Semantic Theory

Summer 2005

Type theory and λ-abstraction

M. Pinkal / A. Koller

Back to the Composition problem

Every student presented a paper

$\forall d\ (\text{student}(d) \rightarrow \exists p\ (\text{paper}(p) \wedge \text{present}(d,p)))$

Noun phrases and FOL representations

John works.	*work(john)*
Somebody works	$\exists$x (work(x))
Every student works	$\forall$x (student(x)$\rightarrow$ work(x))
No student works	$\neg \exists$x (student(x) $\wedge$ work(x))
John and Mary work	*work(john)* $\wedge$ work(mary)

A unified semantics for NPs? An attempt

John works.

$$\frac{\text{john: e} \qquad \text{work: <e,t>}}{\text{work(john): t}}$$

Every student works.

$$\frac{\text{every-student: e} \qquad \text{work: <e,t>}}{\text{work(every-student): t}}$$

?

A type-theoretic solution

Inverting the functor-argument relation by treating oun phrases as second-order predicates:

Every student works.

$$\frac{\text{every-student}: \langle\langle e,t\rangle,t\rangle \quad \text{work}: \langle e,t\rangle}{\text{every-student (work)}: t}$$

Internal NP structure

Determiners like *every/some/no* take a common-noun denotation and return a second-order predicate:

Determiners are functions from first-oder predicates to second-order predicates, i.e., two-place second-order relations:

$$\frac{\dfrac{\text{every}: \langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle \quad \text{student}: \langle e,t\rangle}{\text{every(student)}: \langle\langle e,t\rangle,t\rangle} \quad \text{work}: \langle e,t\rangle}{\text{every(student)(work)}: t}$$

Towards a unified semantics of NPs

John works.

john: e work: <e,t>

work(john): t

Every student works.

every-student: <<e,t>,t> work: <e,t>

every-student (work): t

Towards a unified semantics of NPs

John works.

john: <<e,t>,t> work: <e,t>

john(work): t

Every student works.

every-student: <<e,t>,t> work: <e,t>

every-student(work): t

Towards a unified semantics of NPs

„Type raising" of proper names from e to <<e,t>,t>:

John is represented by a second-order predicate that denotes a function from first-order predicates to truth-values which returns 1 for a predicate φ iff φ applies to the entity John.

Type theory as a semantic representation language:Two problems

- Problem 1:
  If we express quantification via second-order relations without quantifiers: How do we do inference?
- Problem 2:
  Even (basic) type theory has problems with coverage

Another coverage problem

Swimming is healthy

swimming: <e,t> healthy <<e,t>,t>

healthy(swimming): t

Not smoking is healthy

Driving and drinking is dangerous

John drives and drinks

Some people drive and drink

The solution: λ-abstraction

$\lambda x[drive(x) \wedge drink(x)]$

„to be 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.

E.g., the result of abstraction over individual variable x in the formula $drive(x) \wedge drink(x)$ results in the complex predicate $\lambda x[drive(x) \wedge drink(x)]$.

Syntax of λ-abstraction

If $\alpha \in WE_\tau$, $v \in Var_\sigma$, then $\lambda v\alpha \in WE_{<\sigma,\tau>}$.

Note: The scope of the λ-operator is the smallest WE to its right. Wider scope must be indicated by brackets.

Example

$$\dfrac{\dfrac{\text{drive: <e,t>} \quad \text{x:e}}{\text{drive(x): t}} \qquad \dfrac{\text{drink: <e,t>} \quad \text{x:e}}{\text{drink(x): t}}}{\dfrac{\text{drive(x)} \wedge \text{drink(x): t}}{\lambda \text{x[drive(x)} \wedge \text{drink(x)]: <e,t>}}}$$

