Computational Semantics
Lorraine University NLP Master, 2013 – 2014

Sylvain Pogodalla¹

¹mailto:sylvain.pogodalla@inria.fr
http://www.loria.fr/~pogodall
Office: C. 302
LORIA/INRIA Nancy–Grand Est
France

2013 – 2014
Outline

1. Introduction
2. Meaning
3. Types and Model Structure
4. Montague Semantics
5. Phenomena at the Syntax-Semantics Interface
6. Abstract Categorial Grammars
7. Underspecification
8. Discourse
9. Selected Bibliography
1 Introduction

2 Meaning

3 Types and Model Structure

4 Montague Semantics

5 Phenomena at the Syntax-Semantics Interface

6 Abstract Categorial Grammars

7 Underspecification

8 Discourse

9 Selected Bibliography
Computational Semantics?

What is computational semantics?
- A suitable interpretation level
- A way to relate semantic structures and syntax (interpretation or realization)
Introduction

Meaning
- Sense and Denotation
- Models and Truth-Conditionality Criterion [Winter, 2010]
- Building Denotations
- Structural Ambiguity

Types and Model Structure

Montague Semantics

Phenomena at the Syntax-Semantics Interface

Abstract Categorial Grammars

Underspecification

Discourse

Selected Bibliography
Sense and Denotation

Gottlob Frege

- Sense = mode of presentation
- Denotation = the object it refers to

Ex.: 1 + 1 and 2 have the same denotation but not the same sense
Meaning as an Observable Property [Winter, 2010]

- Select a specific property of language, stable across speakers and situations
- Extra-linguistic and intra-linguistic semantic phenomena

**Example (Extra-linguistic)**

- What is common to the objects that people categorize as being red?
- What the effect of asking *please pick a blue card from this pack*?

**Example (Intra-Linguistic)**

- How do speaker identify relations between pairs of words like *red-color, dog-animal*?
- What are the relations between the sentences *please pick a blue card from this pack* and *please pick a card from this pack*?

- **Contrasts**

**Example**

- Red is a color / ?Red is an animal
- Every red thing has a color / ?Every red thing has an animal
Entailment

Example

- Tina is tall and thin
- Tina is thin

Example

- Tina is tall and thin $\Rightarrow$ Tina is thin
- A dog entered the room $\Rightarrow$ An animal entered the room
- John picked a blue card from this pack $\Rightarrow$ John picked a card from this pack

Stability of judgments

Indefeasible reasoning (vs. *Tina is a bird. Tina can fly*)

Entailment judgements $\equiv$ grammaticality judgments

Models and denotation
Models and Truth-Conditionality Criterion

Aim

Establishing a relation between language expressions and objects in models.

- Formal semantics
- Applied semantics

Linking $exp$ and $[exp]^M$ where $M$ is a model.

Denotations of sentences are **truth-values** 1 (for “true”) and 0 (for “false”)

Definition (TCC)

A semantic theory $T$ satisfies the **truth-conditionality criterion** (TCC) if for all sentences $S_1$ and $S_2$ the following conditions are equivalent:

1. Sentences $S_1$ intuitively entails sentence $S_2$
2. For all models $M$ in $T$ $[S_1]^M \leq [S_2]^M$

- 1 is an **empirical** statement
- 2 is a **theoretical** statement
Models and Truth-Conditionality Criterion (cont’d)

\[ S_1 \Rightarrow S_2 \]

Models

\[
\begin{align*}
[S_1]^{M_1} & \leq [S_2]^{M_1} \\
[S_1]^{M_2} & \leq [S_2]^{M_2} \\
[S_1]^{M_3} & \leq [S_2]^{M_3} \\
\ldots
\end{align*}
\]
Building Denotations

For every model $M$:

- In addition to truth-values, there exists $E_M$ an arbitrary non-empty set of entities in $M$
- $Tina$ denotes an arbitrary entity $tina = \llbracket Tina \rrbracket^M$ in $E_M$
- $tall$ and $thin$ denote arbitrary sets tall $= \llbracket tall \rrbracket^M$ and thin $= \llbracket thin \rrbracket^M$ of entities in $E_M$

What does it mean that $Tina$ is thin is true?

- $tina \in thin$
- $\text{is}(x, A) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases}$
- $\llbracket Tina$ is thin $\rrbracket^M = \text{is}(tina, thin)$

**Constants**

\textbf{is} is \textit{constant} across models!

Exercise: What’s the denotation of $Tina$ is tall and thin?
**Tina is tall and thin**

- \( \text{and}(A, B) = A \cap B \)
- \( \llbracket \text{Tina is tall and thin} \rrbracket^M = \text{is}(\text{tina}, \text{and}(\text{tall}, \text{thin})) \)

### Denotations

<table>
<thead>
<tr>
<th>Expressions</th>
<th>Category</th>
<th>Type</th>
<th>Abstract denotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tina</td>
<td>NP</td>
<td>entity</td>
<td>tina</td>
</tr>
<tr>
<td>tall</td>
<td>A</td>
<td>set of entities</td>
<td>tall</td>
</tr>
<tr>
<td>thin</td>
<td>A</td>
<td>set of entities</td>
<td>thin</td>
</tr>
<tr>
<td>tall and thin</td>
<td>AP</td>
<td>set of entities</td>
<td>and(tall, thin)</td>
</tr>
<tr>
<td>Tina is thin</td>
<td>S</td>
<td>truth-value</td>
<td>is(tina)</td>
</tr>
<tr>
<td>Tina is tall and thin</td>
<td>S</td>
<td>truth-value</td>
<td>is(tina, and(tall, thin))</td>
</tr>
</tbody>
</table>

### Example

We assume our semantic theory satisfies the TCC. Show that the interpretations given so far account for the following facts:

- \( \text{Tina is tall and thin} \Rightarrow \text{Tina is thin} \)
- \( \text{Tina is thin} \not\Rightarrow \text{Tina is tall and thin} \)

See [Hopcroft et al., 2003, first chapter, and in particular Section 1.2.2 “Reduction to Definitions”] for an introduction to rigorous (formal) proofs.
Involving Syntactic Structures

Example

- All pianists are composers and Tina is a pianist.
- All composers are pianists and Tina is a pianist.
- Therefore, Tina is a composer.

Compositionality

The denotation of a complex expression is determined by the denotations of its immediate parts and the ways they combine with each other.
Example

```
S
|  NP
|   |  is
|   |   AP
|   |     A
|   |       A
|   |         tall
|   |         thin
```

```
is(tina, and(tall, thin))
  |  tina
  |    is
  |      and
  |   |  tall
  |   |    and
  |   |      thin
```

About Compositionality

- Direct compositionality
- Some syntactic assumptions
- How the denotations of functions know what their arguments are?
Structural Ambiguity

**Example (Tina is not tall and thin)**

\[ Tina \text{ is not tall and thin} \Rightarrow Tina \text{ is thin} \]

- What makes the syntactic ambiguity a semantic ambiguity? **Compositionality**
- What's the denotation of *not*?
- What are the denotations of the two structures?
Structural Ambiguity (cont’s)

is(tina, and(not(tall), thin))

- tina
  - not
  - tall
- is
  - and
  - not(tall)
  - thin

is(tina, not(and(tall, thin)))

- tina
  - is not
  - and
  - tall
  - and
  - thin
  - not
  - and
  - tall
  - and
  - thin

- A
  - and
  - thin

- AP
  - and
  - A
  - thin

- NP
  - Tina
  - is
  - AP
  - not
  - AP
  - tall
Two sentences are called *equivalent* if they entail each other as in the following example:

- *Tina is tall and thin* ⇔ *Tina is both tall and thin*

1. Give more examples for pairs of sentences that you consider intuitively for equivalent sentences
2. State the condition that the TCC requires for equivalent sentences
3. Assuming the structure \([\text{both}[tall \text{ and thin}]]\), how can we define \([\text{both}]^M\)? Is it a constant?
4. Consider the ungrammaticality of the following strings:
   - *Tina is both tall*
   - *Tina is both not tall*
   - *Tina is both tall or thin*

Let’s assume that *both* only appears in adjective phrases as adjacent to *and* conjunctions.

1. Analyze the truth-values assigned to:
   - *Tina is both not tall and thin*
   - *Tina is not both tall and thin*
2. Show that this accounts for:
   - *Tina is both not tall and thin* ⇒ *Tina is thin*
   - *Tina is not both tall and thin* \(\not\Rightarrow\) *Tina is thin*
Consider the following sentences:

- *Tina is not tall and not thin*
- *Tina is neither tall nor thin*

Assume that $[\text{neither}]^M = [\text{both}]^M$? What should then the denotation of *nor* be to have these two sentences equivalent?
Two sentences are called *contradictory* in a given theory if whenever one of them denotes 1 in the theory, the other denotes 0 (e.g. *Mary is not tall* and *Mary is tall*). We may also talk about two contradictory readings/structures of sentences, which is especially useful when sentences are structurally ambiguous.

1. Give more examples for contradictory sentences/structures
2. Consider the sentences *The bottle is empty* and *The bottle is full*. Can you think of a theoretical assumption that would render these sentences contradictory?
3. Give an entailment using this assumption
Introduction

Meaning

Types and Model Structure
- Domains
- Types
- Type Theory and $\lambda$-Calculus
- Higher-Order Logic

Montague Semantics

Phenomena at the Syntax-Semantics Interface

Abstract Categorial Grammars

Underspecification

Discourse

Selected Bibliography
### Types and Domains

#### Denotations

<table>
<thead>
<tr>
<th>Expressions</th>
<th>Category</th>
<th>Type</th>
<th>Abstract denotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tina</td>
<td>NP</td>
<td>entity</td>
<td>tina</td>
</tr>
<tr>
<td>tall</td>
<td>A</td>
<td>set of entities</td>
<td>tall</td>
</tr>
<tr>
<td>tall and thin</td>
<td>AP</td>
<td>set of entities</td>
<td>and (tall, thin)</td>
</tr>
<tr>
<td>Tina is thin</td>
<td>S</td>
<td>truth-value</td>
<td>is (tina, thin)</td>
</tr>
<tr>
<td>is</td>
<td>?</td>
<td>function from entities and set of</td>
<td>is</td>
</tr>
<tr>
<td>and</td>
<td>?</td>
<td>entities to truth-values</td>
<td>and</td>
</tr>
<tr>
<td>not</td>
<td>?</td>
<td>function from set of entities to set</td>
<td>not</td>
</tr>
<tr>
<td></td>
<td></td>
<td>of entities</td>
<td></td>
</tr>
</tbody>
</table>

- Systematic relation between expressions of a given syntactic category to a type of denotation
- Distinction between the objects of the model (entities, set of entities, functions, etc.)

⇒ **Domains**: parts of the model that gathers objects with the same structure

- The property for objects to belong to the same domain is expressed by having same type
Definition (Characteristic Function)

Let $A$ be a subset of $B$. A function $F_A$ from $B$ to $\{0, 1\}$ the set of truth-values is called the characteristic function of $A$ in $B$ if it satisfies for every $x \in B$:

$$F_A(x) = \begin{cases} 
1 & \text{if } x \in A \\
0 & \text{if } x \notin A 
\end{cases}$$

Types and Domains

<table>
<thead>
<tr>
<th>Basic types</th>
<th></th>
<th>Complex types</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td><strong>Domain</strong></td>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>$e$</td>
<td>$E = D_e$</td>
<td>$e \rightarrow t$</td>
</tr>
<tr>
<td>$t$</td>
<td>${0, 1} = D_t$</td>
<td>...</td>
</tr>
</tbody>
</table>

Example

Let $M$ be a model such that $D_e = \{t, j, m\}$. How many possible denotations for *tall* are there?
Types and Model Structure

Types

Definition (Types)

The set of types $\mathcal{T}$ over the basic types $e$ and $t$ is the smallest set such that:

- $\{e, t\} \subseteq \mathcal{T}$
- If $\tau \in \mathcal{T}$ and $\sigma \in \mathcal{T}$ then $\tau \rightarrow \sigma \in \mathcal{T}$

Example

- What’s the type of an adjective?
- What’s the type of not?
- What’s the type of is?
Two-Place Relation

One-Place Predicate

*Tina smiled* is true whenever \( \text{tina} \in \text{smiled} \)

whenever \( F_{\text{smiled}}(\text{tina}) = 1 \)

whenever \( \text{smiled}(\text{tina}) = 1 \)

\( \text{smiled} \in F_{e \rightarrow t} = D_{t}^{D_{e}} \) or \( \text{smiled} : e \rightarrow t \)

Two-Place Relation

*Tina praised Mary* is true whenever \( \text{tina} \) belongs to the set of entities that

*praised Mary*

whenever \( J \text{praised Mary} K(\text{tina}) = 1 \)

\( J \text{praised} K \in D_{t}^{D_{e}} \) or \( J \text{praised} K : e \rightarrow t \)

\( J \text{praised} \) \( K \in D_{t}^{D_{e}D_{e}} \) or \( J \text{praised} \) \( K : e \rightarrow e \rightarrow t \)

Note

- \( \llbracket \text{Tina [praised Mary]} \rrbracket = (\text{praised}(\text{mary}))(\text{tina}) \)
- Characteristic function of a binary relation \( R: (F_{R}(y))(x) = 1 \) iff \( \langle x, y \rangle \in R \)
- Currying: \( f : A \times B \rightarrow C \) and \( g : A \rightarrow B \rightarrow C \) such that \( f(a, b) = (g(a))(b) \)
**Definition (Frame)**

Let $E = D_e$ a set of entities. We define the frame $F^E$ as:

$$F^E \triangleq \bigcup_{\tau \in \mathcal{T}} D_{\tau}$$

**Definition (Lexicon)**

Let $\Sigma$ be a finite vocabulary. A lexical typing function $T_{\mathcal{L}}$ of $\Sigma$ is any function from $\Sigma$ to $\mathcal{T}$.

