Semantic Theory

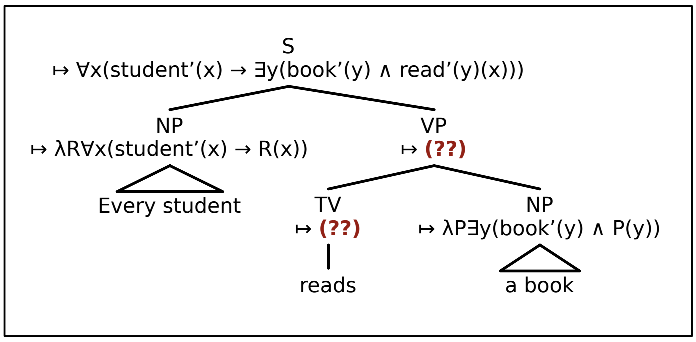
Lecture 4: Cooper Storage

Manfred Pinkal & Stefan Thater
FR 4.7 Allgemeine Linguistik (Computerlinguistik)
Universität des Saarlandes

Summer 2010

Transitive Verbs

- *Every student reads a book*
  - $\forall x(\text{student}'(x) \rightarrow \exists y(\text{book}'(y) \wedge \text{read}'(y)(x)))$



Transitive Verbs (1st attempt)

- *read* ↦ $\text{read}' \in WE_{\langle\langle\langle e,t\rangle, t\rangle, \langle e, t\rangle\rangle}$
- *read a book* ↦ $\text{read}'(\lambda P\exists y(\text{book}'(y) \wedge P(y)) \in WE_{\langle e, t\rangle}$
- *every student reads a book*
  - $\mapsto \lambda R\forall x(\text{student}'(x) \rightarrow R(x))(\text{read}'(\lambda P\exists y(\text{book}'(y) \wedge P(y)))$
  - $\Leftrightarrow \forall x(\text{student}'(x) \rightarrow \text{read}'(\lambda P\exists y(\text{book}'(y) \wedge P(y)))(x))$
- **Problem:** without an additional meaning postulate the formula does not capture the truth-conditions of the sentence.

Transitive Verbs (final version)

- **Solution:**
  - use a more explicit λ-term for transitive verbs
- *read* $\mapsto \lambda Q \lambda z Q(\lambda x(\text{read}'(x)(z))) \in WE_{\langle\langle\langle e,t\rangle, t\rangle, \langle e, t\rangle\rangle}$
  - Note: $\text{read}' \in WE_{\langle e, \langle e, t\rangle\rangle}$
- *read a book*
  - $\mapsto \lambda Q \lambda z Q(\lambda x(\text{read}'(x)(z)))(\lambda P \exists y(\text{book}'(y) \wedge P(y)))$
  - $\Leftrightarrow_\beta \lambda z(\lambda P \exists y(\text{book}'(y) \wedge P(y))(\lambda x(\text{read}'(x)(z))))$
  - $\Leftrightarrow_\beta \lambda z(\exists y(\text{book}'(y) \wedge \lambda x(\text{read}'(x)(z))(y)))$
  - $\Leftrightarrow_\beta \lambda z(\exists y(\text{book}'(y) \wedge \text{read}'(y)(z)))$

Transitive Verbs (final version)

- **Solution:**
  - use a more explicit λ-term for transitive verbs
- *read a book*
  - $\mapsto \lambda z \exists y(\text{book}'(y) \wedge \text{read}'(y)(z))$
- *every student*
  - $\mapsto \lambda R \forall x(\text{student}'(x) \rightarrow R(x))$
- *every student reads a book*
  - $\mapsto \lambda R \forall x(\text{student}'(x) \rightarrow R(x))(\lambda z \exists y(\text{book}'(y) \wedge \text{read}'(y)(z)))$
  - $\Leftrightarrow_\beta \forall x(\text{student}'(x) \rightarrow \lambda z \exists y(\text{book}'(y) \wedge \text{read}'(y)(z))(x))$
  - $\Leftrightarrow_\beta \forall x(\text{student}'(x) \rightarrow \exists y(\text{book}'(y) \wedge \text{read}'(y)(x)))$

