Semantic Theory
Lecture 4: Type Theory 3

Manfred Pinkal
FR 4.7 Computational Linguistics and Phonetics

Summer 2014

UNIVERSITÄT
DES
SAARLANDES

Adjectives Again

- Bill is a poor piano player $\not\models$ Bill is poor
- Adjectives cannot be of type $\langle e, t\rangle$, but must analyzed as modifiers ($\langle\langle e, t\rangle, \langle e, t\rangle\rangle$), in the general case that map predicates onto predicates. Since the mapping is unrestricted, A+N constructions do not entail anything anymore. However, adjectives can be subdivided into different sub-classes with specific inferential properties:
- Bill is a poor piano player $\models$ Bill is a piano player
- Bill is a blond piano player $\models$ Bill is blond
- Bill is a former professor $\models$ Bill isn’t a professor

Adjective Classes

- **Restrictive or subsective adjectives** (“poor”)
  - V_M(poor)(S) ⊆ S, for all S⊆U_M
- **Privative adjectives** (“former”)
  - V_M(former)(S) ∩ S = ∅
- **Intersective adjectives** (“blond”)

  V_M(blond)(S) = S∩T for some specific first-order predicate N⊆U_M (i.e., the predicate denoting the blond persons)

Meaning Postulates

- Semantically appropriate type-theoretic model structures must observe the constraints for the respective adjective classes.

- The constraints can be represented as type-theoretic formulas:
  - $\forall G \forall x(\text{poor}(G)(x) \rightarrow G(x))$
  - $\forall G \forall x(\text{blond}(G)(x) \rightarrow (\text{blond*}(x) \wedge G(x))$

    Note: blond* $\in WE_{(e,t)}$ is used to denote the first-order predicate underlying the interpretation of *blond*
  - $\forall G \forall x(\text{former}(G)(x) \rightarrow \neg G(x))$

- These type-theoretic formulas are assumed to be generally valid axioms that constrain the set of possible model structures. Traditionally, they are are called **meaning postulates**.

The Principle of Compositionality

- The meaning of a complex expression is uniquely determined by the meaning of its parts and its syntactic structure.

  (Gottlob Frege, late 19th century)

- Practically realized as a two-step procedure:

  (1) Semantic Construction: Construct semantic representation φ from NL input sentence S.

  (2) Truth-Conditional Interpretation: Compute $[\![\varphi]\!]$ by recursive application of semantic interpretation rules.

Semantic Construction

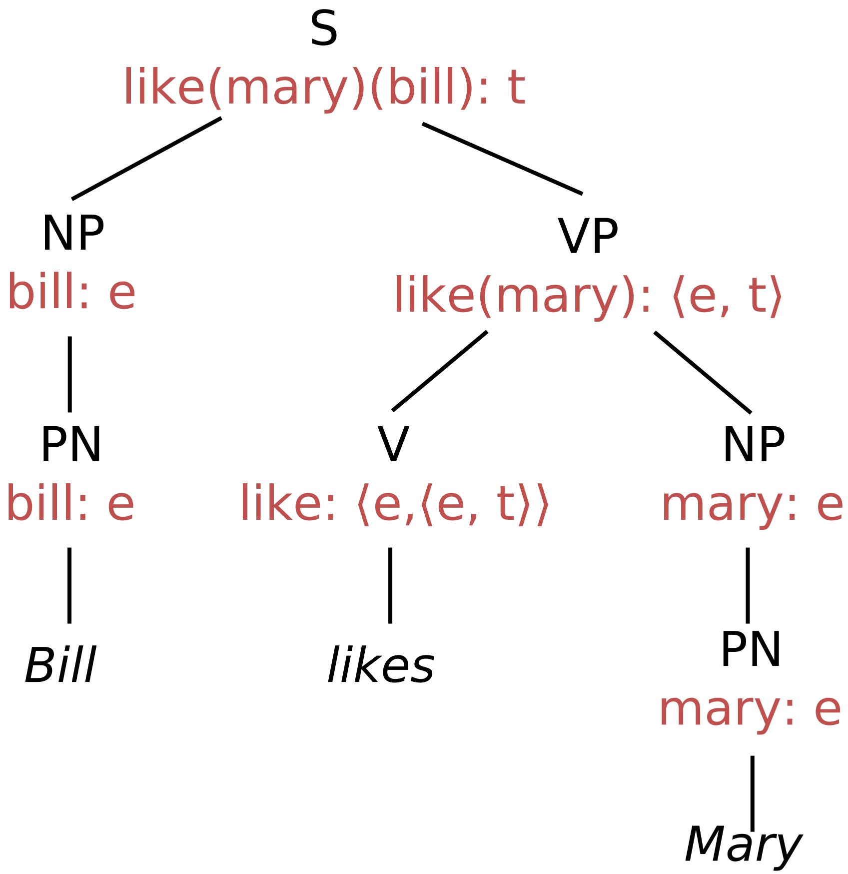
- Combine type-logical expressions to each other, observing type fit and NL syntactic structure.

