Semantic Theory
Scope Ambiguities & Cooper Storage

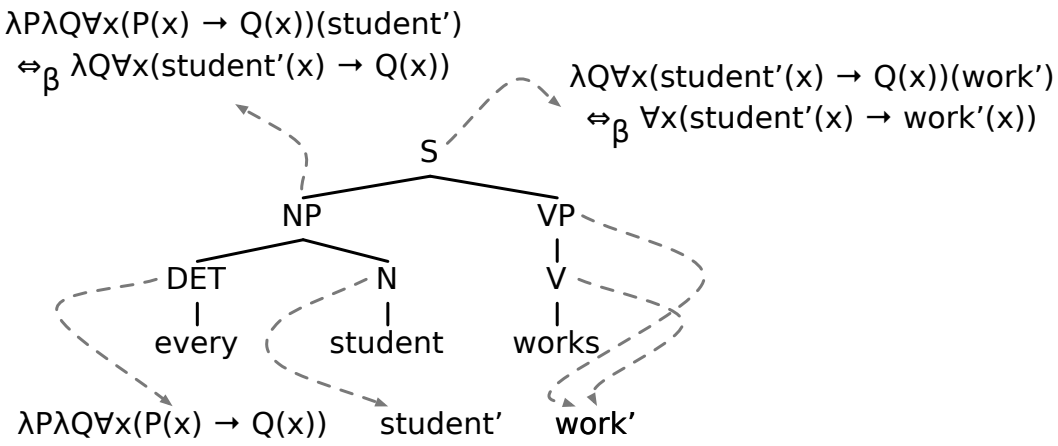
Manfred Pinkal
Stefan Thater

2009-06-30

A Reminder

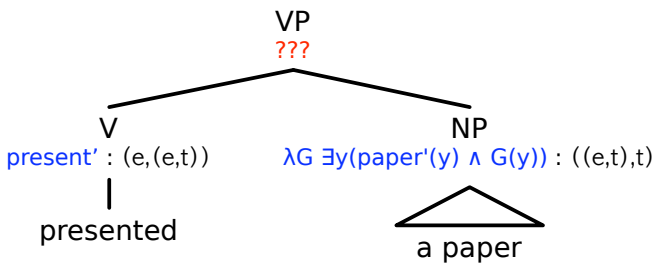
- Noun phrases are uniformly analysed as sets of sets of individuals ("generalised quantifiers"), type ((e,t), t)
  - ⟦Bill⟧ = set of properties (sets of individuals) that Bill has
    - Bill ⇒ λP.P(j)
  - ⟦every student⟧ = set of properties that every student has
    - every student ⇒ λP.∀x(student'(x) → P(x))
  - ⟦a student⟧ = set of properties that a student has
    - a student ⇒ λP.∃x(student'(x) ∧ P(x))

Every student works



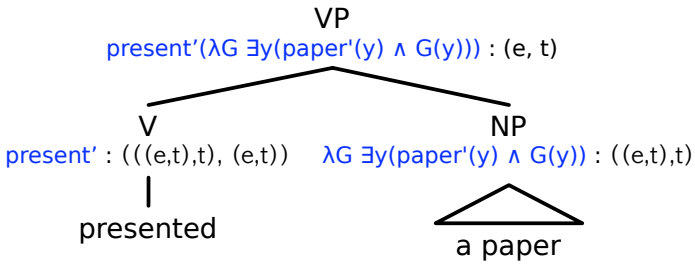
Transitive Verbs

- *Every student presented a paper*
- A composition problem:
  - *a paper* ⇒ λG∃y(paper'(y) ∧ G(y)) : ((e, t), t)
  - *presented* ⇒ present': (e, (e, t))



Transitive Verbs

- *Every student presented a paper*
- Solution: raise the type of the first-order relation:
  - *a paper* ⇒ λGƎy(paper'(y) ∧ G(y)) : ((e, t), t)
  - *presented* ⇒ present': (((e, t), t), (e, t)



Transitive Verbs

- The semantic representation doesn't beta-reduce to a first-order formula if "presented" is translated into a constant *present'* of type (((e,t), t), (e, t)).
  - *Every student presented a paper*
  - ∀x(student'(x) → present'(λGƎy paper'(y) ∧ G(y))(x))
- Meaning postulate for "presented:"
  - ∀G∀x(present'(G)(x) → G(λy.present*(y)(x)))
  - present* is the underlying first-order relation, type (e,(e,t))
- Can be represented as a λ-term as follows:
  - present ⇒ λQλx(Q(λy(present*(y)(x)))) : (((e,t),t),(e,t)),

