Semantic Theory
Week 4 – Typed Lambda Calculus

Noortje Venhuizen

Universität des Saarlandes

Summer 2017

Recap: Type Theory — Syntax

For every type τ, the set of well-formed expressions WE_τ is defined as follows:

(i) $CON_\tau \subseteq WE_\tau$ and $VAR_\tau \subseteq WE_\tau$;

(ii) If $\alpha \in WE_{\langle\sigma, \tau\rangle}$, and $\beta \in WE_\sigma$, then $\alpha(\beta) \in WE_\tau$; (function application)

(iii) If A, B are in WE_t, then $\neg A$, $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$, $(A \leftrightarrow B)$ are in WE_t;

(iv) If A is in WE_t and x is a variable of arbitrary type, then $\forall xA$ and $\exists xA$ are in WE_t;

(v) If α, β are well-formed expressions of the same type, then $\alpha = \beta \in WE_t$;

(vi) Nothing else is a well-formed expression.

Recap: Type Theory —Function application

(ii) If $\alpha \in WE_{\langle\sigma,\tau\rangle}$, *and* $\beta \in WE_{\sigma}$, *then* $\alpha(\beta) \in WE_{\tau}$

"John is a talented piano player"

$$\dfrac{\text{john} :: e \qquad \dfrac{\text{piano\_player} :: \langle e, t\rangle \qquad \text{talented} :: \langle\langle e, t\rangle, \langle e, t\rangle\rangle}{\text{talented(piano\_player)} :: \langle e, t\rangle}}{\text{talented(piano\_player)(john)} :: t}$$

Recap: Type Theory — Semantics

Interpretation relative to a model structure **M = ⟨U, V⟩** and an assignment function **g**, where:

- **U** is a non-empty set of entities and **V** is an interpretation function, which assigns to every $\boldsymbol{\alpha} \in \mathbf{CON}_\tau$ an element of $\mathbf{D}_\tau$

- **g** assigns to every typed variable $\mathbf{v} \in \mathbf{VAR}_\tau$ an element of $\mathbf{D}_\tau$

The domain of possible denotations $\mathbf{D}_\tau$ for every type $\boldsymbol{\tau}$ is given by:

