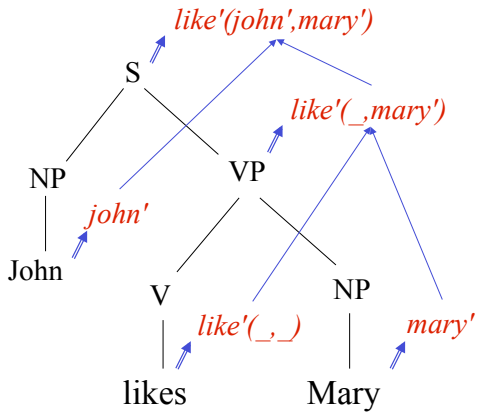
# Semantic Theory

## Semantic Construction 2

Manfred Pinkal/ Stefan Thater
Saarland University

## Basic Semantic Composition






## A Challenge for Semantic Composition

*Every student presented a paper*

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

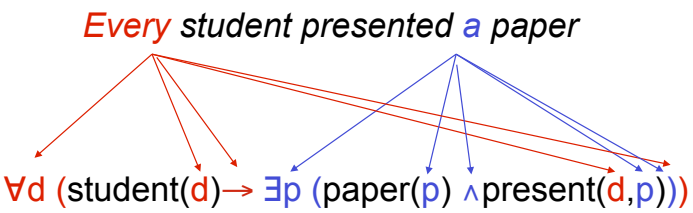


## A Challenge for Semantic Composition

*Every student presented a paper*

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

## A Challenge for Semantic Composition






- In general, FOL semantic representations do not structurally correspond to the syntactic structure of NL sentences.
- How do we model the semantic composition process?
- We approach the problem via a detour: Looking into higher-order phenomena in NL semantics



## FOL Limitations

| | |
|---|---|
| *John is a married piano player* | piano-player(j) ∧ married(j) |
| *John is a blond criminal* | criminal(j) ∧ blond(j) |
| *John is a poor piano player* | piano-player(j) ∧ poor(j) ? |
| *John is an alleged criminal* | criminal(j) ∧ alleged(j) ??? |



## FOL Limitations

*Yesterday, it rained.*
*Probably, it is raining.*
*Unfortunately, it is raining.*

*Bill is blond. Blond is a hair colour.*

$\not\models$ *Bill is a hair colour*

## The Language of Type Theory

### Types:

- The set of basic types is {e, t} :
  - e (for entity) is the type of individual terms
  - t (for truth value) is the type of formulas

- All pairs $(\sigma, \tau)$ made up of (basic or complex) types $\sigma, \tau$ are types. $<\sigma, \tau)$ is the type of functions which map arguments of type $\sigma$ to values of type $\tau$.

- In short: The set of types is the smallest set **T** such that e,t∈**T**, and if $\sigma,\tau \in$**T,** then also $<\sigma,\tau) \in$**T.**



## Some Useful Types for NL Semantics

- Proper name bill: e
- Sentence it_rains: t

- One-place predicate constant:
  *work, student: <e,t>*
- Two-place relation:
  *like, larger_than: <e,<e,t>>*
- Sentence adverbial:
  *yesterday, unfortunately: <t,t>*
- Attributive adjective:
  *married, poor, alleged: <<e,t>,<e,t>>*
- Degree modifier:
  *very, relatively: <<<e,t>,<e,t>>,<<e,t>,<e,t>>>*



## Second-order predicates

*Bill is blond. Blond is a hair colour.*

*bill: e  blond: <e,t>*

*blond(bill): t*

*Blond is a hair colour.*

*blond: <e,t>  hair_colour : <<e,t>,t>*

*hair_colour (blond): t*

*Bill is a hair colour*

- Hair-colour is a second-order predicate. *hair_colour(bill)* is not even a well-formed expression.



