Introduction to Formal Semantics
Lecture 6: Beyond Function Application

Volha Petukhova & Nicolaie Dominik Dascalu

Spoken Language Systems Group
Saarland Univeristy

30.05.2022

UNIVERSITÄT
DES
SAARLANDES

Overview for today

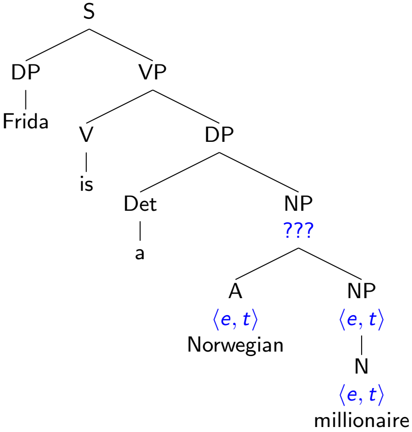
- Recap: Function Application
- Predicate Modification
- Type Shifting
- Predicate Abstraction
- Quantifier Raising

Reading:

- Coppock, E., and Champollion, L. (2021). Invitation to formal semantics. Manuscript, Boston University and New York University (Ch.7)

Quizz (last week)

Assume that *Norwegian* and *millionaire* are both of type $\langle e, t\rangle$ following the style we have developed so far. Is it possible to assign truth conditions to the following sentence using those assumptions? Why or why not?



Adjectives

Nouns denoting sets of individuals: set of millionaires and set of Norwegians, the set they share in common Norwegian millionaires - their intersection. (INTERSECTIVE adjectives. Examples: broken cup, curly haired girl

Adjectives

Nouns denoting sets of individuals: set of millionaires and set of Norwegians, the set they share in common Norwegian millionaires - their intersection. (INTERSECTIVE adjectives. Examples: broken cup, curly haired girl

Many adjectives are SUBSECTIVE adjectives. For example, set of beautiful dancer is a subset of of all dancers.

Adjectives

Nouns denoting sets of individuals: set of millionaires and set of Norwegians, the set they share in common Norwegian millionaires - their intersection. (INTERSECTIVE adjectives. Examples: broken cup, curly haired girl

Many adjectives are SUBSECTIVE adjectives. For example, set of beautiful dancer is a subset of of all dancers.

Some adjectives are PRIVATIVE, they map sets to disjoint sets. For example, fake gun will depend what set gun denotes: only real guns or real and fake guns

Adjectives: intersective

norwegian $\not\rightsquigarrow \lambda x.Norwegian(x)$

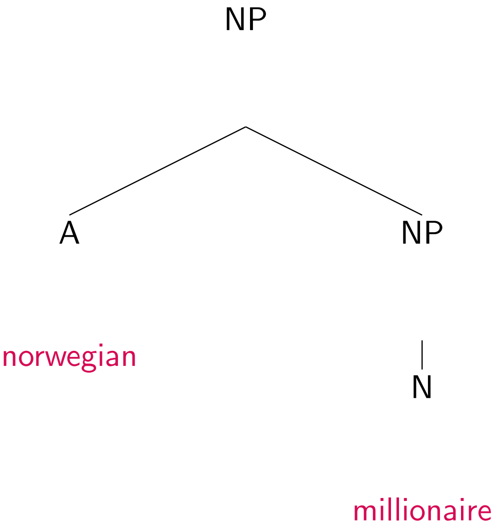
Adjectives: intersective

norwegian $\not\rightsquigarrow \lambda x.\mathit{Norwegian}(x)$

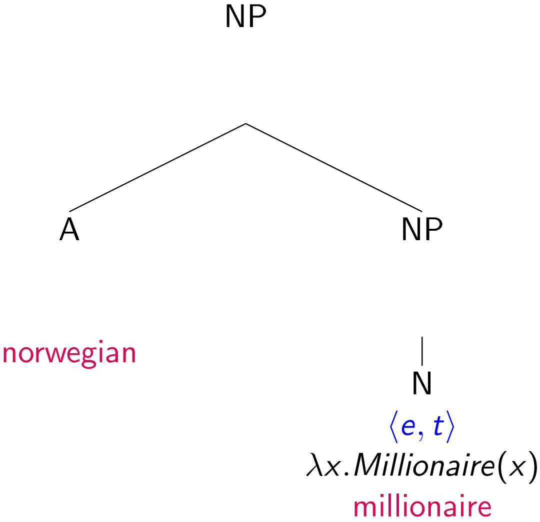
norwegian $\rightsquigarrow \lambda P \lambda x.[\mathit{Norwegian}(x) \wedge P(x)]$

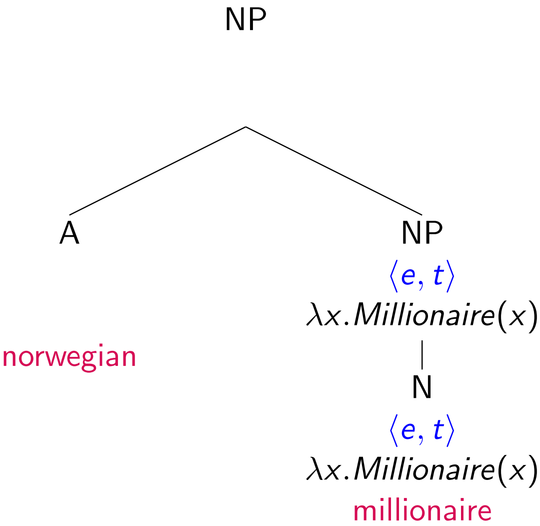
thus of $\langle\langle e, t\rangle, \langle e, t\rangle\rangle$ type: returns a new predicate that are both *norwegians* and in the set of denoted by the input predicate *millionaires*

Adjectives: intersective (cont.)



Adjectives: intersective (cont.)




NP
A
norwegian
NP
$\langle e, t \rangle$
$\lambda x.Millionaire(x)$
N
$\langle e, t \rangle$
$\lambda x.Millionaire(x)$
millionaire

Adjectives: intersective (cont.)

NP

A
$\langle\langle e,t\rangle,\langle e,t\rangle\rangle$
$\lambda P \lambda x.[Norwegian(x) \wedge P(x)]$
norwegian

NP
$\langle e,t\rangle$
$\lambda x.Millionaire(x)$

N
$\langle e,t\rangle$
$\lambda x.Millionaire(x)$
millionaire

Adjectives: intersective (cont.)

- NP $\langle e, t \rangle$ $\lambda x.[Norwegian(x) \wedge Millionaire(x)]$
  - A $\langle\langle e, t \rangle, \langle e, t \rangle\rangle$ $\lambda P \lambda x.[Norwegian(x) \wedge P(x)]$
    - norwegian
  - NP $\langle e, t \rangle$ $\lambda x.Millionaire(x)$
    - N $\langle e, t \rangle$ $\lambda x.Millionaire(x)$
      - millionaire

Adjectives: subsective

beautiful $\rightsquigarrow \forall P \forall x. BeautifulAs(P)(x) \rightarrow P(x)$

thus as function of $\langle\langle e,t\rangle,\langle e,t\rangle\rangle$ type: for every set P (dancers), every beautiful Dancer is a Dancer (MEANING POSTULATE)

Adjectives: subsective

beautiful $\rightsquigarrow \forall P \forall x. BeautifulAs(P)(x) \rightarrow P(x)$

thus as function of $\langle\langle e, t\rangle, \langle e, t\rangle\rangle$ type: for every set P (dancers), every beautiful Dancer is a Dancer (MEANING POSTULATE)

To block interpretation that every beautiful dancer is a beautiful individual

Adjectives: subsective

beautiful $\rightsquigarrow \forall P \forall x. BeautifulAs(P)(x) \rightarrow P(x)$

thus as function of $\langle\langle e,t\rangle,\langle e,t\rangle\rangle$ type: for every set P (dancers), every beautiful Dancer is a Dancer (MEANING POSTULATE)

To block interpretation that every beautiful dancer is a beautiful individual

Example

Adjectives: subsective

beautiful $\rightsquigarrow \forall P \forall x. BeautifulAs(P)(x) \rightarrow P(x)$

thus as function of $\langle\langle e, t\rangle, \langle e, t\rangle\rangle$ type: for every set P (dancers), every beautiful Dancer is a Dancer (MEANING POSTULATE)

To block interpretation that every beautiful dancer is a beautiful individual

Example

Nuriev is a beautiful dancer.

Adjectives: subsective

beautiful $\rightsquigarrow \forall P \forall x.BeautifulAs(P)(x) \rightarrow P(x)$

thus as function of $\langle\langle e,t\rangle,\langle e,t\rangle\rangle$ type: for every set P (dancers), every beautiful Dancer is a Dancer (MEANING POSTULATE)

To block interpretation that every beautiful dancer is a beautiful individual

Example

Nuriev is a beautiful dancer.
$\therefore$ Nuriev is a dancer.

Adjectives: subsective

beautiful $\rightsquigarrow \forall P \forall x.BeautifulAs(P)(x) \rightarrow P(x)$

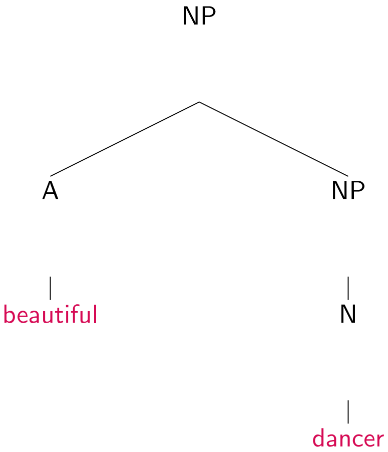
thus as function of $\langle\langle e,t\rangle,\langle e,t\rangle\rangle$ type: for every set P (dancers), every beautiful Dancer is a Dancer (MEANING POSTULATE)

To block interpretation that every beautiful dancer is a beautiful individual

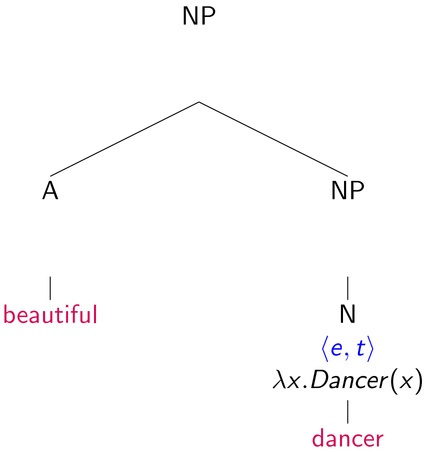
Example

Nuriev is a beautiful dancer.
$\therefore$ Nuriev is a dancer.
$\not\therefore$ Nuriev is beautiful.

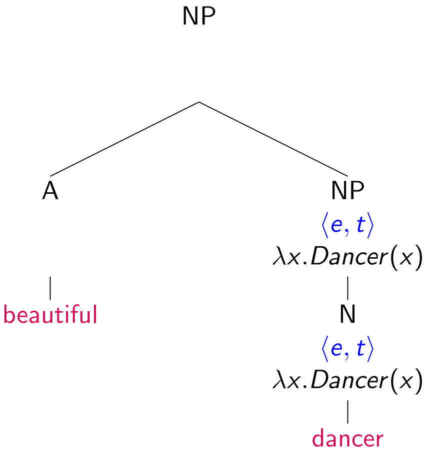
Adjectives: subsective (cont.)



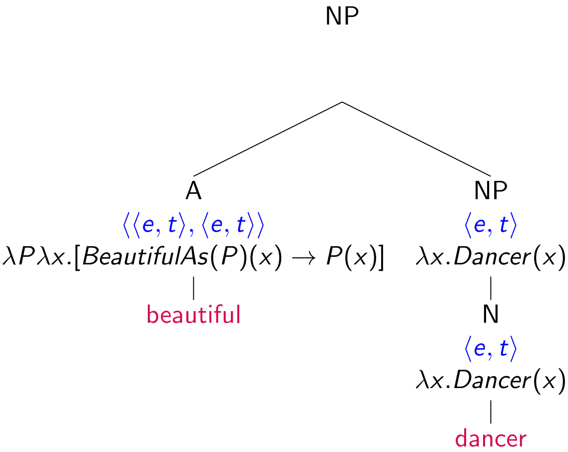
Adjectives: subsective (cont.)



Adjectives: subsective (cont.)



Adjectives: subsective (cont.)



Adjectives: subsective (cont.)

