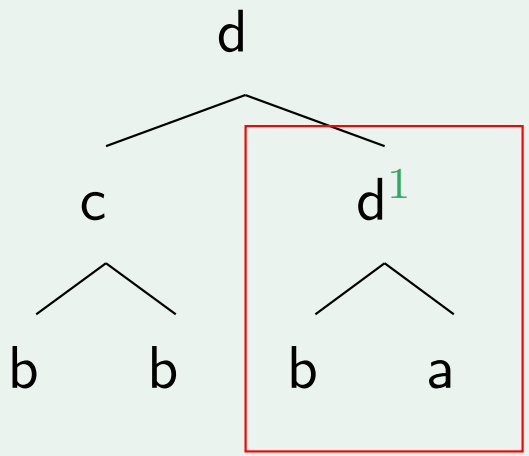
- $0(b) \to b$; $1(b) \to b$
- $1(a) \to a$

Sensing Tree Automata (STA)

Sensing Tree Automaton (Martens 2006)

Deterministic top-down tree automaton with finite look-ahead of 1.

Example: a *c-commanded* by c



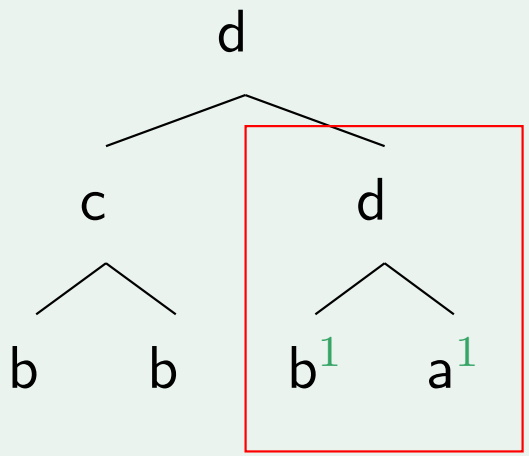
- $0(b) \to b$; $1(b) \to b$
- $1(a) \to a$

Sensing Tree Automata (STA)

Sensing Tree Automaton (Martens 2006)

Deterministic top-down tree automaton with finite look-ahead of 1.

Example: a *c-commanded* by c



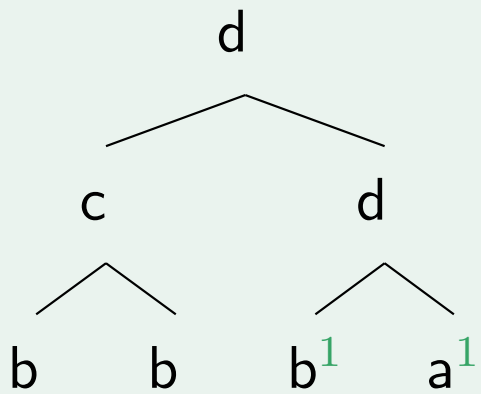
- $0(b) \to b$; $1(b) \to b$
- $1(a) \to a$

Sensing Tree Automata (STA)

Sensing Tree Automaton (Martens 2006)

Deterministic top-down tree automaton with finite look-ahead of 1.

Example: a *c-commanded* by c



- $0(b) \to b$; $1(b) \to b$
- $1(a) \to a$

Interim Summary

We are looking for a complexity upper bound for syntax...

- MG dependency trees (MDEP)
- STA

Upcoming

- MDEP[*merge*] $\subsetneq$ STA
- MDEP[*merge*,*move*] $\subsetneq$ STA iff we restrict *move*

Interim Summary

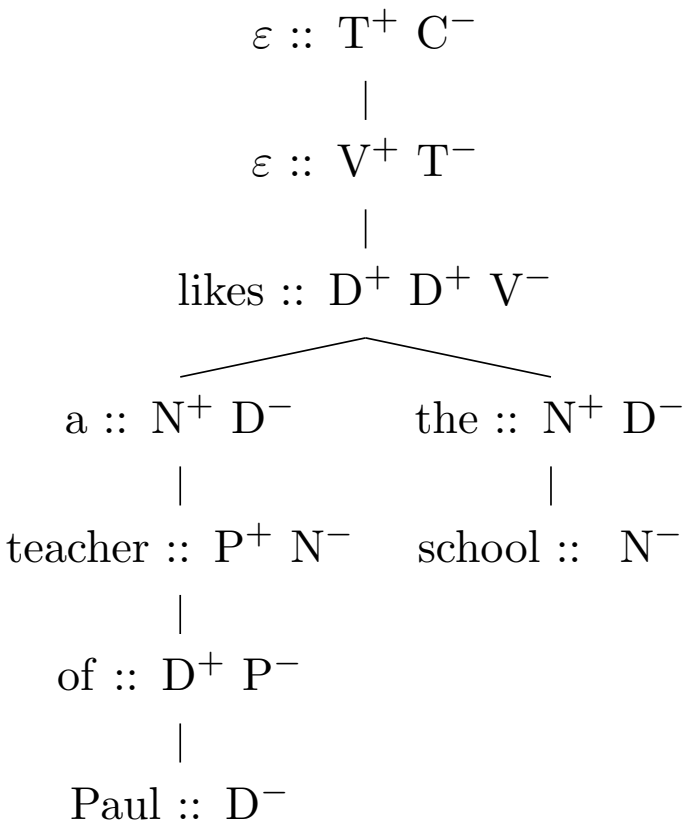
We are looking for a complexity upper bound for syntax...

- MG dependency trees (MDEP)
- STA

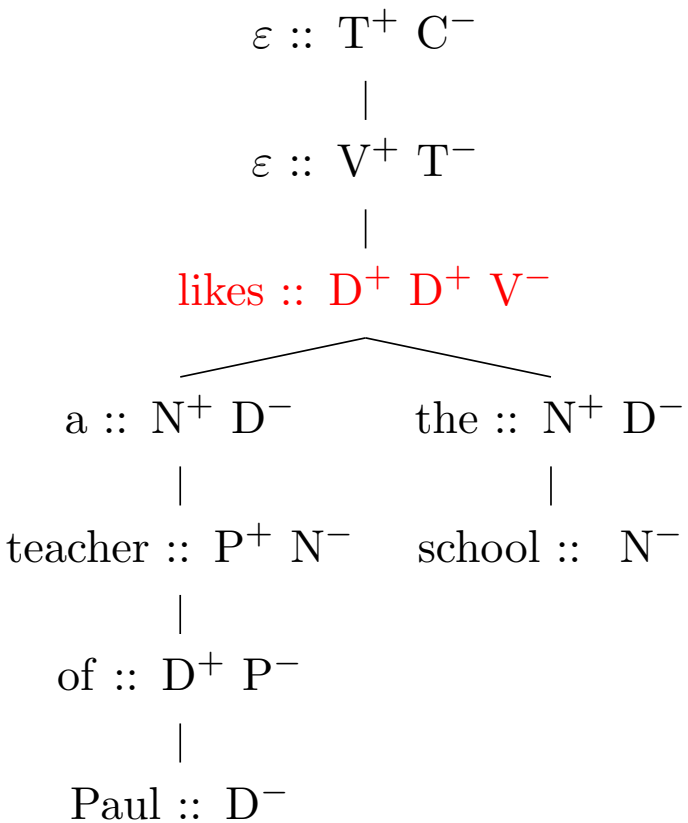
Upcoming

- MDEP[*merge*] $\subsetneq$ STA
- MDEP[*merge*,*move*] $\subsetneq$ STA iff we restrict *move*

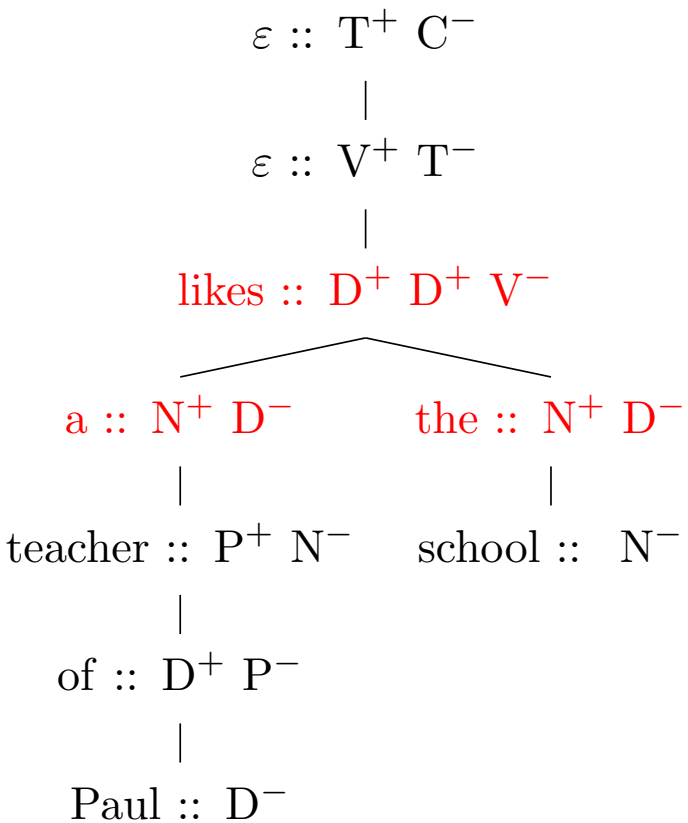
Dependency Trees, Merge, and Head-Argument Relations



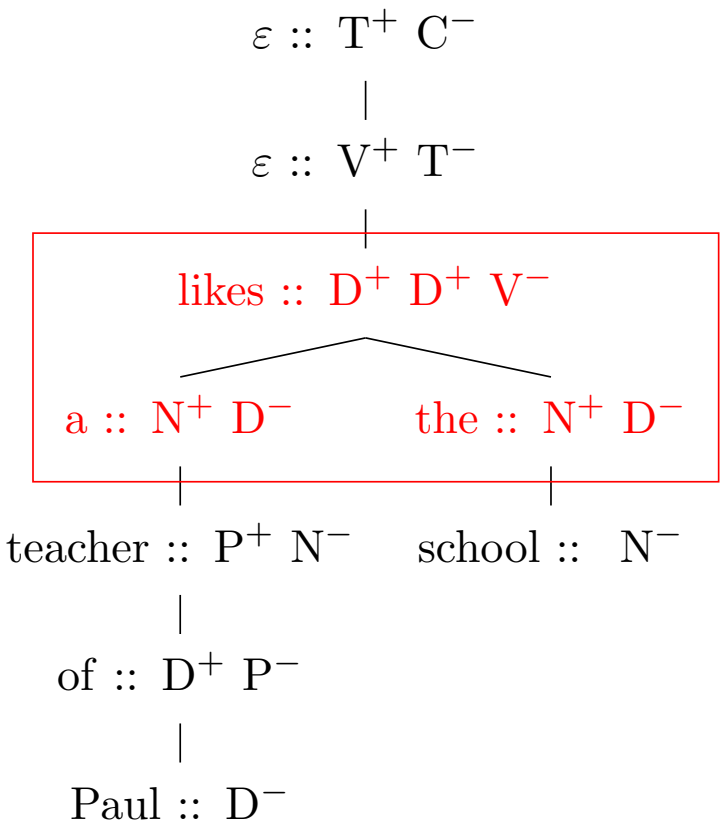
Dependency Trees, Merge, and Head-Argument Relations



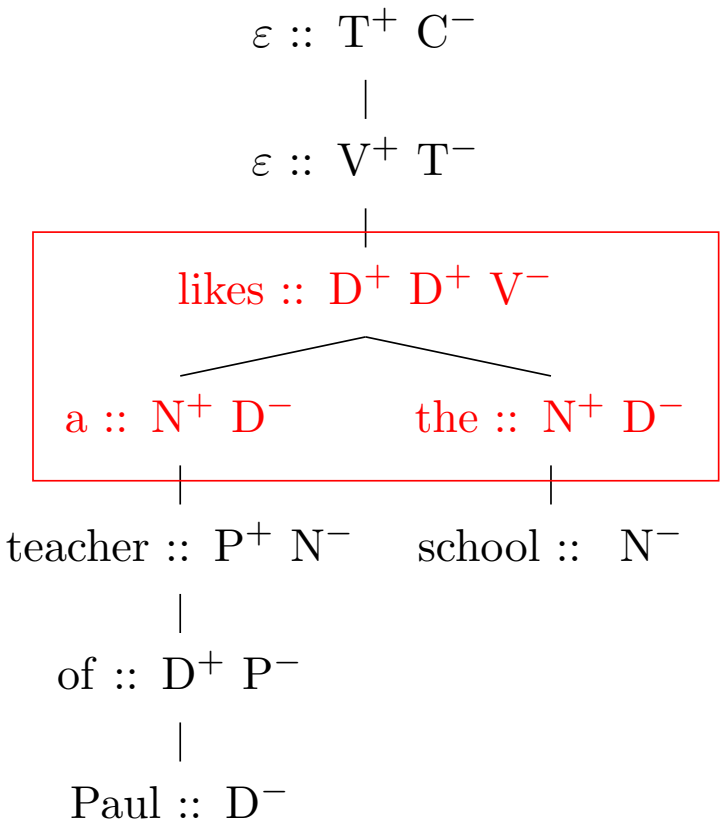
Dependency Trees, Merge, and Head-Argument Relations

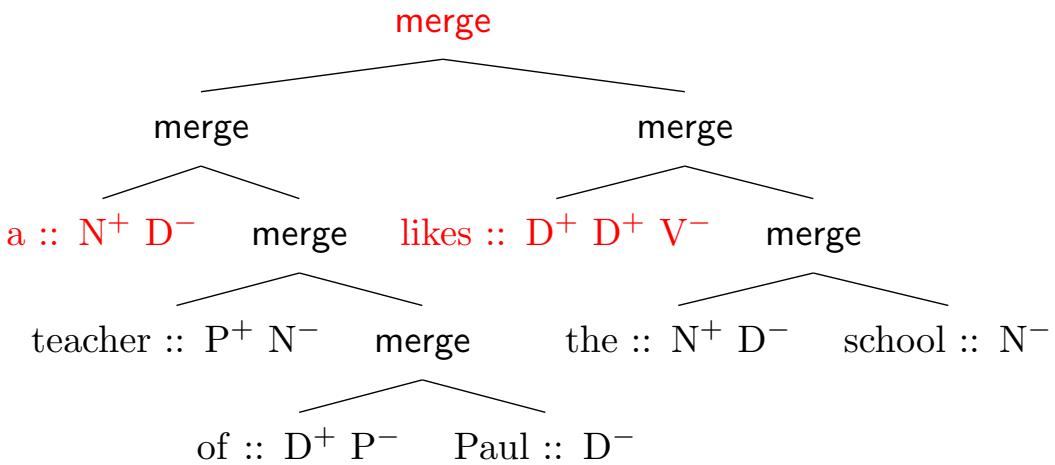


Dependency Trees, Merge, and Head-Argument Relations

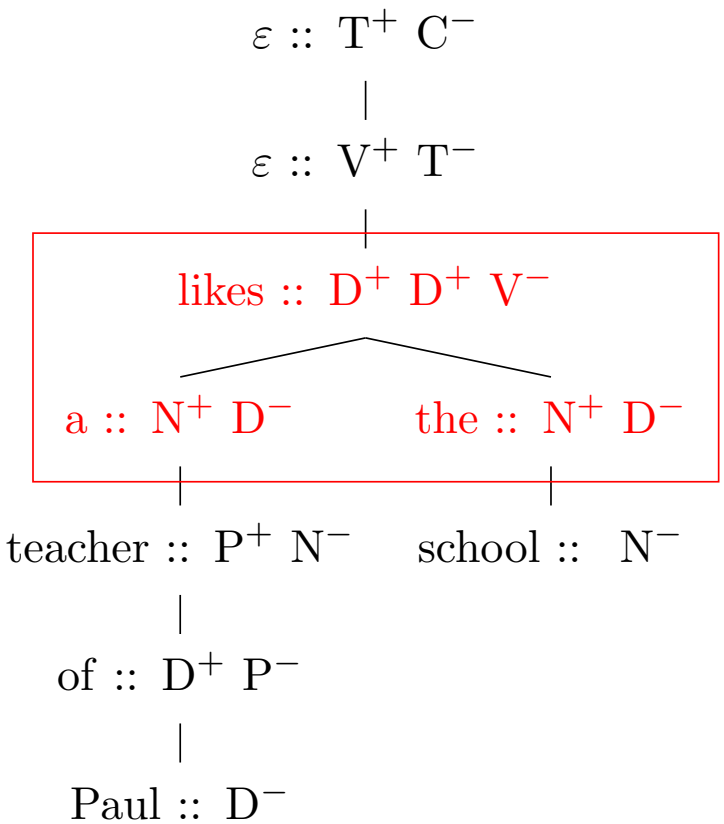


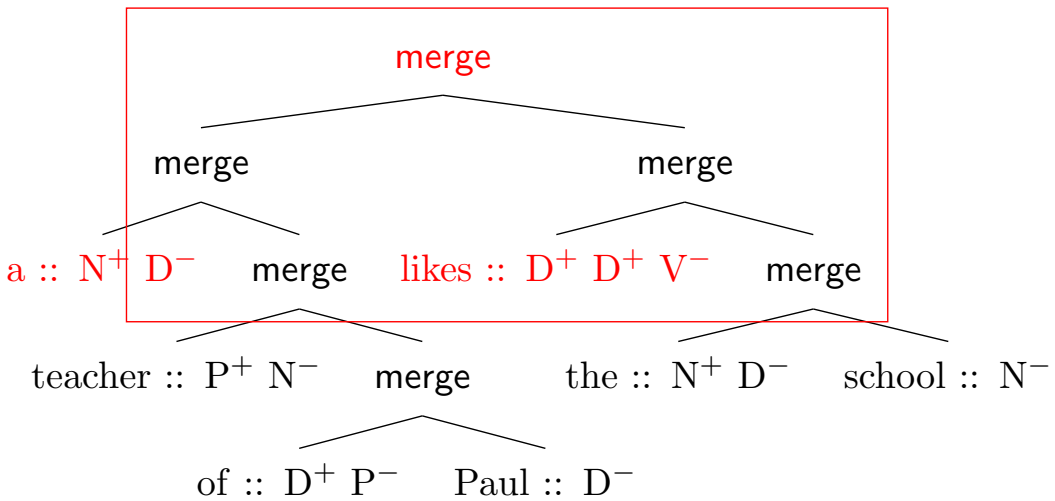
Dependency Trees, Merge, and Head-Argument Relations



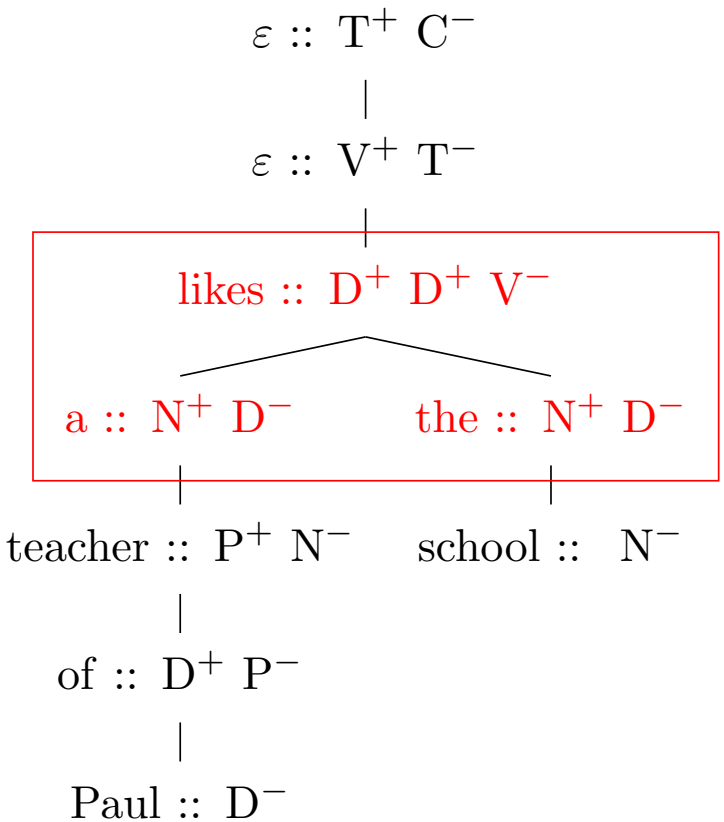


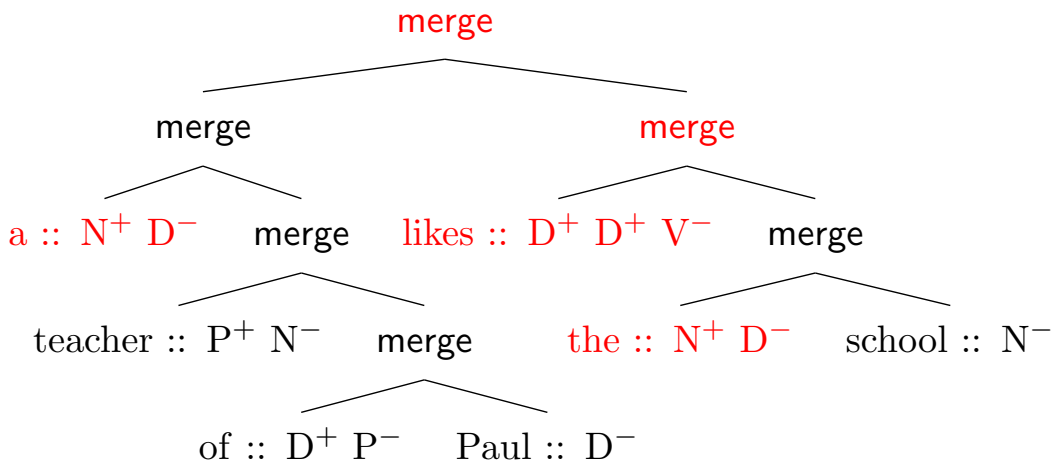
Dependency Trees, Merge, and Head-Argument Relations