Scope Ambiguities

(1) *Every student reads a book*
a. $\forall x(\text{student}'(x) \rightarrow \exists y(\text{book}'(y) \wedge \text{read*}(y)(x)))$
b. $\exists y(\text{book}'(y) \wedge \forall x(\text{student}'(x) \rightarrow \text{read*}(y)(x)))$

(2) *Every student didn't pay attention*
a. $\forall x(\text{student}'(x) \rightarrow \neg\text{pay-attention}'(x))$
b. $\neg\forall x(\text{student}'(x) \rightarrow \text{pay-attention}'(x))$

(3) *Some inhabitant of every midwestern city participated*

(4) *An American flag stood in front of every building*

(5) *John searches a good book about semantics*

(6) *Pola wants to marry a millionaire*

Scope Ambiguities

- Using the semantics construction rules from the previous lecture, we can derive only one reading for sentences exhibiting a scope ambiguity.
  - Assumption: the sentence has a unique syntactic structure.
- Quantifier scope is not determined by the syntactic position in which the corresponding NP occurs.
- Mismatch between syntactic and semantic structure is a challenge for compositional semantics construction.

Cooper Storage

- **Cooper-Storage** is a technique to derive different readings of sentences exhibiting a scope ambiguity
- The different readings are derived by using a **single, surface-based syntactic structure**

Semantic representation

Sentence $\longrightarrow$ Syntactic analysis $\longrightarrow$ $\vdots$

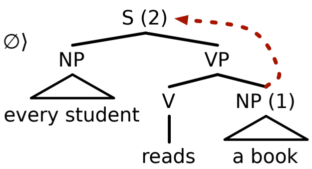
Semantic representation

Cooper Storage

- Natural language expressions are assigned ordered pairs $\langle \alpha, \Delta \rangle$ as semantic values:
  - $\boldsymbol{\alpha} \in \mathbf{WE}_{\tau}$ is the content
  - $\boldsymbol{\Delta} \subseteq \mathbf{WE}_{\langle\langle e,t\rangle,t\rangle}$ is the quantifier store
- Quantifiers (NPs) can either apply *in situ,* or they can be moved to the store for later application (“storage”).
- At sentence nodes, quantifiers can be removed from the store and applied to the content (“retrieval”).
- A term α counts as a semantic representation for a sentence if we can derive $\langle \alpha, \emptyset \rangle$ as its semantic value.

The basic idea

- **Storage at (1)**
  $\langle \lambda G \exists x(bk(x) \wedge G(x)), \emptyset \rangle \Rightarrow$
  $\langle \lambda F.F(\mathbf{x_1}), \{\mathbf{[}\lambda G \exists x(bk(x) \wedge G(x))\mathbf{]_1}\} \rangle$
- **Retrieval at (2)**
  $\langle \forall y(st(y) \rightarrow rd(\mathbf{x_1})(y)), \{\mathbf{[}\lambda G \exists x(bk(x) \wedge G(x))\mathbf{]_1}\} \rangle \Rightarrow$
  $\langle \lambda G \exists x(bk(x) \wedge G(x))(\boldsymbol{\lambda} \mathbf{x_1}(\forall y(st(y) \rightarrow rd(\mathbf{x_1})(y)), \emptyset \rangle$
- **After β-reduction:**
  $\langle \exists x(bk(x) \wedge \forall y(st(y) \rightarrow rd(x)(y))), \emptyset \rangle$



Sample Grammar

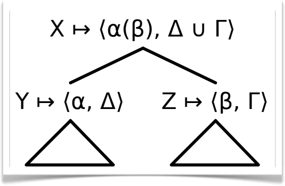
S → NP VP	PN → *Bill* | *John* | …
NP → DET N’	DET → *every* | *the* | …
NP → PN	N → *student* | *book* | …
N’ → N	P → *of*
N’ → N PP	TV → *likes* | *reads* | …
VP → IV	IV → *works* | *sleeps* | …
VP → TV NP	
PP → P NP	