NP

$\langle e, t \rangle$

$\lambda x.\forall x.[BeautifulAs(Dancer)(x) \rightarrow Dancer(x)]$

A

$\langle \langle e, t \rangle, \langle e, t \rangle \rangle$

$\lambda P \lambda x.[BeautifulAs(P)(x) \rightarrow P(x)]$

beautiful

NP

$\langle e, t \rangle$

$\lambda x.Dancer(x)$

N

$\langle e, t \rangle$

$\lambda x.Dancer(x)$

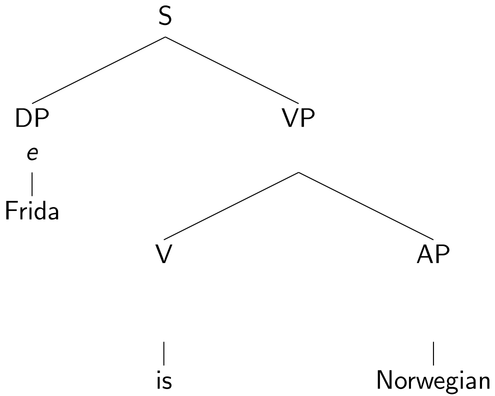
dancer

Adjectives: predicative position

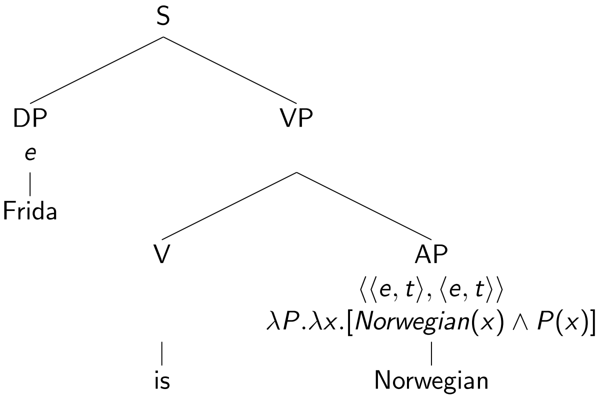
Example

Frida is Norwegian.
This is reasonable.
Hair is curly.

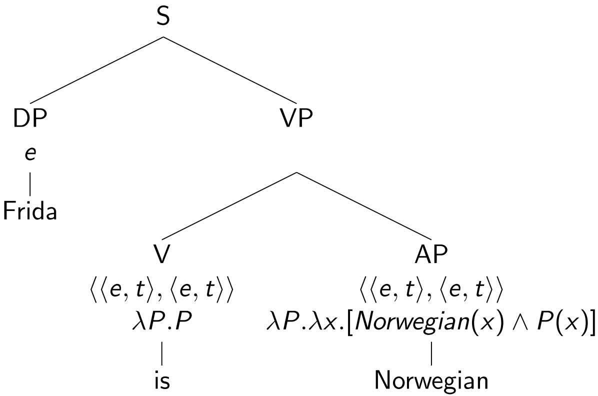
Adjectives: predicative position (cont.)

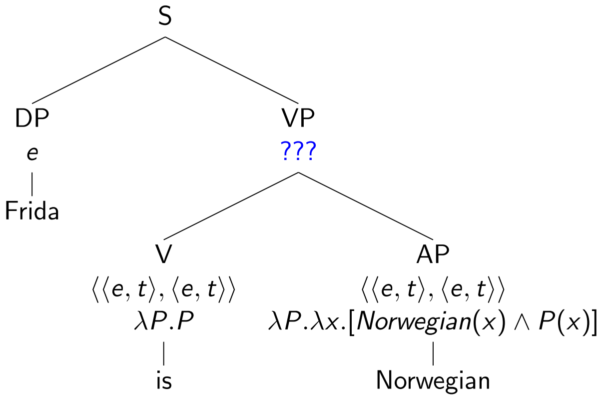


Adjectives: predicative position (cont.)

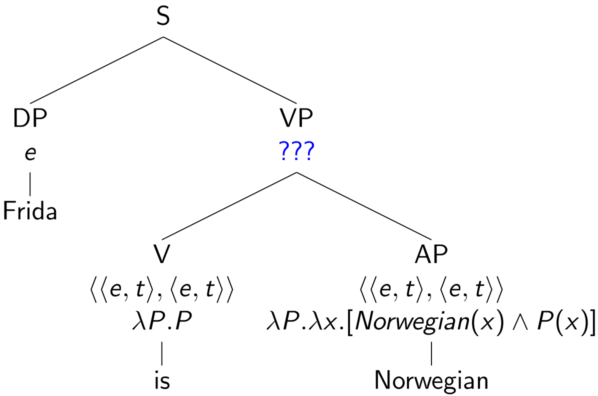


Adjectives: predicative position (cont.)



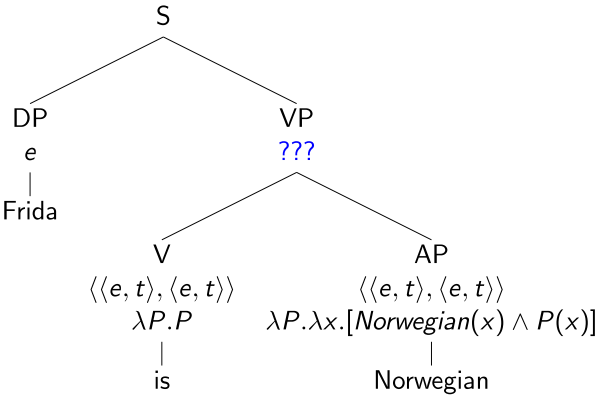

S
DP
e
Frida
VP
???
V
$\langle\langle e, t\rangle, \langle e, t\rangle\rangle$
$\lambda P.P$
is
AP
$\langle\langle e, t\rangle, \langle e, t\rangle\rangle$
$\lambda P.\lambda x.[Norwegian(x) \wedge P(x)]$
Norwegian

Adjectives: predicative position (cont.)



A MODIFIER type analysis as above causes the problem.

Adjectives: predicative position (cont.)



A MODIFIER type analysis as above causes the problem.

TYPE MISMATCH: two sister nodes in a tree have denotations that are not of the right types for any composition rule to combine them.

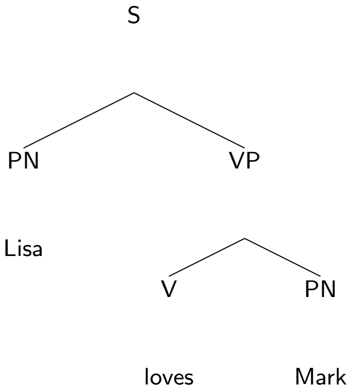
Compositional Rule 1 (recap)

Composition Rule 1: Function Application

Let γ be a syntax tree whose sub-trees are α and β where:

- $\alpha \rightsquigarrow \alpha'$ where α' has type $\langle \sigma, \tau \rangle$
- $\beta \rightsquigarrow \beta'$ where β' has type $\langle \sigma \rangle$

then
$$\gamma \rightsquigarrow \alpha'(\beta')$$



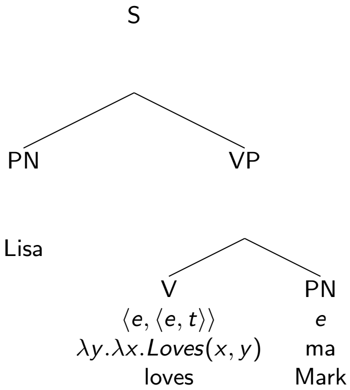
Compositional Rule 1 (recap)

Composition Rule 1: Function Application

Let γ be a syntax tree whose sub-trees are α and β where:

- $\alpha \rightsquigarrow \alpha'$ where α' has type $\langle \sigma, \tau \rangle$
- $\beta \rightsquigarrow \beta'$ where β' has type $\langle \sigma \rangle$

then $$\gamma \rightsquigarrow \alpha'(\beta')$$



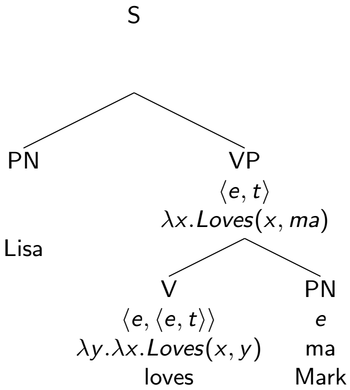
Compositional Rule 1 (recap)

Composition Rule 1: Function Application

Let γ be a syntax tree whose sub-trees are α and β where:

- $\alpha \rightsquigarrow \alpha'$ where α' has type $\langle \sigma, \tau \rangle$
- $\beta \rightsquigarrow \beta'$ where β' has type $\langle \sigma \rangle$

then
$$\gamma \rightsquigarrow \alpha'(\beta')$$



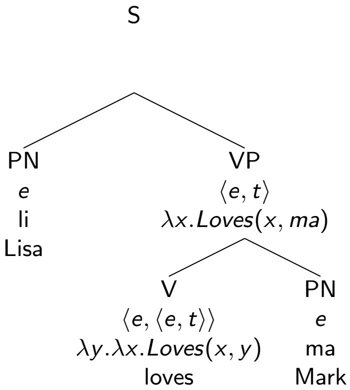
Compositional Rule 1 (recap)

Composition Rule 1: Function Application

Let γ be a syntax tree whose sub-trees are α and β where:

- $\alpha \rightsquigarrow \alpha'$ where α' has type $\langle\sigma, \tau\rangle$
- $\beta \rightsquigarrow \beta'$ where β' has type $\langle\sigma\rangle$

then
$$\gamma \rightsquigarrow \alpha'(\beta')$$



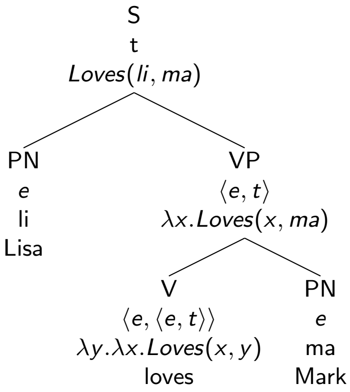
Compositional Rule 1 (recap)

Composition Rule 1: Function Application

Let γ be a syntax tree whose sub-trees are α and β where:

- $\alpha \rightsquigarrow \alpha'$ where α' has type $\langle \sigma, \tau \rangle$
- $\beta \rightsquigarrow \beta'$ where β' has type $\langle \sigma \rangle$

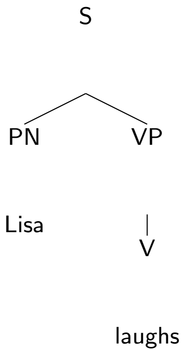
then $$\gamma \rightsquigarrow \alpha'(\beta')$$



Composition Rule 2 (recap)

Composition Rule 2: Non-branching Nodes

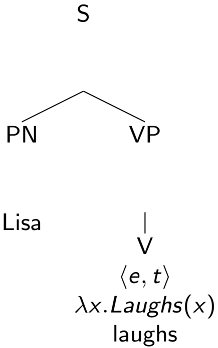
If β is a tree whose only daughter is α, where $\alpha \rightsquigarrow \alpha'$ then $\beta \rightsquigarrow \alpha'$



Composition Rule 2 (recap)

Composition Rule 2: Non-branching Nodes

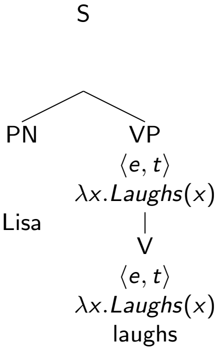
If β is a tree whose only daughter is α, where $\alpha \rightsquigarrow \alpha'$ then $\beta \rightsquigarrow \alpha'$



Composition Rule 2 (recap)

Composition Rule 2: Non-branching Nodes

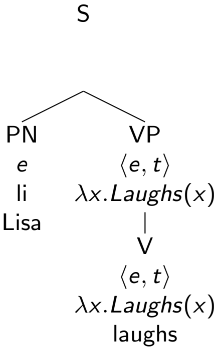
If β is a tree whose only daughter is α, where $\alpha \rightsquigarrow \alpha'$ then $\beta \rightsquigarrow \alpha'$



Composition Rule 2 (recap)

Composition Rule 2: Non-branching Nodes

If β is a tree whose only daughter is α, where $\alpha \rightsquigarrow \alpha'$ then $\beta \rightsquigarrow \alpha'$



Composition Rule 2 (recap)

Composition Rule 2: Non-branching Nodes

If β is a tree whose only daughter is α, where $\alpha \rightsquigarrow \alpha'$ then $\beta \rightsquigarrow \alpha'$

S
t
Laughs(*li*)

PN
e
li
Lisa

VP
$\langle e, t\rangle$
$\lambda x.Laughs(x)$

V
$\langle e, t\rangle$
$\lambda x.Laughs(x)$
laughs

Adjectives: solutions

Solutions:

Adjectives: solutions

Solutions:

(i) generate two translations: one of $\langle e, t\rangle$ for predicative positions and $\langle\langle e, t\rangle, \langle e, t\rangle\rangle$ for attributive positions;

Adjectives: solutions

Solutions:

(i) generate two translations: one of $\langle e, t\rangle$ for predicative positions and $\langle\langle e, t\rangle, \langle e, t\rangle\rangle$ for attributive positions;

(ii) introduce a new composition rule and give intersective adjectives one single translation