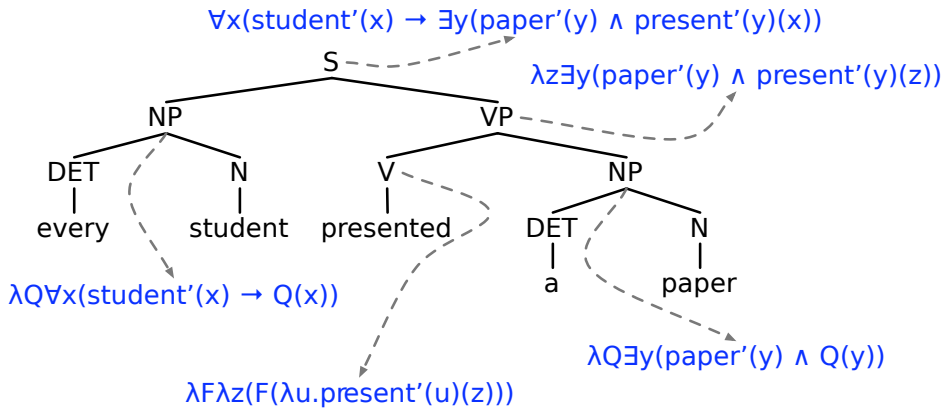
… presented a paper

- *a paper* ⇒ λGƎz(paper'(z) ∧ G(z))
- *presented* ⇒ λQλx[Q(λy[present*(y)(x)])]
- *presented a paper*
  - ⇒ λQλx[Q(λy[present*(y)(x)])](λGƎz(paper'(z) ∧ G(z)))
  - ⇒ λx[λGƎz(paper'(z) ∧ G(z))(λy[present*(y)(x)])]
  - ⇒ λx[Ǝz(paper'(z) ∧ λy[present*(y)(x)](z)]
  - ⇒ λx[Ǝz(paper'(z) ∧ present*(z)(x)]

Every student presented a paper

- *presented a paper*
  - ⇒ λxƎz(paper'(z) ∧ present*(z)(x))
- *every student*
  - ⇒ λG∀w(stud'(w) → G(w))
- *every student presented a paper*
  - ⇒ λG∀w(stud'(w) → G(w))(λxƎz(paper'(z) ∧ presnt*(z)(x)))
  - ⇒ ∀w(stud'(w) → λxƎz(paper'(z) ∧ present*(z)(x)))(w))
  - ⇒ ∀w(stud'(w) → Ǝz(paper'(z) ∧ present*(z)(w)))

Every student presented a paper



Scope – Terminology

- Logic: Quantifier and Scope
  - ∀x(student'(x) → work'(x))
- Natural language semantics:
  - Determiner + Restriction form NP-Denotation
  - NP-denotation is applied to its nuclear scope
  - every'(student')(work')
  - (λPλQ∀x(P(x) → Q(x))(student')(work')

Variable Quantifier-Scope

(1) *Every linguist speaks two languages*

(2) *Our company has an expert for every problem*

(3) Headline: *A search engine for every subject*

Scope-Sensitive Operators

(1) *Every student didn't pay attention.*

(2) *Every citizen can become president.*

(3) *During his visit to China, Helmut Kohl intends to visit a factory for CFC-free refrigerators*

Scope Ambiguities

(1) *Every student presents a paper.*

(a) $\forall x(\text{student}'(x) \rightarrow \exists y(\text{paper}'(y) \wedge \text{present}'(x,y)))$

(b) $\exists y(\text{paper}'(y) \wedge \forall x(\text{student}'(x) \rightarrow \text{present}(x,y)))$

(2) *Every student didn't pay attention.*

(a) $\forall x(\text{student}'(x) \rightarrow \neg\text{pay-attention}'(x))$

(b) $\neg\forall x(\text{student}'(x) \rightarrow \text{pay-attention}'(x))$

Scope Ambiguities

(1) *Every researcher of a company saw some sample.*

(a) $\forall x((\text{res}'(x) \wedge \exists y(\text{cp}'(y) \wedge \text{of}'(x,y))) \rightarrow \exists z(\text{spl}'(z) \wedge \text{see}'(x,z))$