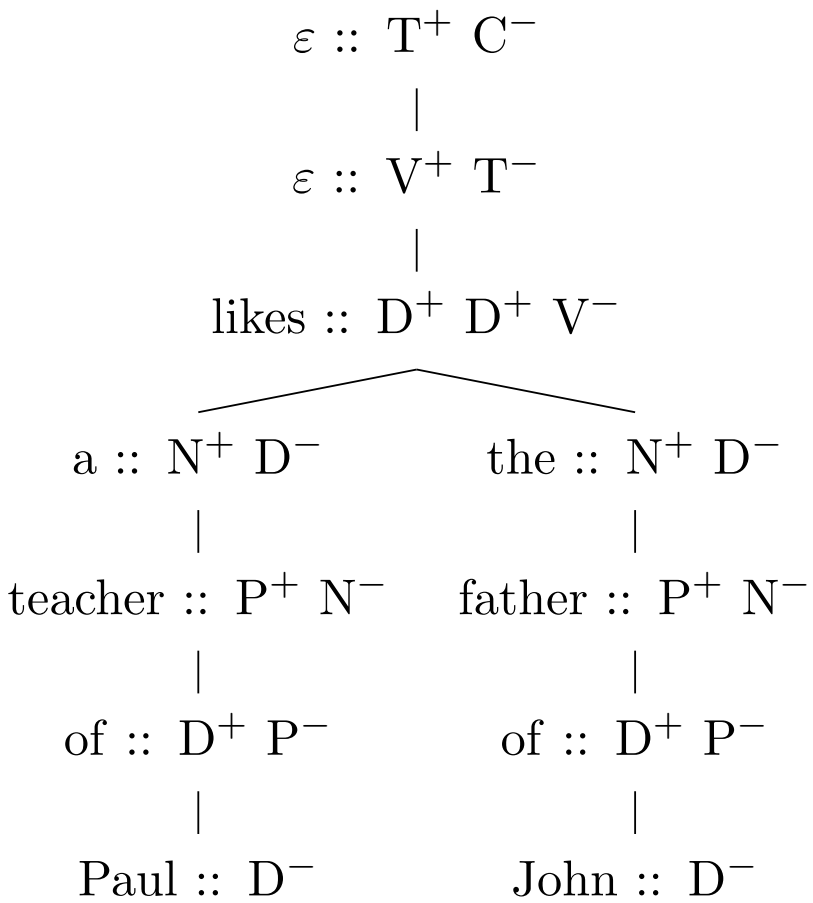


Dependency Trees, Merge, and Head-Argument Relations

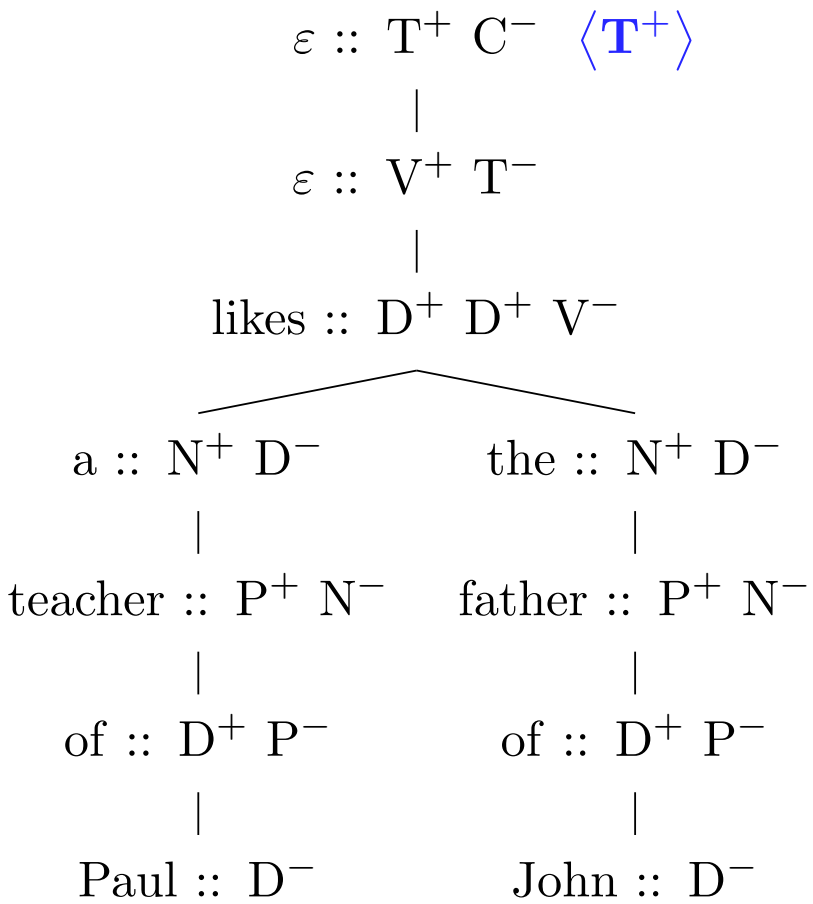




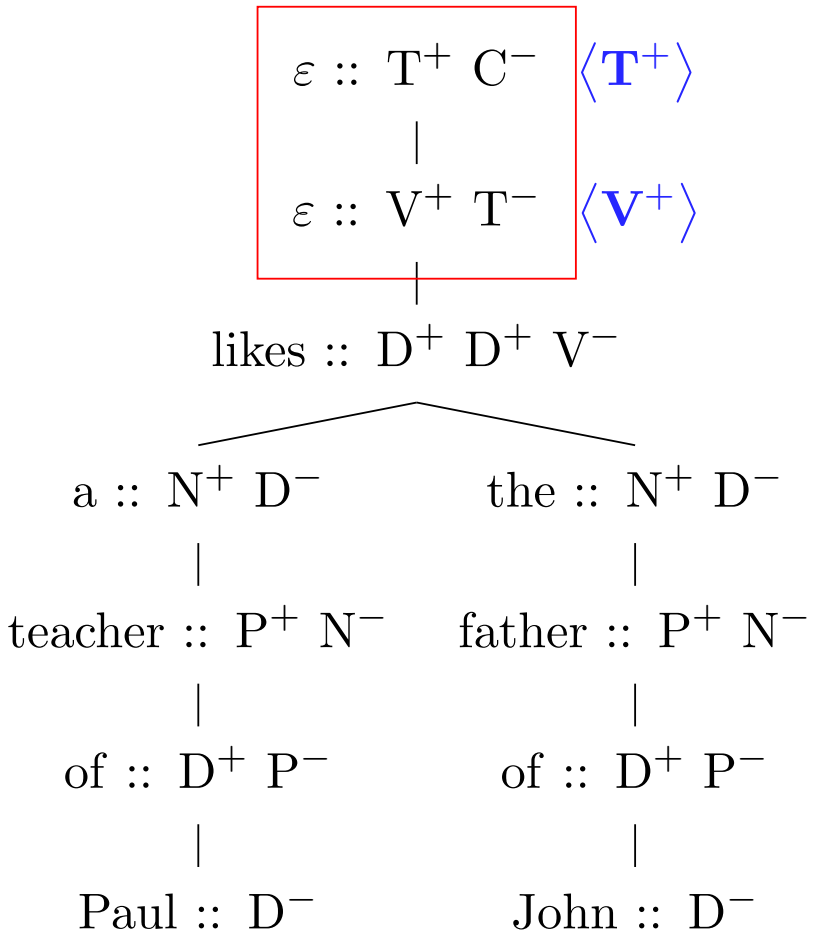
MDEP[merge] $\subsetneq$ STA



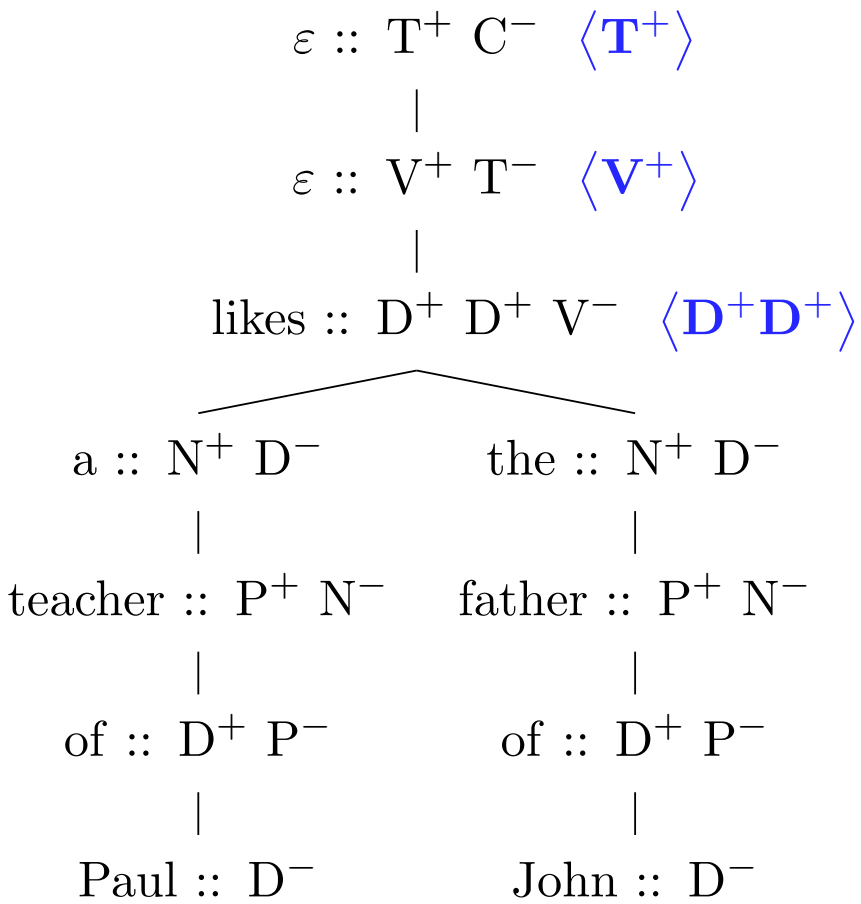
MDEP[merge] $\subsetneq$ STA



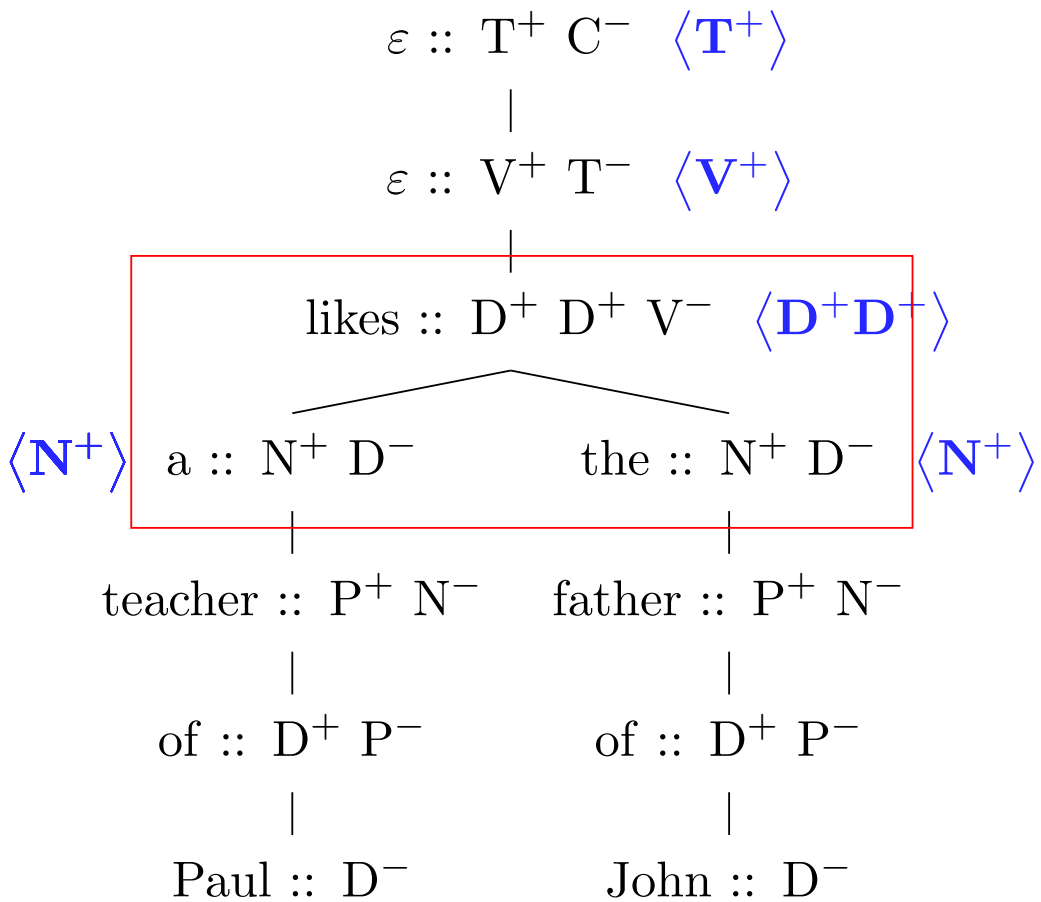
MDEP[merge] $\subsetneq$ STA



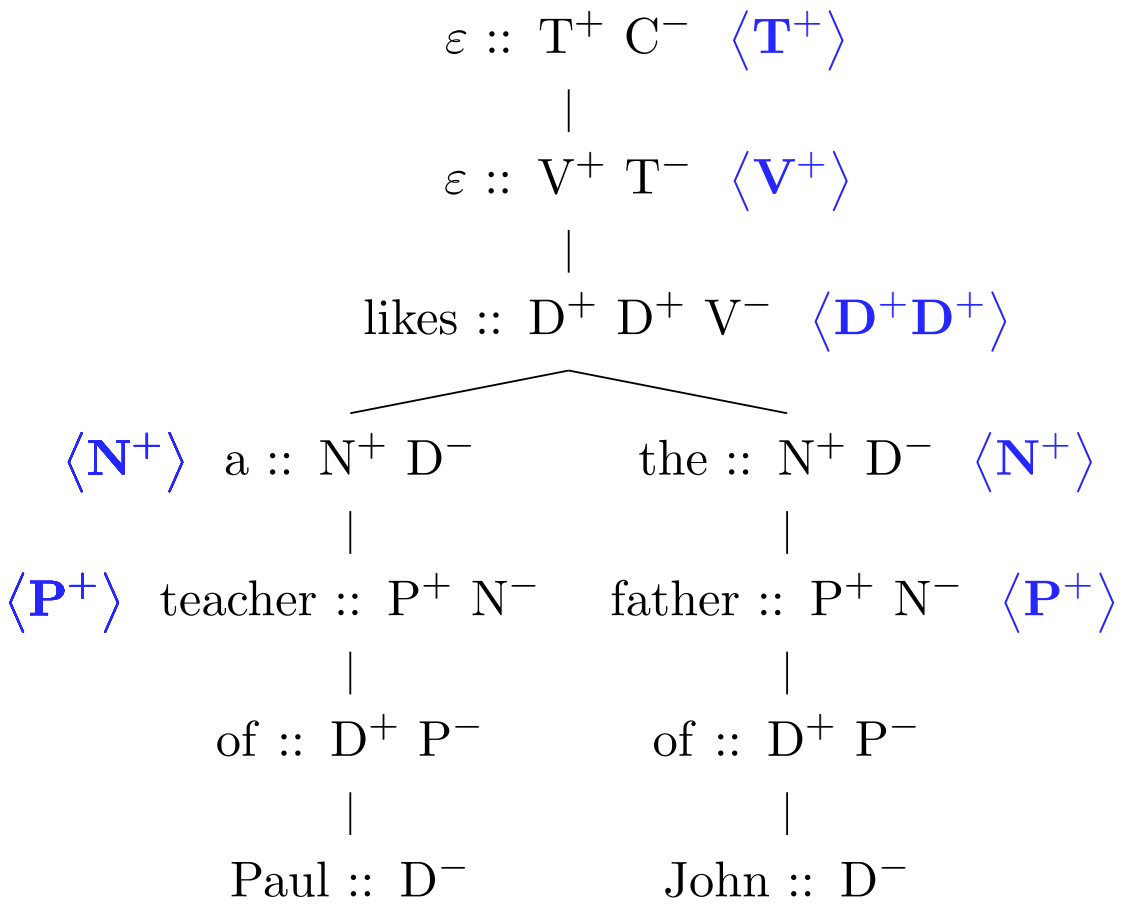
MDEP[merge] $\subsetneq$ STA



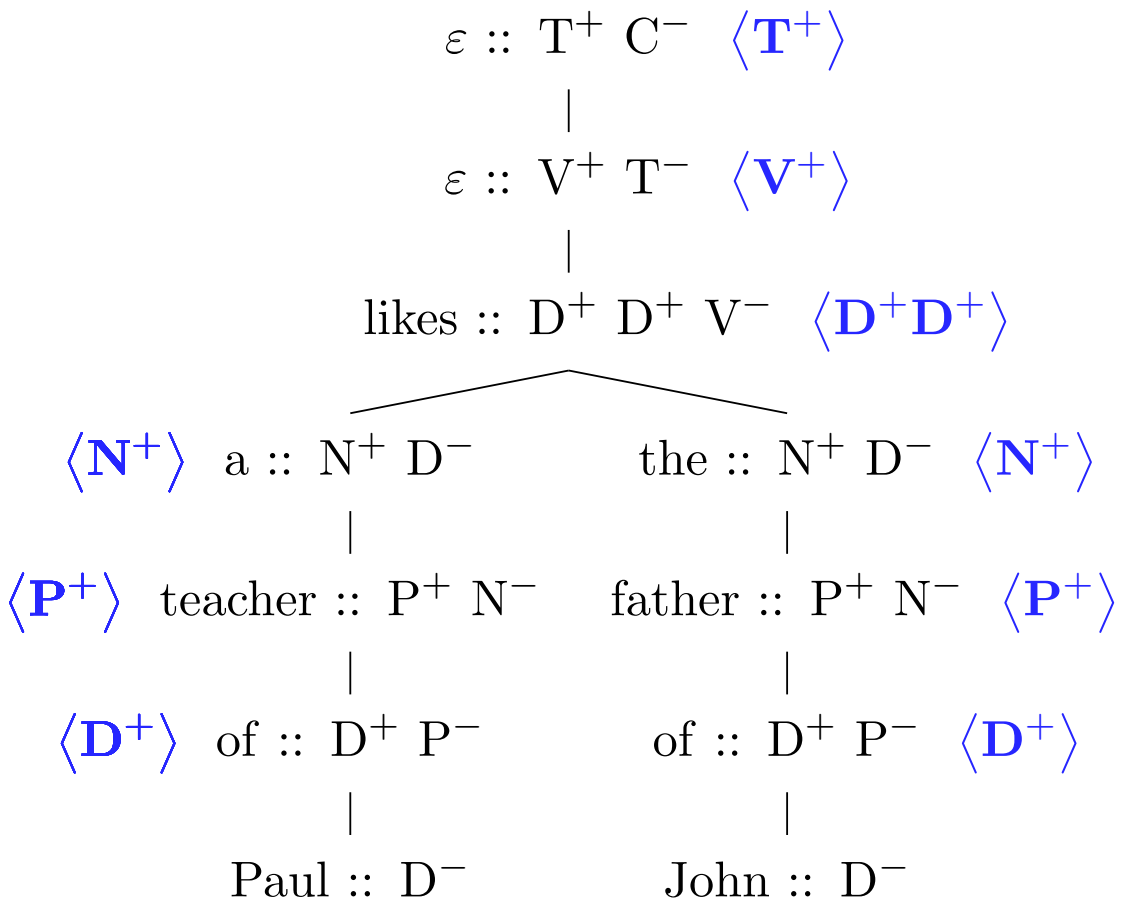
MDEP[merge] $\subsetneq$ STA



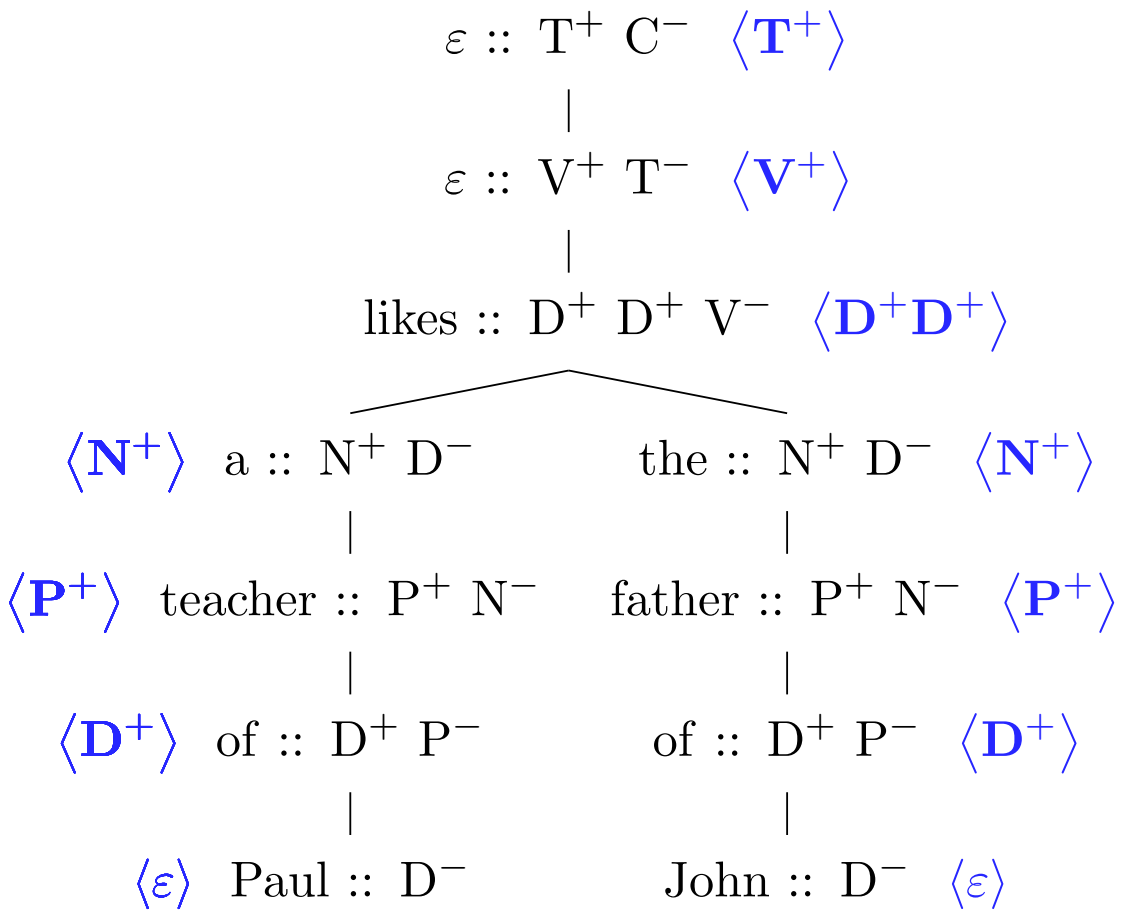
MDEP[merge] $\subsetneq$ STA



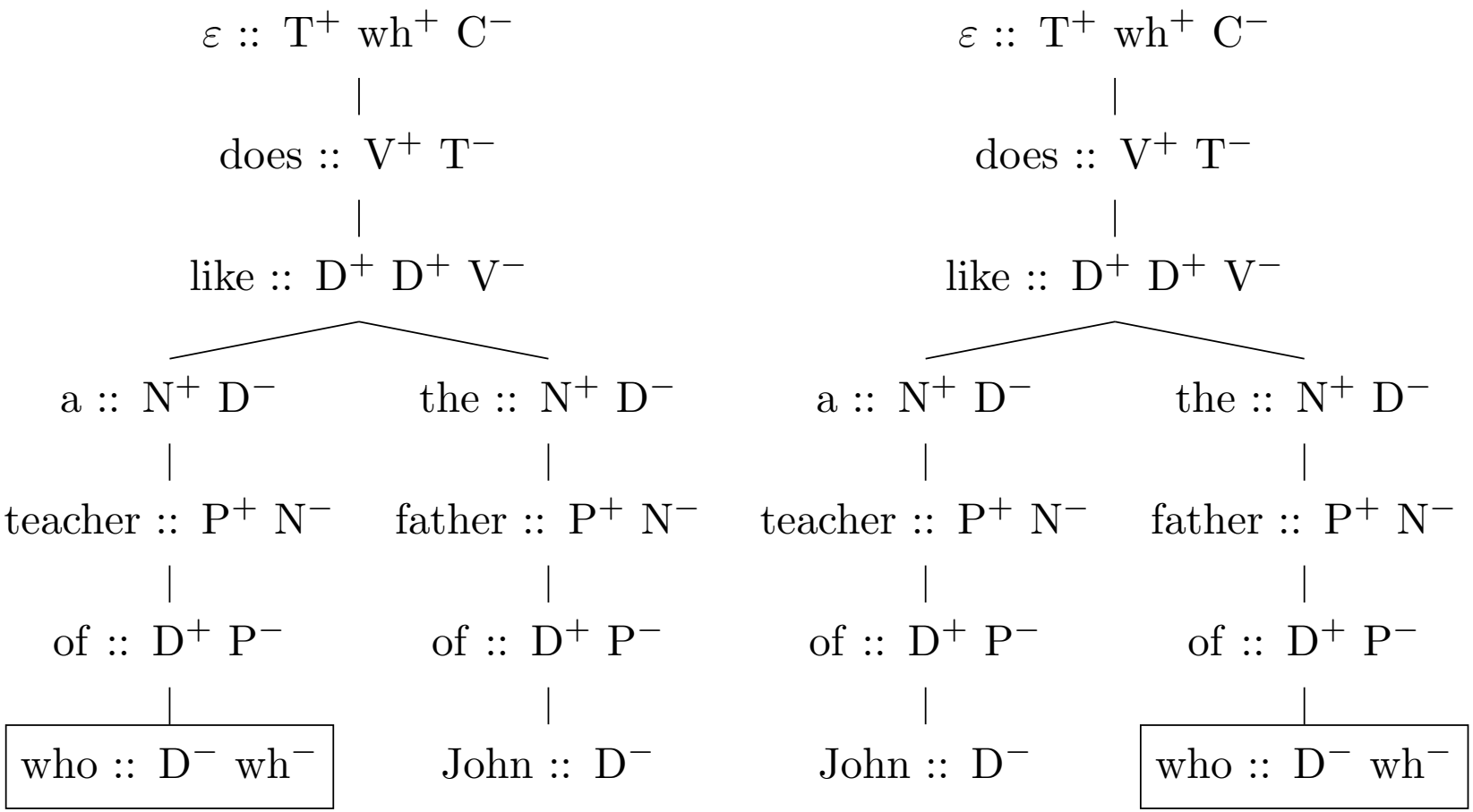
MDEP[merge] $\subsetneq$ STA



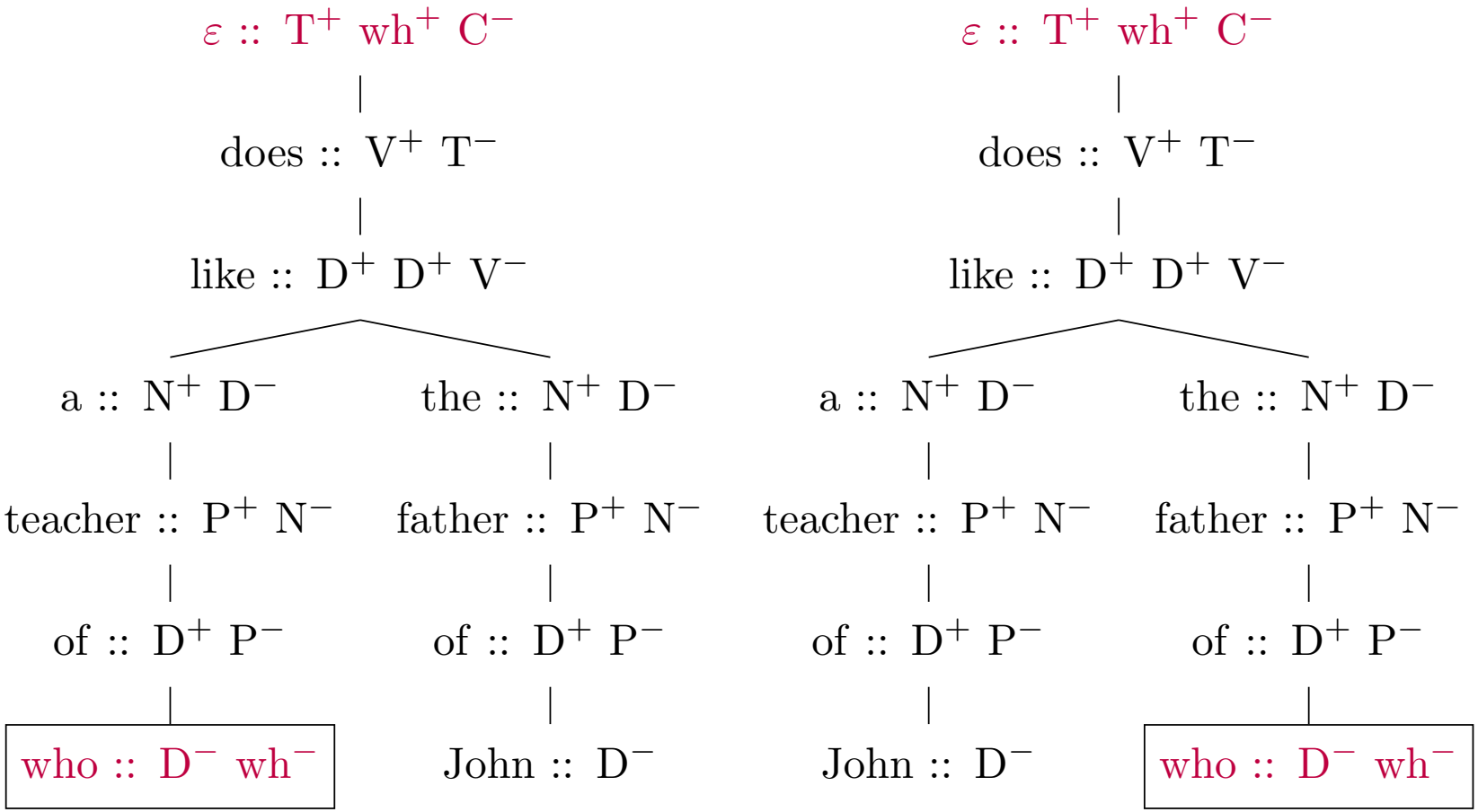
MDEP[merge] $\subsetneq$ STA



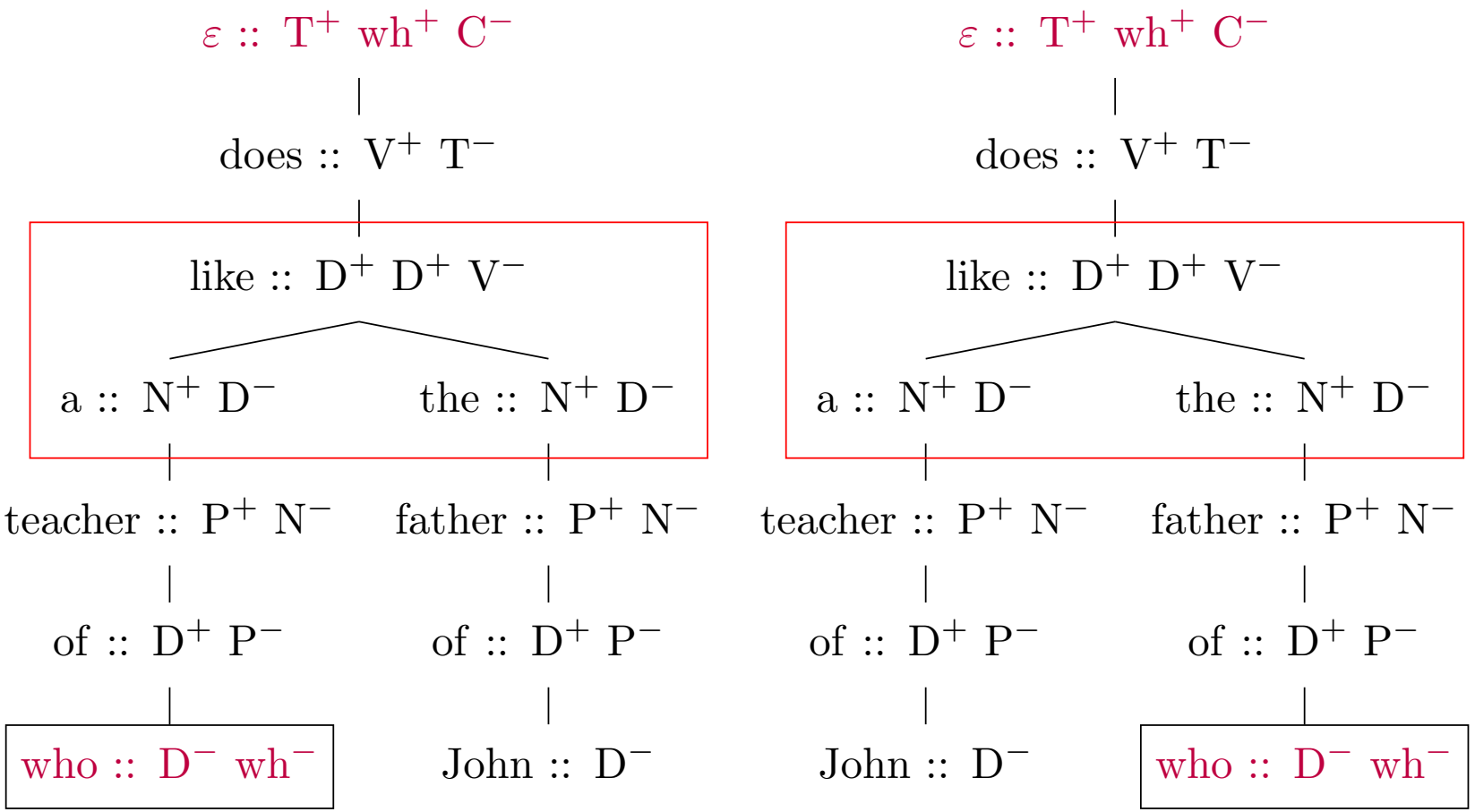
MDEP[merge,move]



MDEP[merge,move]

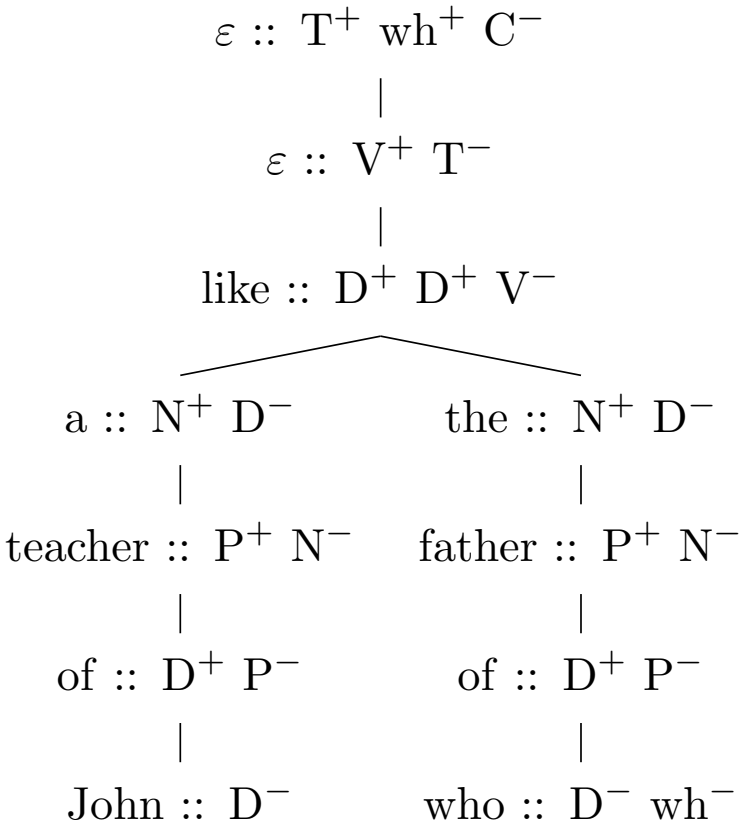


MDEP[merge,move]



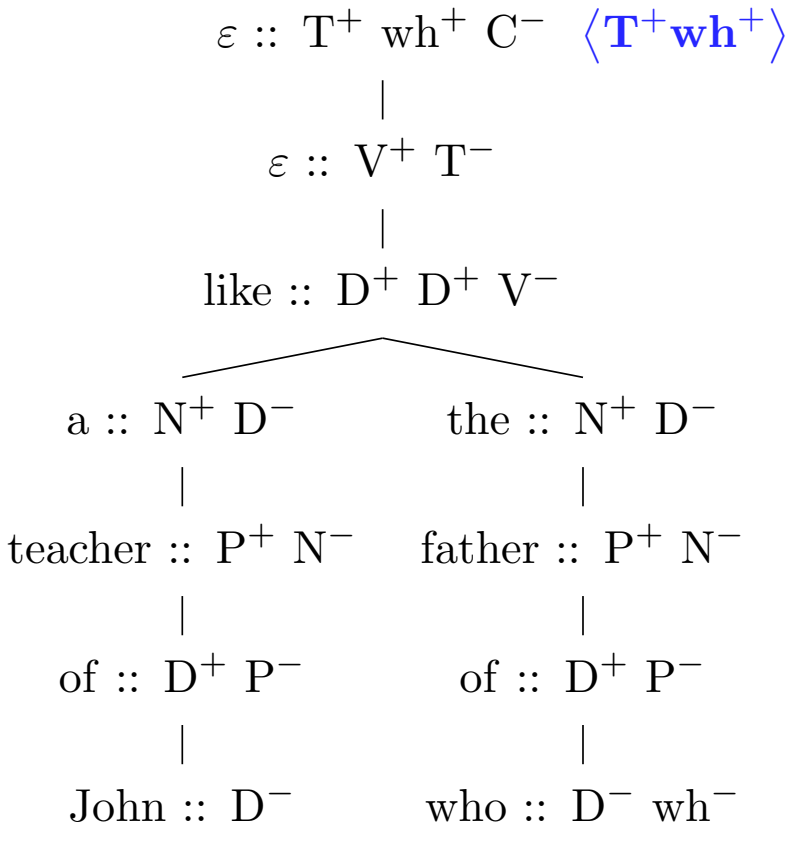
MDEP[merge,move] $\not\subseteq$ STA

Arbitrary movement leads to non-determinism for STAs.



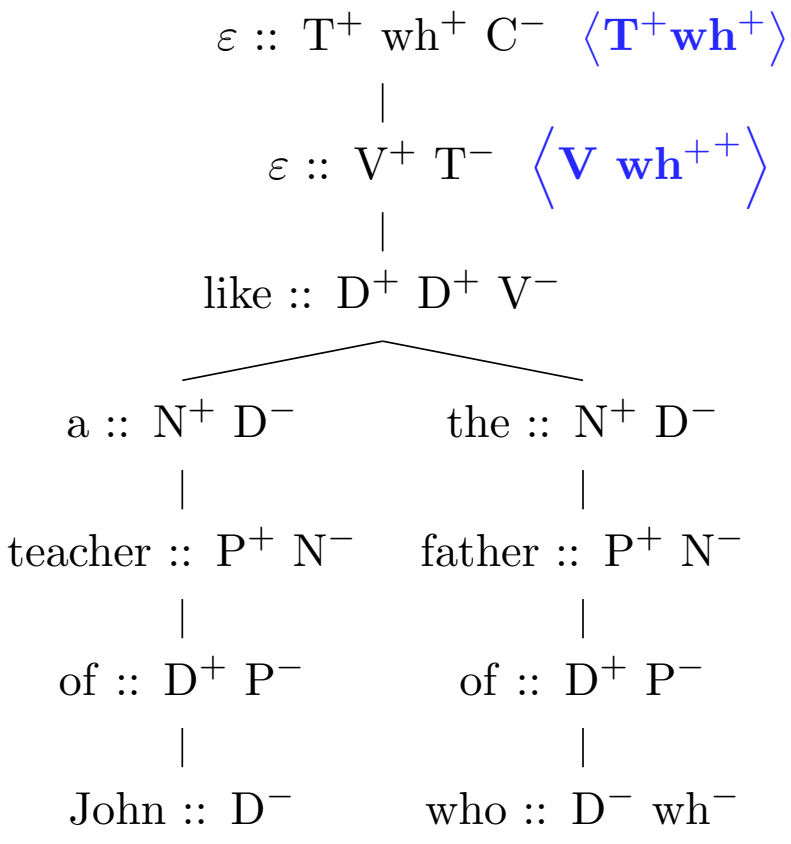
MDEP[merge,move] $\not\subseteq$ STA

Arbitrary movement leads to non-determinism for STAs.



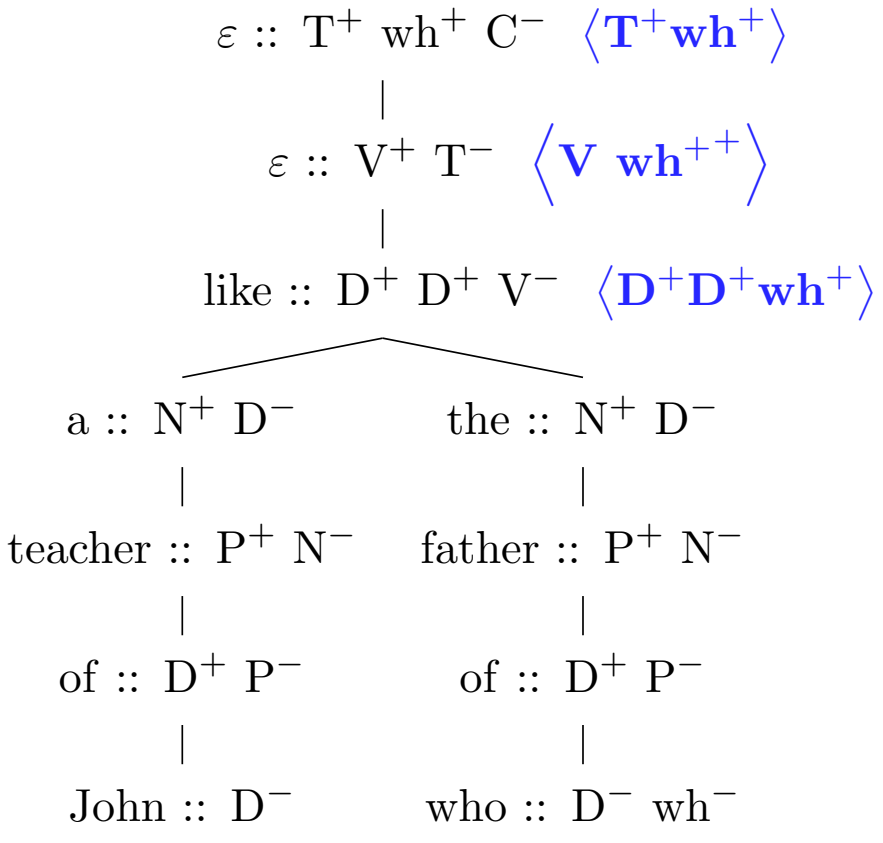
MDEP[merge,move] $\not\subseteq$ STA

Arbitrary movement leads to non-determinism for STAs.



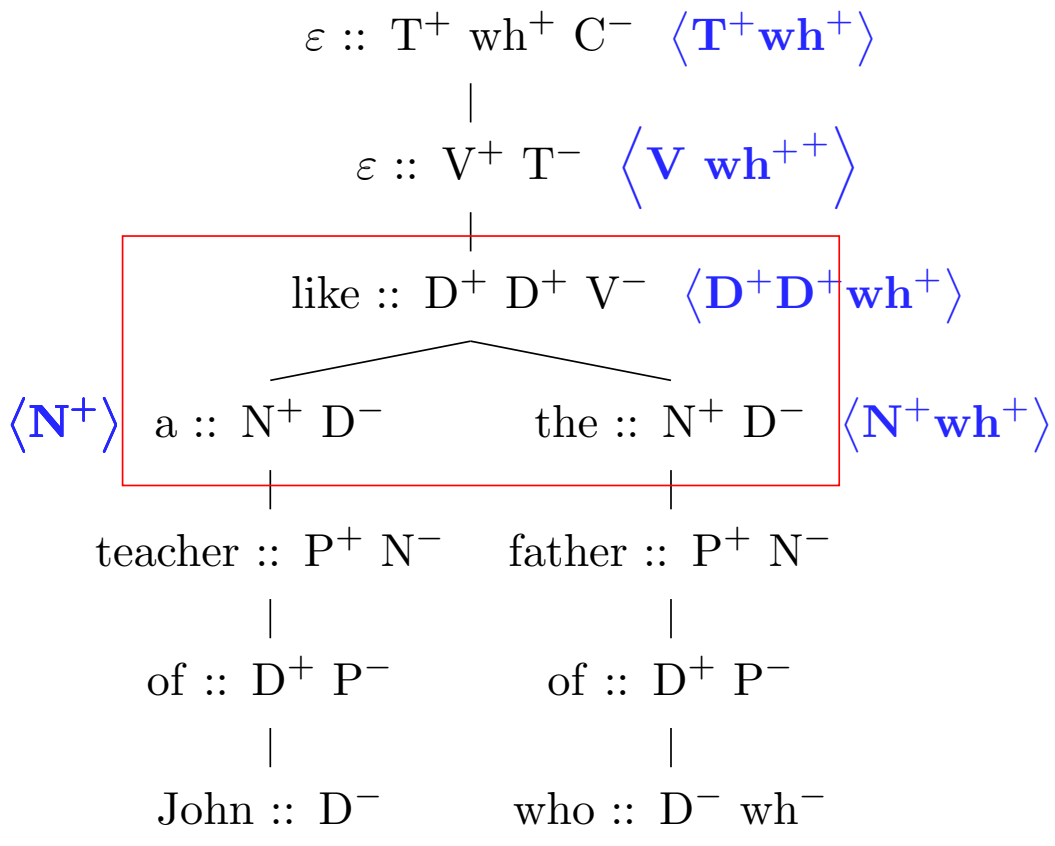
MDEP[merge,move] $\not\subseteq$ STA

Arbitrary movement leads to non-determinism for STAs.



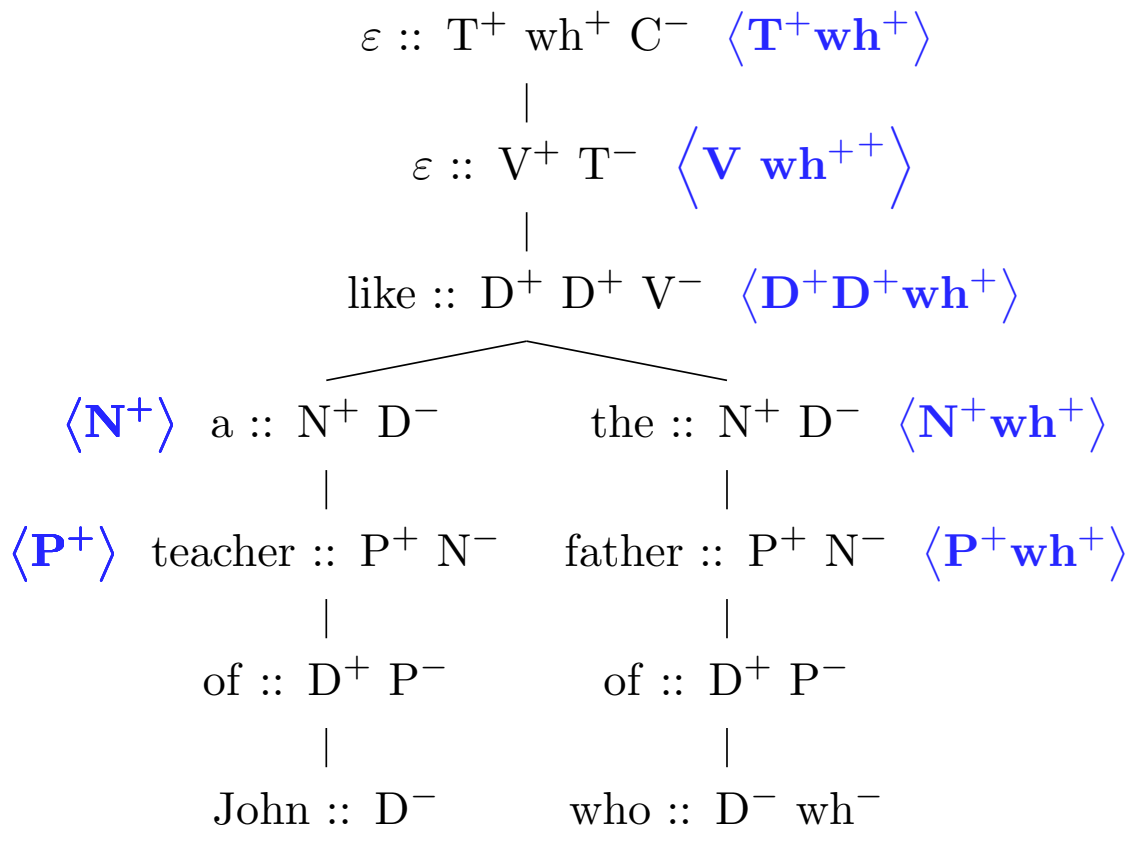
MDEP[merge,move] $\not\subseteq$ STA

Arbitrary movement leads to non-determinism for STAs.



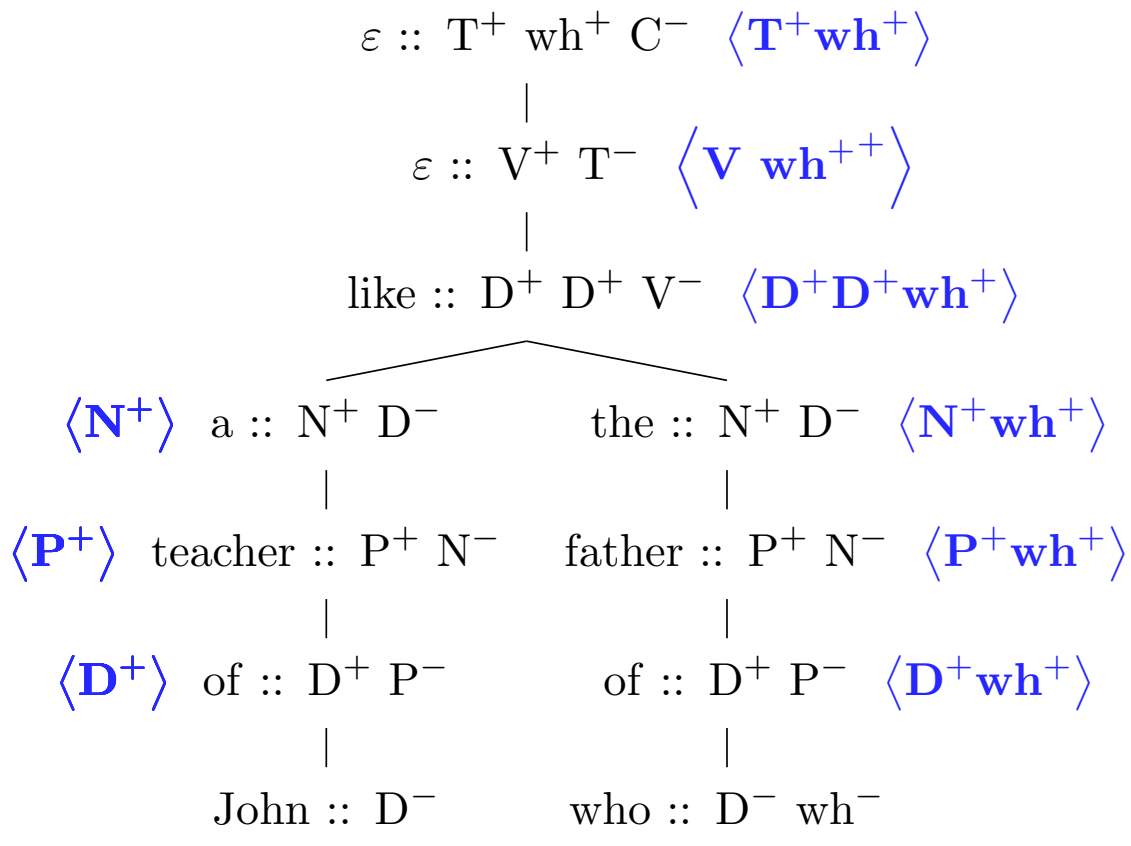
MDEP[merge,move] $\not\subseteq$ STA

Arbitrary movement leads to non-determinism for STAs.



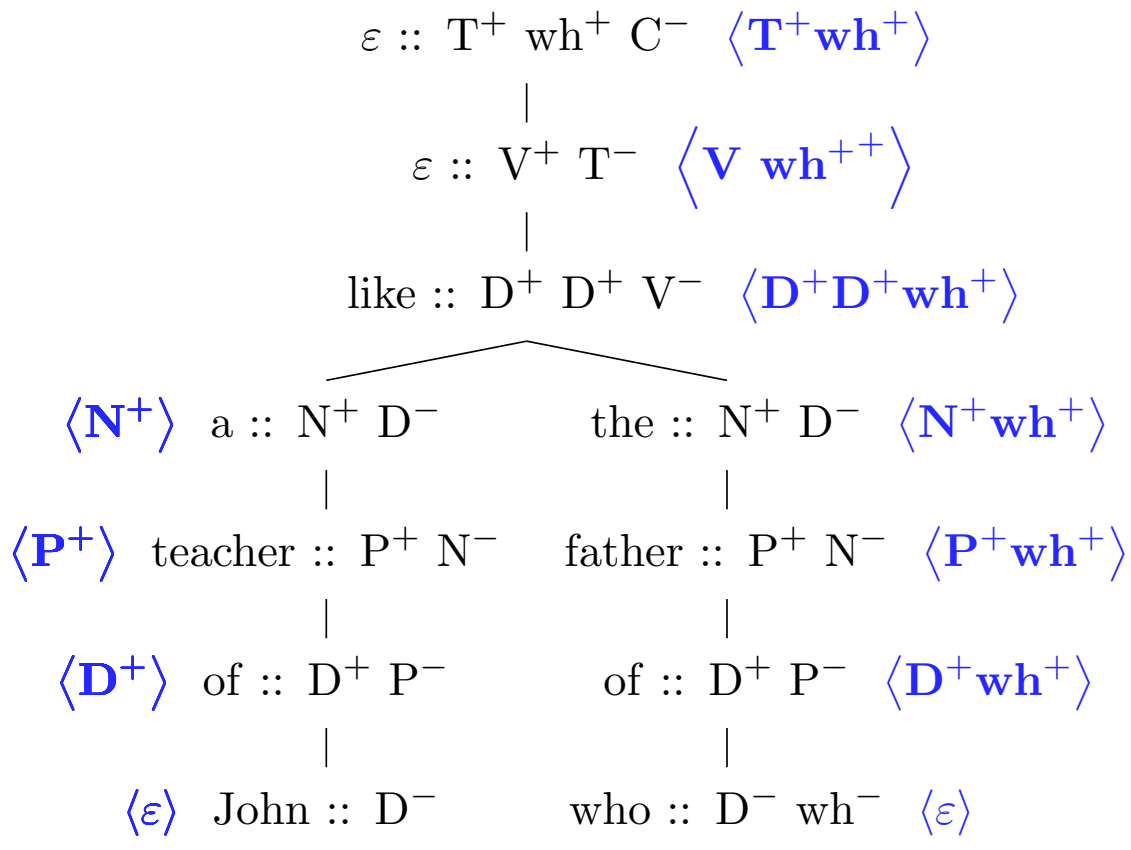
MDEP[merge,move] $\not\subseteq$ STA

Arbitrary movement leads to non-determinism for STAs.



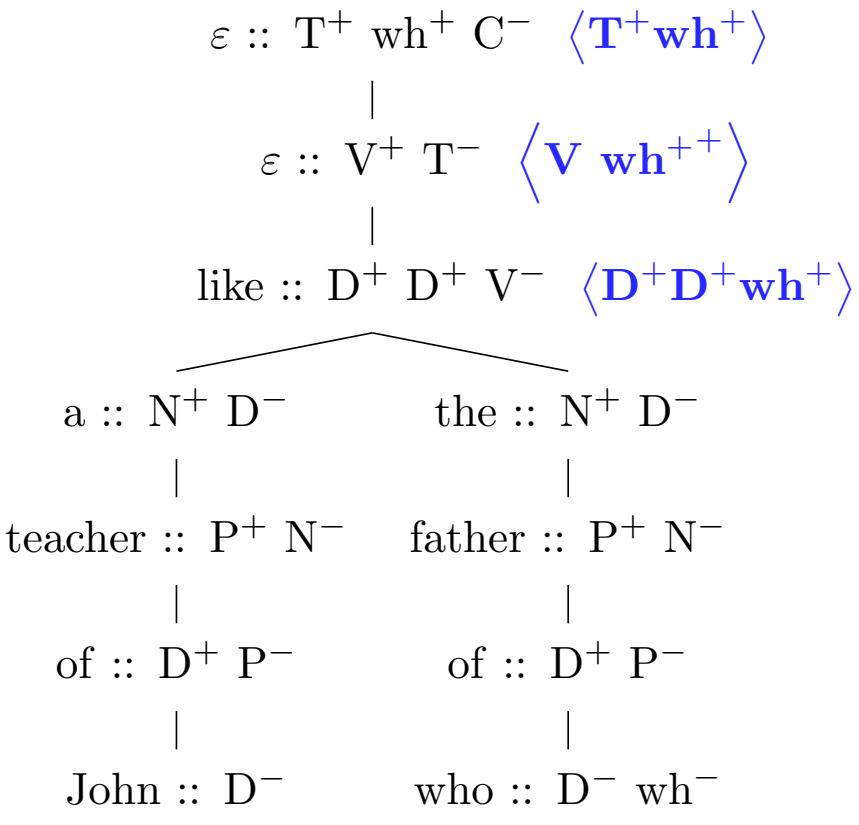
MDEP[merge,move] $\not\subseteq$ STA

Arbitrary movement leads to non-determinism for STAs.



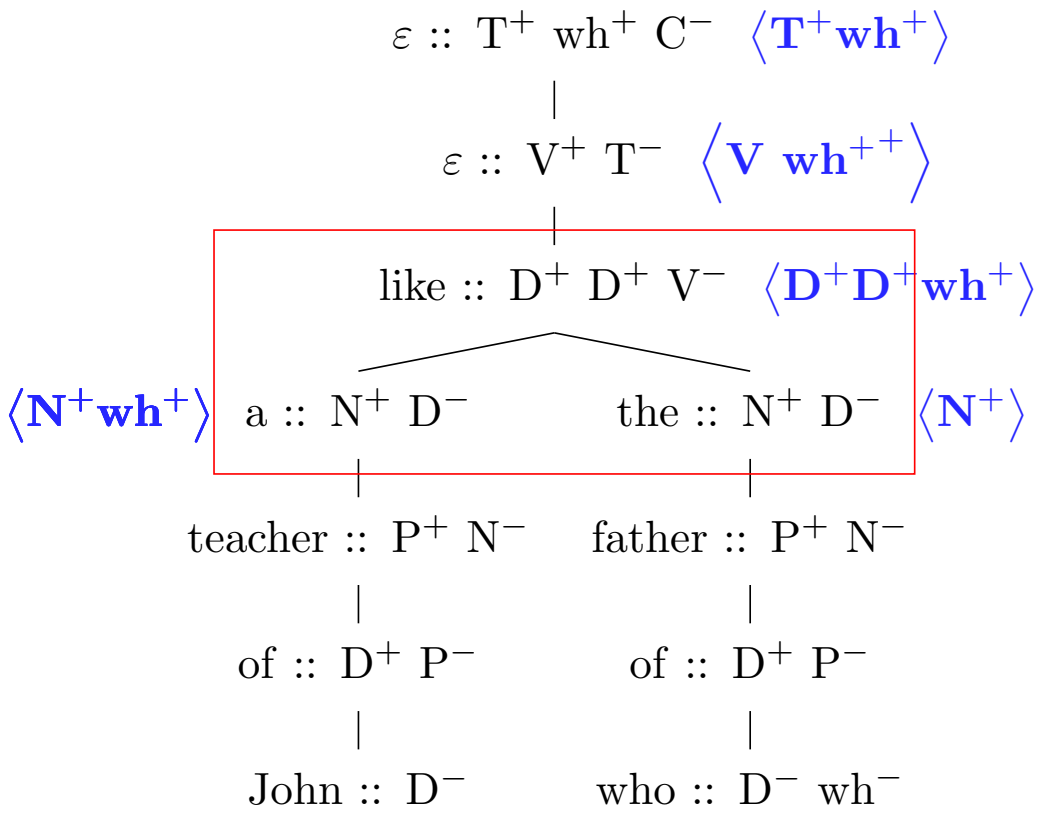
MDEP[merge,move] $\not\subseteq$ STA

Arbitrary movement leads to non-determinism for STAs.



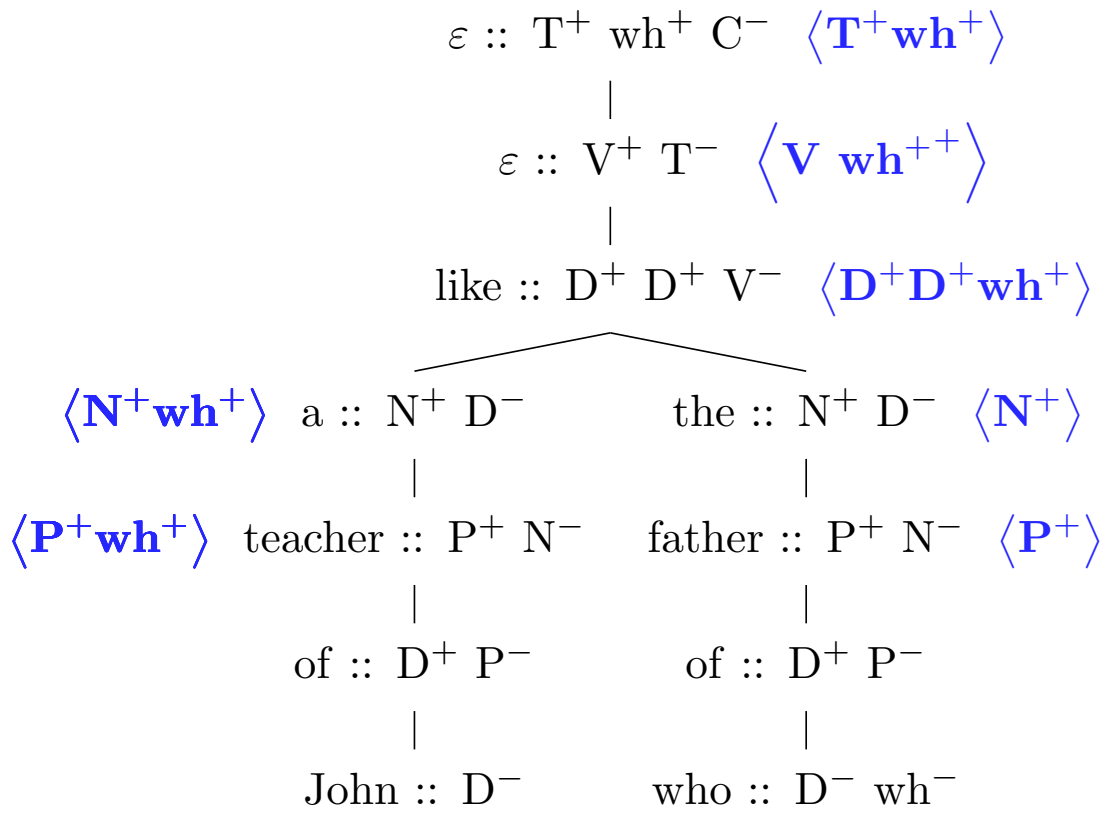
MDEP[merge,move] $\not\subseteq$ STA

Arbitrary movement leads to non-determinism for STAs.