Bill likes Mary ⇒ like'(mary')(bill')

$$\cfrac{\cfrac{\text{like'}: \langle e,\langle e, t\rangle\rangle \quad \text{mary'}: e}{\text{like'(mary')}: \langle e, t\rangle} \quad \text{bill'}: e}{\text{like'(mary')(bill')}: t}$$

Explicit Semantic Construction Rules

- If in a binary branching local syntactic structure B and C are daughters of A, B⇒β:⟨σ,τ⟩ and C ⇒γ:σ, then A⇒ β(γ):τ.
- If in a unary branching tree A is mother of B and B⇒β, then also A⇒β.

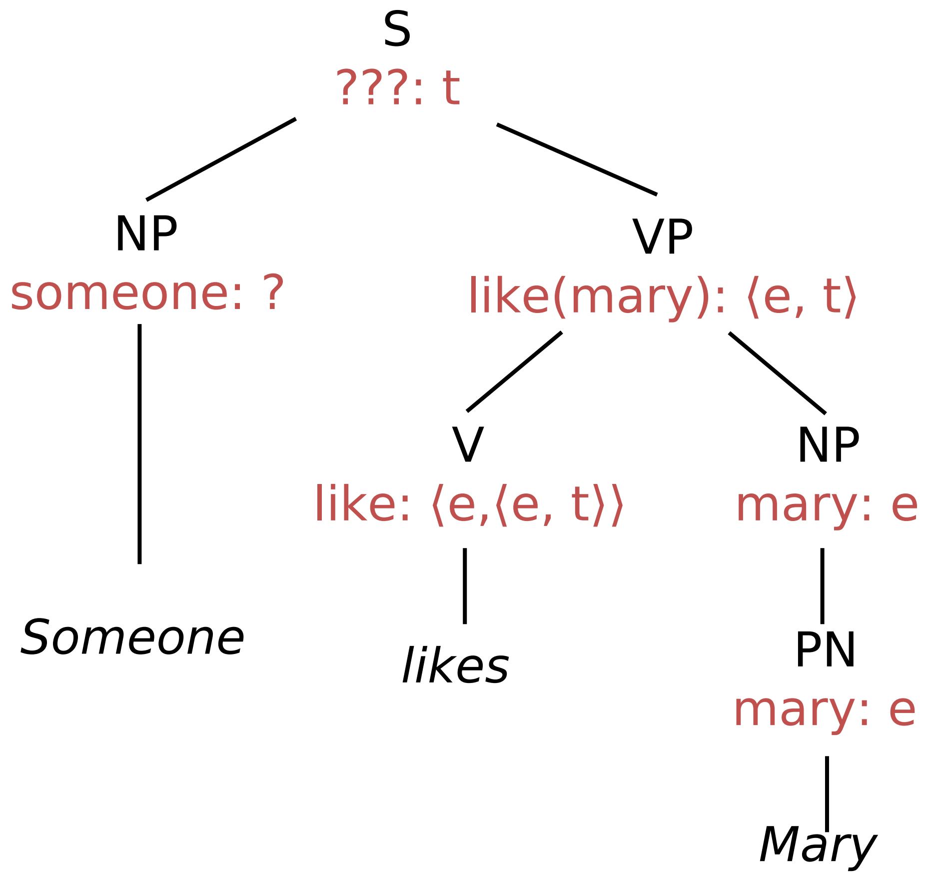


Quantification in NL: A Challenge for Compositional Semantics

Every student presented a paper

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

NL Quantifier Expressions



NL Quantifier Expressions

- First attempt: Assume type e for all kinds of NPs

  Someone works ⇒ work'(someone')

  $$\frac{\text{someone': e} \quad \text{work': ⟨e,t⟩}}{\text{work'(someone'): t}}$$