Given a lexical typing function $T_{\mathcal{L}}$, a corresponding lexical interpretation function over $\Sigma$ and a non-empty set of entities $E$ is any function $I_{\mathcal{L}}$ from $\Sigma$ to $F^E$ such that:

$$\forall w \in \Sigma \quad I_{\mathcal{L}}(w) : T_{\mathcal{L}}(w)$$

**Definition (Model)**

Let $\Sigma$ be a finite vocabulary with $T_{\mathcal{L}}$ a lexical typing function over $\Sigma$. A model over $\Sigma$ is a pair $\langle E, I_{\mathcal{L}} \rangle$ where $E$ is a non-empty set of entities and $I_{\mathcal{L}}$ is a lexical interpretation function over $\Sigma$ and $E$. 
Example

<table>
<thead>
<tr>
<th>Vocabulary</th>
<th>$\Sigma = { \text{Tina, Mary, smiled, praised} }$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typing</td>
<td>$T_L : \Sigma \rightarrow T$</td>
</tr>
<tr>
<td></td>
<td>Tina $\rightarrow e$</td>
</tr>
<tr>
<td></td>
<td>Mary $\rightarrow e$</td>
</tr>
<tr>
<td></td>
<td>smiled $\rightarrow e \rightarrow t$</td>
</tr>
<tr>
<td></td>
<td>praised $\rightarrow e \rightarrow e \rightarrow t$</td>
</tr>
<tr>
<td>Interpretation</td>
<td>$I_L : \Sigma \rightarrow F^E$</td>
</tr>
<tr>
<td></td>
<td>Tina $\rightarrow \text{tina}$</td>
</tr>
<tr>
<td></td>
<td>Mary $\rightarrow \text{mary}$</td>
</tr>
<tr>
<td></td>
<td>smiled $\rightarrow \begin{cases} tina &amp; \rightarrow 1 \ mary &amp; \rightarrow 0 \end{cases}$</td>
</tr>
<tr>
<td></td>
<td>praised $\rightarrow \begin{cases} tina &amp; \rightarrow 1 \ mary &amp; \rightarrow 1 \ tina &amp; \rightarrow 1 \ mary &amp; \rightarrow 0 \end{cases}$</td>
</tr>
<tr>
<td>Lexicon</td>
<td>$\langle \Sigma, T_L \rangle$</td>
</tr>
<tr>
<td>Model</td>
<td>$\langle E, I_L \rangle$</td>
</tr>
</tbody>
</table>

- What’s the denotation of *Mary smiled*?  
- What’s the denotation of *Mary praised Mary*?
About the Lexicon

Definition (Restricting Functor)

Let \( \Sigma \) be a finite vocabulary and \( T_\varphi \) be a lexical typing function from \( \Sigma \) to \( \mathcal{T} \). Let \( E \) be a non-empty set.

A restricting functor \( \mathcal{RF}^E \) over \( \Sigma \) is a function that maps any word \( w \in \Sigma \) to a subset of the domain \( D_{T_\varphi}(w) \).

For \( w \in \Sigma \):
- if \( \mathcal{RF}^E(w) = D_{T_\varphi}(w) \) for every set \( E \), we say that the denotation of \( w \) is **arbitrary**
- if \( \mathcal{RF}^E(w) \) is a singleton for every set \( E \), we say that the denotation of \( w \) is **constant**

- Have we seen **constants**?
- What about “intermediary” kinds of lexical items?
  - \( \text{SYM}^E \) is the set of symmetric functions in \( D_{e\rightarrow e\rightarrow t} \) (\( f \) is symmetric iff \( f(x)(y) = f(y)(x) \))
  - \( \text{RMOD}^E \) is the set of restrictive modifiers in \( D_{(e\rightarrow t)\rightarrow (e\rightarrow t)} \) where \( f \in \text{RMOD}^E \) iff \( \forall g \in D_{e\rightarrow t} \forall x \in D_e \), if \( (f(g))(x) = 1 \) then \( g(x) = 1 \) as well
- Do you have examples?
  - \textit{resemble}
  - \textit{very}
  - \textit{charmingly}
Definition (Intended Model)

Let $\Sigma$ be a finite vocabulary with $T_\mathcal{L}$ a lexical typing function over $\Sigma$. Let $E$ be a non-empty set and $RF^E$ a restricting functor over $\Sigma$. A model $\langle E, I_\mathcal{L} \rangle$ over $\Sigma$ is an intended model if for every word $w \in \Sigma$ $I_\mathcal{L}(w) \in RF^E(w)$.

Definition (Truth-Conditionality Criterion)

A semantic theory $T$ that specifies a typing function $T_\mathcal{L}$ and a restricting functor $RF^E$ over $\Sigma$ satisfies the truth-conditionality criterion (TCC) if for all structures $S_1$ and $S_2$ the following conditions are equivalent:

1. Structure $S_1$ intuitively entails structure $S_2$
2. For all intended models $M$ in $T$ $[S_1]^M \leq [S_2]^M$
Definition (λ-Calculus)

**Syntax** \( \mathcal{V} = \{x, y, \ldots\} \) and \( T ::= \mathcal{V} | \lambda \mathcal{V}. T | (T T) \)

**Free Variables** Let \( t \in T \)
- \( \text{FV}(x) = \{x\} \)
- \( \text{FV}(\lambda x. u) = \text{FV}(u) \setminus \{x\} \)
- \( \text{FV}(t u) = \text{FV}(t) \cup \text{FV}(u) \)

If \( \text{FV}(t) = \emptyset \) \( t \) is **closed**.

Example

What are the free variables of
- \( x \ y \ z \)
- \( \lambda x. x \ y \)
- \( (\lambda x. x \ x)(\lambda y. y \ y) \)
Definition (Substitution)

For $t, u \in T$ and $x \in \mathcal{V}$, the substitution of $u$ for $x$ in $t$, written $t[x := u] \in T$, is defined as follows ($x \neq y$):

- $x[x := u] = u$
- $y[x := u] = y$
- $(v \, w)[x := u] = v[x := u] \, w[x := u]$
- $(\lambda x. v)[x := u] = \lambda x. v$
- $(\lambda y. v)[x := u] = \lambda y. v[x := u] \text{ with } y \notin \text{FV}(u)$

Example

Compute $((\lambda x. x \, y \, z)(\lambda y. x \, y \, z)(\lambda z. x \, y \, z))[x := y]$
Reminder (cont’d)

**Definition (Reduction)**

Reduction \((\lambda x.t) \ u \rightarrow_\beta \ t[x := u]\)

**Church-Rosser Theorem** For all \(\lambda\)-terms \(t, u\) and \(v\) such that

\[ t \rightarrow^*_\beta u \text{ and } t \rightarrow^*_\beta v \]

there exists \(w\) such that

\[ u \rightarrow^*_\beta w \text{ and } v \rightarrow^*_\beta w \]

**Example**

Let \(\delta = \lambda x.x\).x. Reduce \(\Omega = \delta \ \delta\).
Reminder (cont’d)

Definition (Simply Typed \( \lambda \)-Calculus)

\[ \Gamma = x_1 : A_1, \ldots, x_n : A_n \] is a context.

\[ \frac{}{\Gamma, x : A \vdash x : A} \] (Axiom)

\[ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \rightarrow B} \] (Abs.)

\[ \frac{\Gamma \vdash t : A \rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash (t \ u) : B} \] (App.)

Example

- What about the types of \( \lambda x.x \)? Of \( \lambda xy.x \)? Of \( \lambda x.\lambda y.\lambda z.x \, z (y \, z) \)?
- Can you type \( \delta = \lambda x.x \, x \)?
Syntactic Structures

Definition (Binary Structure)
Given a vocabulary $\Sigma$, a binary structure over $\Sigma$ is one of the following:

1. An occurrence of a word $w \in \Sigma$.
2. A sequence $[S_1 \ S_2]$ where $S_1$ and $S_2$ are binary structures over $\Sigma$.

Definition (Denotation of a Structure)
Let $\Sigma$ be a vocabulary, $E$ be a non-empty set of entities, $T_\mathcal{L}$ be a lexical typing function over $\Sigma$ and $I_\mathcal{L}$ be a corresponding lexical denotation function over $\Sigma$. Then for every binary structure $S$ over $\Sigma$, the syntactic typing and denotation functions $T_s$ and $I_s$ extend $T_\mathcal{L}$ and $I_\mathcal{L}$ as follows:

\[
T_s(S) = \begin{cases} 
T_\mathcal{L}(w) & \text{if } S \text{ is a word } w \in \Sigma \\
\beta & \text{if } S = [S_1 \ S_2] \text{ and } T_s(S_1) = \alpha \rightarrow \beta \text{ and } T_s(S_2) = \alpha \\
\beta & \text{if } S = [S_1 \ S_2] \text{ and } T_s(S_1) = \alpha \text{ and } T_s(S_2) = \alpha \rightarrow \beta \\
\text{undefined} & \text{otherwise}
\end{cases}
\]

\[
I_s(S) = \begin{cases} 
I_\mathcal{L}(w) & \text{if } S \text{ is a word } w \in \Sigma \\
(t \ u) & \text{if } S = [S_1 \ S_2] \text{ and } I_s(S_1) = t : \alpha \rightarrow \beta \text{ and } I_s(S_2) = u : \alpha \\
(t \ u) & \text{if } S = [S_1 \ S_2] \text{ and } I_s(S_1) = u : \alpha \text{ and } I_s(S_2) = t : \alpha \rightarrow \beta \\
\text{undefined} & \text{otherwise}
\end{cases}
\]
Examples

Example (Tina praised Mary)

\[
\begin{align*}
\text{Tina} & \quad \text{praised} \quad \text{Mary} \\
\text{praised(mary)(tina)} & : t \\
\text{tina} & : e \\
\text{praised(mary)} & : e \rightarrow t \\
\text{praised} & : e \rightarrow e \rightarrow t \\
\text{mary} & : e
\end{align*}
\]

Example (Tina is tall)

\[
\begin{align*}
\text{Tina} & \quad \text{is} \quad \text{tall} \\
\text{is(tall)(tina)} & : t \\
\text{tina} & : e \\
\text{is(tall)} & : e \rightarrow t \\
\text{is} & : (e \rightarrow t) \rightarrow (e \rightarrow t) \\
\text{tall} & : e \rightarrow t
\end{align*}
\]
Quantification

Example (Everyone smiles)

```
everyone  smiles
```

\[ \text{everyone}(\text{smiles}) : t \]
\[ \text{everyone} : (e \to t) \to t \quad \text{smiles} : e \to t \]

Example (Every man smiles)

```
every  man  smiles
```

\[ \text{every(man)}(\text{smile}) : t \]
\[ \text{every(man)} : (e \to t) \to t \quad \text{smiles} : e \to t \]

\[ \text{every} : (e \to t) \to (e \to t) \to t \]
\[ \text{man} : e \to t \]

- Shake and bake semantics
- What’s the denotation of everyone?
- What’s the denotation of every?
Computing Denotations with Functions or Logical Formulas

So far

tall as:
- Set
- (Characteristic) Function
- Symbol of relation

⇒ Move to logic

What kind of logic?
Higher-Order Logic

HOL

- Two atomic types: $e$ and $t$ (or $ι$ and $o$)
- Logical constants:
  - $\bot : t$
  - $\Rightarrow : t \rightarrow t \rightarrow t$
  - $\forall \alpha : (\alpha \rightarrow t) \rightarrow t$ for each type $\alpha$

- Formulas: well-typed terms of type $t$

- Contrast with first order logic
- Build a HOL formula which is not FOL
Montague Semantics

1. Introduction
2. Meaning
3. Types and Model Structure
4. Montague Semantics
5. Phenomena at the Syntax-Semantics Interface
6. Abstract Categorial Grammars
7. Underspecification
8. Discourse
9. Selected Bibliography
There is in my opinion no important theoretical difference between natural languages and the artificial languages of logicians; indeed, I consider it possible to comprehend the syntax and semantics of both kinds of languages within a single natural and mathematically precise theory. On this point I differ from a number of philosophers (...).