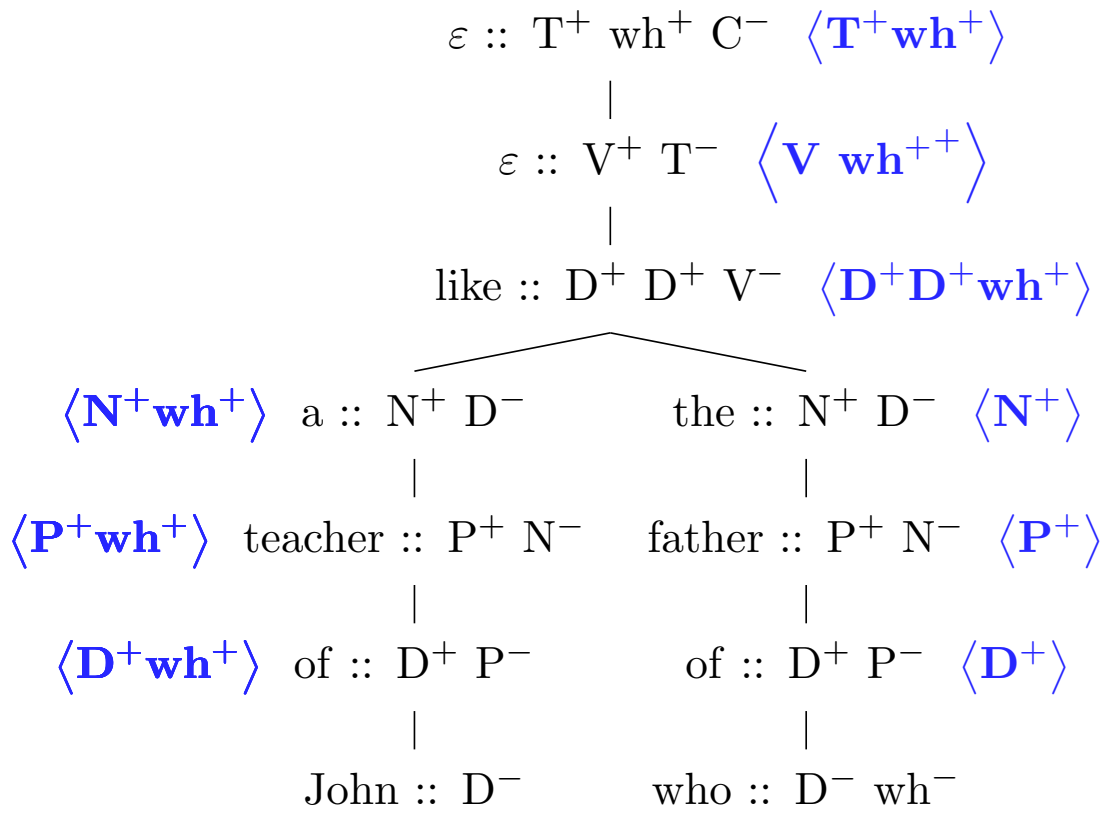
MDEP[merge,move] $\not\subseteq$ STA

Arbitrary movement leads to non-determinism for STAs.



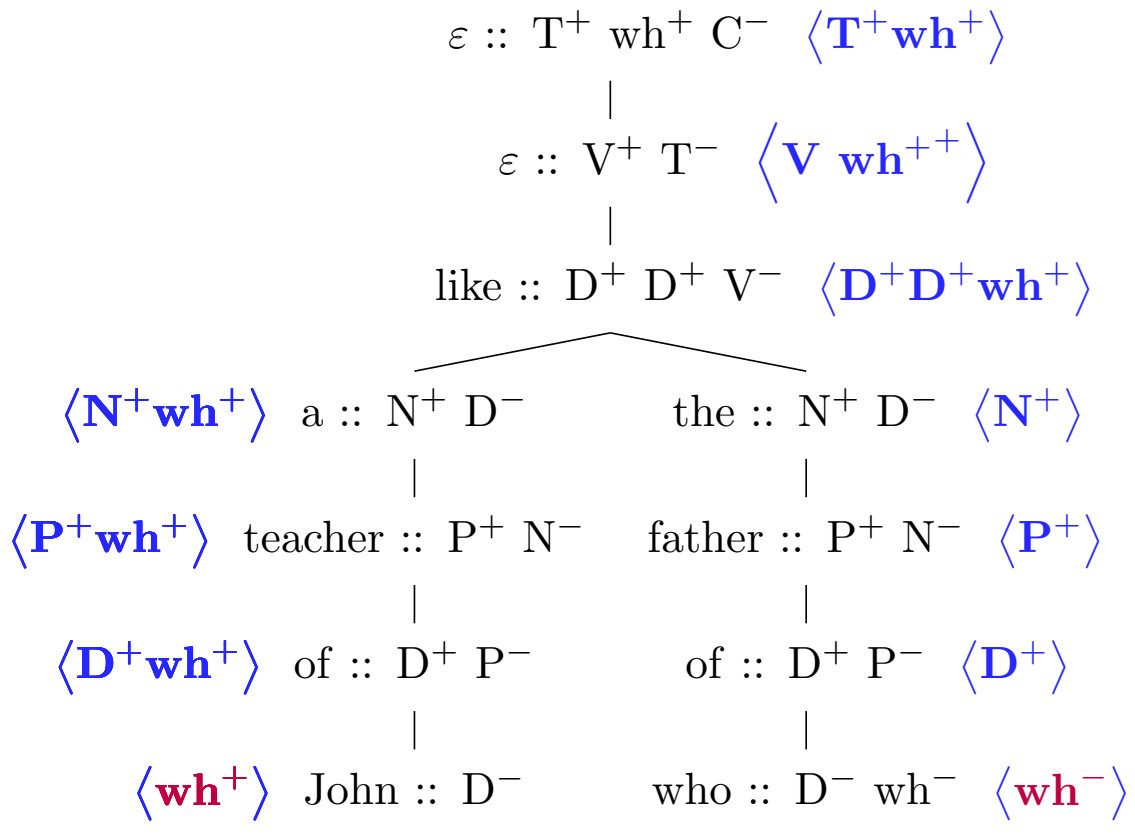
MDEP[merge,move] $\not\subseteq$ STA

Arbitrary movement leads to non-determinism for STAs.

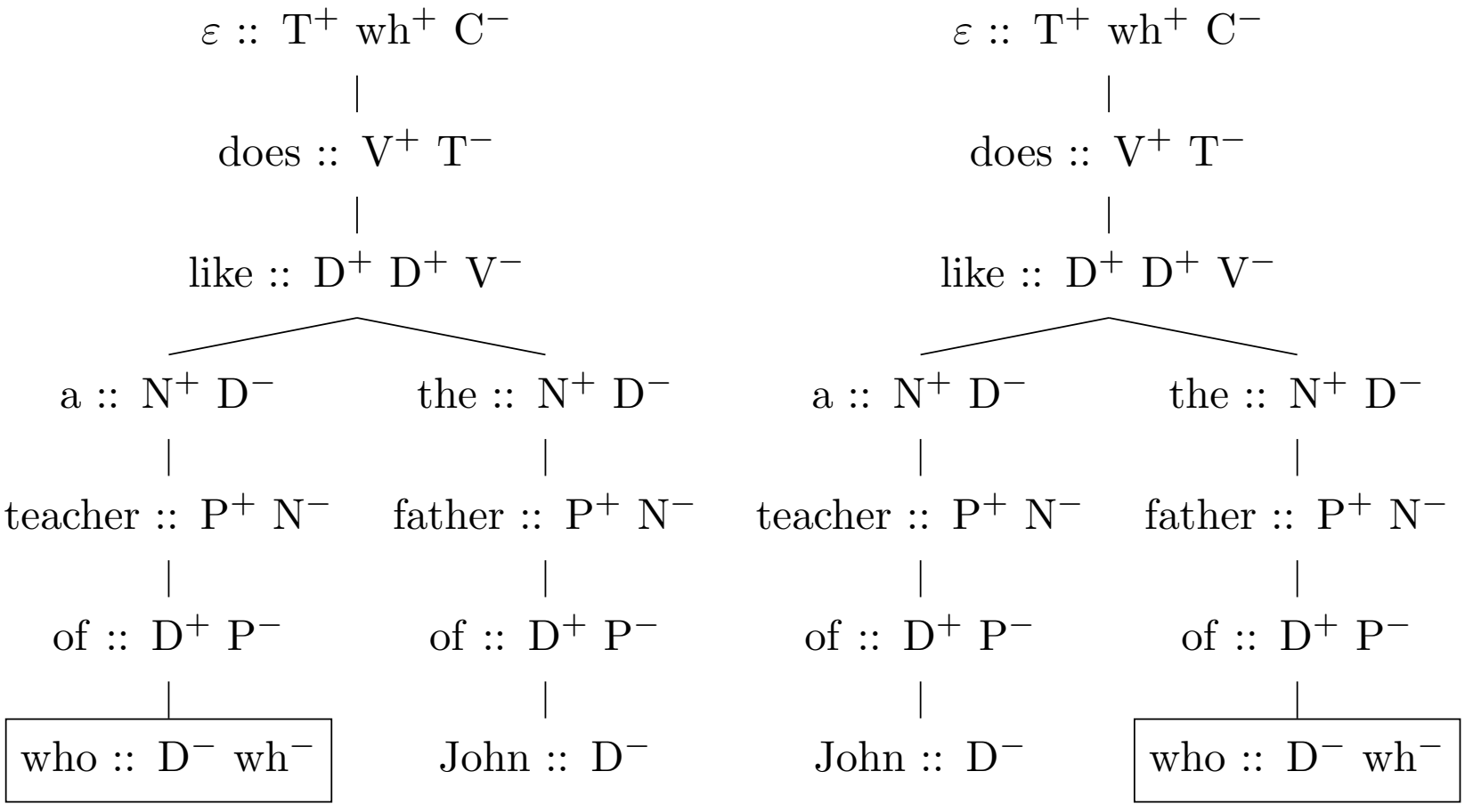


MDEP[merge,move] $\not\subseteq$ STA

Arbitrary movement leads to non-determinism for STAs.



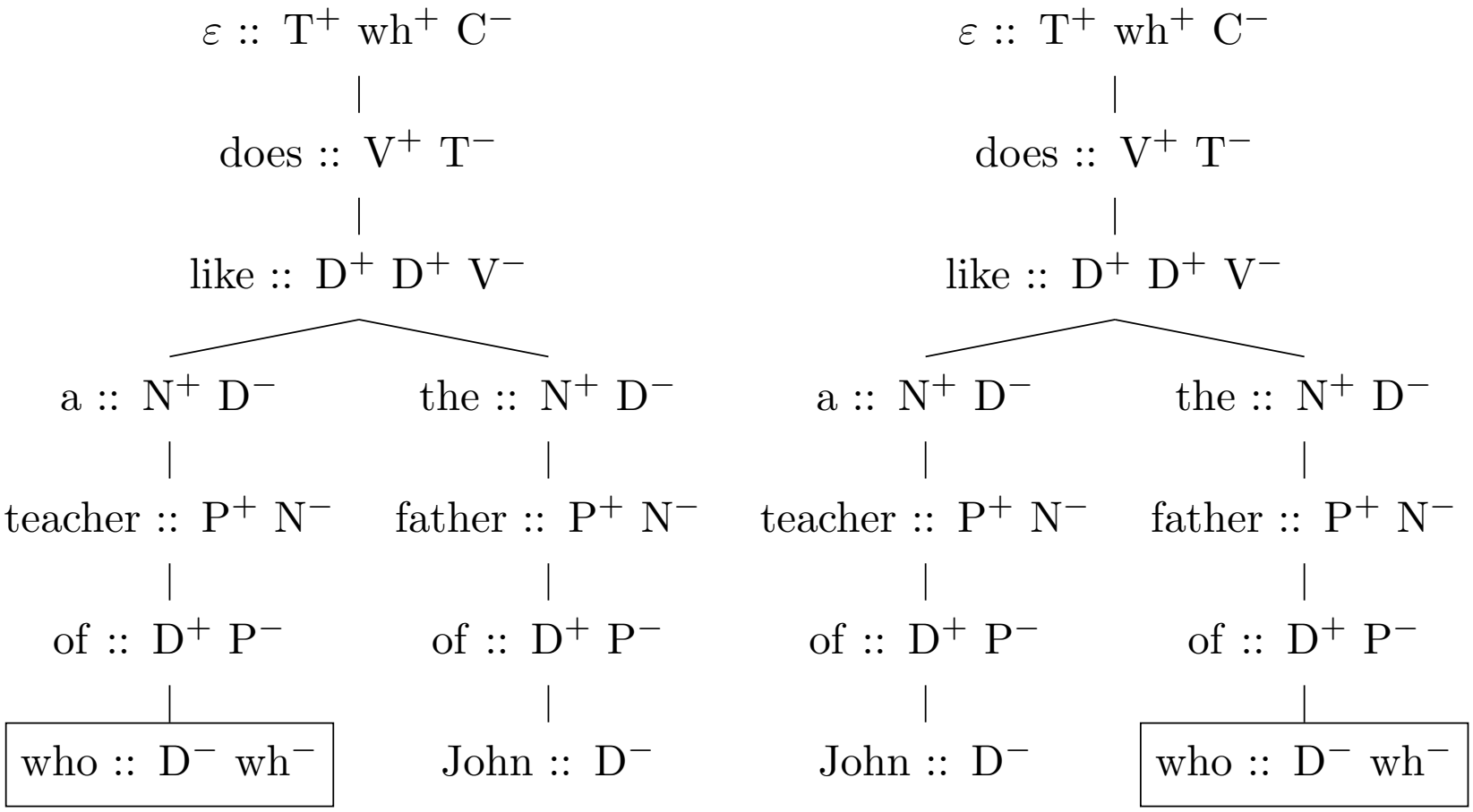
Restricting *move*



The Specifier Island Constraint (SpIC)

1. *Who does a teacher of __ like the father of John?

Restricting *move*

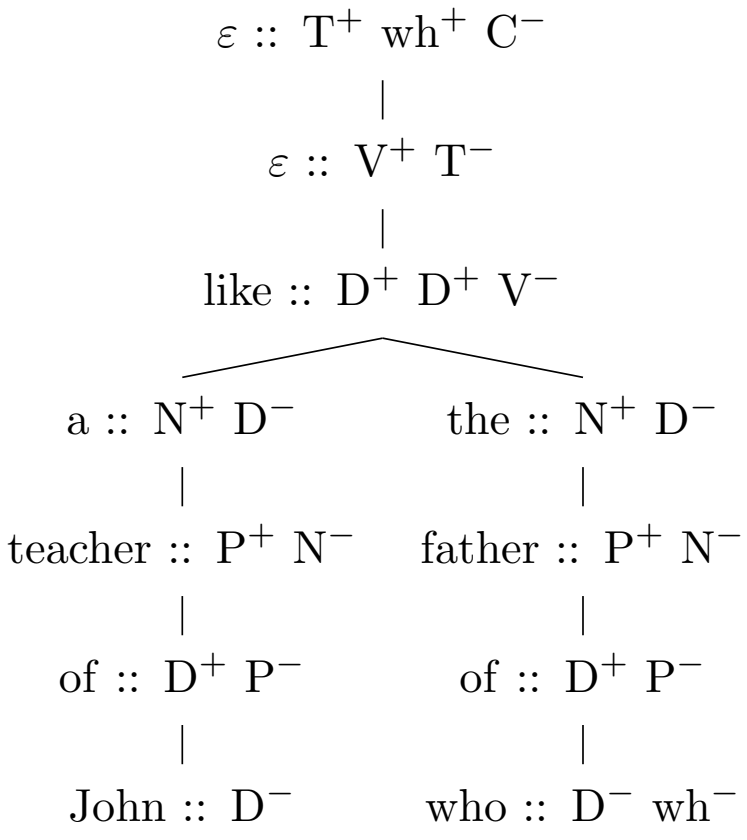


The Specifier Island Constraint (SpIC)

1. *__Who__ does a teacher of __ like the father of John?

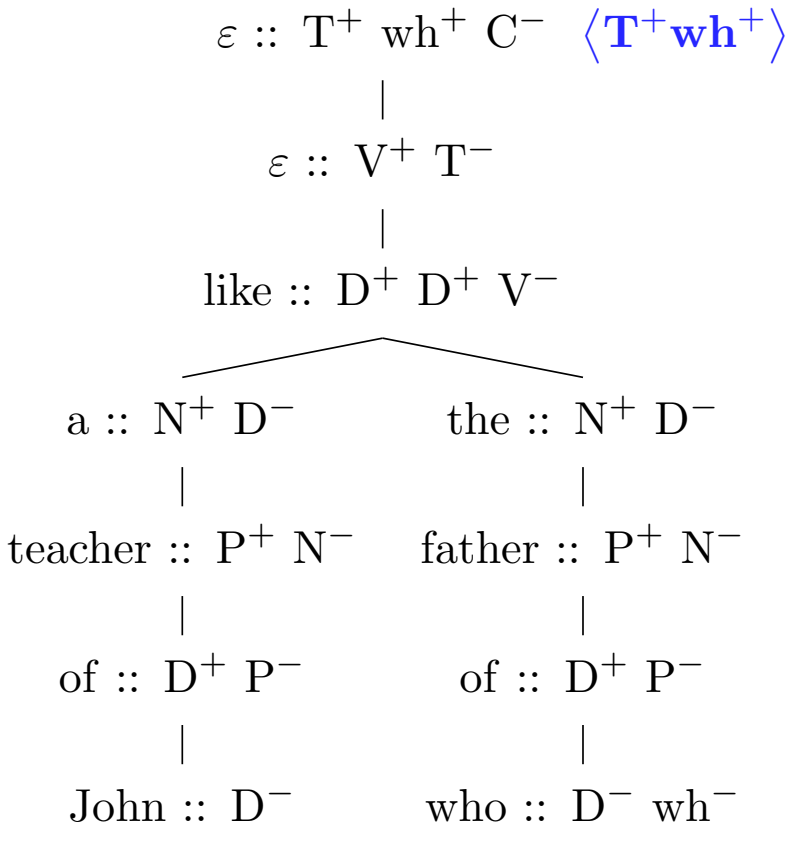
MDEP[merge,move] + SpIC $\subsetneq$ STA

The SpIC guarantees STA recognition.



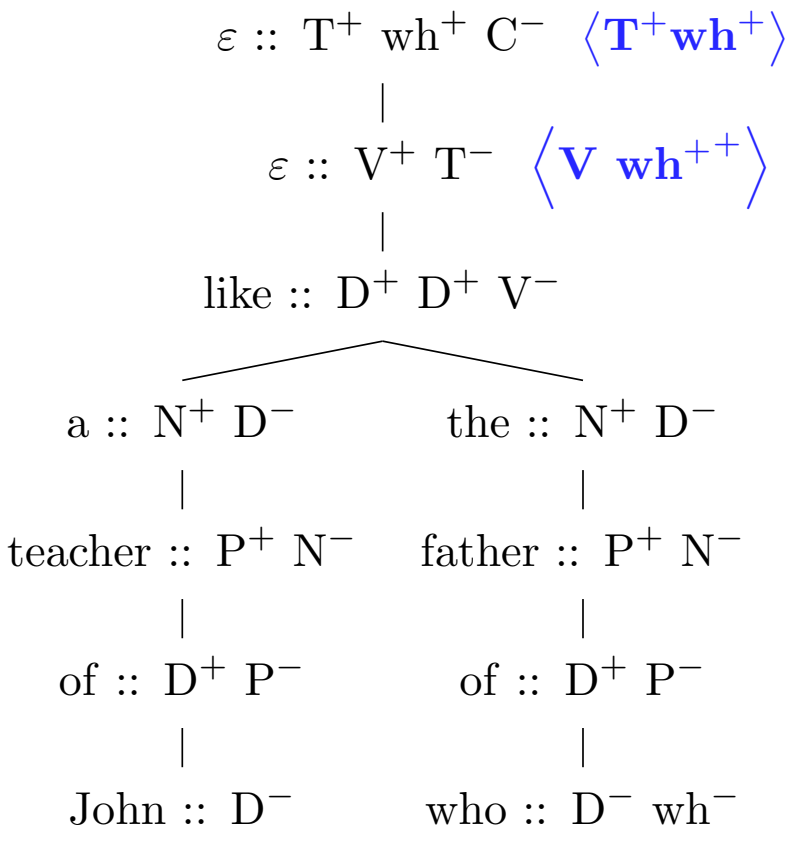
MDEP[merge,move] + SpIC $\subsetneq$ STA

The SpIC guarantees STA recognition.



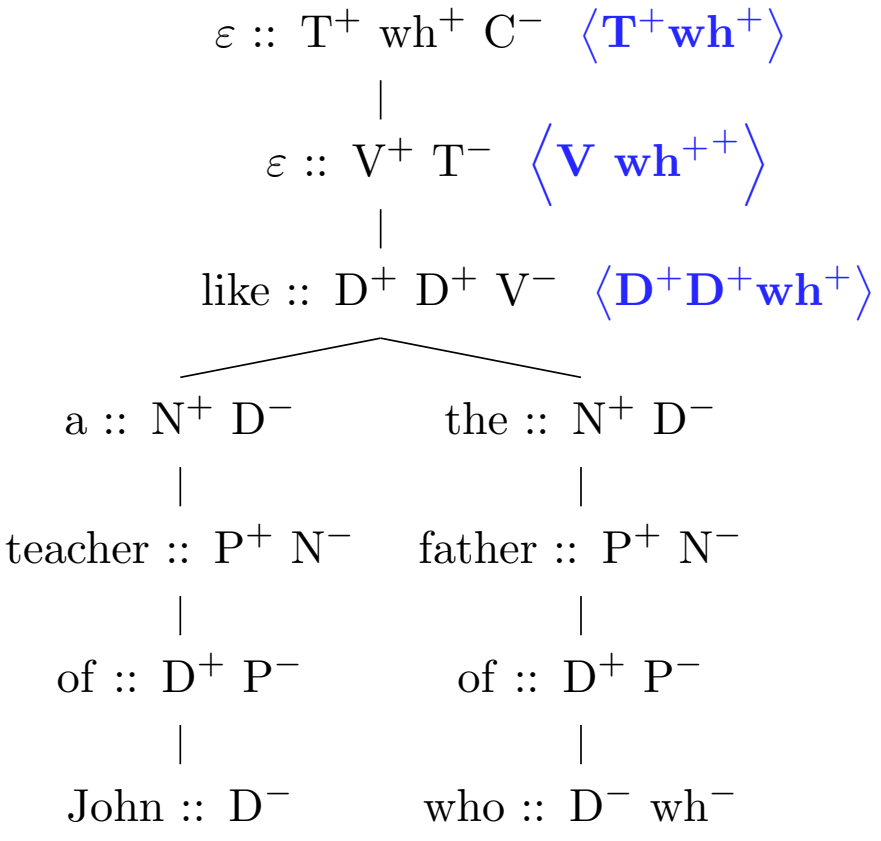
MDEP[merge,move] + SpIC $\subsetneq$ STA

The SpIC guarantees STA recognition.



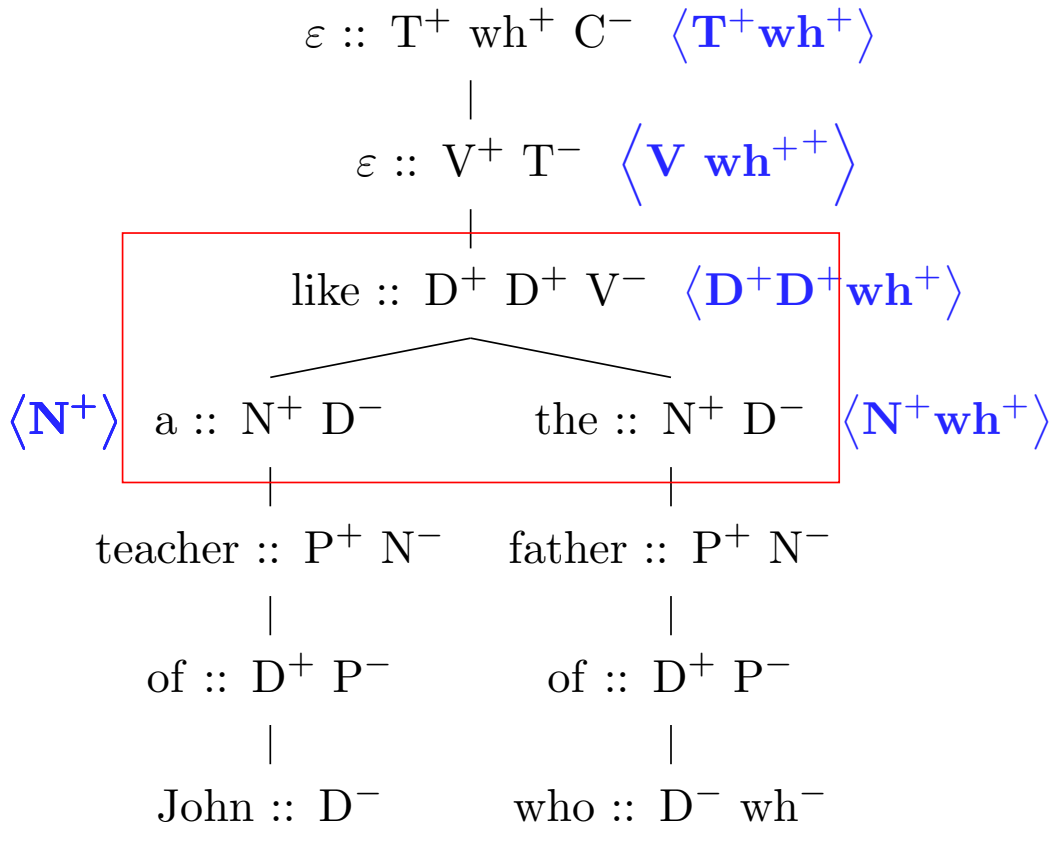
MDEP[merge,move] + SpIC $\subsetneq$ STA

The SpIC guarantees STA recognition.



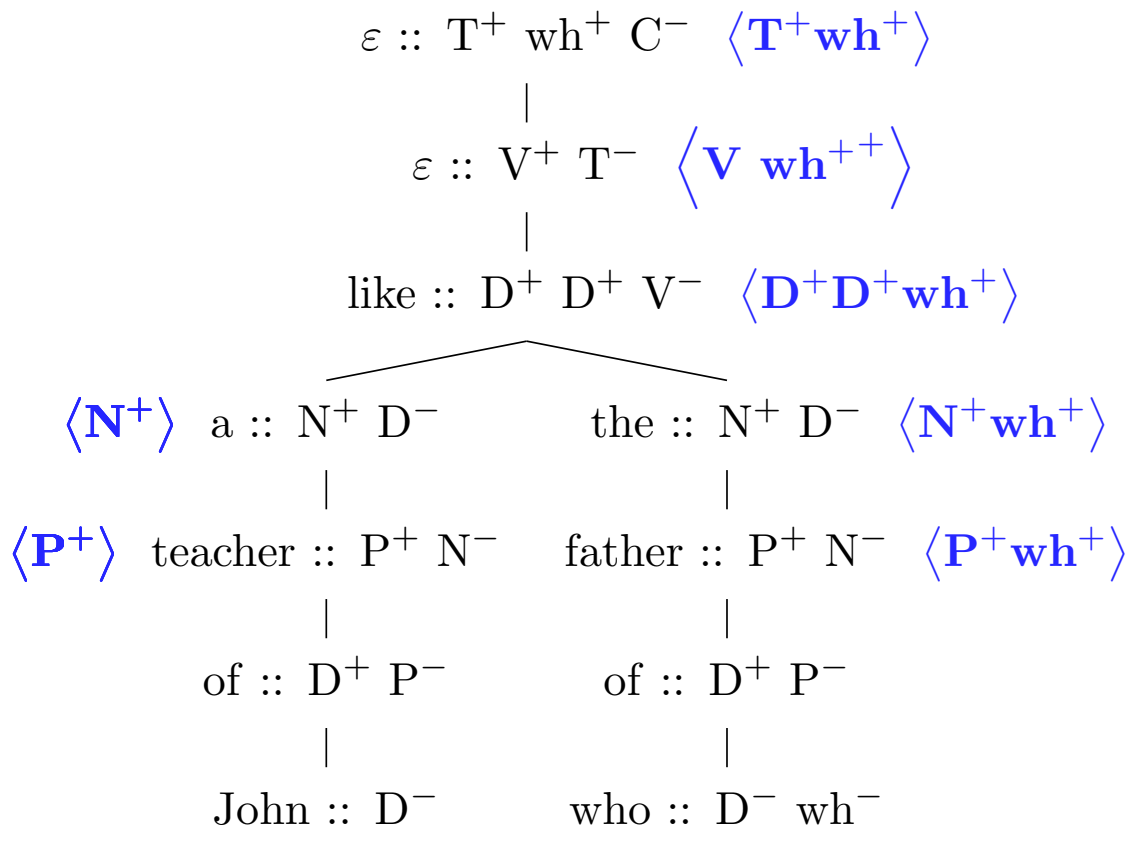
MDEP[merge,move] + SpIC $\subsetneq$ STA

The SpIC guarantees STA recognition.



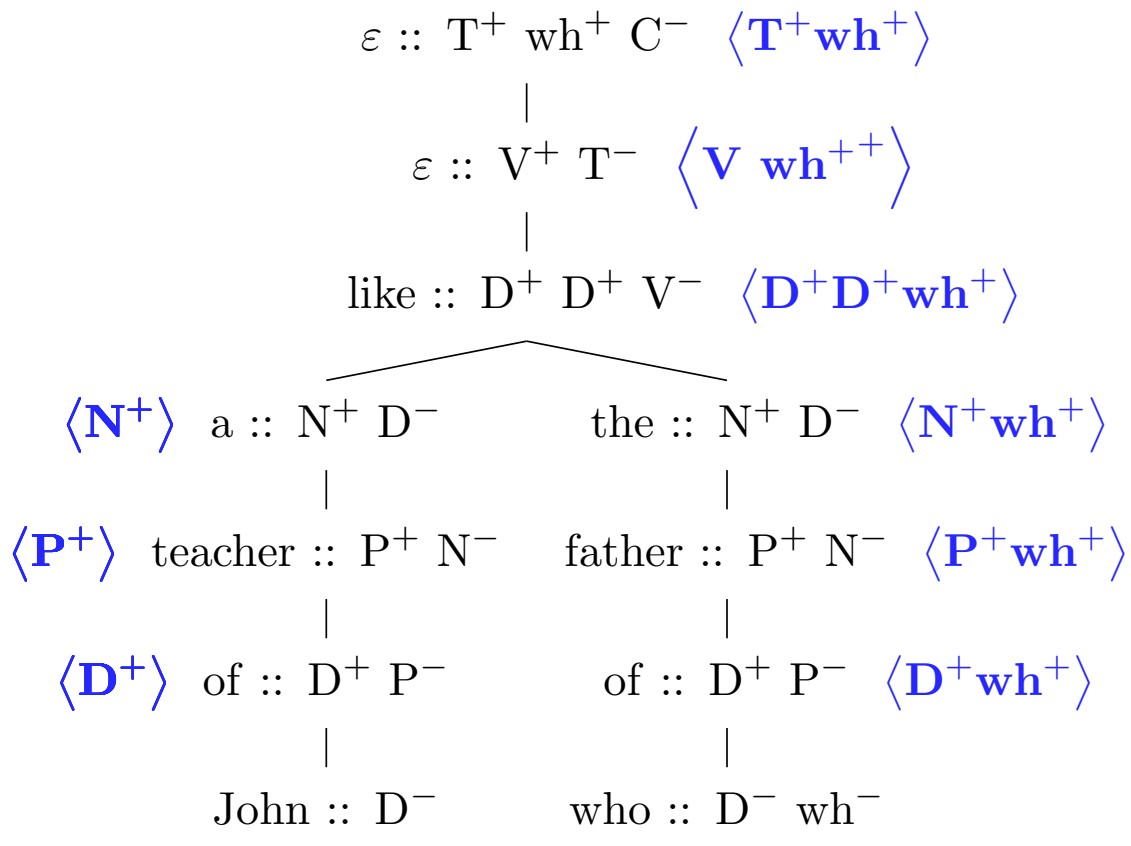
MDEP[merge,move] + SpIC $\subsetneq$ STA

The SpIC guarantees STA recognition.



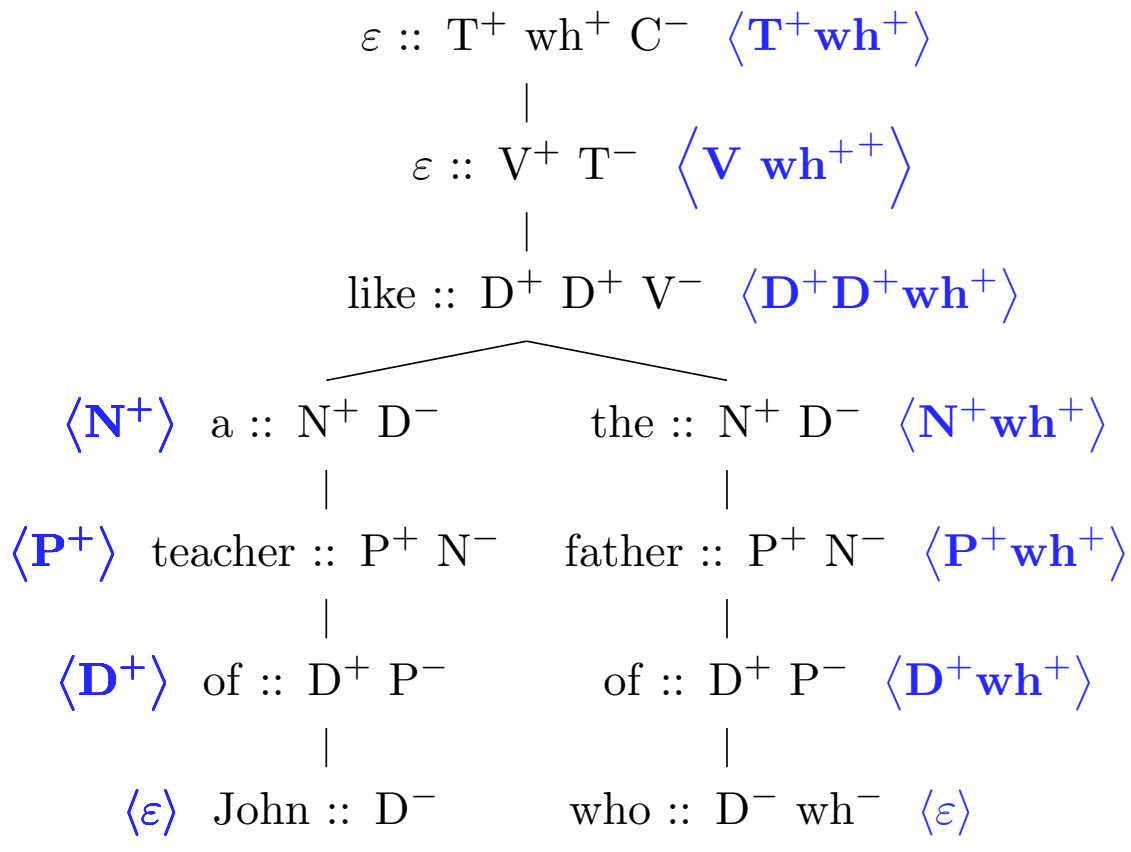
MDEP[merge,move] + SpIC $\subsetneq$ STA

The SpIC guarantees STA recognition.



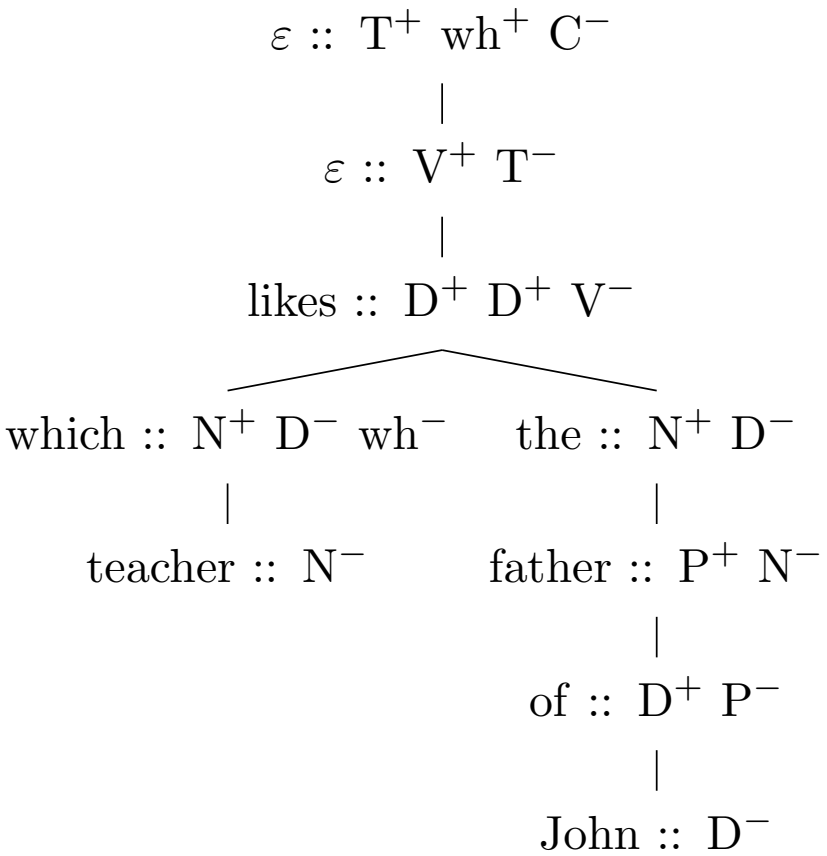
MDEP[merge,move] + SpIC $\subsetneq$ STA

The SpIC guarantees STA recognition.



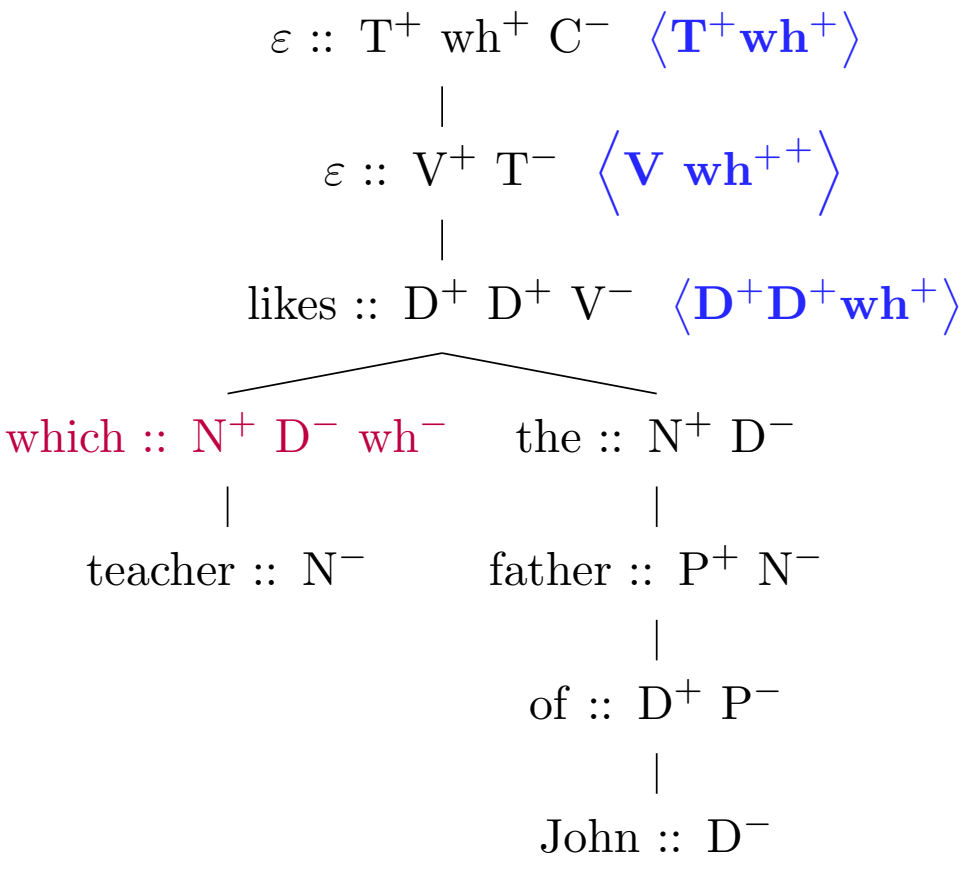
Restricting *move* [cont.]

Movement of a specifier is still ok.