Adjectives: solutions

Solutions:

(i) generate two translations: one of $\langle e, t\rangle$ for predicative positions and $\langle\langle e, t\rangle, \langle e, t\rangle\rangle$ for attributive positions;

(ii) introduce a new composition rule and give intersective adjectives one single translation

for (i) take one translation to be basic and derive the other one from it with the help of either

Adjectives: solutions

Solutions:

(i) generate two translations: one of $\langle e, t\rangle$ for predicative positions and $\langle\langle e, t\rangle, \langle e, t\rangle\rangle$ for attributive positions;

(ii) introduce a new composition rule and give intersective adjectives one single translation

for (i) take one translation to be basic and derive the other one from it with the help of either

a TYPE-SHIFTING RULE (invisible to the syntactic component of the grammar) or

Adjectives: solutions

Solutions:

(i) generate two translations: one of $\langle e, t\rangle$ for predicative positions and $\langle\langle e, t\rangle, \langle e, t\rangle\rangle$ for attributive positions;

(ii) introduce a new composition rule and give intersective adjectives one single translation

for (i) take one translation to be basic and derive the other one from it with the help of either

a TYPE-SHIFTING RULE (invisible to the syntactic component of the grammar) or

a SILENT OPERATOR (a reflection in the syntax)

- A
  $\langle\langle e,t\rangle,\langle e,t\rangle\rangle$
  $\lambda P.\lambda x.[Reasonable(x) \wedge P(x)]$
  - $\langle\langle e,t\rangle,\langle\langle e,t\rangle,\langle e,t\rangle\rangle\rangle$
    $\lambda P'.\lambda P.\lambda x.[P'(x) \wedge P(x)]$
    - MOD
  - A
    $\langle e,t\rangle$
    $\lambda x.Reasonable(x)$
    - reasonable

Adjectives: Type Shifting

Type-Shifting Rule 1: Predicate-to-modifier shift

If $\alpha \rightsquigarrow \alpha'$, where α' is of type $\langle e, t \rangle$,
then $\alpha \rightsquigarrow \lambda P.[\alpha'(x) \wedge P(x)]$ (as long as P and x are not free in α; in that case, use different variables of the same type).

Adjectives: Type Shifting

Type-Shifting Rule 1: Predicate-to-modifier shift

If $\alpha \rightsquigarrow \alpha'$, where α' is of type $\langle e, t\rangle$,
then $\alpha \rightsquigarrow \lambda P.[\alpha'(x) \wedge P(x)]$ (as long as P and x are not free in α; in that case, use different variables of the same type).

$$\begin{array}{c} \text{A} \\ \langle\langle e, t\rangle, \langle e, t\rangle\rangle \\ \lambda P.\lambda x.[Reasonable(x) \wedge P(x)] \\ \Uparrow_{MOD} \\ \text{A} \\ \langle e, t\rangle \\ \lambda x.Reasonable(x) \\ | \\ \text{reasonable} \end{array}$$

Predicate Modification

(ii) assumption is that all intersective adjectives have translations of a single type, and we eliminate type mismatches via a new composition rule

Predicate Modification

(ii) assumption is that all intersective adjectives have translations of a single type, and we eliminate type mismatches via a new composition rule

Composition Rule 3: Predicate Modification

If:

- γ is a tree whose only two subtrees are α and β
- $\alpha \rightsquigarrow \alpha'$
- $\beta \rightsquigarrow \beta'$
- α' and β' are of type $\langle e, t \rangle$

Then:

$\gamma \rightsquigarrow \lambda u.[\alpha'(u) \wedge \beta'(u)]$

where u is a variable of type e that does not occur free in α' or β'.

Predicate Modification (example)

NP
$\langle e, t \rangle$
$\lambda x.[Reasonable(x) \wedge Doubt(x)]$

AP
$\langle e, t \rangle$
$\lambda x.Reasonable(x)$
reasonable

NP
$\langle e, t \rangle$
$\lambda x.Doubt(x)$
doubt

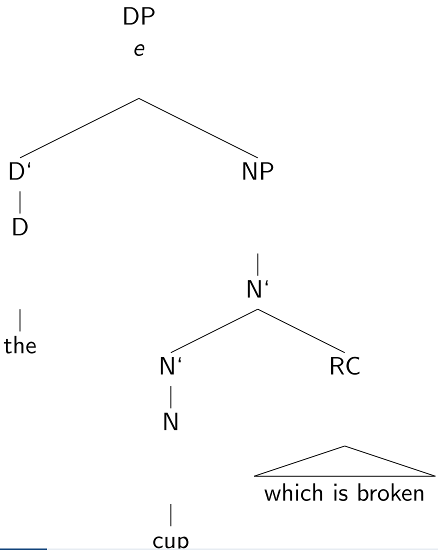
Relative Clauses

Examples

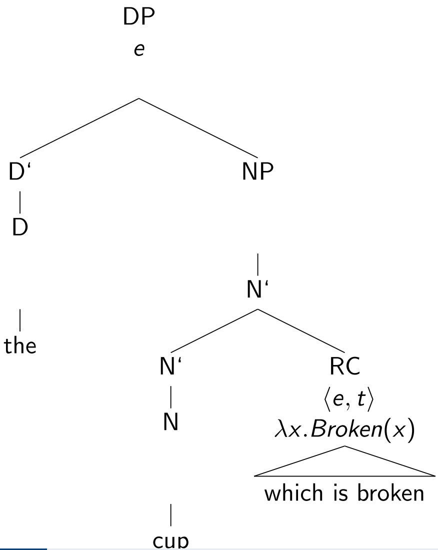
reasonable doubt $\approx$ doubt which is reasonable
the broken cup $\approx$ the cup which is broken

Apply Predicate Modification and use ι operator for definite expression

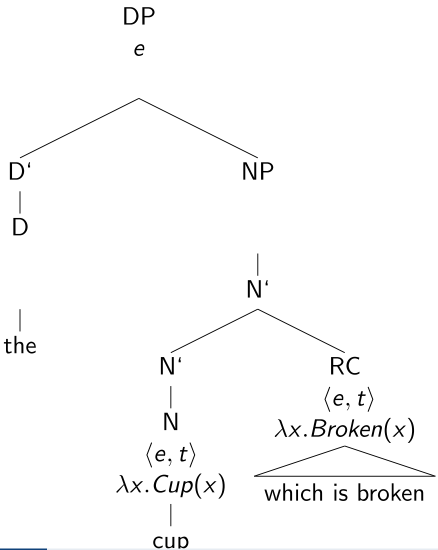
Relative Clauses (example)



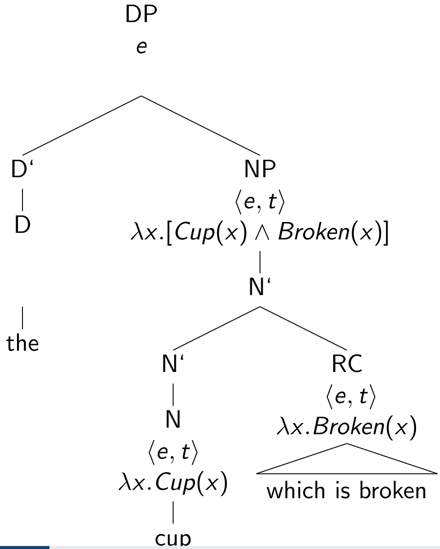
Relative Clauses (example)



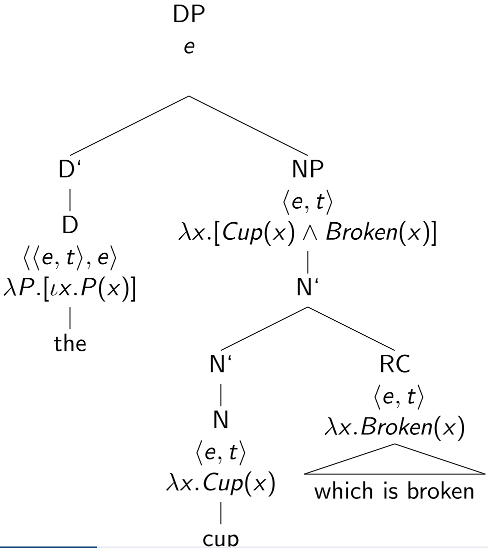
Relative Clauses (example)



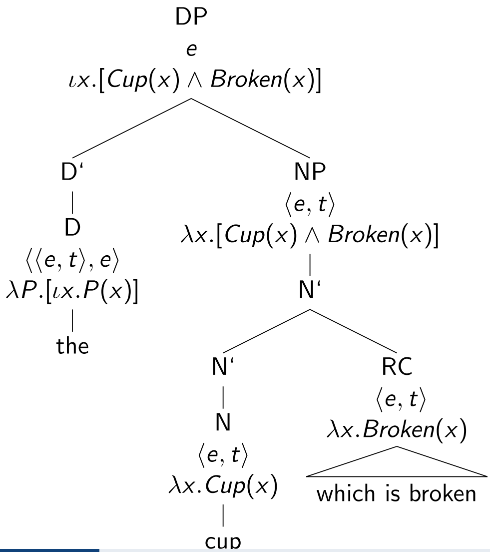
Relative Clauses (example)



Relative Clauses (example)

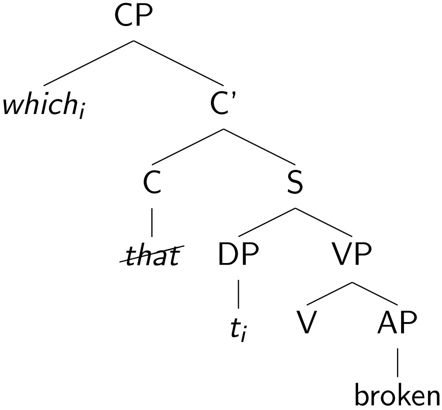


Relative Clauses (example)



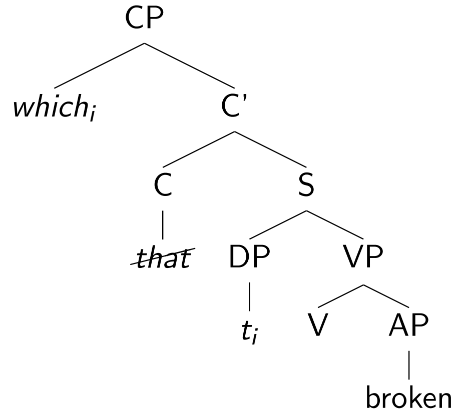
Relative Clauses: TRACE

TRACE or in contemporary theories of syntax often use the term UNPRONOUNCED COPY



Relative Clauses: TRACE

TRACE or in contemporary theories of syntax often use the term UNPRONOUNCED COPY



CP stands for 'Complementizer Phrase', it is headed by a complementizer in relative clauses. The wh-word occupies the so-called 'specifier' position of CP (sister to C'). Specifier comes from X-bar theory of syntax, where all phrases are of the form $[_{XP}(\textit{specifier})[_{X'}[_{X}(\textit{complement})]]]$.

Relative Clauses (cont.)

The key assumptions are the following:

- Relative clauses are formed through a movement operation that leaves a trace.
- Traces are translated as variables.
- A relative clause is interpreted by introducing a lambda operator that binds this variable.

Relative Clauses (cont.)

The key assumptions are the following:

- Relative clauses are formed through a movement operation that leaves a trace.
- Traces are translated as variables.
- A relative clause is interpreted by introducing a lambda operator that binds this variable.

The denotation of the variable v_9 will depend on an assignment : $[\![v_9]\!]^{M,g} = g(v_9)$

Relative Clauses (cont.)

The key assumptions are the following:

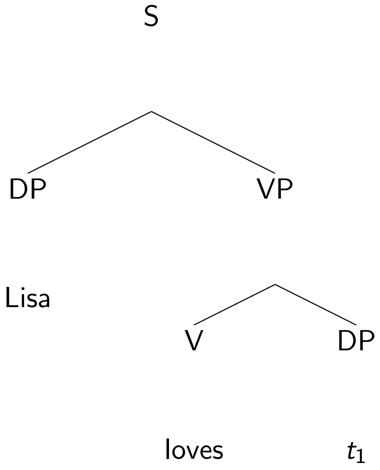
- Relative clauses are formed through a movement operation that leaves a trace.
- Traces are translated as variables.
- A relative clause is interpreted by introducing a lambda operator that binds this variable.

