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)
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 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

\[ \text{norwegian} \not\leftrightarrow \lambda x. \text{Norwegian}(x) \]
Adjectives: intersective

\[ \text{norwegian} \not\rightarrow \lambda x. \text{Norwegian}(x) \]

\[ \text{norwegian} \leadsto \lambda P \lambda x. [\text{Norwegian}(x) \land P(x)] \]

thus of \( \langle e, t \rangle, \langle e, t \rangle \) type: returns a new predicate that are both \textit{norwegians} and in the set of denoted by the input predicate \textit{millionaires}
Adjectives: intersective (cont.)

NP

A

NP

norwegian

N

millionaire
Adjectives: intersective (cont.)

NP

A NP

norwegian

N

\langle e, t \rangle

\lambda x. Millionaire(x)

millionaire
Adjectives: intersective (cont.)

\[
\begin{align*}
\text{norwegian} & \\
\langle e, t \rangle & \\
\lambda x. \text{Millionaire}(x) & \\
\text{millionaire} & 
\end{align*}
\]
Adjectives: intersective (cont.)

NP

\( \lambda P \lambda x. [Norwegian(x) \land P(x)] \)

\( \lambda x. Millionaire(x) \)

norwegian

millionaire
Adjectives: intersective (cont.)

\[ \lambda x. [\text{Norwegian}(x) \land \text{Millionaire}(x)] \]

A

\[ \langle \langle e, t \rangle, \langle e, t \rangle \rangle \]

\[ \lambda P \lambda x. [\text{Norwegian}(x) \land P(x)] \]

\[ \text{norwegian} \]

NP

\[ \langle e, t \rangle \]

\[ \lambda x. \text{Millionaire}(x) \]

NP

\[ \langle e, t \rangle \]

\[ \text{millionaire} \]
Adjectives: subsective

\[ \text{beautiful} \leadsto \forall P \forall x. \text{BeautifulAs}(P)(x) \rightarrow P(x) \]

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

beautiful  \(\sim\)  \(\forall P \forall x. \text{BeautifulAs}(P)(x) \rightarrow P(x)\)

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 $\leadsto \forall P \forall x. \text{BeautifulAs}(P)(x) \rightarrow P(x)$

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 \iff \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 $\iff \forall P \forall x. BeautifulAs(P)(x) \rightarrow P(x)$

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 \iff \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.
\therefore Nuriev is beautiful.
Adjectives: subsective (cont.)

NP

A

NP

| beautiful

| N

| dancer
Adjectives: subsective (cont.)

\[ \forall x. \left[ \text{BeautifulAs}(Dancer)(x) \implies Dancer(x) \right] \]

\[ A \langle \langle e, t \rangle, \langle e, t \rangle \rangle \lambda x. \text{BeautifulAs}(P)(x) \implies P(x) \]

beautiful

Dancer(x)

dancer
Adjectives: subsective (cont.)

\[
\lambda x. \text{Dancer}(x) \rightarrow \text{Dancer}(x)
\]

\[
\lambda x. \text{BeautifulAs}(\langle e, t \rangle) \rightarrow \text{BeautifulAs}(\langle e, t \rangle)
\]

\[
\lambda x. \text{Dancer}(x) \rightarrow \text{Dancer}(x)
\]

\[
\lambda x. \text{Dancer}(x) \rightarrow \text{Dancer}(x)
\]

\[
\lambda x. \text{Dancer}(x) \rightarrow \text{Dancer}(x)
\]

\[
\lambda x. \text{Dancer}(x) \rightarrow \text{Dancer}(x)
\]

\[
\lambda x. \text{Dancer}(x) \rightarrow \text{Dancer}(x)
\]

\[
\lambda x. \text{Dancer}(x) \rightarrow \text{Dancer}(x)
\]

\[
\lambda x. \text{Dancer}(x) \rightarrow \text{Dancer}(x)
\]

\[
\lambda x. \text{Dancer}(x) \rightarrow \text{Dancer}(x)
\]

\[
\lambda x. \text{Dancer}(x) \rightarrow \text{Dancer}(x)
\]

\[
\lambda x. \text{Dancer}(x) \rightarrow \text{Dancer}(x)
\]

\[
\lambda x. \text{Dancer}(x) \rightarrow \text{Dancer}(x)
\]

\[
\lambda x. \text{Dancer}(x) \rightarrow \text{Dancer}(x)
\]

\[
\lambda x. \text{Dancer}(x) \rightarrow \text{Dancer}(x)
\]

\[
\lambda x. \text{Dancer}(x) \rightarrow \text{Dancer}(x)
\]

\[
\lambda x. \text{Dancer}(x) \rightarrow \text{Dancer}(x)
\]

\[
\lambda x. \text{Dancer}(x) \rightarrow \text{Dancer}(x)
\]

\[
\lambda x. \text{Dancer}(x) \rightarrow \text{Dancer}(x)
\]

\[
\lambda x. \text{Dancer}(x) \rightarrow \text{Dancer}(x)
\]
Adjectives: subsective (cont.)

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

\[
\lambda x. Dancer(x)
\]

\[
\lambda x. Dancer(x)
\]
Adjectives: subsective (cont.)

\[ \lambda x. \forall x. [\text{BeautifulAs}(\text{Dancer})(x) \rightarrow \text{Dancer}(x)] \]

\[
\lambda P \lambda x. [\text{BeautifulAs}(P)(x) \rightarrow P(x)]
\]

\[
\lambda x. \text{Dancer}(x)
\]

\[
\lambda x. \text{Dancer}(x)
\]

\[
\text{beautiful}
\]

\[
\text{dancer}
\]
Adjectives: predicative position

Example

Frida is Norwegian.
This is reasonable.
Hair is curly.
Adjectives: predicative position (cont.)

Norwegian 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.
Adjectives: predicative position (cont.)

\[
S \\
\text{DP} \quad \text{VP} \\
\text{e} \\
\text{Frida} \\
\text{V} \quad \text{AP} \\
\langle\langle\text{e}, \text{t}\rangle, \langle\text{e}, \text{t}\rangle\rangle \\
\lambda P \cdot \lambda x. [\text{Norwegian}(x) \land P(x)] \\
\text{is} \\
\text{Norwegian}
\]
Adjectives: predicative position (cont.)

Norwegian 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.

Introduction to Formal Semantics, Summer 2022

{o.petukhova; nndascalu}@lsv.uni-saarland.de
Adjectives: predicative position (cont.)