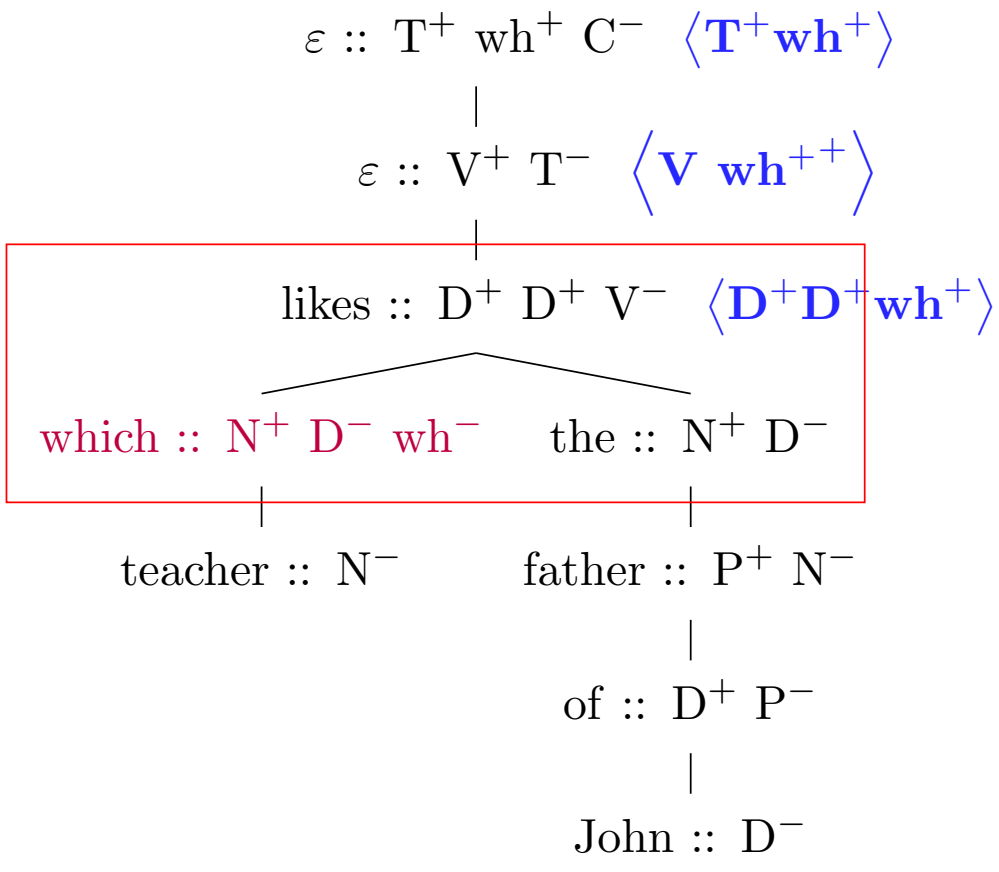
Restricting *move* [cont.]

Movement of a specifier is still ok.



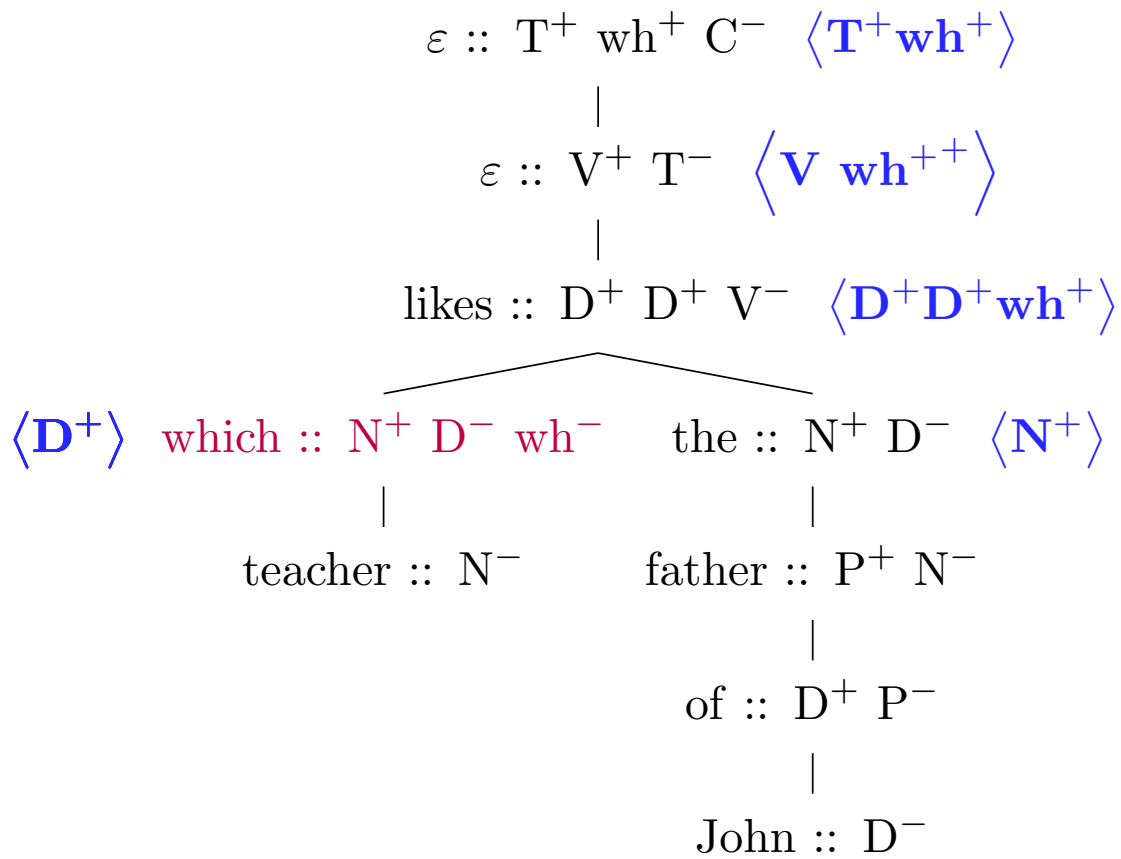
Restricting *move* [cont.]

Movement of a specifier is still ok.



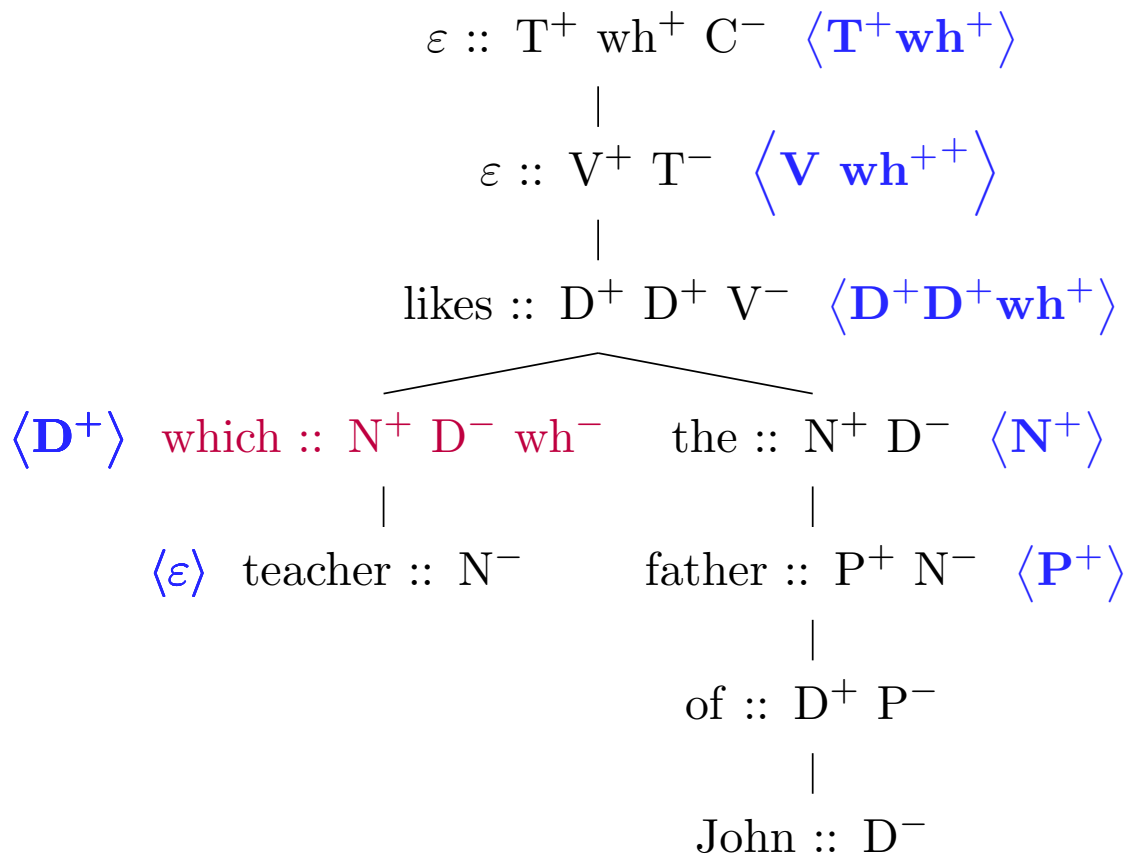
Restricting *move* [cont.]

Movement of a specifier is still ok.



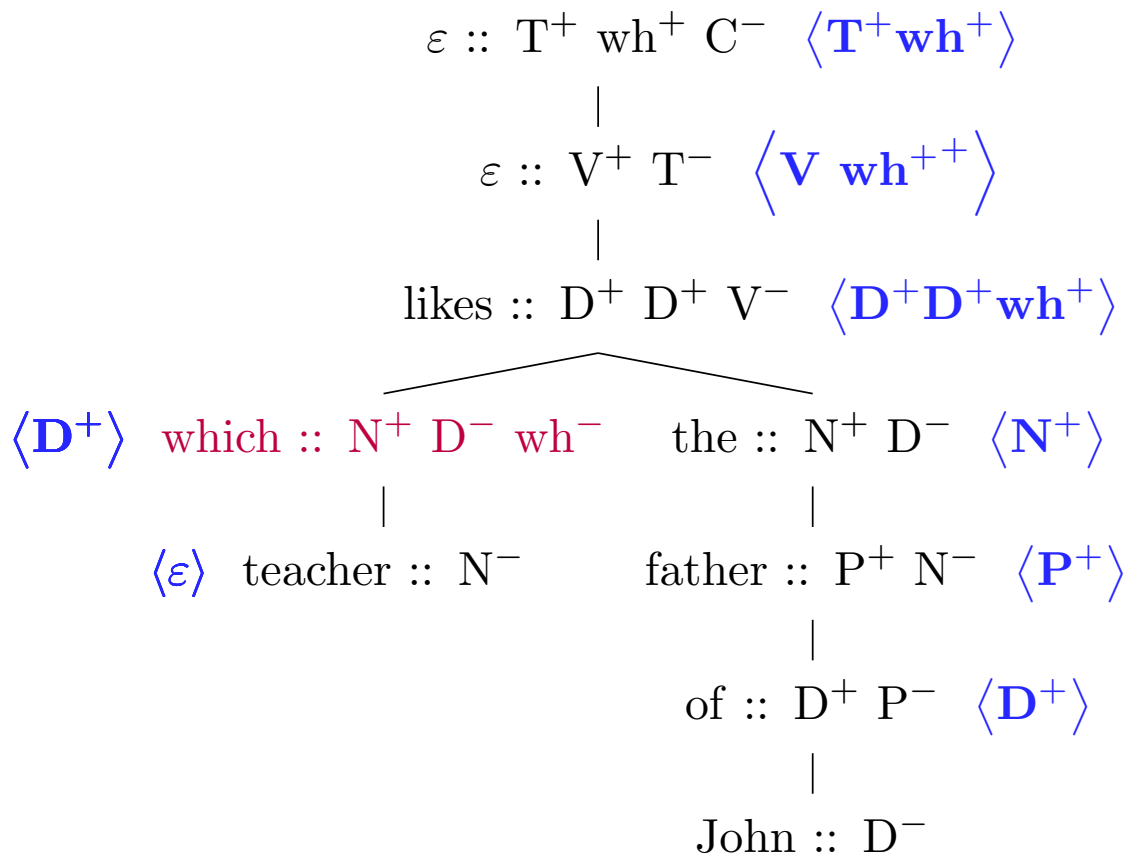
Restricting *move* [cont.]

Movement of a specifier is still ok.



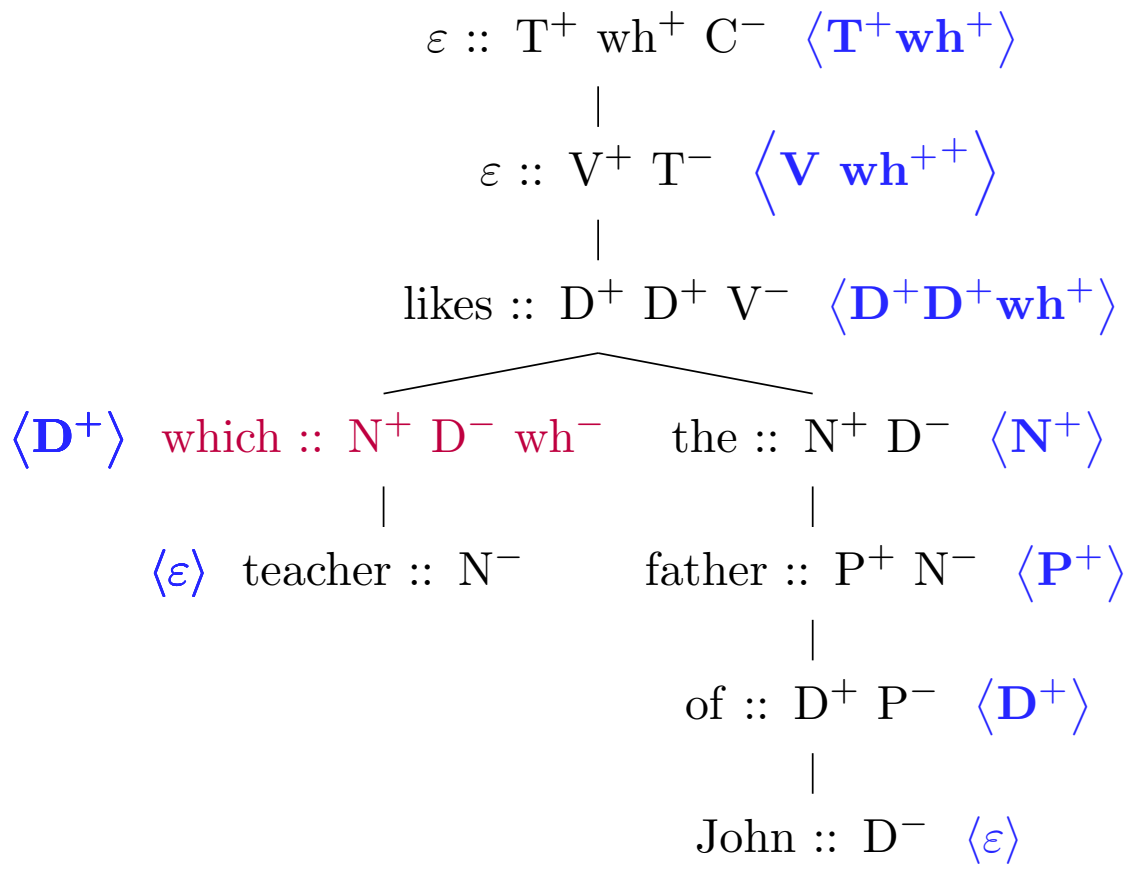
Restricting *move* [cont.]

Movement of a specifier is still ok.



Restricting *move* [cont.]

Movement of a specifier is still ok.



Interim Summary [2]

We are looking for a complexity upper bound for syntax...

The road so far

- MDEP[merge] $\subsetneq$ STA
- MDEP[merge,move] $\not\subseteq$ STA
- **But** MDEP[merge,move] $\subsetneq$ STA if we restrict move
  Movement constraints follow naturally: SpIC, CSC, ...

But syntax is not just about core operations!

Interim Summary [2]

We are looking for a complexity upper bound for syntax...

The road so far

- MDEP[merge] $\subsetneq$ STA
- MDEP[merge,move] $\not\subseteq$ STA
- **But** MDEP[merge,move] $\subsetneq$ STA if we restrict move
  Movement constraints follow naturally: SpIC, CSC, ...

But syntax is not just about core operations!

Licensing Conditions

Syntax is not just about Merge and Move...

NPI licensing

1a) *Every student said that the train **ever** arrives on time.
1b) **No** student said that the train **ever** arrives on time.

Principle A

2a) *John said that **Mary** likes **himself**.
2b) John said that **Mary** likes **herself**.

Graf and Shafiei (2019)

Licensing conditions are (sub)regular over c-command strings.

Licensing Conditions

Syntax is not just about Merge and Move...

NPI licensing

1a) *Every student said that the train **ever** arrives on time.
1b) **No** student said that the train **ever** arrives on time.

Principle A

2a) *John said that **Mary** likes **himself**.
2b) John said that **Mary** likes **herself**.

Graf and Shafiei (2019)

Licensing conditions are (sub)regular over c-command strings.

Licensing Conditions

Syntax is not just about Merge and Move...

NPI licensing

1a) *Every student said that the train **ever** arrives on time.
1b) **No** student said that the train **ever** arrives on time.

Principle A

2a) *John said that **Mary** likes **himself**.
2b) John said that **Mary** likes **herself**.

Graf and Shafiei (2019)

Licensing conditions are (sub)regular over c-command strings.

Licensing Conditions

Syntax is not just about Merge and Move...

NPI licensing

1a) *Every student said that the train **ever** arrives on time.
1b) **No** student said that the train **ever** arrives on time.

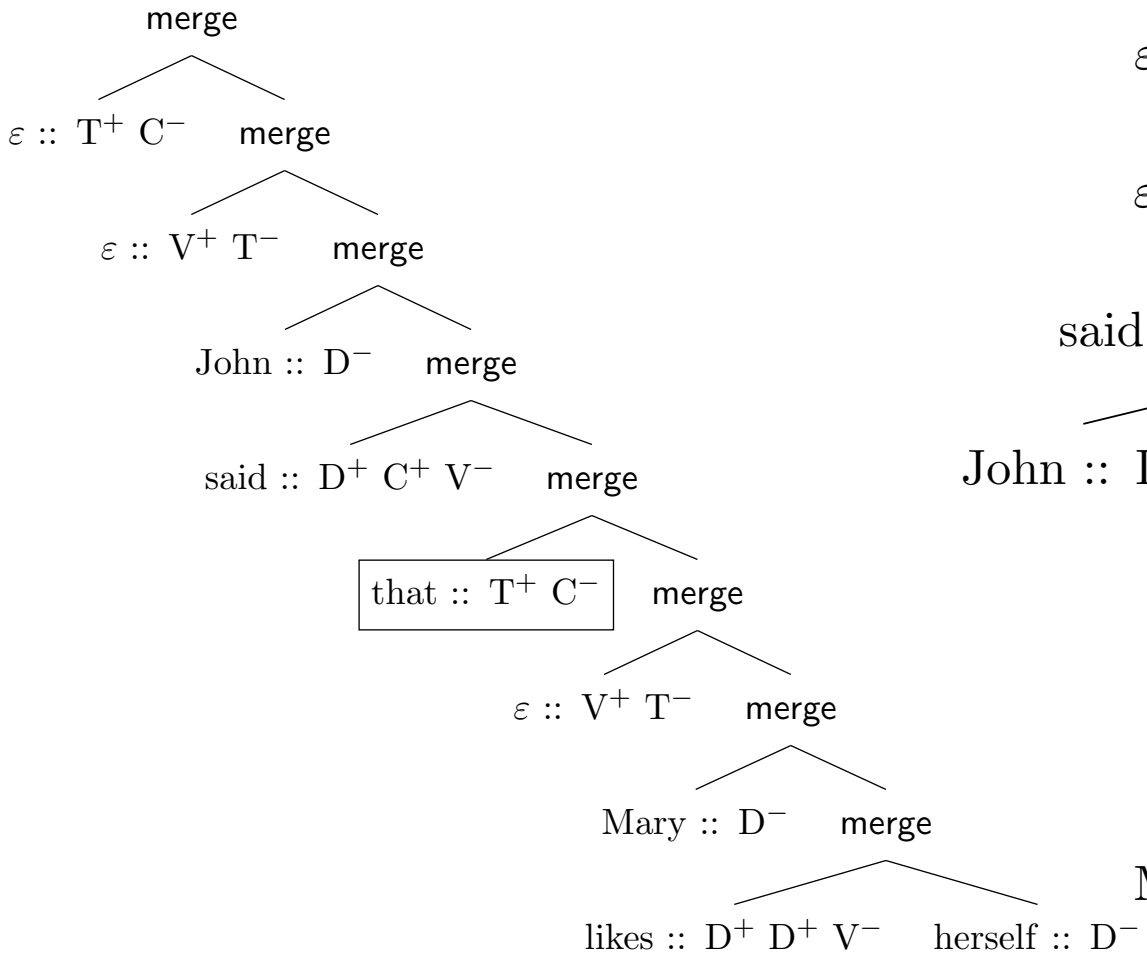
Principle A

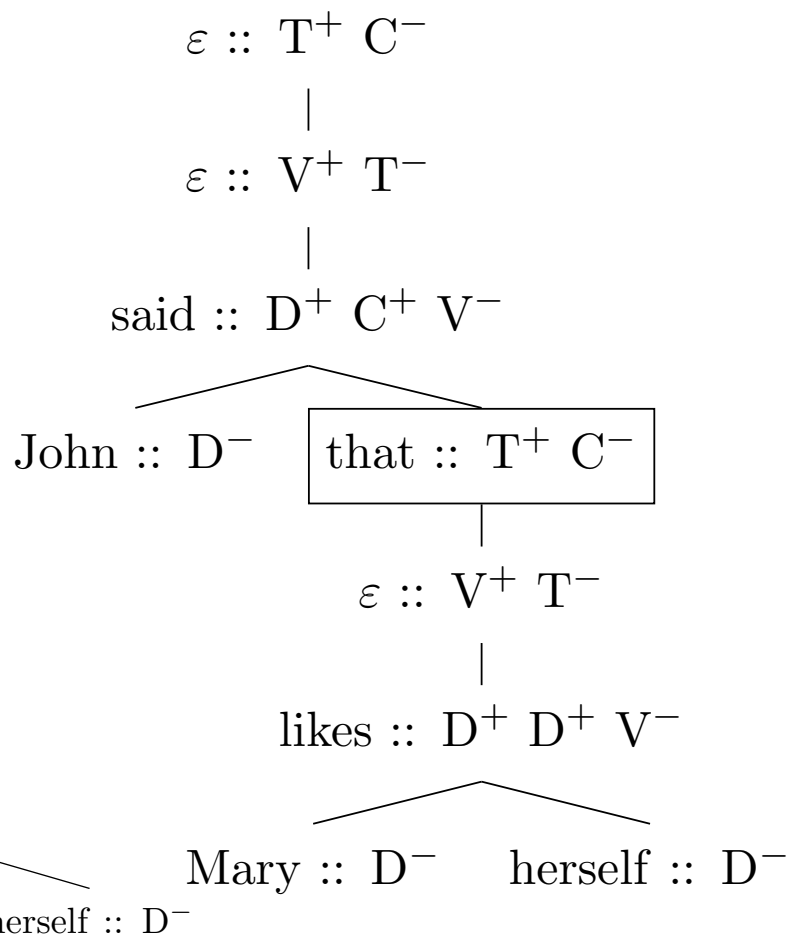
2a) *John said that **Mary** likes **himself**.
2b) John said that **Mary** likes **herself**.

Graf and Shafiei (2019)

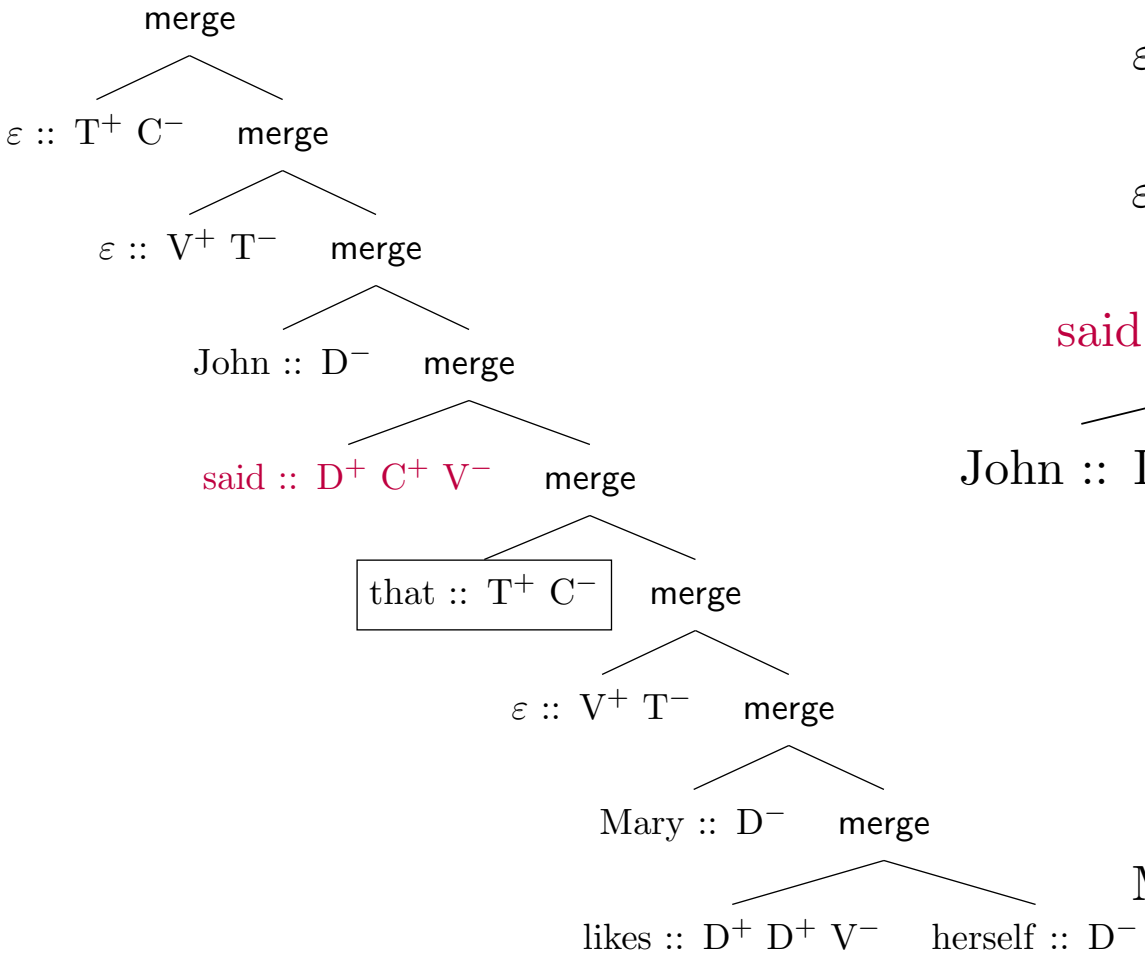
Licensing conditions are (sub)regular over c-command strings.