The denotation of the variable v_9 will depend on an assignment : $[\![v_9]\!]^{M,g} = g(v_9)$

Composition Rule 4: Pronouns and Trace Rule

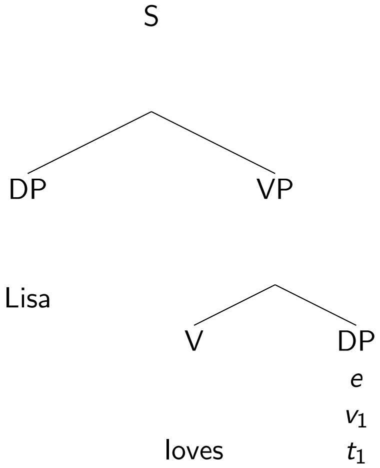
If α is an indexed trace or pronoun, $\alpha_i \rightsquigarrow v_i$

Lisa loves t_1



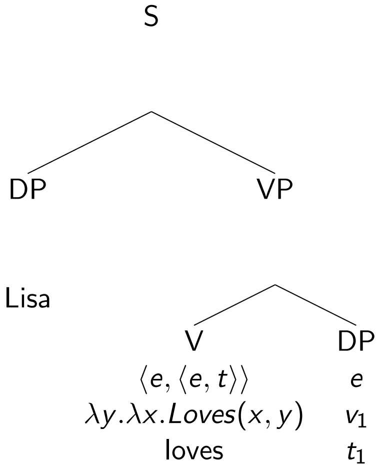
Pronouns

Lisa loves t_1



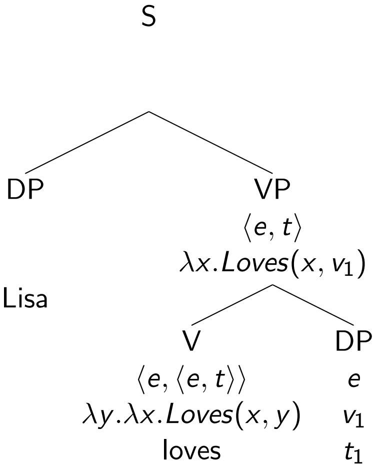
Pronouns

Lisa loves t_1



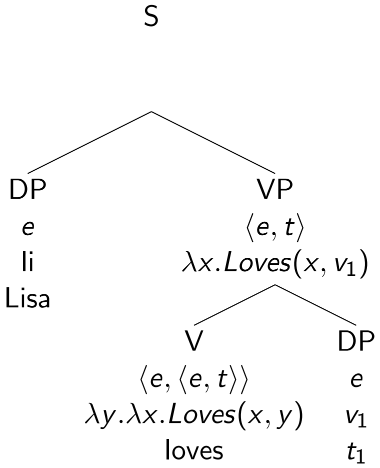
Pronouns

Lisa loves t_1

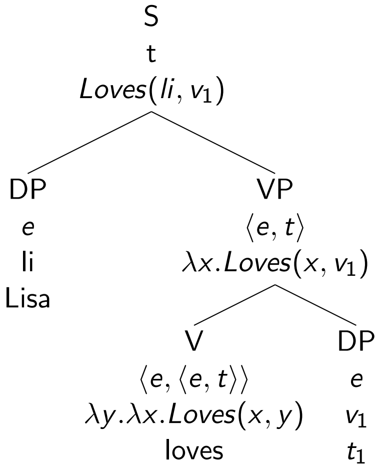


Pronouns

Lisa loves t_1



Lisa loves t_1



Relative Clauses: Predication Abstraction

CP should be of type $\langle e, t \rangle$ therefore one more composition rule

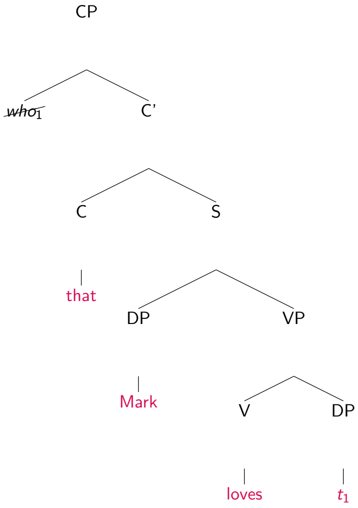
Composition Rule 5: Predicate Abstraction

If:

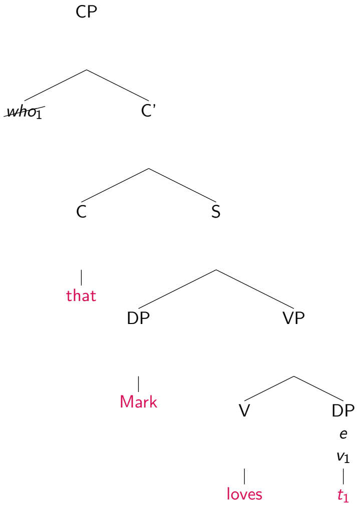
- γ is a tree whose only two subtrees are α_i and β
- $\beta \rightsquigarrow \beta'$
- β' is an expression of type t

Then $\gamma \rightsquigarrow \lambda v_i.\beta'$

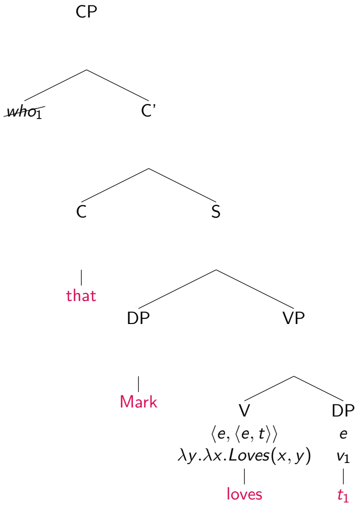
Relative Clauses: Predication Abstraction (cont.)



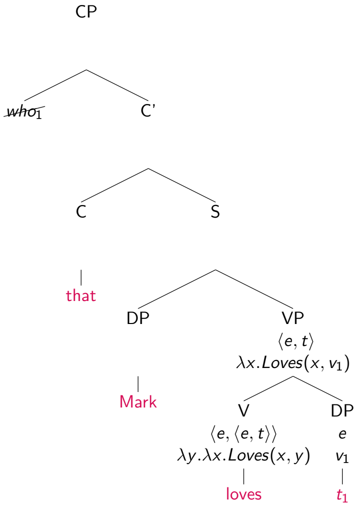
Relative Clauses: Predication Abstraction (cont.)



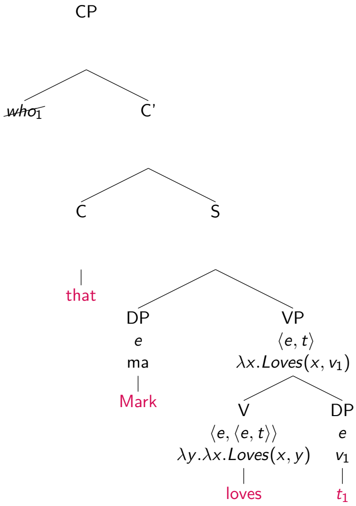
Relative Clauses: Predication Abstraction (cont.)



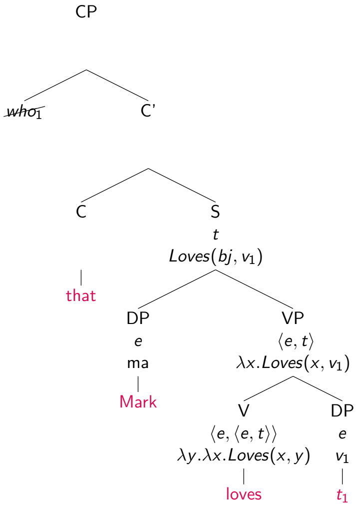
Relative Clauses: Predication Abstraction (cont.)



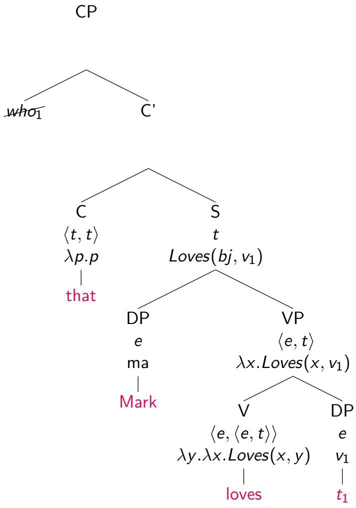
Relative Clauses: Predication Abstraction (cont.)



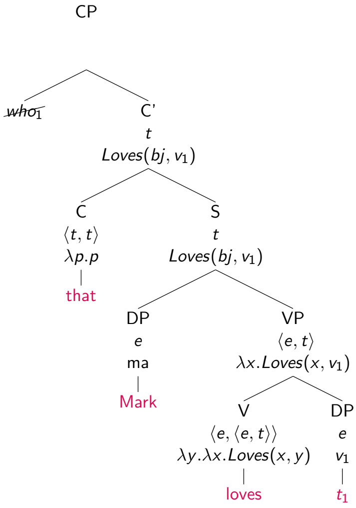
Relative Clauses: Predication Abstraction (cont.)



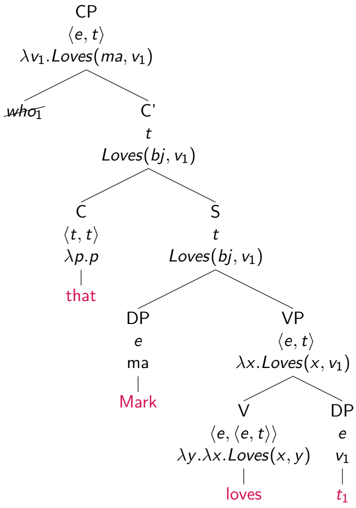
Relative Clauses: Predication Abstraction (cont.)



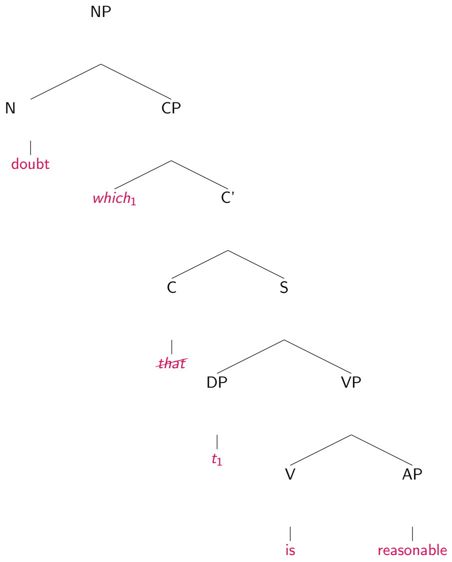
Relative Clauses: Predication Abstraction (cont.)



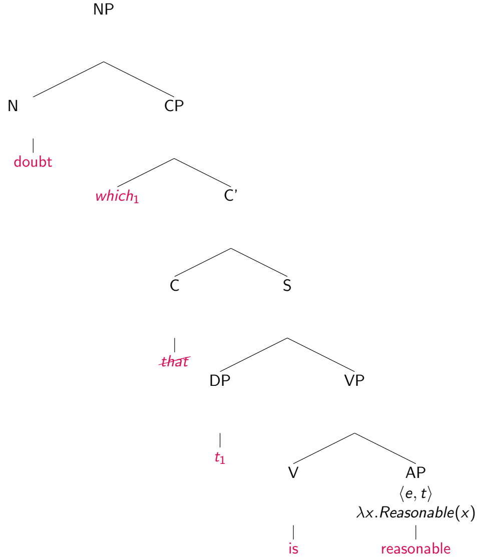
Relative Clauses: Predication Abstraction (cont.)



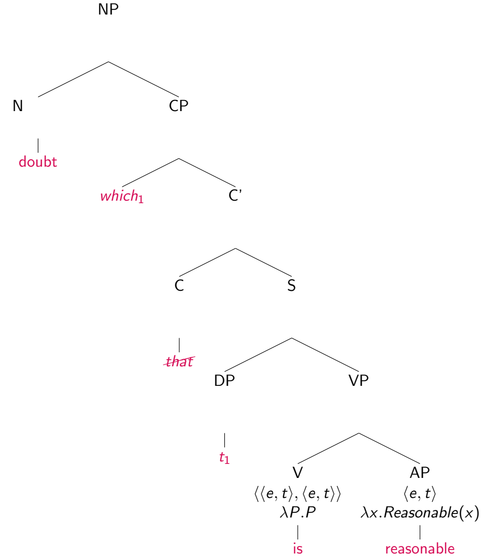
Relative Clauses: Predication Abstraction (cont.)



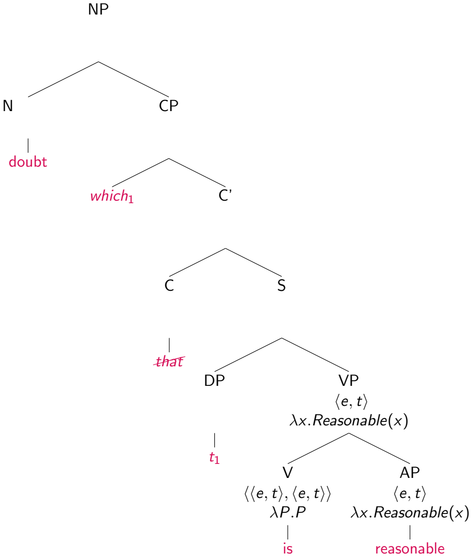
Relative Clauses: Predication Abstraction (cont.)