## Type-theoretic syntax

- Vocabulary:
  - Possibly empty, pairwise disjoint sets of constants: $Con_\tau$, for every type $\tau$
  - Pairwise disjoint sets of variables: $Var_\tau$, for every type $\tau$
  - The usual predicate logic operators: connectives, quantifiers, equality

## Type-theoretic syntax

- Vocabulary:
  - A (possibly empty) set of constants: $Con_\tau$, for every type $\tau$
  - A set of variables: $Var_\tau$, for every type $\tau$
  - The usual FOL operators: connectives, quantifiers, equality
- The sets of well-formed expressions $WE_\tau$ for every type $\tau$ are given by:
  - $Con_\tau \cup Var_\tau \subseteq WE_\tau$ for every type $\tau$
  - If $\alpha \in WE_{<\sigma,\tau>}$, $\beta \in WE_\sigma$, then $\alpha(\beta) \in WE_\tau$.
  - If A, B are in $WE_t$, then so are $\neg$ A, (A$\wedge$B), (A$\vee$B), (A$\rightarrow$B),(A$\leftrightarrow$B)
  - If A is in $WE_t$, then so are $\forall v$A and $\exists v$A, where *v* is a variable of arbitrary type.
  - If $\alpha$, $\beta$ are well-formed expressions of the same type, then $\alpha=\beta \in WE_t$.



## Function Application

- The most important syntactic operation in type-theory is function application:
  If $\alpha \in WE_{<\sigma,\tau>}$, $\beta \in WE_\sigma$, then $\alpha(\beta) \in WE_\tau$.
- A functor of complex type combines with an appropriate argument to a (more complex) expression of less complex type.



## Function Application: Examples

*Bill drives fast*

*drive: <e,t>* *fast: <<e,t>,<e,t>>*

*bill: e* *fast(drive): <e,t>*

*fast(drive)(bill): t*

*Mary works in Saarbrücken*

*mary: e* *work: <e,t>* *in: <e,<t,t>>* *sb: e*

*work(mary): t* *in(sb): <t,t>*

*in(sb)(work(mary)): t*



## Using Higher-Order Variables

- *Bill has the same hair colour as John.*
  $\exists \mathcal{G}$ (hair_colour($\mathcal{G}$) $\wedge$ $\mathcal{G}$ (bill) $\wedge$ $\mathcal{G}$ (john))
- *Santa Claus has all the attributes of a sadist.*
- $\forall \mathcal{F} \forall$a(sadist(a) $\wedge$ $\mathcal{F}$ (a) $\rightarrow$ $\mathcal{F}$(b))

# Type-theoretic semantics [1]

- Let U be a non-empty set of entities. The domain of possible denotations $D_\tau$ for every type $\tau$ is given by:
  - $D_e = U$
  - $D_t = \{0,1\}$
  - $D_{<\sigma,\tau>}$ is the set of all functions from $D_\sigma$ to $D_\tau$
- A model structure for a type theoretic language:
  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 $Con_\tau$ an element of $D_\tau$.
- Variable assignment g assigns every variable of type $\tau$ a member of $D_\tau$.



# Type-theoretic semantics [2]

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

$[[\alpha]]^{M,g} = V_M(\alpha)$, if $\alpha$ constant

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

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

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

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

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

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

$[[\alpha=\beta]]^{M,g} = 1$ iff $[[\alpha]]^{M,g} = [[\beta]]^{M,g}$



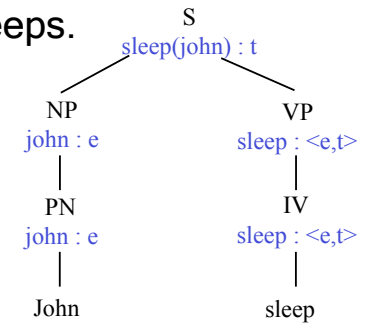
# Recommended Reading

- Textbook: L.T.F. Gamut, Logic, Language, and Meaning. Volume2: Intensional Logic and Logical Grammar. University of Chicago Press 1991



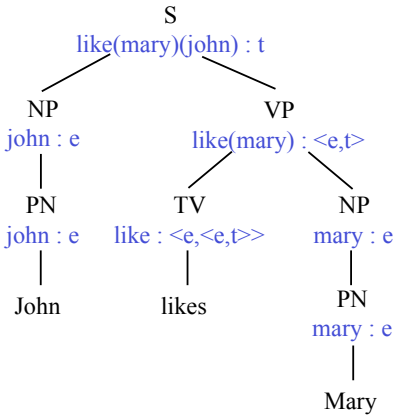
# Semantics construction

- John sleeps.



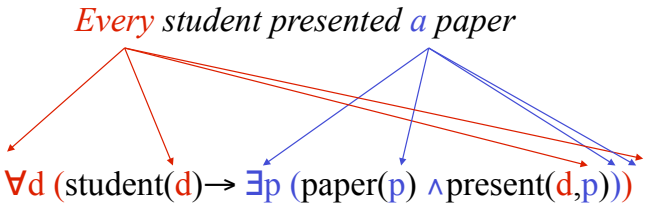
## Semantics construction

- John likes Mary.