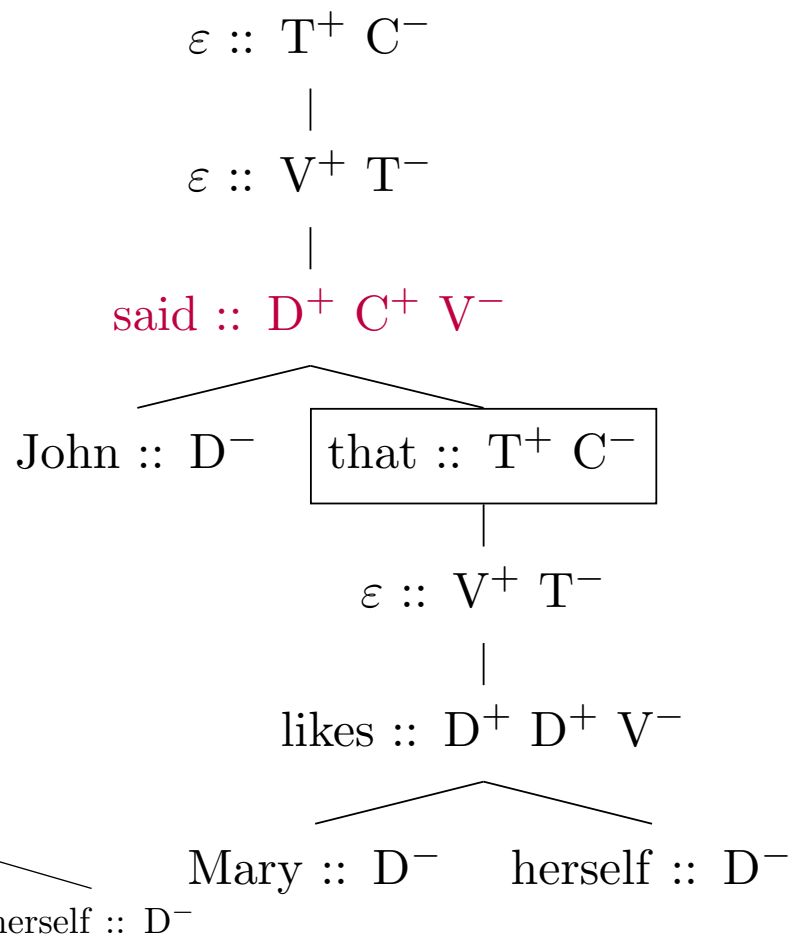
Dependency Trees and C-Command



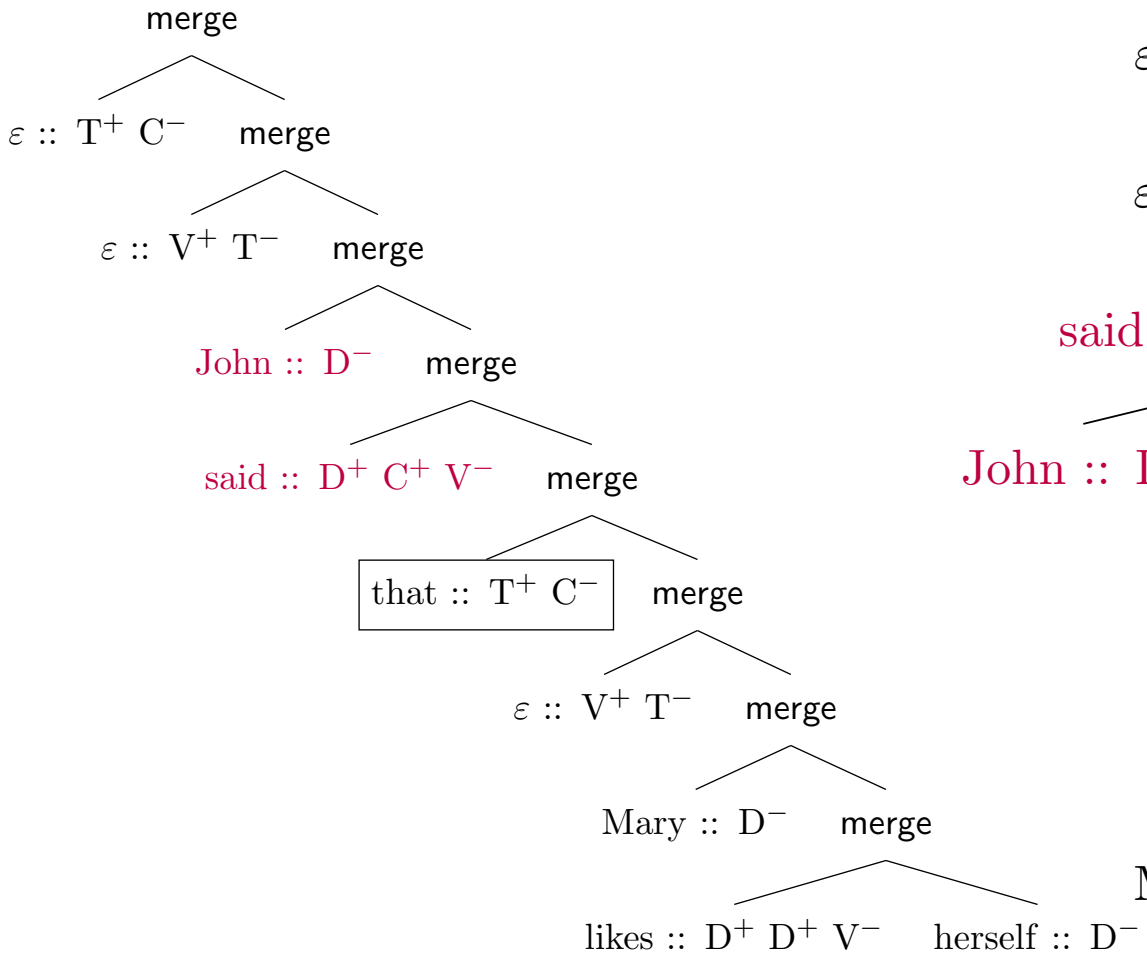


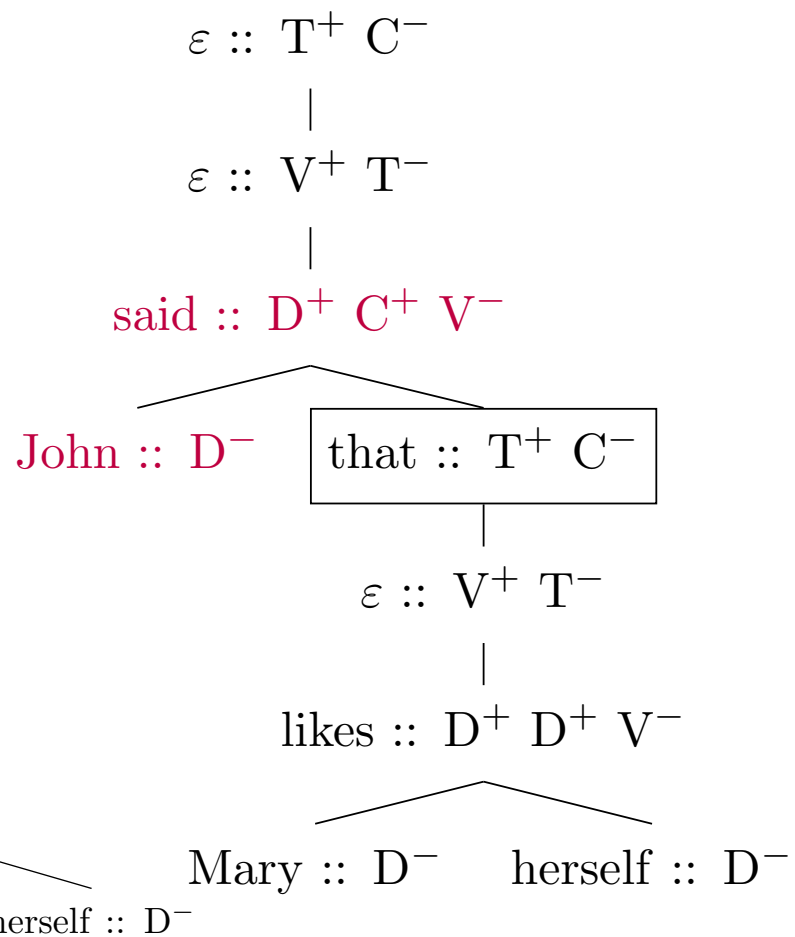
Dependency Trees and C-Command



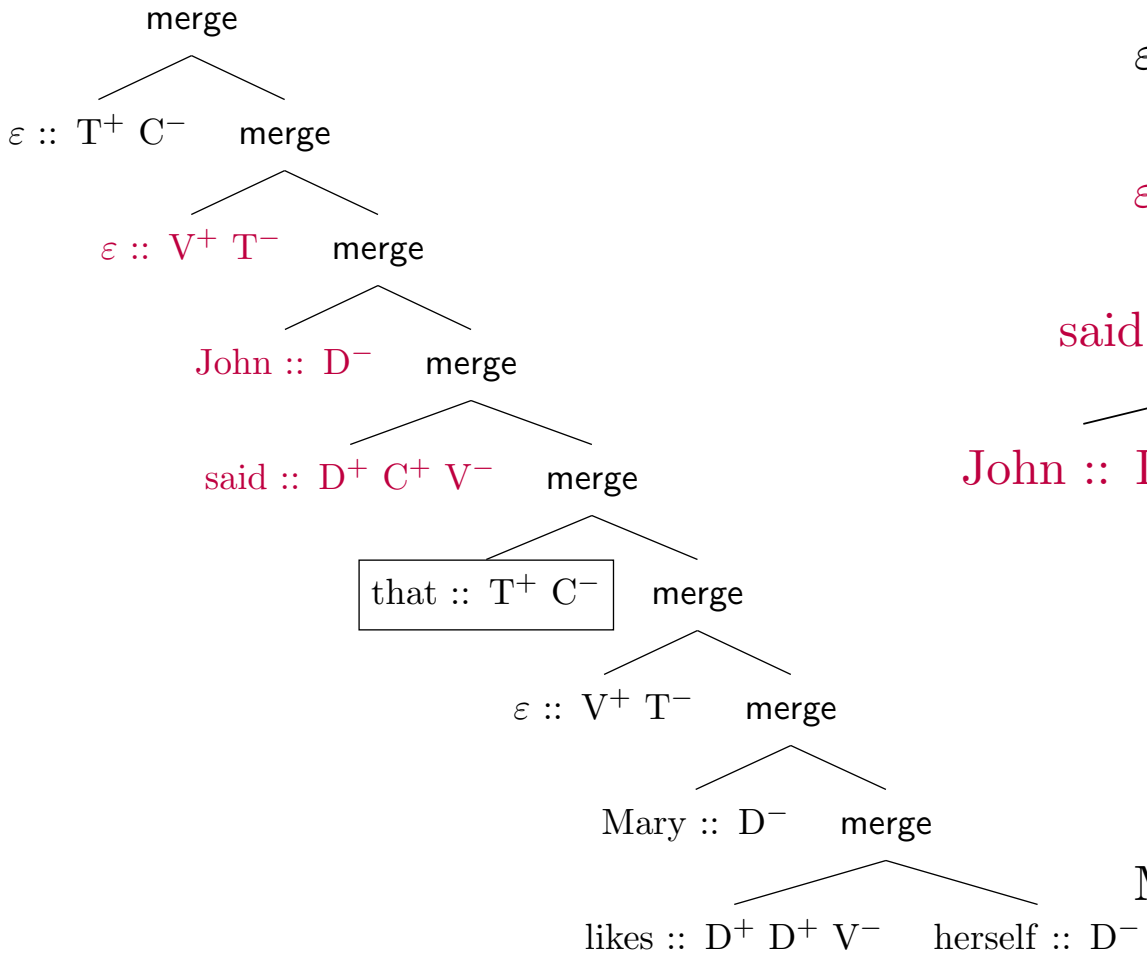


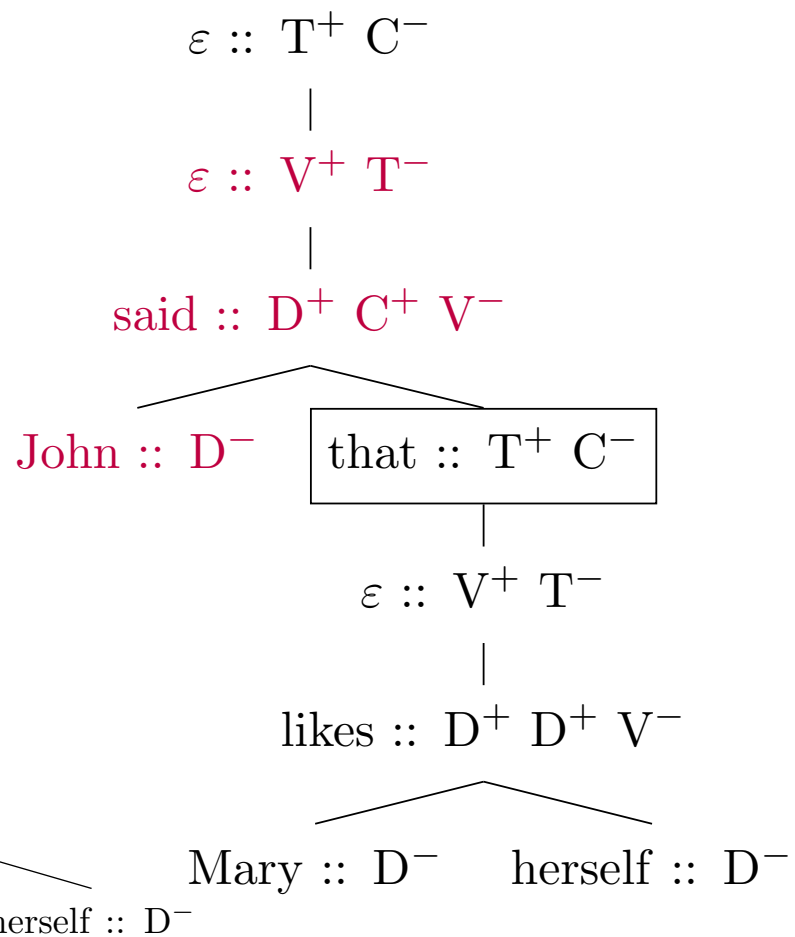
Dependency Trees and C-Command



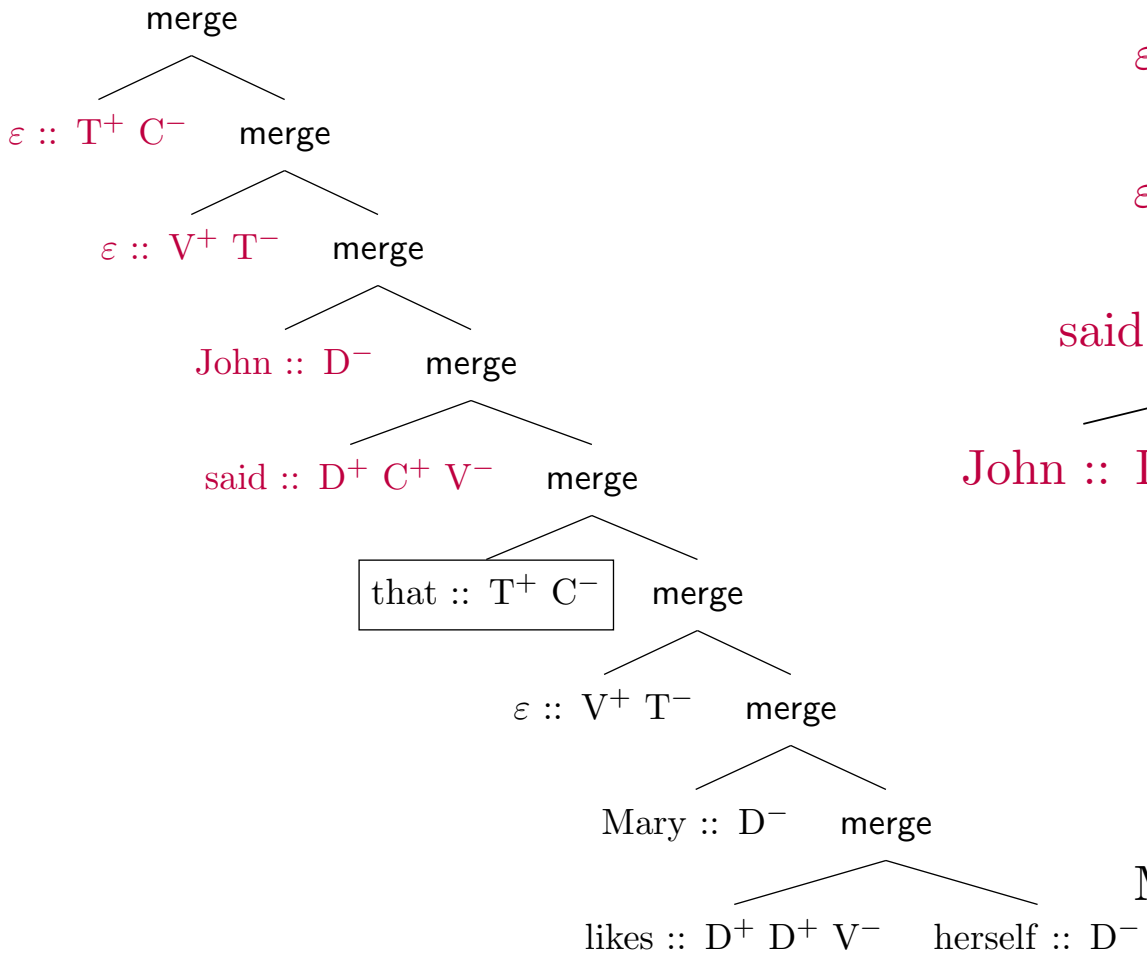


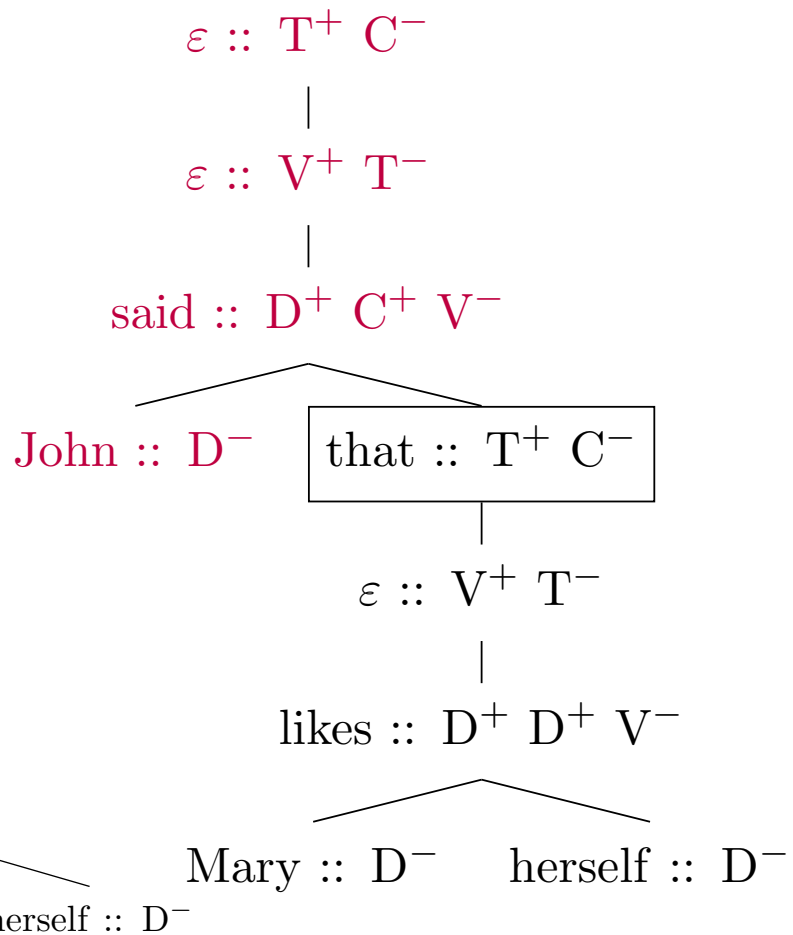
Dependency Trees and C-Command





Dependency Trees and C-Command

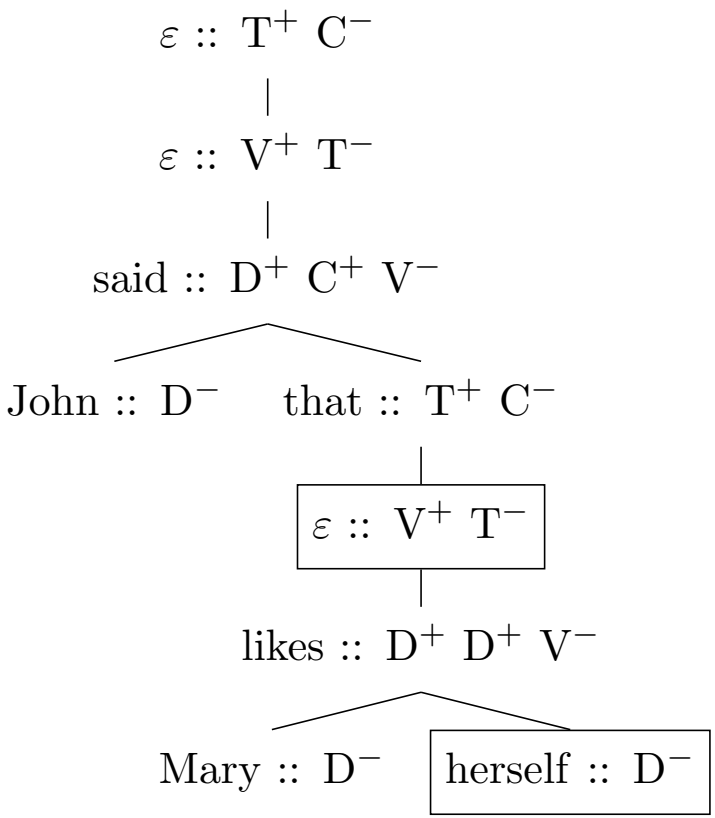




Enforcing Principle A with an STA

Principle A

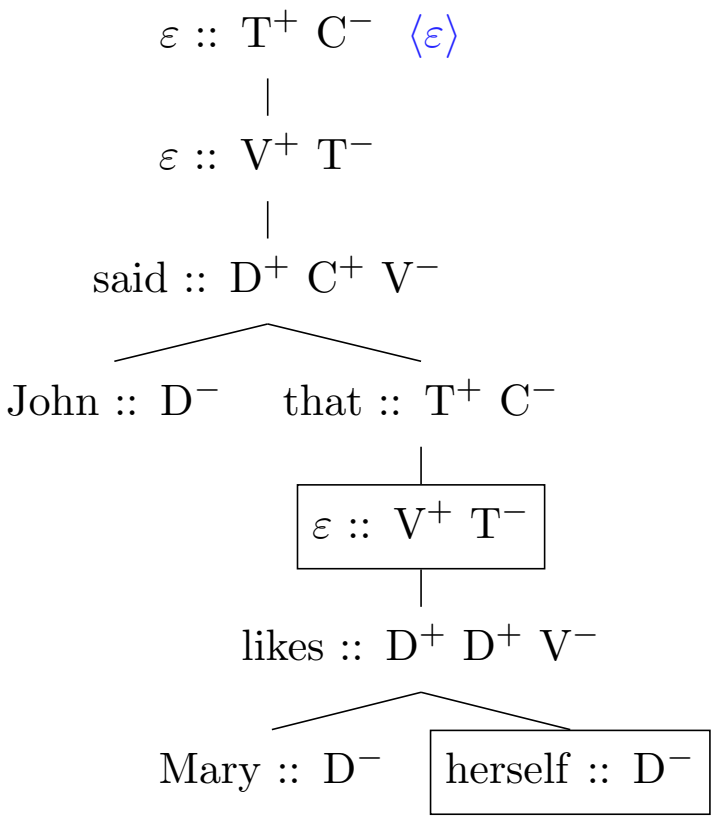
Reflexives must be bound within their binding domain (e.g. TP).



Enforcing Principle A with an STA

Principle A

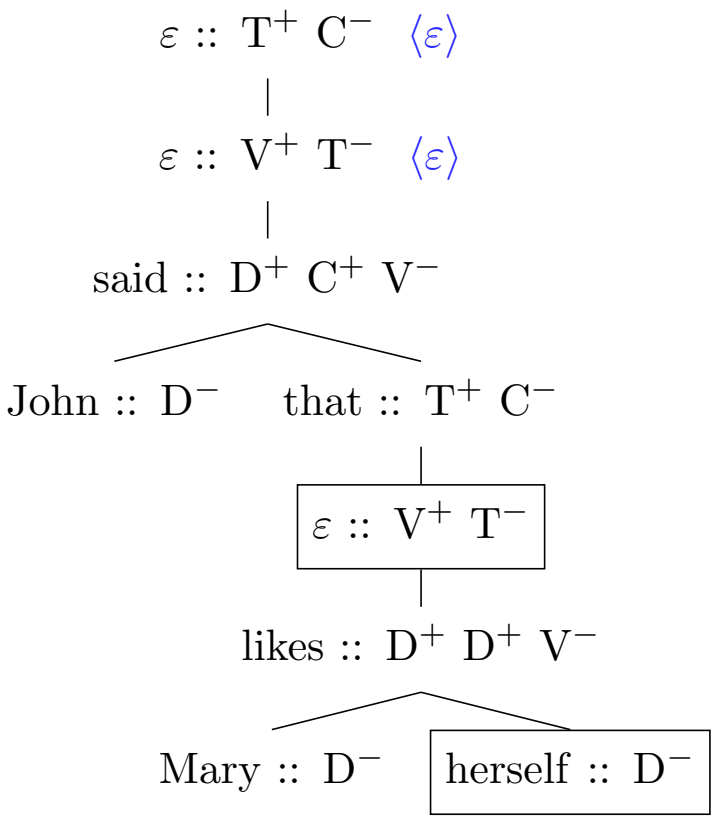
Reflexives must be bound within their binding domain (e.g. TP).



Enforcing Principle A with an STA

Principle A

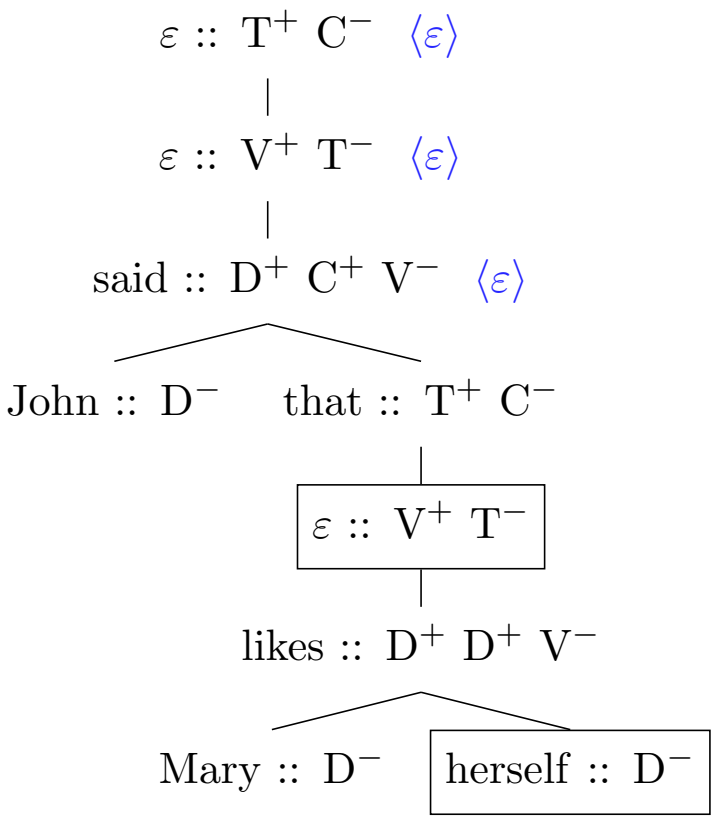
Reflexives must be bound within their binding domain (e.g. TP).



Enforcing Principle A with an STA

Principle A

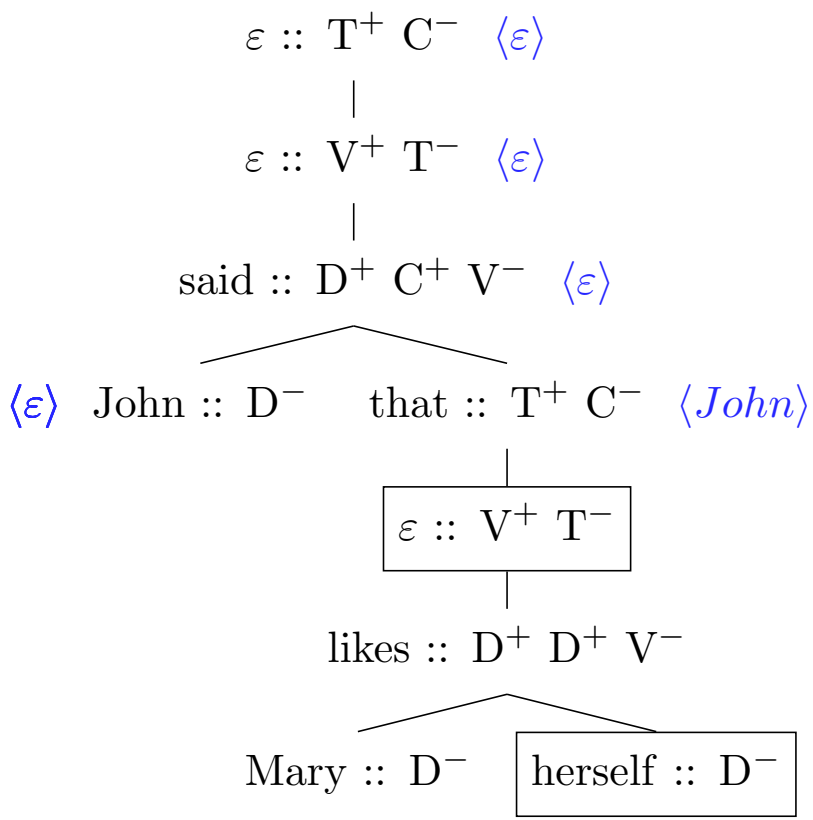
Reflexives must be bound within their binding domain (e.g. TP).



Enforcing Principle A with an STA

Principle A

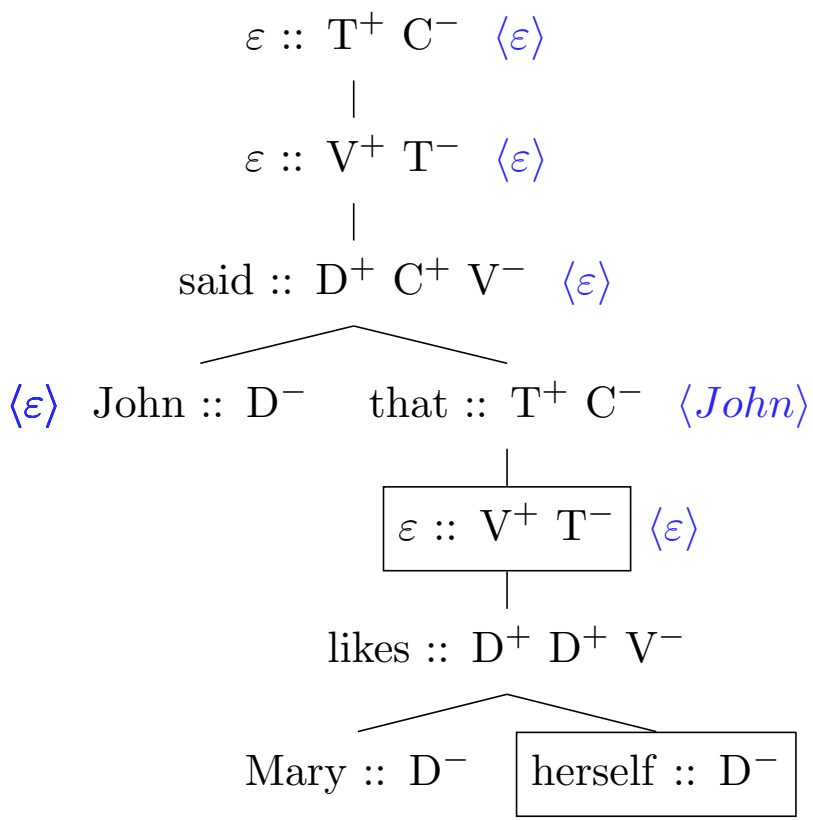
Reflexives must be bound within their binding domain (e.g. TP).



Enforcing Principle A with an STA

Principle A

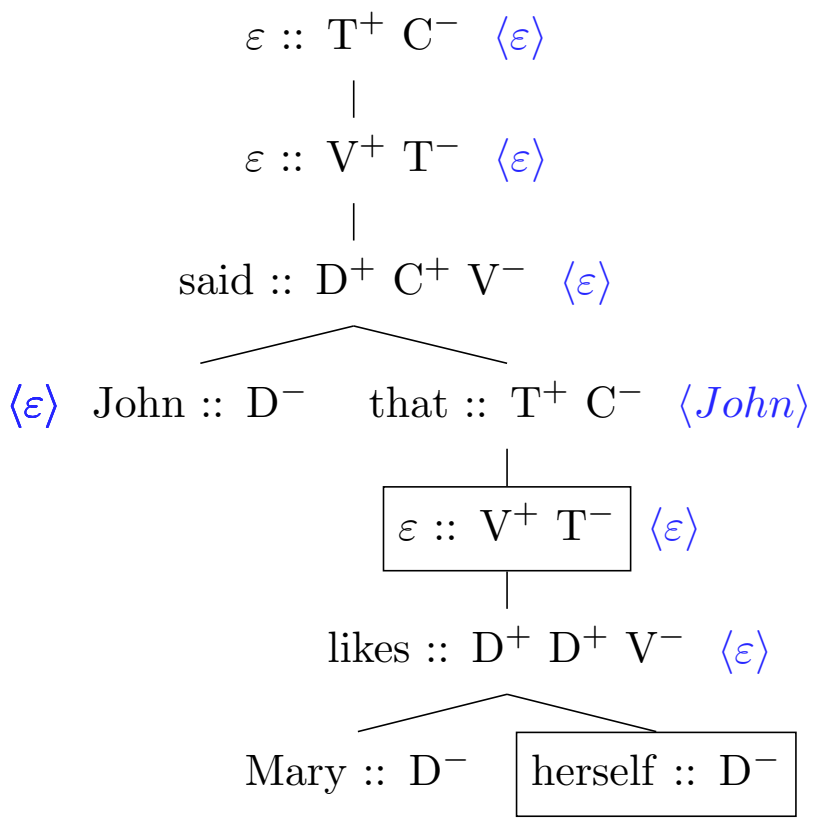
Reflexives must be bound within their binding domain (e.g. TP).



Enforcing Principle A with an STA

Principle A

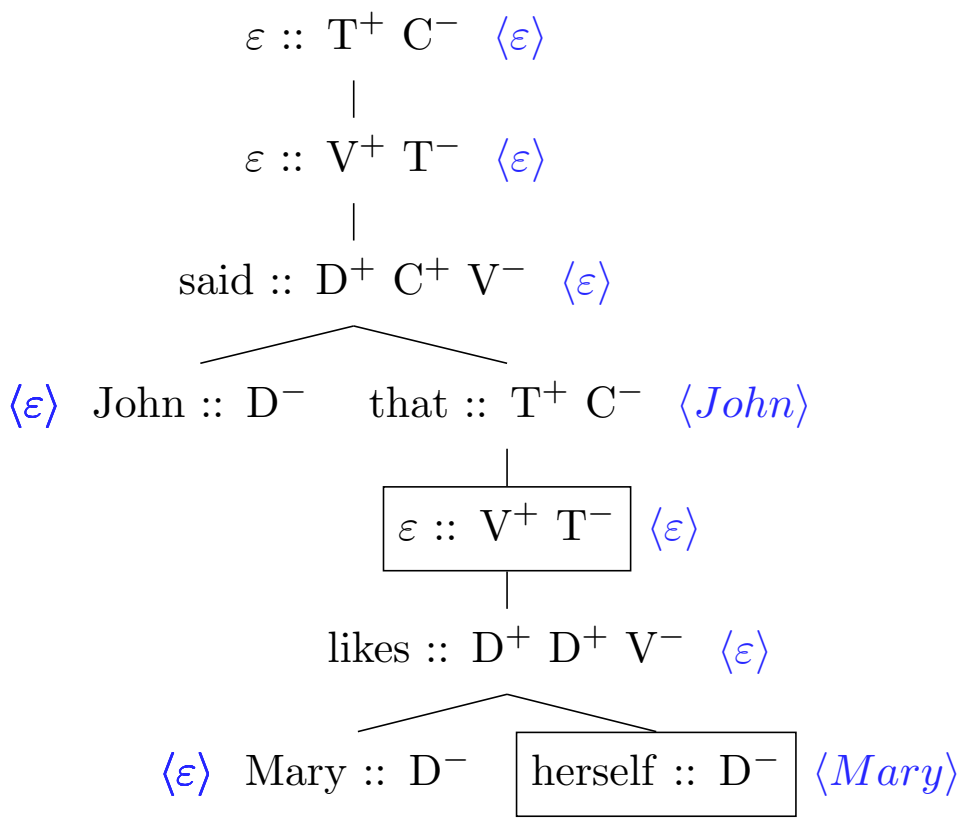
Reflexives must be bound within their binding domain (e.g. TP).



Enforcing Principle A with an STA

Principle A

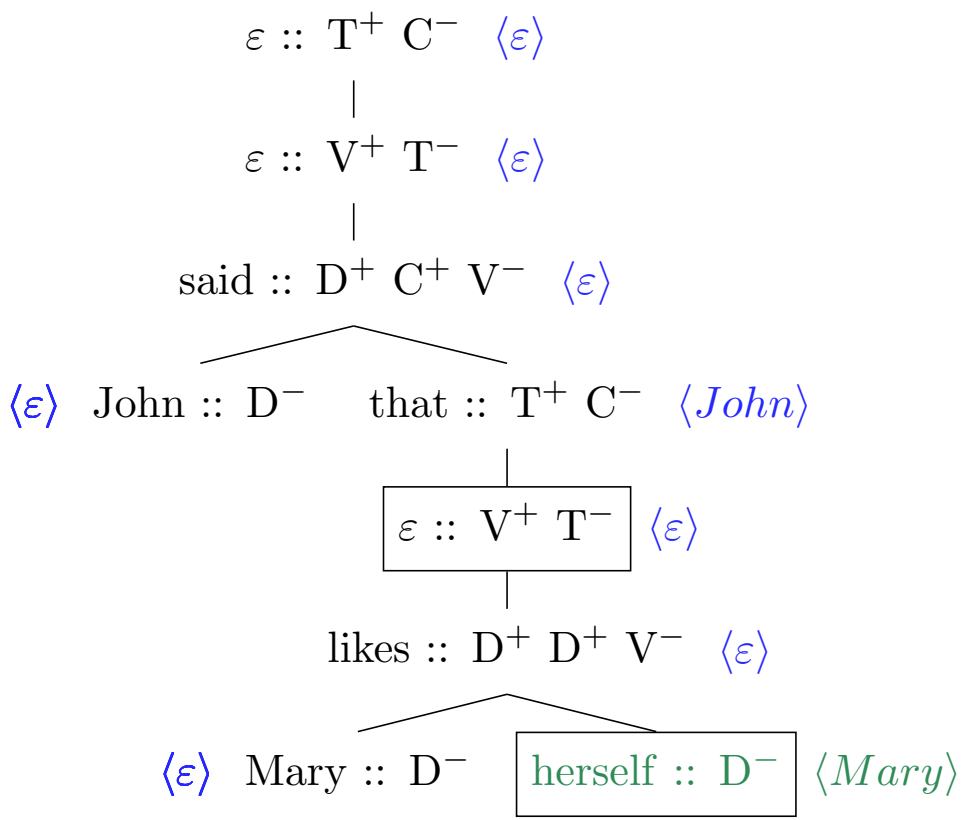
Reflexives must be bound within their binding domain (e.g. TP).



Enforcing Principle A with an STA

Principle A

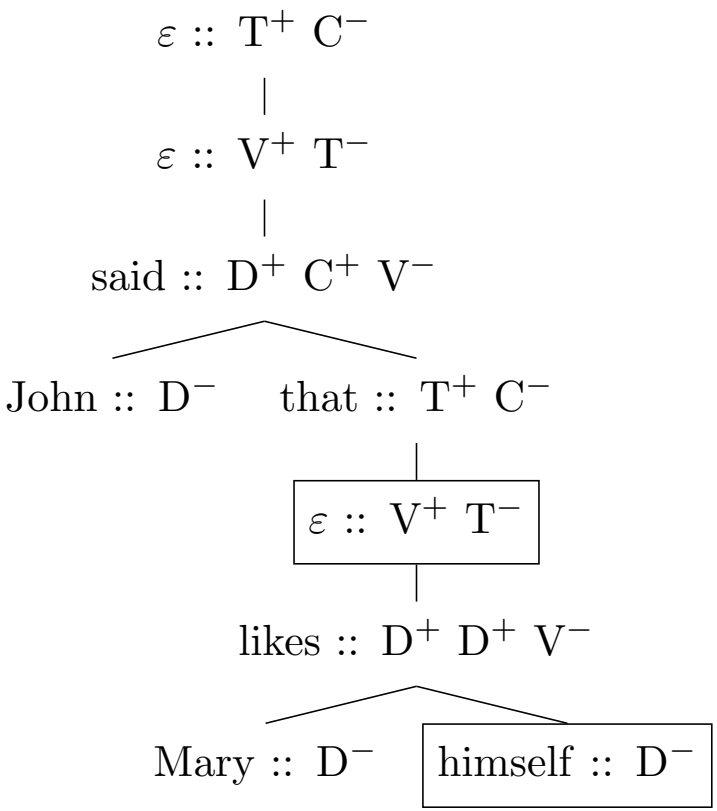
Reflexives must be bound within their binding domain (e.g. TP).



Example: Enforcing Principle A [cont.]

Principle A

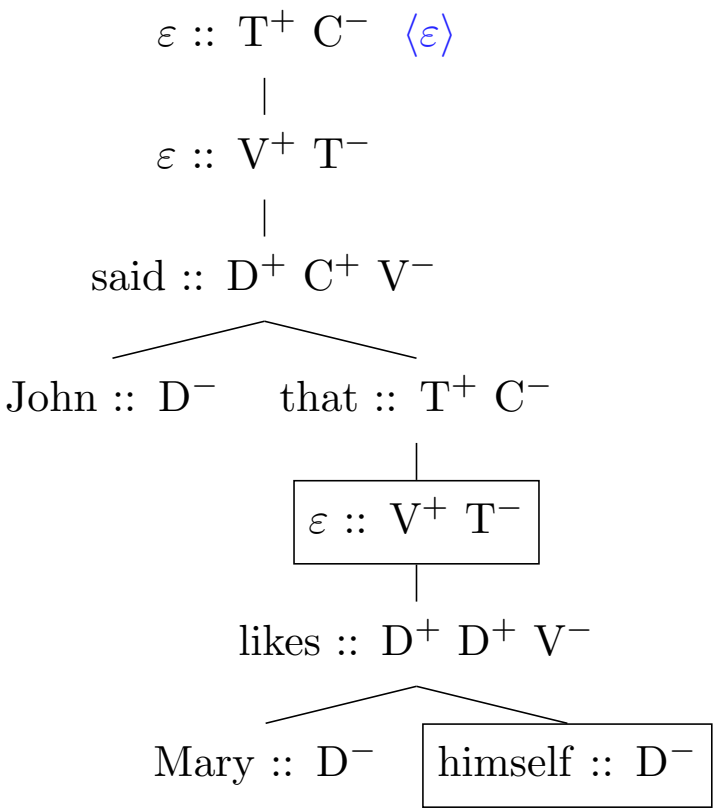
Reflexives must be bound within their binding domain (e.g. TP).



Example: Enforcing Principle A [cont.]

Principle A

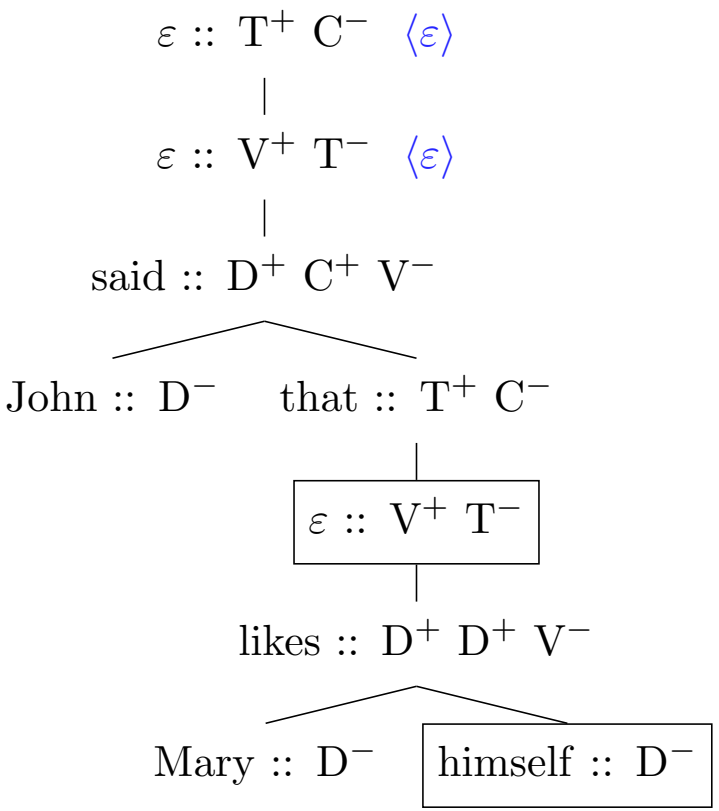
Reflexives must be bound within their binding domain (e.g. TP).



Example: Enforcing Principle A [cont.]

Principle A

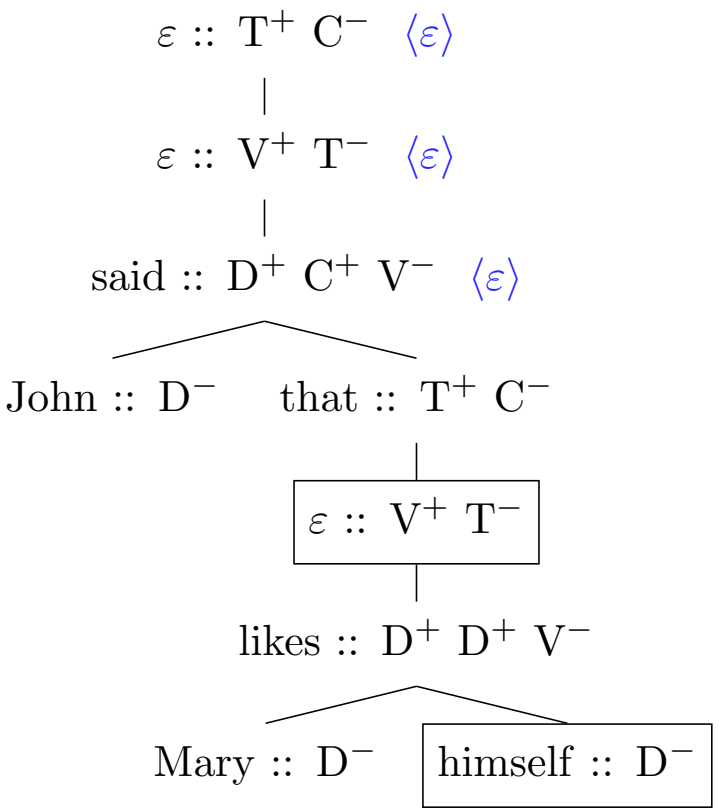
Reflexives must be bound within their binding domain (e.g. TP).



Example: Enforcing Principle A [cont.]

Principle A

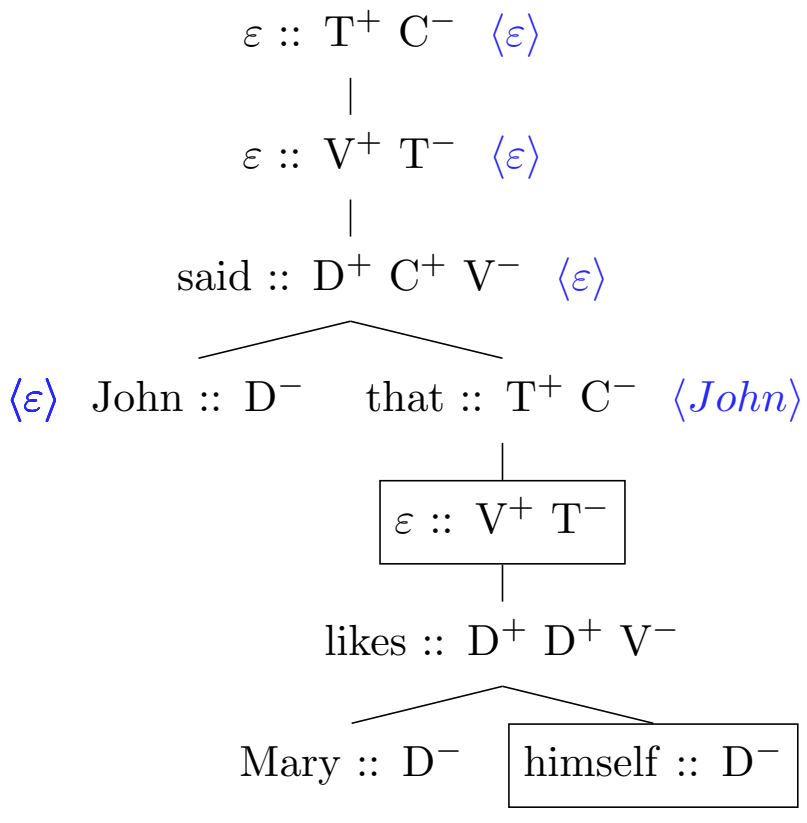
Reflexives must be bound within their binding domain (e.g. TP).



Example: Enforcing Principle A [cont.]

Principle A

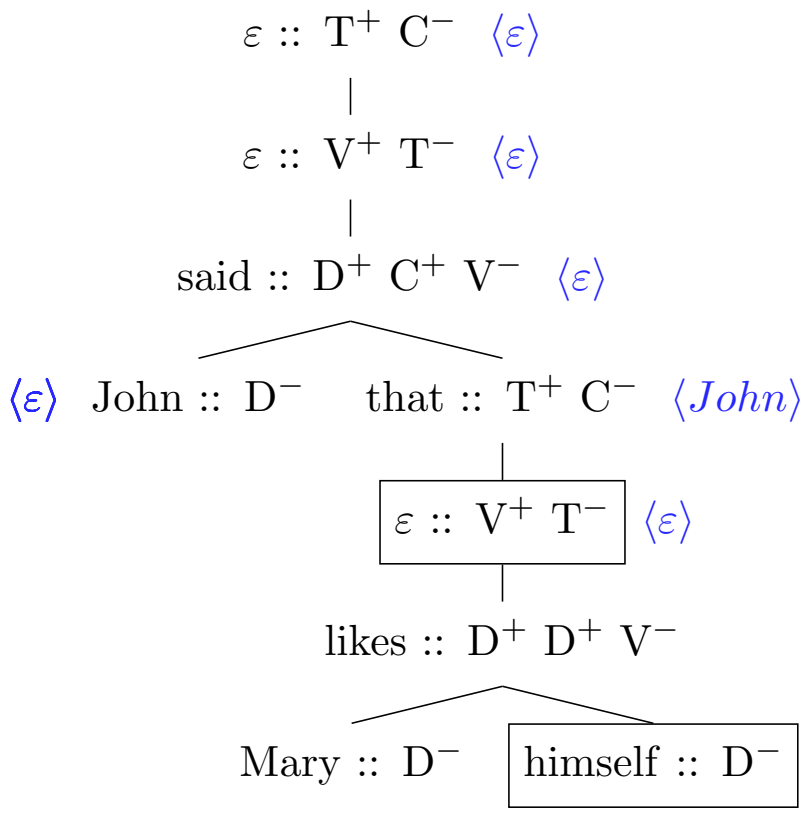
Reflexives must be bound within their binding domain (e.g. TP).



Example: Enforcing Principle A [cont.]

Principle A

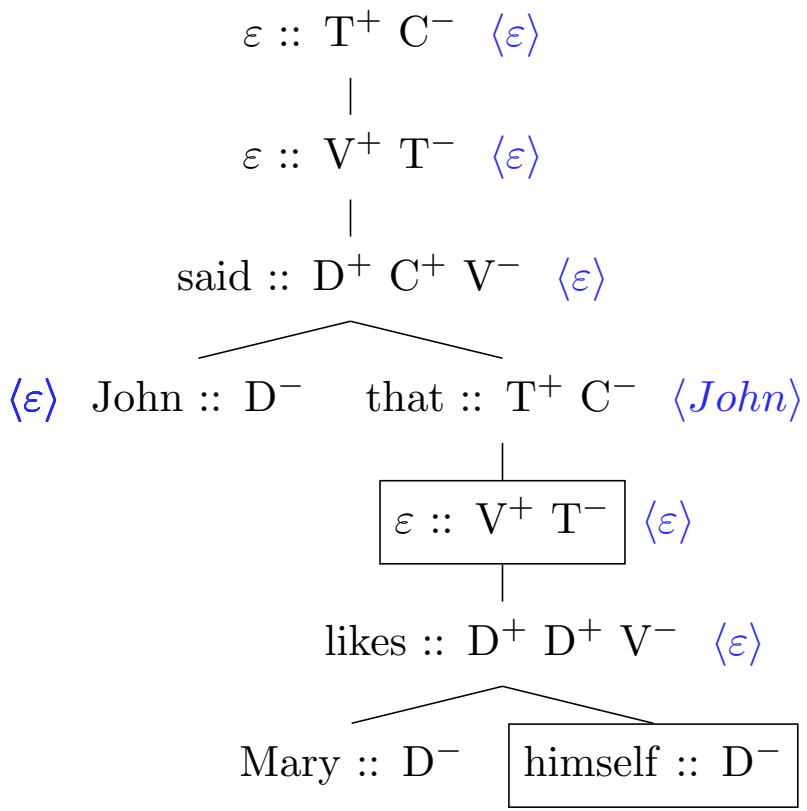
Reflexives must be bound within their binding domain (e.g. TP).



Example: Enforcing Principle A [cont.]

Principle A

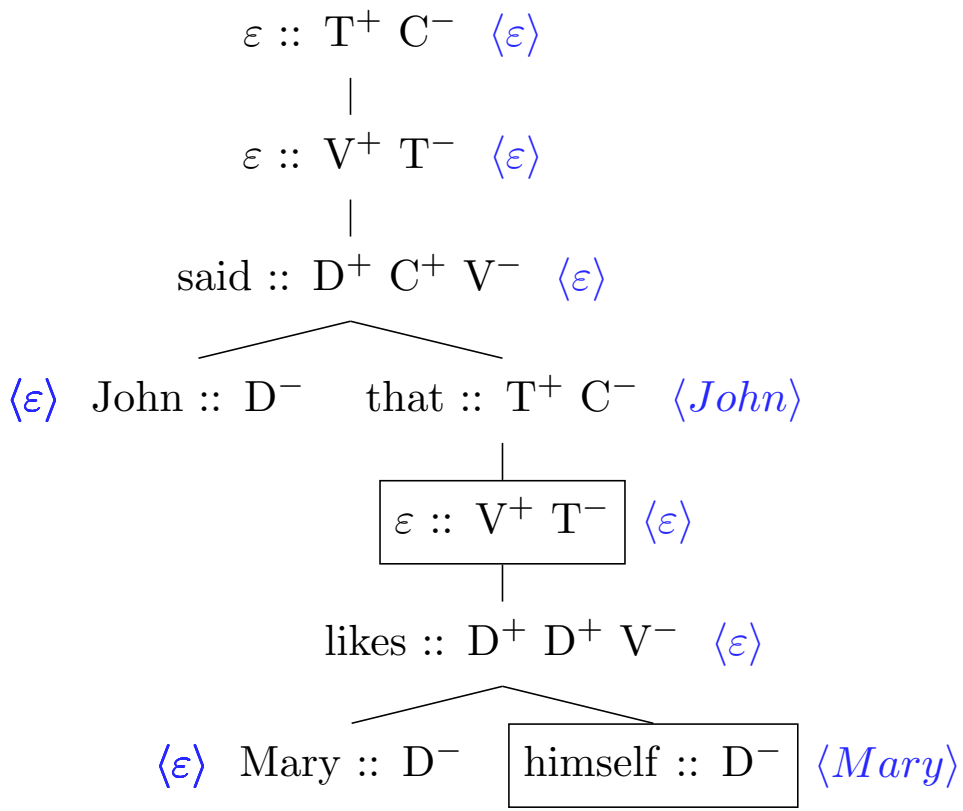
Reflexives must be bound within their binding domain (e.g. TP).



Example: Enforcing Principle A [cont.]