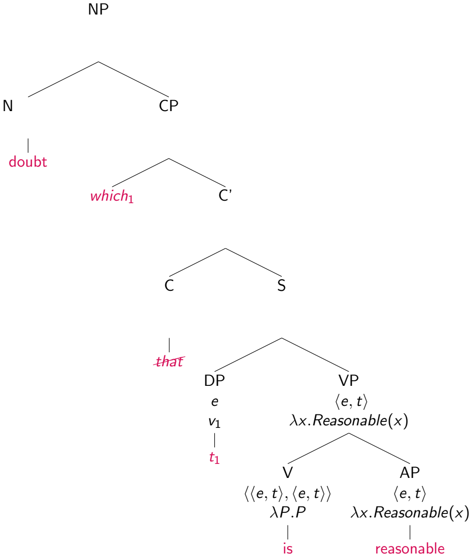
Relative Clauses: Predication Abstraction (cont.)



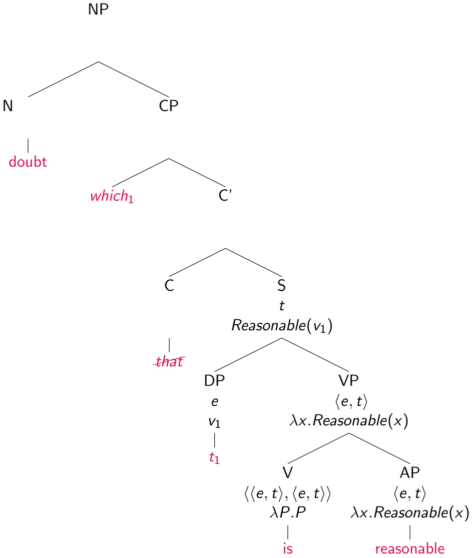
Relative Clauses: Predication Abstraction (cont.)



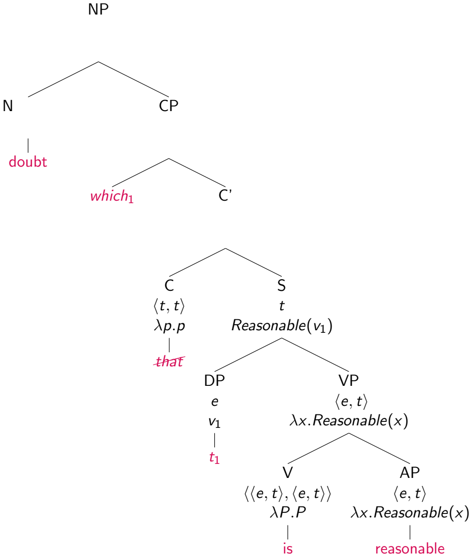
Relative Clauses: Predication Abstraction (cont.)



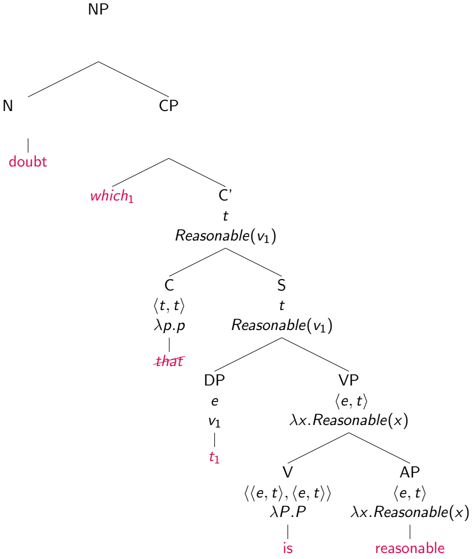
Relative Clauses: Predication Abstraction (cont.)



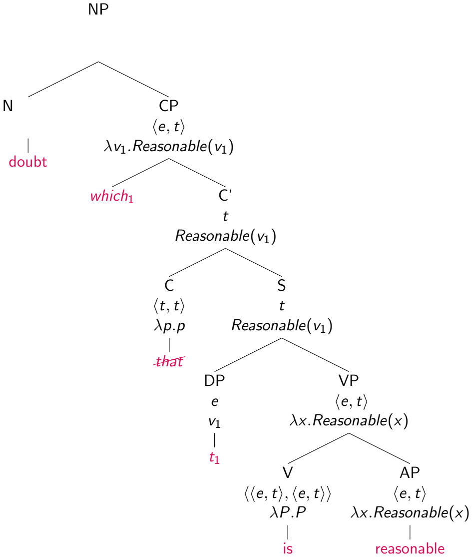
Relative Clauses: Predication Abstraction (cont.)



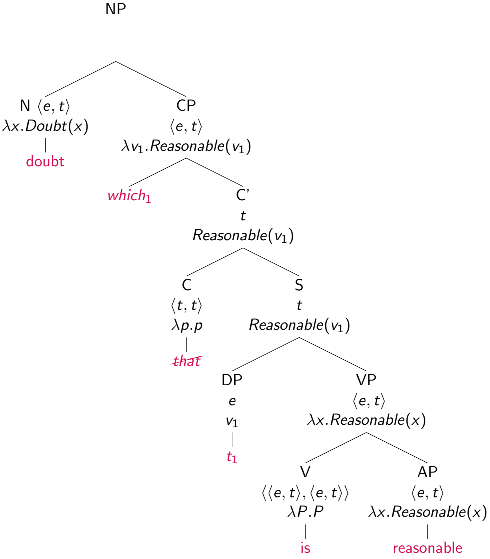
Relative Clauses: Predication Abstraction (cont.)

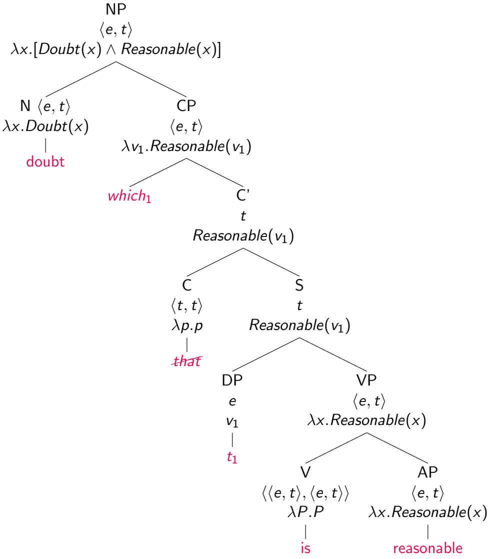


Relative Clauses: Predication Abstraction (cont.)



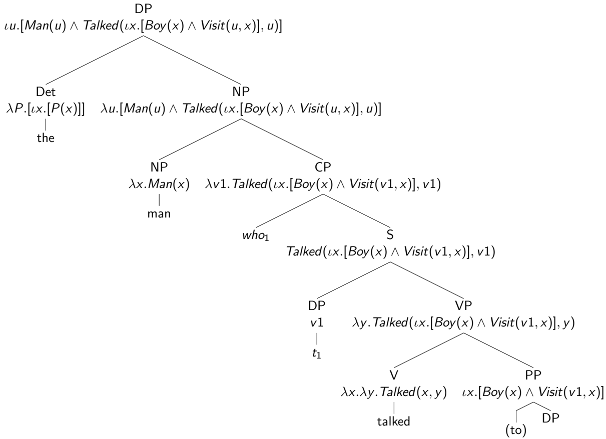
Relative Clauses: Predication Abstraction (cont.)



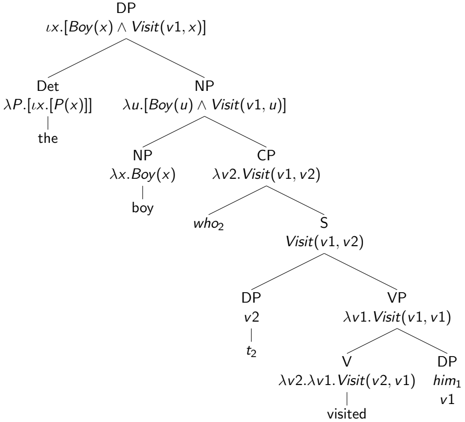

NP
$\langle e,t \rangle$
$\lambda x.[Doubt(x) \wedge Reasonable(x)]$
N $\langle e,t \rangle$
$\lambda x.Doubt(x)$
doubt
CP
$\langle e,t \rangle$
$\lambda v_1.Reasonable(v_1)$
$which_1$
C'
t
$Reasonable(v_1)$
C
$\langle t,t \rangle$
$\lambda p.p$
~~that~~
S
t
$Reasonable(v_1)$
DP
e
v_1
t_1
VP
$\langle e,t \rangle$
$\lambda x.Reasonable(x)$
V
$\langle \langle e,t \rangle, \langle e,t \rangle \rangle$
$\lambda P.P$
is
AP
$\langle e,t \rangle$
$\lambda x.Reasonable(x)$
reasonable

Relative Clauses: complex example

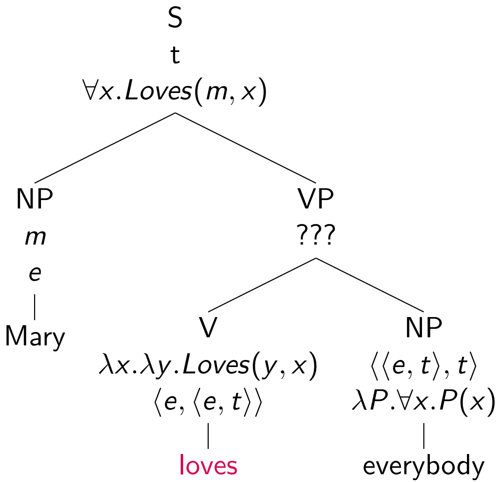
The man who talked to the boy who visited him



(cont.)



Quantification: Object Position



Quantifier Raising

Right Translation

everybody $\rightsquigarrow \lambda P \forall x.P(x)$
loves $\rightsquigarrow \forall x.Loves(m,x)$
$[\lambda P \forall x.P(x)](\lambda x.Loves(m,x)) \equiv \forall x.Loves(m,x)$

Quantifier Raising

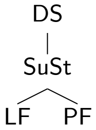
Right Translation

everybody $\rightsquigarrow \lambda P \forall x.P(x)$
loves $\rightsquigarrow \forall x.Loves(m, x)$
$[\lambda P \forall x.P(x)](\lambda x.Loves(m, x)) \equiv \forall x.Loves(m, x)$

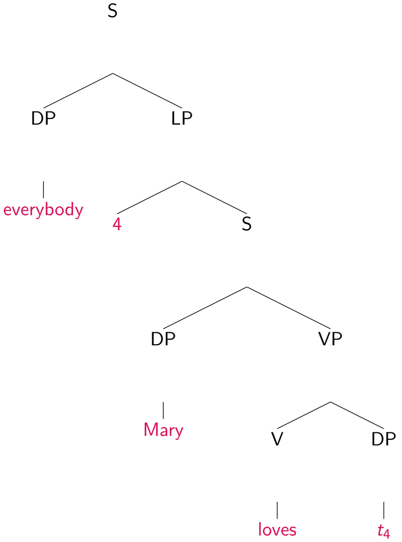
Solution: QUANTIFIER RAISING: syntactic transformation that moves a quantifier, an expression of type $\langle\langle e, t\rangle, t\rangle$, to a position in the tree where it can be interpreted, and leaves a DP trace in its previous position.

Quantifier Raising: Levels of Representation

- Deep Structure (DS): Where active sentences (John kissed Mary) look the same as passive sentences (Mary was kissed by John), and wh- words are in their original positions. For example, Who did you see? is You did see who? at Deep Structure.
- Surface Structure (SuSt): Where the order of the words corresponds to what we see or hear (after e.g. passivization or wh-movement)
- Phonological Form (PF): Where the words are realized as sounds (after e.g. deletion processes)
- Logical Form(LF): The input to semantic interpretation (after e.g. Quantifier Raising)

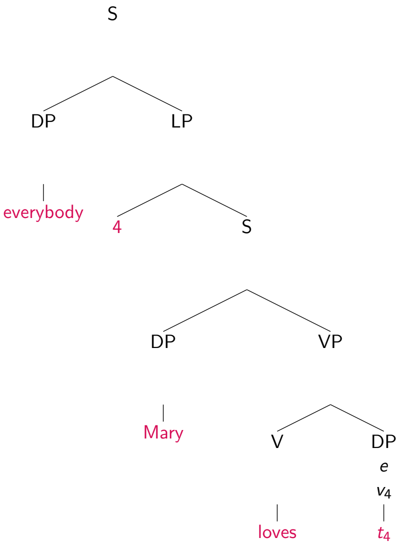


Quantifier Raising (cont.)