Norwegian (x) \land P(x)

\lambda P. \lambda x. [Norwegian(x) \land P(x)]

\lambda P. P

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

\langle \langle e, t \rangle, \langle e, t \rangle \rangle
Adjectives: predicative position (cont.)

A MODIFIER type analysis as above causes the problem.
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 $\gamma$ be a syntax tree whose sub-trees are $\alpha$ and $\beta$ where:

- $\alpha \leadsto \alpha'$ where $\alpha'$ has type $\langle \sigma, \tau \rangle$
- $\beta \leadsto \beta'$ where $\beta'$ has type $\langle \sigma \rangle$

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

\[
S \\
PN \quad VP \\
\text{Lisa} \\
V \quad PN \\
\text{loves} \quad \text{Mark}
\]
Compositional Rule 1 (recap)

**Composition Rule 1: Function Application**

Let $\gamma$ be a syntax tree whose sub-trees are $\alpha$ and $\beta$ where:
- $\alpha \mapsto \alpha'$ where $\alpha'$ has type $\langle \sigma, \tau \rangle$
- $\beta \mapsto \beta'$ where $\beta'$ has type $\langle \sigma \rangle$

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

---

Diagram:

```
S
  PN  VP
  Lisa
    V   PN
    e   e
    \lambda y. \lambda x. Loves(x, y)   ma
    loves                              Mark
```
Compositional Rule 1 (recap)

Composition Rule 1: Function Application

Let $\gamma$ be a syntax tree whose sub-trees are $\alpha$ and $\beta$ where:
- $\alpha \rightsquigarrow \alpha'$ where $\alpha'$ has type $\langle \sigma, \tau \rangle$
- $\beta \rightsquigarrow \beta'$ where $\beta'$ has type $\langle \sigma \rangle$

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

\[
S \\
\text{PN} \quad \text{VP} \\
\langle e, t \rangle \\
\lambda x.\text{Loves}(x, ma) \\
\text{Lisa} \\
\text{V} \quad \text{PN} \\
\langle e, \langle e, t \rangle \rangle \\
\lambda y.\lambda x.\text{Loves}(x, y) \\
\text{loves} \quad \text{Mark}
\]
Compositional Rule 1 (recap)

**Composition Rule 1: Function Application**

Let \( \gamma \) be a syntax tree whose sub-trees are \( \alpha \) and \( \beta \) where:

- \( \alpha \leadsto \alpha' \) where \( \alpha' \) has type \( \langle \sigma, \tau \rangle \)
- \( \beta \leadsto \beta' \) where \( \beta' \) has type \( \langle \sigma \rangle \)

then

\[
\gamma \leadsto \alpha'(\beta')
\]

\[
S
\]

**Diagram**

```
  S
    /
   /  \n PN  VP
  /  \\
 e   \langle e, t \rangle
 /    \\
li    \lambda x. Loves(x, ma)
  \\
Lisa
    /
   /  \n V   PN
  /  \\
\langle e, \langle e, t \rangle \rangle   e
 /    \\
\lambda y. \lambda x. Loves(x, y) ma
      /
      loves Mark
```
Compositional Rule 1 (recap)

Composition Rule 1: Function Application

Let $\gamma$ be a syntax tree whose sub-trees are $\alpha$ and $\beta$ where:
- $\alpha \rightsquigarrow \alpha'$ where $\alpha'$ has type $\langle \sigma, \tau \rangle$
- $\beta \rightsquigarrow \beta'$ where $\beta'$ has type $\langle \sigma \rangle$

then $\gamma \rightsquigarrow \alpha'(\beta')$
Composition Rule 2 (recap)

Composition Rule 2: Non-branching Nodes

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

$$S$$

$$\begin{array}{c}
PN \\
\mid \\
VP
\end{array}$$

Lisa

$$\begin{array}{c}
V
\end{array}$$

laughs
Composition Rule 2 (recap)

Composition Rule 2: Non-branching Nodes

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

$S$

$PN \quad VP$

Lisa $\mid V$

$\langle e, t \rangle$

$\lambda x. Laughs(x)$

laughs
Composition Rule 2 (recap)

**Composition Rule 2: Non-branching Nodes**

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

\[
S \\
PN \quad VP \\
\langle e, t \rangle \\
\lambda x. \text{Laughs}(x) \\
\text{Lisa} \\
\quad \text{V} \\
\langle e, t \rangle \\
\lambda x. \text{Laughs}(x) \\
\text{laughs}
\]
Composition Rule 2: Non-branching Nodes

If $\beta$ is a tree whose only daughter is $\alpha$, where $\alpha \leadsto \alpha'$ then $\beta \leadsto \alpha'$

$$
S
$$

$$
\begin{align*}
\text{PN} & \quad \text{VP} \\
\text{Lisa} & \quad \lambda x. \text{Laughs}(x) \\
\text{li} & \quad \langle e, t \rangle \\
e & \quad \langle e, t \rangle \\
\text{laughs} &
\end{align*}
$$
Composition Rule 2 (recap)

Composition Rule 2: Non-branching Nodes

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

```
S
 t
 Laughs(li)

PN       VP
 e    ⟨e, t⟩
 li  λx.Laughs(x)
Lisa       V
          ⟨e, t⟩
          λx.Laughs(x)
          laughs
```
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

(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)
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
da TYPE-SHIFTING RULE (invisible to the syntactic component of the grammar) or
da SILENT OPERATOR (a reflection in the syntax)
Adjectives: Silent Operator

\[
\begin{align*}
\lambda P. \lambda x. [Reasonable(x) \land P(x)]
\end{align*}
\]
Adjectives: Type Shifting

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

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

\[
\begin{align*}
A & \quad \langle\langle e, t\rangle, \langle e, t\rangle\rangle \\
& \quad \lambda P.\lambda x.[Reasonable(x) \land P(x)] \\
& \quad \uparrow_{MOD} \\
& \quad A \\
& \quad \langle e, t\rangle \\
& \quad \lambda x.\text{Reasonable}(x) \\
& \quad | \\
& \quad \text{reasonable}
\end{align*}
\]
(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:

- $\gamma$ is a tree whose only two subtrees are $\alpha$ and $\beta$
- $\alpha \leadsto \alpha'$
- $\beta \leadsto \beta'$
- $\alpha'$ and $\beta'$ are of type $\langle e, t \rangle$

Then:

$\gamma \leadsto \lambda u. ([\alpha'(u)] \land \beta'(u))$, where $u$ is a variable of type $e$ that does not occur free in $\alpha'$ or $\beta'$.
(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:

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

Then:

$$\gamma \rightsquigarrow \lambda u. \left[ \alpha'(u) \land \beta'(u) \right]$$

where $u$ is a variable of type $e$ that does not occur free in $\alpha'$ or $\beta'$. 
Predicate Modification (example)

\[
\lambda x. [\text{Reasonable}(x) \land \text{Doubt}(x)]
\]

\[
\langle e, t \rangle
\]

\[
\text{AP} \quad \text{NP}
\]

\[
\lambda x. \text{Reasonable}(x) \quad \lambda x. \text{Doubt}(x)
\]

\[
\text{reasonable} \quad \text{doubt}
\]
Relative Clauses

Examples

reasonable doubt ≈ doubt which is reasonable
the broken cup ≈ the cup which is broken

Apply Predicate Modification and use $\iota$ operator for definite expression
Relative Clauses (example)

```
DP
  e

D'
  |  NP
  D

the

N'
  |  RC
  N

which is broken

cup
```
Relative Clauses (example)

\[
\begin{array}{c}
\text{DP} \\
e \\
\text{D'} \\
\text{D} \\
\text{NP} \\
\text{N'} \\
\text{the} \\
\text{N'} \\
\text{N} \\
\text{RC} \\
\langle e, t \rangle \\
\lambda x. \text{Broken}(x) \\
\text{which is broken} \\
\text{cup}
\end{array}
\]
Relative Clauses (example)

\[
\begin{array}{c}
\textbf{DP} \\
e \\
\textbf{D}' \\
\textbf{D} \\
\textbf{NP} \\
\textbf{N'} \\
\text{the} \\
\textbf{N'} \\
\langle e, t \rangle \\
\textbf{N} \\
\langle e, t \rangle \\
\lambda x. \text{Cup}(x) \\
\text{cup} \\
\langle e, t \rangle \\
\lambda x. \text{Broken}(x) \\
\text{which is broken}
\end{array}
\]
Relative Clauses (example)

\[
\begin{array}{c}
\text{DP} \\
e \\
\hline
\text{D'} \\
\text{D} \\
\text{NP} \\
\langle e, t \rangle \\
\lambda x. [\text{Cup}(x) \land \text{Broken}(x)] \\
\text{N'} \\
\text{the} \\
\text{N'} \\
\text{N} \\
\langle e, t \rangle \\
\lambda x. \text{Cup}(x) \\
\text{RC} \\
\langle e, t \rangle \\
\lambda x. \text{Broken}(x) \\
\text{which is broken}
\end{array}
\]
Relative Clauses (example)

\[
\begin{align*}
\text{DP} & \quad e \\
\text{D}' & \quad \langle \langle e, t \rangle, e \rangle \\
\text{D} & \quad \lambda x. [\text{Cup}(x) \land \text{Broken}(x)] \\
\text{NP} & \quad \langle e, t \rangle \\
\text{N}' & \quad \text{the} \\
\text{N}' & \quad \text{RC} \\
\text{N} & \quad \langle e, t \rangle \\
\text{N} & \quad \lambda x. \text{Broken}(x) \\
\text{N} & \quad \langle e, t \rangle \\
\text{N} & \quad \lambda x. \text{Cup}(x) \\
\text{N} & \quad \langle e, t \rangle \\
\text{N} & \quad \text{cup} \\
\text{NP} & \quad \langle e, t \rangle \\
\text{N} & \quad \langle e, t \rangle \\
\text{NP} & \quad \langle e, t \rangle \\
\text{DP} & \quad e
\end{align*}
\]
Relative Clauses (example)

\[ e \in x. [\text{Cup}(x) \land \text{Broken}(x)] \]

\[ \langle e, t \rangle \lambda x. [\text{Cup}(x) \land \text{Broken}(x)] \]

\[ \langle e, t \rangle \lambda P. [\lambda x. P(x)] \]

\[ \text{the} \]

\[ \langle e, t \rangle \lambda x. \text{Cup}(x) \]

\[ \text{cup} \]

\[ \langle e, t \rangle \lambda x. \text{Broken}(x) \]

\[ \text{which is broken} \]
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 \[ \text{XP} (\text{specifier}) \text{X}' (\text{complement}) \].
Relative Clauses: TRACE

TRACE or in contemporary theories of syntax often use the term UNPRONOUNCED COPY CP, which 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 \( [x_P(specifier)] [x'] _x (complement) ] \).
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 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: $\llbracket v_9 \rrbracket^{M,g} = g(v_9)$
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: $\llbracket v_9 \rrbracket^{M,g} = g(v_9)$

Composition Rule 4: Pronouns and Trace Rule

If $\alpha$ is an indexed trace or pronoun, $\alpha_i \rightsquigarrow v_i$
Lisa loves $t_1$

```
S
  /\  \\/
DP  VP
  /\
Lisa
  /\    \\/
V    DP
  /\
  loves  $t_1$
```
Pronouns

Lisa loves $t_1$
Pronouns

Lisa loves $t_1$

\[
S \\
\quad DP \quad VP
\]

\[
S \\
\quad \text{Lisa} \\
\quad \text{V} \quad \text{DP}
\]

\[
\langle e, \langle e, t \rangle \rangle \quad e \\
\lambda y. \lambda x. \text{Loves}(x, y) \quad \nu_1 \\
\text{loves} \quad t_1
\]
Lisa loves $t_1$

```
S

DP         VP

  ⟨e, t⟩

  λx. Loves(x, v₁)

  Lisa

V         DP

  ⟨e, ⟨e, t⟩⟩

  λy. λx. Loves(x, y)

  loves

  t₁
```
Lisa loves $t_1$

\[
S \\
\text{DP} \quad \text{VP} \\
e \quad \langle e, t \rangle \\
\text{li} \quad \lambda x.\text{Loves}(x, v_1) \\
\text{Lisa} \\
\text{V} \quad \text{DP} \\
\langle e, \langle e, t \rangle \rangle \quad e \\
\lambda y.\lambda x.\text{Loves}(x, y) \quad v_1 \\
\text{loves} \quad t_1
\]
Pronouns

Lisa loves $t_1$

```
S
  t
  Loves(li, v₁)
  VP
    ⟨e, t⟩
    λx. Loves(x, v₁)
  DP
    e
    li
    Lisa
  V
    ⟨e, ⟨e, t⟩⟩
    λy. λx. Loves(x, y)
    v₁
    loves
    t₁
```
Relative Clauses: Predication Abstraction