[Montague, 1970]
The aim of this paper is to present in a rigorous way the syntax and semantics of a certain fragment of a certain dialect of English. [Montague, 1974]

- Fragment
- Semantic types as homomorphic image of syntactic types
- Semantic representation as translation of syntactic operations
- Semantic representation through a “certain artificial language”, a logical language
Categories of English

- Basic types: \(e\) and \(t\)
- Type constructors: \(A/B\) and \(A//B\)
- Some definitions:
  - \(\text{IV}\), or the category of intransitive verb phrases, is to be \(t/e\).
  - \(\text{T}\), or the category of terms, is to be \(t/\text{IV}\),
  - \(\text{TV}\), or the category of transitive verb phrases, is to be \(\text{IV}/\text{T}\).
  - \(\text{CN}\), or the category of common noun phrases, is to be \(t//e\).
  - \(...\)
- \(B_A\) the set of basic expressions of the category \(A\). \(P_A\) is the set of phrases of the category \(A\).
  - \(\text{love} \in B_{\text{TV}}\)
  - \(\text{Mary} \in B_{\text{T}}, \text{he}_0 \in B_{\text{T}}\)
  - \(\text{man} \in B_{\text{CN}}\)
Basic Rules

**S1** $B_A \subseteq P_A$ for every category $A$.

**S2** If $\zeta \in P_{CN}$, then $F_0(\zeta), F_1(\zeta), F_2(\zeta) \in P_T$ where:

- $F_0(\zeta) = \text{every } \zeta$
- $F_1(\zeta) = \text{the } \zeta$
- $F_2(\zeta)$ is a $\zeta$ or an $\zeta$ according as the first word in $\zeta$ takes a or an

...
Syntactic Rules

Rules of functional application

S4 If $\alpha \in P_{t/IV}$ and $\delta \in P_{IV}$, then $F_4(\alpha, \delta) \in P_t$, where $F_4(\alpha, \delta) = \alpha \delta'$ and $\delta'$ is the result of replacing the first verb in $\delta$ by its third person singular present.

S5 If $\delta \in P_{IV/T}$ and $\beta \in P_T$, then $F_5(\delta, \beta) \in P_{IV}$, where $F_5(\delta, \beta) = \delta \beta$ if $\beta$ does not have the form $he_n$ and $F_5(\delta, he_n) = \delta \ him_n$.

...
Rules of quantification

**S14** If $\alpha \in P_T$ and $\phi \in P_t$, then $F_{10,n}(\alpha, \phi) \in P_t$, where either:

1. $\alpha$ does not have the form $he_k$, and $F_{10,n}(\alpha, \phi)$ comes from $\phi$ by replacing the first occurrence of $he_n$ or $him_n$ by $\alpha$ and all other occurrences of $he_n$ or $him_n$ by $\begin{cases} he \\ she \end{cases}$ or $\begin{cases} him \\ her \\ it \end{cases}$ respectively, according as the gender of the first $B_{CN}$ or $B_T$ in $\alpha$ is $\begin{cases} masc. \\ fem. \\ neuter \end{cases}$, or

2. $\alpha = he_k$, and $F_{10,n}(\alpha, \phi)$ comes from $\phi$ by replacing all occurrences of $he_n$ or $him_n$ by $he_k$ or $him_k$ respectively

...
Every man loves Mary

every man loves Mary \((t,S_{14}, F_{10,0}(\text{every man}, \text{he}_0 \text{ loves Mary}))\)

every man \((T, S_{2}, F_{0}(\text{man}))\) \quad \text{he}_0 \text{ loves Mary } \((t, S_{4}, F_{4}(\text{he}_0, \text{love Mary}))\)

\text{man} \((\text{CN})\) \quad \text{he}_0 \text{ (}T = t/\text{IV}) \quad \text{love Mary} \((\text{IV}, S_{5}, F_{5}(\text{love, Mary}))\)

\text{love} \((\text{TV} = \text{IV}/T)\) \quad \text{Mary} \((T)\)
Translating English into (Intensional) Logic

Categories to Semantic Types

$f$ is a function such that

- $f(e) = e$
- $f(t) = t$
- $f(A/B) = f(A//B) = f(B) \rightarrow f(A)$ where $A, B$ are categories

Translation rules: the $\overline{\cdot}$ function

T1 If $\alpha$ is in the domain of $g$, then $\overline{\alpha} = g(\alpha)$ [interpretation of constants].

$he_n = \lambda P.P{x_n} . . .$

T2 If $\zeta \in P_{CN}$ then every $\overline{\zeta} = \lambda P.\forall x.\overline{\zeta}(x) \Rightarrow P(x), . . .$

\ldots

T4 If $\delta \in P_{t/IV}$, $\beta \in P_{IV}$ then $\overline{F_4(\delta, \beta)} = \overline{\delta(\beta)}$

T5 If $\delta \in P_{IV/T}$, $\beta \in P_T$ then $\overline{F_5(\delta, \beta)} = \overline{\delta(\beta)}$

\ldots

T14 If $\alpha \in P_T$, $\phi \in P_t$ then $\overline{F_{10,n}(\alpha, \phi)} = \overline{\alpha(\lambda x_n.\phi)}$

\ldots
Every man loves Mary

every man loves Mary \((t, S_{14}, F_{10,0}(\text{every man, he}_0 \text{ loves Mary}))\)

\[\text{every man} (\tau, S_2, F_0(\text{man}))\]

\[\text{man} (\text{CN})\]
\[\text{MAN}\]

\[\text{he}_0 \text{ loves Mary} (t, S_4, F_4(\text{he}_0, \text{love Mary}))\]

\[\text{he}_0 (\tau = t/\text{IV})\]
\[\lambda P. P \ x_0\]

\[\text{love Mary} (\text{IV}, S_5, F_5(\text{love, Mary}))\]

\[\text{love} (\tau_V = \text{IV}/\tau)\]
\[\lambda ox. o (\lambda y. \text{LOVE}(x, y))\]

\[\text{Mary} (\tau)\]
\[\lambda P. P \text{ MARY}\]
Remarks

- Type homomorphism
- Translation

⇒ What about widespread syntactic formalisms?
Conjunction I

Example (Mary and every boy smiles)
Example *(Mary and every boy smiles)*

\[
\text{and (every boy) (Mary)} : (e \to t) \to t \\
\lambda r. (\forall x. (\text{BOY } x \Rightarrow r x)) \land ((\lambda P.P_M) r) \\
\rightarrow_{\beta} \lambda r. (\forall x. (\text{BOY } x \Rightarrow r x)) \land (r M)
\]

\[
\text{and (every boy) (mary)} : (e \to t) \to t \\
\lambda r. (\forall x. (\text{BOY } x \Rightarrow r x)) \land ((\lambda P.P_M) r) \\
\rightarrow_{\beta} \lambda r. (\forall x. (\text{BOY } x \Rightarrow r x)) \land (r M)
\]

\[
\text{smiles : e \to t} \\
\text{SMILE}
\]

\[
\text{mary : (e \to t) \to t} \\
\lambda P.P_M
\]

\[
\text{and :} \\
((e \to t) \to t) \to ((e \to t) \to t) \to (e \to t) \to t \\
\lambda PQ. \lambda r. (P r) \land (Q r)
\]

\[
\text{and (every boy) : } ((e \to t) \to t) \to (e \to t) \to t \\
\lambda Q. \lambda r. ((\lambda Q. \forall x. (\text{BOY } x \Rightarrow Q x)) r) \land (Q r) \\
\rightarrow_{\beta} \lambda Q. \lambda r. ((\lambda Q. \forall x. (\text{BOY } x \Rightarrow r x)) r) \land (Q r)
\]

\[
\text{every boy : (e \to t) \to t} \\
\lambda Q. \forall x. (\text{BOY } x \Rightarrow Q x)
\]

\[
\text{every : (e \to t) \to (e \to t) \to t} \\
\lambda PQ. \forall x. (P x \Rightarrow Q x)
\]

\[
\text{boy : e \to t} \\
\text{BOY}
\]
How can we have both the *NP* subject and the *NP* object be arguments of a transitive verb?

- Allow for *abstraction* (see later)
- Change the denotation of transitive verbs:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>([e] = e)</td>
<td>([e] = e)</td>
</tr>
<tr>
<td>([IV] = t/e)</td>
<td>([IV] = t/e)</td>
</tr>
<tr>
<td>([IV'] = e \rightarrow t)</td>
<td>([IV'] = e \rightarrow t)</td>
</tr>
<tr>
<td>([T] = (e \rightarrow t) \rightarrow t)</td>
<td>([T] = (e \rightarrow t) \rightarrow t)</td>
</tr>
<tr>
<td>([TV] = ((e \rightarrow t) \rightarrow t) \rightarrow e \rightarrow t)</td>
<td>([TV] = ((e \rightarrow t) \rightarrow t) \rightarrow e \rightarrow t)</td>
</tr>
<tr>
<td>([IV'] = t/T)</td>
<td>([IV'] = t/T)</td>
</tr>
<tr>
<td>([TV'] = IV/T = (t/T)/T)</td>
<td>([TV'] = IV/T = (t/T)/T)</td>
</tr>
<tr>
<td>([TV'] = ((e \rightarrow t) \rightarrow t) \rightarrow ((e \rightarrow t) \rightarrow t) \rightarrow t)</td>
<td>([TV'] = ((e \rightarrow t) \rightarrow t) \rightarrow ((e \rightarrow t) \rightarrow t) \rightarrow t)</td>
</tr>
<tr>
<td>([smiles] = \lambda s.s(\lambda x.SMILE x))</td>
<td>([smiles] = \lambda s.s(\lambda x.SMILE x))</td>
</tr>
<tr>
<td>([loves] = \lambda o.s(\lambda o.\lambda x.LOVE x y))</td>
<td>([loves] = \lambda o.s(\lambda o.\lambda x.LOVE x y))</td>
</tr>
</tbody>
</table>
### CFG Based Approach

<table>
<thead>
<tr>
<th>Production</th>
<th>Derivation</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow NP\ VP$</td>
<td>$[S] = [VP][NP]$</td>
<td></td>
</tr>
<tr>
<td>$VP \rightarrow tV\ NP$</td>
<td>$[VP] = [tV][NP]$</td>
<td></td>
</tr>
<tr>
<td>$NP \rightarrow Det\ N$</td>
<td>$[NP] = [Det][N]$</td>
<td></td>
</tr>
<tr>
<td>$Det \rightarrow a$</td>
<td>$[Det] = \lambda PQ.\exists x.Px \land Qx$</td>
<td></td>
</tr>
<tr>
<td>$Det \rightarrow every$</td>
<td>$[Det] = \lambda PQ.\forall x.Px \Rightarrow Qx$</td>
<td></td>
</tr>
<tr>
<td>$tV \rightarrow loves$</td>
<td>$[tV] = \lambda os.s(\lambda x.o(\lambda y.LOVE \times y))$</td>
<td></td>
</tr>
<tr>
<td>$N \rightarrow man$</td>
<td>$[N] = MAN$</td>
<td></td>
</tr>
<tr>
<td>$N \rightarrow woman$</td>
<td>$[N] = WOMAN$</td>
<td></td>
</tr>
<tr>
<td>$NP \rightarrow Mary$</td>
<td>$[NP] = \lambda P.P\ M$</td>
<td></td>
</tr>
<tr>
<td>$NP \rightarrow John$</td>
<td>$[NP] = \lambda P.P\ J$</td>
<td></td>
</tr>
<tr>
<td>$NP \rightarrow NP_1\ Conj\ NP_2$</td>
<td>$[NP] = \lambda r.[Conj](\lambda NP_1 r)(\lambda NP_2 r)$</td>
<td></td>
</tr>
<tr>
<td>$Conj \rightarrow and$</td>
<td>$[Conj] = \lambda s_1 s_2.s_1 \land s_2$</td>
<td></td>
</tr>
</tbody>
</table>

- $[John\ loves\ a\ woman] = ?$
- $[Every\ man\ loves\ some\ woman] = ?$
- How do you get an object wide scope reading?
Adjectives