Principle A

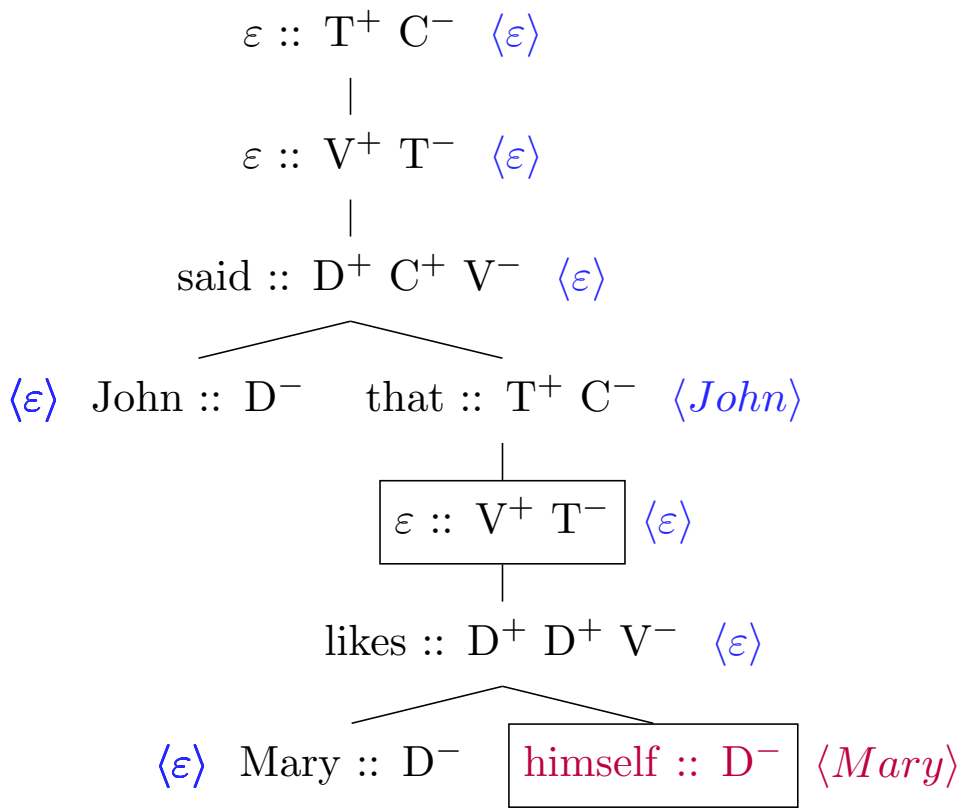
Reflexives must be bound within their binding domain (e.g. TP).



Example: Enforcing Principle A [cont.]

Principle A

Reflexives must be bound within their binding domain (e.g. TP).



Conclusion

STA as an upper bound for syntax

- MDEP[merge,move] $\subsetneq$ STA if we restrict move
- STA and C-Command Conditions

Merge, Move, Licensing enforced by the same machinery!

- MDEP a natural encoding of head-argument relations
- Naturalness of c-command
- STA-recognition $\approx$ syntactically motivated restrictions
- interaction of movement and licensing is expected

Conclusion

STA as an upper bound for syntax

- MDEP[merge,move] $\subsetneq$ STA if we restrict move
- STA and C-Command Conditions

Merge, Move, Licensing enforced by the same machinery!

- MDEP a natural encoding of head-argument relations
- Naturalness of c-command
- STA-recognition $\approx$ syntactically motivated restrictions
- interaction of movement and licensing is expected

Conclusion?

STA as a uniform upper bound. But:

- Too permissive: Enforce arbitrary regular constraints
- Too restrictive? Licensing + c-command...

Expanding the Core Results

- Movement + licensing
- Subcommand
- Adjunct Island Constraint, Coordinate Structure Constraint, ...
- MG derivation trees?
- Improving top-down parsing efficiency

Conclusion?

STA as a uniform upper bound. But:

- Too permissive: Enforce arbitrary regular constraints
- Too restrictive? Licensing + c-command...

Expanding the Core Results

- Movement + licensing
- Subcommand
- Adjunct Island Constraint, Coordinate Structure Constraint, ...
- MG derivation trees?
- Improving top-down parsing efficiency

$\langle Thank\ you! \rangle$

Acknowledgments I

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

References I

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

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

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, 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.

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.

References II

Jardine, Adam. 2016. Computationally, tone is different. *Phonology* URL `http://udel.edu/~ajardine/files/jardinemscomputationallytoneisdifferent.pdf`, to appear.

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.

Martens, Wim. 2006. *Static analysis of xml transformation- and schema languages*. Doctoral Dissertation, Hasselt University.

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

Michaelis, Jens. 2004. Observations on strict derivational minimalism. *Electronic Notes in Theoretical Computer Science* 53:192–209.

Shafiei, Nazila, and Thomas Graf. 2019. The subregular complexity of syntactic islands. Ms., Stony Brook University.

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.

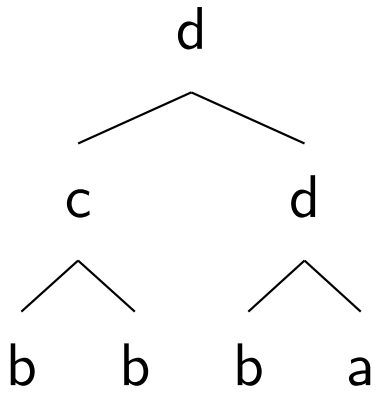
References III

Vu, Mai Ha. 2018. Towards a formal description of NPI-licensing patterns. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2018*, volume 1, 154–163. Article 17.

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.

The Spine of a Node

- Example: spine(a)

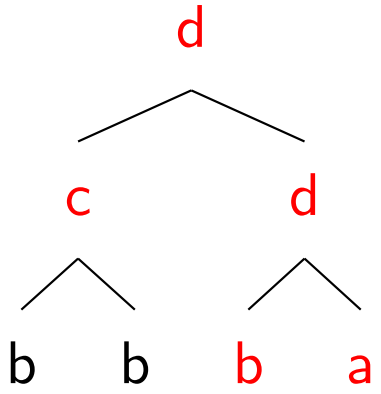


STAs and spine closure (Martens 2006)

A regular tree language L belongs to the class STA iff L is spine closed.

The Spine of a Node

- Example: spine(a)

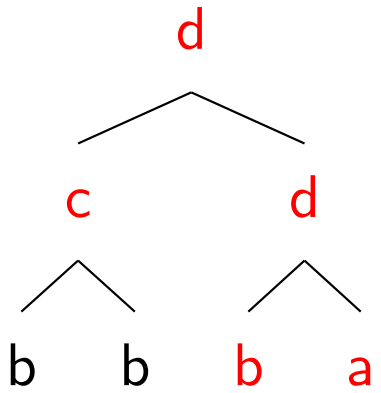


STAs and spine closure (Martens 2006)

A regular tree language L belongs to the class STA iff L is spine closed.

The Spine of a Node

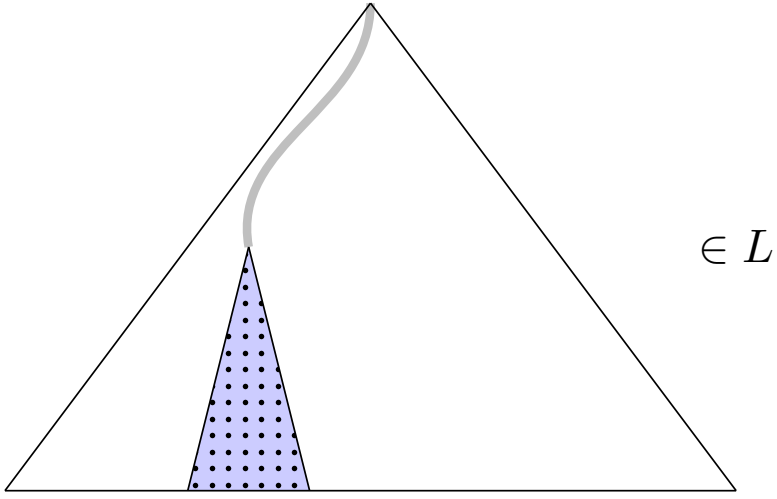
- Example: spine(a)



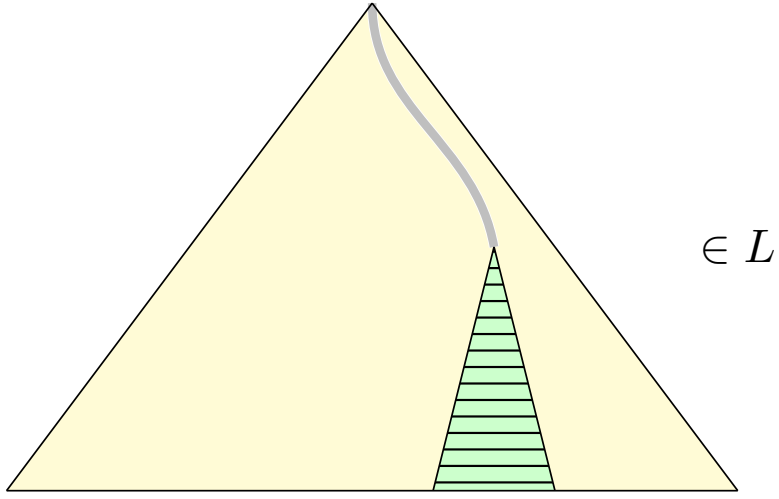
STAs and spine closure (Martens 2006)

A regular tree language L belongs to the class STA iff L is spine closed.

Spine Closure

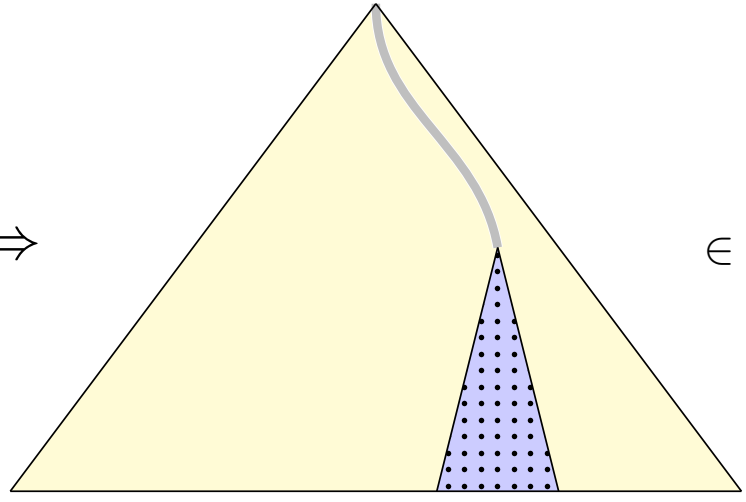


$\in L$



$\in L$

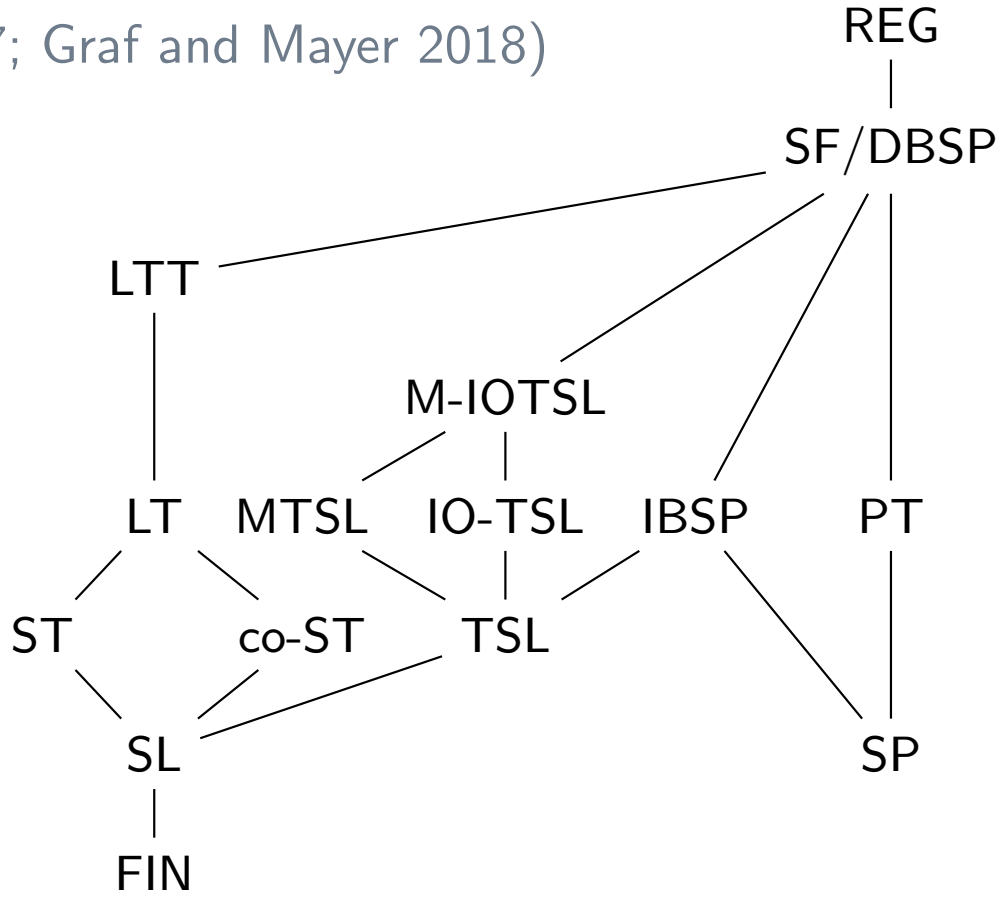
$\Rightarrow$



$\in L$

Subregular Complexity in Phonology

- **Subregular** phonology has proved to be a fruitful enterprise (Heinz et al. 2011; Chandlee 2014; Jardine 2016; McMullin 2016; Graf 2017; Graf and Mayer 2018)



C-Strings and Spines

Graf and Shafiei (2019)

C-command conditions as subregular **c-string** constraints.

Observation

$\text{spine}(u) \approx \text{c-string}(u)$

Theorem

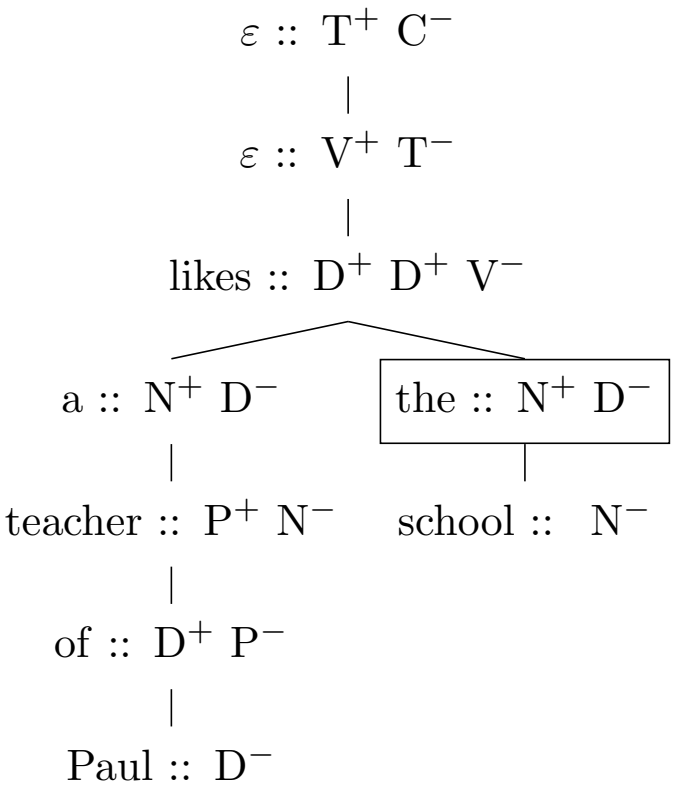
Every regular c-string constraint can be enforced by an STA.

c-string(the :: N^+ D^-) := ε :: T^+ wh^+ C^- ↑ does :: V^+ T^- ↑ like :: D^+ D^+ V^- ↑ a :: N^+ D^- the :: N^+ D^-

C-Strings and Spines

Graf and Shafiei (2019)

C-command conditions as subregular **c-string** constraints.



Observation

$\text{spine}(u) \approx \text{c-string}(u)$

Theorem

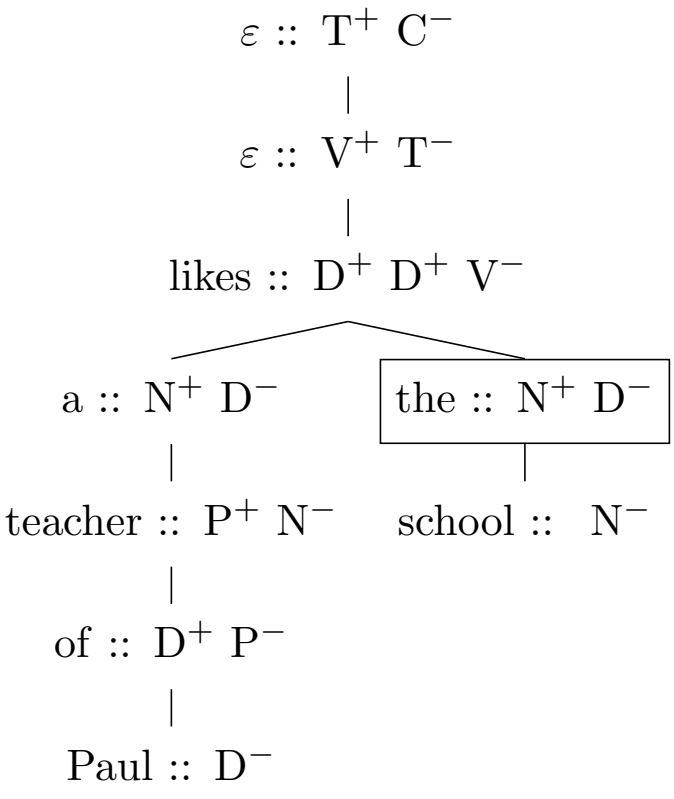
Every regular c-string constraint can be enforced by an STA.

$\text{c-string}(\text{the :: N}^{+}\ \text{D}^{-}) := \varepsilon :: \text{T}^{+}\ \text{wh}^{+}\ \text{C}^{-} \uparrow \text{does :: V}^{+}\ \text{T}^{-} \uparrow \text{like :: D}^{+}\ \text{D}^{+}\ \text{V}^{-} \uparrow \text{a :: N}^{+}\ \text{D}^{-}\ \text{the :: N}^{+}\ \text{D}^{-}$

C-Strings and Spines

Graf and Shafiei (2019)

C-command conditions as subregular **c-string** constraints.



Observation

$\text{spine}(u) \approx \text{c-string}(u)$

Theorem

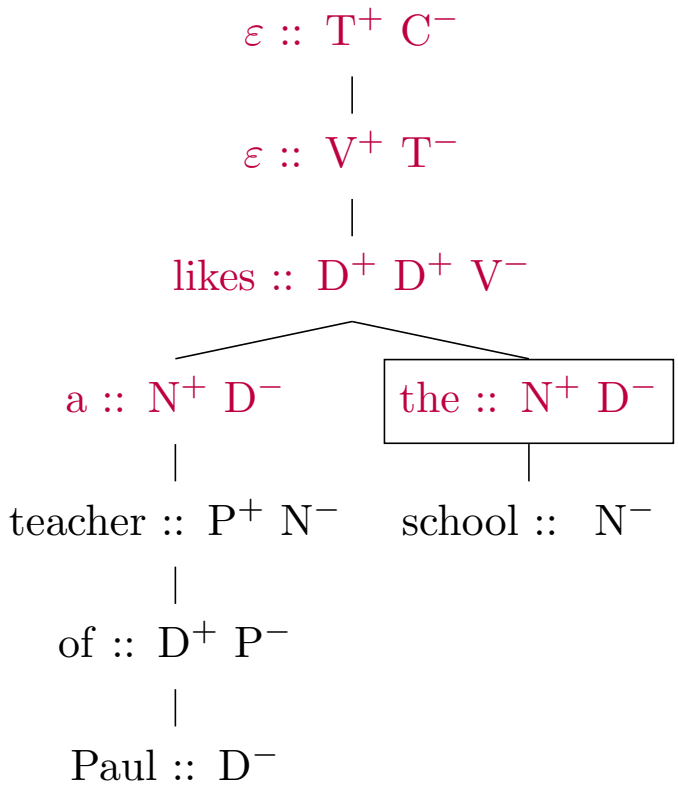
Every regular c-string constraint can be enforced by an STA.

$\text{c-string}(\text{the} :: \text{N}^+\ \text{D}^-) := \varepsilon :: \text{T}^+\ \text{wh}^+\ \text{C}^- \uparrow \text{does} :: \text{V}^+\ \text{T}^- \uparrow \text{like} :: \text{D}^+\ \text{D}^+\ \text{V}^- \uparrow \text{a} :: \text{N}^+\ \text{D}^-\ \text{the} :: \text{N}^+\ \text{D}^-$

C-Strings and Spines

Graf and Shafiei (2019)

C-command conditions as subregular **c-string** constraints.



Observation

spine(u) $\approx$ c-string(u)

Theorem

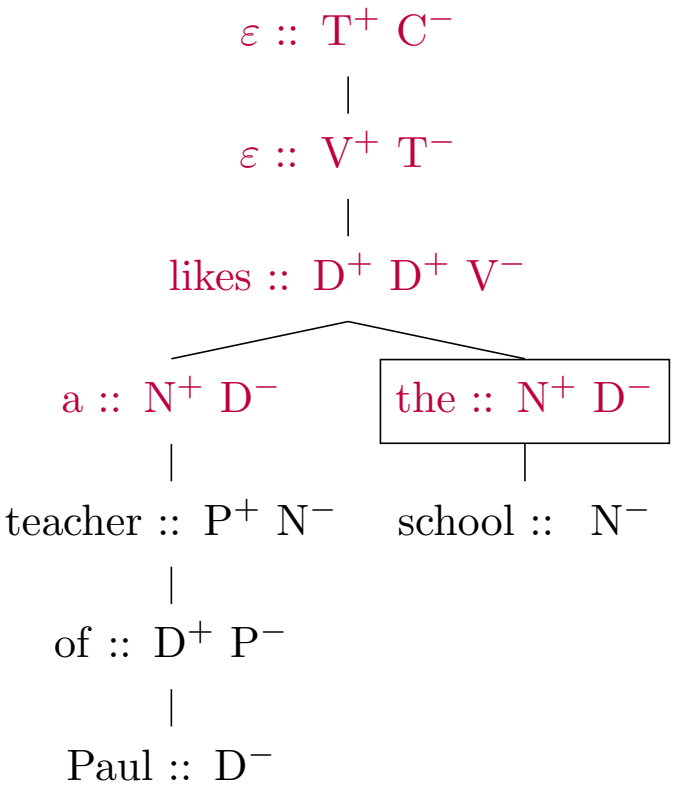
Every regular c-string constraint can be enforced by an STA.

c-string(the :: N^+ D^-) := $\varepsilon :: T^+$ wh^+ C^- ↑ does :: V^+ T^- ↑ like :: D^+ D^+ V^- ↑ a :: N^+ D^- the :: N^+ D^-

C-Strings and Spines

Graf and Shafiei (2019)

C-command conditions as subregular **c-string** constraints.



Observation

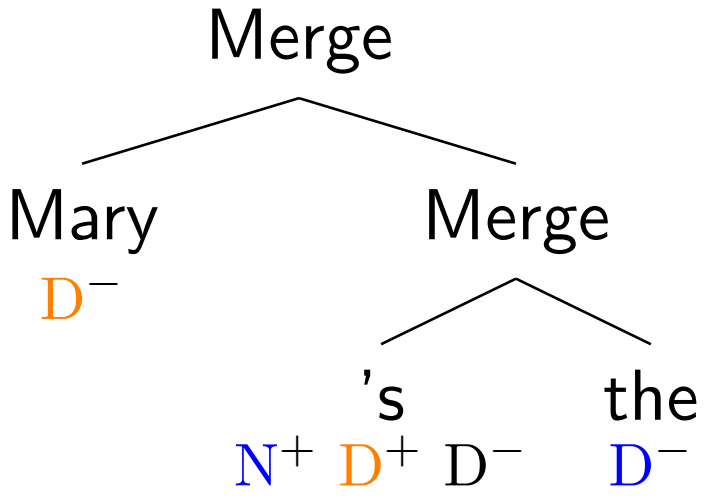
spine(u) $\approx$ c-string(u)

Theorem

Every regular c-string constraint can be enforced by an STA.

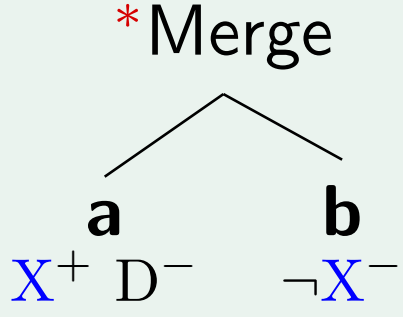
c-string(the :: N^+ D^-) := $\varepsilon :: \mathrm{T}^+$ wh^+ C^- ↑ does :: V^+ T^- ↑ like :: D^+ D^+ V^- ↑ a :: N^+ D^- the :: N^+ D^-

Merge is SL (Graf 2012)

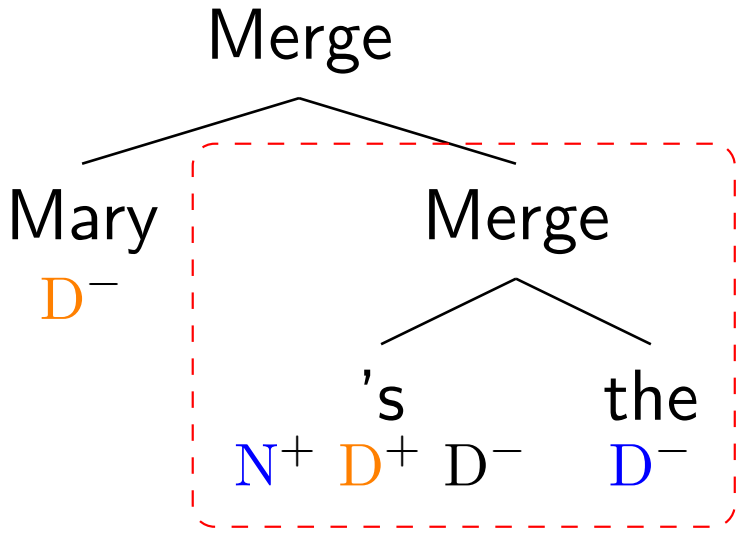


SL constraints on Merge

- We lift constraints from string n-grams to tree n-grams
- We get SL constraints over subtrees.

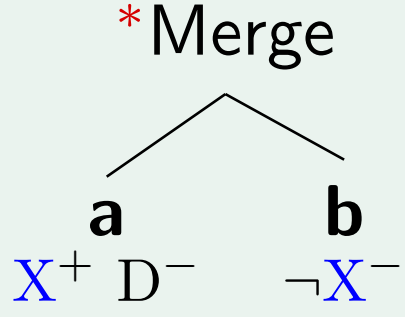


Merge is SL (Graf 2012)



SL constraints on Merge

- We lift constraints from string n-grams to tree n-grams
- We get SL constraints over subtrees.

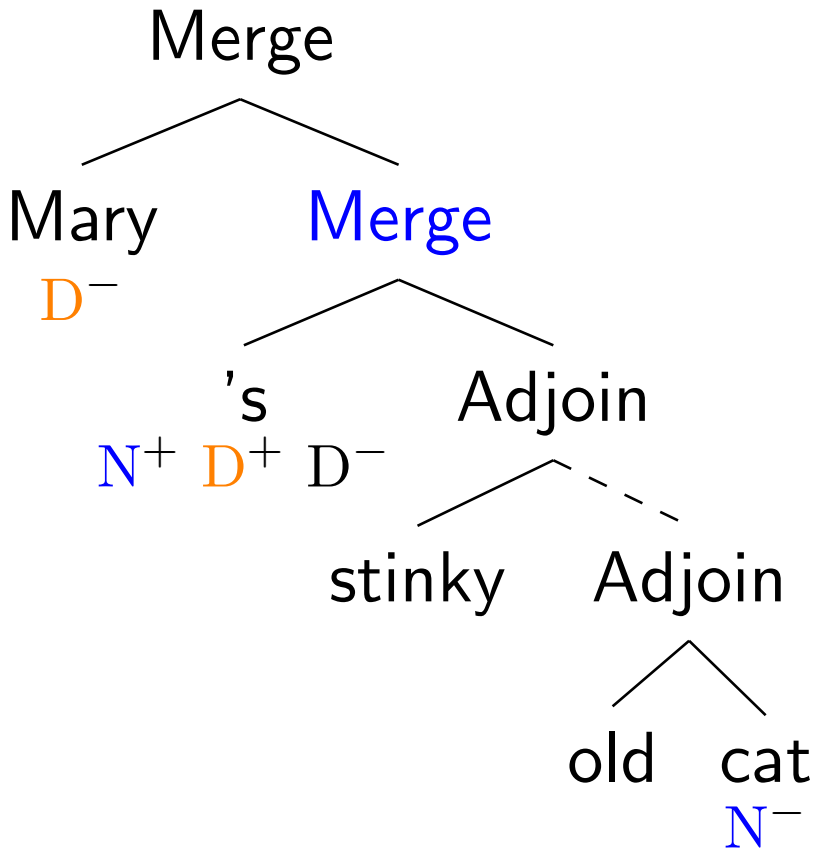


Non-Local Dependencies in Syntax

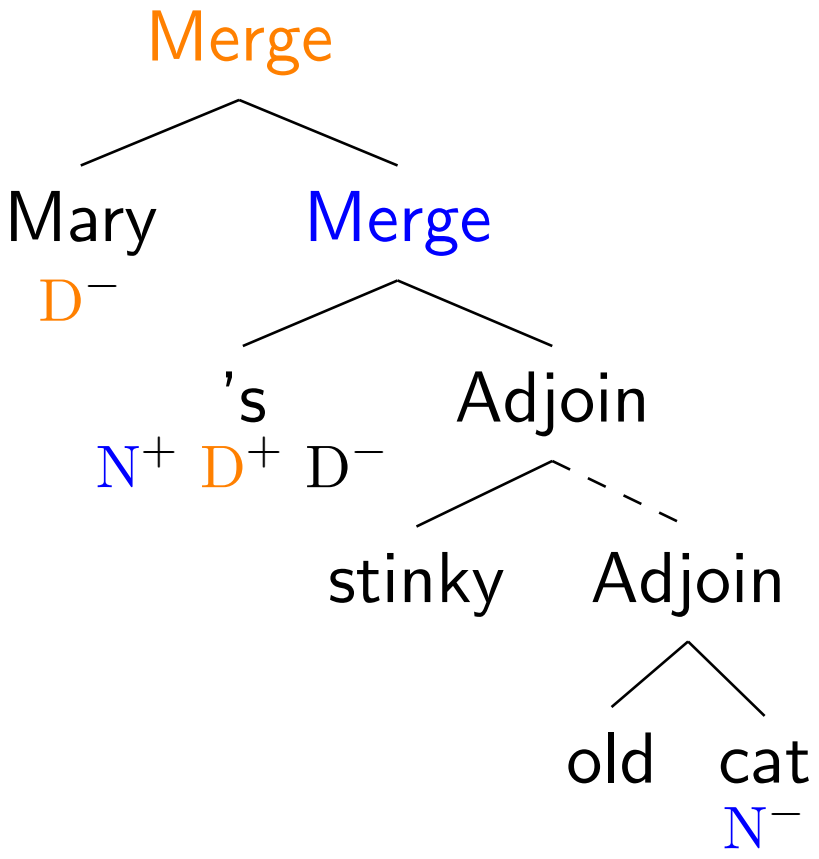
Let's stick to core operations:

- Move
- **Merge**: Unbounded adjunction

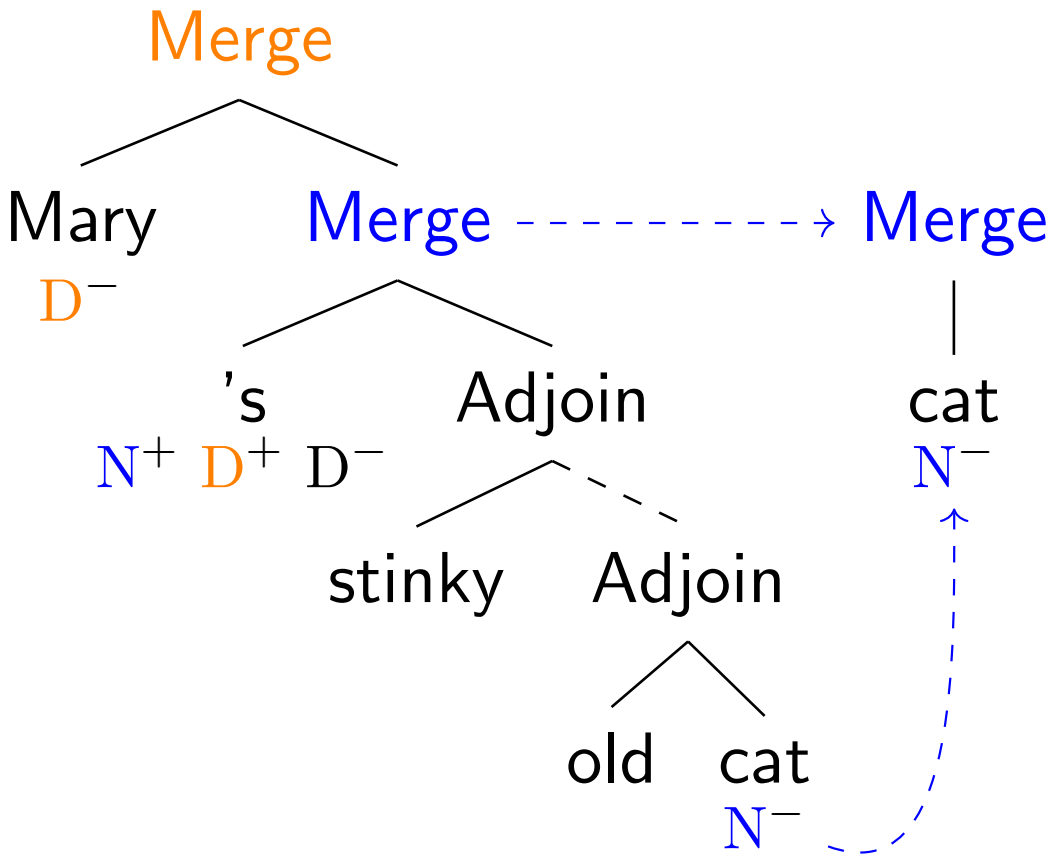
??