- $D_e = U$

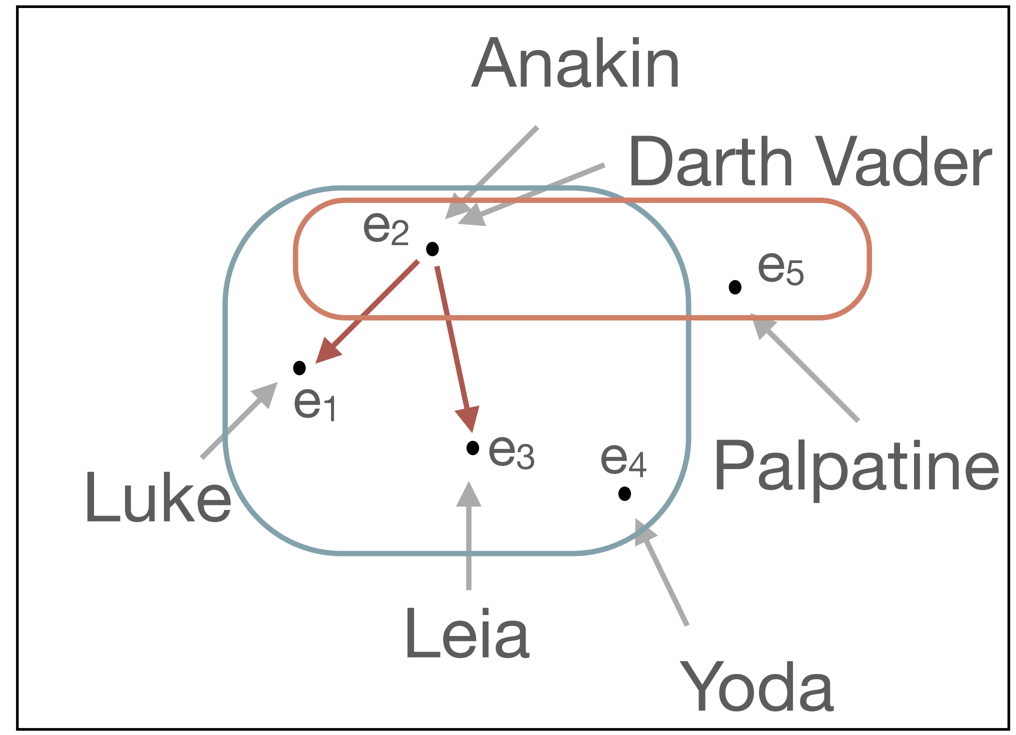
- $D_t = \{0,1\}$

- $D_{\langle\sigma,\tau\rangle}$ is the set of all functions from D_σ to D_τ

Recap: Type Theory — Model

Consider the following Model M:

M:



$D_e = U_M = \{e_1, e_2, e_3, e_4, e_5\}$

$V_M(\text{Anakin}_e) = V_M(\text{Darth Vader}_e) = e_2$

$V_M(\text{Yedi}_{\langle e,t\rangle}) = \begin{bmatrix} e_1 \rightarrow 1 \\ e_2 \rightarrow 1 \\ e_3 \rightarrow 1 \\ e_4 \rightarrow 1 \\ e_5 \rightarrow 0 \end{bmatrix}$ $V_M(\text{Dark\_Sider}_{\langle e,t\rangle}) = \begin{bmatrix} e_1 \rightarrow 0 \\ e_2 \rightarrow 1 \\ e_3 \rightarrow 0 \\ e_4 \rightarrow 0 \\ e_5 \rightarrow 1 \end{bmatrix}$

$$V_M(\text{Powerful}_{\langle\langle e,t\rangle\langle e,t\rangle\rangle}) = \begin{bmatrix} \begin{bmatrix} e_1 \rightarrow 1 \\ e_2 \rightarrow 1 \\ e_3 \rightarrow 1 \\ e_4 \rightarrow 1 \\ e_5 \rightarrow 0 \end{bmatrix} \rightarrow \begin{bmatrix} e_1 \rightarrow 0 \\ e_2 \rightarrow 1 \\ e_3 \rightarrow 0 \\ e_4 \rightarrow 1 \\ e_5 \rightarrow 0 \end{bmatrix} \\ \begin{bmatrix} e_1 \rightarrow 0 \\ e_2 \rightarrow 1 \\ e_3 \rightarrow 0 \\ e_4 \rightarrow 0 \\ e_5 \rightarrow 1 \end{bmatrix} \rightarrow \begin{bmatrix} e_1 \rightarrow 0 \\ e_2 \rightarrow 1 \\ e_3 \rightarrow 0 \\ e_4 \rightarrow 0 \\ e_5 \rightarrow 1 \end{bmatrix} \\ \ldots \end{bmatrix}$$

Recap: Type Theory — Interpretation

Given a model structure M = ⟨U, V⟩ and a variable assignment g:

$[\![\alpha]\!]^{M,g}$ = V(α) if α is a constant
= g(α) if α is a variable

$[\![\alpha(\beta)]\!]^{M,g} = [\![\alpha]\!]^{M,g}([\![\beta]\!]^{M,g})$

$[\![\alpha = \beta]\!]^{M,g}$ = 1 iff $[\![\alpha]\!]^{M,g} = [\![\beta]\!]^{M,g}$

$[\![\neg\phi]\!]^{M,g}$ = 1 iff $[\![\phi]\!]^{M,g} = 0$

$[\![\phi \wedge \psi]\!]^{M,g}$ = 1 iff $[\![\phi]\!]^{M,g}$ = 1 and $[\![\psi]\!]^{M,g}$ = 1

$[\![\phi \vee \psi]\!]^{M,g}$ = 1 iff $[\![\phi]\!]^{M,g}$ = 1 or $[\![\psi]\!]^{M,g}$ = 1

…

For any variable v of type σ:

$[\![\exists v\phi]\!]^{M,g}$ = 1 iff there is a d ∈ D_σ such that $[\![\phi]\!]^{M,g[v/d]} = 1$

$[\![\forall v\phi]\!]^{M,g}$ = 1 iff for all d ∈ D_σ : $[\![\phi]\!]^{M,g[v/d]} = 1$

Compositionality

The principle of compositionality: "The meaning of a complex expression is a function of the meanings of its parts and of the syntactic rules by which they are combined" (Partee et al.,1993)

Compositional semantics construction:

- compute meaning representations for sub-expressions
- combine them to obtain a meaning representation for a complex expression.