Grammar

\[
\begin{align*}
S & \rightarrow NP \ VP & [S] &= [VP][NP] \\
NP & \rightarrow Det \ N & [NP] &= [Det][N] \\
Det & \rightarrow a & [Det] &= \lambda PQ.\exists x. P x \land Q x \\
Det & \rightarrow every & [Det] &= \lambda PQ.\forall x. P x \Rightarrow Q x \\
VP & \rightarrow smiles & [VP] &= \lambda s.s(\lambda x. SMILE x) \\
N & \rightarrow man & [N] &= MAN \\
NP & \rightarrow Mary & [NP] &= \lambda P.P \ M \\
NP & \rightarrow John & [NP] &= \lambda P.P \ J \\
N & \rightarrow Adj \ N & [N] &= \lambda x.(\lambda y. [[Adj][N]] y) x \\
Adj & \rightarrow big & [Adj] &= \lambda N.\lambda x.BIG x \land N x
\end{align*}
\]

- \([\textit{big man}] = \lambda x. (\textit{BIG} x) \land (\textit{MAN} x)\) intersective adjectives
- \([\textit{A big man smiles}] = ?\)
- \([\textit{beautiful dancer}] = ?\) subsective adjectives
- \([\textit{former student}] = ?\) non-intersective and non-subsective adjectives
Abstract Categorial Grammars

1. Introduction
2. Meaning
3. Types and Model Structure
4. Montague Semantics
5. Phenomena at the Syntax-Semantics Interface
6. Abstract Categorial Grammars
   - Architecture of Grammatical Formalisms
   - \(\lambda\)-terms in the Syntax... and Everywhere
   - Principles and Definition
   - ACG Composition: The Picture
   - About Word Order
   - Providing a Syntax-Semantics Interface to Context-Free Grammars
   - Modularity of the Components
   - A Functional View on TAG
   - TAG as ACG
   - The CG Approach to Scope Ambiguity
   - Removing Ambiguity From Syntax
Some Observations on Various Grammatical Formalisms

Syntactic Objects (trees, proofs, f-structures) are somehow prior and semantics must be parasitic on those syntactic objects

[Muskens, 2001]

Changing the syntactic analysis to simplify one mapping makes the other mapping more complex. A third possibility is to keep both correspondences simple by localizing the complexity in the syntactic component itself. (...) [T]here is a mismatch between phonology and meaning, which has to be encoded somewhere in the mapping among the levels of structure. If this mismatch is eliminated at one point in the system, it pops up elsewhere.

[Jackendoff, 2002, p.15]
Mainstream Architectures
On the Place of the Syntactic Component

Three Components

- Generative theory: “Free combinatoriality of language is due to a single source, localized in syntactic structure”
- **Syntactocentric** formalisms = function from the syntactic component to the other ones
A Tripartite Parallel Architecture

Three Components

Phonology \(\rightarrow\) Syntax, Syntactic structures \(\leftarrow\) Semantics

Interface

Weakly Syntactocentric formalisms = relation between the syntactic component and the other ones

Language comprises a number of independent combinatorial systems which are aligned with each other by means of a collection of interface systems. Syntax is among the combinatorial systems, but far from the only one. [Jackendoff, 2002]
What are $\lambda$-terms useful for?

- Montague-like semantics
- Generalization of trees and strings
- Any kind of signatures (atomic types and typed constants): FOL propositions, descriptions (LFG f-structures, URL), other logics
- Very well studied generative system
- Variable binding system

Not that New in Syntax

- Movements in GB/MG to get S-structures.
- Index Transfer syntactic rule in Binding Theory
- TAG $\rightarrow$ MCTAG
The Tectogrammatical and Phenogrammatical Distinction

On the Grammar Architecture [Curry, 1961]

- Tectogrammatical: abstract combinatorial structure of the grammar
- Phenogrammatical: concrete operations on syntactic data structures (strings, trees, descriptions)
- Contrary to the view that:
  - Syntactic objects are the main objects
  - Semantics (and phonology, and ...) are by-products

Related Works

ACG: a Grammatical Framework

Main Features

- ACG is a (grammatical) framework
- An ACG $\mathcal{G}$ generates two languages:
  - The abstract language $\mathcal{A}(\mathcal{G})$
  - The object language $\mathcal{O}(\mathcal{G})$

Abstract language: Admissible structures (as in syntactic structures)
Object language: Realizations of the admissible structures

- Both languages are the same objects: sets of (linear) $\lambda$-terms
### ACG: Formal Properties


<table>
<thead>
<tr>
<th>String language</th>
<th>Tree language</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ACG(_{1,n})</strong> finite</td>
<td>finite</td>
</tr>
<tr>
<td><strong>ACG(_{2,1})</strong> regular</td>
<td>regular</td>
</tr>
<tr>
<td><strong>ACG(_{2,2})</strong> context-free</td>
<td>linear context-free</td>
</tr>
<tr>
<td><strong>ACG(_{2,3})</strong> non-duplicating macro</td>
<td>⊂ 1-visit attribute grammar</td>
</tr>
<tr>
<td>well-nested multiple context-free</td>
<td></td>
</tr>
<tr>
<td><strong>ACG(_{2,4})</strong> mildly context-sensitive</td>
<td>hyperedge replacement grammar</td>
</tr>
<tr>
<td>(multiple context-free)</td>
<td></td>
</tr>
<tr>
<td><strong>ACG(_{2,4+n})</strong></td>
<td>ACG(_{2,4})</td>
</tr>
<tr>
<td><strong>ACG(_{3,n})</strong> MELL decidability</td>
<td>MELL decidability</td>
</tr>
</tbody>
</table>

#### Complexity

- **ACG\(_{2,n}\)** parsing is polynomial, equivalent to datalog querying [Salvati, 2007, Kanazawa, 2007]
- Reduces to best cases with standard techniques (magic set rewriting) with correct prefix Earley algorithms [Kanazawa, 2008]
ACG Definition $\mathcal{G} = \langle \Sigma_a, \Sigma_o, \mathcal{L}, S \rangle$

$\Sigma_a$

$NP, S : type$
$Chris : NP$
$met : NP \rightarrow NP \rightarrow S$

\[
\begin{align*}
\mathcal{L}(NP) &= \sigma \\
\mathcal{L}(S) &= \sigma \\
\mathcal{L}(Chris) &= Chris \\
\mathcal{L}(met) &= \lambda os.s + met + o
\end{align*}
\]

$\Sigma_o$

$\sigma : type$
$Chris : \sigma$
$met : \sigma$
$+ : \sigma \rightarrow \sigma \rightarrow \sigma$

$\Lambda(\Sigma_a)$

$\mathcal{A}(\mathcal{G}) = \{ t \mid \vdash t : S \}$

$\Lambda(\Sigma_o)$

$\mathcal{L}(\alpha \rightarrow \beta) = \mathcal{L}(\alpha) \rightarrow \mathcal{L}(\beta)$
$\mathcal{L}(t u) = \mathcal{L}(t) \mathcal{L}(u)$
$\mathcal{L}(\lambda^o x.t) = \lambda^o x.\mathcal{L}(t)$
Example
My First “Chris met Sandy” ACG Program

\[ \sum_a : \]
\[ NP, S : \text{type} \]
\[ \text{Chris, Sandy} : NP \]
\[ \text{met} : NP \rightarrow NP \rightarrow S \]

\[ \Sigma_o : \]
\[ \sigma : \text{type} \]
\[ + : \sigma \rightarrow \sigma \rightarrow \sigma \]
\[ \text{Chris, Sandy} : \sigma \]
\[ \text{met} : \sigma \]

\[ \mathcal{L}(\text{met Sandy Chris}) \]
\[ = \mathcal{L}(\text{met}) \mathcal{L}(\text{Sandy}) \mathcal{L}(\text{Chris}) \]
\[ = (\lambda^o \text{os.s + met + o})(\text{Sandy})(\text{Chris}) \]
\[ = (\lambda^o \text{s.s + met + Sandy})(\text{Chris}) \]
\[ = \text{Chris + met + Sandy} \]
ACG Architecture
Composition Ability

- Functional composition
- Abstract language sharing (bimorphism)
- Parsing and generation (syntactic realization) in the usual sense: function inversion
Intermediate Conclusion

So far...

- Discussion on possible architectures of grammatical formalisms
- Discussion on function-compositional properties of ACG and *modularity*:
  - Abstract structures are mapped to object structures
  - Object structures: Strings, Simple semantic objects, Complex semantic objects:
    - Continuized semantics: \( S := (t \circ t) \circ t \)
    - Results of ACG composition
    - Dynamic semantics: \( S := \gamma \circ (\gamma \circ t) \circ t \) [de Groote, 2006]
  
  **ARC CAuLD**: Construction Automatique de représentations Logiques du Discours, 
  http://www.loria.fr/~pogodalla/cauld/

- Underspecified representations

- Algorithms for parsing and generation (in the usual sense) are essentially the same: ACG parsing: finding the abstract antecedent of an object
- Abstract structures?
**CFG into ACG Encoding**

**Example (CFG)**

\[\begin{align*}
\rho_0 & : S \rightarrow NP \ VP \\
\rho_1 & : VP \rightarrow tV \ NP \\
\rho_2 & : NP \rightarrow John \\
\rho_3 & : NP \rightarrow Mary \\
\rho_4 & : VP \rightarrow left \\
\rho_5 & : tV \rightarrow saw
\end{align*}\]

**CFG as ACG**

\[
\begin{array}{l}
\Sigma_{Rules} & \quad \Sigma_{Strings} \\
\rho_0 & : NP \rightarrow VP \rightarrow S & : \sigma \rightarrow \sigma \rightarrow \sigma \\
\rho_1 & : tV \rightarrow NP \rightarrow VP & : \sigma \rightarrow \sigma \rightarrow \sigma \\
\rho_2 & : NP & : \sigma \\
\rho_3 & : NP & : \sigma \\
\rho_4 & : VP & : \sigma \\
\rho_5 & : tV & : \sigma
\end{array}
\]

\[
\begin{array}{l}
\mathcal{L}_{CFG} \\
\rho_0 & : S \\
\rho_1 & : VP \\
\rho_2 & : NP \\
\rho_3 & : NP \\
\rho_4 & : VP \\
\rho_5 & : tV
\end{array}
\]

\[\begin{align*}
\rho_0 & : S \\
\rho_1 & : VP \\
\rho_2 & : NP \\
\rho_3 & : NP \\
\rho_4 & : VP \\
\rho_5 & : tV
\end{align*}\]

\[\begin{array}{l}
John \\
\rho_3 \quad NP \\
\rho_5 \quad tV \\
saw \\
Mary
\end{array}\]
CFG into ACG Encoding (cont’d)

<table>
<thead>
<tr>
<th>Σ_{Rules}</th>
<th>Λ_{CFG}</th>
<th>Σ_{Strings}</th>
</tr>
</thead>
<tbody>
<tr>
<td>ρ₀ : NP → VP → S</td>
<td>:= λxy. x + y : σ → σ → σ</td>
<td></td>
</tr>
<tr>
<td>ρ₁ : tV → NP → VP</td>
<td>:= λxy. x + y : σ → σ → σ</td>
<td></td>
</tr>
<tr>
<td>ρ₂ : NP</td>
<td>:= John : σ</td>
<td></td>
</tr>
<tr>
<td>ρ₃ : NP</td>
<td>:= Mary : σ</td>
<td></td>
</tr>
<tr>
<td>ρ₄ : VP</td>
<td>:= left : σ</td>
<td></td>
</tr>
<tr>
<td>ρ₅ : tV</td>
<td>:= saw : σ</td>
<td></td>
</tr>
</tbody>
</table>

\[ \Lambda_{CFG}(ρ₀ \ ρ₂ (ρ₁ ρ₅ ρ₃) : S) = (λx. John + y)((λy. saw + y)Mary) \]
\[ \rightarrow_β (λy. John + y)(saw + Mary) \]
\[ \rightarrow_β John + (saw + Mary) \]
CFG Encoding

CFG derivation trees

CFG string languages

CFG direct semantics

$G_{CFG}$

$G_{sem}$
A Direct Semantics
Sharing Abstract Languages

CFG syntax as ACG

\[ \Sigma_{Rules} \]
\[
\rho_0 : NP \rightarrow VP \rightarrow S := \lambda xy. x + y : \sigma \rightarrow \sigma \rightarrow \sigma \\
\rho_1 : tV \rightarrow NP \rightarrow VP := \lambda xy. x + y : \sigma \rightarrow \sigma \rightarrow \sigma \\
\rho_2 : NP := John : \sigma \\
\rho_3 : NP := Mary : \sigma \\
\rho_4 : VP := left : \sigma \\
\rho_5 : tV := saw : \sigma
\]