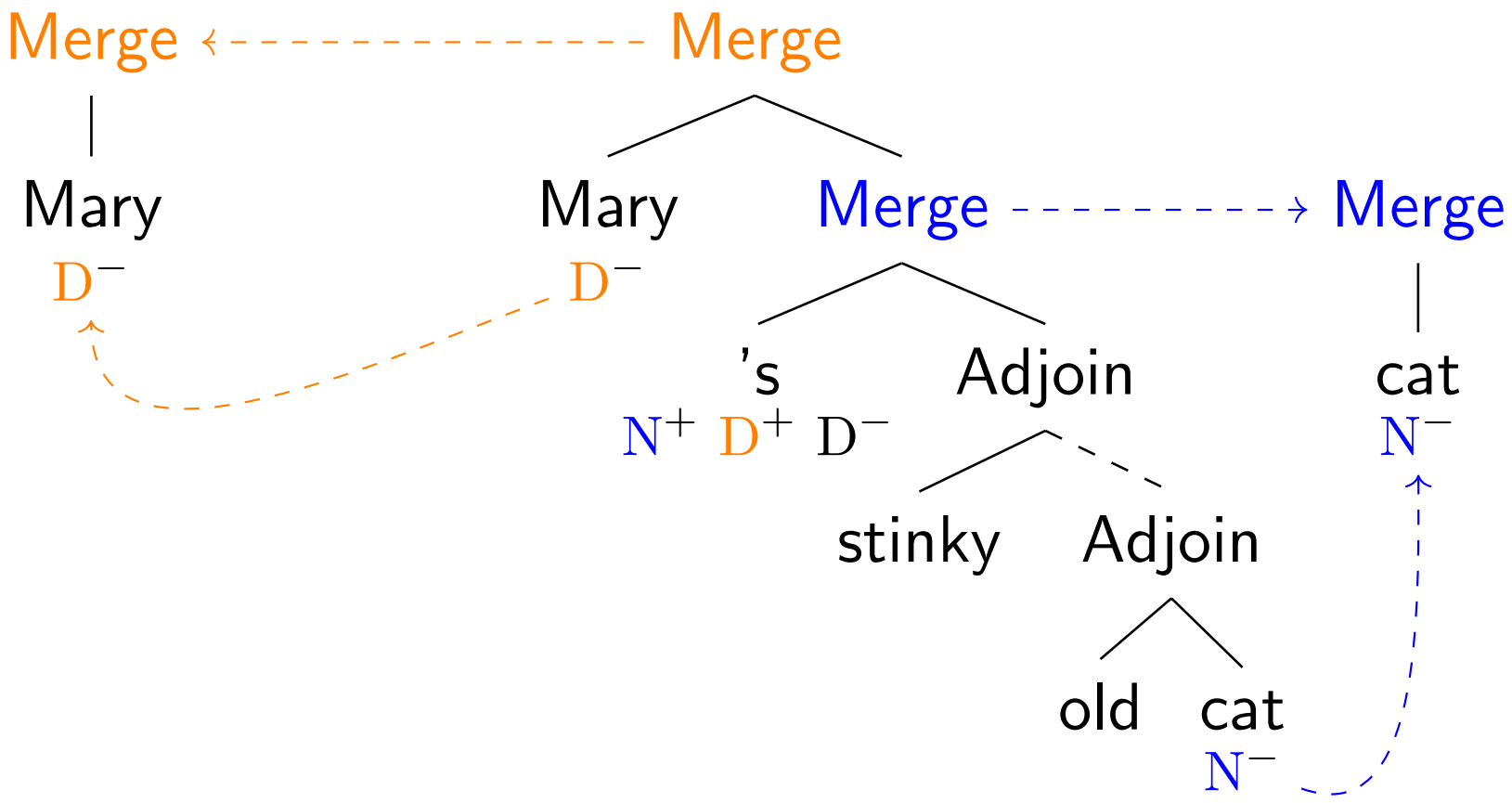
TSL over Trees: Projecting Tiers



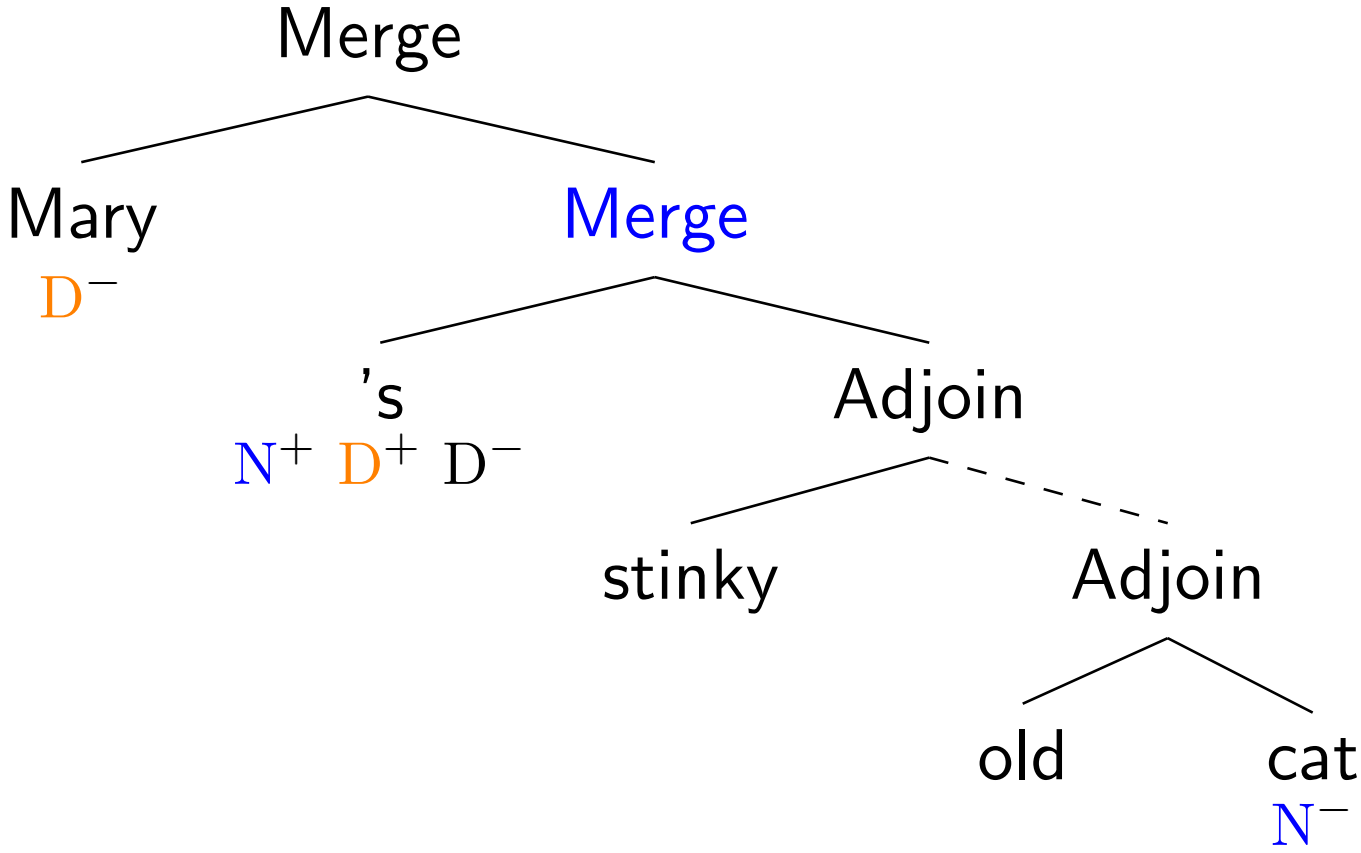
TSL over Trees: Projecting Tiers



TSL over Trees: Projecting Tiers

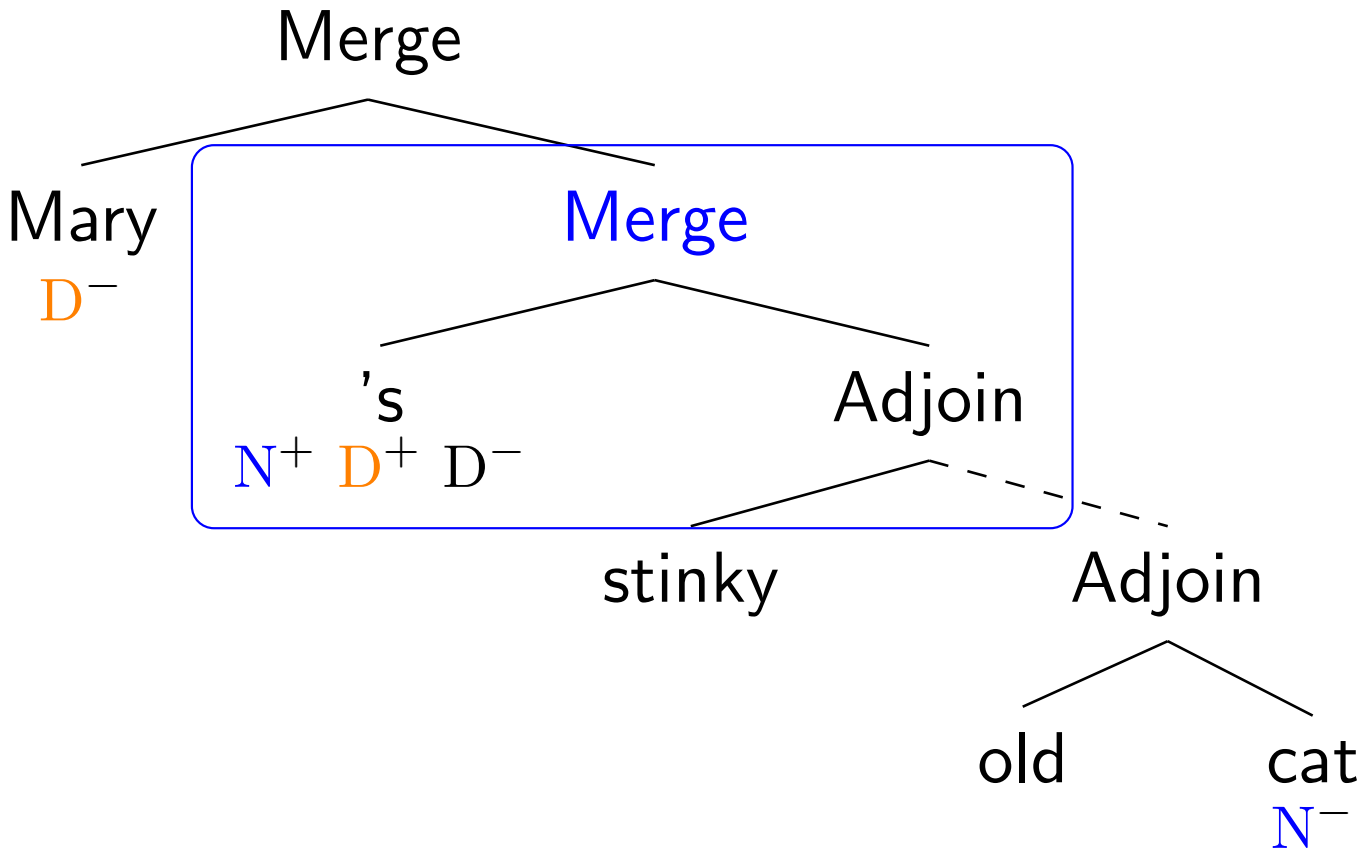


Merge with Adjunction is TSL



A TSL grammar for Merge

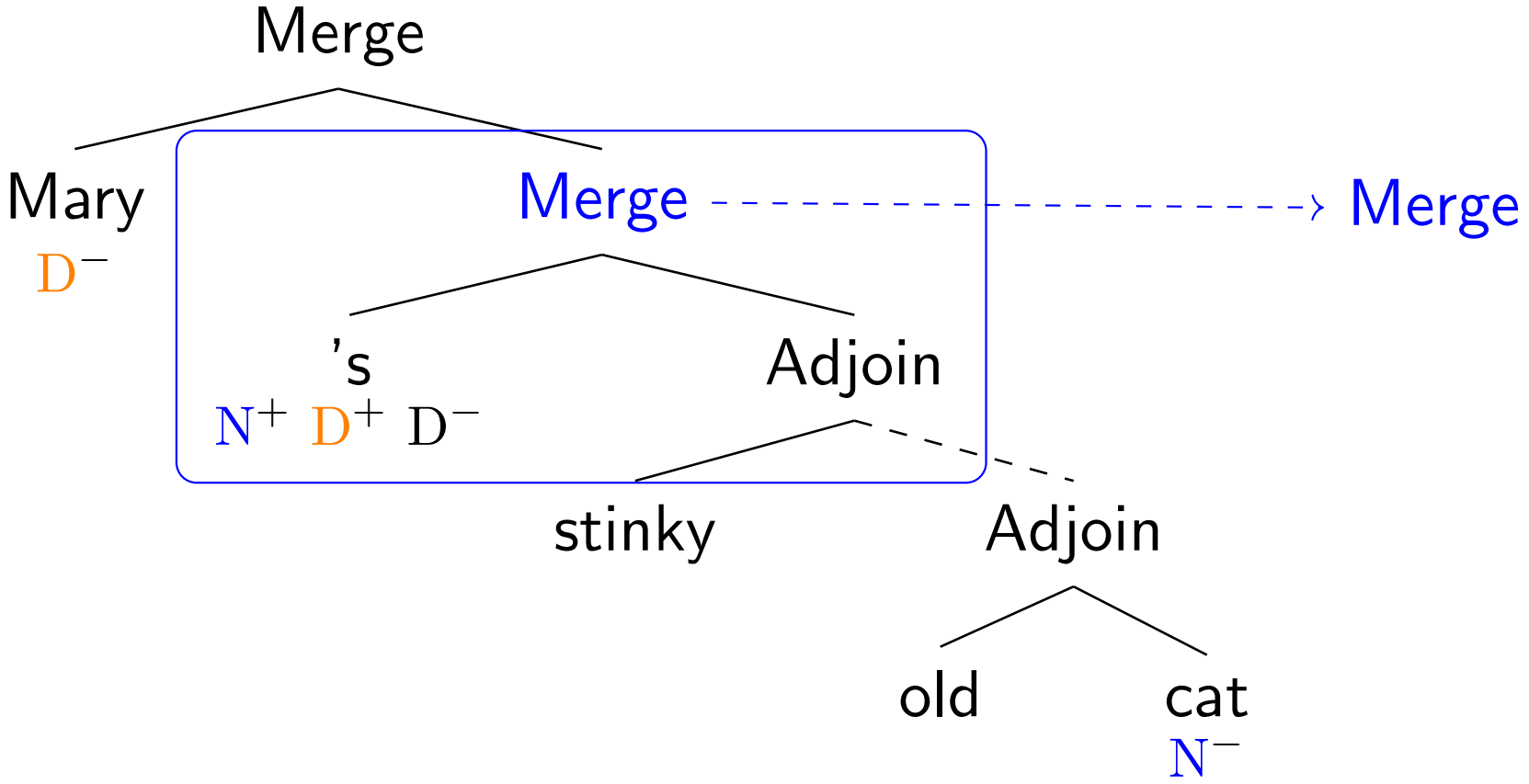
Merge with Adjunction is TSL



A TSL grammar for Merge

1. Project **Merge** iff a child has X^{+} (e.g. $\mathrm{X} = \mathrm{N}$)

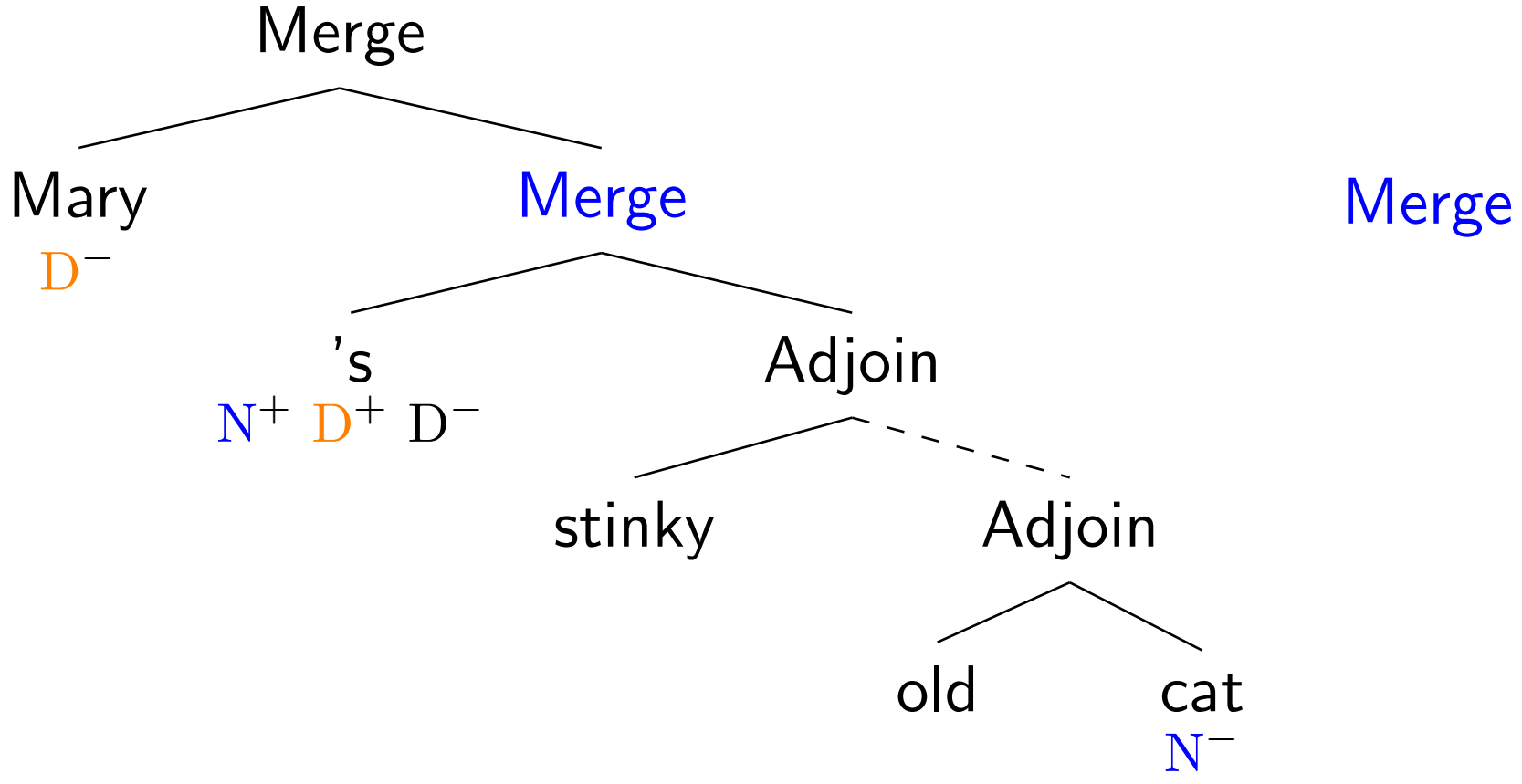
Merge with Adjunction is TSL



A TSL grammar for Merge

1. Project **Merge** iff a child has X^{+} (e.g. $\mathrm{X} = \mathrm{N}$)

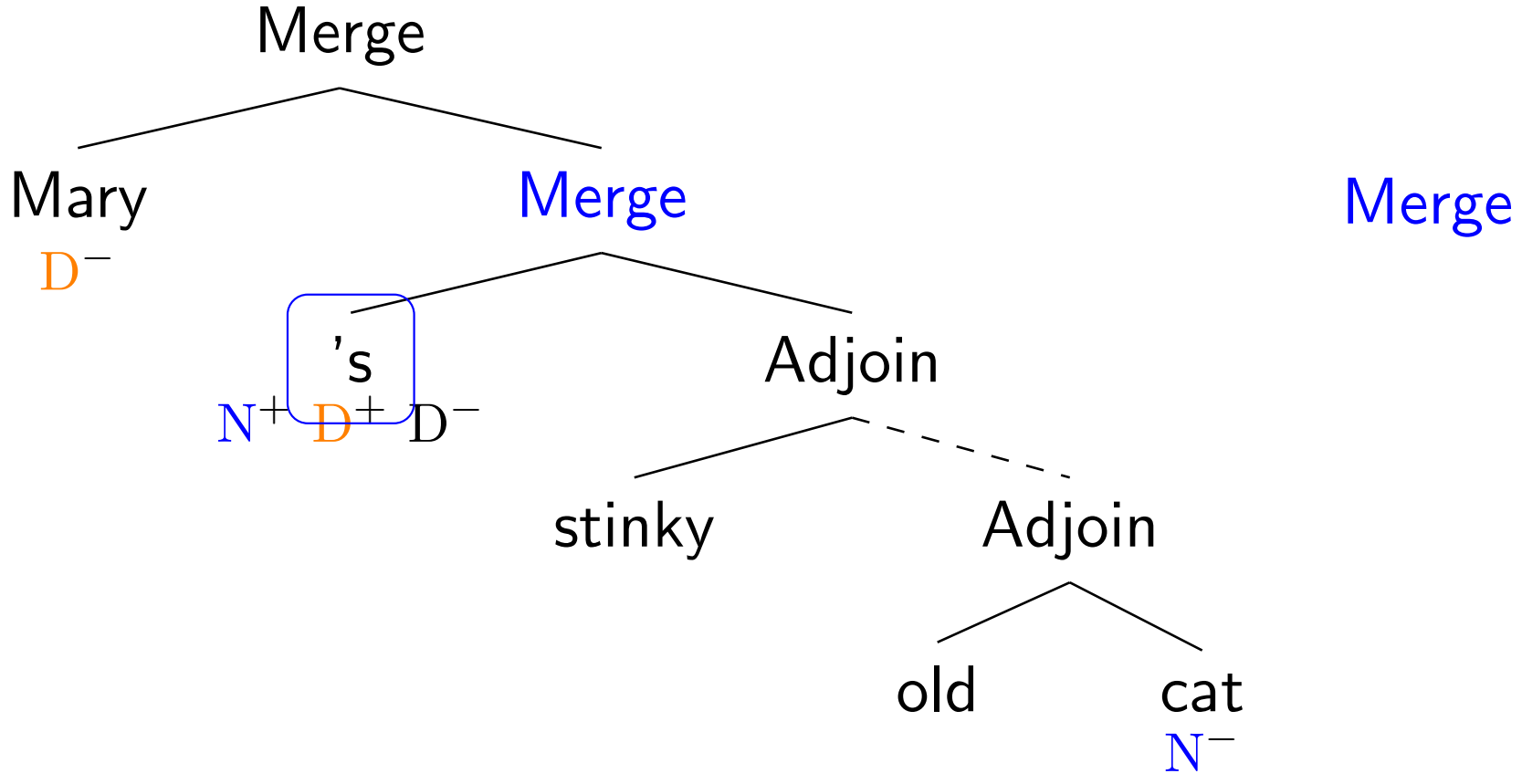
Merge with Adjunction is TSL



A TSL grammar for Merge

1. Project **Merge** iff a child has X^+ (e.g. $X = N$)
2. Project any node which has X^- (e.g. $X = N$)

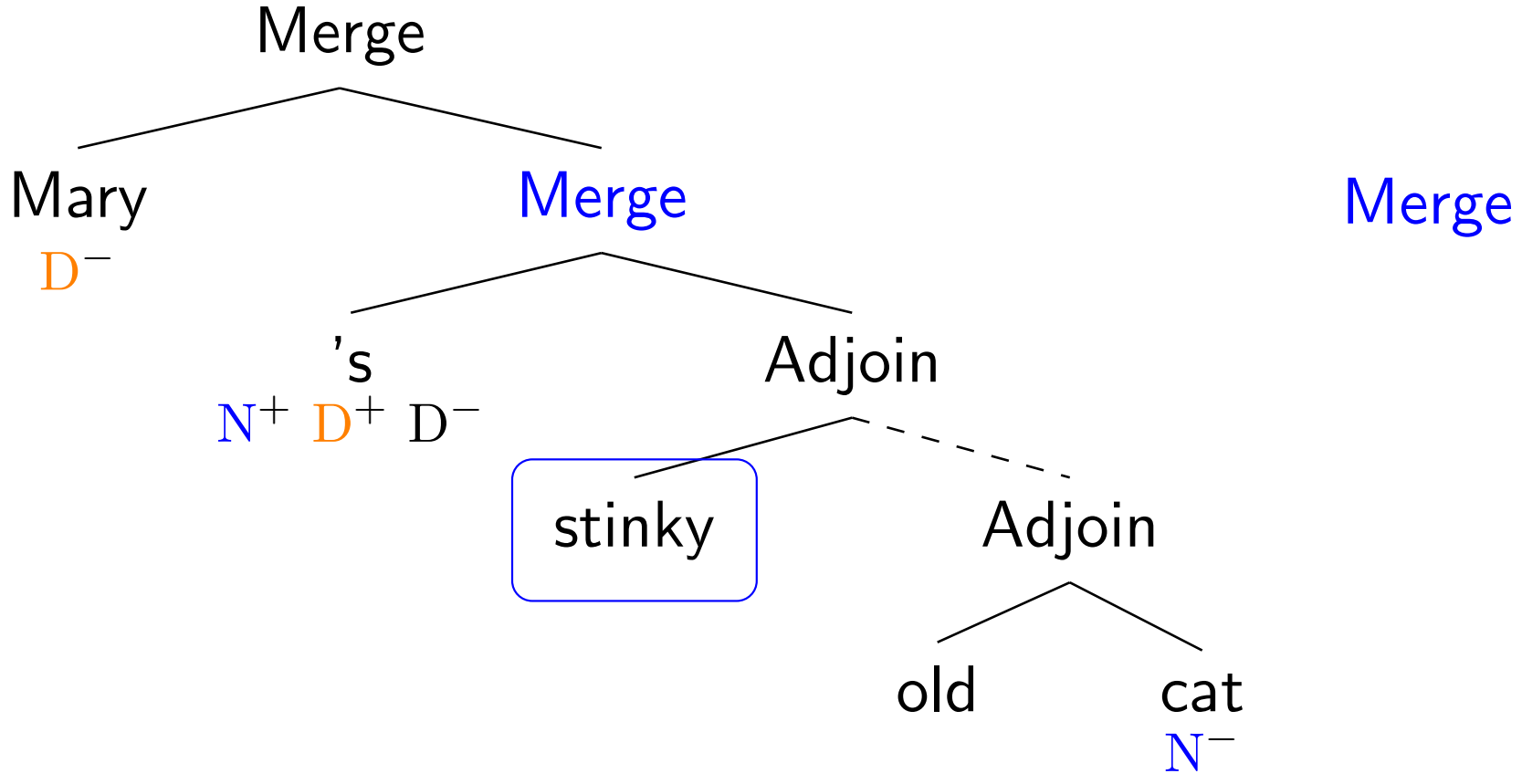
Merge with Adjunction is TSL



A TSL grammar for Merge

1. Project **Merge** iff a child has X^+ (e.g. $X = N$)
2. Project any node which has X^- (e.g. $X = N$)

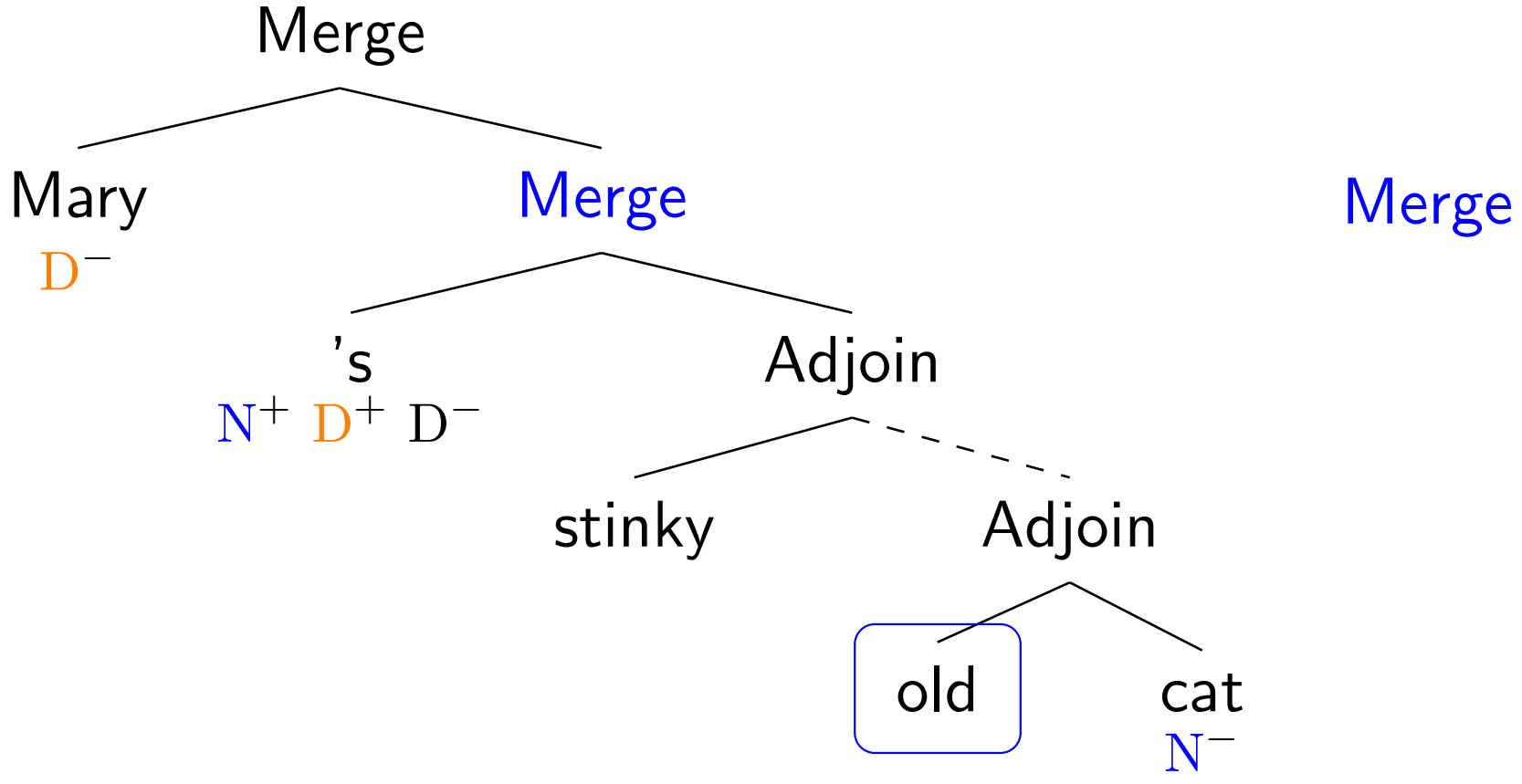
Merge with Adjunction is TSL



A TSL grammar for Merge

1. Project **Merge** iff a child has X^{+} (e.g. $\mathrm{X} = \mathrm{N}$)
2. Project any node which has X^{-} (e.g. $\mathrm{X} = \mathrm{N}$)

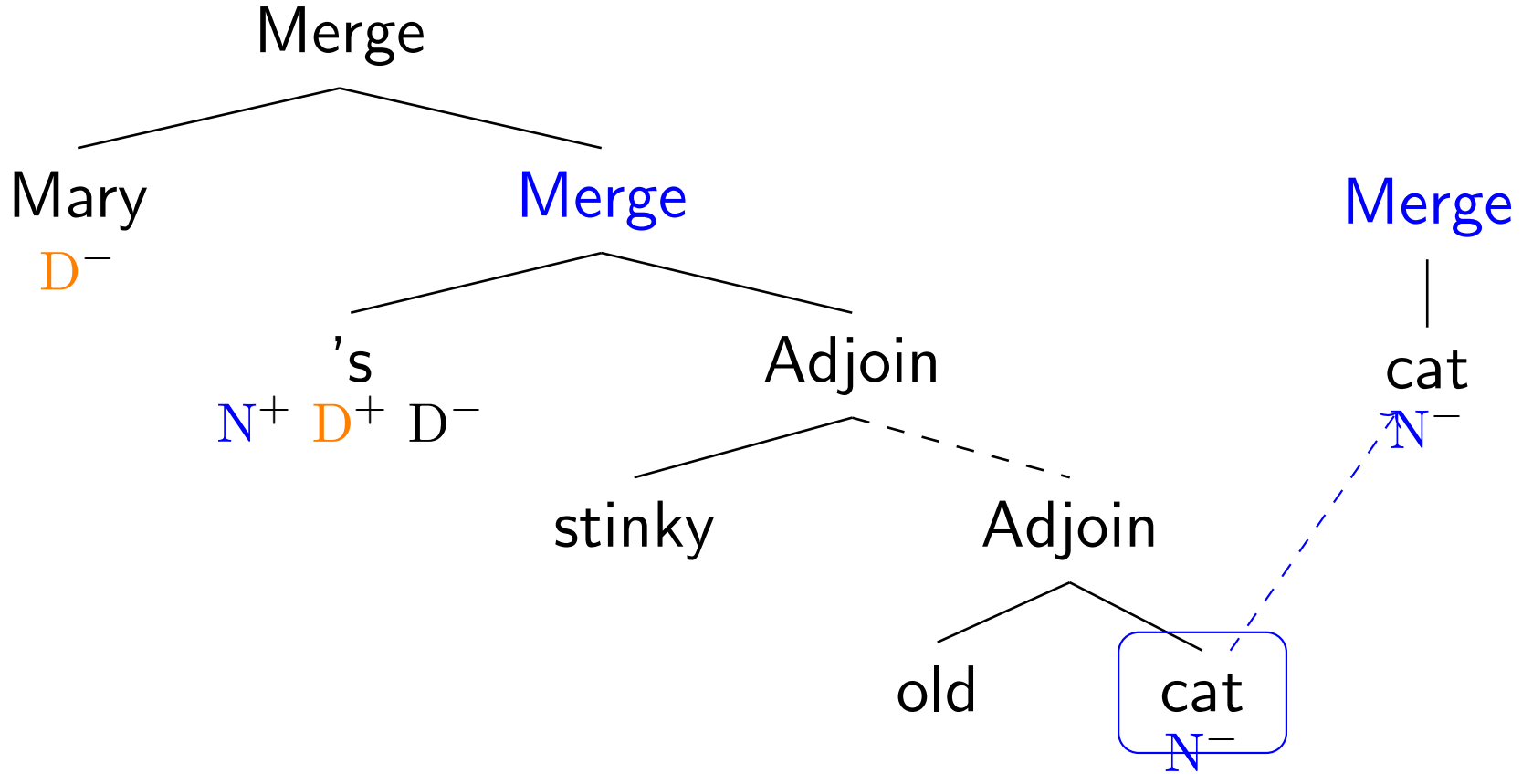
Merge with Adjunction is TSL



A TSL grammar for Merge

1. Project **Merge** iff a child has X^{+} (e.g. $\mathrm{X} = \mathrm{N}$)
2. Project any node which has X^{-} (e.g. $\mathrm{X} = \mathrm{N}$)

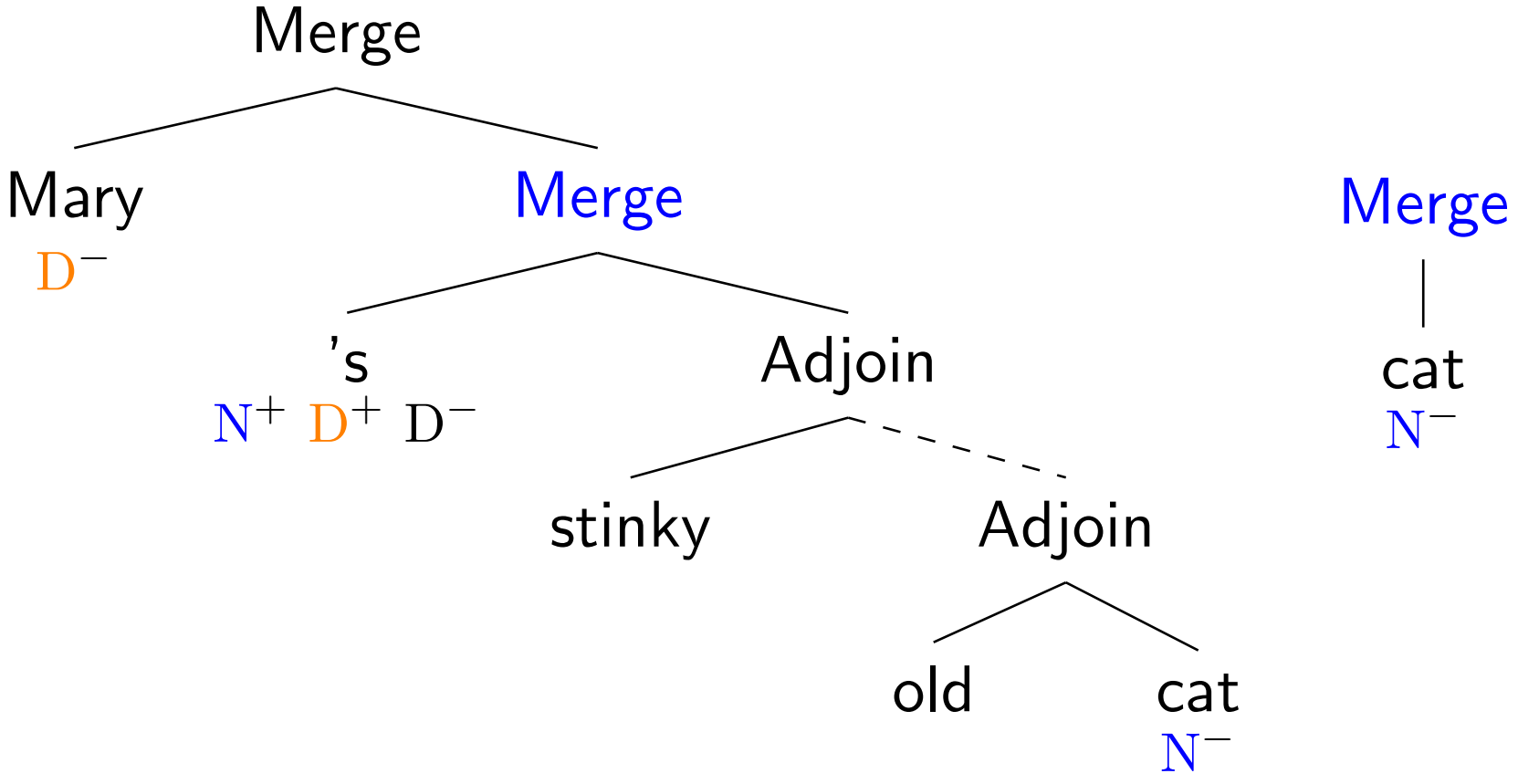
Merge with Adjunction is TSL



A TSL grammar for Merge

1. Project **Merge** iff a child has X^{+} (e.g. $\mathrm{X} = \mathrm{N}$)
2. Project any node which has X^{-} (e.g. $\mathrm{X} = \mathrm{N}$)

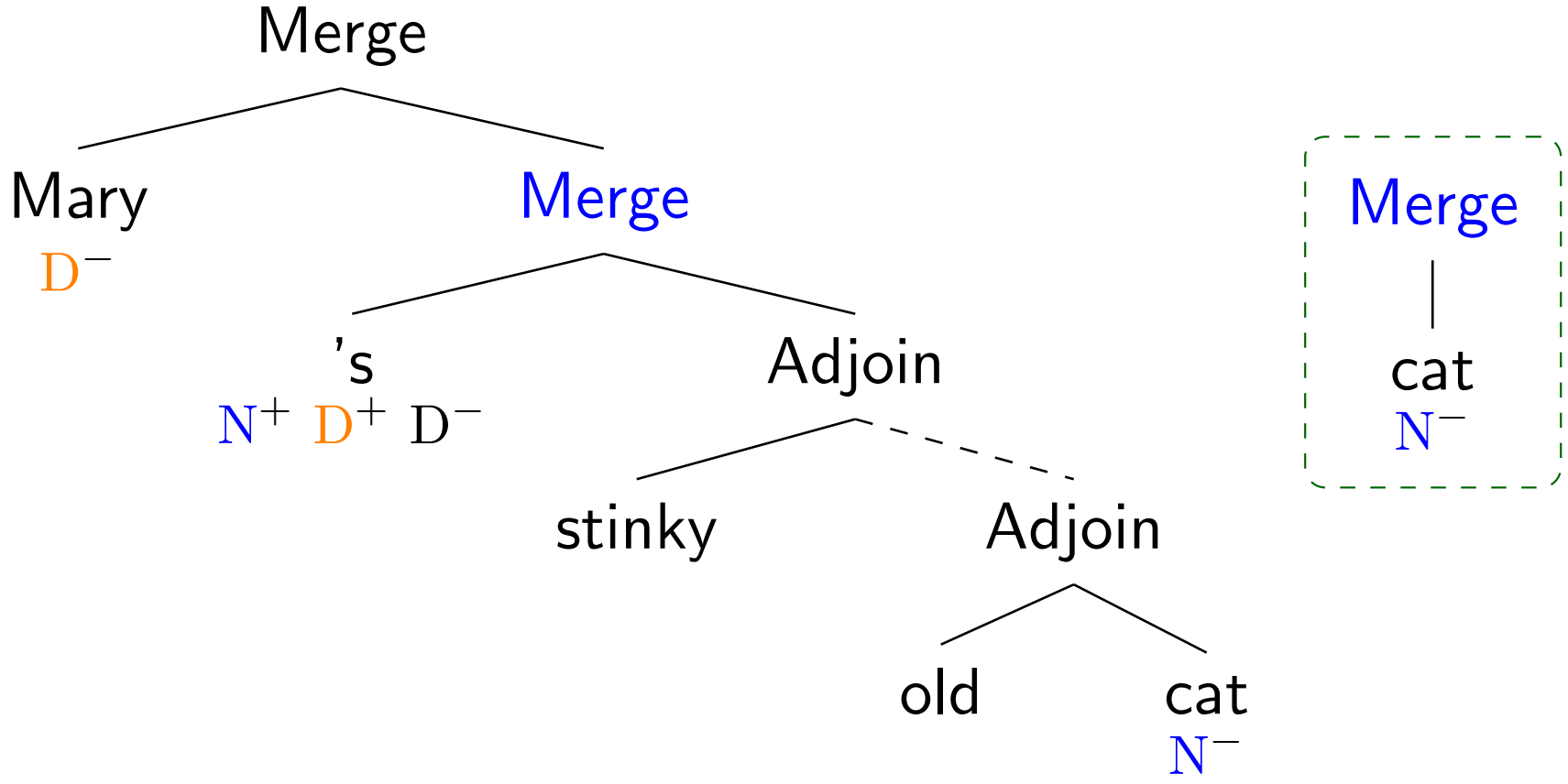
Merge with Adjunction is TSL



A TSL grammar for Merge

1. Project **Merge** iff a child has X^{+} (e.g. $\mathrm{X} = \mathrm{N}$)
2. Project any node which has X^{-} (e.g. $\mathrm{X} = \mathrm{N}$)

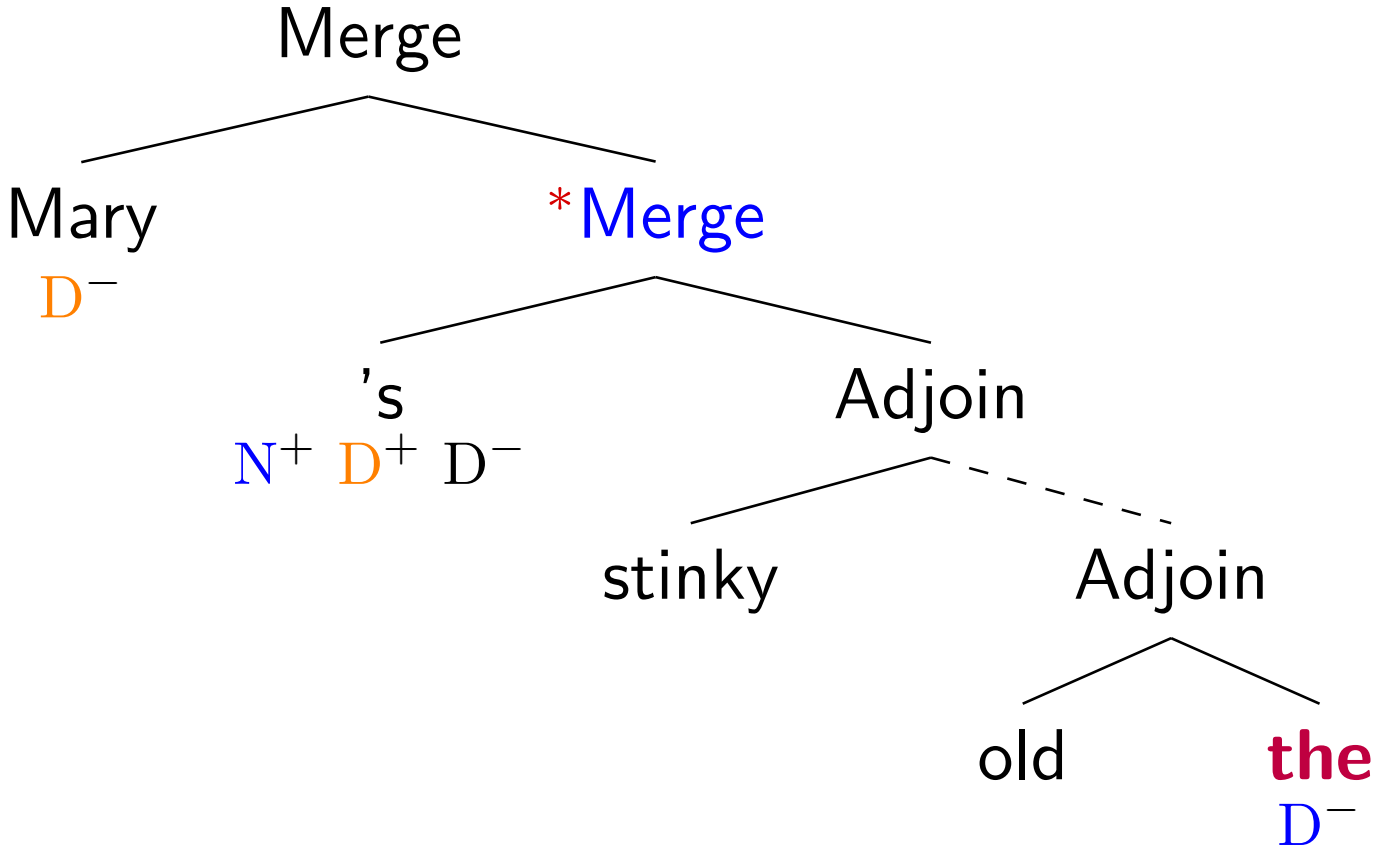
Merge with Adjunction is TSL



A TSL grammar for Merge

1. Project **Merge** iff a child has X^{+} (e.g. $\mathrm{X} = \mathrm{N}$)
2. Project any node which has X^{-} (e.g. $\mathrm{X} = \mathrm{N}$)
3. No Merge without exactly one LI among its daughters.

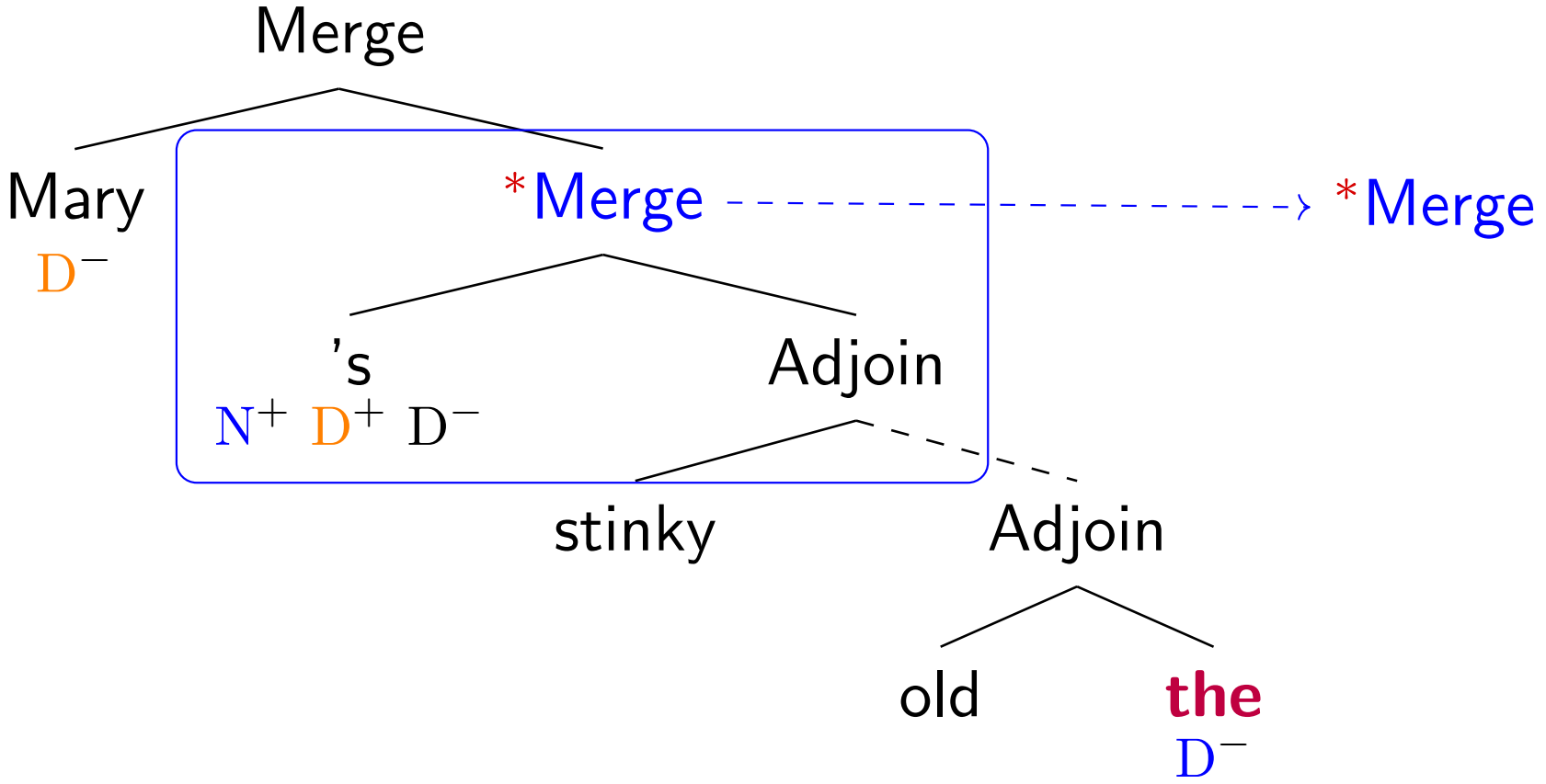
Merge with Adjunction is TSL



A TSL grammar for Merge

1. Project **Merge** iff a child has X^{+} (e.g. $\mathrm{X} = \mathrm{V}$)
2. Project any node which has X^{-} (e.g. $\mathrm{X} = \mathrm{V}$)
3. No Merge without exactly one LI among its daughters.

