Semantic Theory
Semantics Construction

Manfred Pinkal
Stefan Thater

2008-05-06

Last Week: Type Theory

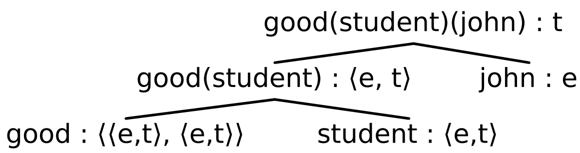
- Expressive limits of first-order logic (FOL)
  - *John is a blond / good / alleged thief*
  - *Mary has all properties of a good student*
- Solution: Generalise FOL to Type Theory
  - basic types: e, t
  - functional types ⟨σ, τ⟩
  - build logic from functional application and the usual logical connectives (over higher-order constants and variables).

Type Theory – Syntax

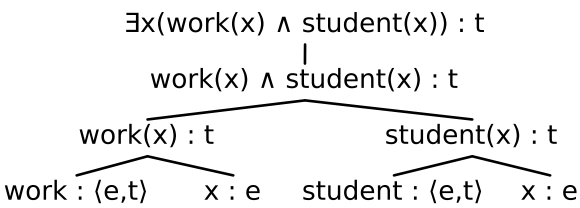
- The sets of well-formed expressions WE_τ for every type τ are given by:
  (1) $CON_\tau \subseteq WE_\tau$, for every type τ
  (2) If α is in $WE_{\langle\sigma,\,\tau\rangle}$, β in WE_σ, then $\alpha(\beta) \in WE_\tau$.
  (3) If A, B are in WE_t, then $\neg A$, $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$, $(A \leftrightarrow B)$ are in WE_t.
  (4) If A is in WE_t, then $\forall vA$ and $\exists vA$ are in WE_t, where v is a variable of arbitrary type.
  (5) If α, β are well-formed expressions of the same type, then $\alpha = \beta \in WE_t$.

Building well-formed expressions

- *John is a good student*



- *A student works*



Type Theory – Semantics [1/3]

- Let U be a non-empty set of entities.
- The domain of possible denotations D_τ for every type τ is given by:
  (1) $D_e = U$
  (2) $D_t = \{0,1\}$
  (3) $D_{(\sigma,\tau)}$ is the set of all functions from D_σ to D_τ

Type Theory – Semantics [2/3]

- A model structure for a type theoretic language consists of a pair $M = (U, V)$, where
  - U (or U_M) is a non-empty domain of individuals
  - V (or V_M) is an interpretation function, which assigns to every member of CON_τ an element of D_τ.

- Variable assignment g assigns every variable of type τ a member of D_τ.

Type Theory – Semantics [3/3]

- Interpretation with respect to model structure M and variable assignment g:

$$[\![\alpha]\!]^{M,g} = V_M(\alpha), \text{ if } \alpha \text{ constant}$$
$$[\![\alpha]\!]^{M,g} = g(\alpha), \text{ if } \alpha \text{ variable}$$
$$[\![\alpha(\beta)]\!]^{M,g} = [\![\alpha]\!]^{M,g}([\![\beta]\!]^{M,g})$$
$$[\![\neg\phi]\!]^{M,g} = 1 \text{ iff } [\![\phi]\!]^{M,g} = 0$$
$$[\![\phi \wedge \psi]\!]^{M,g} = 1 \text{ iff } [\![\phi]\!]^{M,g} = 1 \text{ and } [\![\psi]\!]^{M,g} = 1, \text{ etc.}$$
$$[\![\alpha = \beta]\!]^{M,g} = 1 \text{ iff } [\![\alpha]\!]^{M,g} = [\![\beta]\!]^{M,g}$$

- if $v \in VAR\tau$:

$$[\![\exists v\phi]\!]^{M,g} = 1 \text{ iff there is } d \in D_\tau \text{ such that } [\![\phi]\!]^{M,g[v/d]} = 1$$
$$[\![\forall v\phi]\!]^{M,g} = 1 \text{ iff for all } d \in D_\tau : [\![\phi]\!]^{M,g[v/d]} = 1$$