CFG (direct) semantics as ACG

\[ \Sigma_{Rules} \]
\[
\rho_0 : NP \rightarrow VP \rightarrow S := \lambda sP. P s : e \rightarrow (e \rightarrow t) \rightarrow t \\
\rho_1 : tV \rightarrow NP \rightarrow VP := \lambda Pos. P s o : (e \rightarrow e \rightarrow t) \rightarrow e \rightarrow e \rightarrow t \\
\rho_2 : NP := John : e \\
\rho_3 : NP := Mary : e \\
\rho_4 : VP := left : e \rightarrow t \\
\rho_5 : tV := saw : e \rightarrow e \rightarrow t
\]
### A Direct Semantics (cont’d)

#### CFG (direct) semantics as ACG

<table>
<thead>
<tr>
<th>Rules</th>
<th>$\mathcal{L}_{\text{sem}}$</th>
<th>$\Sigma_{\text{Log}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho_0 : \text{NP} \rightarrow \text{VP} \rightarrow S$</td>
<td>$\lambda sP.P\ s$</td>
<td>$e \rightarrow (e \rightarrow t) \rightarrow t$</td>
</tr>
<tr>
<td>$\rho_1 : \text{tV} \rightarrow \text{NP} \rightarrow \text{VP}$</td>
<td>$\lambda\text{Pos}.P\ s\ o$</td>
<td>$(e \rightarrow e \rightarrow t) \rightarrow e \rightarrow e \rightarrow t$</td>
</tr>
<tr>
<td>$\rho_2 : \text{NP}$</td>
<td>$\text{John}$</td>
<td>$e$</td>
</tr>
<tr>
<td>$\rho_3 : \text{NP}$</td>
<td>$\text{Mary}$</td>
<td>$e$</td>
</tr>
<tr>
<td>$\rho_4 : \text{VP}$</td>
<td>$\text{left}$</td>
<td>$e \rightarrow t$</td>
</tr>
<tr>
<td>$\rho_5 : \text{tV}$</td>
<td>$\text{saw}$</td>
<td>$e \rightarrow e \rightarrow t$</td>
</tr>
</tbody>
</table>

\[
\mathcal{L}_{\text{sem}}(\rho_0\ \rho_2\ (\rho_1\ \rho_5\ \rho_3) : S) = (\lambda sP.P\ s)\text{John}((\lambda\text{Pos}.P\ s\ o)\text{saw}\ \text{Mary})
\]

$\rightarrow_\beta (\lambda P.P\ \text{John})((\lambda o.saw\ s\ o)\text{Mary})$

$\rightarrow_\beta (\lambda P.P\ \text{John})(\lambda o.saw\ s\ \text{Mary})$

$\rightarrow_\beta (\lambda o.saw\ s\ \text{Mary})\text{John}$

$\rightarrow_\beta \text{saw}\ \text{John}\ \text{Mary}$
A Continuized (Higher–Order) Semantics

CFG continuized semantics as ACG [Barker, 2002]

<table>
<thead>
<tr>
<th>Rule</th>
<th>( \Sigma_{Rules} )</th>
<th>( \mathcal{L}_{sem} )</th>
<th>( \Sigma_{Log} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S )</td>
<td>( (t \rightarrow t) \rightarrow t )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( NP )</td>
<td>( (e \rightarrow t) \rightarrow t )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( VP )</td>
<td>( ((e \rightarrow t) \rightarrow t) \rightarrow t )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( tV )</td>
<td>( ((e \rightarrow e \rightarrow t) \rightarrow t) \rightarrow t )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( N )</td>
<td>( ((e \rightarrow t) \rightarrow t) \rightarrow t )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( Det )</td>
<td>( (((e \rightarrow t) \rightarrow t) \rightarrow t) \rightarrow ((e \rightarrow t) \rightarrow t) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \rho_0 )</td>
<td>( NP \rightarrow VP \rightarrow S )</td>
<td>( \lambda^o svp.v(\lambda^o P.s(\lambda^o x.p(Px))) )</td>
<td></td>
</tr>
<tr>
<td>( \rho_1 )</td>
<td>( tV \rightarrow NP \rightarrow VP )</td>
<td>( \lambda^o voP.v(\lambda^o R.o(\lambda^o y.P(Ry))) )</td>
<td></td>
</tr>
<tr>
<td>( \rho_2 )</td>
<td>( NP )</td>
<td>( \lambda^o P.P \text{ John} )</td>
<td></td>
</tr>
<tr>
<td>( \rho_5 )</td>
<td>( tV )</td>
<td>( \lambda^o P.P \text{ saw} )</td>
<td></td>
</tr>
<tr>
<td>( \rho_{every} )</td>
<td>( Det )</td>
<td>( \lambda^o KP.K(\lambda^o Q.\forall x.(Q x) \Rightarrow (P x)) )</td>
<td></td>
</tr>
</tbody>
</table>

- Scope ambiguity
- Scope displacement
- NP as a scope island
Abstract Categorial Grammars

Modularity of the Components

CFG Encoding

CFG derivation trees

CFG string languages

CFG continuized semantics

CFG direct semantics

$G_{CFG}$

$G_{cont.\ sem}$

$G_{sem}$
Trees as $\lambda$-Terms

Trees Build on a Ranked Alphabet

- $S$ of arity 2 (non-terminal)
- $NP$ of arity 1 (non-terminal)
- $VP$? $VP_1$ of arity 1 and $VP_2$ of arity 2 (non-terminals)
- $John$ of arity 0 (terminal)

- $S_2 : \tau \rightarrow \tau \rightarrow \tau$
- $NP_1 : \tau \rightarrow \tau$
- $VP_1 : \tau \rightarrow \tau$, $VP_2 : \tau \rightarrow \tau \rightarrow \tau$
- $John : \tau$
A Functional View on TAG

Tree Adjunction:
Auxiliary Trees as Functions

Example

\[
\lambda^o x. \left( \lambda a. \left( S \xrightarrow{\beta} \right) \right)
\]

\[
\left( \begin{array}{c}
\text{apparently} \\
\end{array} \right)
\]

\[
\left( \begin{array}{c}
\text{likes} \\
\end{array} \right)
\]

\[
\left( \begin{array}{c}
\text{apparently} \\
\end{array} \right)
\]

\[
\left( \begin{array}{c}
\text{likes} \\
\end{array} \right)
\]

\[
\left( \begin{array}{c}
\text{apparently} \\
\end{array} \right)
\]

\[
\left( \begin{array}{c}
\text{likes} \\
\end{array} \right)
\]
Adjunction as Functional Application

\[ \gamma' \text{likes} \gamma' \text{apparently} = \]

\[ \left( \lambda^o a. \right) \text{NP} \]

\[ \rightarrow_\beta \]

\[ \text{NP} \]

\[ \left( \lambda^o x. \right) \text{apparently} \text{VP} \]

\[ \rightarrow_\beta \]

\[ \text{apparently} \text{VP} \]

\[ \rightarrow_\beta \]

\[ \text{likes} \text{NP} \]
Substitution Operation
Substitution as Functional Application

Example

$$\lambda^o os. \left( \begin{array}{c} S \\ \text{likes} \end{array} \right) \left( \begin{array}{c} NP \\ \text{Chris} \end{array} \right) \left( \begin{array}{c} NP \\ \text{Sandy} \end{array} \right) \rightarrow^\beta \left( \begin{array}{c} NP \\ \text{Chris} \end{array} \right) \left( \begin{array}{c} NP \\ \text{Sandy} \end{array} \right)$$

$$\left( \begin{array}{c} S \\ \text{likes} \\ o \end{array} \right)$$
Putting Everything Together

\[ \Sigma_{\text{trees}}: \]

\[
\begin{align*}
\tau & : \text{type} \\
\gamma_{\text{apparently}} & = \lambda^\circ ax.a \left( \begin{array}{c}
\text{apparently} \\
x
\end{array} \right) : (\tau \to \tau) \to \tau \\
l & = \lambda^\circ x.x : \tau \\
NP & \\
\gamma_{\text{John}} & = \begin{array}{c}
\text{John}
\end{array} : \tau
\end{align*}
\]

\[
\begin{align*}
\gamma_{\text{likes}} & = \lambda^\circ Saso.S \left( \begin{array}{c}
S \\
\text{likes}
\end{array} \right) : (\tau \to \tau) \\
s & \\
a & \\
VP & \\
o &
\end{align*}
\]

\[
\begin{align*}
\left( \tau \to \tau \right) & : \tau \\
\tau & : \tau \\
\tau & : \tau \\
\tau & : \tau
\end{align*}
\]
TAG Derivation as Term Application

Example

\[
\gamma_{\text{likes}} \mid \gamma_{\text{John}} \gamma_{\text{Mary}} = \\
(\lambda^o \text{Saso.} \text{S}) \left( \begin{array}{c}
\text{S} \\
\gamma_{\text{likes}} \mid \gamma_{\text{John}} \gamma_{\text{Mary}} \\
\end{array} \right)
\]

\[
\rightarrow_\beta (\lambda^o \text{so.} \text{S}) \left( \begin{array}{c}
\text{S} \\
\gamma_{\text{likes}} \mid \gamma_{\text{John}} \gamma_{\text{Mary}} \\
\end{array} \right)
\]

\[
\rightarrow_\beta \text{S} \left( \begin{array}{c}
\text{NP} \\
\text{VP} \\
\text{likes} \mid \gamma_{\text{John}} \gamma_{\text{Mary}} \\
\end{array} \right)
\]
Yield as an ACG

Derived Tree Signature

- $S_2 : \tau \rightarrow \tau \rightarrow \tau$
- $NP_1 : \tau \rightarrow \tau$
- $VP_1 : \tau \rightarrow \tau$, $VP_2 : \tau \rightarrow \tau \rightarrow \tau$
- $John : \tau$

String signature (as before):

- $\sigma : \text{type}$
- $John, \text{likes} \ldots : \sigma$

$\mathcal{G}_{Yield}$

- $\tau := \sigma$
- $John := John$
- $X_1 := \lambda x.x$
- $X_2 := \lambda xy.x + y$
- $\ldots$

$S_2(NP_1 John)(VP_2 \text{likes}(NP_1 Mary)) := John + (\text{likes} + Mary)$
TAG as ACG
The Current Picture

\[ \Lambda(\Sigma_{\text{trees}}) \rightarrow \Lambda(\Sigma_{\text{derivations}}) \rightarrow \text{TAG derived trees} \]
\[ \text{TAG string languages} \rightarrow \text{TAG derivation trees} \]

\[ \mathcal{G}_{\text{yield}} \rightarrow \mathcal{G}_{\text{typed trees}} \]
### Abstract Categorial Grammars

#### A Functional View on TAG

\[ A(\mathcal{G}_{\text{yield}}) = \text{TAG Derived Trees?} \]

#### Σ_{trees}:

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>τ</td>
<td>type</td>
</tr>
<tr>
<td>γ_{apparently} = \lambda^o ax.x (apparently x) : (τ \circ τ) \circ τ \circ τ</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>: τ \circ τ</td>
</tr>
<tr>
<td>NP</td>
<td>: τ</td>
</tr>
<tr>
<td>γ_{John} = [John]</td>
<td></td>
</tr>
</tbody>
</table>

### Equation:

\[ γ_{apparently} l γ_{John} = \text{apparently} \text{NP} \text{John} \]
TAG as ACG

Category Induced Constraints

- The site of an adjunction has the same category as the root (and foot) node of the auxiliary tree.
- The site of a substitution has the same category as the root node of the substituted tree.

\[ \Sigma_{\text{derivations}} \xrightarrow{\mathcal{L}_{\text{typed trees}}} \Sigma_{\text{trees}} \]

\[ c_{\text{John}} : NP \quad \vdash: \quad \gamma_{\text{John}} : \tau \]

\[ c_{\text{apparently}} : (VP \rightarrow VP) \rightarrow VP \rightarrow VP \quad \vdash: \quad \gamma_{\text{apparently}} : (\tau \rightarrow \tau) \rightarrow \tau \rightarrow \tau \]

\[ NP, VP, S \ldots \quad \vdash: \quad \sigma \]

Example

There is no \( t : VP \in \Lambda(\Sigma_{\text{derivations}}) \) such that \( t := \text{apparently} \)

\[
\begin{array}{c}
NP \\
\downarrow \text{John}
\end{array}
\xrightarrow{\text{apparently}}
\begin{array}{c}
VP
\end{array}
\]
Control on the Derived Trees

\( G_{\text{typed trees}} = \langle \Sigma_{\text{derivations}}, \Sigma_{\text{trees}}, \mathcal{L}_{\text{typed trees}}, S \rangle \)

\[ NP, VP, S \ldots \]
\[ c_{\text{John}} : NP \]
\[ c_{\text{apparently}} : (VP \rightarrow VP) \rightarrow VP \rightarrow VP \]
\[ c_{\text{likes}} : (S \rightarrow S) \rightarrow (VP \rightarrow VP) \rightarrow NP \rightarrow NP \rightarrow S \]

\[ NP, VP, S \ldots \]
\[ c_{\text{John}} : NP \]
\[ c_{\text{apparently}} : (VP \rightarrow VP) \rightarrow VP \rightarrow VP \]
\[ c_{\text{likes}} : (S \rightarrow S) \rightarrow (VP \rightarrow VP) \rightarrow NP \rightarrow NP \rightarrow S \]

\[ c_{\text{likes}}(c_{\text{apparently}} l_{VP})c_{\text{John}}c_{\text{Mary}} : S = \]
\[ NP \rightarrow S \rightarrow VP \rightarrow VP \rightarrow NP \rightarrow \]
TAG Derivation Trees as Abstract Terms

\[ c_{\text{likes}} \cdot I_S( c_{\text{apparently}} \cdot I_{\text{VP}} ) \cdot c_{\text{John}} \cdot c_{\text{Mary}} : S = I_S \]

\[ \begin{array}{c}
  0 \\
  1 \\
  2 \\
  3
\end{array} \]

\[ c_{\text{likes}} \]

\[ \begin{array}{c}
  (S \rightarrow S) \\
  (VP \rightarrow VP) \\
  NP \\
  NP \\
  S
\end{array} \]
Let’s Build Some Semantic Representation

Forgetting few seconds about TAG, we have:

- A higher-order signature $\Sigma_{\text{derivations}}$:
  - $VP$, $NP$, $S$ : types
  - $c_{\text{john}}$, $c_{\text{mary}}$ : $NP$
  - $c_{\text{apparently}}$ : $VP \rightarrow VP$
  - $c_{\text{likes}}$ : $(VP \rightarrow VP) \rightarrow NP \rightarrow S$

- Some knowledge about Montague-like semantics?