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

**Composition Rule 5: Predicate Abstraction**

If:

- \( \gamma \) is a tree whose only two subtrees are \( \alpha_i \) and \( \beta \)
- \( \beta \overset{\sim}{\rightarrow} \beta' \)
- \( \beta' \) is an expression of type \( t \)

Then \( \gamma \overset{\sim}{\rightarrow} \lambda v_i. \beta' \)
Relative Clauses: Predication Abstraction (cont.)

\[
\lambda v_1. \text{Loves}(\text{ma}, v_1)
\]

\[
\langle t, t \rangle \lambda p. p
\]

\[
\text{Loves}(\text{bj}, v_1)
\]

\[
\text{Loves}(x, y)
\]

\[
\text{loves}
\]

\[
\text{Mark}
\]

\[
\text{VP}
\]

\[
\text{DP}
\]

\[
\text{CP}
\]

\[
\langle e, t \rangle \lambda v_1.
\]

\[
\langle e, \langle e, t \rangle \rangle \lambda y. \lambda x. \text{Loves}(x, y)
\]

\[
\text{who}_1\ C'
\]

\[
\text{C'}
\]

\[
\text{C}
\]

\[
\text{S}
\]

\[
\text{that}
\]

\[
\text{DP}
\]

\[
\text{VP}
\]

\[
\text{Introduction to Formal Semantics, Summer 2022}
\]

\[
\text{o.petukhova; nndascalu}@lsv.uni-saarland.de
\]
Relative Clauses: Predication Abstraction (cont.)

\[ \langle e, \langle e, t \rangle \rangle \lambda v_1. \text{Loves}(x, y) \]

\[ \langle e, \langle e, t \rangle \rangle \lambda y. \lambda x. \text{Loves}(x, y) \]

\[ \langle e, \langle e, t \rangle \rangle \lambda x. \text{loves} \]

\[ \text{Mark} \]

\[ \text{CP} \]

\[ \text{who}_1 \]

\[ \text{C} \]

\[ \text{S} \]

\[ \text{that} \]

\[ \text{DP} \]

\[ \text{VP} \]

\[ \text{loves} \]

\[ t_1 \]
Relative Clauses: Predication Abstraction (cont.)

\[
\begin{align*}
\text{CP} \\
\text{C} & \quad \text{S} \\
\langle \text{who}_1, \text{C'} \rangle & \\
\lambda v_1. \text{Loves}(\text{ma}, v_1) & \\
\text{C'} & \\
\langle \text{t}, \text{t} \rangle & \\
\lambda p. \text{p} & \\
\text{that} & \\
\text{S} & \\
\text{Loves}(\text{bj}, v_1) & \\
\text{DP} & \\
\text{Mark} & \\
\langle e, \langle e, t \rangle \rangle & \\
\lambda x. \text{Loves}(x, v_1) & \\
\text{V} & \\
\langle e, \langle e, t \rangle \rangle & \\
\text{loves} & \\
\text{DP} & \\
\text{e} & \\
\lambda y. \lambda x. \text{Loves}(x, y) & \\
v_1 & \\
\text{t}_1 & \\
\end{align*}
\]
Relative Clauses: Predication Abstraction (cont.)

\[
\langle e, \langle e, t \rangle \rangle \lambda y. \lambda x. Loves(x, y)
\]

\[
\lambda x. Loves(x, v_1)
\]

\[
\text{that}
\]

\[
\langle e, \langle e, t \rangle \rangle \lambda x. Loves(x, v_1)
\]

\[
\langle e, \langle e, t \rangle \rangle \lambda x. Loves(x, v_1)
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\langle e, \langle e, t \rangle \rangle \lambda x. Loves(x, v_1)
\]

\[
\langle e, \langle e, t \rangle \rangle \lambda x. Loves(x, v_1)
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]

\[
\text{that}
\]
Relative Clauses: Predication Abstraction (cont.)
Relative Clauses: Predication Abstraction (cont.)

\[ \langle t, t \rangle \lambda p. p \]  
\[ \text{that} \]  
\[ \text{Mark} \]  
\[ \langle e, \langle e, t \rangle \rangle \lambda y. \lambda x. \text{Loves}(x, y) \ni \text{loves} t_1 \]
Relative Clauses: Predication Abstraction (cont.)

\[
\begin{array}{c}
\text{CP} \\
/_{\text{who}_1}/ \text{C'} \\
\text{C} \quad \text{t} \\
\text{Loves}(bj, v_1) \\
\text{C} \quad \text{S} \\
\langle t, t \rangle \quad \text{t} \\
\lambda p. p \\
/_{\text{that}}/ \\
\text{DP} \quad \text{VP} \\
\text{e} \quad \langle e, t \rangle \\
\text{ma} \quad \lambda x. \text{Loves}(x, v_1) \\
\text{Mark} \\
\langle e, \langle e, t \rangle \rangle \quad \text{e} \\
\lambda y. \lambda x. \text{Loves}(x, y) \\
\text{loves} \quad \langle e, \langle e, t \rangle \rangle \\
\text{DP} \quad \text{t}_1 \quad v_1 \\
\end{array}
\]
Relative Clauses: Predication Abstraction (cont.)

\[
\begin{align*}
CP & \\
\langle e, t \rangle & \\
\lambda v_1. Loves(ma, v_1) & \\
\cdot & \\
\lambda p. p & \\
that & \\
\text{DP} & \\
\langle e, \langle e, t \rangle \rangle & \\
\lambda y. \lambda x. Loves(x, y) & \\
\cdot & \\
\text{loves} & \\
\text{DP} & \\
\langle e, t_1 \rangle & \\
\end{align*}
\]
Relative Clauses: Predication Abstraction (cont.)
Relative Clauses: Predication Abstraction (cont.)

\[ \lambda x. \text{Reasonable}(x) \]

\{ o.petukhova; nnrdascalu \}@lsv.uni-saarland.de

Introduction to Formal Semantics, Summer 2022
Relative Clauses: Predication Abstraction (cont.)

```
NP
  N  CP
   |   |
  doubt
 |
which₁ C'
 C  S

|that|
DP  VP
  |
  V  AP
  |
  λ₁{(e,t),(e,t)} λx.Reasonable(x)
  |
  is  reasonable
```
Relative Clauses: Predication Abstraction (cont.)