Today: Semantics Construction

- Elementary semantics construction:
  - the principle of compositionality
  - compositional semantics construction using type theory
- Quantified noun phrases
- The lambda operator in type theory

The Principle of Compositionality

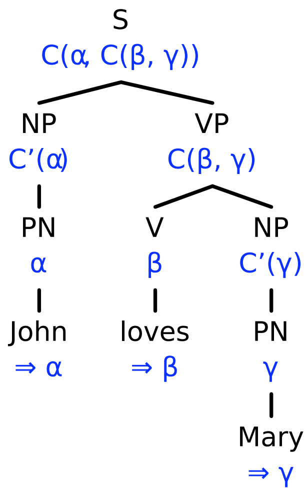
- The meaning of a complex expression is uniquely determined by the meanings of its sub-expressions and the syntactic rules by which they are combined.
- (The principle is also called “Frege’s principle”)

Two Levels of Interpretation

- Semantic interpretation is a two-step process
  - Natural language (NL) expressions are assigned a semantic representation (logical formulas).
  - The semantic representation is truth-conditionally interpreted.
- Truth-conditional interpretation of logical representations is strictly compositional.
- We also want this for the process of computing logical representations from NL expressions.

Compositional Semantics Construction

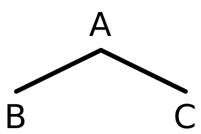
- Basic idea: we start with a syntactic analysis of an NL expression, and
- assign each syntactic node in the syntax tree a semantic representation
- by combining the representations of its daughter nodes.



Basic Composition Rules

- Rule of functional application

$$\frac{B \Rightarrow \beta : \langle \sigma,\tau \rangle \qquad C \Rightarrow \gamma : \sigma}{A \Rightarrow \beta(\gamma) : \tau} \quad \text{or} \quad \frac{B \Rightarrow \beta : \sigma \qquad C \Rightarrow \gamma : \langle \sigma,\tau \rangle}{A \Rightarrow \gamma(\beta) : \tau}$$



- Rule for non-branching nodes

$$\frac{B \Rightarrow \beta : \tau}{A \Rightarrow \beta : \tau}$$

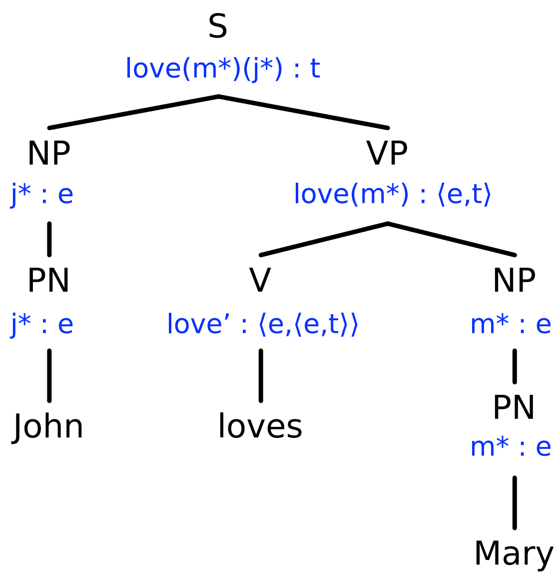
Basic Composition Rules

- Rule for lexical nodes:

$$\frac{}{A \Rightarrow \beta : \tau}$$

- The semantic representation β for a word w is supplied by the lexicon.

An Example



Noun phrases and compositionality

John works ⇒ work'(j*)

Somebody works ⇒ ∃x(work'(x))

Every student works ⇒ ∀x(student'(x) → work'(x))

No student works ⇒ ¬∃x(student'(x) ∧ work'(x))

John and Mary work ⇒ work'(j*) ∧ work(m*)

- What's the semantic representation of a noun phrase?

