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

Hrafn Loftsson[1] Hannes Högni Vilhjálmsson[1]

[1]School of Computer Science, Reykjavik University

November 2010

Outline

Outline

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.

Semantics

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

λ 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`
- We can use lambda calculus to map constituents to λ-*expressions* (functions).

Example

- "is a waiter" $= \lambda x.waiter(x)$ (called a λ-abstraction)
- $\lambda x.waiter(x)(Bill) = waiter(Bill)$ (called β-reduction or function application)

Lambda calculus in Prolog

- We can imitate lambda calculus in Prolog.
- We put λ-expressions into the DCG rules.
- The semantic representation is then constructed while we parse.
- It is common to make `X^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)

Prolog example 0

```
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 λ-expressions into the rules

- The semantic representation of common nouns or adjectives is that of a property like λx.waiter(x)
- Nouns incorporate their semantic representation as an argument in DCG rules:
  - `noun(X^waiter(X)) --> [waiter].`
- "X is a waiter" is roughly equivalent to the predicate waiter(X).
- The verb "be" has no real semantic content.
- The semantics of the verb phrase is then simply that of its noun phrase:
  - `vp(Semantics) --> verb, np(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^called(X,Y)`.
  - X is the subject, Y is the object.

Prolog example 2

```
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'], []).

Outline

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.

First-Order Predicate Calculus

- Assertions are represented by **terms** (í. liðir)
- 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).

Representing nouns and adjectives

- Words like *waiter*, *patron*, *yellow*, *hot* are properties that we map onto predicates of arity 1 (one argument).
- $\lambda x.waiter(x)$, in Prolog: `X^waiter(X)`
  - $\lambda x.waiter(x)(Bill) = waiter(Bill)$
- $\lambda x.hot(x)\ meal(x)$, in Prolog: `X^(hot(X), meal(X))`

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.
- $\lambda x.ran(x)$, in Prolog: `X^ran(X)`
  - $\lambda x.ran(x)(Pierre) = ran(Pierre)$
- $\lambda y \lambda x.brought(x,y)$, in Prolog: `Y^X^brought(X,Y)`
  - *Roby brought a plate*: `brought('Roby',Z^plate(Z))`
- Prepositions often link two noun groups:
  - `Y^X^with(X, Y)`, *The table with a napkin*, with(`Z^table(Z), T^napkin(T)`)

Quantifiers (í. magnarar)

1. *A waiter ran.*
2. *Every waiter ran.*
3. *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.

Quantifiers

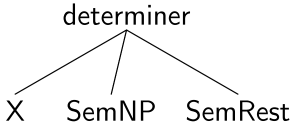
Sentences	Logic representation
A waiter ran	$\exists x(\text{waiter}(x) \wedge \text{ran}(x))$ exists(X, waiter(X), ran(X))
Every waiter ran	$\forall x(\text{waiter}(x) \Rightarrow \text{ran}(x))$ all(X, waiter(X), ran(X))
The waiter ran	$\exists ! x(\text{waiter}(x) \wedge \text{ran}(x))$ the(X, waiter(X), ran(X))

Quantifiers

Sentence	Logical form
A waiter brought a meal	$\exists$x(waiter(x)$\wedge\exists$y(meal(y)$\wedge$ brought(x,y))) exists(X, waiter(X), exists(Y, meal(Y), brought(X,Y)))
Every waiter brought a meal	$\forall$x(waiter(x) $\Rightarrow$ $\exists$y(meal(y)$\wedge$brought(x,y)))) all(X, waiter(X), exists(Y, meal(Y), brought(X,Y)))
The waiter brought a meal	$\exists$!x(waiter(x)$\wedge\exists$y(meal(y)$\wedge$ brought(x,y))) the(X, waiter(X), exists(Y, meal(Y), brought(X,Y)))

Quantifiers

- 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) --> np((X^SemRest)^Sem), vp(X^SemRest).

np((X^SemRest)^Sem) --> determiner((X^SemNP)^(X^SemRest)^Sem), noun(X^SemNP).

vp(X^SemRest) --> verb(X^SemRest).
vp(X^SemRest) --> verb(Y^X^SemVerb), np((Y^SemVerb)^SemRest).

noun(X^waiter(X)) --> [waiter].
noun(X^patron(X)) --> [patron].
noun(X^meal(X)) --> [meal].

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

determiner((X^SemNP)^(X^SemRest)^a(X, SemNP, SemRest)) --> [a].
determiner((X^SemNP)^(X^SemRest)^the(X, SemNP, SemRest)) --> [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)))
```