Semantic Lexicon

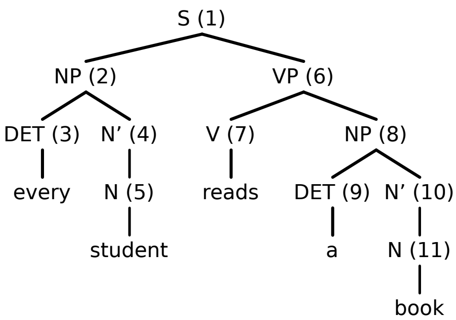
Bill ↦ $\lambda F(F(b^*))$	$\in WE_{\langle\langle e,t\rangle,t\rangle}$
every ↦ $\lambda F \lambda G \forall x(F(x) \rightarrow G(x))$	$\in WE_{\langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle}$
a ↦ $\lambda F \lambda G \exists x(F(x) \wedge G(x))$	$\in WE_{\langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle}$
sleeps ↦ sleep’	$\in WE_{\langle e,t\rangle}$
student ↦ student’	$\in WE_{\langle e,t\rangle}$
reads ↦ $\lambda Q \lambda x(Q(\lambda y(read^*(y)(x))))$	$\in WE_{\langle\langle\langle e,t\rangle,t\rangle, \langle e,t\rangle\rangle}$
of ↦ [⇒ *exercise*]	$\in WE_{\langle\langle\langle e,t\rangle,t\rangle, \langle\langle e,t\rangle, \langle e,t\rangle\rangle}$

Semantic Construction [1/3]

- **X → Y Z** or **X → Z Y**
  - if Y ↦ ⟨α, Δ⟩, $\alpha \in WE_{\langle\sigma,\tau\rangle}$
  - and Z ↦ ⟨β, Γ⟩, $\beta \in WE_{\sigma}$
  - then X ↦ ⟨α(β), Δ ∪ Γ⟩
- **X → Y**
  - if Y ↦ ⟨α, Δ⟩
  - then X ↦ ⟨α, Δ⟩
- **X → w**
  - X ↦ ⟨α, ∅⟩, where α = SemLex(w)



Every student reads a book



Every student reads a book

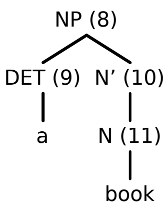
(9) ⟨λFλG∃x(F(x) ∧ G(x)), ∅⟩

(11) ⟨book’, ∅⟩

(10) ⟨book’, ∅⟩

(8) ⟨λFλG∃x(F(x) ∧ G(x))(book’), ∅⟩

$\Leftrightarrow_{\beta}$ ⟨λG∃x(book’(x) ∧ G(x)), ∅⟩



Semantic Construction [2/3]

- **Storage:** $\langle Q, \Delta \rangle \Rightarrow_S \langle \lambda P.P(\mathbf{x_i}), \Delta \cup \{\mathbf{[}Q\mathbf{]_i}\} \rangle$
  - if A is an noun phrase whose semantic value is $\langle Q, \Delta \rangle$, then $\langle \lambda P.P(x_i), \Delta \cup \{[Q]_i\} \rangle$ is also a semantic value for A, where $i \in N$ is a new index.
  - The original content is moved to the store.
  - The new content is a placeholder of type $\langle \langle e,t \rangle,t \rangle$
- **Note:** by using this rule, we can assign more than one semantic value to noun phrases.