Problematic case: "Not smoking$_{\langle e,t\rangle}$ is healthy$_{\langle\langle e,t\rangle,t\rangle}$"

!

Lambda abstraction

λ-abstraction is an operation that takes an expression and "opens" specific argument positions.

Syntactic definition:

If α is in WE_τ, and x is in VAR_σ then $\lambda x(\alpha)$ is in $WE_{\langle\sigma, \tau\rangle}$

- The scope of the λ-operator is the smallest WE to its right. Wider scope must be indicated by brackets.
- We often use the "dot notation" $\lambda x.\phi$ indicating that the λ-operator takes widest possible scope (over ϕ).

Interpretation of Lambda-expressions

If $\alpha \in WE_\tau$ and $v \in VAR_\sigma$, then $[\![\lambda v\alpha]\!]^{M,g}$ is that function $f : D_\sigma \rightarrow D_\tau$ such that for all $a \in D_\sigma$, $f(a) = [\![\alpha]\!]^{M,g[v/a]}$

If the λ-expression is applied to some argument, we can simplify the interpretation:

- $[\![\lambda v\alpha]\!]^{M,g}(x) = [\![\alpha]\!]^{M,g[v/x]}$

Example: *"Bill is a non-smoker"*

$[\![\lambda x(\neg S(x))(b')]\!]^{M,g} = 1$

iff $[\![\lambda x(\neg S(x))]\!]^{M,g}([\![b']\!]^{M,g}) = 1$

iff $[\![\neg S(x)]\!]^{M,g[x/[\![b']\!]^{M,g}]} = 1$

iff $[\![S(x)]\!]^{M,g[x/[\![b']\!]^{M,g}]} = 0$

iff $[\![S]\!]^{M,g[x/[\![b']\!]^{M,g}]}([\![x]\!]^{M,g[x/[\![b']\!]^{M,g}]}) = 0$

iff $V_M(S)(V_M(b')) = 0$

β-Reduction

$[\![\lambda v(\alpha)(\beta)]\!]^{M,g} = [\![\alpha]\!]^{M,g[v/[\![\beta]\!]^{M,g}]}$

⇒ all (free) occurrences of the λ-variable in α get the interpretation of β as value.

This operation is called β-reduction

- $\lambda v(\alpha)(\beta) \Leftrightarrow [\beta/v]\alpha$
- $[\beta/v]\alpha$ is the result of replacing all free occurrences of v in α with β.

Achtung: The equivalence is not unconditionally valid!

Variable capturing

Q: Are $\lambda v(\alpha)(\beta)$ and $[\beta/v]\alpha$ always equivalent?

- $\lambda x(\text{drive'}(x) \wedge \text{drink'}(x))(j') \Leftrightarrow \text{drive'}(j') \wedge \text{drink'}(j')$
- $\lambda x(\text{drive'}(x) \wedge \text{drink'}(x))(y) \Leftrightarrow \text{drive'}(y) \wedge \text{drink'}(y)$
- $\lambda x(\forall y\ \text{know'}(x)(y))(j') \Leftrightarrow \forall y\ \text{know}(j')(y)$
- NOT: $\lambda x(\forall y\ \text{know'}(x)(y))(y) \Leftrightarrow \forall y\ \text{know}(y)(y)$

Let v, v' be variables of the same type, and let α be any well-formed expression.

- v is free for v' in α iff no free occurrence of v' in α is in the scope of a quantifier or a λ-operator that binds v.

