University of Malta
BSc IT (Hons.) Year III

CSA305: Natural Language Processing

Lecture 8

Semantic Interpretation

- Logical Form
- Rule-to-Rule Hypothesis
- Very simple sentences
- Simple Noun Phrases

Department of Computer Science and AI
2000/01

Logical Form

A common way to model the semantics of NL is by means of a **logical form** which captures the *truth conditions* associated with sentences.

- Truth conditions: conditions under which a sentence is true.
- "Snow is white" if and only if snow is white.
- Truth conditions are associated with complete sentences, but not with other phrase types.

We will assume that the logical form of an English sentence is a sentence of first order logic.

Logical Form of Phrases Examples

Constituent	**Logical Form**
proper noun Mike	logical constant mike
common noun cow	1-place predicate cow(x)
adjective red	1-place predicate red(x)
noun w. adjectives big,red car	conjunction car(x) $\wedge$ red(x) $\wedge$ big(x)
intrans. verb phrase sleeps	1-place predicate sleeps(x)
trans. verb phrase kick	2-place predicate kick(x,y)
ditrans. verb phrase give	3-place predicate give(x,y,z)

Logical Form of Sentences

John kicked fido.

kick(john,fido).

every student wrote a program

$\forall$s student(s) $\rightarrow$ $\exists$p (program(p) $\wedge$ wrote(s,p))

In order to construct the logical form systematically, we appeal to the **Principle of Compositionality**.

Principle of Compositionality

The Principle of Compositionality states that the logical form of a complex phrase formed by means of a constructor $\circ$ is a some function $\Phi_\circ$ of the logical forms of its subphrases.

LF($\alpha \circ \beta$)) = $\Phi_\circ$(LF(α),LF(β)))

In practice this means that we can discover the logical form of a structure by (i) finding the logical form of its parts, and then (ii) combining the logical forms of the parts in some systematic way corresponding to the function $\Phi_\circ$.

The same principle applies to the process of evaluating a complex arithmetic expression. The only difference is that the atomic values are arithmetic quantities, and the operators are arithmetic ones.

Principle of Compositionality Example 1: Arithmetic Expressions

The value V of the following complex expression formed with the constructor "+"

3 + (4 * 2)

is a function Φ_+ of the values of the parts:

V(3 + (4 * 2)) = Φ_+ (V(3), V(4*2))

- Bottoms out with integers, where V(N) = N
- For each "phrase type" there is a corresponding "semantic" rule

  V(X * Y) = Φ_* (V(X), V(Y))

Rule to Rule Hypothesis (Richard Montague)

Each context free syntax rule is associated with a semantic rule that describes how the logical form of the left hand category is composed from the logical forms of its subconsitutents.

- The RR hypothesis is a particular instance of the compositionality principle.
- The most controversial implicit claim is that the syntax drives the semantics.
- In Montague grammar, there are two distinct sets of rules in 1:1 correspondence.
- In DCGs we can model this organisation by annotating each rule with programs that implement the desired semantic combination.

Rule to Rule Examples

1. Example 1
   - **Syntactic Rule:** s $\rightarrow$ np vp
   - **Semantic Rule:** If the logical form of the np is np' and the logical form of the vp is vp', then the logical form of the s is vp'(np') (i.e. vp' *applied to* np').
2. Example 2
   - **Syntactic Rule:** vp $\rightarrow$ v np
   - **Semantic Rule:** If the logical form of the v is v' and the logical form of the np is np', then the logical form of the vp is v'(np')

The Lambda Calculus

The logical form of most *non-sentential* phrases are mostly incomplete logical propositions with certain parts missing.

For example: a verb phrase such as *sleeps* can be thought of as a incomplete proposition that is missing an argument. More usefully, we can treat it as a **function** that maps from (the logical form of) noun phrases to a complete proposition.

We can model this behaviour by extending the logical form language to include lambda expressions

$$\lambda x.\phi$$

where x is the variable marking the argument of the function.

β reduction

The rule of β-reduction says that

$$(\lambda x.\phi)(\mathrm{a})$$

can be reduced to the expression

$$\phi\{x/a\}$$

where all occurrences of x are replaced with a.

$$(\lambda x.x+1)(3)$$

$$\Downarrow$$

$$(3+1)$$

$$\Downarrow$$

$$4$$

Implementation in Prolog

Example Rules

```
s(S)    -->  np(NP), vp(VP)  { reduce(VP,NP,S) }.
vp(VP)  -->  v(V), np(NP)    { reduce(V,NP,VP }.
```

where *reduce(function,argument,result)* is a relation that "applies" *function* to *argument* to yield result *result*.

function	argument	result
λ x . sleeps(x)	uther	sleeps(uther)
λ y . λ . x kicks(x,y)	fido	λ x . kicks(x,fido)

Definition of Reduce

```
reduce(Arg^Function,Arg,Function).

?- reduce(X^sleeps(X),uther,Y).
Y = sleeps(uther).
yes
```


