T-(538|725)-MALV, Natural Language Processing Semantics

Hrafn Loftsson¹  Hannes Högni Vilhjálmsson¹

¹School of Computer Science, Reykjavik University

October 2008
Outline

1. Semantics
2. Lambda calculus
3. First-Order Predicate Calculus
Outline

1 Semantics

2 Lambda calculus

3 First-Order Predicate Calculus
Semantics (í. merkingarfræði)

Formal semantics

- Assumes that logic can model language, and, by extension, human thought.
- Formal semantics techniques attempt to map sentences onto logical forms (formulas).
- The logical form can then by used, for example, to determine if a given statement is true or false.
Principle of compositionality

- In some cases, the logical form can be obtained simultaneously while parsing.
- Based on the *Principle of compositionality*:
  - Assumes that it is possible to compose the meaning of a sentence from the meaning of its parts.
Outline

1. Semantics
2. Lambda calculus
3. First-Order Predicate Calculus
What is it?

- A formal system designed to investigate function definition, function application and recursion – Church (1941)
- [http://en.wikipedia.org/wiki/Lambda_calculus](http://en.wikipedia.org/wiki/Lambda_calculus)
- We can use lambda calculus to map constituents to λ-expressions (functions).

Example

- “is a waiter” = \( \lambda x.\text{waiter}(x) \) (called a \( \lambda \)-abstraction)
- \( \lambda x.\text{waiter}(x)(Bill) = \text{waiter}(Bill) \) (called \( \beta \)-reduction or function application)
What is it?

- A formal system designed to investigate function definition, function application and recursion – Church (1941)
- [http://en.wikipedia.org/wiki/Lambda_calculus](http://en.wikipedia.org/wiki/Lambda_calculus)
- We can use lambda calculus to map constituents to \( \lambda \)-expressions (functions).

Example

- “is a waiter” = \( \lambda x.\text{waiter}(x) \) (called a \( \lambda \)-abstraction)
- \( \lambda x.\text{waiter}(x)(\text{Bill}) = \text{waiter}(\text{Bill}) \) (called \( \beta \)-reduction or function application)
We can imitate lambda calculus in Prolog.

- We put \( \lambda \)-expressions into the DCG rules.
- The semantic representation is then constructed while we parse.
- It is common to make \( X^\text{waiter}(X) \) stand for \( \lambda x.\text{waiter}(x) \)
- Let’s consider Prolog code which can derive semantic structure for sentences like:
  - *Mark is a waiter* (Example 1) . . .
    - . . .but first without any semantic analysis (Example 0)
  - and *Mr. Schmidt called Bill* (Example 2)
s --> np, vp.
vp --> verb, np.

np --> ['Bill'].
np --> ['Mark'].
np --> det, noun.

noun --> [waiter].
det --> [a].
verb --> [is].

?- s(['Mark', is, a, waiter], []).
Using DCG rules for compositional analysis

Embed \( \lambda \)-expressions into the rules

- The semantic representation of common nouns or adjectives is that of a property like \( \lambda x.\text{waiter}(x) \)
- Nouns incorporate their semantic representation as an argument in DCG rules:
  - \( \text{noun}(X^\text{waiter}(X)) \rightarrow [\text{waiter}] \).
- “X is a waiter” is roughly equivalent to the predicate \( \text{waiter}(X) \).
- The verb “be” has no real semantic content.
- The semantics of the verb phrase is then simply that of its noun phrase:
  - \( \text{vp}(\text{Semantics}) \rightarrow \text{verb}, \text{np}(\text{Semantics}) \).
Prolog example 1

s(Predicate) --> np(Subject), vp(Subject^Predicate).
vp(Semantics) --> verb, np(Semantics).

np('Bill') --> ['Bill'].
np('Mark') --> ['Mark'].
np(X) --> det, noun(X).

noun(X^waiter(X)) --> [waiter].
det --> [a].
verb --> [is].

