Montague Grammar

Stefan Thater

Blockseminar “Underspecification”

10.04.2006

Overview

- Introduction
- Type Theory
- A Montague-Style Grammar
- Scope Ambiguities
- Summary

Introduction

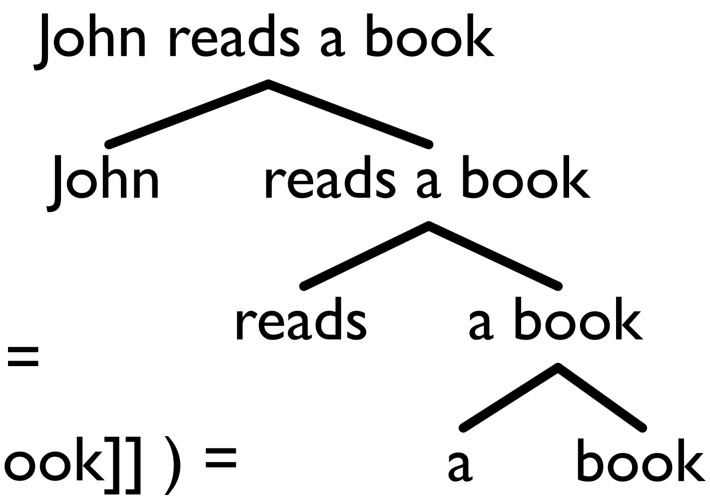
- The basic assumption underlying Montague Grammar is that the meaning of a sentence is given by its truth conditions.
  - “Peter reads a book” is true iff Peter reads a book
- Truth conditions can be represented by logical formulae
  - “Peter reads a book” → $\exists x(book(x) \wedge read(p^*, x))$
- Indirect interpretation:
  - natural language → logic → models

Compositionality

- An important principle underlying Montague Grammar is the so called "principle of compositionality"

  The meaning of a complex expression is a function of the meanings of its parts, and the syntactic rules by which they are combined (Partee & al, 1993)

Compositionality



[[John reads a book]] =

C_1([[John]], [[reads a book]]) =

C_1([[John]], C_2([[reads]] , [[a book]]) =

C_1([[John]], C_2([[reads]] , C_3([[a]], [[book]]))

Representing Meaning

- First order logic is in general not an adequate formalism to model the meaning of natural language expressions.
- Expressiveness
  - "John is an intelligent student" ⇒ intelligent(j*) ∧ stud(j*)
  - "John is a good student" ⇒ good(j*) ∧ stud(j*) ??
  - "John is a former student" ⇒ former(j*) ∧ stud(j*) ???
- Representations of noun phrases, verb phrases, …
  - "is intelligent" ⇒ intelligent(·) ?
  - "every student" ⇒ ∀x(student(x) ⇒ ·) ???

Type Theory

- First order logic provides only n-ary first order relations, which is insufficient to model natural language semantics.
- Type theory is more expressive and flexible – it provides higher-order relations and functions of different kinds.
- Some type theoretical expressions
  - "John is a good student" ⇒ good(student)(j*)
  - "is intelligent" ⇒ intelligent
  - "every student" ⇒ $\lambda P \forall x(\text{student}(x) \Rightarrow P(x))$

Types

- A set of basic types, for instance {e, t}
  - e is the type of individual terms (“entity”)
  - t is the type of formulas (“truth value”)
- The set T of types is the smallest set such that
  - if σ is a basic type, then σ is a type
  - if σ, τ are types, then $\langle\sigma, \tau\rangle$ is a type
- The type $\langle\sigma, \tau\rangle$ is the type of functions that map arguments of type σ to values of type τ.

