Semantic Theory

Summer 2005

Scope Ambiguities

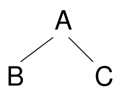
M. Pinkal / A. Koller

Where are we right now?

- Goal: Compositional construction of semantic representations out of syntactic analyses:
  - The meaning of a complex expression is uniquely determined by the meanings of its sub-expressions and its syntactic structure.
- Type theory.
- Assign each syntactic constituent a lambda term; construction rules look at local trees.
- Rules for quantifiers, NPs, intransitive verbs, relative clauses, ...
- Transitive verbs get surprising type.

Some basic rules

- Rule of functional application:



$$\frac{B \Rightarrow \beta: \langle\sigma, \tau\rangle \qquad C \Rightarrow \gamma: \sigma}{A \Rightarrow \beta(\gamma): \tau} \quad \text{or} \quad \frac{B \Rightarrow \beta: \sigma \qquad C \Rightarrow \gamma: \langle\sigma, \tau\rangle}{A \Rightarrow \gamma(\beta): \tau}$$

- Rule of non-branching nodes:

$$\frac{B \Rightarrow \beta: \tau}{A \Rightarrow \beta: \tau}$$

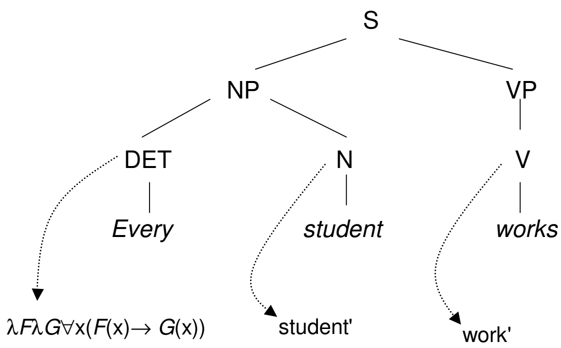
Some basic rules

- Rule of lexical nodes:

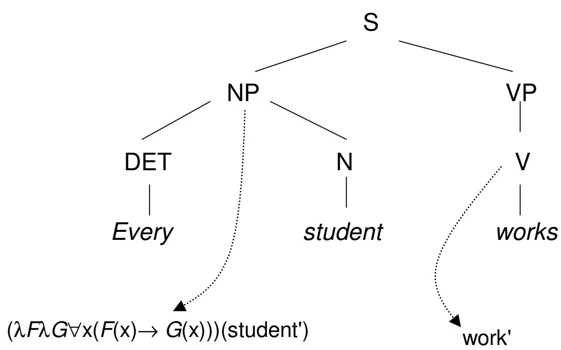
$$\frac{}{A \Rightarrow \beta: \tau}$$

The semantic representation β for the word "a" is supplied by the lexicon.

An example

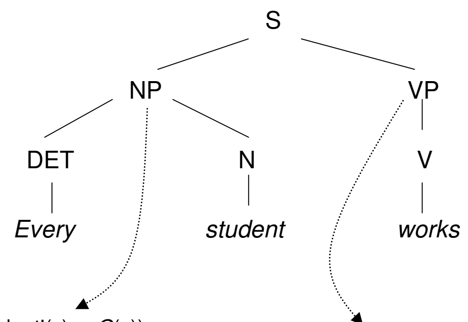


An example



$\Leftrightarrow_\beta$ $\lambda G \forall x(\text{student}'(x) \rightarrow G(x))$

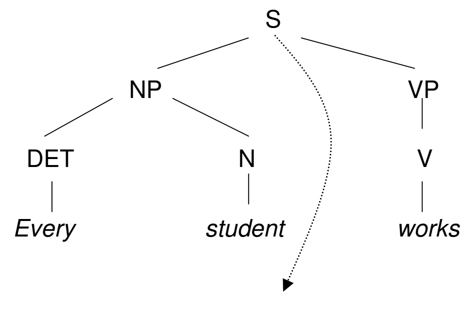
An example



$\lambda G \forall x(\text{student}'(x) \rightarrow G(x))$

work'

