University of Malta
BSc IT (Hons.) Year III

CSA305: Natural Language Processing

Lecture 9

Semantics II

- Dealing with Adjectives
- Determiners and Noun Phrases
- Generalised Quantifiers

Department of Computer Science and AI
2000/01

Adjectives and the verb "is"

John is a narrowminded bigot

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

- Identify what structure you are trying to build
- Make it parse by adding lexical entries and grammar rules with dummy variables.
- Fix variables to build required structure.

What Structure

For the sentence "John is a narrowminded idiot", assume that a reasonable semantics would be

idiot(john) & narrowminded(john).

To achieve this we further assume that

- John translates as the atom `john`
- Adjectives and common nouns translate as one-place functions mapping NPs into atomic propositions, e.g. λx idiot(x). The Prolog translation is X^idiot(X)

Next we taks a shot at the augmented grammar. To handle the NP pattern

ARTICLE ADJECTIVES NOUN

we introduce the new category "n1"

Grammar First Shot The Rules

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

np(N)  --> n(N).
np(_)  --> d(_), n1(_).

n1(_)   --> n(_).
n1(_)  -->  a(_), n1(_).

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

lex(a,d,_).
lex(an,d,_).
lex(bigot,n,X^bigot(X)).
lex(is,v,_).
lex('Fido',n,fido).
lex('John',n,john).
lex(narrowminded,a,X^narrowminded(X)).
lex(saw,v,Y^X^saw(X,Y)).
lex(slept,v,X^slept(X)).
lex(ugly,a,X^ugly(X)).
```

Second Shot Fix Variables

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

np(N)        --> n(N).
np(N1)       --> d(_), n1(N1).

n1(N)        --> n(N).
n1(X^(A,N))  --> a(X^A), n1(X^N).

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

lex(a,d,_).
lex(an,d,_).
lex(bigot,n,X^bigot(X)).
lex(is,v,X^X).
lex('Fido',n,fido).
lex('John',n,john).
lex(narrowminded,a,X^narrowminded(X)).
lex(saw,v,Y^X^saw(X,Y)).
lex(slept,v,X^slept(X)).
lex(ugly,a,X^ugly(X)).
```


Results

```
?- [gram].
gram compiled, 0.00 sec, 4,524 bytes.

Yes
?- s(X,['John',is,a,narrowminded,bigot],[]).

X = narrowminded(john), bigot(john)
```

Checkpoint

What would you need to do to get the result

X = narrowminded(john) & bigot(john)

Determiners

So far the only determiner we have met is "a", coming after a *copula* – a verb which serves to link subject and predicate directly. In this context, the "a" is present for syntactic reasons – having no semantic function at all (cf. Maltese which leaves out the indefinite article).

However, in other contexts, "a" does have a semantic function, e.g.

a dog barked

Here the meaning can be paraphrased: there is something which is a dog and it barked. This meaning corresponds rather well with the logic sentence:

$\exists x(\text{dog}(x) \wedge \text{barked}(x))$

In general, there is a correspondence of some kind between NL determiners and logic quantifiers. But the correspondence is not always straightforward.

Determiners and Quantifiers

Here are some other examples

A dog barked	$\exists x(d(x) \wedge b(x))$
Every dog barked	$\forall x(d(x) \rightarrow b(x))$
Fido ate a cat	$\exists x(c(x) \wedge a(f, x))$
Fido ate every cat	$\forall x(c(x) \rightarrow ate(f, x))$

Note that in these examples:

- "a" translates as $\exists$
- "every" translates as $\forall$
- The position of the determiner in the syntax tree is different from that in the logic expression. Syntactically, the determiner is embedded inside the noun phrase, semantically it comes out at the top level.
- $\exists$ usually goes with the connective $\wedge$
- $\forall$ usually goes with the connective $\rightarrow$

Other Determiners

We have only dealt with two NL determiners: "a" and "every". There are many others including *the, this, that, which, two, how many, most*.

The fact that determiners are an important aspect of NL, and that their meaning can be approximated by quantified expressions leads to the following questions:

- Are there any general properties shared by all determiners?

  Assumption: all determiners translate to quantified logic expressions
- What are the differences between different determiners?

  Assumption: different determiners are represented by different quantifiers
- How might we represent a quantified expression in a computer?
- What is the meaning of a quantified expression and how can we describe it?

