1 Semantic Composition
   - Review: compositionality, lambda expressions, and logical forms
   - Examples
   - Type raising

2 Semantic (Scope) Ambiguity
   - Definition
   - Semantic Scope
   - Approaches to Scope Ambiguity
   - Underspecification: General Idea
**Compositionality**: The meaning of a complex expression is a function of the meaning of its parts and of the rules by which they are combined.

Do we have sufficient tools to systematically compute meaning representations according to this principle?

- The meaning of a complete sentence will hopefully be a FOPL formula, which we consider as having type \( t \) (truth values).
- But the meaning of smaller fragments of the sentence will have other types. E.g.
  
  \[
  \text{has a bone } \quad \langle e, t \rangle \\
  \text{every dog } \quad \langle \langle e, t \rangle, t \rangle
  \]

- The idea is to show how to associate a meaning with such fragments, and how these meanings combine.

- To do this, we need to extend our language of FOPL with \( \lambda \) expressions (\( \lambda = \) lambda; written as \( \backslash \) in Haskell).
**Lambda (\(\lambda\)) Expressions**

\(\lambda\)-expressions are an extension to FOPL that allows us to work with ‘partially constructed’ formulae. A \(\lambda\)-expression consists of:

- the Greek letter \(\lambda\), followed by a variable (formal parameter);
- a FOPL expression that may involve that variable.

\(\lambda x.\text{sleep}(x) : < e, t >\)

‘The function that takes an entity \(x\) to the statement sleep(\(x\))’

\(\left(\lambda x.\text{sleep}(x)\right)(\text{Kim}) : t\)

A \(\lambda\)-expression can be applied to a term.

(The above has the same truth value as sleep(\(Kim\)).)
Lambda expressions can be nested. We can use nesting to create functions of several arguments that accept their arguments one at a time.

\[ \lambda y.\lambda x. \text{love}(x,y) : < e, < e, t >> \]
‘The function that takes \( y \) to (the function that takes \( x \) to the statement \( \text{love}(x,y) \))’

\[ \lambda z.\lambda y.\lambda x. \text{give}(x,y,z) : < e, < e, < e, t >>> \]
‘The function that takes \( z \) to (the function that takes \( y \) to (the function that takes \( x \) to the statement \( \text{give}(x,y,z) \)))’
When a lambda expression applies to a term, a reduction operation \((\text{beta (}\beta\text{) reduction})\) can be used to replace its formal parameter with the term and simplify the result. In general:

\[(\lambda x. M)N \Rightarrow_\beta M[x \mapsto N] \quad (M \text{ with } N \text{ substituted for } x)\]

\((\lambda x. \text{sleep}(x))(\text{Kim}) \Rightarrow_\beta \text{sleep(Kim)}\)

\[(\lambda y. \lambda x. \text{love}(x, y))(\text{crabapples}) \Rightarrow_\beta \lambda x. \text{love}(x, \text{crabapples})\]

\[(\lambda x. \text{love}(x, \text{crabapples}))(\text{Kim}) \Rightarrow_\beta \text{love(Kim, crabapples)}\]
To build a compositional semantics for NL, we attach valuation functions to grammar rules (semantic attachments).

These show how to compute the interpretation of the LHS of the rule from the interpretations of its RHS components.

For example, VP.Sem(NP.Sem) means apply the interpretation of the VP to the interpretation of the NP.

Types have been added to ease understanding.
Compositional Semantics: example

\[ S[\lambda x.\text{love}(x, \text{Kim})(\text{Sam}) \Rightarrow_\beta \text{love}(\text{Sam}, \text{Kim})] \]

\[ \text{NP[Sam]} \quad \text{VP[}\lambda y.\lambda x.\text{love}(x, y)(\text{Kim}) \Rightarrow_\beta \lambda x.\text{love}(x, \text{Kim})] \]

\[ \text{NPR[Sam]} \quad \text{TV[}\lambda y.\lambda x.\text{love}(x, y)] \quad \text{NP[Kim]} \]

\[ \text{Sam} \quad \text{loves} \quad \text{NPR[Kim]} \]

\[ \text{Kim} \]
Compositional Semantics: example

\[ S[\lambda x.\text{love}(x, Kim)(Sam) \Rightarrow_{\beta} \text{love}(Sam, Kim)] \]

\[ \text{NP}[\text{Sam}] \quad \text{VP}[\lambda y.\lambda x.\text{love}(x, y)(Kim) \Rightarrow_{\beta} \lambda x.\text{love}(x, Kim)] \]

\[ \text{NPR}[\text{Sam}] \quad \text{TV}[\lambda y.\lambda x.\text{love}(x, y)] \quad \text{NP}[\text{Kim}] \]