## The composition problem again






## Varieties of Noun Phrases

- Challenge 1:
  A local semantic interpretation for full quantified noun phrases which brings about the correct global sentence analysis (through the operation of functional application).



## The Semantics of Quantified NPs

First attempt: Generalize proper name interpretation to all kinds of noun phrases

*John works.*

john: e  work: (e,t)

work(john): t

*Every student works.*

every-student: e  work: (e,t)

every-student(work): t

This does not work !!!

# The Semantics of Quantified NPs

So we try it the other way round:

*Every student works.*

every-student': ((e,t),t) work': (e,t)

every-student'(work'): t

'*Every student*' denotes a second-order predicate that is true of a first-order predicate, if all students are in the denotation of that predicate.

More technically speaking, for $A \subseteq U_M$:

$[\![\text{every-student'}]\!]^{M,g}(A)=1$ iff $V_M[\![\text{student'}]\!] \subseteq A$

Simliarly for '*a student*' and '*no student*'



# Internal NP structure

- Challenge 1b:

  Provide an interpretation of determiners like *every* which allows to compositionally derive the semantics of the complex noun phrase.

  Determiners denote functions from first-oder predicates („student") to second-order predicates („every student") ; in other words: functions from first-order predicates to functions from first-order predicates to truth values.

every': ((e,t),((e,t),t)) student': (e,t)

every'(student'): ((e,t),t) work: (e,t)

every'(student')(work'): t

For $B \in U_M$: $[\![\text{every}]\!]^{M,g}(A)$ is that function $\varphi \in D_{((e,t),t)}$ such that $\varphi(B)=1$ iff $A \subseteq B$.

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

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



# Internal NP structure

- Other determiners, like „no" or the indefinite article can be interpreted accordingly:

$V_M(\text{every'})(A)(B) = 1$ iff $A \subseteq B$

$V_M(\text{a'})(A)(B) = 1$ iff $A \cap B \neq \varnothing$

$V_M(\text{no'})(A)(B) = 1$ iff $A \cap B = \varnothing$

- Curry's result allows us to conceive of a determiner denotation as a second-order two-place relation (between two first-order predicates):

  *every* expresses the inclusion relation between the first and the second, *no* the mutual exclusion relation, and *a* or *some* the relation of overlap.



# A unified NP Semantics

- Challenge 2:

  A unified semantic interpretation for all types of NPs, including proper names and quantified 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

# A unified NP semantics

- Proper names and full NPs are of the same syntactic category and syntactically substitutable.
- It is desirable to give them the same type, in order to allow semantic construction to work in a uniform way through the syntactic structure.
- So far, they have different types.
- „Every student“ cannot take type e.
- So we generalise the other way: We give proper names the type ((e,t),t).

*John works.*

john‘: ((e,t),t) work‘: (e,t)

john‘(work‘): t



# A unified NP semantics

- Proper names of course have to be given a special interpretation:
  *John*’ denotes a second-order predicate that is true of a first-order predicate , if and only if John is in the denotation of that predicate.
  More technically speaking, there is a designated individual $a \in U_M$ (i.e., the person John) such that for every $A \subseteq U_M$:
  $\text{john}'^{M,g}(A)=1$ iff $a \in A$
- We can capture this constraint by an axiom or meaning postulate. Let j* be a designated constant of type e, denoting John.
  $\forall \mathcal{F}\,(\text{john}'(\mathcal{F}) \leftrightarrow \mathcal{F}(j^*))$
- The technique of lifting the type of an expression to a more complex type, ensuring by special interpretation rules or axioms that it keeps its original semantic information, is called type raising.



# Compositionality and Inference

- Higher-order logic provides a framework for a unified treatment of all types of NPs in semantic construction.
- However, by representing quantified NPs as basic second-order predicates, without FOL quantifiers and connectives, we lose one of the main motivations for a logical treatment: The availability of deduction systems and theorem provers which support correct and efficient reasoning.

- Challenge 3:
  A semantic representation that preserves compositionality of quantified expressions and at the same time provides access to the deductive power of FOL.
- The solution: Extending higher-order logic to typed λ-calculus.