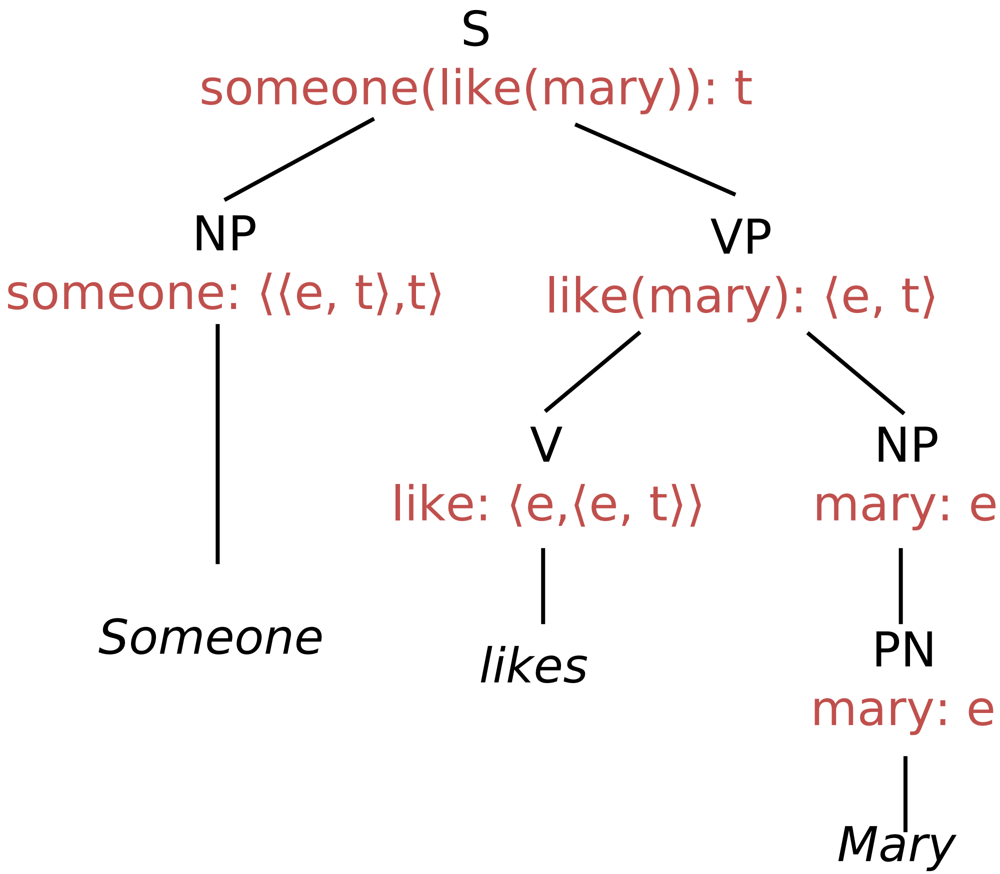
- This does not work. So we try it the other way round:

  Someone works ⇒ someone'(work')

  $$\frac{\text{someone': ⟨⟨e,t⟩,t⟩} \quad \text{work': ⟨e,t⟩}}{\text{someone'(work'): t}}$$

- We analyse “someone“ as a second-order predicate.

NL Quantifier Expressions



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 $\mathcal{P}(U)$ to {0,1},

  which in turn is equivalent to $\mathcal{P}(\mathcal{P}(U))$.
- Thus, V_M(someone’) $\subseteq \mathcal{P}(U_M)$. More specifically:
- V_M(someone’) = $\{S \subseteq U_M \mid S \neq \emptyset\}$, if U_M is a domain of persons
- V_M(everyone’) = $\{U_M\}$, if U_M is a domain of persons

NL Quantifier Expressions: Interpretation

- $[\![\text{someone'(work')}]\!]^{M,g}$ =
  $[\![\text{someone'}]\!]^{M,g}([\![\text{work'}]\!]^{M,g})$ =
  V_M(someone')(V_M(work'))