?- s(S, ['Mark', is, a, waiter], []).
?- s(waiter('Bill'), L, []).
Semantic composition of verbs

Other verbs than “be” play a role

- Bill rushed ⇒ rushed('Bill').
- *rushed* is intransitive (no object) ⇒ X^rushed(X).
- Mark called Bill ⇒ called('Mark','Bill').
- *called* is transitive ⇒ Y^X^rushed(X,Y).
  - X is the subject, Y is the object.
s(Semantics) --> np(Subject), vp(Subject^Semantics).

vp(Subject^Semantics) --> verb(Subject^Semantics).
vp(Subject^Semantics) --> verb(Object^Subject^Semantics), np(Object).

np('Bill') --> ['Bill'].
np('Mark') --> ['Mark'].

verb(X^rushed(X)) --> [rushed].
verb(Y^X^called(X,Y)) --> [called].

?- s(S, ['Bill', rushed], []).
?- s(S, ['Mark', called, 'Bill'], []).
First-Order Predicate Calculus (FOPC) (í. umsagnarrökfræði)

- We may need to be able to represent the “state of the world” somehow.
  - Objects, animals, people, observable facts, properties of things, etc.
- It is common to use predicate-argument structures (í. umsagnar-rökliðaformgerðir)
- Maps words, phrases and sentences onto symbols and structures characterising things or properties in a given context: the universe of discourse.
- First-Order Predicate Calculus is a convenient tool to represent things and relations.
- http://en.wikipedia.org/wiki/First-order_logic
- Prolog’s base is FOPC.
Assertions are represented by terms (í. líðir).

Simple terms:
- **Constants**, e.g. 'Socrates', 'Pierre'.
- **Variables**, e.g. X, Y, Z.

Compound terms:
- Stand for **predicates or relations** (í. umsagnir), e.g.:
  - person('Pierre').
  - object(table).
  - on('Pierre', table).
Words like \textit{waiter}, \textit{patron}, \textit{yellow}, \textit{hot} are properties that we map onto predicates of arity 1 (one argument).

\begin{itemize}
  \item $\lambda x.\text{waiter}(x)$, in Prolog: $X^\text{waiter}(X)$
    \begin{itemize}
      \item $\lambda x.\text{waiter}(x)(\text{Bill}) = \text{waiter}(\text{Bill})$
    \end{itemize}
  \item $\lambda x.\text{hot}(x)$ \textit{meal}(x), in Prolog: $X^\text{(hot}(X), \text{meal}(X))$
\end{itemize}
Representing verbs and prepositions

- Verbs like *run*, *bring* and *serve* are relations which we map onto predicates of arity 1 or 2 depending on whether they are intransitive or transitive.

  \[ \lambda x.\text{ran}(x), \text{in Prolog: } X^\text{ran}(X) \]
  \[ \lambda x.\text{ran}(x)(\text{Pierre}) = \text{ran}(\text{Pierre}) \]

  \[ \lambda y \lambda x.\text{brought}(x, y), \text{in Prolog: } Y^X^\text{brought}(X, Y) \]
  \[ \text{Roby brought a plate: } \text{brought}('\text{Roby}', Z^\text{plate}(Z)) \]

- Prepositions often link two noun groups:

  \[ Y^X^\text{with}(X, Y), \text{The table with a napkin,} \]
  \[ \text{with}(Z^\text{table}(Z), T^\text{napkin}(T)) \]
A waiter ran.

Every waiter ran.

The waiter ran.

These sentences have a completely different meaning, although the differ only by their determiners.

Quantifiers

- **existential quantifier**, $\exists$.
  - $\exists x. P$, there exists $x$ such that $P$ is true.

- **universal quantifier**, $\forall$.
  - $\forall x. P$, for all $x$, $P$ is true.