Towards a unified semantics of Noun Phrases

- *John works*

$$\frac{j^* : e \qquad \text{work}' : \langle e,t\rangle}{\text{work}'(j^*) : t}$$

- *Every student works*

$$\frac{\text{every-student}' : \langle\langle e,t\rangle,t\rangle \qquad \text{work}' : \langle e,t\rangle}{\text{every-student}'(\text{work}') : t}$$

Towards a unified semantics of Noun Phrases

- *John works*

$$\frac{\text{john}' : \langle\langle e,t\rangle,t\rangle \qquad \text{work}' : \langle e,t\rangle}{\text{john}'(\text{work}') : t}$$

- *Every student works*

$$\frac{\text{every-student}' : \langle\langle e,t\rangle,t\rangle \quad \text{work}' : \langle e,t\rangle}{\text{every-student}'(\text{work}') : t}$$

A Coverage Problem

- *Swimming is healthy*

$$\frac{\text{swim}' : \langle e,t\rangle \qquad \text{healthy}' : \langle\langle e,t\rangle,t\rangle}{\text{healthy}'(\text{swim}') : t}$$

- *Not smoking is healthy*
- *Drinking and driving is dangerous*

Summing up

- We have the following kinds of problems:
  - We want uniform semantic representations for noun phrases, and we don't seem to have the syntax to write them down.
  - Some natural language expressions seem to require us to say "an x with property P."
- Solution: λ-abstraction

λ-Abstraction

- Syntax:
  - If $\alpha \in WE_\tau$ and $v \in VAR_\sigma$, then $\lambda v\alpha \in WE_{(\sigma,\tau)}$.
- Example:
  - $\lambda x(drive(x) \wedge drink(x))$
- Notational conventions:
  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" $\lambda x. \ldots$ indicating that the λ-operator takes widest possible scope.

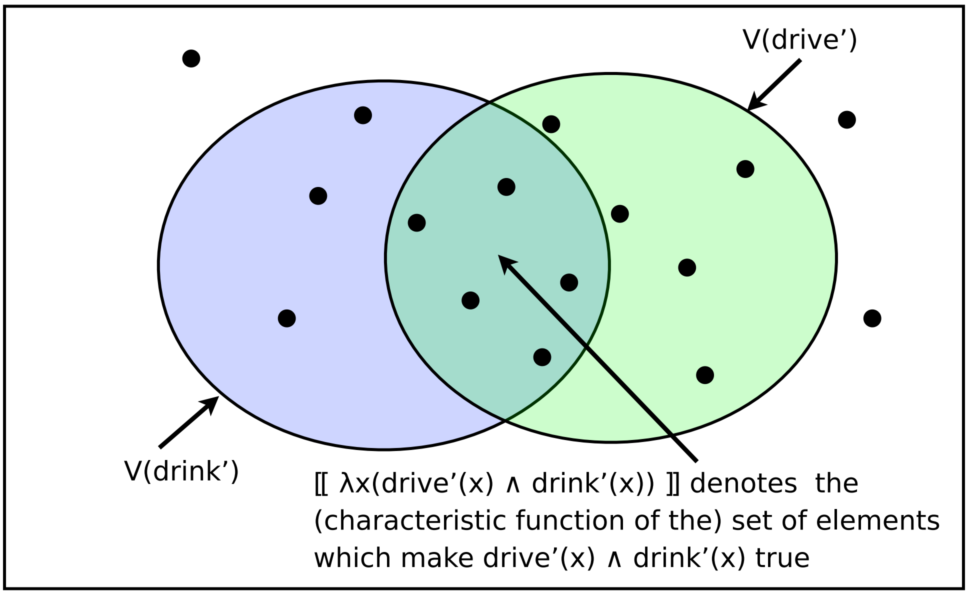
λ-Abstraction

- $\lambda x[\mathrm{drive}(x) \wedge \mathrm{drink}(x)]$
- … a term of type $\langle e,t \rangle$
- … denotes the property of being "an x such that x drives and drinks"
- λ-abstraction is an operation that takes an expression and "opens" a specific argument positions. The result of abstraction over individual variable x in the formula $\mathrm{drive}(x) \wedge \mathrm{drink}(x)$ results in the complex predicate $\lambda x[\mathrm{drive}(x) \wedge \mathrm{drink}(x)]$.