An example



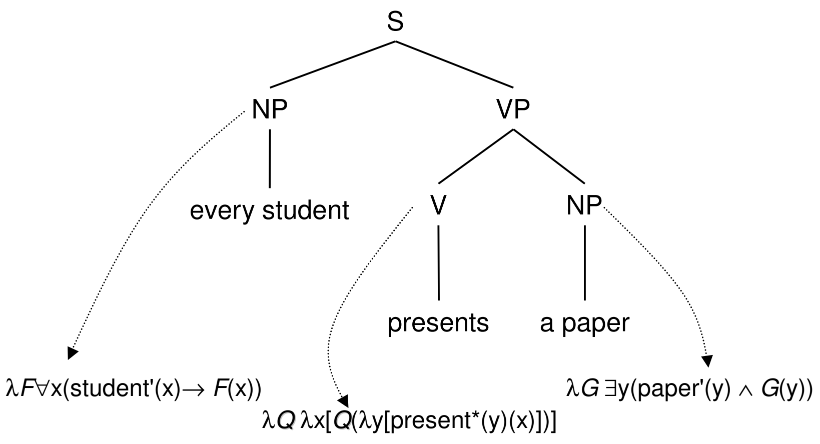
$(\lambda G \forall x(\text{student}'(x) \rightarrow G(x)))(\text{work}')$

$\Leftrightarrow_\beta \forall x(\text{student}'(x) \rightarrow \text{work}(x))$

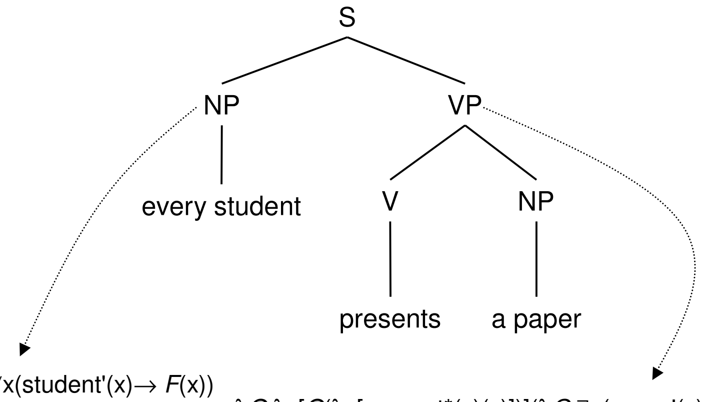
Transitive verbs

- Type-raised analysis of transitive verbs:
  present': <<<e,t>,t>,<e,t>>
- This is necessary because the semantic representation of the transitive verb must be combined with two NPs of type <<e,t>,t>.
- First apply the verb representation to the object representation; then apply the subject representation to the result.

Transitive verbs



Transitive verbs

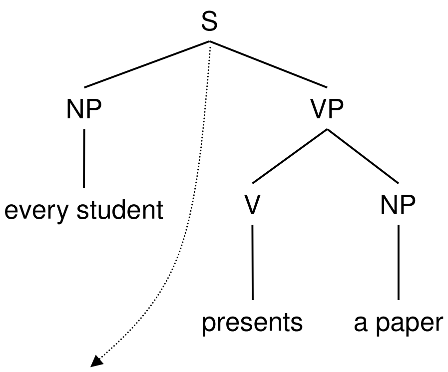


$\lambda F \forall x(\text{student}'(x) \rightarrow F(x))$

$\lambda Q\, \lambda x[Q(\lambda y[\text{present}^*(y)(x)])](\lambda G\, \exists y(\text{paper}'(y) \wedge G(y)))$

$\Leftrightarrow_\beta \lambda x\, \exists y(\text{paper}'(y) \wedge \text{present}^*(y)(x))$

Transitive verbs



$\lambda F \forall x(\text{student}'(x) \rightarrow F(x))(\lambda x\, \exists y(\text{paper}'(y) \wedge \text{present}^*(y)(x)))$

$\Leftrightarrow_\beta \forall x(\text{student}'(x) \rightarrow \exists y(\text{paper}'(y) \wedge \text{present}^*(y)(x)))$

Scope ambiguities

- Some sentences have more than one possible semantic representation:

Every student presents a paper.

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

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

Every student didn't pay attention.

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

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

Scope ambiguities

- The number of readings of a sentence with scope ambiguities grows with the number of NPs:

Every researcher of a company saw some sample.

