Semantic Theory

Lecture 4 – Cooper Storage

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

Summer 2013

Semantics Sonstruction (recap)

- **Semantic lexicon**
  - maps words to semantic representations (type theory)
- **Semantics construction rules**
  - tell for each syntactic rule $X \rightarrow Y_1\ Y_2$ how to combine the semantic represenatations of Y_1 and Y_2 to obtain a semantic representation for X
  - we assume here that there is only a single operation to combine meaning representation: functional application
- **Note:** all syntactic categories (N, V, NP, VP, …) are mapped to semantic representations with the same type
  - all N's have type ⟨e, t⟩, all NP's have type ⟨⟨e, t⟩, t⟩, …

Semantics Sonstruction (recap)

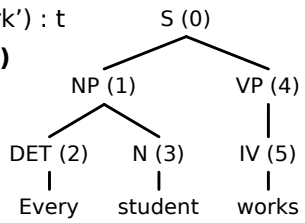
(2) ↦ λPλQ∀x(P(x) → Q(x)) : ⟨⟨e, t⟩, ⟨⟨e, t⟩, t⟩⟩

(3) ↦ student' : ⟨e, t⟩

(1) ↦ λPλQ∀x(P(x) → Q(x))(student'): ⟨⟨e, t⟩, t⟩
⇒$_\beta$ λQ∀x(student'(x) → Q(x))

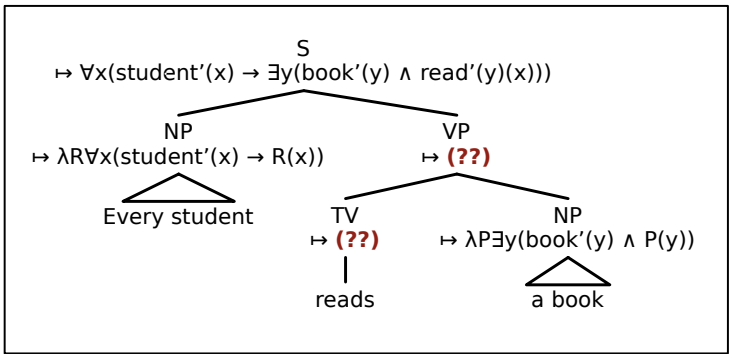
(4) = (5) ↦ work' : ⟨e, t⟩

(0) ↦ λQ∀x(student'(x) → Q(x))(work') : t
⇒$_\beta$ **∀x(student'(x) → work'(x))**



Transitive Verbs

- *Every student reads a book*
  - ∀x(student'(x) → ∃y(book'(y) ∧ read'(y)(x))



Transitive Verbs (1st attempt)

- *read* ↦ read' ∈ $WE_{\langle\langle e,t\rangle, t\rangle, \langle e, t\rangle\rangle}$
- *read a book* ↦ read'(λP∃y(book'(y) ∧ P(y)) ∈ $WE_{\langle e, t\rangle}$
- *every student reads a book*
  - ↦ λR∀x(student'(x) → R(x))(read'(λP∃y(book'(y) ∧ P(y)))
  - ⇔ ∀x(student'(x) → read'(λP∃y(book'(y) ∧ 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* ↦ λQλzQ(λx(read*(x)(z))) ∈ $WE_{\langle\langle e,t\rangle, t\rangle, \langle e, t\rangle\rangle}$
  - Note: read* ∈ $WE_{\langle e, \langle e, t\rangle\rangle}$
- *read a book*
  - ↦ λQλzQ(λx(read*(x)(z)))(λP∃y(book'(y) ∧ P(y)))
  - $\Leftrightarrow_\beta$ λz(λP∃y(book'(y) ∧ P(y))(λx(read*(x)(z))))
  - $\Leftrightarrow_\beta$ λz(∃y(book'(y) ∧ λx(read*(x)(z))(y)))
  - $\Leftrightarrow_\beta$ λz(∃y(book'(y) ∧ read*(y)(z)))

Transitive Verbs (final version)

- **Solution:**
  - use a more explicit λ-term for transitive verbs
- *read a book*
  - ↦ λz∃y(book'(y) ∧ read*(y)(z))
- *every student*
  - ↦ λR∀x(student'(x) → R(x))
- *every student reads a book*
  - ↦ λR∀x(student'(x) → R(x))(λz∃y(book'(y) ∧ read*(y)(z)))
  - $\Leftrightarrow_\beta$ ∀x(student'(x) → λz∃y(book'(y) ∧ read*(y)(z))(x))
  - **$\Leftrightarrow_\beta$ ∀x(student'(x) → ∃y(book'(y) ∧ read*(y)(x)))**

