Semantic Theory

Semantics Construction

Manfred Pinkal
Stefan Thater

Summer 2007

Outline

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

The Principle of Compositionality

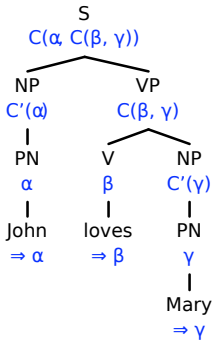
- The meaning of a complex expression is uniquely determined by the meanings of its sub-expressions and the syntactic rules by which they are combined.
- (The principle is also called “Frege’s principle”)

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

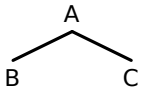
- Basic idea: we start with a syntactic analysis of an NL expression, and
- assign each syntactic node in the syntax tree a semantic representation
- by combining the representations of its daughter nodes.



Basic Rules

- Rule of functional application

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



- Rule for non-branching nodes

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

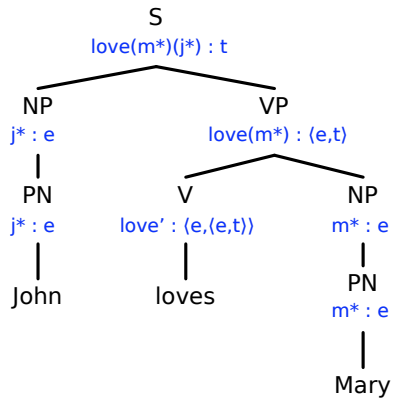
Basic Rules

- Rule of functional application

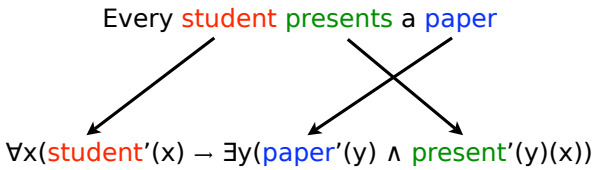
$$\frac{}{A \Rightarrow \beta : \tau}$$

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

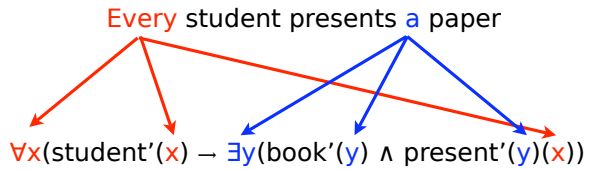
An Example



Noun phrases and compositionality



Noun phrases and compositionality



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 NPs

- "John works."

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

- "Every student works."

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

Towards a unified semantics of NPs

- "John works."

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

- "Every student works."

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

A coverage problem

- "Swimming is healthy."

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

- But …
  - "Not smoking is healthy"
  - "Drinking and driving is dangerous"
  - "Some people drink and drive"

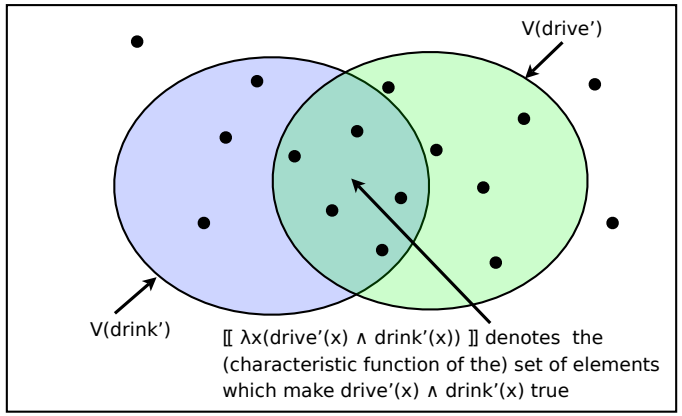
Summing up

- 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

- λx(drive(x) ∧ drink(x))
  - a term of type ⟨e,t⟩
  - denotes the property of being "an x such that x drives and drinks"
- λ-abstraction is an operation that takes an expression and "opens" or "re-opens" specific argument positions by abstracting over a variable
- The result of abstraction over individual variable x in the formula 'drive(x) ∧ drink(x)' results in the complex predicate 'λx(drive(x) ∧ drink(x)).'

λx(drive'(x) ∧ drink'(x))



Type Theory: Syntax

- The sets of well-formed expressions WE_τ for every type τ are given by:
  - $CON_\tau \cup VAR_\tau \subseteq WE_\tau$ for every type τ
  - If $\alpha \in WE_{\langle\sigma, \tau\rangle}$, $\beta \in WE_\sigma$, then $\alpha(\beta) \in WE_\tau$.
  - …
  - If φ is in WE_t , then so are ∀vφ and ∃vφ, where v is a variable of arbitrary type.
  - If α is in WE_τ and v is a variable of type σ, then λvα is a well-formed expression of type ⟨σ, τ⟩.