(b) $\exists z(\text{spl}'(z) \wedge \forall x((\text{res}'(x) \wedge \exists y(\text{cp}'(y) \wedge \text{of}'(x,y))) \rightarrow \text{see}'(x,z))$

(c) $\exists y(\text{cp}'(y) \wedge \forall x((\text{res}'(x) \wedge \text{of}'(x,y)) \rightarrow \exists z(\text{spl}'(z) \wedge \text{see}'(x,z)))$

(d) $\exists y(\text{cp}'(y) \wedge \exists z(\text{spl}'(z) \wedge \forall x((\text{res}'(x) \wedge \text{of}'(x,y)) \rightarrow \text{see}'(x,z)))$

(e) $\exists z(\text{spl}'(z) \wedge \exists y(\text{cp}'(y) \wedge \forall x((\text{res}'(x) \wedge \text{of}'(x,y)) \rightarrow \text{see}'(x,z)))$

(2) *Every researcher of a company saw some sample of most products.*

The number of readings can grow exponentially with the number of noun-phrases!

Scope Ambiguities – Problems

- The scope of noun phrases is not determined by the syntactic position in which they occur.
- Divergence between syntactic and semantic structure is a challenge for compositionality and (compositional) semantics constructions.
- Scope ambiguities may lead to a combinatorial explosion of readings.

Compositionality

- The Compositionality Principle as stated earlier:
  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)

Solving the Problem: Basic Idea

(1) *Every student presents a paper.*

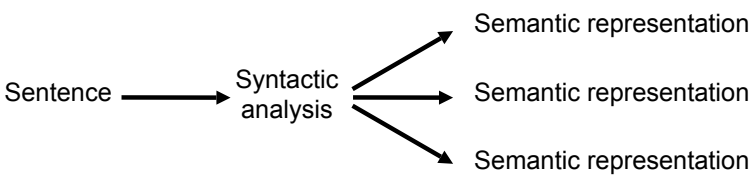
(a) $\forall x(\text{student}'(x) \rightarrow \exists y(\text{paper}'(y) \wedge \text{present}'(x,y)))$

(b) $\exists y(\text{paper}'(y) \wedge \forall x(\text{student}'(x) \rightarrow \text{present}'(x,y)))$

- We can obtain the second reading if we delay the application of the inner noun phrase ("a paper").
  - temporarily store the noun phrase representation away
  - bind the object argument position by a placeholder variable
  - make sure that the correct argment position will be bound, when the "real" noun prase denotation is eventually applied

Nested Cooper Storage

- One algorithm that implements this idea is Nested Cooper Storage (Keller 1988).
- Nested Cooper Storage is an extension of the original Cooper Storage technique (Cooper 1975).
- (Nested) Cooper Storage computes the set of all semantic readings nondeterministically from a single syntactic analysis:



Nested Cooper Storage: Principles

- The semantic values of syntactic constituents are ordered pairs (α, Δ):
  - $\alpha \in WE_\tau$ is the content
  - Δ is the quantifier store: a set of NP representations that must still be applied.
- At NP nodes, we may store the content in Δ.
- At sentence nodes, we can retrieve NP representations from the store in arbitrary order and apply them to the appropriate argument positions.

Nested Cooper Storage: Storage

- Storage: If B is an NP node whose semantic value is (γ, Δ), then $(\lambda P.P(x_i), \{(\gamma, \Delta)_i\})$ is also a semantic value for B, where $i \in N$ is a new index.

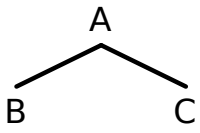
$$\frac{B \Rightarrow (\gamma, \Delta)}{B \Rightarrow (\lambda P.P(x_i), \{(\gamma, \Delta)_i\})}$$

- Using this rule, we can assign more than one semantic value to NP nodes.
- The content of the new semantic value is a placeholder of type ((e,t),t), and the original value (including its store) is moved to the store.