\[ \text{Sam} \quad \text{loves} \quad \text{NPR}[\text{Kim}] \]

\[ \text{Kim} \]
Compositional Semantics: example

\[ S[\lambda x.\text{love}(x, \text{Kim})(\text{Sam}) \Rightarrow_{\beta} \text{love}(\text{Sam}, \text{Kim})] \]

\[
\begin{array}{l}
\text{NP}[	ext{Sam}] \quad \text{VP}[\lambda y.\lambda x.\text{love}(x, y)(\text{Kim}) \Rightarrow_{\beta} \lambda x.\text{love}(x, \text{Kim})] \\
\text{NPR}[	ext{Sam}] \quad \text{TV}[\lambda y.\lambda x.\text{love}(x, y)] \quad \text{NP}[	ext{Kim}] \\
\text{Sam} \quad \text{loves} \quad \text{NPR}[	ext{Kim}] \\
\text{Kim}
\end{array}
\]
Compositional Semantics: example

\[ S[\lambda x. love(x, Kim)(Sam) \Rightarrow_\beta love(Sam, Kim)] \]

\[
\begin{array}{c}
\text{NP}[Sam] \\
\text{NPR}[Sam] \\
\text{Sam}
\end{array}
\]

\[
\begin{array}{c}
\text{NP}[Kim] \\
\text{NPR}[Kim] \\
\text{Kim}
\end{array}
\]

\[
\begin{array}{c}
\text{VP}[\lambda y. \lambda x. love(x, y)(Kim) \Rightarrow_\beta \lambda x. love(x, Kim)] \\
\text{TV}[\lambda y. \lambda x. love(x, y)]
\end{array}
\]
Compositional Semantics: example

\[ S[\lambda x.\text{love}(x, \text{Kim})(\text{Sam}) \Rightarrow_\beta \text{love}(\text{Sam}, \text{Kim})] \]
Compositional Semantics: example

\[ S[\lambda x.\text{love}(x, Kim)(Sam) \Rightarrow_\beta \text{love}(Sam, Kim)] \]
A minor variation

The following alternative semantics assigns the same overall meaning to sentences. Only the treatment of the arguments of ‘love’ is different.

<table>
<thead>
<tr>
<th>Grammar 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
</tr>
<tr>
<td>VP → TV NP</td>
</tr>
<tr>
<td>NP → NPR</td>
</tr>
<tr>
<td>TV → loves</td>
</tr>
<tr>
<td>NPR → Kim</td>
</tr>
<tr>
<td>NPR → Sam</td>
</tr>
<tr>
<td>{VP.Sem(NP.Sem)}</td>
</tr>
<tr>
<td>{λx.TV.Sem(x)(NP.Sem)}</td>
</tr>
<tr>
<td>{NPR.Sem}</td>
</tr>
<tr>
<td>{Kim}</td>
</tr>
<tr>
<td>{Sam}</td>
</tr>
<tr>
<td>t</td>
</tr>
<tr>
<td>&lt; e, t &gt;</td>
</tr>
<tr>
<td>e</td>
</tr>
<tr>
<td>e</td>
</tr>
<tr>
<td>&lt; e, &lt; e, t &gt;&gt;</td>
</tr>
<tr>
<td>e</td>
</tr>
<tr>
<td>e</td>
</tr>
</tbody>
</table>
What about the interpretation of an NP other than a proper name? The FOPL interpretation should often contain an existential (∃) or a universal (∀) quantifier:

Sam has access to a computer.
∃x(\text{computer}(x) \land \text{have_access_to}(\text{Sam}, x))

Every student has access to a computer.
∀x(\text{student}(x) \rightarrow ∃y(\text{computer}(y) \land \text{have_access_to}(x, y)))

Can we build such interpretations up from their component parts in the same way as with proper names?
A halfway stage.

<table>
<thead>
<tr>
<th>Grammar II</th>
</tr>
</thead>
<tbody>
<tr>
<td>S \rightarrow NPR VP { VP.Sem(NPR.Sem) } t</td>
</tr>
<tr>
<td>VP \rightarrow TV a Nom { \lambda x.\exists y.\text{Nom.Sem}(y) &amp; TV.Sem(y)(x) }</td>
</tr>
<tr>
<td>Nom \rightarrow N { N.Sem } &lt; e, t &gt;</td>
</tr>
<tr>
<td>Nom \rightarrow A Nom { \lambda x.\text{Nom.Sem}(x) &amp; A.Sem(x) } &lt; e, t &gt;</td>
</tr>
<tr>
<td>NPR \rightarrow Sam { Sam } e</td>
</tr>
<tr>
<td>TV \rightarrow loves { \lambda y.\lambda x.\text{love}(x, y) } &lt; e, &lt; e, t &gt;&gt;</td>
</tr>
<tr>
<td>N \rightarrow woman { \lambda z.\text{woman}(z) } &lt; e, t &gt;</td>
</tr>
<tr>
<td>A \rightarrow tall { \lambda z.\text{tall}(z) } &lt; e, t &gt;</td>
</tr>
</tbody>
</table>

