Semantic Theory
Lecture 3 – Semantics Construction

Manfred Pinkal & Stefan Thater
FR 4.7 Allgemeine Linguistik (Computerlinguistik)
Universität des Saarlandes

Summer 2012

First-order logic

- Formulas of first-order logic can talk about properties of and relations between individuals.
- Constants and variables denote individuals.
- Quantification is restricted to quantification over individuals.

Limits of first-order logic

- First-order logic is not expressive enough to capture the full range of meaning of natural language:
  - Modification (“good student”, “former professor”)
  - Sentence embedding verbs (“knows that ...”)
  - Higher order quantification (“have the same hair color”)
  - ...
- First-order logic does not support compositional semantics construction.

Limits of first-order logic

- **The principle of compositionality (recap):** *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* (cited from Partee &al., 1993)
- **Compositional semantics construction:**
  - compute meaning representations for sub-expressions.
  - combine them to obtain a meaning representation for a complex expression.
- *a man walks* ↦ ∃x(man’(x) ∧ walk’(x))
  - *a man* ↦ **(?)**
  - *walks* ↦ **(?)**

Type Theory

- The types of non-logical expressions provided by first-order logic are not sufficient to describe the semantic function of all natural language expressions.
- Type theory provides a much richer inventory of types: higher-order relations and functions of different kinds.

Types

- **Basic types:**
  - **e** – the type of individual terms (“**e**ntities”)
  - **t** – the type of formulas (“**t**ruth-values”)
- **Complex types:**
  - If σ, τ are types, then ⟨σ, τ⟩ is a type.
  - ⟨σ, τ⟩ is the type of functions mapping arguments of type σ to values of type τ.
- Types indicate, how many arguments a predicate has, and what types the arguments must have.

Types of Predicate Logic

- Individual constants and variables: e
- One-place predicates (sleep, walk, …)
  - ⟨e, t⟩
- Two-place predicates (read, admire, …)
  - ⟨e, ⟨e, t⟩⟩
- Three-place predicates (give, …)
  - ⟨e, ⟨e, ⟨e, t⟩⟩⟩

Type Theory – Vocabulary

- **Constants:** For every type τ a possibly empty set of non-logical constants CON_τ (pairwise disjoint)
- **Variables:** For every type τ an infinite set of variables VAR_τ (pairwise disjoint)
- **Logical symbols:** ∀, ∃, ∧, ∨, …
- **Brackets:** (,)

Type Theory – Syntax

- The sets of **well-formed expressions WE_τ** for every type τ are given by:

(i) $CON_\tau \subseteq WE_\tau$ and $VAR_\tau \subseteq WE_\tau$, for every type τ

(ii) If α is in $WE_{\langle\sigma, \tau\rangle}$, β in WE_σ, then $\alpha(\beta) \in WE_\tau$.

(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 v is a variable of arbitrary type, then $\forall vA$ and $\exists vA$ are in WE_t.

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

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:
  - $D_e = U$
  - $D_t = \{0, 1\}$
  - $D_{\langle\sigma, \tau\rangle}$ is the set of all functions from D_σ to D_τ
- Expressions of type τ denote elements of D_τ
- For instance
  - $\alpha \in WE_{\langle e, t\rangle}$ denotes a set of individual
  - $\alpha \in WE_{\langle\langle e, t\rangle, t\rangle}$ denotes a set of sets of individuals

Characteristic Functions

- Many natural language expression have a type ⟨σ, t⟩.
- ⟨σ, t⟩ the type of functions mapping elements of type σ to true or false.
- Such function are also known as **characteristic functions**, and can be thought of as subsets of D_σ.
- Example: “student” is a constant of type ⟨e, t⟩ and can be seen as characterising the set of students.