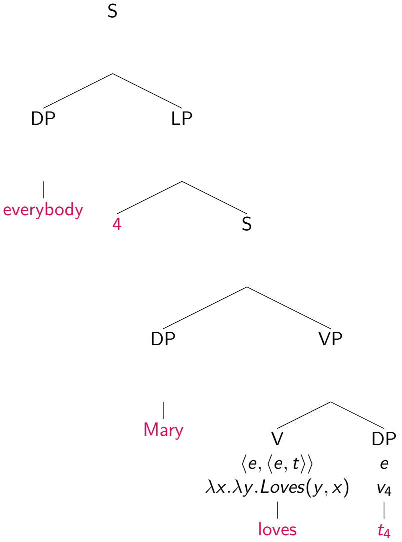
The LP node is a semantic fiction without syntactic evidence to support it; it provides a place for the Predicate Abstraction rule to apply

Quantifier Raising (cont.)



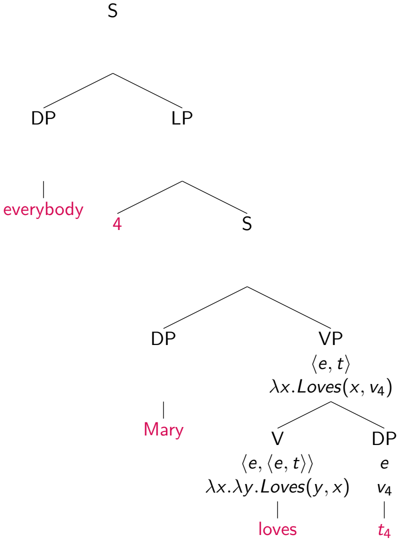
The LP node is a semantic fiction without syntactic evidence to support it; it provides a place for the Predicate Abstraction rule to apply

Quantifier Raising (cont.)



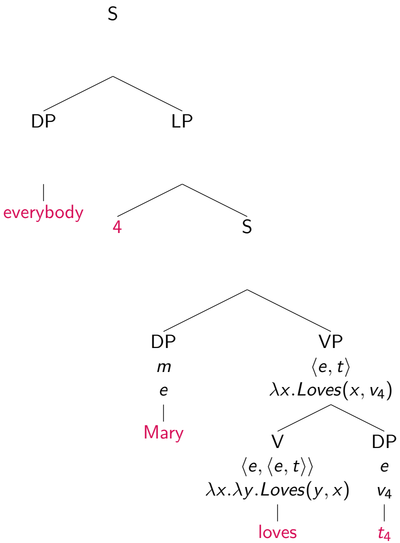
The LP node is a semantic fiction without syntactic evidence to support it; it provides a place for the Predicate Abstraction rule to apply

Quantifier Raising (cont.)



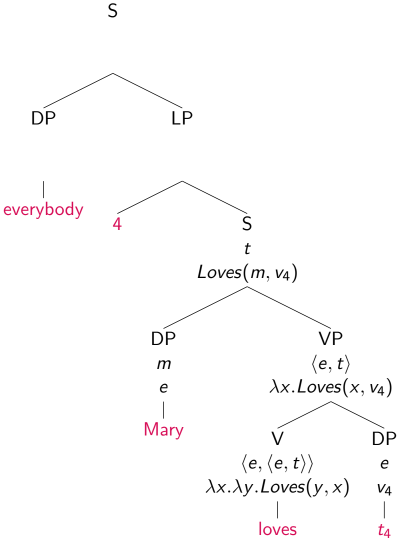
The LP node is a semantic fiction without syntactic evidence to support it; it provides a place for the Predicate Abstraction rule to apply

Quantifier Raising (cont.)



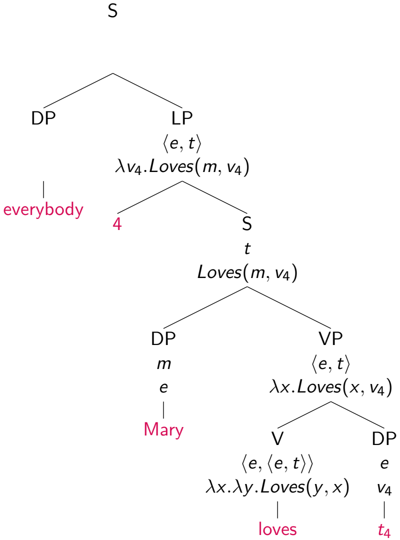
The LP node is a semantic fiction without syntactic evidence to support it; it provides a place for the Predicate Abstraction rule to apply

Quantifier Raising (cont.)



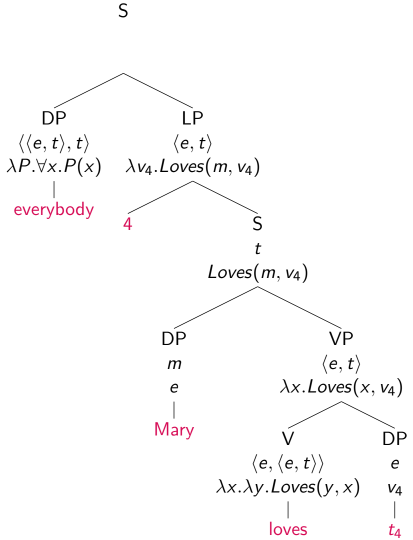
The LP node is a semantic fiction without syntactic evidence to support it; it provides a place for the Predicate Abstraction rule to apply

Quantifier Raising (cont.)



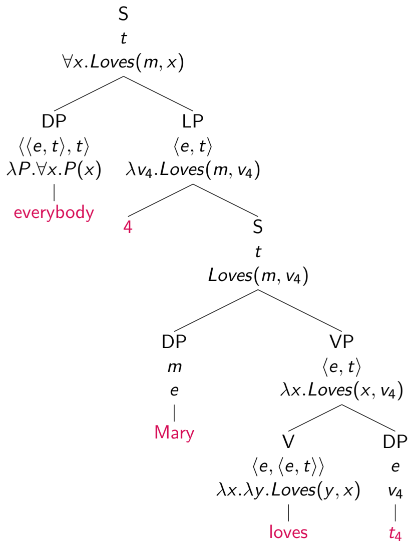
The LP node is a semantic fiction without syntactic evidence to support it; it provides a place for the Predicate Abstraction rule to apply

Quantifier Raising (cont.)



The LP node is a semantic fiction without syntactic evidence to support it; it provides a place for the Predicate Abstraction rule to apply

Quantifier Raising (cont.)



The LP node is a semantic fiction without syntactic evidence to support it; it provides a place for the Predicate Abstraction rule to apply

Quantifiers: Type Shifting Approach

QR is not an option in non-transformational generative approaches like HPSG (Pollard & Sag, 1994) or LFG (Bresnan, 2001)

Quantifiers: Type Shifting Approach

QR is not an option in non-transformational generative approaches like HPSG (Pollard & Sag, 1994) or LFG (Bresnan, 2001)

Direct Compositionality: syntax and semantics work in tandem

Quantifiers: Type Shifting Approach

QR is not an option in non-transformational generative approaches like HPSG (Pollard & Sag, 1994) or LFG (Bresnan, 2001)

Direct Compositionality: syntax and semantics work in tandem

Storage Mechanisms: syntactic node associated with a set of quantifiers are "in store" and when a node of type t is reached, quantifiers can be "discharged"

Quantifiers: Type Shifting Approach

QR is not an option in non-transformational generative approaches like HPSG (Pollard & Sag, 1994) or LFG (Bresnan, 2001)

Direct Compositionality: syntax and semantics work in tandem

Storage Mechanisms: syntactic node associated with a set of quantifiers are "in store" and when a node of type t is reached, quantifiers can be "discharged"

Type Shifting: repair the mismatch, e.g. predicate of type $\langle e, \langle e, t \rangle \rangle$ can be converted into one that is expecting a quantifier for its first or second argument, or both

Quantifiers: Type Shifting Approach

QR is not an option in non-transformational generative approaches like HPSG (Pollard & Sag, 1994) or LFG (Bresnan, 2001)
Direct Compositionality: syntax and semantics work in tandem
Storage Mechanisms: syntactic node associated with a set of quantifiers are "in store" and when a node of type t is reached, quantifiers can be "discharged"
Type Shifting: repair the mismatch, e.g. predicate of type $\langle e, \langle e, t \rangle \rangle$ can be converted into one that is expecting a quantifier for its first or second argument, or both

A general type-shifting scheme is called ARGUMENT RAISING (Hendriks (1993)

Quantifiers: Type Shifting Approach

QR is not an option in non-transformational generative approaches like HPSG (Pollard & Sag, 1994) or LFG (Bresnan, 2001)
Direct Compositionality: syntax and semantics work in tandem
Storage Mechanisms: syntactic node associated with a set of quantifiers are "in store" and when a node of type t is reached, quantifiers can be "discharged"
Type Shifting: repair the mismatch, e.g. predicate of type $\langle e, \langle e, t\rangle\rangle$ can be converted into one that is expecting a quantifier for its first or second argument, or both

A general type-shifting scheme is called ARGUMENT RAISING (Hendriks (1993)

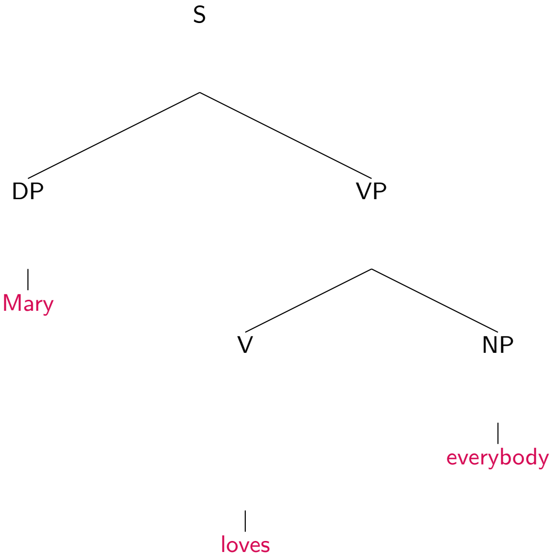
Type Shifting Rule 2: Object Raising (RAISE-O)

If an English expression α is translated into a logical expression α' of type $\langle e, \langle \alpha, t\rangle\rangle$ for any type α, then α also has a translation of type $\langle\langle\langle e, t\rangle, t\rangle, \langle \alpha, t\rangle\rangle$ of the following form:

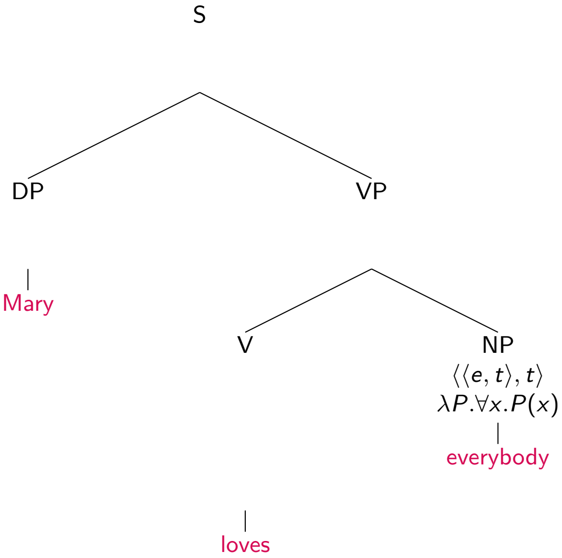
$$\lambda Q_{\langle\langle e,t\rangle,t\rangle} \lambda x_\alpha . Q(\lambda y.\alpha'(y)(x))$$

(unless Q, y or x occurs in α'; in that case, use different variables).