- Note we haven’t given a meaning here to a tall woman.
- Could take this to have the same meaning as tall woman.
- This would be fine for this example (also in Assignment 2). But what about every tall woman?
A halfway stage.

Grammar II

<table>
<thead>
<tr>
<th>Rule</th>
<th>Left-hand side</th>
<th>Right-hand side</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NPR VP</td>
<td>{ VP.Sem(NPR.Sem) }</td>
<td>t</td>
</tr>
<tr>
<td>VP → TV a Nom</td>
<td>λx.∃y. Nom.Sem(y) &amp; TV.Sem(y)(x)</td>
<td>&lt; e, t &gt;</td>
</tr>
<tr>
<td>Nom → N</td>
<td>{ N.Sem }</td>
<td>&lt; e, t &gt;</td>
</tr>
<tr>
<td>Nom → A Nom</td>
<td>λx. Nom.Sem(x) &amp; A.Sem(x)</td>
<td>&lt; e, t &gt;</td>
</tr>
<tr>
<td>NPR → Sam</td>
<td>{ Sam }</td>
<td>e</td>
</tr>
<tr>
<td>TV → loves</td>
<td>λy.λx.love(x, y)</td>
<td>&lt; e, &lt; e, t &gt;&gt;</td>
</tr>
<tr>
<td>N → woman</td>
<td>λz.woman(z)</td>
<td>&lt; e, t &gt;</td>
</tr>
<tr>
<td>A → tall</td>
<td>λz.tall(z)</td>
<td>&lt; e, t &gt;</td>
</tr>
</tbody>
</table>

- Note we haven’t given a meaning here to a tall woman.
- Could take this to have the same meaning as tall woman.
- This would be fine for this example (also in Assignment 2). But what about every tall woman?
Grammar II

\[
S \rightarrow \text{NPR} \ \text{VP} \quad \{ \text{VP.Sem(NPR.Sem)} \} \quad t
\]

\[
\text{VP} \rightarrow \text{TV} \ a \ \text{Nom} \quad \{ \lambda x.\exists y.\text{Nom.Sem}(y) \& \text{TV.Sem}(y)(x) \} \quad < e, t >
\]

\[
\text{Nom} \rightarrow \text{N} \quad \{ \text{N.Sem} \} \quad < e, t >
\]

\[
\text{Nom} \rightarrow \text{A} \ \text{Nom} \quad \{ \lambda x.\text{Nom.Sem}(x) \& \text{A.Sem}(x) \} \quad < e, t >
\]

\[
\text{NPR} \rightarrow \text{Sam} \quad \{ \text{Sam} \} \quad e
\]

\[
\text{TV} \rightarrow \text{loves} \quad \{ \lambda y.\lambda x.\text{love}(x, y) \} \quad < e, < e, t >>
\]

\[
\text{N} \rightarrow \text{woman} \quad \{ \lambda z.\text{woman}(z) \} \quad < e, t >
\]

\[
\text{A} \rightarrow \text{tall} \quad \{ \lambda z.\text{tall}(z) \} \quad < e, t >
\]

- Note we haven’t given a meaning here to a tall woman.
- Could take this to have the same meaning as tall woman.
- This would be fine for this example (also in Assignment 2). But what about every tall woman?
Grammar II

<table>
<thead>
<tr>
<th>Rule</th>
<th>Definition</th>
<th>Semantic Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NPR VP</td>
<td>{ VP.Sem(NPR.Sem) }</td>
<td>t</td>
</tr>
<tr>
<td>VP → TV a Nom</td>
<td>{ λx.∃y. Nom.Sem(y) &amp; TV.Sem(y)(x) }</td>
<td>&lt; e, t &gt;</td>
</tr>
<tr>
<td>Nom → N</td>
<td>{ N.Sem }</td>
<td>&lt; e, t &gt;</td>
</tr>
<tr>
<td>Nom → A Nom</td>
<td>{ λx. Nom.Sem(x) &amp; A.Sem(x) }</td>
<td>&lt; e, t &gt;</td>
</tr>
<tr>
<td>NPR → Sam</td>
<td>{ Sam }</td>
<td>e</td>
</tr>
<tr>
<td>TV → loves</td>
<td>{ λy.λx.love(x, y) }</td>
<td>&lt; e, &lt; e, t &gt;&gt;</td>
</tr>
<tr>
<td>N → woman</td>
<td>{ λz.woman(z) }</td>
<td>&lt; e, t &gt;</td>
</tr>
<tr>
<td>A → tall</td>
<td>{ λz.tall(z) }</td>
<td>&lt; e, t &gt;</td>
</tr>
</tbody>
</table>

