More NLP

Parsing Strategies

Prolog example: Multiple parses (multipleparses.pl)

“Time flies like an arrow.”

Parsing strategies

- Parsing is like any other kind of search, with rules used to generate nodes
- Can be done top-down or bottom-up, breadth-first or depth-first
- Unlike some searches, we want to find ALL solutions (parses), so that we can use other strategies to determine which one is correct.
- Problem: How do we know when to terminate the search? cf. left-recursive rules

Left-recursive rules

S → NP VP
VP → V NP
NP → AP NP
NP → N
AP → AP A
AP → A

N → cats
V → love
A → fat

Try a top down and bottom up search, and parse: “Cats love fat cats.”
Do you have a problem? Describe it. How would Prolog handle this?

Agreement and Features

Agreement in natural language

What's wrong with these sentences?

- The cat love dogs.*
- The cats loves a dogs.*
- Cat love some dog.*
- The cat ate and drink.*

Most natural languages require some kind of *agreement* between the words in the sentence.

One way to handle agreement

S → SingNP SingVP.

S → PlurNP PlurVP.

SingVP → SingV SingNP.

PlurVP → PlurV SingNP.

SingNP → SingDet SingNoun.

PlurNP → PlurDet PlurNoun.

SingVP → SingV PlurNP.

...and so on! Problem: You end up with a *lot* of rules, especially when you consider all the kinds of agreement that matter in English. Also, it doesn't capture the generalities of the language.

A better way to handle agreement

Use ***features***. Features keep track of important linguistic attributes like number, tense, person and so on. This way, the grammar need only say that the features must be the same, without saying what they should be. For example:

S → NP(number=X) VP(number=X).

Covers both cases (number=plural and number=singular).

Example

- See featuregrammar.pl .

Exercise

Can you come up with a feature grammar that can parse the following good sentences, without parsing any bad ones?

Good: She has a dog. She has an animal. She has some dogs. She has some animals. She has some milk.

Bad: She has an dog. She has a animal. She has some dog. She has a milk.

Semantics

Semantics

Goal: to go from a sentence like:

“Kim likes Hallowe’en”

to a semantic representation like:

likes(kim, halloween)

automatically!

Compositionality

We assume:

- The meaning of a constituent is composed *solely* of the meanings of its subconstituents.
- So, given a parse tree, we can construct a semantic representation of a node based on the representations of the daughter nodes.

Lambda abstraction

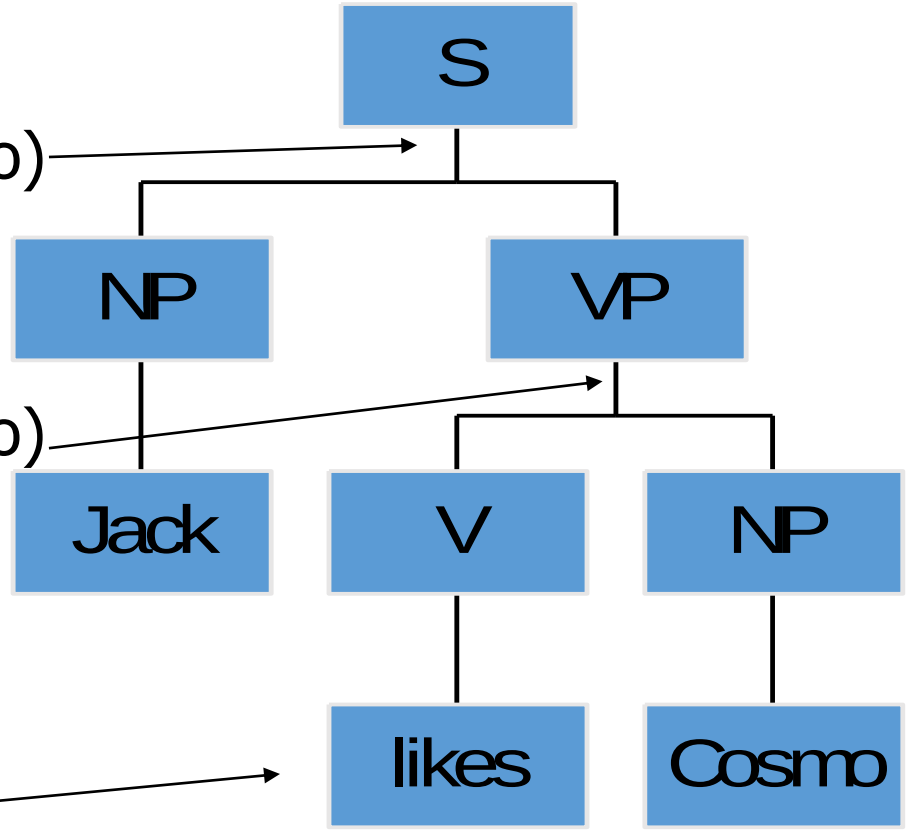
- Formal distinction between an object and its name.
- Expression of incomplete logical forms, such as meanings of words, phrases and other constituents.
- Uses compositionality
- Specifies the contribution of each part to the overall meaning, as well as how parts can be combined.

Lambda calculus

(LZ: likes(Z,cosmo))(jack) = likes(jack, cosmo)

(LYLX: likes(X,Y))(cosmo) = LZ: likes(Z,cosmo)

LYLX: likes(X,Y)



Look at:

- mathgrammar.pl

Exercise

- Write a grammar that parses numbers up to one thousand written as text (e.g. “thirty three” or “fourteen”) and returns the appropriate numerals (e.g. 33 or 14).

Semantics

- Thanks to *compositionality*, representations of the semantics of sentences can be constructed in much the same way that we built up equations for numbers.
- Look at simplesemanticgrammar.pl.
  - Here, ^ means “there exists”.
  - merge/3 merges two lists into a third list.
  - Note that it *doesn’t* distinguish between “a” and “the”, or singular and plural, in the semantic representation – pretty basic!!
  - Took out syntactic features for clarity… - look at “semanticgrammar.pl” for the works!!