Scope Ambiguities

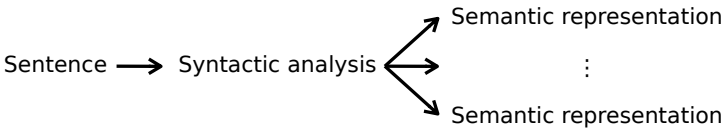
- *Every student reads a book*
  a. ∀x(student'(x) → ∃y(book'(y) ∧ read*(y)(x)))
  b. ∃y(book'(y) ∧ ∀x(student'(x) → read*(y)(x)))
- *Every student didn't pay attention*
  a. ∀x(student'(x) → ¬pay-attention'(x))
  b. ¬∀x(student'(x) → pay-attention'(x))
- *Some inhabitant of every midwestern city participated*
- *An American flag stood in front of every building*
- *John searches a good book about semantics*
- *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.
  - (… if 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**



Cooper Storage

- Natural language expressions are assigned ordered pairs ⟨α, Δ⟩ as semantic values:
  - **α ∈ WE$_\tau$** is the content
  - **Δ ⊆ 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 ⟨α, ∅⟩ as its semantic value.

The basic idea

- **Storage at (1)**

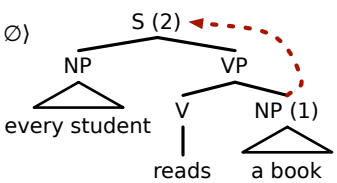
  ⟨λGλx(bk(x) ∧ G(x)), ∅⟩ ⇒

  ⟨λF.F(**x$_1$**), {**[**λGλx(bk(x) ∧ G(x))**]$_1$**}⟩
- **Retrieval at (2)**

  ⟨∀y(st(y) → rd(**x$_1$**)(y)), {**[**λGλx(bk(x) ∧ G(x))**]$_1$**}⟩ ⇒

  ⟨λGλx(bk(x) ∧ G(x))(**λx$_1$**(∀y(st(y) → rd(**x$_1$**)(y)), ∅⟩
- **After β-reduction:**

  ⟨λx(bk(x) ∧ ∀y(st(y) → rd(x)(y))), ∅⟩



Sample Grammar

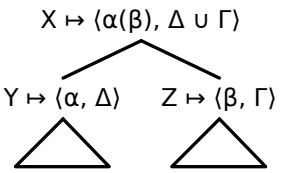
S → NP VP	PN → *Bill* | *John* | …
NP → DET N’	DET → *every* | *a* | some
NP → PN	N → *student* | *book* | …
N’ → N	P → *of* | *at* | …
N’ → N PP	TV → *reads* | *likes* | …
VP → IV	IV → *works* | *sleeps* | …
VP → TV NP	
PP → P NP	

Semantic Lexicon

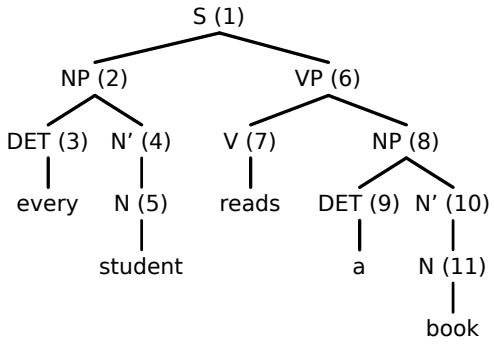
Bill ↦ λF(F(b*))	$\in WE_{\langle\langle e,t\rangle,t\rangle}$
every ↦ λFλG∀x(F(x) → G(x))	$\in WE_{\langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle}$
a ↦ λFλG∃x(F(x) ∧ G(x))	$\in WE_{\langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle}$
works ↦ work’	$\in WE_{\langle e,t\rangle}$
student ↦ student’	$\in WE_{\langle e,t\rangle}$
book ↦ book’	$\in WE_{\langle e,t\rangle}$
university ↦ university’	$\in WE_{\langle e,t\rangle}$
reads ↦ λQλx(Q(λy(read*(y)(x))))	$\in WE_{\langle\langle\langle e,t\rangle,t\rangle,\langle e,t\rangle\rangle}$
of, at ↦ [⇒ *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

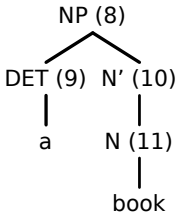
(9) ⟨λFλGλx(F(x) ∧ G(x)), ∅⟩

(11) ⟨book’, ∅⟩

(10) ⟨book’, ∅⟩

(8) ⟨λFλGλx(F(x) ∧ G(x))(book’), ∅⟩

⇔$_\beta$ ⟨λGλx(book’(x) ∧ G(x)), ∅⟩



Semantic Construction [2/3]