- Note we haven’t given a meaning here to **a tall woman**.
- Could take this to have the same meaning as **tall woman**.
- This would be fine for this example (also in Assignment 2). But what about **every tall woman**?
Before we add more, let’s use Grammar II to compute the semantics of *Sam loves a tall woman*.

<table>
<thead>
<tr>
<th>loves</th>
<th>TV</th>
<th>$\lambda y x. \text{love}(x, y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>tall woman</td>
<td>Nom</td>
<td>$\lambda x. (\lambda z.\text{woman}(z))(x) &amp; (\lambda z.\text{tall}(z))(x)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Rightarrow_\beta \lambda x. \text{woman}(x) &amp; \text{tall}(x)$</td>
</tr>
<tr>
<td>loves a tall woman</td>
<td>VP</td>
<td>$\lambda x. \exists y. (\lambda x. \text{woman}(x) &amp; \text{tall}(x))(y) &amp; (\lambda y x. \text{love}(x, y))(y)(x)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Rightarrow_\beta \lambda x. \exists y. (\text{woman}(y) &amp; \text{tall}(y)) &amp; \text{love}(x, y)$</td>
</tr>
<tr>
<td>Sam loves a tall woman</td>
<td>S</td>
<td>$(\lambda x.\exists y. \cdots)(\text{Sam})$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Rightarrow_\beta \exists y. \text{woman}(y) &amp; \text{tall}(y) &amp; \text{love}(\text{Sam}, y)$</td>
</tr>
</tbody>
</table>
Before we add more, let’s use Grammar II to compute the semantics of *Sam loves a tall woman*.

- **loves**
  - TV: $\lambda y x. \text{love}(x, y)$

- **tall woman**
  - Nom: $\lambda x. (\lambda z. \text{woman}(z))(x) \& (\lambda z. \text{tall}(z))(x)$
  - $\Rightarrow \beta$ $\lambda x. \text{woman}(x) \& \text{tall}(x)$

- **loves a tall woman**
  - VP: $\lambda x. \exists y. (\lambda x. \text{woman}(x) \& \text{tall}(x))(y) \& (\lambda y x. \text{love}(x, y))(y)(x)$
  - $\Rightarrow \beta$ $\lambda x. \exists y. (\text{woman}(y) \& \text{tall}(y)) \& \text{love}(x, y)$

- **Sam loves a tall woman**
  - S: $(\lambda x. \exists y. \cdots)(\text{Sam})$
  - $\Rightarrow \beta$ $\exists y. \text{woman}(y) \& \text{tall}(y) \& \text{love}(\text{Sam}, y)$
Before we add more, let’s use Grammar II to compute the semantics of Sam loves a tall woman.

<table>
<thead>
<tr>
<th>loves</th>
<th>TV</th>
<th>$\lambda yx. \text{love}(x, y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>tall woman</td>
<td>Nom</td>
<td>$\lambda x. (\lambda z. \text{woman}(z))(x) &amp; (\lambda z. \text{tall}(z))(x)$</td>
</tr>
<tr>
<td></td>
<td>$\Rightarrow \beta$</td>
<td>$\lambda x. \text{woman}(x) &amp; \text{tall}(x)$</td>
</tr>
<tr>
<td>loves a tall woman</td>
<td>VP</td>
<td>$\lambda x. \exists y. (\lambda x. \text{woman}(x) &amp; \text{tall}(x))(y) &amp; (\lambda yx. \text{love}(x, y))(y)(x)$</td>
</tr>
<tr>
<td></td>
<td>$\Rightarrow \beta$</td>
<td>$\lambda x. \exists y. (\text{woman}(y) &amp; \text{tall}(y)) &amp; \text{love}(x, y)$</td>
</tr>
<tr>
<td>Sam loves a tall woman</td>
<td>S</td>
<td>$(\lambda x. \exists y. \cdots)(\text{Sam})$</td>
</tr>
<tr>
<td></td>
<td>$\Rightarrow \beta$</td>
<td>$\exists y. \text{woman}(y) &amp; \text{tall}(y) &amp; \text{love}(\text{Sam}, y)$</td>
</tr>
</tbody>
</table>
Before we add more, let’s use Grammar II to compute the semantics of *Sam loves a tall woman*.