Every student reads … (cont'd)

(9) $\langle \lambda F \lambda G \exists x(F(x) \wedge G(x)), \varnothing \rangle$

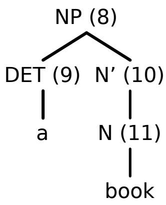
(10) $\langle book', \varnothing \rangle$

(11) $\langle book', \varnothing \rangle$

(8) $\langle \lambda F \lambda G \exists x(F(x) \wedge G(x))(book'), \varnothing \rangle$

$\Leftrightarrow_\beta \langle \lambda G \exists x(book'(x) \wedge G(x)), \varnothing \rangle$

→ $\Rightarrow_S \langle \lambda P.P(\mathbf{x_1}), \{\mathbf{[}\lambda G \exists x(book'(x) \wedge G(x))\mathbf{]_1}\} \rangle$



Every student reads … (cont'd)

→ (8) $\langle \lambda P.P(\mathbf{x_1}), \{\mathbf{[}\lambda G \exists x(book'(x) \wedge G(x))\mathbf{]_1}\} \rangle$

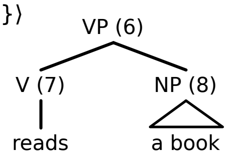
(7) $\langle \lambda Q \lambda x(Q(\lambda y(read^*(y)(x)))), \varnothing \rangle$

(6) $\langle \boldsymbol{\lambda Q} \lambda x(\mathbf{Q}(\lambda y(read^*(y)(x))))\boldsymbol{(\lambda P.P(x_1))}, \{\mathbf{[}\lambda G \exists x(\ldots)\mathbf{]_1}\} \rangle$

$\Leftrightarrow_\beta \langle \lambda x(\boldsymbol{\lambda P}(\mathbf{P}(\mathbf{x_1}))\boldsymbol{(\lambda y(read^*(y)(x)))}), \{\mathbf{[}\lambda G \exists x(\ldots)\mathbf{]_1}\} \rangle$

$\Leftrightarrow_\beta \langle \lambda x(\boldsymbol{\lambda y}(read^*(\mathbf{y})(x))\boldsymbol{(x_1)}), \{\mathbf{[}\lambda G \exists x(\ldots)\mathbf{]_1}\} \rangle$

$\Leftrightarrow_\beta \langle \lambda x(read^*(\mathbf{x_1})(x)), \{\mathbf{[}\lambda G \exists x(\ldots)\mathbf{]_1}\} \rangle$



Every student reads … (cont'd)

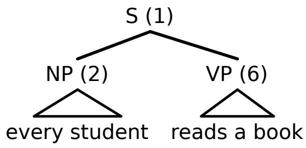
(6) $\langle \lambda x(\text{read*}(\mathbf{x_1})(x)), \{\mathbf{[}\lambda G \exists x(\text{student'}(x) \land G(x))\mathbf{]_1}\} \rangle$

(2) $\langle \lambda G \forall y(\text{student'}(y) \rightarrow G(y)), \varnothing \rangle$

(1) $\langle \boldsymbol{\lambda G} \forall y(\text{student'}(y) \rightarrow \mathbf{G}(y))\boldsymbol{(\lambda x(\text{read*}(x_1)(x)))}, \{\mathbf{[...]_1}\} \rangle$

$\Leftrightarrow_\beta \langle \forall y(\text{student'}(y) \rightarrow \boldsymbol{\lambda x}(\text{read*}(x_1)(\mathbf{x}))\mathbf{(y)}), \{\mathbf{[...]_1}\} \rangle$

$\Leftrightarrow_\beta \langle \forall y(\text{student'}(y) \rightarrow \text{read*}(\mathbf{x_1})(y)), \{\mathbf{[...]_1}\} \rangle$



Semantic Construction [3/3]

- **Retrieval:** $\langle \alpha, \Delta \cup \{\mathbf{[}Q\mathbf{]_i}\} \rangle \Rightarrow_R \langle Q(\boldsymbol{\lambda x_i}\ \alpha), \Delta \rangle$
  - if A is any sentence with semantic value $\langle \alpha, \Delta \cup \{[Q]_i\} \rangle$, then $\langle Q(\lambda x_i\ \alpha), \Delta \rangle$ is also a semantic value for A.
  - Notation: read "∪" as "disjoint union"