A standard interpretation

$$
S := t  \quad NP := (e \rightarrow t) \rightarrow t
$$

$$
VP := e \rightarrow t
$$

$$
c_{\text{john}} := \lambda^o P.P \mathbf{j}
$$

$$
c_{\text{apparently}} := \lambda^o aP.a(\lambda x.\text{apparently}(P x))
$$

$$
l_{VP} := \lambda x.x
$$

$$
c_{\text{likes}} := \lambda a o s.s(a(\lambda x.o(\lambda y.\text{like} \ x \ y)))
$$

How to get the object wide scope reading?
TAG with Semantics

\[ \Lambda(\Sigma_{\text{trees}}) \]

\[ \Lambda(\Sigma_{\text{derivations}}) \]

\[ \Lambda(\Sigma_{\text{Log}}) \]

TAG derivation trees

TAG derived trees

TAG string languages

\[ G_{\text{yield}} \]

\[ G_{\text{typed trees}} \]

\[ G_{\text{Log}} \]
Intermediate Conclusion

So far
- Trees as $\lambda$-terms
- Yield as an ACG
- Typing control: ACG from derivation trees to derived trees
- Some semantics added. Is it a function from syntax?

Questions?
- Any TAG feature missing?
- Order of $G_{\text{typed trees}}$? ($c_{\text{likes}} : (VP \rightarrow VP) \rightarrow NP \rightarrow S$)
The Actual Picture

\[ \Lambda(\Sigma_{TAG}) \]

\[ \mathcal{G}_{TAG} \]

\[ \mathcal{G}_{yield} \]

\[ \mathcal{G}_{Log} \]

\[ \mathcal{G}_{typed \ trees} \]
The Final Picture

\[ \Lambda(\Sigma_{\text{MCTAG}}) \xrightarrow{\mathcal{G}_{\text{MCTAG}}} \Lambda(\Sigma_{\text{tuple}}) \xrightarrow{\mathcal{G}_{\text{tuple}}} \Lambda(\Sigma_{\text{TAG}}) \xrightarrow{\mathcal{G}_{\text{TAG}}} \Lambda(\Sigma_{\text{yield}}) \xrightarrow{\mathcal{G}_{\text{typed trees}}} \Lambda(\Sigma_{\text{Log}}) \]
Scope Ambiguities

\[
\text{every man loves some woman} \rightarrow \begin{cases} \forall x. \text{man} \ x \rightarrow (\exists y. \text{woman} \ y \land \text{love} \ x \ y) \\ \exists y. \text{woman} \ y \land (\forall x. \text{man} \ x \rightarrow \text{love} \ x \ y) \end{cases}
\]

CFG-like systems

\[
\begin{align*}
[\text{loves}] &= \lambda o.s(\lambda x.o(\lambda y.\text{LOVE}(x, y))) \\
[\text{loves}] &= \lambda o.s(\lambda y.s(\lambda x.\text{LOVE}(x, y)))
\end{align*}
\]

Underspecified framework (see the section on underspecification)

\[
\text{every man loves some woman} \rightarrow \begin{array}{c}
\square \\
\triangle
\end{array} \rightarrow \begin{cases} \forall x. \text{man} \ x \rightarrow (\exists y. \text{woman} \ y \land \text{love} \ x \ y) \\ \exists y. \text{woman} \ y \land (\forall x. \text{man} \ x \rightarrow \text{love} \ x \ y) \end{cases}
\]

Type Logical framework

\[
\text{every man loves some woman} \rightarrow \begin{array}{c}
\triangle \\
\triangle
\end{array} \rightarrow \begin{cases} \forall x. \text{man} \ x \rightarrow (\exists y. \text{woman} \ y \land \text{love} \ x \ y) \\ \exists y. \text{woman} \ y \land (\forall x. \text{man} \ x \rightarrow \text{love} \ x \ y) \end{cases}
\]
# Strengths and Weaknesses

## Underspecified framework

<table>
<thead>
<tr>
<th>Pros:</th>
<th>Cons:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- One syntactic analysis</td>
<td>- Description language</td>
</tr>
<tr>
<td>- Expressivity</td>
<td>- Ambiguity in the semantic recipe, not in the interface</td>
</tr>
</tbody>
</table>

## TL framework

<table>
<thead>
<tr>
<th>Pros:</th>
<th>Cons:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- No intermediate language</td>
<td>- Syntactic ambiguity</td>
</tr>
<tr>
<td>- Ambiguity handled by the process</td>
<td></td>
</tr>
</tbody>
</table>

## Question

Is there an ACG way providing a proof-theoretic approach with only one syntactic structure and no intermediate language?
Scope Ambiguity in Categorial grammars

The standard way

\[
\begin{align*}
\text{loves} & \quad [NP] \\
\text{someone} & \quad (NP/S) \backslash S \\
\text{everyone} & \quad S/(NP/S) \\
S & \quad NP/S \\
\end{align*}
\]
\[
C_{\text{someone}} (\lambda^o y. C_{\text{everyone}} (\lambda^o x. C_{\text{loves}} y x))
\]

The ACG way

- Replace \ and / by \(\rightarrow\)
- \(C_{\text{everyone}} : (NP \rightarrow S) \rightarrow S\)
Scope Ambiguity in ACG: The Semantics

\[
\Sigma_{CG} \\
C_{loves} : NP \rightarrow NP \rightarrow S \\
C_{everyone} : (NP \rightarrow S) \rightarrow S \\
C_{someone} : (NP \rightarrow S) \rightarrow S \\
\]

\[
\Sigma_{log} \\
\text{LOVES} : e \rightarrow e \rightarrow t \\
\forall, \exists : (e \rightarrow t) \rightarrow \\
\land, \Rightarrow : t \rightarrow t \rightarrow t \\
\]

\[
L_{amb-log} \\
NP \quad := \; e \\
S \quad := \; t \\
C_{loves} \quad := \; \lambda^o os.\text{LOVES}(s.o) \\
C_{everyone} \quad := \; \lambda^o P.\forall x.\text{HUMAN}(x) \Rightarrow P x \\
C_{someone} \quad := \; \lambda^o P.\exists y.\text{HUMAN}(y) \land P y \\
\]

\[
C_{everyone}(\lambda^o x. C_{someone}(\lambda^o y. C_{loves} x y)) \\
:= \text{amb-log} (\lambda^o P.\forall x.\text{HUMAN}(x) \Rightarrow P x)((\lambda^o x.(\lambda^o P.\exists y.\text{HUMAN}(y) \land P y)(\lambda^o y.\text{LOVES}(x, y))))) \\
\rightarrow_\beta (\lambda^o P.\forall x.\text{HUMAN}(x) \Rightarrow P x)((\lambda^o x.(\exists y.\text{HUMAN}(y) \land \text{LOVES}(x, y)))) \\
\rightarrow_\beta (\forall x.\text{HUMAN}(x) \Rightarrow (\exists y.\text{HUMAN}(y) \land \text{LOVES}(x, y)))
\]
Scope Ambiguity in ACG: The Semantics (cont’d)

\[ \Sigma_{CG} \]
\[ C_{loves} : NP \rightarrow NP \rightarrow S \]
\[ C_{everyone} : (NP \rightarrow S) \rightarrow S \]
\[ C_{someone} : (NP \rightarrow S) \rightarrow S \]

\[ \Sigma_{log} \]
\[ LOVES : e \rightarrow e \rightarrow t \]
\[ \forall, \exists : (e \rightarrow t) \rightarrow \]
\[ \wedge, \Rightarrow : t \rightarrow t \rightarrow t \]

\[ \mathcal{L}_{amb-log} \]
\[ NP := e \]
\[ S := t \]
\[ C_{loves} := \lambda^o os. LOVEs(s.o) \]
\[ C_{everyone} := \lambda^o P. \forall x. HUMAN(x) \Rightarrow P x \]
\[ C_{someone} := \lambda^o P. \exists y. HUMAN(y) \land P y \]

\[ C_{someone}(\lambda^o y. C_{everyone}(\lambda^o x. C_{loves} x y)) \]
\[ := \text{amb-log } (\lambda^o P. \exists y. HUMAN(y) \land P y)((\lambda^o x.(\lambda^o P. \forall x. HUMAN(x) \Rightarrow P y)(\lambda^o y. LOVES(x, y)))) ) \]
\[ \rightarrow \beta (\lambda^o P. \exists y. HUMAN(y) \land P x)((\lambda^o x.(\forall x. HUMAN(x) \Rightarrow LOVES(x, y)))) ) \]
\[ \rightarrow \beta (\exists y. HUMAN(y) \land (\forall x. HUMAN(x) \Rightarrow LOVES(x, y))) \]
## Scope Ambiguity in ACG

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Rule</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Sigma_{CG}$</td>
<td>$C_{\text{loves}} : NP \rightarrow NP \rightarrow S$</td>
<td>$\Sigma_{CG}$</td>
</tr>
<tr>
<td>$\lambda$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$C_{\text{everyone}} : (NP \rightarrow S) \rightarrow S$</td>
<td>$\Sigma_{string}$</td>
</tr>
<tr>
<td></td>
<td>$C_{\text{someone}} : (NP \rightarrow S) \rightarrow S$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\Lambda_{\text{amb-string}}$</td>
<td>$C_{\text{loves}} := \lambda^o os.s + loves + o$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$C_{\text{everyone}} := \lambda^o P.P \text{ everyone}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$C_{\text{someone}} := \lambda^o P.P \text{ someone}$</td>
<td></td>
</tr>
<tr>
<td>$C_{\text{everyone}}(\lambda^o x. C_{\text{someone}}(\lambda^o y. C_{\text{loves}} x y))$</td>
<td>$:= \Lambda_{\text{amb-string}}(\lambda^o P.P \text{ everyone})((\lambda^o x. \lambda^o P.P \text{ someone})(\lambda^o y.(\lambda^o os.s + loves + o) y x))$</td>
<td></td>
</tr>
<tr>
<td>$\rightarrow_\beta$</td>
<td>$(\lambda^o P.P \text{ everyone})((\lambda^o x. \lambda^o P.P \text{ someone})(\lambda^o y. x + loves + y))$</td>
<td></td>
</tr>
<tr>
<td>$\rightarrow_\beta$</td>
<td>$(\lambda^o P.P \text{ everyone})(\lambda^o x.(\lambda^o y. x + loves + y) \text{ someone})$</td>
<td></td>
</tr>
<tr>
<td>$\rightarrow_\beta$</td>
<td>$(\lambda^o P.P \text{ everyone})(\lambda^o x. x + loves + \text{ someone})$</td>
<td></td>
</tr>
<tr>
<td>$\rightarrow_\beta$</td>
<td>$(\lambda^o x.x + loves + \text{ someone}) \text{ everyone}$</td>
<td></td>
</tr>
<tr>
<td>$\rightarrow_\beta$</td>
<td>$\text{ everyone + loves + someone}$</td>
<td></td>
</tr>
</tbody>
</table>
Scope Ambiguity in ACG (cont’d)

\[ \Sigma_{CG} \]