Some Example Types

- One-place predicate constant: sleep, walk, student, …
  - ‹e, t›
- Two-place relation: read, write, …
  - ‹e, ‹e,t››
- Attributive adjective: good, intelligent, former, …
  - ‹‹e,t›, ‹e,t››

Vocabulary

- Pairwise disjoint, possibly empty sets of non-logical constants:
  - Con_τ, for every type τ
- Infinite and pairwise disjoint sets of variables:
  - Var_τ, for every type τ
- Logical constants:
  - $\forall, \exists, \wedge, \neg, \ldots, \lambda$

Syntax

- For every type τ, we define the set of meaningful expressions ME_τ as follows:
  - $Con_\tau \subseteq ME_\tau$ and $Var_\tau \subseteq ME_\tau$, for every type τ
  - If $\alpha \in ME_{\langle\sigma, \tau\rangle}$ and $\beta \in ME_\sigma$, then $\alpha(\beta) \in ME_\tau$.
  - If $A, B \in ME_t$, then so are $\neg A$, $(A \wedge B)$, $(A \Rightarrow B)$, …
  - If $A \in ME_t$, then so are $\forall xA$ and $\exists xA$, where x is a variable of arbitrary type.
  - If α, β are well-formed expressions of the same type, then $\alpha = \beta \in ME_t$.
  - If $\alpha \in ME_\tau$ and $x \in Var_\sigma$, then $\lambda x\alpha \in ME_{\langle\sigma, \tau\rangle}$.

Some Examples

- "John works."

$$\frac{j^* \in ME_e \qquad work \in ME_{\langle e, t\rangle}}{work(j^*)}$$

- "Every student works."

$$\frac{\dfrac{every \in ME_{\langle\langle e, t\rangle, \langle\langle e, t\rangle, t\rangle} \qquad student \in ME_{\langle e, t\rangle}}{every(student) \in ME_{\langle\langle e, t\rangle, \langle\langle e, t\rangle, t\rangle}} \qquad work \in ME_{\langle e, t\rangle}}{every(student)(work) \in ME_t}$$

Semantics

- Let U be a non-empty set of entities. For every type τ, the domain of possible denotations D_τ is given by
  - $D_e = U$
  - $D_t = \{0,1\}$
  - $D_{\langle\sigma,\tau\rangle}$ = the set of functions from D_σ to D_τ
- A model structure is a structure $M = (U_M, V_M)$
  - U_M is a non-empty set of individuals
  - V_M is a function that assigns every non-logical constant of type τ an element of D_τ.
- Variable assignment g: $Var_\tau \rightarrow D_\tau$

Semantics

- Let M be a model structure and g a variable assignment
  - $[[\alpha]]^{M,g} = V_M(\alpha)$, if α is a constant
  - $[[\alpha]]^{M,g} = g(\alpha)$, if α is a 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.
  - $[[\exists v\varphi]]^{M,g} = 1$ iff there is $a \in D_\tau$ such that $[[\varphi]]^{M,g[v/a]} = 1$
  - $[[\forall v\varphi]]^{M,g} = 1$ iff for all $a \in D_\tau$, $[[\varphi]]^{M,g[v/a]} = 1$
  - $[[\alpha = \beta]]^{M,g} = 1$ iff $[[\alpha]]^{M,g} = [[\beta]]^{M,g}$

Semantics of λ-Expressions

- Let M be a model structure and g a variable assignment
  - If $\alpha \in ME_\tau$ and $v \in Var_\sigma$, then $[[\lambda v \alpha]]^{M,g}$ is that function f from D_σ to D_τ such that for any $a \in D_\sigma$, $f(a) = [[\alpha]]^{M,g[v/a|}$
- "Syntactic shortcut:" β-reduction
  - $(\lambda x \varphi)(\psi) \equiv \varphi[\psi/x]$

    if all free variables in ψ are free for x in φ
  - A variable y is free for x in φ if no free occurence of x in ψ is in the scope of a ∃y, ∀y, λy

Noun Phrases

- "John works" → $\text{work}(j^*)$
- "A student works." → $\exists x(\text{student}(x) \wedge \text{work}(x))$
- "Every student works." → $\forall x(\text{student}(x) \Rightarrow \text{work}(x))$
- "John and Mary work." → $\text{work}(j^*) \wedge \text{work}(m^*)$

Noun Phrases

- Using λ-abstraction, noun phrases can be given a uniform interpretation as "generalized quantifiers"
  - "John" → $\lambda P.P(j^*)$
  - "A student" → $\lambda P \exists x(student(x) \wedge P(x))$
  - "Every student" → $\lambda P \forall x(student(x) \Rightarrow P(x))$
  - "John and Mary" → $\lambda P.P(j^*) \wedge P(m^*)$