- **loves**
  - TV: $\lambda y x. \text{love}(x, y)$
- **tall woman**
  - Nom: $\lambda x. (\lambda z. \text{woman}(z))(x) \& (\lambda z. \text{tall}(z))(x)$
    - $\Rightarrow \beta$: $\lambda x. \text{woman}(x) \& \text{tall}(x)$
- **loves a tall woman**
  - VP: $\lambda x. \exists y. (\lambda x. \text{woman}(x) \& \text{tall}(x))(y) \& (\lambda y x. \text{love}(x, y))(y)(x)$
    - $\Rightarrow \beta$: $\lambda x. \exists y. (\text{woman}(y) \& \text{tall}(y)) \& \text{love}(x, y)$
- **Sam loves a tall woman**
  - S: $(\lambda x. \exists y. \cdots)(\text{Sam})$
    - $\Rightarrow \beta$: $\exists y. \text{woman}(y) \& \text{tall}(y) \& \text{love}(\text{Sam}, y)$
Before we add more, let’s use Grammar II to compute the semantics of **Sam loves a tall woman**.

- **loves**: $TV \quad \lambda xy. love(x, y)$
- **tall woman**: $Nom \quad \lambda x. (\lambda z. woman(z))(x) \land (\lambda z. tall(z))(x)$
  \[ \Rightarrow \beta \quad \lambda x. woman(x) \land tall(x) \]
- **loves a tall woman**: $VP \quad \lambda x. \exists y. (\lambda x. woman(x) \land tall(x))(y) \land (\lambda y. love(x, y))(y)(x)$
  \[ \Rightarrow \beta \quad \lambda x. \exists y. (woman(y) \land tall(y)) \land love(x, y) \]
- **Sam loves a tall woman**: $S \quad (\lambda x. \exists y. \cdots)(Sam)$
  \[ \Rightarrow \beta \quad \exists y. woman(y) \land tall(y) \land love(Sam, y) \]
Before we add more, let’s use Grammar II to compute the semantics of *Sam loves a tall woman*.

- **loves**
  - **TV** \( \lambda y x. \text{love}(x, y) \)
- **tall woman**
  - **Nom** \( \lambda x. (\lambda z. \text{woman}(z))(x) \& (\lambda z. \text{tall}(z))(x) \)
  - \( \Rightarrow \beta \) \( \lambda x. \text{woman}(x) \& \text{tall}(x) \)
- **loves a tall woman**
  - **VP** \( \lambda x. \exists y. (\lambda x. \text{woman}(x) \& \text{tall}(x))(y) \& (\lambda y x. \text{love}(x, y))(y)(x) \)
  - \( \Rightarrow \beta \) \( \lambda x. \exists y. (\text{woman}(y) \& \text{tall}(y)) \& \text{love}(x, y) \)
- **Sam loves a tall woman**
  - **S** \( (\lambda x. \exists y. \cdots)(\text{Sam}) \)
  - \( \Rightarrow \beta \) \( \exists y. \text{woman}(y) \& \text{tall}(y) \& \text{love}(\text{Sam}, y) \)
Before we add more, let’s use Grammar II to compute the semantics of *Sam loves a tall woman*.

<table>
<thead>
<tr>
<th>loves</th>
<th>TV</th>
<th>$\lambda y x. \text{love}(x, y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>tall woman</td>
<td>Nom</td>
<td>$\lambda x. (\lambda z. \text{woman}(z))(x) &amp; (\lambda z. \text{tall}(z))(x)$</td>
</tr>
<tr>
<td></td>
<td>$\Rightarrow_\beta$</td>
<td>$\lambda x. \text{woman}(x) &amp; \text{tall}(x)$</td>
</tr>
<tr>
<td>loves a tall woman</td>
<td>VP</td>
<td>$\lambda x. \exists y. (\lambda x. \text{woman}(x) &amp; \text{tall}(x))(y) &amp; (\lambda y x. \text{love}(x, y))(y)(x)$</td>
</tr>
<tr>
<td></td>
<td>$\Rightarrow_\beta$</td>
<td>$\lambda x. \exists y. (\text{woman}(y) &amp; \text{tall}(y)) &amp; \text{love}(x, y)$</td>
</tr>
</tbody>
</table>

**Sam loves a tall woman**

<table>
<thead>
<tr>
<th>Sam loves a tall woman</th>
<th>S</th>
<th>$(\lambda x. \exists y. \cdots)(\text{Sam})$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\Rightarrow_\beta$</td>
<td>$\exists y. \text{woman}(y) &amp; \text{tall}(y) &amp; \text{love}(\text{Sam}, y)$</td>
</tr>
</tbody>
</table>
We’ve given Sam, Kim the semantic type \( e \), and woman the semantic type \(<e, t>\).