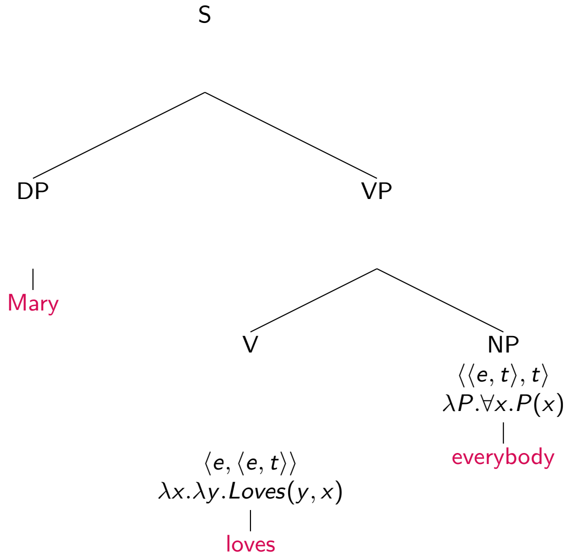
Type Shifting: RAISE-O



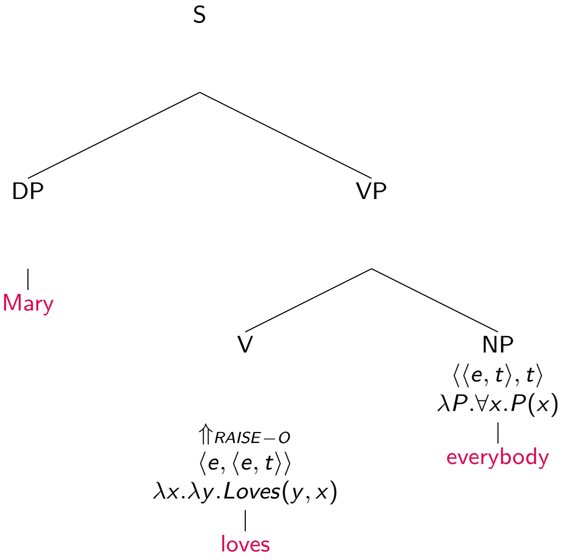
Type Shifting: RAISE-O



Type Shifting: RAISE-O



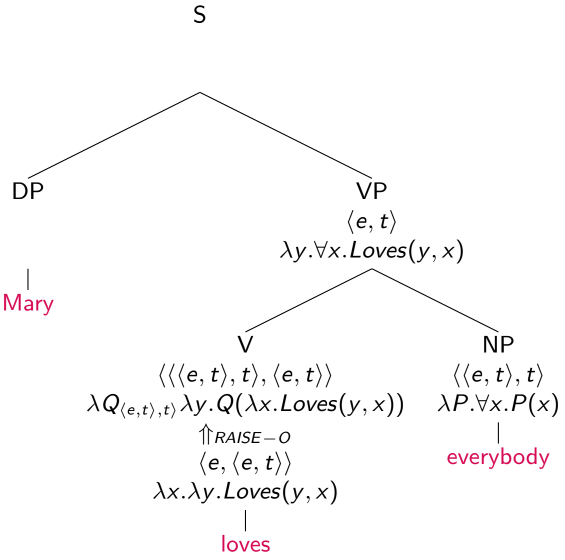
Type Shifting: RAISE-O



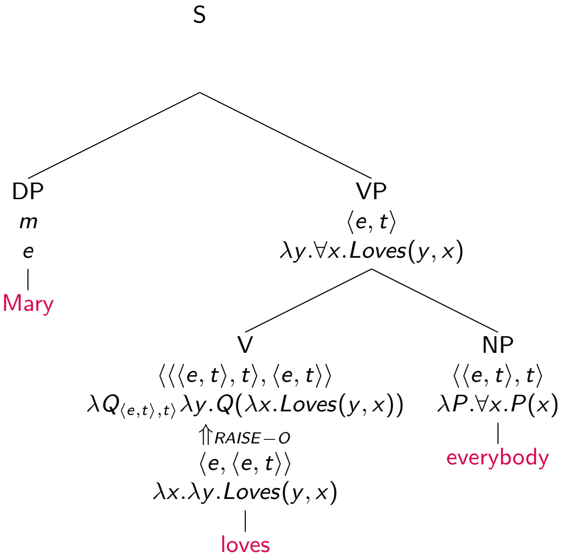
Type Shifting: RAISE-O

- S
  - DP
    - Mary
  - VP
    - V
      - $\langle\langle\langle e,t\rangle,t\rangle,\langle e,t\rangle\rangle$
      - $\lambda Q_{\langle\langle e,t\rangle,t\rangle}\lambda y.Q(\lambda x.Loves(y,x))$
      - $\Uparrow RAISE-O$
      - $\langle e,\langle e,t\rangle\rangle$
      - $\lambda x.\lambda y.Loves(y,x)$
        - loves
    - NP
      - $\langle\langle e,t\rangle,t\rangle$
      - $\lambda P.\forall x.P(x)$
        - everybody

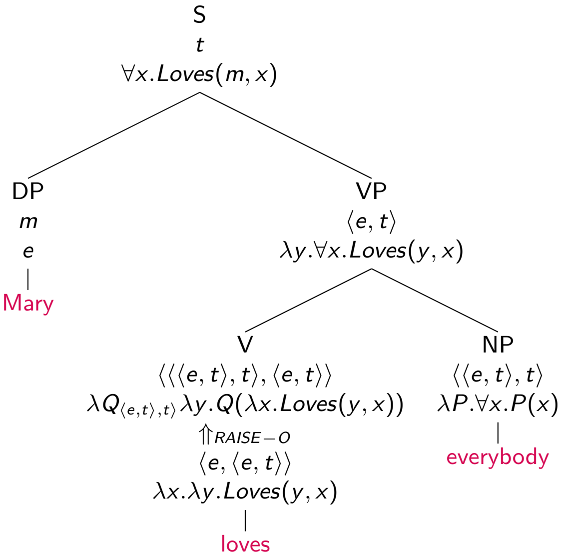
Type Shifting: RAISE-O



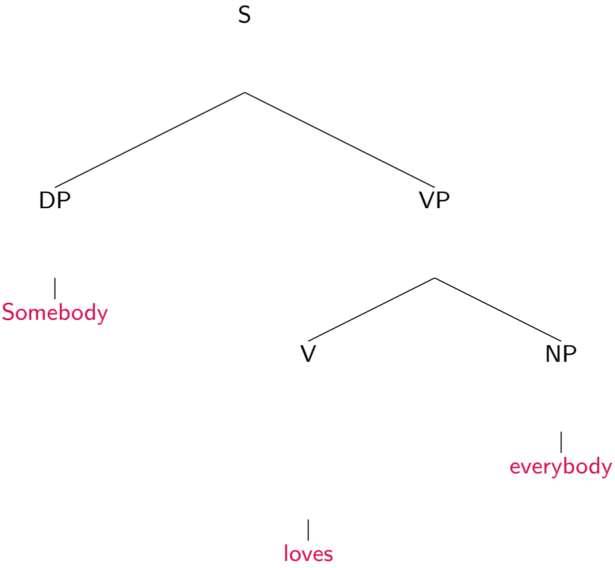
Type Shifting: RAISE-O



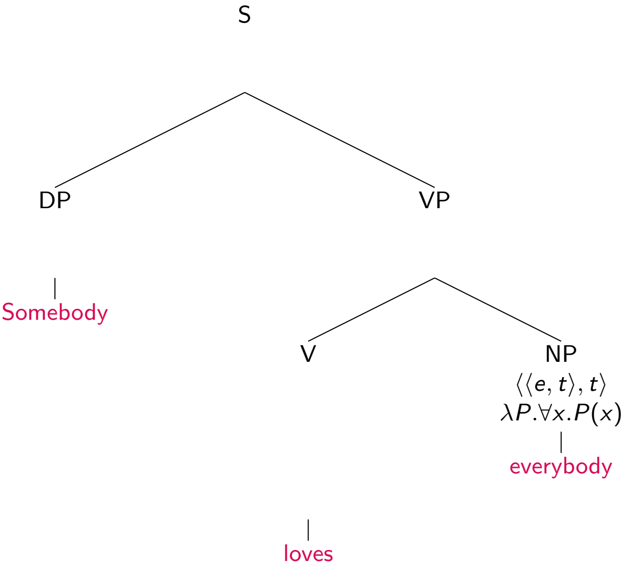
Type Shifting: RAISE-O



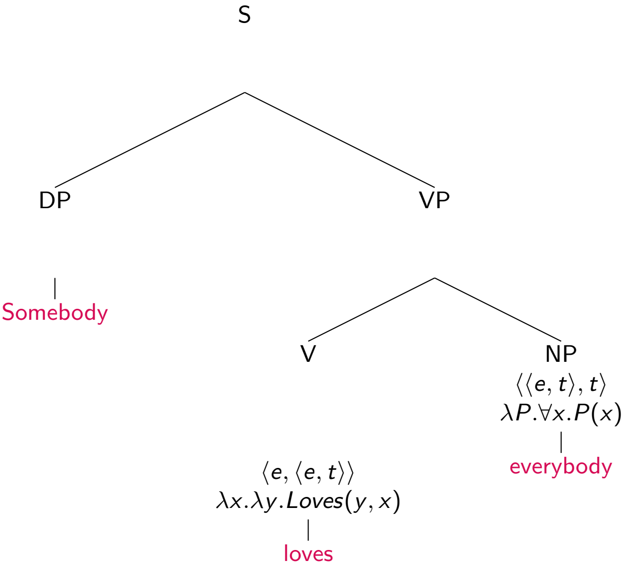
Type Shifting: Two Quantifiers



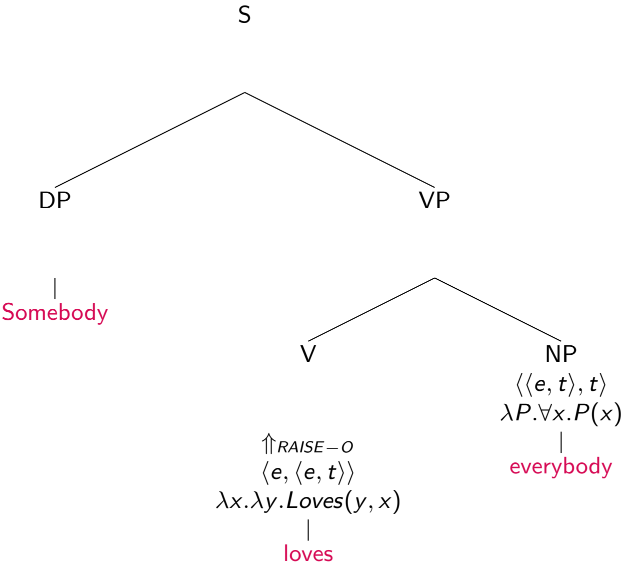
Type Shifting: Two Quantifiers



Type Shifting: Two Quantifiers



Type Shifting: Two Quantifiers



Type Shifting: Two Quantifiers

S

DP

Somebody

VP

V

$\langle\langle\langle e,t\rangle,t\rangle,\langle e,t\rangle\rangle$

$\lambda Q_{\langle\langle e,t\rangle,t\rangle}\lambda y.Q(\lambda x.Loves(y,x))$

$\Uparrow_{RAISE-O}$

$\langle e,\langle e,t\rangle\rangle$

$\lambda x.\lambda y.Loves(y,x)$

loves

NP

$\langle\langle e,t\rangle,t\rangle$

$\lambda P.\forall x.P(x)$

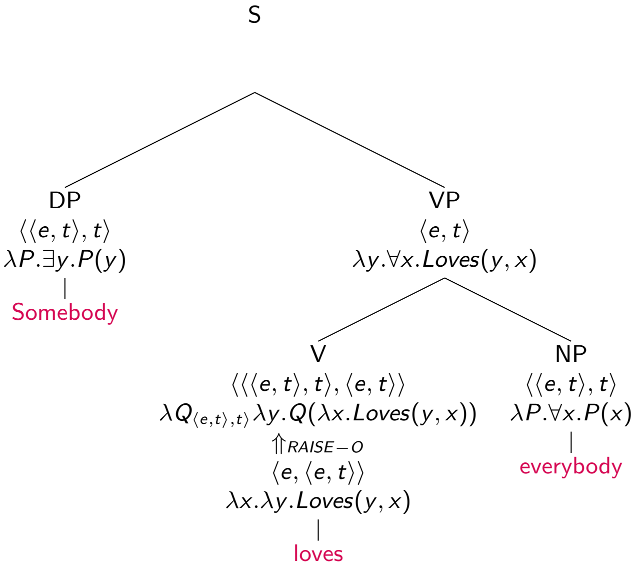
everybody

Type Shifting: Two Quantifiers

S

- DP
  - Somebody
- VP
  $\langle e, t \rangle$
  $\lambda y. \forall x. Loves(y, x)$
  - V
    $\langle \langle \langle e, t \rangle, t \rangle, \langle e, t \rangle \rangle$
    $\lambda Q_{\langle \langle e,t \rangle, t \rangle} \lambda y. Q(\lambda x. Loves(y, x))$
    $\Uparrow_{RAISE-O}$
    $\langle e, \langle e, t \rangle \rangle$
    $\lambda x. \lambda y. Loves(y, x)$
    - loves
  - NP
    $\langle \langle e, t \rangle, t \rangle$
    $\lambda P. \forall x. P(x)$
    - everybody

Type Shifting: Two Quantifiers



Type Shifting: Two Quantifiers

S
t
$\exists y.\forall x.Loves(y,x)$

DP
$\langle\langle e,t\rangle,t\rangle$
$\lambda P.\exists y.P(y)$
Somebody

VP
$\langle e,t\rangle$
$\lambda y.\forall x.Loves(y,x)$

V
$\langle\langle\langle e,t\rangle,t\rangle,\langle e,t\rangle\rangle$
$\lambda Q_{\langle\langle e,t\rangle,t\rangle}\lambda y.Q(\lambda x.Loves(y,x))$
$\Uparrow_{RAISE-O}$
$\langle e,\langle e,t\rangle\rangle$
$\lambda x.\lambda y.Loves(y,x)$
loves

NP
$\langle\langle e,t\rangle,t\rangle$
$\lambda P.\forall x.P(x)$
everybody

Type Shifting: Subject Raising

But what about inverse scope reading?