Nested Cooper Storage: Composition Rules

- Rule of functional application

$$\frac{B \Rightarrow (\beta, \Delta) \quad C \Rightarrow (\gamma, \Gamma)}{A \Rightarrow (\beta(\gamma), \Delta \cup \Gamma)} \quad \text{or} \quad \frac{B \Rightarrow (\beta, \Delta) \quad C \Rightarrow (\gamma, \Gamma)}{A \Rightarrow (\gamma(\beta), \Delta \cup \Gamma)}$$



- Rule for non-branching nodes:

$$\frac{B \Rightarrow (\beta, \Delta)}{A \Rightarrow (\beta, \Delta)}$$

- Rule for lexical nodes:

$$\frac{}{A \Rightarrow (\beta, \varnothing)}$$

Nested Cooper Storage

- A syntactic constituent may be associated with multiple semantic values.
- A lambda term M counts as a semantic representation for the entire sentence iff we can derive (M, ∅) as a value for the root of the syntax tree.
- There may be more than one valid semantic representation for the complete sentence.

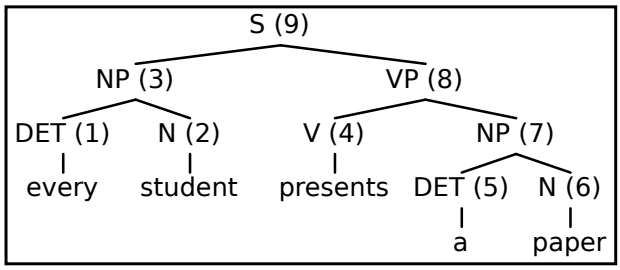
Nested Cooper Storage: Retrieval

- If B is a sentence node, we can retrieve quantifiers from the store:

$$\frac{B \Rightarrow (\alpha, \Delta \cup \{(\gamma, \Gamma)_i\})}{B \Rightarrow (\gamma(\lambda x_i.\alpha), \Delta \cup \Gamma)}$$

- Using this rule, we can apply a previously stored NP.
- At this point, the correct λ-abstraction for the variable associated with the stored element is introduced.
- The old store Γ is released into the store for A.

An Example



(1) $(\lambda F \lambda P \forall x(F(x) \rightarrow P(x)), \emptyset)$

(2) $(\text{student}', \emptyset)$

(3) $(\lambda F \lambda P \forall x(F(x) \rightarrow P(x))(\text{student}'), \emptyset)$
$(\lambda P \forall x(\text{student}'(x) \rightarrow P(x)), \emptyset)$ (β-reduction)
$(\lambda P.P(x_1), \{(\lambda P \forall x(\text{student}'(x) \rightarrow P(x)), \emptyset)_1\})$ (storage)

(4) $(\lambda G \lambda x(G(\lambda y(\text{pres}^*(y)(x)))), \emptyset)$

(7) $(\lambda Q \exists y(\text{paper}'(y) \wedge Q(x)), \emptyset)$
$(\lambda P.P(x_2), \{(\lambda Q \exists y(\text{paper}'(y) \wedge Q(x)), \emptyset)_2\})$ (storage)

(8) $(\lambda G \lambda x(G(\lambda y(\text{pres}^*(y)(x))))(\lambda P.P(x_2)), \{(\lambda Q \exists y(\text{paper}'(y) \wedge Q(x)), \emptyset)_2\})\}$
$(\lambda x.\text{pres}^*(x_2)(x), \{(\lambda Q \exists y(\text{paper}'(y) \wedge Q(x)), \emptyset)_2\})\}$ (β-redcution)

(9) $(\text{pres}^*(x_2)(x_1), \{(\lambda P \forall x(\text{student}'(x) \rightarrow P(x)), \emptyset)_1, (\lambda Q \exists y(\text{paper}'(y) \wedge Q(x)), \emptyset)_2\})$

Retrieval: Reading #1

- By applying the Retrieval rule, we can derive the following representation for the S node:

$$
\begin{array}{ll}
 & (\ \text{pres}^*(x_2)(x_1), \quad \{\ (\ \lambda P \forall x[\text{student}'(x) \rightarrow P(x)],\ \emptyset)_1, \\
 & \qquad\qquad\qquad\qquad\ (\ \lambda Q \exists y[\text{paper}'(y) \wedge Q(y)]\ ,\ \emptyset)_2\ \}) \\
\Rightarrow_R & (\ \lambda Q \exists y[\text{paper}'(y) \wedge Q(y)](\lambda x_2.\text{pres}^*(x_2)(x_1)), \\
 & \qquad\qquad\qquad\qquad \{\ (\ \lambda P \forall x[\text{student}'(x) \rightarrow P(x)],\ \emptyset)_1\ \}) \\
\Rightarrow_\beta & (\ \exists y[\text{paper}'(y) \wedge \text{pres}^*(y)(x_1)], \\
 & \qquad\qquad\qquad\qquad \{\ (\ \lambda P \forall x[\text{student}'(x) \rightarrow P(x)],\ \emptyset)_1\ \}) \\
\Rightarrow_R & (\ \lambda P \forall x[\text{student}'(x) \rightarrow P(x)](\lambda x_1.\exists y[\text{paper}'(y) \wedge \text{pres}^*(y)(x_1)]),\ \emptyset) \\
\Rightarrow_\beta & (\ \forall x[\text{student}'(x) \rightarrow \exists y[\text{paper}'(y) \wedge \text{pres}^*(y)(x)]],\ \emptyset)
\end{array}
$$

Retrieval: Reading #2

- By applying the Retrieval rule, we can derive the following representation for the S node:

$$
\begin{array}{ll}
 & (\ \text{pres}^*(x_2)(x_1), \quad \{\ (\ \lambda P \forall x[\text{student}'(x) \rightarrow P(x)],\ \emptyset)_1, \\
 & \qquad\qquad\qquad\qquad\ (\ \lambda Q \exists y[\text{paper}'(y) \wedge Q(y)]\ ,\ \emptyset)_2\ \}) \\
\Rightarrow_R & (\ \lambda P \forall x[\text{student}'(x) \rightarrow P(x)](\lambda x_1.\text{pres}^*(x_2)(x_1)), \\
 & \qquad\qquad\qquad\qquad \{\ (\ \lambda Q \exists y[\text{paper}'(y) \wedge Q(y)]\ ,\ \emptyset)_2\ \}) \\
\Rightarrow_\beta & (\ \forall x[\text{student}'(x) \rightarrow \text{pres}^*(x_2)(x)], \\
 & \qquad\qquad\qquad\qquad \{\ (\ \lambda Q \exists y[\text{paper}'(y) \wedge Q(y)]\ ,\ \emptyset)_2\ \}) \\
\Rightarrow_R & (\ \lambda Q \exists y[\text{paper}'(y) \wedge Q(y)](\lambda x_2.\forall x[\text{student}'(x) \rightarrow \text{pres}^*(x_2)(x)]),\ \emptyset) \\
\Rightarrow_\beta & (\ \exists y[\text{paper}'(y) \wedge \forall x[\text{student}'(x) \rightarrow \text{pres}^*(y)(x)],\ \emptyset)
\end{array}
$$

Nested Stores

(1) *[Every researcher of a company] saw some sample.*

- Nested stores are needed to model nested NPs as in (1)
- If both NPs are stored, we must make sure that “every researcher (of)” is retrived before “a company.”
  - Otherwise, we would obtain a wrong semantic representation containing a free variable for the complete sentence.
- The nesting of quantifier stores forces the quantifier for the nested NP to take scope over the quantifier for the nesting NP (if both NP-representations are stored).

Compositionality

- The Compositionality Principle as stated earlier:
  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)

Compositionality

- Two versions of the Compositionality Principle:
  - on the level of denotations
  - on the level of semantic representations
- Nested Cooper Storage is clearly compositional on the level of semantic representations - but in a less straightforward way than last week‘s construction algorithm.
- Compositional on the level of denotations: only in a very indirect sense.

Scope Islands

- Nested Cooper Storage makes the simplifying assumption that NPs can be retrieved at all sentence nodes.
- This is not true in general because sentence-embedding verbs create “scope islands:”

  (1) *John said that he saw a girl.* (2 readings)

  (2) *John said that he saw every girl.* (1 reading)
- Non-existential quantifiers may not cross scope island boundaries: The second sentence doesn’t mean “for every girl x, John said that he saw x.”

Summary

- The simple syntax-semantics-interface presented last week cannot deal with semantically ambiguous sentences.
- Scope ambiguity: Application order of NP representations is not determined by the syntactic structure.
- Nested Cooper Storage: Equip semantic representations with a quantifier store to allow flexible application of quantifiers; multiple semantic representations per syntactic constituents allowed.