But what type should some woman or every woman have?

Idea: Since we wish to combine an NP.Sem with a VP.Sem (of type \(<e, t>\)) to get an S.Sem (of type \(t\)), let’s try again with NP.Sem having type \(<<e, t>, t>\).

\[
\begin{align*}
\text{Sam} & \quad \lambda P. P(Sam) \quad \text{(type raising)} \\
\text{every woman} & \quad \lambda P. \forall x. \text{woman}(x) \Rightarrow P(x)
\end{align*}
\]

The appropriate semantic attachment for NP VP is then

\[
S \rightarrow \text{NP VP} \quad \{\text{NP.Sem (VP.Sem)}\} 
\]
Using this approach, we can also derive the semantics of ‘every woman’ from that of ‘every’ and ‘woman’.

We’ve seen that ‘woman’ has semantic type \(< e, t >\), and ‘every woman’ has semantic type \(<< e, t >, t >\).

So the interpretation of ‘every’ should have type \(<< e, t >, << e, t >, t >>\). Similarly for other determiners (e.g. every, a, no, not every).

\[
\begin{align*}
\text{woman} & \quad \lambda x. \text{woman}(x) \quad < e, t > \\
\text{every} & \quad \lambda Q. \lambda P. \forall x. Q(x) \Rightarrow P(x) \quad <<< e, t >, <<< e, t >, t >> \\
a & \quad \lambda Q. \lambda P. \exists x. Q(x) \land P(x) \quad <<< e, t >, <<< e, t >, t >> \\
\text{NP} \rightarrow \text{Det N} & \quad \{ \text{Det.Sem (N.Sem)} \} \quad <<< e, t >, t >
\end{align*}
\]

We can now compute the semantics of ‘every woman’ and check that it \(\beta\)-reduces to \(\lambda P. \forall x. \text{woman}(x) \Rightarrow P(x)\).
The semantics of “every woman”:
The natural rule for VP is now VP → TV NP.

Since the semantic type for NP has now been raised to $<\langle e, t, t \rangle, t>$, and we want VP to have semantic type $<e, t>$, what should the semantic type for TV be?
More on type raising

- The natural rule for VP is now VP → TV NP.
- Since the semantic type for NP has now been raised to << e, t >, t >, and we want VP to have semantic type < e, t >, what should the semantic type for TV be?

It had better be << < e, t >, t >, < e, t >>. (A 3rd order function type!)

TV → loves \{ λR<<e,t>,t>.λz^e. R(λw^e. loves(z, w)) \}
VP → TV NP \{ TV.Sem(NP.Sem) \}
To summarize where we’ve got to:

**Grammar III**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Definition</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>{ NP.Sem(VP.Sem) } t</td>
<td></td>
</tr>
<tr>
<td>VP → TV NP</td>
<td>{ TV.Sem(NP.Sem) } &lt; e, t &gt;</td>
<td></td>
</tr>
<tr>
<td>NP → Sam</td>
<td>{ λP.P(Sam) } &lt;&lt; e, t &gt;, t &gt;</td>
<td></td>
</tr>
<tr>
<td>NP → Det Nom</td>
<td>{ Det.Sem(Nom.Sem) } &lt;&lt; e, t &gt;, t &gt;</td>
<td></td>
</tr>
<tr>
<td>Det → a</td>
<td>{ λQ.λP.∃x.Q(x) ∧ P(x) } &lt;&lt; e, t &gt;, &lt;&lt; e, t &gt;, t &gt;</td>
<td></td>
</tr>
<tr>
<td>Det → every</td>
<td>{ λQ.λP.∀x.Q(x) ⇒ P(x) } &lt;&lt; e, t &gt;, &lt;&lt; e, t &gt;, t &gt;&gt;</td>
<td></td>
</tr>
<tr>
<td>Nom → N</td>
<td>{ N.Sem } &lt; e, t &gt;</td>
<td></td>
</tr>
<tr>
<td>Nom → A Nom</td>
<td>{ λx. Nom.Sem(x) &amp; A.Sem(x) } &lt; e, t &gt;</td>
<td></td>
</tr>
<tr>
<td>TV → loves</td>
<td>{ λR.λz. R(λw. loves(z, w)) } &lt;&lt;&lt; e, t &gt;, t &gt;, &lt; e, t &gt;&gt;</td>
<td></td>
</tr>
<tr>
<td>N → woman</td>
<td>{ λz.woman(z) } &lt; e, t &gt;</td>
<td></td>
</tr>
<tr>
<td>A → tall</td>
<td>{ λz.tall(z) } &lt; e, t &gt;</td>
<td></td>
</tr>
</tbody>
</table>