- **restricted existential quantifier**, $\exists!$.
  - $\exists! x. P$, there exists exactly one $x$ such that $P$ is true.
<table>
<thead>
<tr>
<th>Sentences</th>
<th>Logic representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A waiter ran</td>
<td>$\exists x (\text{waiter}(x) \land \text{ran}(x))$ exists($X$, waiter($X$), ran($X$))</td>
</tr>
<tr>
<td>Every waiter ran</td>
<td>$\forall x (\text{waiter}(x) \Rightarrow \text{ran}(x))$ all($X$, waiter($X$), ran($X$))</td>
</tr>
<tr>
<td>The waiter ran</td>
<td>$\exists! x (\text{waiter}(x) \land \text{ran}(x))$ the($X$, waiter($X$), ran($X$))</td>
</tr>
<tr>
<td>Sentence</td>
<td>Logical form</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>A waiter brought a meal</td>
<td>$\exists x (\text{waiter}(x) \land \exists y (\text{meal}(y) \land \text{brought}(x,y)))$</td>
</tr>
<tr>
<td></td>
<td>exists(X, waiter(X), exists(Y, meal(Y), brought(X,Y)))</td>
</tr>
<tr>
<td>Every waiter brought a meal</td>
<td>$\forall x (\text{waiter}(x) \Rightarrow \exists y (\text{meal}(y) \land \text{brought}(x,y)))$</td>
</tr>
<tr>
<td></td>
<td>all(X, waiter(X), exists(Y, meal(Y), brought(X,Y)))</td>
</tr>
<tr>
<td>The waiter brought a meal</td>
<td>$\exists! x (\text{waiter}(x) \land \exists y (\text{meal}(y) \land \text{brought}(x,y)))$</td>
</tr>
<tr>
<td></td>
<td>the(X, waiter(X), exists(Y, meal(Y), brought(X,Y)))</td>
</tr>
</tbody>
</table>
We have problems with words like *two*, *three*, *several*, *many*, *this*, *that*, etc.

Instead of using logic quantifier names, we can use the determines themselves as functors of Prolog terms:

- A *waiter brought a meal*
  - a(X, waiter(X), a(Y, meal(Y), brought(X,Y)))

- Two *waiters brought our meals*
  - two(X, waiter(X), our(Y, meal(Y), brought(X,Y)))
Prolog example 3

\[
s(Sem) \rightarrow np((X^SemRest)^Sem), \ vp(X^SemRest).
\]

\[
np((X^SemRest)^Sem) \rightarrow \text{determiner}((X^SemNP)^((X^SemRest)^Sem)), \ noun(X^SemNP).
\]

\[
vp(X^SemRest) \rightarrow \text{verb}(X^SemRest).
\]

\[
vp(X^SemRest) \rightarrow \text{verb}(Y^X^SemVerb), \ np((Y^SemVerb)^SemRest).
\]

\[
noun(X^\text{waiter}(X)) \rightarrow [\text{waiter}].
\]

\[
noun(X^\text{patron}(X)) \rightarrow [\text{patron}].
\]

\[
noun(X^\text{meal}(X)) \rightarrow [\text{meal}].
\]

\[
verb(X^\text{rushed}(X)) \rightarrow [\text{rushed}].
\]

\[
verb(Y^X^\text{ordered}(X,Y)) \rightarrow [\text{ordered}].
\]

\[
verb(Y^X^\text{brought}(X,Y)) \rightarrow [\text{brought}].
\]

\[
\text{determiner}((X^SemNP)^((X^SemRest)^a(X, \text{SemNP}, \text{SemRest}))) \rightarrow [a].
\]

\[
\text{determiner}((X^SemNP)^((X^SemRest)^\text{the}(X, \text{SemNP}, \text{SemRest}))) \rightarrow [\text{the}].
\]

?- s(Sem, [the, patron, ordered, a, meal], []).
Sem = the(\_G549, patron(\_G549), a(\_G576, meal(\_G576), ordered(\_G549, \_G576)))
i.e. Sem = the(X, patron(X), a(Y, meal(Y), ordered(X, Y)))