Noun Phrases

- "John works"

$$\frac{\dfrac{\lambda P.P(j^*) \in ME_{\langle\langle e, t\rangle, t\rangle} \quad work \in ME_{\langle e, t\rangle}}{(\lambda P.P(j^*))(work) \in ME_t}}{work(j^*) \in ME_t}$$

- "Every student works."

$$\frac{\dfrac{\lambda P \forall x(student(x) \Rightarrow P(x)) \in ME_{\langle\langle e, t\rangle, t\rangle} \quad work \in ME_{\langle e, t\rangle}}{(\lambda P \forall x(student(x) \Rightarrow P(x)))(work) \in ME_t}}{\forall x(student(x) \Rightarrow work(x)) \in ME_t}$$

Determiners

- Determiners like "a," "every," "no" denote higher order functions taking (denotations of) common nouns and return a higher order relation.
  - "every" → $\lambda P \lambda Q \forall x (P(x) \Rightarrow Q(x))$
  - "some" → $\lambda P \lambda Q \exists x (P(x) \wedge Q(x))$
  - "no" → $\lambda P \lambda Q \neg \exists x (P(x) \wedge Q(x))$
- "Every student"

$$\dfrac{\dfrac{\lambda P \lambda Q \forall x (P(x) \Rightarrow Q(x)) \qquad \text{student}}{(\lambda P \lambda Q \forall x (P(x) \Rightarrow Q(x)))(\text{student})}}{\lambda Q \forall x (\text{student}(x) \Rightarrow Q(x))}$$

A Montague-Style Grammar for a Fragment of English

Syntactic Component

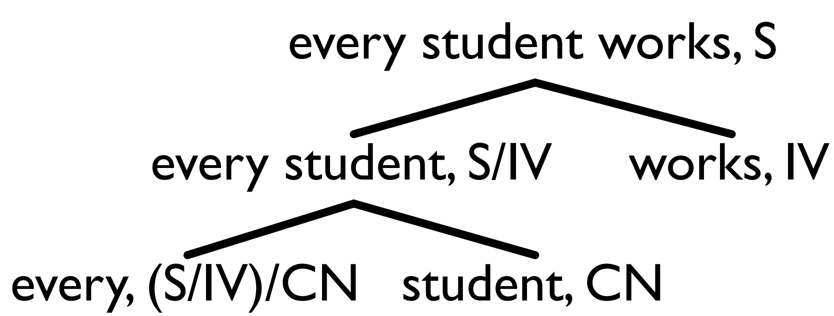
- Montague Grammar is based upon (a particular version of) categorial grammar.
- The set of categories is the smallest set such that
  - S, IV, CN are categories
  - If A, B are categories, then A/B is a category
- Some categories
  - IV/T [= TV] transitive verbs
  - S/IV [= T] terms (= noun phrases)
  - T/CN determiners

Lexicon

- For each category A, we assume a possibly empty set B_A of basic expressions of category A.
- For instance
  - B_T = { John, Mary, he_0, he_1, … }
  - B_{CN} = { student, man, woman, … }
  - B_{IV} = { sleep, work, … }
  - $B_{IV/T}$ = { read, … }
  - $B_{T/CN}$ = { a, every, no, the, … }

Syntactic Rules (Simplified)

- General rule schema:
  - $B_A \subseteq P_A$
  - If $\alpha \in P_A$ and $\delta \in P_{B/A}$, then $\delta\alpha \in P_B$
- "Every student works"



Translation into Type Theory

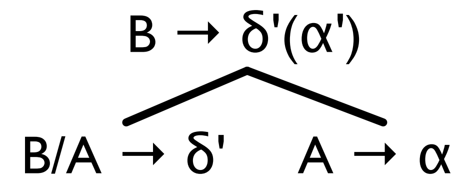
- A translation of natural language into type theory is a homomorphism that assigns each $\alpha \in P_A$ an $\alpha' \in ME_{f(A)}$
- f maps categories to types as follows
  - $f(S) = t$
  - $f(CN) = f(IV) = \langle e, t \rangle$
  - $f(A/B) = \langle f(B), f(A) \rangle$