Can add similar entries for ‘student’, ‘computer’, ‘has access to’.
The semantics for ‘every student has access to a computer’.

\[ \lambda Q.\lambda P.\forall x.Q(x) \Rightarrow P(x) \rightarrow \beta \lambda P.\forall x.\text{student}(x) \Rightarrow P(x) \]

\[ \lambda Q.\lambda P.\exists x.Q(x) \land P(x) \rightarrow \beta \lambda P.\exists x.\text{computer}(x) \land P(x) \]

Note: In the last \( \beta \)-step, we've renamed 'x' to 'y' to avoid capture.
Example

The semantics for ‘every student has access to a computer’.

every student \((\lambda Q.\lambda P. \forall x. Q(x) \Rightarrow P(x))(\lambda x.\text{student}(x))\)  
\rightarrow_{\beta} \lambda P. \forall x. \text{student}(x) \Rightarrow P(x)
Example

The semantics for ‘every student has access to a computer’.

\[
\begin{align*}
\text{every student} & \quad (\lambda Q. \lambda P. \forall x. Q(x) \implies P(x))(\lambda x. \text{student}(x)) \\
& \quad \rightarrow_\beta \quad \lambda P. \forall x. \text{student}(x) \implies P(x)
\end{align*}
\]

\[
\begin{align*}
\text{a computer} & \quad (\lambda Q. \lambda P. \exists x. Q(x) \land P(x))(\lambda x. \text{computer}(x)) \\
& \quad \rightarrow_\beta \quad \lambda P. \exists x. \text{computer}(x) \land P(x)
\end{align*}
\]
Example

The semantics for ‘every student has access to a computer’.

- **every student** \((\lambda Q. \lambda P. \forall x. Q(x) \Rightarrow P(x))(\lambda x. \text{student}(x))\)
  \[\to_{\beta} \lambda P. \forall x. \text{student}(x) \Rightarrow P(x)\]

- **a computer** \((\lambda Q. \lambda P. \exists x. Q(x) \land P(x))(\lambda x. \text{computer}(x))\)
  \[\to_{\beta} \lambda P. \exists x. \text{computer}(x) \land P(x)\]

- **h.a.t. a computer** \(\cdots \to_{\beta} \cdots\)
  \[\to_{\beta} \lambda z. \exists x. \text{computer}(x) \land h\_a\_t(z, x)\]
The semantics for ‘every student has access to a computer’.

<table>
<thead>
<tr>
<th>every student</th>
<th>((\lambda Q.\lambda P. \forall x. Q(x) \Rightarrow P(x))(\lambda x.\text{student}(x)))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(\rightarrow_\beta \ \lambda P. \forall x. \text{student}(x) \Rightarrow P(x))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a computer</th>
<th>((\lambda Q.\lambda P. \exists x. Q(x) \land P(x))(\lambda x.\text{computer}(x)))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(\rightarrow_\beta \ \lambda P. \exists x. \text{computer}(x) \land P(x))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>h.a.t. a computer</th>
<th>(\cdots \rightarrow_\beta \cdots)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(\rightarrow_\beta \ \lambda z. \exists x. \text{computer}(x) \land h.a.t(z, x))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(whole sentence)</th>
<th>(\cdots \rightarrow_\beta \cdots)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(\rightarrow_\beta \ \forall x. \text{student}(x) \Rightarrow \exists y. \text{computer}(y) \land h.a.t(x, y))</td>
</tr>
</tbody>
</table>
Example

The semantics for ‘every student has access to a computer’.

\[
\text{every student} \quad (\lambda Q. \lambda P. \forall x. Q(x) \Rightarrow P(x))(\lambda x. \text{student}(x)) \\
\rightarrow_{\beta} \quad \lambda P. \forall x. \text{student}(x) \Rightarrow P(x)
\]

\[
\text{a computer} \quad (\lambda Q. \lambda P. \exists x. Q(x) \land P(x))(\lambda x. \text{computer}(x)) \\
\rightarrow_{\beta} \quad \lambda P. \exists x. \text{computer}(x) \land P(x)
\]

\[
\text{h.a.t. a computer} \quad \cdots \rightarrow_{\beta} \cdots \\
\rightarrow_{\beta} \quad \lambda z. \exists x. \text{computer}(x) \land \text{h.a.t}(z, x)
\]

\[
(\text{whole sentence}) \cdots \rightarrow_{\beta} \cdots \\
\rightarrow_{\beta} \quad \forall x. \text{student}(x) \Rightarrow \exists y. \text{computer}(y) \land \text{h.a.t}(x, y)
\]

Note: In the last $\beta$-step, we’ve renamed ‘$x$’ to ‘$y$’ to avoid capture.
Question

Suppose that the predicate $L(x, y)$ means $x$ loves $y$. Which of the following is not a possible representation of the meaning of *Everybody loves somebody*?