λ-Abstraction: Semantics

- $[\![\lambda v \alpha]\!]^{M,g}$ is that function $f : D_\sigma \rightarrow D_\tau$ such that for all $a \in D_\sigma$, $f(a) = [\![\alpha]\!]^{M,g[v/a]}$ (for $\alpha \in WE_\tau$, $v \in VAR_\sigma$)
- Notice that of course $f \in D_{\langle \sigma,\tau \rangle}$.
- In general:
  $[\![(\lambda v \alpha)(\beta)]\!]^{M,g} = [\![\alpha]\!]^{M,g[v / [\![\beta]\!]^{M,g}]}$

λx(drive'(x) ∧ drink'(x))



Back to the Coverage Problem

- *Not smoking is healthy*
  - *healthy*(λx.¬smoke(x))
- *Drinking and driving is unwise*
  - ¬*wise*(λx.drink(x) ∧ drive(x))

β-Reduction

- By the modified variable assignment, the value of the argument of the λ-expression is passed through its body and becomes the value of all occurrences of variables bound by the λ-operator.
- We obtain the same result, if we first substitute the free occurrences of the λ-variable in λvα(β) by the argument β, and only then interpret the result:
  - $[\![\lambda v \alpha(\beta)]\!]^{M,g} = [\![\alpha]\!]^{M,g[v/ [\![\beta]\!]M,g]}$ to
  - $[\![\lambda v \alpha(\beta)]\!]^{M,g} = [\![[\beta/v]\alpha]\!]^{M,g}$
- This is the basic idea behind the λ-calculus.

Variable capturing

- Are λvα(β) and [β/v]α always equivalent?
  - λx[drive'(x) ∧ drink'(x)](j*) ⇒ drive'(j*) ∧ drink'(j*)
  - λx[drive'(x) ∧ drink'(x)](y) ⇒ drive'(y) ∧ drink'(y)
  - λx[∀y know'(x)(y)](j*) ⇒ ∀y know(j*)(y)
  - λx[∀y know'(x)(y)](y) ⇏ ∀y know(y)(y)
- Let v, v' be variables of the same type, α 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 in the λ-calculus

- β-conversion:
  λvα(β) ⇔ [β/v]α if all free variables in β are free for v in α.
- α-conversion:
  λvα ⇔ λv'[v'/v]α if v' is free for v in α.
- η-conversion:
  λv(α(v)) ⇔ α
- The rule which we will use most in semantics construction is β-conversion in the left-to-right direction (β-reduction), which allows us to simplify representations.

An Example

- *John drives and drinks.*

$$\cfrac{\cfrac{\cfrac{\cfrac{\text{drive}' : \langle e,t\rangle \quad x : e}{\text{drive}'(x) : t} \qquad \cfrac{\text{drink}' : \langle e,t\rangle \quad x : e}{\text{drink}'(x) : t}}{\text{drive}'(x) \wedge \text{drink}'(x) : t}}{\lambda x\,(\text{drive}'(x) \wedge \text{drink}'(x)) : \langle e,t\rangle} \qquad j^* : e}{\lambda x\,(\text{drive}'(x) \wedge \text{drink}'(x))\,(j^*)}$$

$$\Rightarrow_\beta \text{drive}'(j^*) \wedge \text{drink}'(j^*)$$

Back to Noun Phrases

- We were looking for a uniform representation for noun phrases:
  - All noun phrases are uniformly represented as terms of type ⟨⟨e,t⟩,t⟩ i.e., expressions that denote sets of first-order properties (type ⟨e,t⟩).
  - Interpretation of “John:” the set of properties P such that John has property P.
  - Interpretation of “every student:” the set of properties P such that every student has P.
  - and so on …