Translation: Lexical Categories

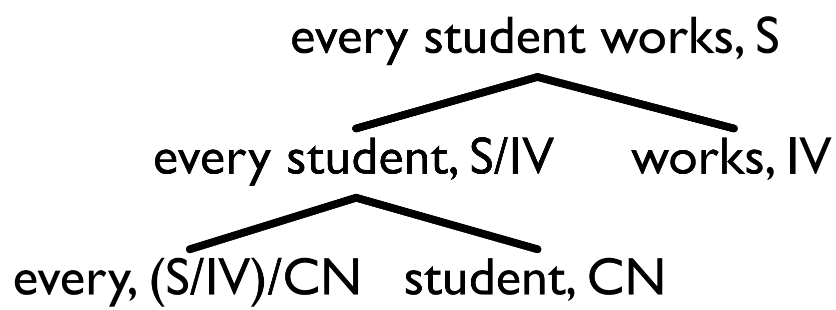
- “John” $\rightarrow$ $\lambda P.P(j^*)$
- “every” $\rightarrow$ $\lambda P \lambda Q \forall x(P(x) \Rightarrow Q(x))$
- “a” $\rightarrow$ $\lambda P \lambda Q \exists x(P(x) \wedge Q(x))$
- “student” $\rightarrow$ student
- “book” $\rightarrow$ book
- “works” $\rightarrow$ work
- …

Translation: Phrasal Categories

- Syntactic rule:
  - If $\alpha \in P_A$ and $\delta \in P_{B/A}$, then $\delta\alpha \in P_B$
- Corresponding translation rule:
  - If $\alpha \rightarrow \alpha'$, $\delta \rightarrow \delta'$, then $\delta\alpha \rightarrow \delta'(\alpha')$



"Every student works"



- "every" → $\lambda P \lambda Q \forall x(P(x) \Rightarrow Q(x))$
- "student" → student
- "every student" → $\lambda P \lambda Q \forall x(P(x) \Rightarrow Q(x))(\text{student})$

  $= \lambda Q \forall x(\text{student}(x) \Rightarrow Q(x))$
- "every student works" → $\lambda Q \forall x(\text{student}(x) \Rightarrow Q(x))(\text{work})$

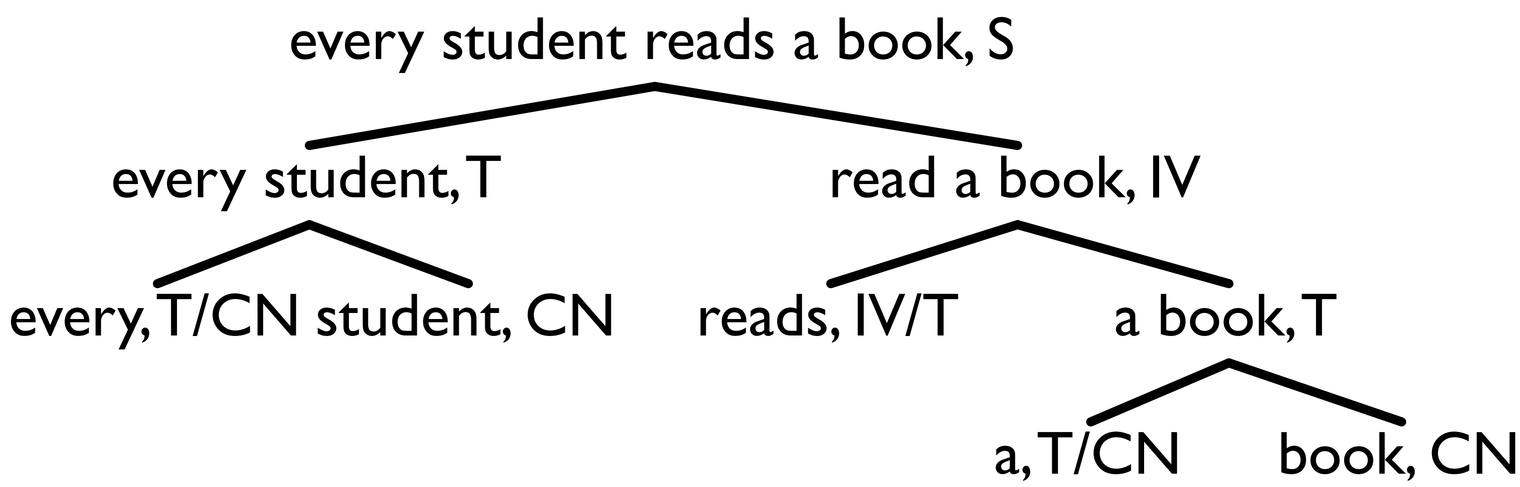
  $= \forall x(\text{student}(x) \Rightarrow \text{work}(x))$