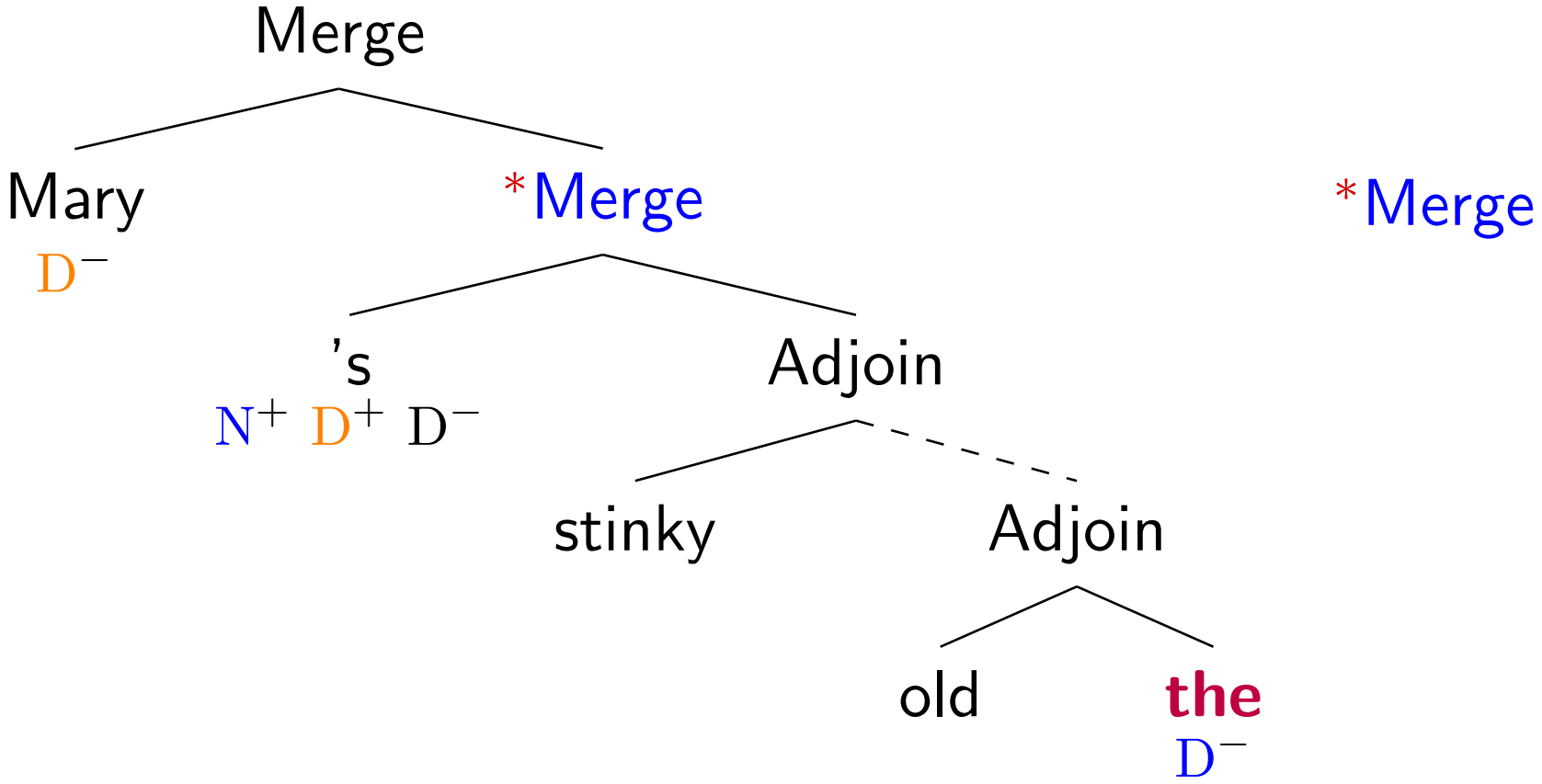
Merge with Adjunction is TSL



A TSL grammar for Merge

1. Project **Merge** iff a child has X^{+} (e.g. $\mathrm{X} = \mathrm{V}$)
2. Project any node which has X^{-} (e.g. $\mathrm{X} = \mathrm{V}$)
3. No Merge without exactly one LI among its daughters.

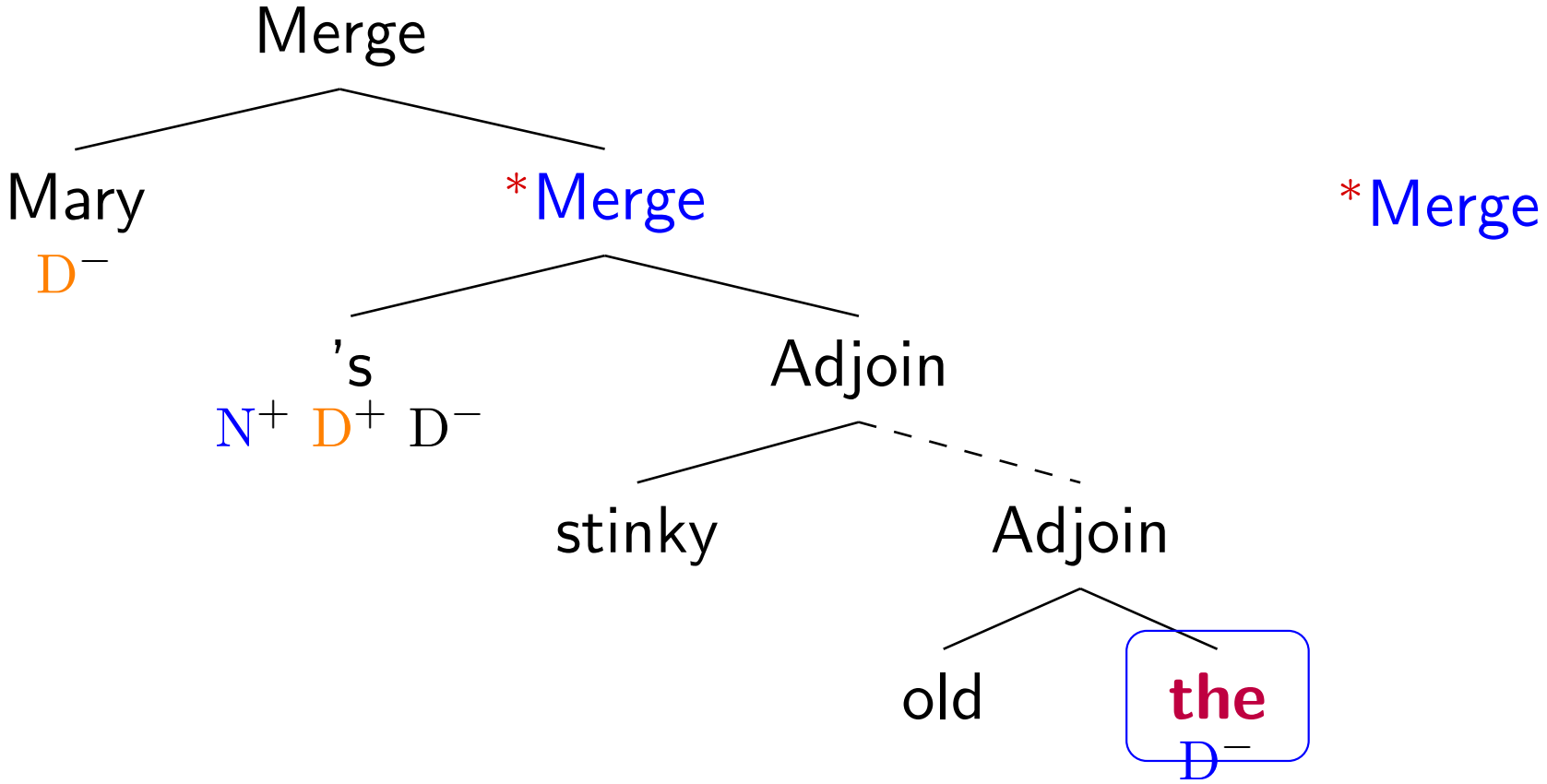
Merge with Adjunction is TSL



A TSL grammar for Merge

1. Project **Merge** iff a child has X^- (e.g. $\mathrm{X} = \mathrm{V}$)
2. Project any node which has X^+ (e.g. $\mathrm{X} = \mathrm{V}$)
3. No Merge without exactly one LI among its daughters.

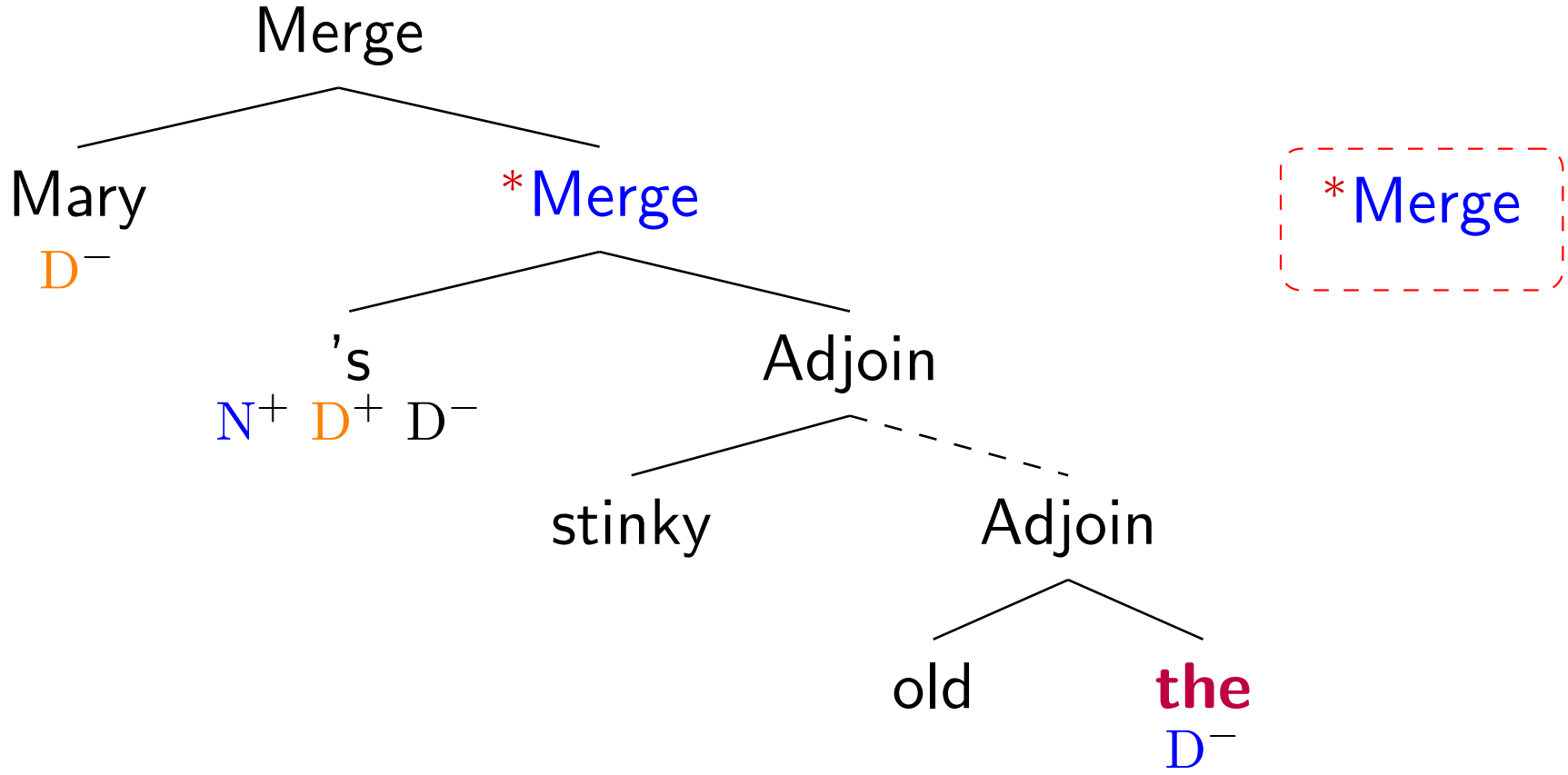
Merge with Adjunction is TSL



A TSL grammar for Merge

1. Project **Merge** iff a child has X^- (e.g. $\mathrm{X} = \mathrm{V}$)
2. Project any node which has X^+ (e.g. $\mathrm{X} = \mathrm{V}$)
3. No Merge without exactly one LI among its daughters.

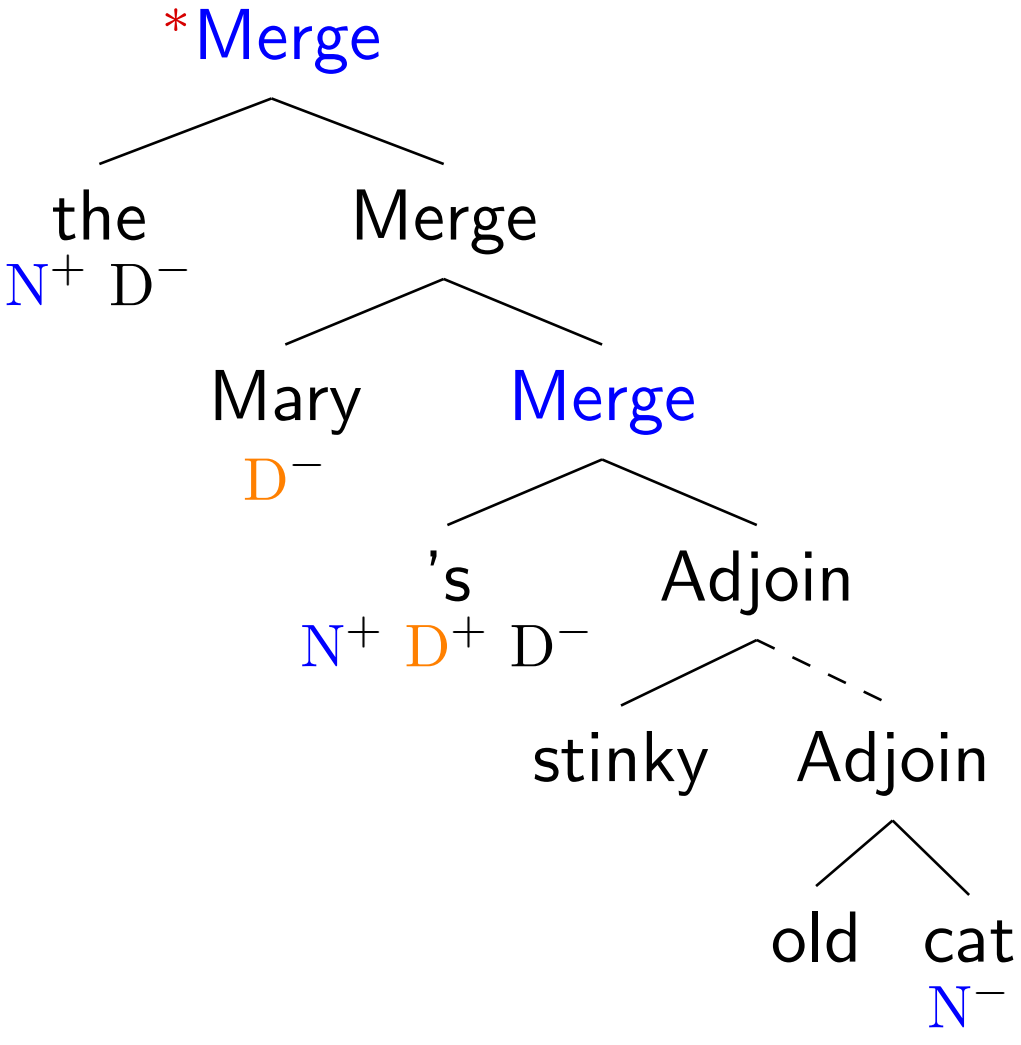
Merge with Adjunction is TSL



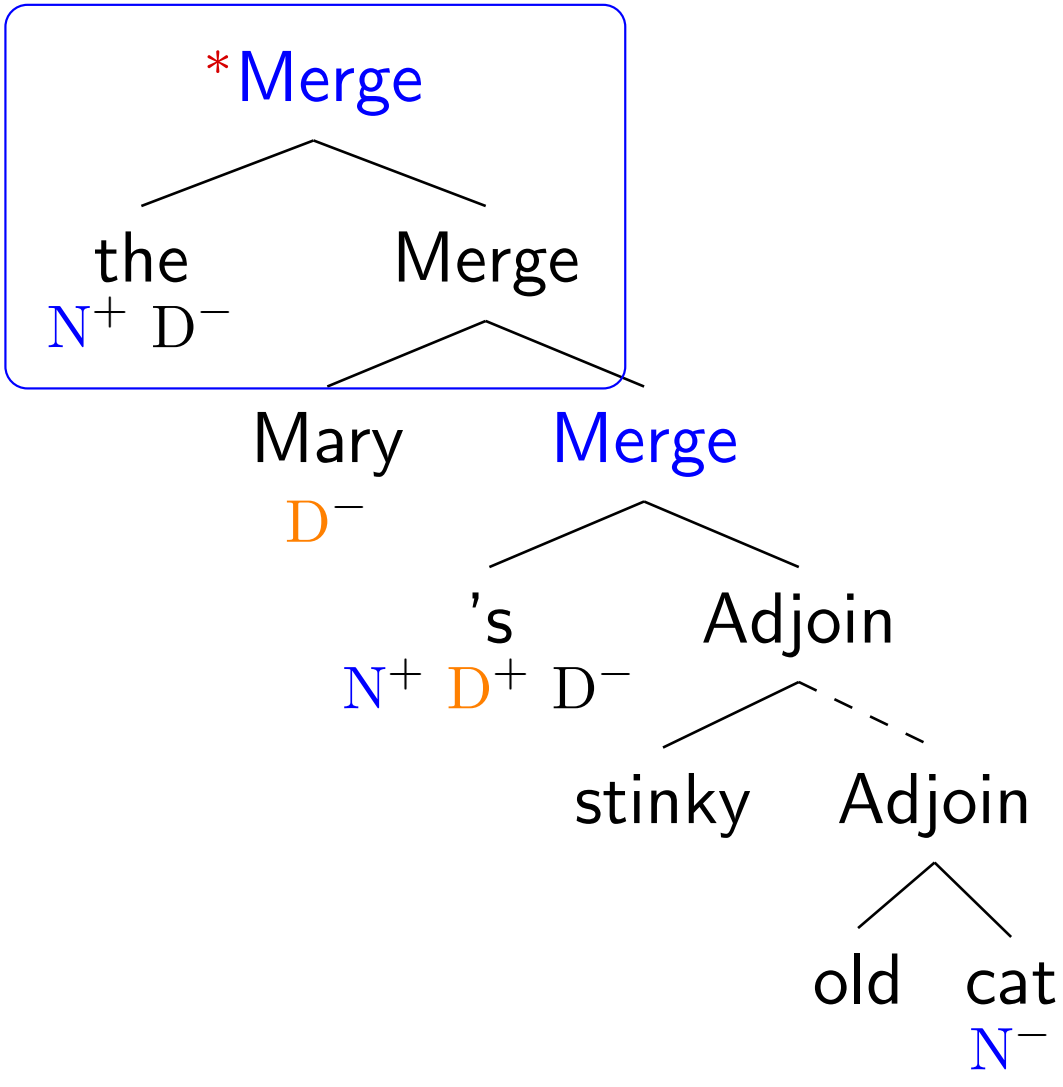
A TSL grammar for Merge

1. Project **Merge** iff a child has X^- (e.g. $\mathrm{X} = \mathrm{V}$)
2. Project any node which has X^+ (e.g. $\mathrm{X} = \mathrm{V}$)
3. No Merge without exactly one LI among its daughters.

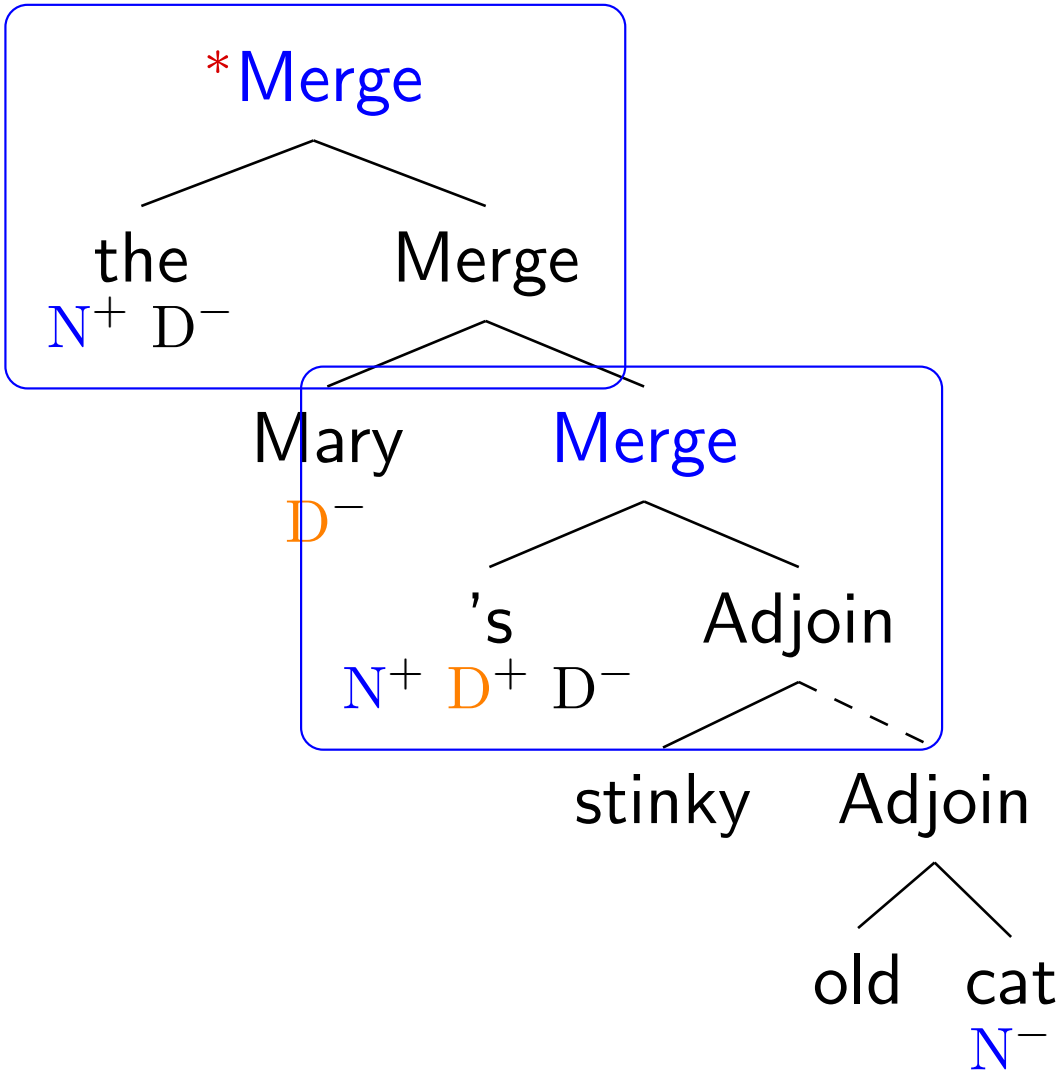
TSL Merge: Understanding the Constraint



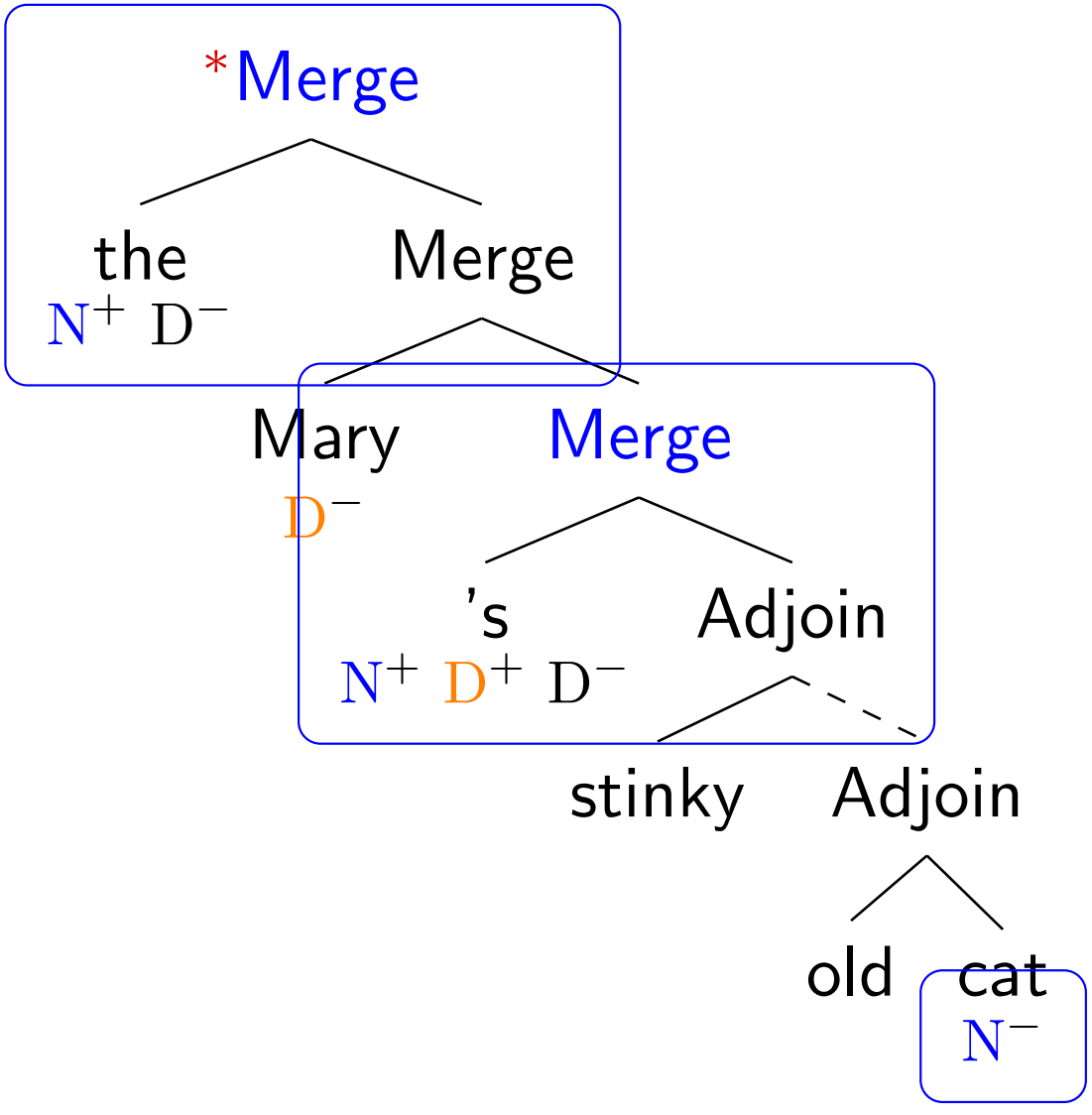
TSL Merge: Understanding the Constraint



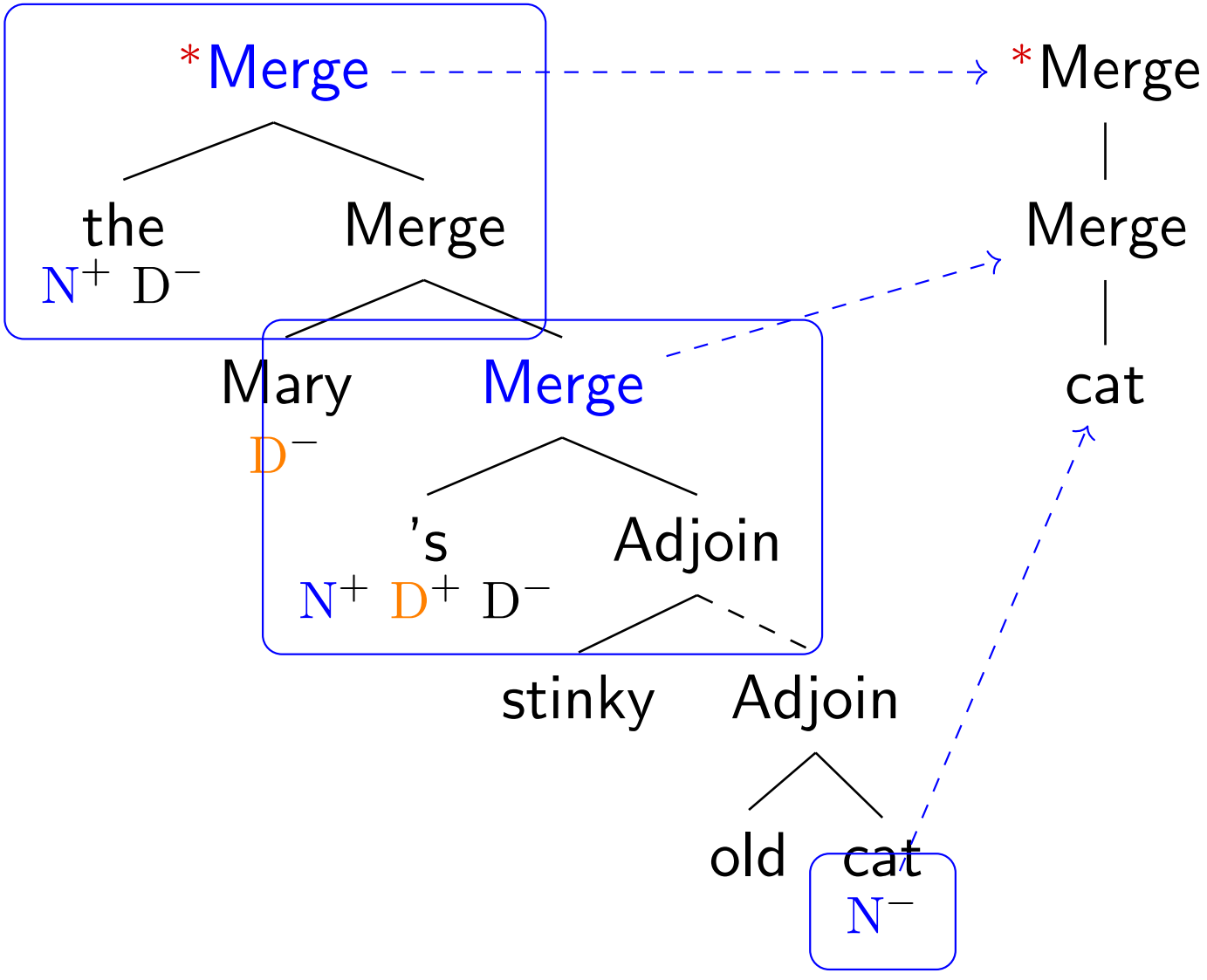
TSL Merge: Understanding the Constraint



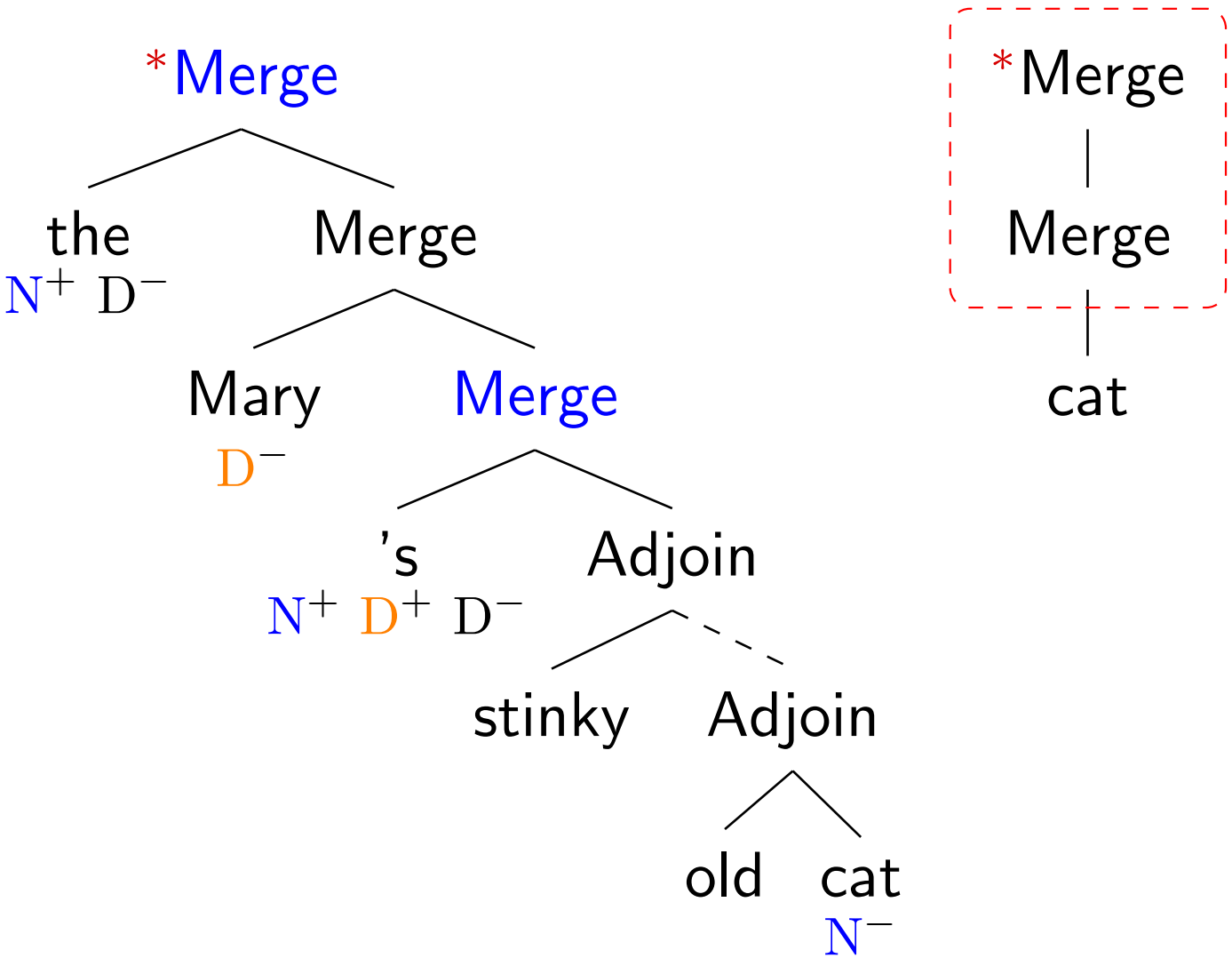
TSL Merge: Understanding the Constraint



TSL Merge: Understanding the Constraint



TSL Merge: Understanding the Constraint



Constraints on Move

What about Move?

Suppose our MG is in **single movement normal form**,
i.e. every phrase moves at most once.
Then movement is regulated by two constraints. (Graf 2012)

Constraints on Movement

Move	Every head with a negative Move feature is dominated by a matching Move node.
SMC	Every Move node is a closest dominating match for exactly one head.

Constraints on Move

What about Move?

Suppose our MG is in **single movement normal form**,
i.e. every phrase moves at most once.
Then movement is regulated by two constraints. (Graf 2012)

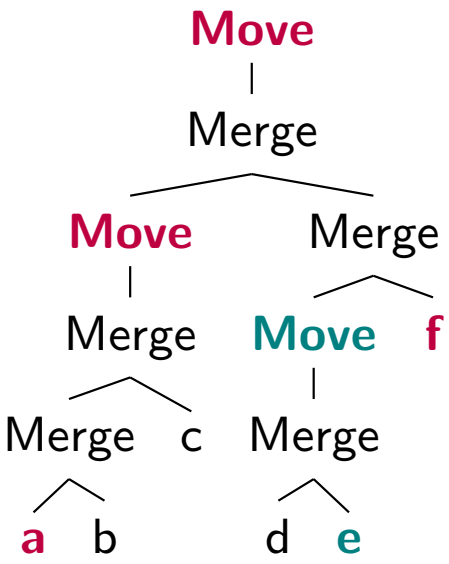
Constraints on Movement	
Move	Every head with a negative Move feature is dominated by a matching Move node.
SMC	Every Move node is a closest dominating match for exactly one head.

Tiers for Movement

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

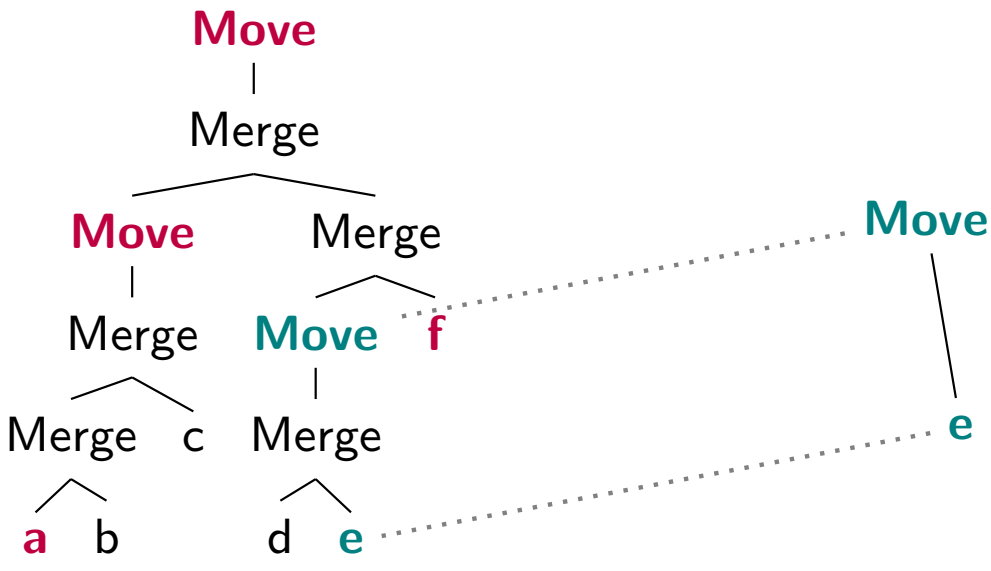
Tiers for Movement

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



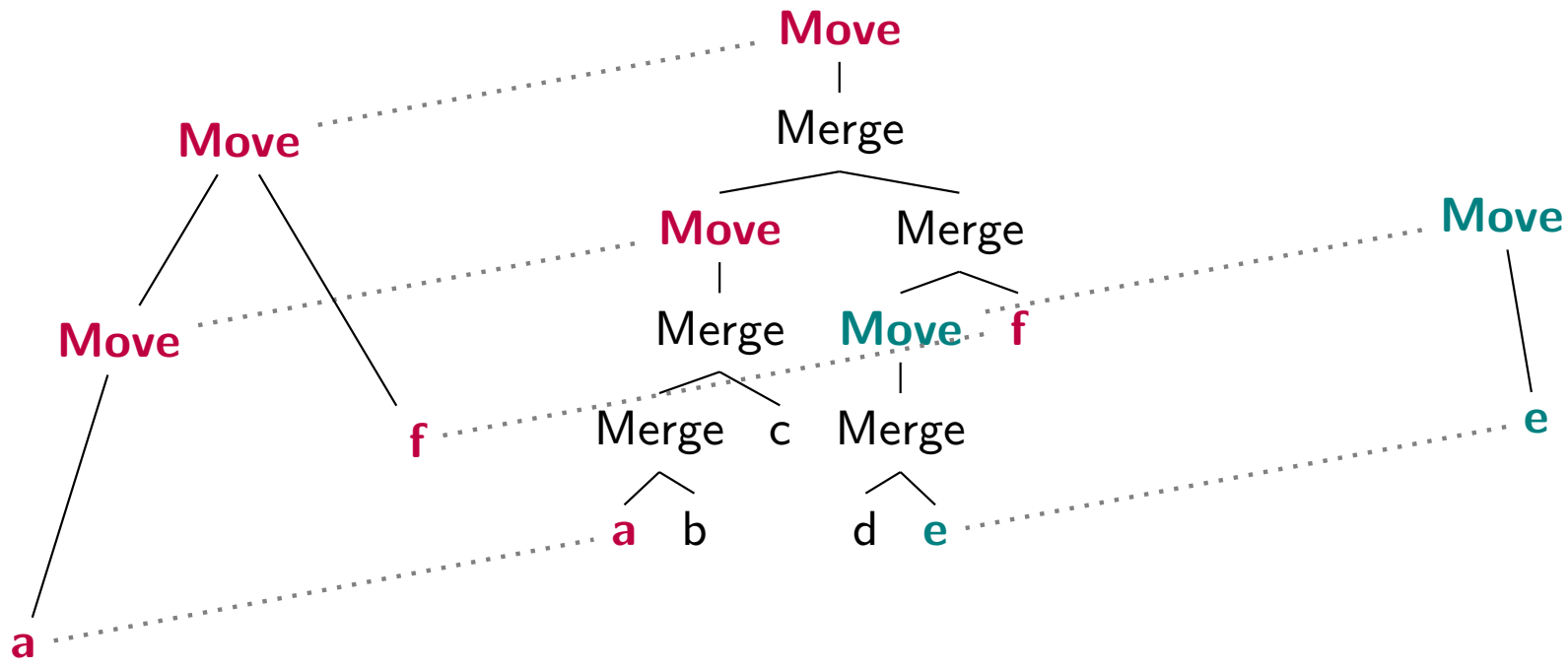
Tiers for Movement

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



Tiers for Movement

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



Move Constraints over Tiers

	Original	**Tier**
Move	Every head with a negative Move feature is dominated by a matching Move node.	Every lexical item has a **mother** labeled Move.
SMC	Every Move node is a closest dominating match for exactly one head.	Exactly one of a Move node's **daughters** is a lexical item.

Tree n-gram Templates

Move	**SMC1**	**SMC2**
$	Move	Move
≥ 1 LI	no LI	≥ 2 LIs