NP

N

CP

doubt

which

C'

C

S

DP

that

VP

\langle e, t \rangle

\lambda x. Reasonable(x)

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

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\langle e, t \rangle

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\langle e, t \rangle

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\langle e, t \rangle

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\langle e, t \rangle

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\langle e, t \rangle

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\langle e, t \rangle

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\langle e, t \rangle

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\langle e, t \rangle

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\langle e, t \rangle

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\langle e, t \rangle

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\langle e, t \rangle

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\langle e, t \rangle

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\langle e, t \rangle

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\langle e, t \rangle

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\langle e, t \rangle

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\langle e, t \rangle

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\langle e, t \rangle

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\langle e, t \rangle

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\langle e, t \rangle

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\langle e, t \rangle

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\langle e, t \rangle

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\langle e, t \rangle

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)

\langle e, t \rangle

\lambda P.P

\langle e, t \rangle

\lambda x. Reasonable(x)
Relative Clauses: Predication Abstraction (cont.)

NP

\[ \lambda x. \text{Reasonable}(x) \]

C

DP

\[ \langle e, t \rangle \]

VP

\[ \lambda x. \text{Reasonable}(x) \]

AP

\[ \lambda P. P \]

\[ \text{is} \]

\[ \text{reasonable} \]
Relative Clauses: Predication Abstraction (cont.)

\[
\begin{align*}
\text{NP} & \quad \text{\(\langle e, t \rangle\)} \\
\text{NP} & \quad \lambda x. \text{Reasonable\( (x) \)} \\
\text{CP} & \quad \text{\(\langle e, t \rangle\)} \\
\text{DP} & \quad \text{\(\langle e, t \rangle\)} \\
\text{VP} & \quad \text{\(\langle e, t \rangle\)} \\
\text{AP} & \quad \text{\(\langle e, t \rangle\)} \\
\text{S} & \quad \text{\(\langle e, t \rangle\)} \\
\text{C} & \quad \text{\(\langle e, t \rangle\)} \\
\text{C'} & \quad \text{\(\langle e, t \rangle\)} \\
\end{align*}
\]

Introduction to Formal Semantics, Summer 2022
Relative Clauses: Predication Abstraction (cont.)

\[
\begin{array}{c}
\text{NP} \\
\text{N [doubt]} \\
\text{CP [\(\lambda x. [\text{Doubt}(x) \land \text{Reasonable}(x)]\)]} \\
\end{array}
\]

\[
\begin{array}{c}
\text{C} \\
\text{S} \\
\langle t, t \rangle \\
\lambda p.p \\
| \\
\text{that} \\
\end{array}
\rightarrow
\begin{array}{c}
\text{DP} \\
\text{e} \\
\text{v}_1 \\
| \\
\text{C'} \\
\langle e, t \rangle \\
\lambda x. \text{Reasonable}(x) \\
| \\
\text{VP} \\
\langle e, t \rangle \\
\lambda x. \text{Reasonable}(x) \\
| \\
\text{AP} \\
\langle \langle e, t \rangle, \langle e, t \rangle \rangle \\
\lambda P.P \\
| \\
\text{V} \\
\langle e, t \rangle \\
\text{reasonable} \\
\end{array}
\]
Relative Clauses: Predication Abstraction (cont.)

NP

N          CP

| doubt

which

Reasonable(v₁)

C          S

{⟨t, t⟩, ⟨e, t⟩}  \lambda p.p

reason

that

Reasonable(v₁)

doubt

which₁

W

that

Reasonable(v₁)

C

t

\lambda x. Reasonable(x)

\lambda P.P

\lambda x. Reasonable(x)

is

reasonable

{0.petukhova; nndascalu}@lsv.uni-saarland.de

Introduction to Formal Semantics, Summer 2022
Relative Clauses: Predication Abstraction (cont.)

NP

N

C

CP

⟨e, t⟩

λν₁. Reasonable(ν₁)

which

C

t

Reasonable(ν₁)

C

S

⟨t, t⟩

λp.p

that

that

Reasonable(ν₁)

DP

e

v₁

⟨e, t⟩

VP

λx. Reasonable(x)

DP

e

V

AP

⟨⟨e, t⟩, ⟨e, t⟩⟩

λP.P

⟨e, t⟩

λx. Reasonable(x)

is

reasonable
Relative Clauses: Predication Abstraction (cont.)
Relative Clauses: Predication Abstraction (cont.)

\[\lambda x. [Doubt(x) \land Reasonable(x)]\]

\[\lambda v_1. Reasonable(v_1)\]

\[\lambda v_1. Reasonable(v_1)\]

\[\lambda p. p\]

\[\lambda x. Reasonable(x)\]

\[\lambda P.P\]

\[\lambda x. Reasonable(x)\]

\[\text{is}\]

\[\text{reasonable}\]
Relative Clauses: complex example

The man who talked to the boy who visited him

(cont.)
Quantification: Object Position

\[ \forall x. Loves(m, x) \]

\[ e \in \langle e, \langle e, t \rangle \rangle \]

\[ \lambda P. \forall x. P(x) \]

\[ \langle e, t \rangle, t \]

\[ \lambda x. \lambda y. Loves(y, x) \]

\[ \lambda x. \lambda y. Loves(y, x) \]

\[ \lambda P. \forall x. P(x) \]

\[ \langle e, \langle e, t \rangle \rangle \]

\[ \lambda P. \forall x. P(x) \]

\[ \langle e, t \rangle, t \]

\[ \forall x. Loves(m, x) \]

\[ \langle e, \langle e, t \rangle \rangle \]

\[ \forall x. Loves(m, x) \]

\[ \forall x. Loves(m, x) \]

\[ \forall x. Loves(m, x) \]
Quantifier Raising

Right Translation

everybody \leadsto \lambda P \forall x. P(x)
loves \leadsto \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)

\[ \text{DS} \quad \begin{array}{c} \text{SuSt} \\ \hline \text{LF} \quad \text{PF} \end{array} \]
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.
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.
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 \( \alpha \) is translated into a logical expression \( \alpha' \) of type \( \langle e, \langle \alpha, t \rangle \rangle \) for any type \( \alpha \), then \( \alpha \) also has a translation of type \( \langle \langle \langle e, t \rangle, t \rangle, \langle \alpha, t \rangle \rangle \) of the following form:

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

(unless \( Q, y \) or \( x \) occurs in \( \alpha' \); in that case, use different variables).
Type Shifting: RAISE-O