Characteristic Functions

- U = {a, b, c, d}
- X = {a, b}
- Characterisitic function f_X of X (over U):
  - $f_X(a) = 1$
  - $f_X(b) = 1$
  - $f_X(c) = 0$
  - $f_X(d) = 0$

$$\begin{bmatrix} a & \rightarrow 1 \\ b & \rightarrow 1 \\ c & \rightarrow 0 \\ d & \rightarrow 0 \end{bmatrix}$$

- More generally: For all a ∈ U, $f_X(a) = 1$ iff a ∈ X

Type Theory – Semantics [2/3]

- **A model structure** for a type theoretic language consists of a pair **M = ⟨U, V⟩**, where
  - U is a non-empty domain of individuals
  - V 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** a model structure **M = ⟨U, V⟩** and a variable assignment **g**:

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

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

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

(iii) $[\![\neg\varphi]\!]^{M,g} = 1$ iff $[\![\varphi]\!]^{M,g} = 0$

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

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

…

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

Type Theory – Semantics [3/3]

- **Interpretation with respect to** a model structure **M = ⟨U, V⟩** and a variable assignment **g**:

(v) $[\![\exists v\varphi]\!]^{M,g} = 1$ iff there is a $d \in D_\tau$ such that $[\![\varphi]\!]^{M,g[v/d]} = 1$

$[\![\forall v\varphi]\!]^{M,g} = 1$ iff for all $d \in D_\tau$: $[\![\varphi]\!]^{M,g[v/d]} = 1$

(where v is a variable of type τ)

Examples [⇒ whiteboard]

- *Bill reads a book*
  - $[\![\exists x(\text{book'}(x) \wedge \text{read'}(x)(b^*)]\!]^{M,g} = 1$ iff …
- *Bill is a good student*
  - $[\![\text{good'}(\text{student'})(b^*)]\!]^{M,g} = 1$ iff …

read' : ⟨e, ⟨e, t⟩⟩	b* : e
book' : ⟨e, t⟩	x : e
student' : ⟨e, t⟩	
good' : ⟨⟨e, t⟩, ⟨e, t⟩⟩	

Adjective Classes & Meaning Postulates

- **Natural language:**
  - Bill is a good student ⊨ Bill is a student
- **Type theory:**
  - $\text{good'}(\text{student'})(b^*) \not\models \text{student'}(b^*)$
- We need additional "meaning postulates" to get the intended entailment relations
- Meaning postulates are restrictions on models and constrain the possible meaning of certain words

Adjective Classes & Meaning Postulates

- **Intersective adjectives** ("blond")
  - $[\![\text{blond N}]\!] = [\![\text{blond}]\!] \cap [\![\text{N}]\!]$
  - Meaning postlate: $\forall G \forall x(\text{blond}(G)(x) \rightarrow (\text{blond}^*(x) \wedge G(x))$
  - Note: $\text{blond} \in WE_{\langle\langle e, t\rangle, \langle e, t\rangle\rangle}$, $\text{blond}^* \in WE_{\langle e,t\rangle}$
- **Subsective adjectives** ("good")
  - $[\![\text{good N}]\!] \subseteq [\![\text{N}]\!]$
  - Meaning postlate: $\forall G \forall x(\text{good}(G)(x) \rightarrow G(x))$
- **Privative adjectives** ("former")
  - $[\![\text{former N}]\!] \cap [\![\text{N}]\!] = \emptyset$
  - Meaning postlate: $\forall G \forall x(\text{former}(G)(x) \rightarrow \neg G(x))$

Semantics Construction

- **The principle of compositionality (recap):** 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 (cited from Partee &al.,1993)
- **Compositional semantics construction:**
  - compute meaning representations for sub-expressions
  - combine them to obtain a meaning representation for a complex expression.