- V_M(someone')(V_M(work')) = 1 iff
  V_M(work') $\in$ V_M(someone') iff
  V_M(work') $\neq \emptyset$, which holds just in the case that
  some person/entity in model structure M works

NL Determiners

- *Every student works*

every’: ? student’: ⟨e,t⟩

??? : ⟨⟨e,t⟩,t⟩ work’: ⟨e,t⟩

???(work’): t

every’: ⟨⟨e,t⟩,⟨⟨e,t⟩,t⟩⟩ student’: ⟨e,t⟩

every’(student’): ⟨⟨e,t⟩,t⟩ work’: ⟨e,t⟩

every’(student’)(work’): t

NL Determiners: Interpretation

- every’$\in$ $CON_{\langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle}$, so V_M(every’) $\in D_{\langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle}$
- $D_{\langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle}$ is the set of functions from $D_{\langle e,t\rangle}$ to $D_{\langle\langle e,t\rangle,t\rangle}$, i.e., the set of functions from possible first-order predicates to possible second-order predicates (the latter being functions from first-order-predicates to truth values).
- In other words (considering characteristic-function/set equivalence and currying), $D_{\langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle}$ is $\mathcal{P}(D_{\langle e,t\rangle}\times D_{\langle e,t\rangle})$, i.e., the set of second-order two-place relations between first-order one-place predicates.
- Thus V_M(every’) $\subseteq \mathcal{P}(U_M)\times\mathcal{P}(U_M)$. More specifically:
- V_M(every’) = $\{\langle S,T\rangle \mid S, T \subseteq U_M$ and $S \subseteq T\}$

NL Determiners: Interpretation

Every student works ⇒ every'(student')(work')

$[\![\text{every'(student')(work')}]\!]^{M,g} =$

$[\![\text{every'(student')}]\!]^{M,g}([\![\text{work'}]\!]^{M,g}) =$

$[\![\text{every'}]\!]^{M,g}([\![\text{student'}]\!]^{M,g})([\![\text{work'}]\!]^{M,g}) =$

$V_M(\text{every'})(V_M(\text{student'}))\ (V_M(\text{work'}))$

$V_M(\text{every'})(V_M(\text{student'}))\ (V_M(\text{work'})) = 1$ iff (char. function!)

$V_M(\text{work'}) \in V_M(\text{every'})(V_M(\text{student'}))$ iff (currying!)

$\langle V_M(\text{student'}), V_M(\text{work'})\rangle \in V_M(\text{every'})$ iff (interpr. of every)

$V_M(\text{student'}) \subseteq V_M(\text{work'})$

Some More Determiners

- every', some'/a', no', most'$\in$ $CON_{\langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle}$

- V_M(every') = $\{\langle S,T\rangle \mid S \subseteq T\}$
- V_M(some') = $\{\langle S,T\rangle \mid S \cap T \neq \emptyset\}$
- V_M(no') = $\{\langle S,T\rangle \mid S \cap T = \emptyset\}$
- V_M(most') = $\{\langle S,T\rangle \mid |S \cap T| \geq |S - T|\}$

Proper Names: Revised Analysis

- Proper names and quantified NPs have different types, proper names are arguments, quantified NPs are functors.

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

- How can we obtain a unified semantics of noun phrases?
- Assigning type e to someone' does not work.
- So we do it the other way round: "**Raising**" proper names to type ⟨⟨e,t⟩,t⟩.

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

Proper Names: Interpretation

- $john' \in CON_{\langle\langle e,t\rangle,t\rangle}$, so $V_M(john') \in D_{\langle\langle e,t\rangle,t\rangle}$
- Proper names are second-order predicates denoting sets of sets.
- Thus $V_M(john') \subseteq \mathcal{P}(U_M)$. More specifically:
- $V_M(john') = \{S \subseteq U_M \mid j \in S\}$, for some specific entity $j \in U_M$ (i.e., the person John)

- $[\![john'(work')]\!]^{M,g} = 1$ iff

  $V_M(john')(V_M(work')) = 1$ iff

  $V_M(work') \in V_M(john')$ iff

  $j \in V_M(work')$