Type Shifting: Subject Raising

But what about inverse scope reading?

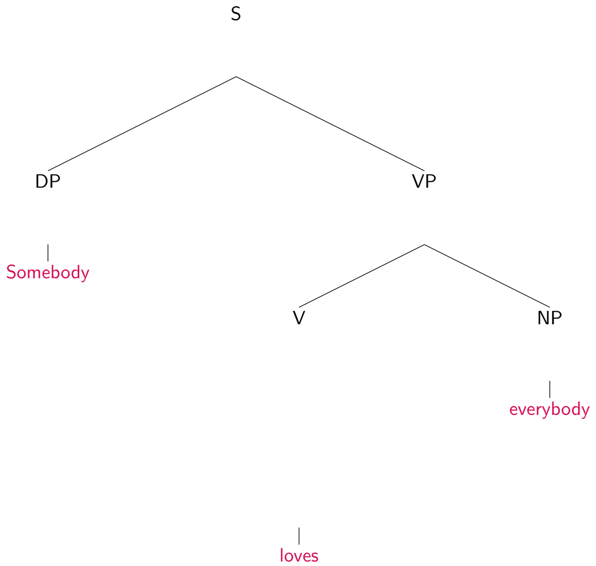
Type Shifting Rule 3: Subject Raising (RAISE-O)

If an English expression α is translated into a logical expression α' of type $\langle \alpha, \langle e, t \rangle \rangle$ for any type α, then α also has a translation of type $\langle \alpha \langle \langle \langle e, t \rangle, t \rangle, t \rangle \rangle$ of the following form:

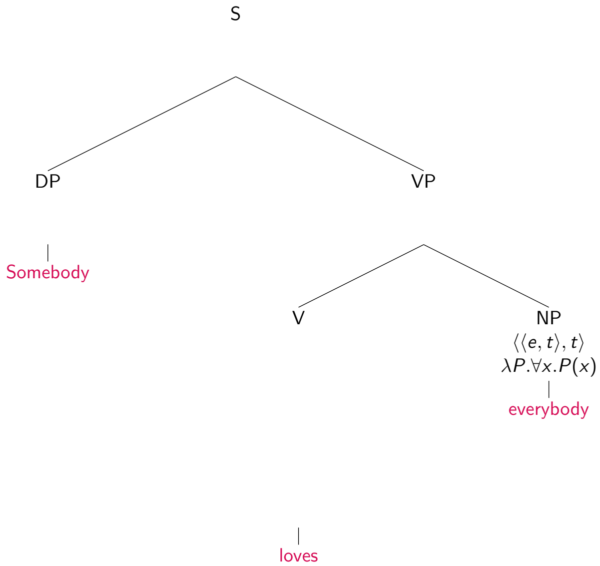
$$\lambda y_\alpha \lambda Q_{\langle \langle e,t \rangle, t \rangle} . Q(\lambda x_e . \alpha'(y)(x))$$

(unless Q, y or x occurs in α'; in that case, use different variables).

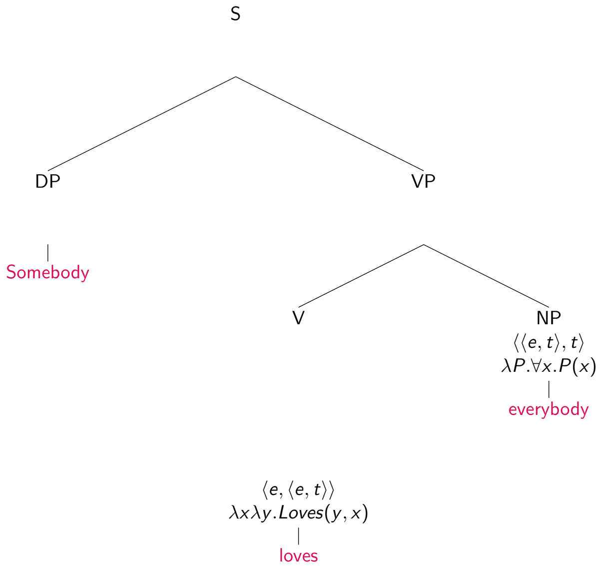
Type Shifting: RAISE-S



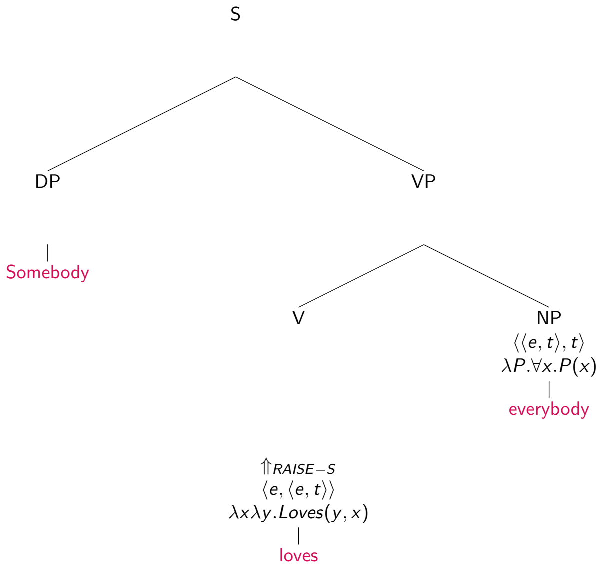
Type Shifting: RAISE-S



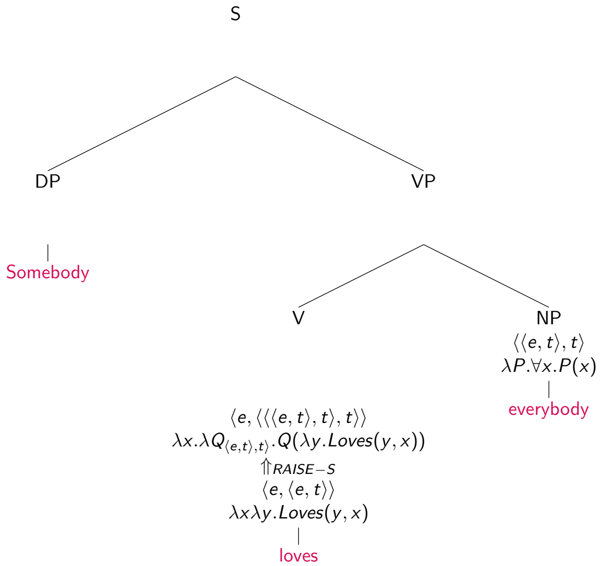
Type Shifting: RAISE-S



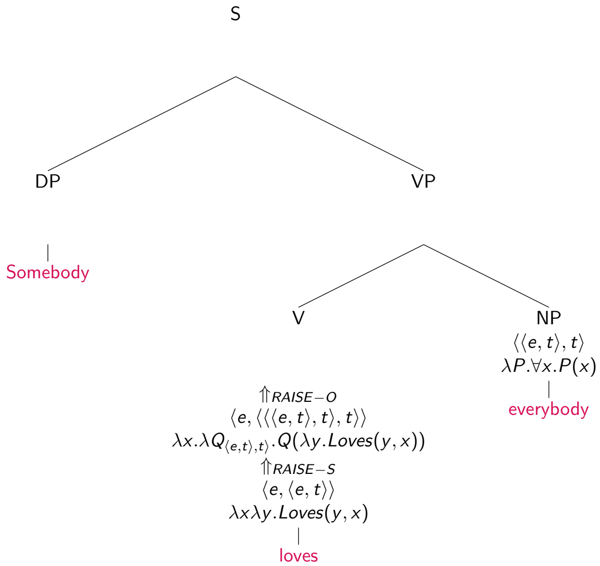
Type Shifting: RAISE-S



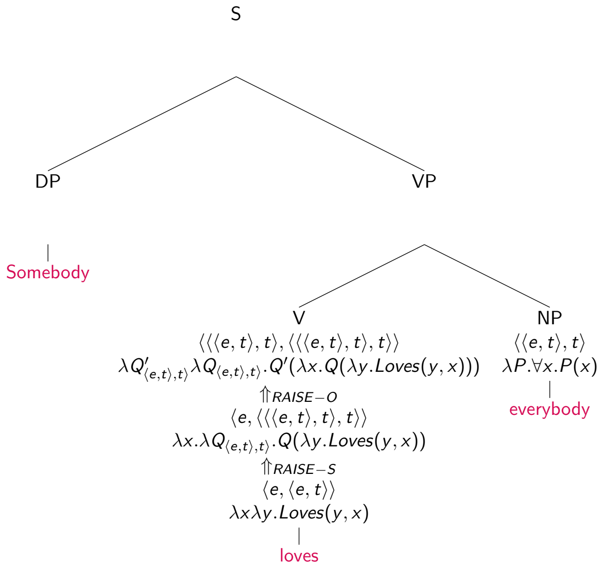
Type Shifting: RAISE-S



Type Shifting: RAISE-S



Type Shifting: RAISE-S



Type Shifting: RAISE-S

- S
  - DP
    - Somebody
  - VP
    $\langle\langle\langle e,t\rangle,t\rangle,t\rangle$
    $\lambda Q_{\langle\langle e,t\rangle,t\rangle}.\forall x.Q(\lambda y.Loves(y,x))$
    - V
      $\langle\langle\langle e,t\rangle,t\rangle,\langle\langle\langle e,t\rangle,t\rangle,t\rangle\rangle$
      $\lambda Q'_{\langle\langle e,t\rangle,t\rangle}\lambda Q_{\langle\langle e,t\rangle,t\rangle}.Q'(\lambda x.Q(\lambda y.Loves(y,x)))$
      $\Uparrow RAISE-O$
      $\langle e,\langle\langle\langle e,t\rangle,t\rangle,t\rangle\rangle$
      $\lambda x.\lambda Q_{\langle\langle e,t\rangle,t\rangle}.Q(\lambda y.Loves(y,x))$
      $\Uparrow RAISE-S$
      $\langle e,\langle e,t\rangle\rangle$
      $\lambda x\lambda y.Loves(y,x)$
      - loves
    - NP
      $\langle\langle e,t\rangle,t\rangle$
      $\lambda P.\forall x.P(x)$
      - everybody

Type Shifting: RAISE-S

S

DP
$\langle\langle e,t\rangle,t\rangle$
$\lambda P.\exists y.P(y)$
Somebody

VP
$\langle\langle\langle e,t\rangle,t\rangle,t\rangle$
$\lambda Q_{\langle e,t\rangle,t\rangle}.\forall x.Q(\lambda y.Loves(y,x))$

V
$\langle\langle\langle e,t\rangle,t\rangle,\langle\langle\langle e,t\rangle,t\rangle,t\rangle\rangle$
$\lambda Q'_{\langle e,t\rangle,t\rangle}\lambda Q_{\langle e,t\rangle,t\rangle}.Q'(\lambda x.Q(\lambda y.Loves(y,x)))$
$\Uparrow RAISE-O$
$\langle e,\langle\langle\langle e,t\rangle,t\rangle,t\rangle\rangle$
$\lambda x.\lambda Q_{\langle e,t\rangle,t\rangle}.Q(\lambda y.Loves(y,x))$
$\Uparrow RAISE-S$
$\langle e,\langle e,t\rangle\rangle$
$\lambda x\lambda y.Loves(y,x)$
loves

NP
$\langle\langle e,t\rangle,t\rangle$
$\lambda P.\forall x.P(x)$
everybody

Type Shifting: RAISE-S

S
t
$\forall x \exists y.Loves(y, x)$

DP
$\langle\langle e, t\rangle, t\rangle$
$\lambda P.\exists y.P(y)$
Somebody

VP
$\langle\langle\langle e, t\rangle, t\rangle, t\rangle$
$\lambda Q_{\langle e,t\rangle,t\rangle}.\forall x.Q(\lambda y.Loves(y, x))$

V
$\langle\langle\langle e, t\rangle, t\rangle, \langle\langle\langle e, t\rangle, t\rangle, t\rangle\rangle$
$\lambda Q'_{\langle e,t\rangle,t\rangle}\lambda Q_{\langle e,t\rangle,t\rangle}.Q'(\lambda x.Q(\lambda y.Loves(y, x)))$
$\Uparrow RAISE-O$
$\langle e, \langle\langle\langle e, t\rangle, t\rangle, t\rangle\rangle$
$\lambda x.\lambda Q_{\langle e,t\rangle,t\rangle}.Q(\lambda y.Loves(y, x))$
$\Uparrow RAISE-S$
$\langle e, \langle e, t\rangle\rangle$
$\lambda x \lambda y.Loves(y, x)$
loves

NP
$\langle\langle e, t\rangle, t\rangle$
$\lambda P.\forall x.P(x)$
everybody

Quizz for Today

TBA