1. $\forall x(res'(x) \wedge \exists y(cp'(y) \wedge of'(x,y)) \rightarrow \exists z(spl'(z) \wedge see'(x,z))$
2. $\exists z(spl'(z) \wedge \forall x(res'(x) \wedge \exists y(cp'(y) \wedge of'(x,y)) \rightarrow see'(x,z))$
3. $\exists y(cp'(y) \wedge \forall x(res'(x) \wedge of'(x,y)) \rightarrow \exists z(spl'(z) \wedge see'(x,z))$
4. $\exists y(cp'(y) \wedge \exists z(spl'(z) \wedge \forall x(res'(x) \wedge of'(x,y)) \rightarrow see'(x,z))$
5. $\exists z(spl'(z) \wedge \exists y(cp'(y) \wedge \forall x(res'(x) \wedge of'(x,y)) \rightarrow see'(x,z))$

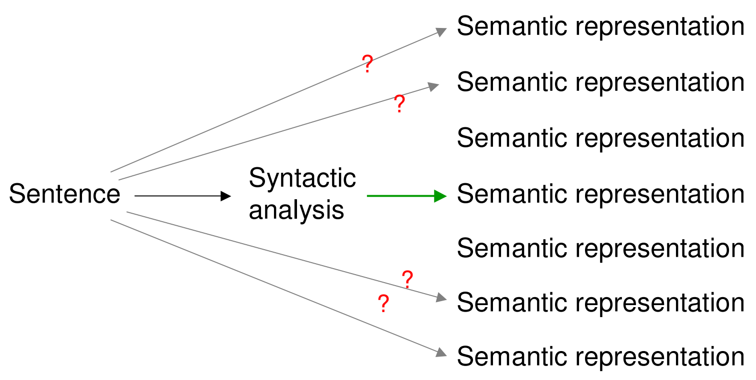
Every researcher of a company saw some samples of most products.

etc.

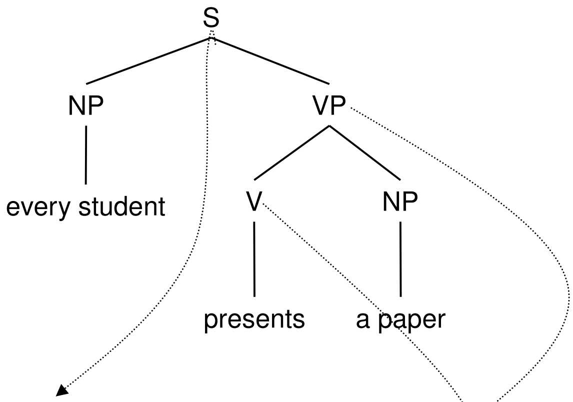
The problem with scope

- Sentences with scope ambiguities can have multiple semantic representations for a syntactic constituent.
- The order of the scope-bearing elements (quantifiers, negation, adverbs, ...) don't necessarily follow the order of the syntactic combination.
- But: With the approach we have so far, we can only derive a single semantic representation for each constituent!
- How can we solve this problem?

Semantic ambiguity: A picture



Solving the scope problem: Intuition



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

$\exists y(\text{paper}'(y) \wedge \text{present}^*(y)(x_1))$

$\text{present}^*(x_2)(x_1)$

The missing reading

- We get one reading of the sentence by deriving the following terms:

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

$\exists y(\text{paper}'(y) \wedge \text{present}^*(y)(x_1))$

$\text{present}^*(x_2)(x_1)$

- We could construct the second reading as follows:

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

$\forall x(\text{student}'(x) \rightarrow \text{present}^*(x_2)(x))$

$\text{present}^*(x_2)(x_1)$

Solving the scope problem: Principles

- Structural ambiguity: We can obtain the two readings by embedding an intermediate term into the NP representations in different orders.
- Invariant variable binding: At the same time, we must make sure that the variables will be bound in the same way in both readings.
- To a certain degree, we can solve both problems using lambda abstraction in a clever way.

Using lambda abstraction

- Intermediate results are all of type t. Abstract over the correct variable and then apply the NP representation to the abstracted term.

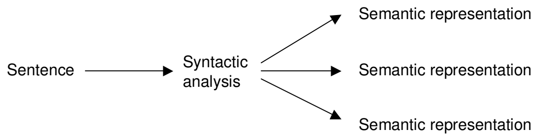
$$\lambda F \forall x(\text{student}'(x) \rightarrow F(x))(\lambda x_1.\ \lambda G \exists y(\text{paper}'(y) \wedge G(y))(\lambda x_2.\text{present}^*(x_2)(x_1)))$$
$$\lambda G \exists y(\text{paper}'(y) \wedge G(y))(\lambda x_2.\text{present}^*(x_2)(x_1))$$
$$\text{present}^*(x_2)(x_1)$$

$$\lambda G \exists y(\text{paper}'(y) \wedge G(y))(\lambda x_2.\ \lambda F \forall x(\text{student}'(x) \rightarrow F(x))(\lambda x_1.\text{present}^*(x_2)(x_1)))$$
$$\lambda F \forall x(\text{student}'(x) \rightarrow F(x))(\lambda x_1.\text{present}^*(x_2)(x_1))$$
$$\text{present}^*(x_2)(x_1)$$

- Problem: How can we do this compositionally?