S

<table>
<thead>
<tr>
<th>DP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VP</th>
</tr>
</thead>
<tbody>
<tr>
<td>everybody</td>
</tr>
</tbody>
</table>

| loves       |

| NP          |

| Mary loves everybody |

DP: Mary
VP: everybody, loves
Type Shifting: RAISE-O

\[
S \\
/ \ \\
DP \quad VP \\
\mid \\
Mary \\
/ \\
V \quad NP \\
\mid \\
\langle \langle e, t \rangle, t \rangle \\
\lambda P. \forall x. P(x) \\
\mid \\
everybody \\
/ \\
lives 
\]
Type Shifting: RAISE-O

\[ S \]

\[ \text{DP} \quad \text{VP} \]

| Mary |

\[ V \]

\[ \langle \langle e, t \rangle, t \rangle \quad \lambda P. \forall x. P(x) \]

\[ \langle e, \langle e, t \rangle \rangle \quad \lambda x. \lambda y. \text{Loves}(y, x) \]

\[ \text{loves} \]

\[ \langle \langle e, t \rangle, t \rangle, \langle e, t \rangle \rangle \]

\[ \lambda x. \lambda y. \text{Loves}(y, x) \]

\[ \text{everybody} \]
Type Shifting: RAISE-O

\[
S \\
\downarrow \\
\text{DP} \quad \text{VP} \\
\text{Mary} \\
\downarrow \\
\text{V} \quad \text{NP} \\
\langle\langle e, t \rangle, t \rangle \\
\lambda P. \forall x. P(x) \\
\downarrow \\
\text{everybody} \\
\downarrow \\
\uparrow^{\text{RAISE-O}} \\
\langle e, \langle e, t \rangle \rangle \\
\lambda x. \lambda y. \text{Loves}(y, x) \\
\downarrow \\
\text{loves}
Type Shifting: RAISE-O

S

 DP  VP

Mary

V

\[ \langle \langle e, t \rangle, t \rangle, \langle e, t \rangle \rangle \]
\[ \lambda Q_{\langle e, t \rangle, t} \lambda y. Q(\lambda x. Loves(y, x)) \]
\[ \uparrow^{RAISE-O} \]
\[ \langle e, \langle e, t \rangle \rangle \]
\[ \lambda x. \lambda y. Loves(y, x) \]

NP

\[ \langle e, t \rangle, t \rangle \]
\[ \lambda P. \forall x. P(x) \]

 everybody

loves
Type Shifting: RAISE-O

\[
S \\
\text{DP} \quad \text{VP} \\
\langle e, t \rangle \\
\lambda y. \forall x. \text{Loves}(y, x) \\
\langle \langle \langle e, t \rangle, t \rangle, \langle e, t \rangle \rangle \\
\lambda Q_{\langle e, t \rangle, t} \lambda y. Q(\lambda x. \text{Loves}(y, x)) \\
\uparrow \text{RAISE-O} \\
\langle e, \langle e, t \rangle \rangle \\
\lambda x. \lambda y. \text{Loves}(y, x) \\
\langle \langle e, t \rangle, t \rangle \rangle \\
\lambda P. \forall x. P(x) \\
\text{everybody} \\
\lambda P_{\langle e, t \rangle, t} \lambda y. P(y) \\
\text{loves} \\
\text{Mary}
\]
Type Shifting: RAISE-O

\[
\begin{aligned}
S & \rightarrow \text{DP} \rightarrow \text{VP} \rightarrow \text{NP} \\
\text{DP} & \rightarrow m \\
\text{VP} & \rightarrow \langle e, t \rangle \\
\text{NP} & \rightarrow \langle \langle e, t \rangle, t \rangle \\
& \quad \uparrow \text{RAISE-O} \\
& \quad \langle e, \langle e, t \rangle \rangle \\
& \quad \lambda x. \lambda y. \text{Loves}(y, x) \\
\text{everybody} & \rightarrow \langle e, t \rangle \\
& \quad \lambda P. \forall x. P(x) \\
\end{aligned}
\]
Type Shifting: RAISE-O

\[
S
\]
\[
t
\]
\[
\forall x. Loves(m, x)
\]

\[\lambda y. \forall x. Loves(y, x)\]

\[
V
\]
\[
\langle\langle\langle e, t\rangle, t\rangle, t\rangle
\]
\[
\lambda Q_{\langle e, t\rangle, t} \lambda y. Q(\lambda x. Loves(y, x))
\]
\[\uparrow_{\text{RAISE-O}}\]
\[
\langle e, \langle e, t\rangle\rangle
\]
\[
\lambda x. \lambda y. Loves(y, x)
\]
\[
\mid\text{loves}
\]

\[\text{DP}\]
\[
m\]
\[
e\]
\[
\mid\text{Mary}
\]

\[\text{VP}\]
\[
\langle e, t\rangle
\]

\[\text{VP}\]
\[
\langle e, t\rangle
\]
\[
\lambda y. \forall x. Loves(y, x)
\]
\[\mid\text{loves}\]

\[\text{NP}\]
\[
\langle\langle\langle e, t\rangle, t\rangle, t\rangle
\]
\[
\lambda P. \forall x. P(x)
\]
\[\mid\text{everybody}\]
Type Shifting: Two Quantifiers

\[ \exists y \forall x. \text{Loves}(y, x) \]

\[ \langle\langle e, t \rangle, t \rangle \lambda P. \exists y. P(y) \]

\[ \langle\langle e, t \rangle, \langle e, t \rangle \rangle \lambda Q \langle\langle e, t \rangle, t \rangle \lambda y. Q(\lambda x. \text{Loves}(y, x)) \]

\[ \langle\langle e, t \rangle, \langle e, t \rangle \rangle \lambda x. \lambda y. \text{Loves}(y, x) \]

\[ \text{Somebody} \]

\[ \text{everybody} \]

\[ \text{loves} \]
Type Shifting: Two Quantifiers

\[ \exists y. \forall x. \text{Loves}(y, x) \]

\[ \langle\langle e, t\rangle, t\rangle, t \rangle \lambda P. \forall x. P(x) \]

\[ \lambda P. \forall x. P(x) \]

\[ \langle\langle e, t\rangle, t\rangle \rangle \lambda Q\langle\langle e, t\rangle, t\rangle, t \rangle \lambda y. Q(\lambda x. \text{Loves}(y, x)) \]

\[ \uparrow \text{RAISE} \]

\[ \lambda x. \lambda y. \text{Loves}(y, x) \]

\[ \text{loves} \]

\[ \langle\langle e, t \rangle, t \rangle, t \rangle \lambda P. \forall x. P(x) \]