Notational Conventions

- Upper-case letters are used for higher-order variables, lower-case letters for individual variables.
- The scope of the λ-operator is the smallest WE to its right. Wider scope must be indicated by brackets.
  - λx x(j*) stands for (λx x)(j*), not λx (x(j*))
- We sometimes use square brackets / dots to improve readability
  - λP[P(j*)]
  - λP.P(j*)

An Example

- "drive and drink"

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

λ-Abstraction: Semantics

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

Back to the coverage problem

- "Not smoking is healthy"
  - healthy'(λx¬swim(x))
- "Drinking and driving is dangerous"
  - dangerous'(λx(drive'(x) ∧ drink'(x)))
- "Some people drink and drive"
  - (not yet)

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."

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

$\Rightarrow_\beta$ drive'(j*) ∧ 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 P (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:
  - λP[P(j*)]
- Interpretation of "every student:" P belongs to the set if every student has property P:
  - λP[∀x(student'(x) → P(x))]
- Interpretation of "a student:" P belongs to the set if a student has property P:
  - λP[∃x(student'(x) ∧ P(x))]

More Noun Phrases

John ⇒ λG[G(j*)]

Somebody ⇒ λG ∃xG(x)

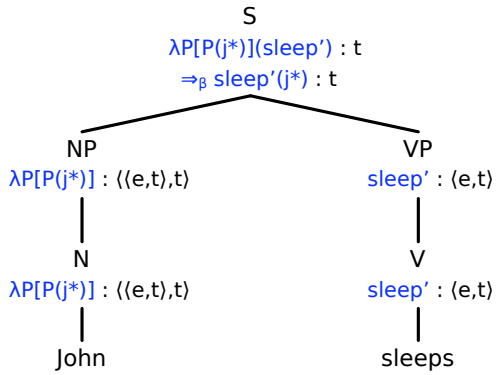
A student ⇒ λG ∃x(student(x) ∧ G(x))

No student ⇒ λG ¬∃x(student(x) ∧ G(x))

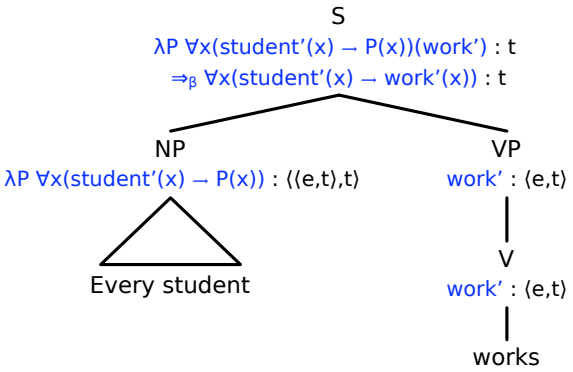
John ⇒ λG[G(j*)]

John and Mary ⇒ λG[G(j*) ∧ G(m*)]

"John sleeps"



"Every student works"



Determiners

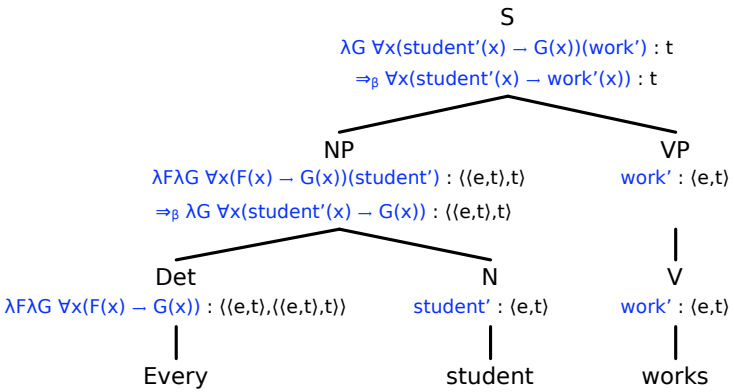
a, some ⇒ λFλG ∃x(F(x) ∧ G(x))

every ⇒ λFλG ∀x(F(x) → G(x))

no ⇒ λFλG ¬∃x(F(x) ∧ G(x))

most ⇒ most' (a constant)

“Every student works.”



Conclusion

- Semantics construction turned is not so easy for nontrivial sentences.
- With lambda-abstraction and application, these sentences can be treated in a straightforward way.
- Lambda abstraction is a very natural and straightforward extension to lambda-free type theory, and belongs to standard definitions of type theory.