A simple grammar

S → NP VP	PN → Bill
NP → PN	PN → Mary
VP → IV	IV → works
VP → TV NP	TV → likes

- *Bill works*
- *Bill likes Mary*
- …

Semantic lexicon

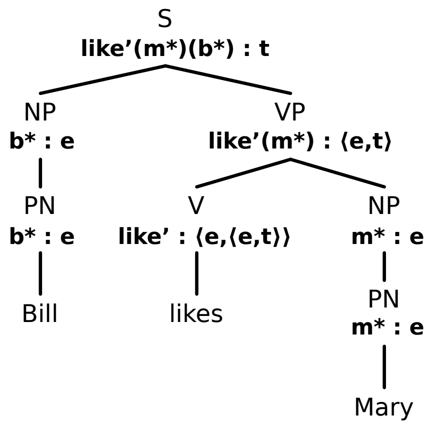
- Bill ↦ b* : e
- Mary ↦ m* : e
- likes ↦ like’ : ⟨e, ⟨e, t⟩⟩
- works ↦ work’ : ⟨e, t⟩

- read “↦” as “translates into”

Semantics Construction Rules (1st Version)

- **S → NP VP**
  if VP ↦ α’ and NP ↦ β’, then S ↦ α’(β’)
- **NP → PN**
  if PN ↦ α’, then NP ↦ α’
- **VP → IV**
  if IV ↦ α’, then VP ↦ α’
- **VP → TV NP**
  if TV ↦ α’ and NP ↦ β’, then VP ↦ α’(β’)

Bill likes Mary



Noun Phrases

John *works* ↦ work(j)

Somebody *works* ↦ **∃x**(work(**x**))

Every student *works* ↦ **∀x(student(x) →** work(**x**))

A student *works* ↦ **∃x(student(x) ∧** work(**x**))

No student *works* ↦ **¬∃x(student(x) ∧** work(**x**))

John and Mary *work* ↦ work(**j**) ∧ work(**m**)

λ-Abstraction

- **λx(drive(x) ∧ drink(x))**
  - a term of type ⟨e, t⟩
  - denotes the property (set of individuals) of being "an x such that x drives and drinks"
- λ-abstraction is an operation that takes an expression and "opens" specific argument positions.
- The result of abstraction over individual variable x in the formula "drive(x) ∧ drink(x)" results in the complex expression "λx(drive(x) ∧ drink(x))."

Type Theory with λ-Operator

- Syntax like basic type theory, plus:
  - **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⟩.**
- 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" **λx. …** indicating that the λ-operator takes widest possible scope.

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 $a \in D_\sigma$, $f(a) = [\![\alpha]\!]^{M,g[v/a]}$
- $[\![\lambda x(\text{drink}(x) \wedge \text{drive}(x))]\!]^{M,g} = \ldots$
  - [⇒ whiteboard]

λ-Abstraction – Semantics

- If $\alpha \in WE_\tau$, $v \in VAR_\sigma$, then $[\![\lambda v\alpha]\!]^{M,g}$ is that function $f : D_\sigma \to D_\tau$ 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:
  - $[\![\lambda v\alpha]\!]^{M,g}(A) = [\![\alpha]\!]^{M,g[v/A]}$
- $[\![\lambda x(drink(x) \wedge drive(x))(b^*)]\!]^{M,g} = 1$
  - iff $[\![\lambda x(drink(x) \wedge drive(x))]\!]^{M,g}([\![b^*]\!]^{M,g}) = 1$
  - iff $[\![\lambda x(drink(x) \wedge drive(x))]\!]^{M,g}(V_M(b^*)) = 1$
  - iff $[\![drink(x) \wedge drive(x)]\!]^{M,g[x/V(b^*)]} = 1$
  - iff $[\![drink(x)]\!]^{M,g[x/V(b^*)]}$ and $[\![drive(x)]\!]^{M,g[x/V(b^*)]} = 1$
  - iff $V_M(drink)(V_M(b^*)) = 1$ and $V_M(drive)(V_M(b^*)) = 1$

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.
- **Syntactic shortcut: 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