Transitive Verbs

- Transitive verbs have category IV/T (= IV/(S/IV)), the corresponding type is ‹‹‹e, t›, t›, ‹e, t››
- On the other hand, transitive verbs like "read," "present," … denote a two-place first order relation (type ‹e, ‹e, t››)
  - "John reads a book" → $\exists y(\text{book}(y) \wedge \text{read}(y)(j^*))$

- "read" → $\lambda Q \lambda x.Q(\lambda y.\text{read}^*(y)(x))$
  - $\text{read}^* \in ME_{\langle e, \langle e, t\rangle\rangle}$

“Every student reads a book”



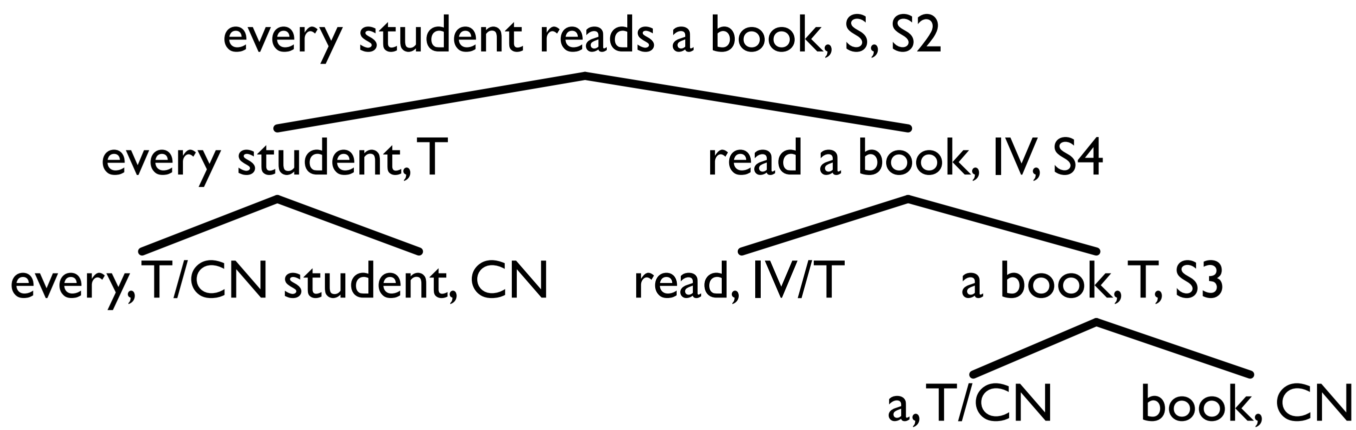
“Every student reads a book”

- “a book” → $\lambda P \exists z(book(z) \wedge P(z))$
- “reads” → $\lambda Q \lambda x.Q(\lambda y.read^*(y)(x))$
- “reads a book”
  - → $\lambda Q \lambda x.Q(\lambda y.read^*(y)(x))(\lambda P \exists z(book(z) \wedge P(z)))$
  - → $\lambda x.\lambda P \exists z(book(z) \wedge P(z))(\lambda y.read^*(y)(x))$
  - → $\lambda x.\exists z(book(z) \wedge (\lambda y.read^*(y)(x))(z))$
  - → $\lambda x.\exists z(book(z) \wedge read^*(z)(x))$
- “every student reads a book”
  - → $\lambda P \forall w(student(w) \Rightarrow P(w))(\lambda x.\exists z(book(z) \wedge read^*(z)(x))$
  - → $\forall w(student(w) \Rightarrow \exists z(book(z) \wedge read^*(z)(w)))$