Partial Execution

Given that reduce merely ensures that certain bindings hold, we can eliminate its occurrence by performing the binding in the rule itself, i.e.

```
s(S)    -->  np(NP), vp(VP)  { reduce(VP,NP,S) }.
vp(VP)  -->  v(V), np(NP)    { reduce(V,NP,VP }.
```

are replaced with

```
s(S)   --> np(NP), vp(NP^S)
vp(VP) --> v(NP^VP), np(NP)
```

Full Example: dcg1sem.pl

```
s(S)   --> np(NP), vp(NP^S).
vp(VP) --> v(VP).
vp(VP) --> v(NP^VP), np(NP).
np(N)  --> n(N).

n(LF)   --> [W], { lex(W,n,LF) }.
v(LF)   --> [W], { lex(W,v,LF) }.

lex('John',n,john).
lex('Fido',n,fido).
lex(saw,v,Y^X^saw(X,Y)).
lex(slept,v,X^slept(X)).
```


?- s(LF,['John',slept],[])

```
1  1  Call: s(_295,['John',slept],[]) ?
2  2  Call: np(_782,['John',slept],_784) ?
3  3  Call: n(_782,['John',slept],_784) ?
4  4  Call: 'C'(['John',slept],_1578,_784) ?
4  4  Exit: 'C'(['John',slept],'John',[slept])
5  4  Call: lex('John',n,_782) ?
5  4  Exit: lex('John',n,john) ?
3  3  Exit: n(john,['John',slept],[slept]) ?
2  2  Exit: np(john,['John',slept],[slept]) ?
6  2  Call: vp(john^_295,[slept],[]) ?
7  3  Call: v(john^_295,[slept],[]) ?
8  4  Call: 'C'([slept],_4614,[]) ?
8  4  Exit: 'C'([slept],slept,[]) ?
9  4  Call: lex(slept,v,john^_295) ?
9  4  Exit: lex(slept,v,john^slept(john)) ?
7  3  Exit: v(john^slept(john),[slept],[]) ?
6  2  Exit: vp(john^slept(john),[slept],[]) ?
1  1  Exit: s(slept(john),['John',slept],[]) ?

LF = slept(john) ?
```


?- s(LF,['John',saw,'Fido'],[])

```
| ?- s(LF,['John',saw,'Fido'],[]).
   1  1  Call: s(_295,['John',saw,'Fido'],[]) ?
   2  2  Call: np(_791,['John',saw,'Fido'],_793) ?
   3  3  Call: n(_791,['John',saw,'Fido'],_793) ?
   4  4  Call: 'C'(['John',saw,'Fido'],_1589,_793) ?
   4  4  Exit: 'C'(['John',saw,'Fido'],'John',[saw,'Fido'])
   5  4  Call: lex('John',n,_791) ?
   5  4  Exit: lex('John',n,john) ?
   3  3  Exit: n(john,['John',saw,'Fido'],[saw,'Fido']) ?
   2  2  Exit: np(john,['John',saw,'Fido'],[saw,'Fido']) ?
   6  2  Call: vp(john^_295,[saw,'Fido'],[]) ?
   7  3  Call: v(_4222^john^_295,[saw,'Fido'],_4227) ?
   8  4  Call: 'C'([saw,'Fido'],_4661,_4227) ?
   8  4  Exit: 'C'([saw,'Fido'],saw,['Fido']) ?
   9  4  Call: lex(saw,v,_4222^john^_295) ?
   9  4  Exit: lex(saw,v,_4222^john^saw(john,_4222)) ?
   7  3  Exit: v(_4222^john^saw(john,_4222),[saw,'Fido'],['F
   10  3  Call: np(_4222,['Fido'],[]) ?
   11  4  Call: n(_4222,['Fido'],[]) ?
   12  5  Call: 'C'(['Fido'],_7424,[]) ?
   12  5  Exit: 'C'(['Fido'],'Fido',[]) ?
   13  5  Call: lex('Fido',n,_4222) ?
   13  5  Exit: lex('Fido',n,fido) ?
   11  4  Exit: n(fido,['Fido'],[]) ?
   10  3  Exit: np(fido,['Fido'],[]) ?
   6  2  Exit: vp(john^saw(john,fido),[saw,'Fido'],[]) ?
   1  1  Exit: s(saw(john,fido),['John',saw,'Fido'],[]) ?

LF = saw(john,fido) ?
```


Handling Adjectives

Suppose we want to handle NPs which include adjectives and noun

a big dog
the big green door

- What is the logical form of such NPs?
- What are the logical forms of the parts?
- What are the syntactic rules that describe the structure of the NP?
- How can we *annotate* the rule to yield the correct structure?

Adjectives and Nouns

We will first concentrate on that part of the NP which does not include the quantifier:

big dog
big green door

Syntax

n1 $\rightarrow$ n
n1 $\rightarrow$ adj n1

Semantics

$\lambda x(\text{big}(x) \wedge \text{green}(x) \wedge \text{door}(x))$

Exercises

1. Assuming that the semantics will be represented by a Prolog term such as

   ```
   X^(big(X),green(X))
   ```

   write the lexical entries and grammar rules.

2. Extend the grammar to handle sentences such as

   James is a narrowminded bigot.

   and compute the corresponding semantics:

   ```
   narrowminded(james), bigot(james)
   ```