Generalised Quantifiers

All quantified expressions can be regarded as having the abstract structure: of a three place relation:

quant(var,restrictor,scope) where

- *quant* is a quantifier
- *var* is a variable ranging over some universe
- *restrictor* and *scope* are predicates that may mention the variable.

For instance, the logic expression

$\forall$ x (cat(x) $\rightarrow$ animal(x))

can be regarded as having the following generalised representation

all(x,cat(x),animal(x))

Generalised Quantifiers

```
all(x,cat(x),animal(x))
 |  |  |       |
 |  |  |     scope
 |  | restrictor
 | var
quant
```

The idea is that the variable is used to associated restrictor and scope with two sets of bindings, whilst the quantifier controls the relationship between the two sets.

- `all(x,R,S)` is true if $R \subseteq S$
- `exist(x,R,S)`

  This succeeds if there is some x for which both x $\in$ R and x $\in$ S.

Generalised Quantifiers

The advantages of generalised quantifiers are that

- We can create new quantifiers to order, e.g.

  the(X,Restrictor,Scope): exactly one (and no more) value of X satisfies the Restrictor and Scope.

  two(X,Restrictor,Scope): exactly two members of the set satisfying the restrictor also satisfy the scope.

  most(X,Restrictor,Scope): more than half the set satisfying the restrictor also satisfies the scope.
- We can - in a general sense - use this format to analyse the subject-predicate structure of sentences: noun phrases come out as restrictors, whilst verb phrases come out as scopes.

Exercise: Define Prolog predicates for *some, all, most, one*

Building Quantified Structures

Assuming that we are going to build quantified structures of the above type, as the representation of sentences, key questions are:

- How to represent determiners in the lexicon
- How to organise the grammar so that the lexical entries for determiners are harnessed in the right way.

The basic idea is that the lexical entry for a determiner D has the following structure:

$$\lambda\ R.\ \lambda\ S.\ (D\ x)\ R(x) \rightarrow S(x)$$

where R is a restrictor and S is a scope. When such an expression is "applied" to a common noun, R becomes bound and we are left with the representation for a noun phrase.

"every program" λ calculus version

every: $\lambda\ R.\ \lambda\ S.\ (\forall\ x)\ R(x) \rightarrow S(x)$

program: $\lambda\ z.\ prog(z)$

every(program):

$$\lambda\ R.\ \lambda\ S.\ (\forall\ x)\ R(x) \rightarrow S(x)$$
$$(\lambda\ z.\ prog(z))$$
$$\Downarrow$$
$$\lambda\ S.\ (\forall\ x)\ \lambda\ z.\ prog(z)(x) \rightarrow S(x)$$
$$\Downarrow$$
$$\lambda\ S.\ (\forall\ x)\ prog(x) \rightarrow S(x)$$

This last expression is the representation for a noun phrase.

Noun Phrases and Sentences

It should be clear that the expression

$\lambda\ S.\ (\forall\ x)\ prog(x) \rightarrow S(x)$

is essentially a determiner with the restrictor filled in. When it is applied to a scope, it will yield the semantic representation of a sentence. Now we apply it to $\lambda\ x\ halts(x)$, the representation of the verb phrase "halts".

$$(\lambda(S).(\forall p)program(p) \rightarrow S(p))\ (\lambda(x).halts(x))$$
$$\Downarrow$$
$$(\forall p)program(p) \rightarrow (\lambda(x).halts(x))(p)$$
$$\Downarrow$$
$$(\forall p)program(p) \rightarrow halts(p)$$

Representation in Prolog

$\lambda\ R.\ \lambda\ S.\ (\forall\ x)\ R(x) \rightarrow S(x)$

```
R^S^all(X,R1,S1)
where reduce(R,X,R1) and reduce(S,X,S1)
```

Through partial execution we obtain

```
(X^R)^(X^S)^all(X,R,S)
```

Issues

- The scheme we have developed works reasonably well for simple sentences involing *all, every*.
- However, you should note that we have rearranged the semantic rule for sentences. Before, the VP applied to an NP to produce a sentence. Now the NP applies to a VP (i.e. the scope) to produce the sentence.
- To handle this, we must rearrange the arguments in the sentence rule.

What Remains?

- Other determiners

  a, the, some, three, several

- Quantifier Scope

  All the students ran a program

- Questions

- Which students wrote a program?

- Relative clauses

  A man who wrote every program kicked the dog.