\[
\begin{align*}
C_{\text{loves}} &: NP \rightarrow NP \rightarrow S \\
C_{\text{everyone}} &: (NP \rightarrow S) \rightarrow S \\
C_{\text{someone}} &: (NP \rightarrow S) \rightarrow S
\end{align*}
\]

\[ \Sigma_{\text{string}} \]

\[
\begin{align*}
\text{loves} &: \sigma \\
\text{everyone} &: \sigma \\
\text{someone} &: \sigma
\end{align*}
\]

\[ \mathcal{L}_{\text{amb-string}} \]

\[
\begin{align*}
C_{\text{loves}} &= \lambda^o os.s + \text{loves} + o \\
C_{\text{everyone}} &= \lambda^o P.P \text{ everyone} \\
C_{\text{someone}} &= \lambda^o P.P \text{ someone}
\end{align*}
\]

\[
\begin{align*}
C_{\text{someone}}(\lambda^o y. C_{\text{someone}}(\lambda^o x. C_{\text{loves}} x y)) \\
&=: \text{amb-string} \ (\lambda^o P.P \text{ someone})(((\lambda^o y. \lambda^o P.P \text{ everyone})(\lambda^o x. (\lambda^o os.s + \text{loves} + o) y x)) \\
&\rightarrow_\beta (\lambda^o P.P \text{ someone})(((\lambda^o y. \lambda^o P.P \text{ everyone})(\lambda^o x. x + \text{loves} + y)) \\
&\rightarrow_\beta (\lambda^o P.P \text{ someone})(\lambda^o y. (\lambda^o x. x + \text{loves} + y) \text{ everyone}) \\
&\rightarrow_\beta (\lambda^o P.P \text{ someone})(\lambda^o y. \text{everyone + loves} + y) \\
&\rightarrow_\beta (\lambda^o y. \text{everyone + loves} + x) \text{ someone} \\
&\rightarrow_\beta \text{ everyone + loves} + \text{ someone}
\end{align*}
\]
Scope Ambiguity
Non Injective Lexicon

\[ \Lambda(\Sigma_{\text{CG}}) \]

\[ \Lambda(\Sigma_{\text{Log}}) \]

\[ \Lambda(\Sigma_{\text{string}}) \]

\[ \mathcal{G}_{\text{amb-string}} \]

\[ \mathcal{G}_{\text{Log}} \]
Scope Ambiguity in ACG (cont’d)

\[ \Sigma_{CG} \]

\[
\begin{align*}
C_{loves} & : NP \rightarrow NP \rightarrow S \\
C_{everyone} & : (NP \rightarrow S) \rightarrow S \\
C_{someone} & : (NP \rightarrow S) \rightarrow S \\
\end{align*}
\]

\[ \Sigma_{SimpleSyn} \]

\[
\begin{align*}
c_{loves} & : NP \rightarrow NP \rightarrow S \\
c_{everyone} & : NP \\
c_{someone} & : NP \\
\end{align*}
\]

\[ \mathcal{L}_{amb} \]

\[
\begin{align*}
C_{loves} & := \lambda^{o} os.\text{loves} o s \lambda^{o} os.\text{loves} o s \\
C_{everyone} & := \lambda^{o} P. P \text{everyone} c_{everyone} \\
C_{someone} & := \lambda^{o} P. P \text{someone} c_{someone} \\
\end{align*}
\]

\[
\begin{align*}
C_{everyone}(\lambda^{o} x. C_{someone}(\lambda^{o} y. C_{loves} x y)) & := \text{amb } c_{loves} c_{someone} c_{everyone} \\
C_{someone}(\lambda^{o} y. C_{someone}(\lambda^{o} x. C_{loves} x y)) & := \text{amb } c_{loves} c_{someone} c_{everyone} \\
\end{align*}
\]
Scope Ambiguity
Non Injective Lexicon
Conjunction

*John and every kid ran*

\[
C_{\text{and}} := \lambda^o PQR.P(\lambda^o x.Q(\lambda^o y.R(c_{\text{and}} \times y)))
\]

\[
C_{\text{run}} := c_{\text{run}}
\]

\[
C_{\text{kid}} := c_{\text{kid}}
\]

\[
C_{\text{John}} := c_{\text{John}}
\]

\[
c_{\text{run}} : NP \rightarrow S
\]

\[
c_{\text{kid}} : N
\]

\[
c_{\text{John}} : NP
\]

\[
c_{\text{and}} : NP \rightarrow NP \rightarrow NP
\]

\[
c_{\text{run}}(c_{\text{and}}c_{\text{John}}(c_{\text{every}}c_{\text{kid}}))
\]

\[
c_{\text{run}} := \lambda^o s.s + \text{ran}
\]

\[
c_{\text{kid}} := \text{kid}
\]

\[
c_{\text{John}} := \text{John}
\]

\[
c_{\text{and}} := \lambda^o xy.x + \text{and} + y
\]

\[
\Sigma_{\text{string}}
\]

*\((\text{John} + \text{and} + (\text{every} + \text{kid})) + \text{ran}\)*

\[
\Sigma_{\text{Log}}
\]

*\((\text{run} \ j) \land (\forall x.\text{kid} x \Rightarrow \text{run} x)\)*
De re and De dicto Readings

\[ C_{\text{seek}} := \lambda^o x P. P(\lambda^o y. c_{\text{seek}} x y) \]
\[ C_{\text{book}} := c_{\text{book}} \]
\[ C_{\text{seek}} : NP \rightarrow NP \rightarrow S \]
\[ C_{\text{book}} : N \]
\[ C_{\text{seek}} C_{\text{John}}(C_{\text{a}} c_{\text{book}}) \]
\[ c_{\text{seek}} := \lambda^o x y. x + \text{seeks} + y \]
\[ c_{\text{book}} := \text{book} \]

\[ \Sigma_{\text{string}} \]
\[ \text{John} + \text{seeks} + (a + \text{book}) \]

\[ \Sigma_{\text{Log}} \]
\[ \exists y. (\text{book } y) \land (\text{try } j (\lambda^o x. \text{find } x y)) \]
\[ \text{try } j (\lambda^o x. \exists y. (\text{book } y) \land (\text{find } x y)) \]
And More...

So far

- Conjunction
- *De re* and *de dicto* readings
- Coordination of quantified and non-quantified NPs
- VP ellipsis *John saw a kid and so did Bill*
- *de re* and *de dicto* readings
- Quantification and negation (*every kid didn’t run*)

Generalization: the Scoping Constructor

\[
\Gamma \vdash_{\text{TL}} t : \beta \uparrow \alpha \quad \Delta, x : \beta \vdash_{\text{TL}} u : \alpha \\
\frac{}{\Gamma, \Delta \vdash_{\text{TL}} t(\lambda x. u) : \alpha} (E_{\uparrow}) \quad \frac{}{\Gamma \vdash_{\text{TL}} \lambda x.(x t) : \beta \uparrow \alpha} (I_{\uparrow})
\]

Syntactically behaves as a $\beta$ and semantically as a $(\beta \rightarrow \alpha) \rightarrow \alpha$

Given a lexical entre $w : a \in TL(A)$, we have $c_w : a^{\text{syn}}$ and $C_w : a^{\text{sem}}$ such that:

- if $a \in A$ then $a^{\text{syn}} = a$ and $a^{\text{sem}} = a$
- if $a = \alpha \rightarrow \beta$ then $a^{\text{syn}} = \alpha^{\text{syn}} \rightarrow \beta^{\text{syn}}$ and $a^{\text{sem}} = \alpha^{\text{sem}} \rightarrow \beta^{\text{sem}}$.
- if $a = \alpha \uparrow \beta$ then $a^{\text{syn}} = \alpha^{\text{syn}}$ and $a^{\text{sem}} = (\alpha^{\text{sem}} \rightarrow \beta^{\text{sem}}) \rightarrow \beta^{\text{sem}}$.
Conclusion on ACG

- A large number of grammatical formalisms can be encoded into (2nd order) ACG
- Semantic ambiguity in type-logical grammars arises from higher-order (3rd order) types
- Type-logical grammars can be provided a “simple syntactic” level of 2nd order
- We can apply the same higher-order technics to any 2nd order ACG!
- Encoding in a same framework: sharing and comparing analysis
- ACG composition modes: flexible and open architectures
- “Syntax”, “function”, “relation”, “compositionality”, “rule-to-rule” intuitions may be realized by different mathematical notions
- Shallow opposition between syntactocentric and parallel formalisms
Introduction

Meaning

Types and Model Structure

Montague Semantics

Phenomena at the Syntax-Semantics Interface

Abstract Categorial Grammars

Underspecification

Discourse

Selected Bibliography
Main Features [Egg, 2010]

Principles

**Principle** To omit some information from linguistic descriptions

**Aim** To capture alternative realisations in *one single representation*

**Aim** To avoid *enumeration* of the alternatives

Example

- Morphological features in grammar rules:
  
  \[
  S \rightarrow NP_{nom} \ VP \\
  NP_{nom} \rightarrow John \\
  NP_{acc} \rightarrow John
  \]

- Unification grammars:
  
  \[
  S \rightarrow NP[\text{case} = \text{nom}] \ VP \\
  NP \rightarrow John
  \]
Features [Bos, 1995]

- Two levels of description:
  - an object-level of linguistic representations
  - a meta-level of describing these representations
- Fragments of object-level semantic representations
- Glue points (hole) in the fragments
- Relation between glue points and fragments

\[
\forall x. \text{man}(x) \Rightarrow \square \quad \exists y. \text{woman}(y) \land \square
\]

\[\text{loves}(x, y)\]
Description and Models

Getting the Readings

- Each hole gets filled by a fragment
- Respecting the description

We get a **model** a **description**
Semantic Ambiguity: Example

\[ \forall x. \text{man}(x) \Rightarrow \Box \quad \exists y. \text{woman}(y) \land \Box \]

\[ \text{loves}(x, y) \]

\[ \forall x. \text{man}(x) \Rightarrow \Box \]

\[ \exists y. \text{woman}(y) \land \Box \]

\[ \text{loves}(x, y) \]

\[ \forall x. \text{man}(x) \Rightarrow \Box \]

\[ \exists y. \text{woman}(y) \land \Box \]

\[ \text{loves}(x, y) \]
Introduction

Meaning

Types and Model Structure

Montague Semantics

Phenomena at the Syntax-Semantics Interface

Abstract Categorial Grammars

Underspecification

Discourse
- Challenges for Compositionality
- Accessibility According to DRT
- Accessibility According to Discourse Hierarchy
- Dynamic Logic
- Continuation Semantics
Challenging the Compositionality Principle
From Sentence to Discourse

Example (Anaphoric Pronouns and their Antecedents)

- She is extraordinarily beautiful
- Vincent looks at Mia. She dances.
- After he lost the match, Butch left town.

Challenges

- How to resolve the pronouns
- How to represent the pronouns
Pronoun Resolution

Highly Visible Constraints
- Agreement (gender, number...)
- Enough for Resolution?

Example
- Butch threw a TV at the window. It broke.
- Butch threw a vase at the wall. It broke.
Example

George Burns and Gracie Allen: The Salesgirl

Gracie: Oh, yeah... and then Mr. and Mrs. Jones were having matromonial trouble, and my brother was hired to watch Mrs. Jones.

George: Well, I imagine she was a very attractive woman.

Gracie: She was, and my brother watched her day and night for six months.

George: Well, what happened?

Gracie: She finally got a divorce.

George: Mrs. Jones?

Gracie: No, my brother’s wife.
Discourse Challenges for Compositionality

Pronoun Resolution
Approaches

See for instance [Jurafsky and Martin, 2000]

- Heuristic and statistical approaches
- Focus-base approaches
- Centering

But how to represent them?

A Bit of History

- Beginning of the 80’s
First-Order Logic and Discourse

Example

- Mia is a woman
- \texttt{woman(Mia)}
- She loves Vincent.
- \texttt{love(x, Vincent)}
- Mia is a woman. She loves Vincent.
- \texttt{woman(Mia) \land love(x, Vincent) \land x = Mia}

Example

- A woman snorts. She collapses.
- \exists z. (\texttt{woman z \land snort z} \land \texttt{collapse(x) \land x = z})
Intra-Sentential Anaphora

Example

Donkey Sentences

- If John owns a donkey, he beats it.
- $\exists x. (\text{donkey}(x) \land \text{owns}(\text{John}, x) \Rightarrow \text{beats}(\text{John}, y) \land x = y)$
- $\forall x. (\text{donkey}(x) \land \text{owns}(\text{John}, x) \Rightarrow \text{beats}(\text{John}, y) \land x = y)$
Accessibility
Anaphoric pronouns and their antecedents

Example (Existentials, proper nouns, and negation)

- John owns a car. It is red.
- $\exists x \text{car } x \land \text{own~} j \ x \land \text{red } x$
- John doesn’t own a car. *It is red.
- $\neg (\exists x \text{car } x \land \text{own~} j \ x) \land \text{red } x$
- John doesn’t own a car. He is ecology-minded.
- $\neg (\exists x \text{car } x \land \text{own~} j \ x) \land \text{ecolo~} j$
DRT and Context Change Potential

Taking the Context into Account

- A woman snorts
- Statement about the world
- A referent is made available in the context for further reference
Discourse Representation Structures

Example

A woman snorts. She collapses.