\[ \text{everybody} \]

\[ \langle\langle e, t \rangle, t \rangle, t \rangle \lambda Q\langle\langle e, t \rangle, t \rangle, t \rangle \lambda y. Q(\lambda x. \text{Loves}(y, x)) \]

\[ \text{Somebody} \]
Type Shifting: Two Quantifiers

\[
S \\
\downarrow \\
\begin{array}{c}
\text{DP} \\
\| \\
\text{Somebody}
\end{array} \\
\downarrow \\
\begin{array}{c}
\text{VP} \\
\downarrow \\
\langle\langle e, t \rangle, t \rangle \\
\lambda P. \forall x. P(x)
\end{array} \\
\downarrow \\
\langle\langle e, (e, t) \rangle \rangle \\
\lambda x. \lambda y. Loves(y, x)
\]

\[
\langle\langle e, t \rangle, t \rangle \\
\lambda P. \forall x. P(x)
\]

\[
\langle\langle e, (e, t) \rangle \rangle \\
\lambda x. \lambda y. Loves(y, x)
\]

\[
\text{loves}
\]

\[
\text{everybody}
\]
Type Shifting: Two Quantifiers

S

DP  VP

| Somebody

V  NP

\[ \uparrow^{RAISE-o} \]

\[ \langle e, \langle e, t \rangle \rangle \]

\[ \lambda x. \lambda y. Loves(y, x) \]

loves

\[ \lambda P. \forall x. P(x) \]

everybody

\[ \langle \langle e, t \rangle, t \rangle \]
Type Shifting: Two Quantifiers

S

DP

VP

Somebody

V

NP

\[ \langle \langle e, t \rangle, t \rangle, \langle e, t \rangle \rangle \]

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

\[ \uparrow \text{RAISE} - 0 \]

\[ e, \langle e, t \rangle \]

\[ \lambda x. \lambda y. Loves(y, x) \]

\[ \text{loves} \]

\[ \langle e, t \rangle \]

\[ \lambda P. \forall x. P(x) \]

everybody

\[ \text{everybody} \]
Type Shifting: Two Quantifiers

\[ S \]

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

\[ \uparrow RAISE - o \]

\[ \langle e, \langle e, t \rangle \rangle \lambda x. \lambda y. Loves(y, x) \]

\[ \text{loves} \]

\[ \langle \langle e, t \rangle, t \rangle \rangle \lambda P. \forall x. P(x) \]

\[ \text{everybody} \]
Type Shifting: Two Quantifiers

\[ S \]

\[ \langle\langle e, t\rangle, t\rangle \]
\[ \lambda P. \exists y. P(y) \]
\[ \text{Somebody} \]

\[ V \]
\[ \langle\langle\langle e, t\rangle, t\rangle, \langle e, t\rangle\rangle \]
\[ \lambda Q_{\langle e, t\rangle, t} \lambda y. Q(\lambda x. Loves(y, x)) \]
\[ \uparrow \text{RAISE} - o \]
\[ \langle e, \langle e, t\rangle\rangle \]
\[ \lambda x. \lambda y. Loves(y, x) \]
\[ \text{loves} \]

\[ \text{NP} \]
\[ \langle\langle e, t\rangle, t\rangle \]
\[ \lambda P. \forall x. P(x) \]
\[ \text{everybody} \]
Type Shifting: Two Quantifiers

\[ S \]
\[ t \]
\[ \exists y. \forall x. \text{Loves}(y, x) \]

\[ \text{DP} \]
\[ \langle (e, t), t \rangle \]
\[ \lambda P. \exists y. P(y) \]
\[ \text{Somebody} \]

\[ \text{VP} \]
\[ \langle e, t \rangle \]
\[ \lambda y. \forall x. \text{Loves}(y, x) \]

\[ \text{V} \]
\[ \langle (e, t), \langle e, t \rangle \rangle \]
\[ \lambda Q_{(e, t)} \lambda y. Q((\lambda x. \text{Loves}(y, x))) \]
\[ \uparrow \text{RAISE} - o \]
\[ \langle (e, e, t) \rangle \]
\[ \lambda x. \lambda y. \text{Loves}(y, x) \]
\[ \text{loves} \]

\[ \text{NP} \]
\[ \langle (e, t), t \rangle \]
\[ \lambda P. \forall x. P(x) \]
\[ \text{everybody} \]

{ o.petukhova; nndascalu }@lsv.uni-saarland.de
Type Shifting: Subject Raising

But what about inverse scope reading?

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

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

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

(unless $Q$, $y$ or $x$ occurs in $\alpha'$; in that case, use different variables).
But what about inverse scope reading?

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

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

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

(unless $Q$, $y$ or $x$ occurs in $\alpha'$; in that case, use different variables).
Type Shifting: RAISE-S

\[
\forall x \exists y. \text{Loves}(y, x)
\]

\[
\exists y. \lambda P. \text{Loves}(y, x)
\]

\[
\forall x. \lambda Q. \text{Loves}(y, x)
\]

⇑ RAISE − O

\[
\lambda x. \lambda Q. \text{Loves}(y, x)
\]

⇑ RAISE − S
Type Shifting: RAISE-S

\[ \forall x \exists y. \text{Loves}(y, x) \]

\[ \langle\langle e, t\rangle, t\rangle, \lambda P. \forall x. P(x) \]

\[ \langle\langle e, t\rangle, t\rangle, \langle\langle e, t\rangle, t\rangle, t\rangle, \lambda P. \forall x. P(x) \]

\[ \langle\langle e, t\rangle, t\rangle, \langle\langle e, t\rangle, t\rangle, t\rangle, \lambda P. \forall x. P(x) \]

\[ \text{Somebody} \]

\[ \langle\langle e, t\rangle, t\rangle, \text{everybody} \]

\[ \text{loves} \]
Type Shifting: RAISE-S

S

| DP
| | Somebody
|

| VP
| | V
| | NP
| | ⟨⟨⟨e, t⟩, t⟩⟩
| | λP.∀x.P(x)
| | everybody

⟨⟨⟨e, t⟩, t⟩⟩
λxλy.Loves(y, x)

loves
Type Shifting: RAISE-S

\[ \forall x \exists y. \text{Loves}(y, x) \]
Type Shifting: RAISE-S

\[
\lambda x. \lambda Q_{(e, t)}. Q(\lambda y. \text{Loves}(y, x))
\]

\[
\uparrow_{\text{RAISE-S}}
\]

\[
\lambda x \lambda y. \text{Loves}(y, x)
\]