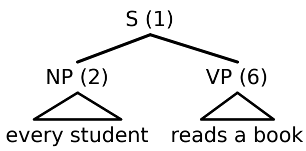
Every student reads … (cont'd)

(1) $\langle \forall y(\text{student'}(y) \rightarrow \text{read*}(\mathbf{x_1})(y)), \{\mathbf{[}\lambda G \exists x(\ldots)\mathbf{]_1}\} \rangle$

$\Rightarrow_R \langle \lambda G \exists x(\text{book'}(x) \land G(x))(\boldsymbol{\lambda x_1}(\forall y(\ldots\ \mathbf{x_1}\ \ldots))), \varnothing \rangle$

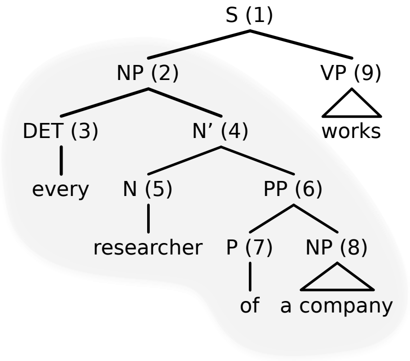
$\Leftrightarrow_\beta \langle \exists x(\text{book'}(x) \land \boldsymbol{\lambda x_1}(\forall y(\ldots\ \mathbf{x_1}\ \ldots))(x)), \varnothing \rangle$

$\Leftrightarrow_\beta \langle \exists x(\text{book'}(x) \land \forall y(\text{student'}(y) \rightarrow \text{read*}(x)(y))), \varnothing \rangle$



Problem: Nested noun phrases

(1) *Every researcher of a company works*



Problem: Nested noun phrases

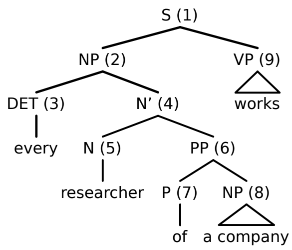
(8) $\langle \lambda F(F(\mathbf{x_1})), \{\mathbf{[}\lambda G \exists x(comp(x) \wedge G(x))\mathbf{]_1}\}\rangle$

(4) $\langle \lambda x(res(x) \wedge of(\mathbf{x_1})(x)), \{\mathbf{[...]_1}\}\rangle$

(2) $\langle \lambda G \forall y((res(y) \wedge of(\mathbf{x_1})(y)) \rightarrow G(y)), \{\mathbf{[...]_1}\}\rangle$

$\Rightarrow_S \langle \lambda F(F(\mathbf{x_2})), \{\mathbf{[}\lambda G \forall y((res(y) \wedge of(\mathbf{x_1})(y)) \rightarrow G(y))\mathbf{]_2}, \mathbf{[...]_1}\}\rangle$

(1) $\langle work(\mathbf{x_2}), \{\mathbf{[...]_2}, \mathbf{[...]_1}\}\rangle$



Problem: Nested noun phrases

$\langle work(x_2), \{ \mathbf{[}Q_2 = \lambda G \forall y((res(y) \wedge of(\mathbf{x_1})(y)) \rightarrow G(y))\mathbf{]_2},$
$\mathbf{[}Q_1 = \lambda G \exists x(comp(x) \wedge G(x))\mathbf{]_1}\}\rangle$

$\Rightarrow_R \langle Q_1(\lambda x_1.work(x_2)), \{\mathbf{[}Q_2\mathbf{]_2}\}\rangle$

$\Leftrightarrow_\beta \langle \exists x(comp(x) \wedge work(x_2)), \{\mathbf{[}Q_2\mathbf{]_2}\}\rangle$

$\Rightarrow_R \langle Q_2(\lambda x_2.\exists x(comp(x) \wedge work(x_2))), \emptyset\rangle$

$\Leftrightarrow_\beta \langle \forall y((res(y) \wedge of(\mathbf{x_1})(y)) \rightarrow \exists x(comp(x) \wedge work(y))), \emptyset\rangle$

Not a reading! Variable x_1 occurs free!