Example

Some people drive and drink

drive: <e,t> x:e   drink: <e,t> x:e

drive(x): t   drink(x): t

some:<<e,t>,<<e,t>,t>> people: <e,t>   drive(x)∧drink(x): t

some(people):<<e,t>,t>   λx[drive(x)∧drink(x)]: <e,t>

some(people)(λx[drive(x)∧drink(x)]):t

Semantics of λ-expressions

- If $\alpha \in WE_\tau$, $v \in Var_\sigma$, then $[[\lambda v\alpha]]^{M,g}$ is that function $\varphi \in D_{<\sigma,\tau>}$, such that for all $a \in D_\sigma$: $\varphi(a) = [[\alpha]]^{M,g[v/a]}$
- In general: $[[\lambda v\alpha(\beta)]]^{M,g} = [[\alpha]]^{M,g[v/\,[[\beta]]^{M,g}]}$
- The scope of the λ-operator is the smallest WE to its right. Wider scope must be indicated by brackets.

A syntactic shortcut for the evaluation of λ-expressions

- 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 $\lambda v\alpha(\beta)$ 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.

λ-conversion

- Are $\lambda v\alpha(\beta)$ and $[\beta/v]\alpha$ always equivalent?
  - $\lambda x[\text{drive}(x)\wedge\text{drink}(x)](\text{john}) = \text{drive}(\text{john})\wedge\text{drink}(\text{john})$
  - $\lambda x[\text{drive}(x)\wedge\text{drink}(x)](y) = \text{drive}(y)\wedge\text{drink}(y)$
  - $\lambda x\,[\forall y\ \text{know}(x)(y)]\ (\text{john}) = \forall y\ \text{know}(\text{john})(y)$
  - $\lambda x\,[\forall y\ \text{know}(x)(y)]\ (y) \neq \forall y\ \text{know}(y)(y)$
- Let v, v' be variables of identical 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:

  $\lambda v\alpha(\beta) \Leftrightarrow {}^{[\beta/v]}\alpha$, if all free variables in β are free for v in α.
- α-conversion:

  $\lambda v\alpha \Leftrightarrow \lambda v'^{[v'/v]}\alpha$, if v' is free for v in α.
- η-conversion:

  $\lambda v\alpha(v) \Leftrightarrow \alpha$

The relevant rule for semantic interpretation which we really need, is β-conversion in the left-to-right direction (β-reduction), which allows to simplify representations.

Type theory as a semantic representation language:Two problems

- Problem 1:

  If we express quantification via second-order relations without quantifiers: How do we do inference?
- Problem 2:

  Even (basic) type theory has problems with coverage

Noun phrases interpretation again

- *every student* denotes a second-order property (<<e,t>,t>) which holds of a (first-oder) property ω iff all students are in ω.
- This semantic information can be straightforwardly encoded as a lambda term:
  $\lambda G\ \forall x(\text{student}(x) \rightarrow G(x))$
- Accordingly, the determinator *every* can be represented as:
  $\lambda F \lambda G \forall x(F(x) \rightarrow G(x))$

More noun phrases

John	$\lambda G[G(j^*)]$
Somebody	$\lambda G\ \exists x G(x)$
A student	$\lambda G\ \exists x(\text{student}(x) \wedge G(x))$
No student	$\lambda G\ \neg \exists x\ (\text{student}(x) \wedge G(x))$
John	$\lambda G[G(j^*)]$
John and Mary *work*	$\lambda G[G(j^*) \wedge G(m^*)]$

More determinators

a, some	$\lambda F \lambda G\, \exists x(F(x) \wedge G(x))$
no	$\lambda F \lambda G\, \neg \exists x\, (F(x) \wedge G(x))$

An example

Every student works

$\underline{\lambda F \lambda G \forall x(F(x) \rightarrow G(x)) : \langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle \quad \text{student}: \langle e,t\rangle}$

$\underline{\lambda F \lambda G \forall x(F(x) \rightarrow G(x))(\text{student}): \langle\langle e,t\rangle,t\rangle}$

(by β-red.:) $\underline{\lambda G \forall x(\text{student}(x) \rightarrow G(x)): \langle\langle e,t\rangle,t\rangle \qquad \text{work}: \langle e,t\rangle}$

$\underline{\lambda G \forall x(\text{student}(x) \rightarrow G(x))(\text{work}): t}$

(by β-red.:) $\forall x(\text{student}(x) \rightarrow \text{work}(x)): t$