Nested Cooper Storage

- One algorithm for deriving such representations compositionally is Nested Cooper Storage (Keller 1988). It repairs some problems of the original Cooper Storage (Cooper 1975).
- Cooper Storages compute the set of all semantic readings nondeterministically from a single syntactic analysis:



Nested Cooper Storage: Principles

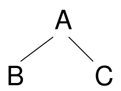
- The semantic values of syntactic constituents are ordered pairs $\langle \alpha, \Delta \rangle$:
  - $\alpha \in WE_\tau$ is the content
  - Δ is the quantifier store: a set of NP representations that must still be applied.
- Rather than applying the representation of an NP immediately, we can store it 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: Principles

- A lambda term M counts as a semantic representation for the syntactic analysis iff we can derive $\langle M, \varnothing \rangle$ as a value for the entire syntax tree.
- Because some rules are nondeterministic, there may be more than one M for which we can derive $\langle M, \varnothing \rangle$.

Nested Cooper Storage: Old Rules

- Rule of functional application:

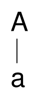


$$\frac{B \Rightarrow \langle \beta, \Delta \rangle \quad C \Rightarrow \langle \gamma, \Gamma \rangle}{A \Rightarrow \langle \beta(\gamma), \Delta \cup \Gamma \rangle} \quad \text{or} \quad \frac{B \Rightarrow \langle \beta, \Delta \rangle \quad C \Rightarrow \langle \gamma, \Gamma \rangle}{A \Rightarrow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

- Rule of non-branching nodes:

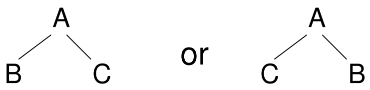
$$\frac{B \Rightarrow \langle \beta, \Delta \rangle}{A \Rightarrow \langle \beta, \Delta \rangle}$$

- Rule of lexical nodes:



$$\frac{}{A \Rightarrow \langle \beta, \varnothing \rangle}$$

Nested Cooper Storage: Storage



$B \Rightarrow \langle \gamma, \Gamma \rangle$ B is an NP node

$C \Rightarrow \langle \beta, \Delta \rangle$ $\beta \in WE_{\langle e, \tau \rangle}$

$A \Rightarrow \langle \beta(x_i), \Delta \cup \{\langle \gamma, \Gamma \rangle_i\} \rangle$, $i \in$ **N** is a new index

Nested Cooper Storage: Retrieval

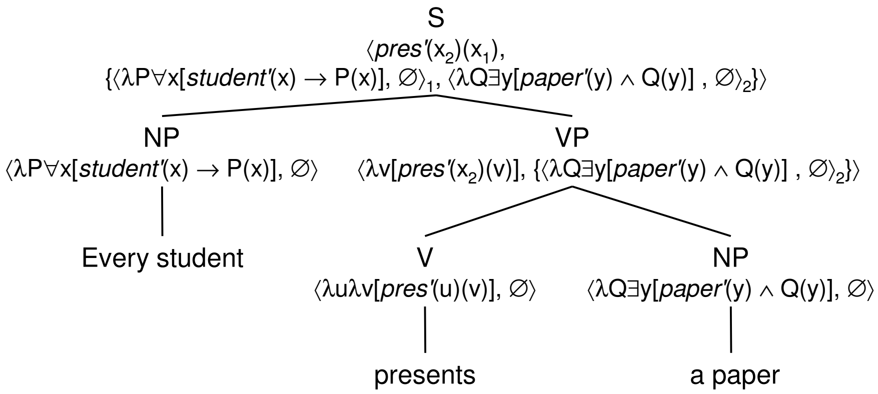
$A \Rightarrow \langle \alpha, \Delta \cup \{\langle \gamma, \Gamma \rangle_i\} \rangle$ A is any sentence node

---

$A \Rightarrow \langle \gamma(\lambda x_i \alpha), \Delta \cup \Gamma \rangle$

Nested Cooper Storage: Example

Every student presents a paper.