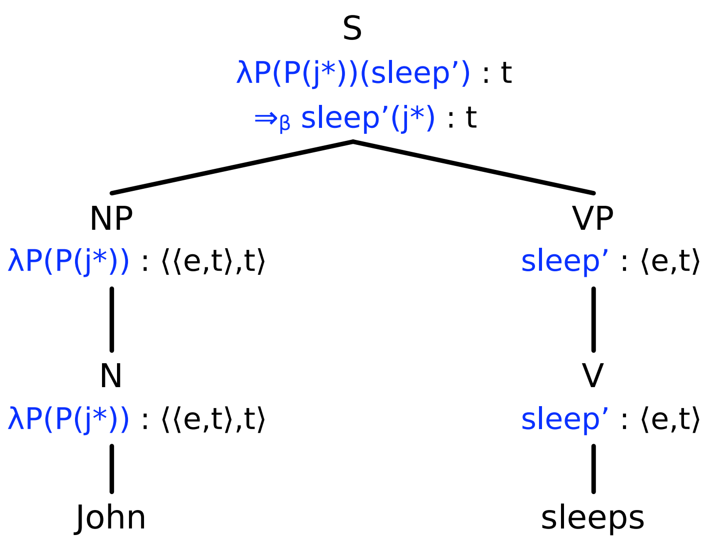
Back to Noun Phrases

- Interpretation of “John:” the set of properties P such that John has property P:
  - $\lambda P(P(j^*))$
- Interpretation of “every student:” P belongs to the set if every student has property P:
  - $\lambda P(\forall x(\text{student}'(x) \rightarrow P(x)))$
- Interpretation of “a student:” P belongs to the set if a student has property P:
  - $\lambda P(\exists x(\text{student}'(x) \wedge P(x)))$

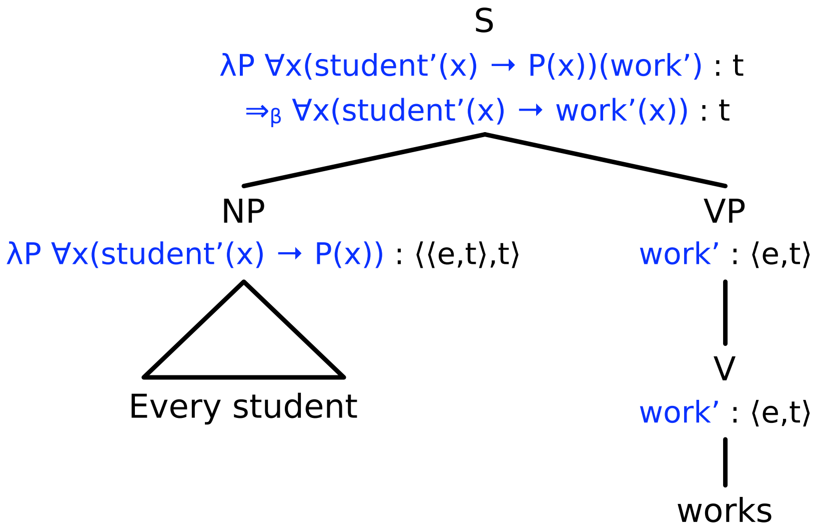
More Noun Phrases

John	$\Rightarrow \lambda G(G(j^*))$
Somebody	$\Rightarrow \lambda G\ \exists x G(x)$
A student	$\Rightarrow \lambda G\ \exists x(student(x) \wedge G(x))$
No student	$\Rightarrow \lambda G\ \neg\exists x(student(x) \wedge G(x))$
John	$\Rightarrow \lambda G(G(j^*))$
John and Mary	$\Rightarrow \lambda G(G(j^*) \wedge G(m^*))$

“John sleeps”



“Every student works”



Determiners

a, some $\Rightarrow \lambda F \lambda G\ \exists x(F(x) \wedge G(x))$

every $\Rightarrow \lambda F \lambda G\ \forall x(F(x) \rightarrow G(x))$

no $\Rightarrow \lambda F \lambda G\ \neg\exists x(F(x) \wedge G(x))$

most $\Rightarrow$ most’ (a constant)

“Every student works.”