Scope

- Sentences with multiple scope bearing operators – e.g., quantified noun phrases or negations – are often ambiguous.
- "Every student reads a book"
  - $\forall x(\text{student}(x) \Rightarrow \exists y(\text{book}(y) \wedge \text{read}(y)(x)))$
  - $\exists y(\text{book}(y) \wedge \forall x(\text{student}(x) \Rightarrow \text{read}(y)(x)))$
- "Every student did not pay attention"
  - $\forall x(\text{student}(x) \Rightarrow \neg\, \text{pay attention}(x))$
  - $\neg\, \forall x(\text{student}(x) \Rightarrow \text{pay attention}(x))$

The Problem

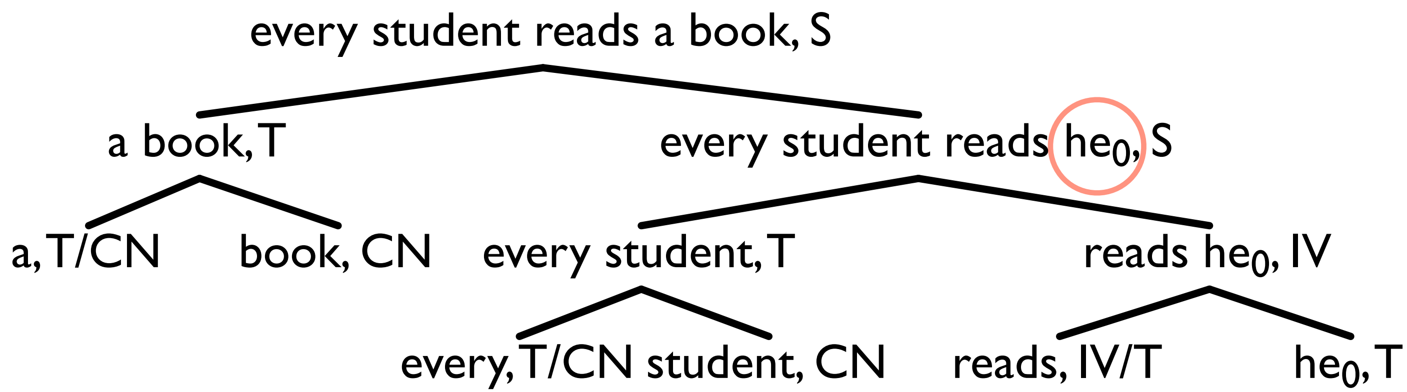


- The principle of compositionality implies that syntactic derivation trees are mapped to a unique type theoretical semantic representation.
- Hence the second reading cannot be derived, unless …

“Montague’s Trick”

- Special rule of quantification (aka “Quantifying-in”)
  - Terms $\alpha \in P_T$ can combine with sentences $\xi \in P_S$ to form a sentence $\xi' \in P_S$,
  - where ξ' is obtained from ξ by replacing all occurrences of “he_i” with α.
  - For instance: “a book” + “… he_1 …” = “… a book …”
- Sentences can be assigned distinct syntactic derivations

"Montague's Trick"



- "he_0" → $\lambda P.P(x_0)$
- "every student reads he_0" → $\forall y(student(y) \Rightarrow read(x_0)(y))$
- "every student reads a book"

  → $\lambda P \exists x(book(x) \wedge P(x))(\lambda x_0 \forall y(student(y) \Rightarrow read(x_0)(y)))$

  → $\exists x(book(x) \wedge \forall y(student(y) \Rightarrow read(x)(y)))$

“Montague’s Trick”

- The quantification rule allows to derive different scope readings of ambiguous sentences, but …
  - the syntax is made more ambiguous than it actually is
  - no surface oriented analysis

Summary

- The principle of compositionality
  - links syntax and semantics of natural language
- Type theory offers
  - flexibility
  - expressiveness
- Montage like semantics construction …
  - follows the principle of compositionality
  - assumes a strict one-to-one correspondence between syntax and corresponding semantic representations,
  - but needs a “trick” to model scope ambiguities