Problem: Nested noun phrases

- The unstructered store does not reflect the dependencies between quantifiers in complex noun phrases like „every [reasearcher of a company]“
- ⇒ quantifiers can be retrieved in any order!
- $\langle \text{work}(x_2), \{\mathbf{[}\lambda G \forall y(\ldots\ \mathbf{x_1} \ldots))\mathbf{]_2}, \mathbf{[}\lambda G \exists x(\ldots)\mathbf{]_1}\}\rangle$
  - **We want:** Q_1 cannot be retrieved if Q_2 is still on the store

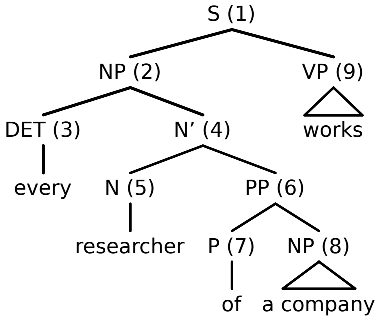
Nested Cooper Storage

- **Storage:** $\langle Q, \Delta\rangle \Rightarrow_S \langle \lambda P.P(\mathbf{x_i}), \{\boldsymbol{\langle} Q, \Delta\boldsymbol{\rangle_i}\}\rangle$
  - If A is a noun phrase whose semantic value is $\langle Q, \Delta\rangle$, then $\langle \lambda P.P(x_i), \{\langle Q, \Delta\rangle_i\}\rangle$ is also a semantic value for A, where $i \in N$ is a new index.
- The original semantic value **including its store** is moved to the store.

Nested Cooper Storage

- **Retrieval:** $\langle \alpha, \Delta \cup \{\boldsymbol{\langle} Q, \Gamma\boldsymbol{\rangle_i}\}\rangle \Rightarrow \langle Q(\boldsymbol{\lambda x_i}\ \alpha), \Delta \cup \Gamma\rangle$
  - If A is a sentence with semantic value $\langle \alpha, \Delta \cup \{\langle Q, \Gamma\rangle_i\}\rangle$, then $\langle Q(\lambda x_i.\alpha), \Delta \cup \Gamma\rangle$ is also a semantic value of the sentence.
  - ⇒ nested stores are **not accessible** for retrieval

Every reasearcher of a …



Every reasearcher of a …

(8) $\langle \lambda G \exists x(comp(x) \wedge G(x)), \emptyset \rangle$

$\Rightarrow_S \langle \lambda F.F(x_1), \{\langle Q_1 = \lambda G(\exists x(comp(x) \wedge G(x)), \emptyset\rangle_1\}\rangle$

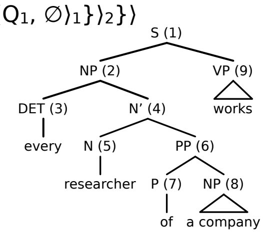
(4) $\langle \lambda y(res(y) \wedge of(x_1)(y)), \{\langle Q_1, \emptyset\rangle_1\}\rangle$

(2) $\langle \lambda G \forall z((res(z) \wedge of(x_1)(z)) \rightarrow G(z)), \{\langle Q_1, \emptyset\rangle_1\}\rangle$

$\Rightarrow_S \langle \lambda F.F(x_2), \{\langle Q_2 = \lambda G \forall z(\ldots), \{\langle Q_1, \emptyset\rangle_1\}\rangle_2\}\rangle$

(9) $\langle work, \emptyset \rangle$

(1) $\langle work(x_2), \{\langle Q_2, \{\langle Q_1, \emptyset\rangle_1\}\rangle_2\}\rangle$



Every reasearcher of a …

$\langle work(x_2), \{\langle Q_2, \{\langle Q_1, \emptyset\rangle_1\}\rangle_2\}\rangle$