Retrieval: Reading 1

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

$\langle pres'(x_2)(x_1), \{\langle \lambda P \forall x[student'(x) \rightarrow P(x)], \varnothing\rangle_1, \langle \lambda Q \exists y[paper'(y) \wedge Q(y)], \varnothing\rangle_2\}\rangle$

$\Rightarrow_R \langle \lambda Q \exists y[paper'(y) \wedge Q(y)](\lambda x_2.pres'(x_2)(x_1)),$
$\{\langle \lambda P \forall x[student'(x) \rightarrow P(x)], \varnothing\rangle_1\}\rangle$

$\Rightarrow_\beta \langle \exists y[paper'(y) \wedge pres'(y)(x_1)], \{\langle \lambda P \forall x[student'(x) \rightarrow P(x)], \varnothing\rangle_1\}\rangle$

$\Rightarrow_R \langle \lambda P \forall x[student'(x) \rightarrow P(x)](\lambda x_1.\exists y[paper'(y) \wedge pres'(y)(x_1)]), \varnothing\rangle$

$\Rightarrow_\beta \langle \forall x[student'(x) \rightarrow \exists y[paper'(y) \wedge pres'(y)(x)]], \varnothing\rangle$

Retrieval: Reading 2

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

$\langle pres'(x_2)(x_1), \{\langle \lambda P \forall x[student'(x) \rightarrow P(x)], \varnothing \rangle_1, \langle \lambda Q \exists y[paper'(y) \wedge Q(y)], \varnothing \rangle_2\}\rangle$

$\Rightarrow_R \langle \lambda P \forall x[student'(x) \rightarrow P(x)] (\lambda x_1.pres'(x_2)(x_1)),$
$\{\langle \lambda Q \exists y[paper'(y) \wedge Q(y)], \varnothing \rangle_2\}\rangle$

$\Rightarrow_\beta \langle \forall x[student'(x) \rightarrow pres'(x_2)(x)], \{\langle \lambda Q \exists y[paper'(y) \wedge Q(y)], \varnothing \rangle_2\}\rangle$

$\Rightarrow_R \langle \lambda Q \exists y[paper'(y) \wedge Q(y)](\lambda x_2.\forall x[student'(x) \rightarrow pres'(x_2)(x)]), \varnothing \rangle$

$\Rightarrow_\beta \langle \exists y[paper'(y) \wedge \forall x[student'(x) \rightarrow pres'(y)(x)]], \varnothing \rangle$

Compositionality

- The Compositionality Principle as stated earlier:
  The meaning of a complex expression is uniquely determined by the meanings of its sub-expressions and its syntactic structure.
- Nested Cooper Storage shows: We can maintain this principle even in the face of semantic (scope) ambiguity.

Compositionality and NCS

- 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.

Other types of scope ambiguities

- Nested Cooper Storage makes the simplifying assumption that only NPs can participate in scope ambiguities.
- This is not true in general:
  - Every student didn't pay attention.
  - Sometimes every student is sleepy.
- NCS could probably be extended to deal with these, but we'll do something better next week anyway.

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:
  - John said that he saw a girl. (2 readings)
  - John said that he saw every girl. (1 reading)

De dicto/de re ambiguities

- De dicto/de re ambiguities are a special kind of scope ambiguity in which one scope bearer is a verb:

  Gerhard Schröder wants to visit a car factory.

  $\exists x.\text{factory}(x) \wedge \text{want}(gs, {}^\wedge\text{visit}(gs,x))$ (de re)

  $\text{want}(gs, {}^\wedge\exists x.\text{factory}(x) \wedge \text{visit}(gs,x))$ (de dicto)
- Are we talking about a specific or just any arbitrary factory? Does the sentence claim that a factory exists?
- We need a more expressive (intensional) logic to represent the different readings, but the ambiguity is just a scope ambiguity and can be resolved by NCS.

Scope ambiguities in the real world

- Scope ambiguities are not a very intuitive type of ambiguity, and are sometimes not seen as a serious problem for computational linguistics.
- In practice, they are often resolved by context, world knowledge, preferences, etc.
- We consider them here because they pose a fundamental challenge for semantics construction.
- If we want "deep" semantic representations that say something about scope, we must take scope ambiguities into account.

Scope ambiguities in the real world

- Also, some large-scale grammars (e.g. the English Resource Grammar) compute semantic representations with scope.
- The ERG analyses all NPs as scope bearers to keep the grammar simple. (This is not necessarily correct: proper names, definites, etc.)
- Median number of scope readings in the Rondane corpus: 55.

Conclusion

- Last week's type-driven semantics construction is a nice first step.
- But it is fundamentally unable to deal with semantically ambiguous sentences.
- Scope ambiguity: Application order of NP representations can be different from syntactic structure.
- Nested Cooper Storage: Equip semantic representations with a quantifier store to allow flexible application of quantifiers.