- **Storage:** ⟨Q, Δ⟩ ⇒$_S$ ⟨λP.P(**x$_i$**), Δ ∪ {**[**Q**]$_i$**}⟩
  - if A is an noun phrase whose semantic value is ⟨Q, Δ⟩, then ⟨λP.P(x$_i$), Δ ∪ {[Q]$_i$}⟩ is also a semantic value for A, where i ∈ N is a new index.
  - The original content is moved to the store.
  - The new content is a placeholder of type ⟨⟨e,t⟩,t⟩
- **Note:** by using this rule, we can assign more than one semantic value to a noun phrase.

Every student reads … (cont'd)

(9) ⟨λFλGλx(F(x) ∧ G(x)), ∅⟩

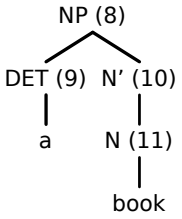
(10) ⟨book', ∅⟩

(11) ⟨book', ∅⟩

(8) ⟨λFλGλx(F(x) ∧ G(x))(book'), ∅⟩

⇔β ⟨λGλx(book'(x) ∧ G(x)), ∅⟩

→ ⇒S ⟨λP.P(**x₁**), {**[**λGλx(book'(x) ∧ G(x))**]₁**}⟩



Every student reads … (cont'd)

→ (8) ⟨λP.P(**x₁**), {**[**λGλx(book'(x) ∧ G(x))**]₁**}⟩

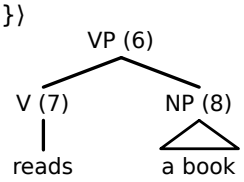
(7) ⟨λQλx(Q(λy(read*(y)(x)))), ∅⟩

(6) ⟨**λQ**λx(**Q**(λy(read*(y)(x))))**(λP.P(x₁))**, {**[**λGλx(…)**]₁**}⟩

⇔β ⟨λx(**λP**(**P**(**x₁**))**(λy(read*(y)(x)))**), {**[**λGλx(…)**]₁**}⟩

⇔β ⟨λx(**λy**(read*(**y**)(x))**(x₁)**), {**[**λGλx(…)**]₁**}⟩

⇔β ⟨λx(read*(**x₁**)(x)), {**[**λGλx(…)**]₁**}⟩



Every student reads … (cont'd)

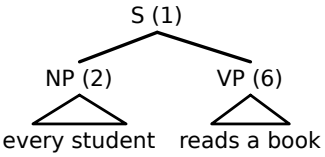
→ (6) ⟨λx(read*(**x₁**)(x)), {**[**λGλx(book'(x) ∧ G(x))**]₁**}⟩

(2) ⟨λGλy(student'(y) → G(y)), ∅⟩

(1) ⟨**λG**λy(student'(y) → **G**(y))**(λx(read*(x₁)(x)))**, {**[…]₁**}⟩

⇔β ⟨λy(student'(y) → **λx**(read*(x_1)(**x**))**(y)**), {**[…]₁**}⟩

⇔β ⟨λy(student'(y) → read*(**x₁**)(y)), {**[…]₁**}⟩



Semantic Construction [3/3]

- **Retrieval:** $\langle \alpha, \Delta \cup \{[Q]_i\}\rangle \Rightarrow_R \langle Q(\boldsymbol{\lambda x_i}\ \alpha), \Delta\rangle$
  - 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 "$\cup$" as "disjoint union"

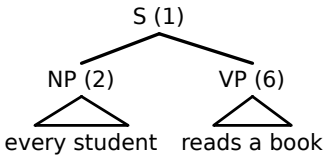
Every student reads … (cont'd)

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

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

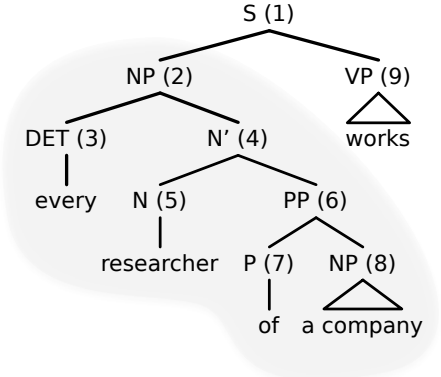
$\Leftrightarrow_\beta \langle \exists x(\text{book}'(x) \wedge \boldsymbol{\lambda x_1}(\forall y(\ldots\ \mathbf{x_1}\ \ldots))(x)), \varnothing\rangle$