$\Rightarrow_R \langle Q_2 (\lambda x_2.work(x_2)), \{\langle Q_1, \emptyset\rangle_1\}\rangle$

$\Leftrightarrow_\beta \langle \forall z((res(z) \wedge of(x_1)(z)) \rightarrow work(z)), \{\langle Q_1, \emptyset\rangle_1\}\rangle$

$\Rightarrow_R \langle Q_1(\lambda x_1.\forall z((res(z) \wedge of(x_1)(z)) \rightarrow work(z))), \emptyset\rangle$

$\Leftrightarrow_\beta \langle \exists x(comp(x) \wedge \forall z((res(z) \wedge of(x)(z)) \rightarrow work(z))), \emptyset\rangle$

Every reasearcher of a …

$\langle \text{work}(x_2), \{\langle \lambda G \forall z(\ldots), \{\langle \lambda G \exists x(\ldots), \emptyset\rangle_1\}\rangle_2\}\rangle$

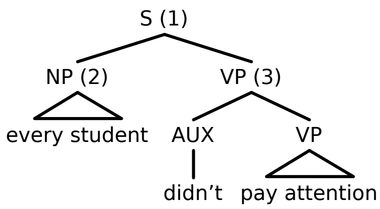
$\Rightarrow^*_R \exists x(\text{comp}(x) \wedge \forall z((\text{res}(z) \wedge \text{of}(x)(z)) \rightarrow \text{work}(z)))$

No other reading can be derived!

- But how do we derive the “direct scope” reading?
- Simple answer: don’t store, apply quantifiers “in situ”

Can we derive all readings?

- Storing a quantifier means to “move it upwards” in the syntax tree (roughly speaking).
- *Every student did not pay attention*
  - “Every student” is higher in the tree than the negation
  - ⇒ the negation cannot take scope over “every student”



Some restrictions on scope

(1) *Some inhabitant of every midwestern city participated*

- two readings: (a) direct scope and (b) every ⊲* some

(2) *Someone who inhabits every midwestern city participated*

- only the direct scope reading available

Finite clauses can create “scope islands”

- Quantifiers must take scope within such clauses

(see Ruys & Winter, 2008)

Some restrictions on scope

(1) *You will inherit a fortune if every man dies*

- "every man" cannot take scope over complete sentence

(2) *If a friend of mine from Texas had died in a fire, I would have inherited a fortune* (Fodor & Sag 1982)

- "a friend of mine from Texas" can take wide scope

Finite clauses can create "scope islands"

- Quantifiers must take scope within such clauses
- Indefinites can "escape" scope islands

Compositionality

- **Denotations** ("D-compositionality")
  The denotation of a complex expression is a function of the denotations its parts.
- **Semantic representations** ("S-compositionality")
  The semantic representation of a complex expression is a function of the semantic representations of its parts.

Compositionality

- Storage techniques are (up to non-determinism) compositional on the level of semantic representations.
- But are not compositional on the level of denotations: Semantic values $\langle \alpha, \Delta \rangle$ don't receive an interpretation.

Literature

- Patrick Blackburn, Johan Bos (2005): Representation and Inference for Natural Language. A First Course in Computational Semantics. CSLI Press.
- W. R. Keller (1988). Nested Cooper storage: The proper treatment of quantification in ordinary noun phrases. In Reyle, Rohrer (Ed.). Natural Language Parsing and Linguistic Theories
- E. G. Ruys, Yoad Winter (2008). Quantifier scope in formal linguistics. To appear in: Handbook of Philosophical Logic, 2nd Edition.

Next Lecture: Underspecification

- Markus Egg, Alexander Koller, Joachim Niehren (2001). The constraint-language for lambda structures. Journal of Logic, Language, and Information.
- Patrick Blackburn, Johan Bos (2005): Representation and Inference for Natural Language. A First Course in Computational Semantics. CSLI Press.