1. $\forall x. \exists y. L(x, y)$
2. $(\lambda P. \forall x. \exists y. P(x, y))(\lambda x. \lambda y. L(x, y))$
3. $(\lambda P. \forall x. \exists y. P(x, y))(\lambda x. \lambda y. L(y, x))$
4. $(\lambda P. \forall x. \exists y. P(y, x))(\lambda x. \lambda y. L(y, x))$
Whilst every student has access to a computer is neither syntactically nor lexically ambiguous, it has two different interpretations because of its determiners:

- **every**: interpreted as \( \forall \) (universal quantifier)
- **a**: interpreted as \( \exists \) (existential quantifier)

### Meaning 1
Possibly a different computer per student
\[
\forall x (\text{student}(x) \rightarrow \exists y (\text{computer}(y) \land \text{have_access_to}(x, y)))
\]

### Meaning 2
Possibly the same computer for all students
\[
\exists y (\text{computer}(y) \land \forall x (\text{student}(x) \rightarrow \text{have_access_to}(x, y)))
\]
The ambiguity arises because *every* and *a* each has its own **scope**:  

Interpretation 1:  *every* has scope over *a*  
Interpretation 2: *a* has scope over *every*  

- Scope is not uniquely determined either by left-to-right order, or by position in the parse tree.
- We therefore need other mechanisms to ensure that the ambiguity is reflected by there being multiple interpretations assigned to S.
The number of interpretations grows exponentially with the number of scope operators:

### Every student at some university has access to a laptop.

1. Not necessarily same laptop, not necessarily same university
   \[
   \forall x (\text{stud}(x) \land \exists y (\text{univ}(y) \land \text{at}(x, y)) \rightarrow \exists z (\text{laptop}(z) \land \text{have_access}(x, z)))
   \]

2. Same laptop, not necessarily same university
   \[
   \exists z (\text{laptop}(z) \land \forall x (\text{stud}(x) \land \exists y (\text{univ}(y) \land \text{at}(x, y)) \rightarrow \text{have_access}(x, z)))
   \]

3. Not necessarily same laptop, same university
   \[
   \exists y (\text{univ}(y) \land \forall x (((\text{stud}(x) \land \text{at}(x, y)) \rightarrow \exists z (\text{laptop}(z) \land \text{have_access}(x, z))))
   \]

4. Same university, same laptop
   \[
   \exists y (\text{univ}(y) \land \exists z (\text{laptop}(z) \land \forall x (((\text{stud}(x) \land \text{at}(x, y)) \rightarrow \text{have_access}(x, z))))
   \]

5. Same laptop, same university
   \[
   \exists z (\text{laptop}(z) \land \exists y (\text{univ}(y) \land \forall x (((\text{stud}(x) \land \text{at}(x, y)) \rightarrow \text{have_access}(x, z))))
   \]
   where 4 & 5 are equivalent

### Every student at some university does not have access to a computer.

\[
\rightarrow 18 \text{ interpretations}
\]
Coping with Scope: options

1. **Enumerate all interpretations.** Computationally unattractive!

2. Use an **underspecified representation** that can be further specified to each of the multiple interpretations on demand.

Sometimes the surrounding context will help us choose between interpretations:

Every student has access to a computer. It can be borrowed from the ITO. (⇒ Meaning 2)
The idea in underspecified representations is that instead of trying to associate a single FOPL formula with a sentence, we associate fragments of formulae with various parts of the sentence.

These fragments can have holes into which other fragments can be plugged. Since there may be some freedom in the order of plugging, the same bunch of fragments can give rise to several formulae with different scoping orders.

There may also be constraints on the order of plugging, corresponding to partial information about the intended interpretation derived e.g. from the discourse context.

See J&M Chapter 18.3 for more on this.
Syntax guides semantic composition in a systematic way.
Lambda expressions facilitate the construction of compositional semantic interpretations.
Logical forms can be constructed by attaching valuation functions to grammar rules.
However, this approach is not adequate enough for quantified NPs, as LFs are not always isomorphic with syntax.
We can elegantly handle scope by building an abstract underspecified representation and disambiguate on demand.