Conversion rules

- β-conversion: $\lambda v(\alpha)(\beta) \Leftrightarrow [\beta/v]\alpha$

  (if all free variables in β are free for v in α)

- α-conversion: $\lambda v\alpha \Leftrightarrow \lambda w[w/v]\alpha$

  (if w is free for v in α)

- η-conversion: $\lambda v(\alpha(v)) \Leftrightarrow \alpha$

Determiners as lambda-expressions

- a student works ➔ $\exists x(\text{student'}(x) \wedge \text{work'}(x)) :: t$
  - a student ➔ $\lambda P \exists x(\text{student'}(x) \wedge P(x)) :: \langle\langle e,t\rangle,t\rangle$
  - a, some ➔ $\lambda Q \lambda P \exists x(Q(x) \wedge P(x)) :: \langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle$
- every student ➔ $\lambda P \forall x(\text{student'}(x) \rightarrow P(x)) :: \langle\langle e,t\rangle,t\rangle$
  - every ➔ $\lambda Q \lambda P \forall x(Q(x) \rightarrow P(x)) :: \langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle$
- no student ➔ $\lambda P \neg \exists x(\text{student}(x) \wedge P(x)) :: \langle\langle e,t\rangle,t\rangle$
  - no ➔ $\lambda Q \lambda P \neg \exists x(Q(x) \wedge P(x)) :: \langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle$
- someone ➔ $\lambda F \exists x F(x) :: \langle\langle e,t\rangle,t\rangle$

NL Quantifier Expressions: Interpretation

- someone'$\in$ $CON_{\langle\langle e,t\rangle,t\rangle}$, so V_M(someone') $\in$ $D_{\langle\langle e,t\rangle,t\rangle}$
- $D_{\langle\langle e,t\rangle,t\rangle}$ is the set of functions from $D_{\langle e,t\rangle}$ to D_t , i.e.,
  the set of functions from $\mathcal{P}(U_M)$ to {0,1},
  which in turn is equivalent to $\mathcal{P}(\mathcal{P}(U_M))$.
- Thus, V_M(someone') $\subseteq$ $\mathcal{P}(U_M)$. More specifically:
- V_M(someone') = $\{S \subseteq U_M \mid S \neq \varnothing\}$, if U_M is a domain of persons

β-Reduction Example

Every student works.

(2) $\lambda P \lambda Q \forall x(P(x) \rightarrow Q(x)) : \langle\langle e, t\rangle, \langle\langle e, t\rangle, t\rangle$

(3) $student' : \langle e, t\rangle$

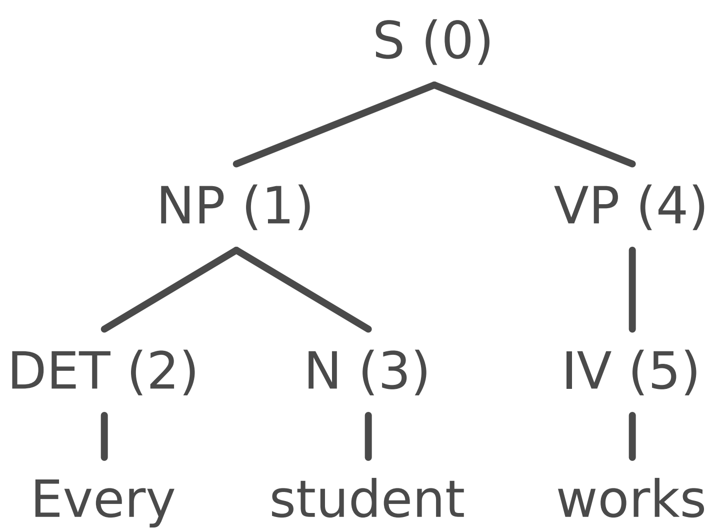
(1) $\lambda P \lambda Q \forall x(P(x) \rightarrow Q(x))(student')$

$\Leftrightarrow^{\beta} \lambda Q \forall x(student'(x) \rightarrow Q(x)) : \langle\langle e, t\rangle, t\rangle$