\[
\text{loves}
\]

\[\langle\langle e, t\rangle, t\rangle\]

\[\lambda P. \forall x. P(x)\]

\[\text{everybody}\]

\[\text{Somebody}\]

\[\langle\langle e, t\rangle, t\rangle, t\rangle\]

\[\langle\langle\langle e, t\rangle, t\rangle, t\rangle\rangle\]
Type Shifting: RAISE-S

\[ \forall x \exists y. \text{Loves}(y, x) \]

\[ \langle \langle e, \langle \langle e, t \rangle, t \rangle \rangle, t \rangle \lambda P. \forall x. P(x) \]

\[ \langle \langle e, t \rangle, t \rangle \lambda P. \forall x. P(x) \]

\[ \forall x. \lambda y. \text{Loves}(y, x) \]

\[ \langle \langle e, \langle e, t \rangle \rangle, t \rangle \lambda x. \lambda y. \text{Loves}(y, x) \]

\[ \langle \langle e, t \rangle, t \rangle \lambda P. \forall x. P(x) \]

\[ \forall x. \lambda y. \text{Loves}(y, x) \]

\[ \lambda x. \lambda y. \text{Loves}(y, x) \]

\[ \lambda P. \forall x. P(x) \]

\[ \lambda x. \lambda y. \text{Loves}(y, x) \]

\[ \lambda x. \lambda y. \text{Loves}(y, x) \]

\[ \lambda P. \forall x. P(x) \]

\[ \forall x. \lambda y. \text{Loves}(y, x) \]

\[ \lambda x. \lambda y. \text{Loves}(y, x) \]

\[ \lambda x. \lambda y. \text{Loves}(y, x) \]

\[ \lambda P. \forall x. P(x) \]

\[ \forall x. \lambda y. \text{Loves}(y, x) \]

\[ \lambda x. \lambda y. \text{Loves}(y, x) \]

\[ \lambda P. \forall x. P(x) \]

\[ \forall x. \lambda y. \text{Loves}(y, x) \]
Type Shifting: RAISE-S

\[
\forall x \exists y. \text{Loves}(y, x)
\]

\[
\text{DP} \langle\langle e, t \rangle, t \rangle \lambda P. \exists y. P(y)
\]

\[
\text{VP} \langle\langle e, t \rangle, t \rangle \lambda Q \langle e, t \rangle, t \rangle \lambda x. \text{Q}(\lambda y. \text{Loves}(y, x))
\]

\[
\text{V} \langle\llangle e, t \rrangle, t \rrangle, t \rrangle \rangle \lambda Q' \langle e, t \rangle, t \rangle \lambda x. Q(\lambda y. \text{Loves}(y, x))
\]

\[
\text{NP} \langle\langle e, t \rangle, t \rangle \lambda P. \forall x. P(x)
\]

Somebody

\[
\text{loves}
\]
Type Shifting: RAISE-S

\[
S \
\downarrow \text{Somebody} \
\downarrow \lambda Q_{(e,t)} \cdot \forall x. Q(\lambda y. \text{Loves}(y, x)) \
\downarrow \lambda x. \lambda Q_{(e,t)} \cdot Q(\lambda y. \text{Loves}(y, x)) \
\uparrow \text{RAISE-S} \
\langle e, \langle e, t \rangle \rangle \
\lambda x \lambda y. \text{Loves}(y, x) \
\text{loves}
\]
Type Shifting: RAISE-S

\[ \lambda x. \lambda y. \text{Loves}(y, x) \]
\[ \uparrow \text{RAISE-S} \]
\[ \langle e, \langle e, t \rangle \rangle \]
\[ \lambda x. \lambda Q_{(e, t), t} \cdot Q(\lambda y. \text{Loves}(y, x)) \]
\[ \uparrow \text{RAISE-O} \]
\[ \langle e, \langle e, t \rangle, t \rangle \]
\[ \lambda Q'_{(e, t), t} \lambda Q_{(e, t), t} \cdot Q'(\lambda x. Q(\lambda y. \text{Loves}(y, x))) \]
\[ \langle (e, t), t \rangle \]
\[ \lambda P. \forall x. P(x) \]
\[ \forall x. \lambda Q_{(e, t), t} \cdot Q(\lambda y. \text{Loves}(y, x)) \]
\[ \exists y. P(y) \]
\[ \exists y. \text{Loves}(y, x) \]
\[ \langle (e, t), t \rangle \]
\[ \lambda P. \exists y. P(y) \]

DP
\[ \langle (e, t), t \rangle \]
\[ \lambda P. \exists y. P(y) \]
\[ \forall x. \text{Loves}(y, x) \]
\[ \text{Somebody} \]

VP
\[ \langle (e, t), t \rangle \]
\[ \lambda Q_{(e, t), t} \cdot \forall x. Q(\lambda y. \text{Loves}(y, x)) \]

V
\[ \langle (e, t), t \rangle \]
\[ \forall x. \text{Loves}(y, x) \]

NP
\[ \langle (e, t), t \rangle \]
\[ \lambda P. \forall x. P(x) \]
\[ \text{everybody} \]
Type Shifting: RAISE-S

\[
S \quad t
\]
\[
\forall x \exists y. \text{Loves}(y, x)
\]

\[
\text{Somebody}
\]

\[
\langle\langle e, t \rangle, t \rangle
\]
\[
\lambda P. \exists y. P(y)
\]

\[
\langle\langle\langle e, t \rangle, t \rangle, t \rangle
\]
\[
\lambda Q_{\langle e, t \rangle, t}. \forall x. Q(\lambda y. \text{Loves}(y, x))
\]

\[
\text{everybody}
\]

\[
\langle\langle\langle e, t \rangle, t \rangle, t \rangle
\]
\[
\lambda Q_{\langle e, t \rangle, t}. \forall x. Q(\lambda y. \text{Loves}(y, x))
\]

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

\[
\langle\langle e, t \rangle, t \rangle
\]
\[
\lambda P. \forall x. P(x)
\]

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

\[
\langle\langle e, t \rangle, t \rangle
\]
\[
\lambda x. \lambda y. \text{Loves}(y, x)
\]

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

\[
\langle\langle e, t \rangle, t \rangle
\]
\[
\lambda x. \lambda y. \text{Loves}(y, x)
\]

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

\[
\langle\langle e, t \rangle, t \rangle
\]
\[
\lambda x. \lambda y. \text{Loves}(y, x)
\]
Quizz for Today

TBA