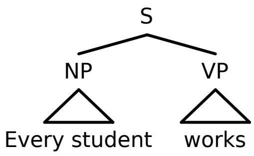
Variable capturing

- Are $\lambda v\alpha(\beta)$ and $[\beta/v]\alpha$ always equivalent?
  - $\lambda x[drive'(x) \wedge drink'(x)](j^*) \Leftrightarrow drive'(j^*) \wedge drink'(j^*)$
  - $\lambda x[drive'(x) \wedge drink'(x)](y) \Leftrightarrow drive'(y) \wedge drink'(y)$
  - $\lambda x[\forall y\, know'(x)(y)](j^*) \Leftrightarrow \forall y\, know(j^*)(y)$
  - **NOT:** $\lambda x[\forall y\, know'(x)(y)](y) \Leftrightarrow \forall y\, know(y)(y)$
- 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

- **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:** $\lambda v(\alpha(v)) \Leftrightarrow \alpha$

Back to noun phrases



- *Every student*
  - $\mapsto \lambda P\forall x(\text{student}'(x) \rightarrow P(x))$
  - Type: $\langle\langle e, t\rangle, t\rangle$
- Interpretation:
  - *Every student* denotes the set of properties that apply to every student (“property” = sets of individuals).
  - ⟦*Every student*⟧ = { P | every student has property P }
- Semantic construction rule for S → NP VP:
  - if VP $\mapsto \alpha'$ and NP $\mapsto \beta'$, then S $\mapsto \beta'(\alpha')$

Back to noun phrases

- Interpretation of “every student:” the set of properties P that apply to every student
  - *every student* $\mapsto \lambda P\forall x(\text{student}'(x) \rightarrow P(x))$
- Interpretation of “a student:” the set of properties P that apply to some student.
  - *a student* $\mapsto \lambda P\exists x(\text{student}'(x) \wedge P(x))$
- Interpretation of “Bill:” the set of properties P that apply to Bill.
  - *Bill* $\mapsto \lambda P.P(b^*)$

Determiners

- *a, some* $\mapsto \lambda F \lambda G \exists x(F(x) \wedge G(x))$
- *every* $\mapsto \lambda F \lambda G \forall x(F(x) \rightarrow G(x))$
- *no* $\mapsto \lambda F \lambda G \neg \exists x(F(x) \wedge G(x))$
- …

Semantics Construction Rules (2nd Version)

- **S → NP VP**
  if VP $\mapsto \alpha'$ and NP $\mapsto \beta'$, then S $\mapsto \beta'(\alpha')$
- **NP → DET N**
  if DET $\mapsto \alpha'$ and N $\mapsto \beta'$, then NP $\mapsto \alpha'(\beta')$
- **NP → PN**
  if PN $\mapsto \alpha'$, then NP $\mapsto \alpha'$
- **VP → IV**
  if IV $\mapsto \alpha'$, then VP $\mapsto \alpha'$

Every student works

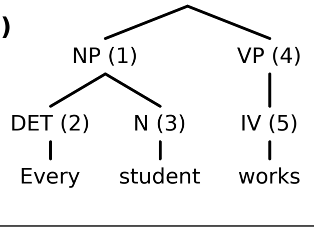
(2) $\mapsto \lambda P \lambda Q \forall x(P(x) \rightarrow Q(x)) : \langle\langle e, t\rangle, \langle\langle e, t\rangle, t\rangle$

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

(1) $\mapsto \lambda P \lambda Q \forall x(P(x) \rightarrow Q(x))(\text{student}')$: $\langle\langle e, t\rangle, t\rangle$
$\Rightarrow_\beta \lambda Q \forall x(\text{student}'(x) \rightarrow Q(x))$

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

(0) $\mapsto \lambda Q \forall x(\text{student}'(x) \rightarrow Q(x))(\text{work}')$: t
$\Rightarrow_\beta$ $\boldsymbol{\forall x(\text{student}'(x) \rightarrow \text{work}'(x))}$



Literature

- L.T.F. Gamut (1991): *Logic, Language and Meaning, Vol II.* University of Chicago Press. Chapter 4
- David Dowty, Robert Wall and Stanley Peters (1981): *Introduction to Montague Semantics.* Dordrecht, Reidel. Chapter 4.