<table>
<thead>
<tr>
<th>x, y</th>
</tr>
</thead>
<tbody>
<tr>
<td>woman(x)</td>
</tr>
<tr>
<td>snort(x)</td>
</tr>
<tr>
<td>collapse(y)</td>
</tr>
<tr>
<td>y = x</td>
</tr>
</tbody>
</table>

DRSs

- Discourse Referents
- Conditions
- Conditions can be complex (with logical connectives, except quantifiers, and other DRSs)
Discourse Representation Structures

1. If $x_1, \ldots, x_n (n \geq 0)$ are discourse referents and $\gamma_1, \ldots, \gamma_m (m > 0)$ are conditions then
   \[
   \begin{array}{c|c}
   x_1, \ldots, x_n & \gamma_1 \\
   \hline
   & \vdots \\
   & \gamma_n \\
   \end{array}
   \]
   is a DRS

2. If $R$ is a relation symbol of arity $n$ and $x_1, \ldots, x_n$ are some discourse referents, then
   $R(x_1, \ldots, x_n)$ is a condition

3. If $t_1$ and $t_2$ are discourse referents or constants, then $t_1 = t_2$ is a condition

4. If $K_1$ and $K_2$ are DRSs, then $K_1 \Rightarrow K_2$ is a condition

5. If $K_1$ and $K_2$ are DRSs, then $K_1 \lor K_2$ is a condition

6. If $K$ is a DRSs, then $\neg K$ is a condition

7. Nothing else is a condition or a DRS
Accessibility Constraint

Example

John has a car

\[
x, y
\]

\[
x = \text{John} \\
car(y) \\
owns(x, y)
\]
Accessibility Constraint

Example

John doesn’t have a car. It is red

<table>
<thead>
<tr>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>x, y</td>
</tr>
<tr>
<td>¬ x = John</td>
</tr>
<tr>
<td>car(y)</td>
</tr>
<tr>
<td>owns(x, y)</td>
</tr>
<tr>
<td>red(z)</td>
</tr>
<tr>
<td>z =??</td>
</tr>
</tbody>
</table>

Accessibility and Subordination

1. $K_1$ is immediately subordinate to $K_2$ iff $\neg K_1$ (or $K_1 \Rightarrow K$ for some $K$) is a condition of $K_2$, or if $K_2 \Rightarrow K_1$ is a condition of some $K$

2. $K_1$ is subordinate to $K_2$ ($K_1 < K_2$) iff either
   1. $K_1$ is immediately subordinate to $K_2$, or
   2. there is $K_3$ such that $K_3 < K_2$ and $K_1$ is immediately subordinate to $K_3$

3. For $K$ a DRS, $x$ a referent, and $\gamma$ a condition, $x$ is accessible from $\gamma$ in $K$ iff there are $K_2 \leq K_1 \leq K$ such that $x$ is in the universe of $K_1$ and $\gamma$ in the conditions of $K_2$
Interpreting DRSs

Several Possibilities

- Kamp’s original embedding semantics
- Groenendijk and Stokhof’s dynamic semantics [Groenendijk and Stokhof, 1991]
- Translation into first-order logic [Blackburn and Bos, 2005]

And several drawbacks: destructive assignments.

Available tools

- BOXER [Bos, 2008] and C and C tools
- Grail [Moot, 1999]
Accessibility
Anaphoric Pronouns and Their Antecedents

Example (Hierarchical structure of the discourse [Busquets et al., 2001])

- John went to the hospital.
- Mary broke his nose.
- Peter broke his arm
- He* She even bit him.

What we’ve learned from theories on rhetorical structure

- Segments of the discourse stand in relation to each other
- Depending on the relation (coordinating, subordinating), discourse markers are accessible or not
Dynamic Logics

Technical Issues

- Non-standard interpretation of formulas:
  - Interpretation as relations between assignment functions
    \[ \llbracket \phi \rightarrow \psi \rrbracket = \{ \langle g, h \rangle | h = g & \forall k. h \llbracket \phi \rrbracket k \Rightarrow \exists j. h \llbracket \psi \rrbracket j \} \]
  - \( (\exists x. \phi) \land \psi \Leftrightarrow \exists x. (\phi \land \psi) \) (scope theorem)

- Destructive assignment and variable clash
  - \( \llbracket \phi \Rightarrow \psi \rrbracket = ? \)

Formal Semanticist or Logician?

- What are the useful data to feed the context with?
- How do discourse and sentences combine?
- What are the semantic recipes of the lexical items
- Should I design a new logic? **Continuation semantics**
Continuation Semantics

Accessibility: The Context (Accessible Discourse Referents) as an Argument

Principles [de Groote, 2006]

\[
\begin{align*}
[S] & = \gamma \rightarrow (\gamma \rightarrow t) \rightarrow t \\
[NP] & = (e \rightarrow [S]) \rightarrow [S] \\
[N] & = e \rightarrow [S] \\
[S_1 \cdot S_2] & = \lambda e.\lambda \phi. [S_1] \ e \ (\lambda e'.[S_2] \ e' \ \phi)
\end{align*}
\]

Interpretation of sentences

\[
\begin{array}{c}
e: \gamma \\
\hline
S \\
\hline
\phi: \gamma \rightarrow t
\end{array}
\]

Composition of sentences

\[
\begin{array}{c}
e: \gamma \\
\hline
S_1 \cdot S_2 \\
\hline
\phi: \gamma \rightarrow t
\end{array}
\]

\[\lambda e'.[S_2] \ e' \ \phi: \gamma \rightarrow t\]
CS: an Example

Example

A man is sleeping.
\( \lambda e. \lambda \phi. \exists x. (\text{man } x) \land (\text{sleeping } x) \land (\phi (x :: e)) \)

He is snoring.
\( \lambda e. \lambda \phi. (\text{snoring } (\text{sel } e)) \land (\phi e) \)

Composition of sentences

\[ [T.S] = \lambda e. \lambda \phi. [T] e (\lambda e'. [S] e' \phi) \]

\( \lambda e. \phi.[\lambda e. \phi. \exists x. (\text{man } x) \land (\text{sleeping } x) \land (\phi (x :: e))] e \\
(\lambda e'.(\lambda e. \phi. (\text{snoring } (\text{sel } e)) \land (\phi e)) e' \phi) \)

\( \rightarrow_{\beta} \lambda e. \phi.[\lambda \phi. \exists x. (\text{man } x) \land (\text{sleeping } x) \land (\phi (x :: e))] \\
(\lambda e'.(\text{snoring } (\text{sel } e')) \land (\phi e')) \)

\( \rightarrow_{\beta} \lambda e. \phi.[\exists x. (\text{man } x) \land (\text{sleeping } x) \land ((\lambda e'.(\text{snoring } (\text{sel } e')) \land (\phi e')) (x :: e))] \)

\( \rightarrow_{\beta} \lambda e. \phi.[\exists x. (\text{man } x) \land (\text{sleeping } x) \land ((\text{snoring } (\text{sel } (x :: e)) \land (\phi (x :: e))))] \)
Accessibility
The Context (Accessible Discourse Referents) as an Argument

Composition of sentences

\[ [S_1.S_2] = \lambda \phi.[S_1] e (\lambda e'.[S_2] e' \phi) \]

Example

\[ \begin{array}{c|c|c|c} j & y & \lambda e\phi. \exists y. \text{car } y \land \text{own } j y \land \phi (y :: e) \\
\hline \text{car } y & \text{own } j y & \lambda P e\phi. P (\text{sel } e) e \phi \\
\hline \text{it} & z = ? & \lambda e\phi. \text{red}(\text{sel } e) \land \phi e \\
\hline \text{it is red} & \text{red } z & z = ? \\
\hline \end{array} \]

\[ \begin{array}{c|c|c|c} j & y & \lambda e\phi. \exists y. \text{car } y \land \text{own } j y \land \text{red}(\text{sel } y :: e) \land \phi (y :: e) \\
\hline \text{car } y & \text{own } j y & \text{red } z \\
\hline \text{it is red} & z = y \\
\hline \end{array} \]
Lexical Semantics

**Lexicon**

\[
\begin{align*}
[John] &= \lambda Pe^\phi. P j e^\phi \\
[owns] &= \lambda OS. S(\lambda x. O(\lambda ye^\phi'. \text{own } x y \land \phi' e')) \\
[a] &= \lambda PQe^\phi. \exists y. P y (y :: e) \land Q y (y :: e) \phi \\
[car] &= \lambda xe^\phi . \text{car } x
\end{align*}
\]

**Example**

\[
\begin{align*}
[a][car] &= \lambda Qe^\phi. \exists y. \text{car } y \land Q y (y :: e) \phi \\
[owns][[a][car]] &= \lambda S. S(\lambda x. (\lambda Qe^\phi. \exists y. \text{car } y \land Q y (y :: e) \phi) \\
&\quad (\lambda ye^\phi'. \text{own } x y \land \phi' e')) \\
&= \lambda S. S(\lambda x. (\lambda e^\phi. \exists y. \text{car } y \land (\lambda ye^\phi'. \text{own } x y \land \phi' e') y (y :: e) \phi)) \\
&= \lambda S. S(\lambda x. (\lambda e^\phi. \exists y. \text{car } y \land (\text{own } x y \land \phi (y :: e)))) \\
[owns][[a][car]][John] &= (\lambda Pe^\phi. P j e^\phi)(\lambda x. (\lambda e^\phi. \exists y. \text{car } y \land (\text{own } x y \land \phi (y :: e)))) \\
&= (\lambda e^\phi. (\lambda x. (\lambda e^\phi. \exists y. \text{car } y \land (\text{own } x y \land \phi (y :: e)))) j e^\phi) \\
&= \lambda e^\phi. \exists y. \text{car } y \land (\text{own } j y \land \phi (y :: e))
\end{align*}
\]
A New Dynamic Logic [de Groote, 2007]

Logical connectives

Conjunction: \( A \sqcap B \overset{\Delta}{=} \lambda e . A \ e (\lambda e'. B \ e') \)

Existential quantification: \( \Sigma x . P x \overset{\Delta}{=} \lambda e . \exists x . P x (x :: e) \phi \)

Negation: \( \sim A \overset{\Delta}{=} \lambda e . \neg (A \ e (\lambda e . \top)) \wedge \phi e \)

Universal quantification: \( \Pi x . P x \overset{\Delta}{=} \sim (\Sigma x . \sim (P x) \)

Example

Lexical semantics

<table>
<thead>
<tr>
<th>a, some</th>
<th>Montague semantics</th>
<th>Dynamics semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>every</td>
<td>( \lambda P \ Q . \exists x . P x \wedge Q x )</td>
<td>( \lambda P \ Q . \Sigma x . P x \sqcap Q x )</td>
</tr>
<tr>
<td></td>
<td>( \lambda P \ Q . \forall x . P x \Rightarrow Q x )</td>
<td>( \lambda P \ Q . \Pi x . P x \sqsupset Q x )</td>
</tr>
</tbody>
</table>
See Philippe de Groote's slide
Introduction

Meaning

Types and Model Structure

Montague Semantics

Phenomena at the Syntax-Semantics Interface

Abstract Categorial Grammars

Underspecification

Discourse

Selected Bibliography


*Simpler Syntax.*
Oxford University Press.

Some logical aspects of grammatical structure.

Towards a montagovian account of dynamics.

Yet another dynamic logic.
Presentation at the 4th Lambda Calculus and Formal Grammar workshop.
On the expressive power of Abstract Categorial Grammars: Representing context-free formalisms.  

Grammatical relations and montague grammar.  

Semantic underspecification: Concepts and applications.  
Lecture at JSM’10.  

Semantic construction in feature-based tag.  
In *Proceedings of the 10th Meeting of the European Chapter of the Association for Computational Linguistics (EACL)*.

Dynamic predicate logic.  


Second-order abstract categorial grammars as hyperedge replacement grammars.  
http://www.springerlink.com/content/p644605651088uv6/.

Generating control languages with abstract categorial grammars.  
CSLI Publications.  

*The Mathematics of Language*.  

Universal grammar.  
The proper treatment of quantification in ordinary english.

Grail: an interactive parser for categorial grammars.

Anaphora and the Logic of Change.

Combining montague semantics and discourse representation.
*Linguistics and Philosophy, 19*(2).