(4)/(5) $work' : \langle e, t\rangle$

(0) $\lambda Q \forall x(student'(x) \rightarrow Q(x))(work')$

$\Leftrightarrow^{\beta} \forall x(student'(x) \rightarrow work'(x)) : t$



Transitive Verbs: Type Clash

- Someone reads a book

$$\dfrac{\text{someone} :: \langle\langle e, t\rangle,t\rangle \qquad \dfrac{\text{read} :: \langle e,\langle e, t\rangle\rangle \qquad \text{a book} :: \langle\langle e, t\rangle,t\rangle}{?? :: ??}}{?? :: t}$$

Solution: reverse functor-argument relation (again)

$\text{read}_{\langle\langle\langle e,\ t\rangle,t\rangle,\langle e,\ t\rangle\rangle}$ *(Type Raising)*

Type Raising

It's not enough to just change the type of the transitive verb:

- read ➔ read' $\in$ $\mathrm{CON}_{\langle\langle\langle e,t\rangle, t\rangle, \langle e, t\rangle\rangle}$

  someone reads a book:
  $\lambda F \exists x F(x)(\mathrm{read}'(\lambda P \exists y(\mathrm{book}'(y) \wedge P(y)))$
  $\Leftrightarrow^{\beta} \exists x(\mathrm{read}'(\lambda P \exists y(\mathrm{book}'(y) \wedge P(y)))(x)$

  …but this does not support the following entailment:
  someone reads a book $\vDash$ *there exists a book*

We need a more explicit λ-term:

- read ➔ $\lambda Q \lambda z.Q(\lambda x(\mathrm{read}^*(x)(z)))$ $\in$ $\mathrm{WE}_{\langle\langle\langle e,t\rangle, t\rangle, \langle e, t\rangle\rangle}$
  where: read* $\in$ $\mathrm{WE}_{\langle e, \langle e, t\rangle\rangle}$ is the "underlying" first-order relation

Transitive Verbs: example

someone reads a book

$\lambda F \exists x F(x)(\lambda Q \lambda z.Q(\lambda x(read^*(x)(z)))(\lambda R \lambda P.\exists y(R(y) \wedge P(y))\ (book')))$

$\Leftrightarrow_\beta \lambda F \exists x F(x)(\lambda Q \lambda z.Q(\lambda x(read^*(x)(z)))(\lambda P.\exists y(book'(y) \wedge P(y))))$

$\Leftrightarrow_\beta \lambda F \exists x F(x)(\lambda z.(\lambda P.\exists y(book'(y) \wedge P(y)))(\lambda x(read^*(x)(z))))$

$\Leftrightarrow_\beta \lambda F \exists x F(x)(\lambda z.\exists y(book'(y) \wedge \lambda x(read^*(x)(z))(y)))$

$\Leftrightarrow_\beta \lambda F \exists x F(x)(\lambda z.\exists y(book'(y) \wedge read^*(y)(z)))$

$\Leftrightarrow_\beta \exists x(\lambda z.\exists y(book'(y) \wedge read^*(y)(z)))(x)$

$\Leftrightarrow_\beta \exists x \exists y(book'(y) \wedge read^*(y)(x))$

Type inferencing examples: revisited

6. Yoda$_e$ <u>encouraged</u> Obi-Wan$_e$ to take$_{\langle e,\langle e,t\rangle\rangle}$ the exam$_e$.

   LF1: encourage(o)(T(e))(y*)
   encourage$_{\langle e,\langle\langle e,t\rangle,\langle e,t\rangle\rangle\rangle}$ = λxλPλy(encourage(x)(P)(y))

   LF2: encourage(o)(T(e)(o))(y*)
   encourage$_{\langle e,\langle\langle e,t\rangle,\langle e,t\rangle\rangle\rangle}$ = λxλPλy(encourage(x)(P(x))(y))

We could take a similar approach for *expects* in:

5. Obi-Wan$_e$ <u>expects</u> to pass$_{\langle e,t\rangle}$.

Background reading material

- Gamut: Logic, Language, and Meaning Vol II
    - Chapter 4 (minus 4.3)