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



Problem: Nested noun phrases

- *Every researcher of a company works*



Problem: Nested noun phrases

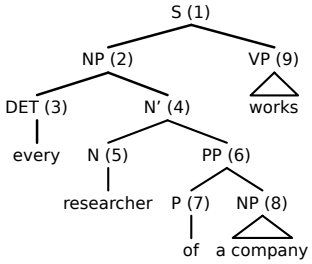
(8) ⟨λF(F(**x₁**)), {**[**λG∃x(comp(x) ∧ G(x))**]₁**}⟩

(4) ⟨λx(res(x) ∧ of(**x₁**)(x)), {**[…]₁**}⟩

(2) ⟨λG∀y((res(y) ∧ of(**x₁**)(y)) → G(y)), {**[…]₁**}⟩

⇒$_S$ ⟨λF(F(**x₂**)),{**[**λG∀y((res(y) ∧ of(**x₁**)(y)) → G(y))**]₂**, **[…]₁**}⟩

(1) ⟨work(**x₂**), {**[…]₂**, **[…]₁**}⟩



Problem: Nested noun phrases

⟨work(x_2), { **[**Q_2 = λG∀y((res(y) ∧ of(**x₁**)(y)) → G(y))**]₂**,
[Q_1 = λG∃x(comp(x) ∧ G(x))**]₁**}⟩

⇒$_R$ ⟨Q_1(λx_1.work(x_2)), {**[**Q_2**]₂**}⟩

⇔$_\beta$ ⟨∃x(comp(x) ∧ work(x_2)), {**[**Q_2**]₂**}⟩

⇒$_R$ ⟨Q_2(λx_2.∃x(comp(x) ∧ work(x_2))), ∅⟩

⇔$_\beta$ ⟨∀y((res(y) ∧ of(**x₁**)(y)) → ∃x(comp(x) ∧ work(y))), ∅⟩

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!
- ⟨work(x_2), {**[**λG∀y(… **x₁** …))**]₂**, **[**λG∃x(…)**]₁**}⟩
  - **We want:** Q_1 cannot be retrieved if Q_2 is still on the store

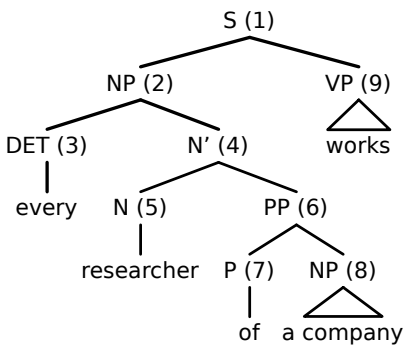
Nested Cooper Storage

- **Storage:** $\langle Q, \Delta \rangle \Rightarrow_S \langle \lambda P.P(\mathbf{x_i}), \{\mathbf{\langle} Q, \Delta \mathbf{\rangle_i}\} \rangle$
  - 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:** $\langle \alpha, \Delta \cup \{\mathbf{\langle} Q, \Gamma \mathbf{\rangle_i}\} \rangle \Rightarrow \langle Q(\mathbf{\lambda x_i}\ \alpha), \Delta \cup \Gamma \rangle$
  - 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) $\langle \lambda G \exists x(comp(x) \wedge G(x)), \emptyset \rangle$

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

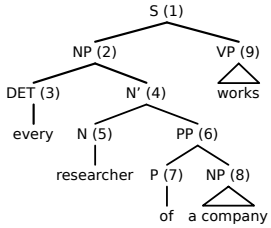
(4) $\langle \lambda y(res(y) \wedge of(x_1)(y)), \{\langle Q_1, \emptyset \rangle_1\} \rangle$

(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 …

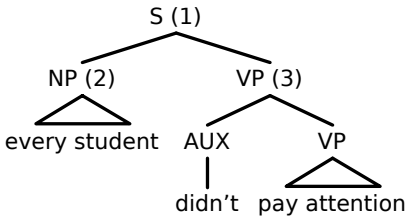
$\langle work(x_2), \{\langle \lambda G \forall z(\ldots), \{\langle \lambda G \exists x(\ldots), \emptyset \rangle_1\} \rangle_2\} \rangle$

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

- **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”



(see Ruys & Winter, 2008)

Some restrictions on scope

- *Some inhabitant of every midwestern city participated*
  - two readings: (a) direct scope and (b) every ◁* some
- *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

- *You will inherit a fortune if every man dies*
  - “every man” cannot take scope over complete sentence
- *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 ⟨α, Δ⟩ 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.