LING 388: Language and Computers

Sandiway Fong

Lecture 9
Administrivia

• Reminder
  – Homework 3 due Wednesday night...
  – Have questions? TA Ben Martin is here to help...
Computing Parse Trees

- Getting Prolog to build a representation of the parse tree
  
  \[[S [NP [Det the] man][VP [V took][NP [Det the] book]]]
  
  sentence(np(det(the),man), vp(verb(took), np(det(the),book)))

```
Sentence
   /   
  NP   VP
 |     |
the man Verb
   /     /
   NP   NP
took the book
```
Computing Parse Trees

- **Programming technique:**
  1. add one argument (a Prolog term) to each non-terminal on the left-hand-side of each grammar rule to hold the part of the parse tree it computes
  2. add one argument (a variable) to each non-terminal on the right-hand-side of each grammar rule
    - *the variable stores the value of the subtree computed for the non-terminal*
      - e.g.
      - \( s \rightarrow \text{np}, \text{vp} \).
      - \( s(s(\text{NP},\text{VP})) \rightarrow \text{np}(\text{NP}), \text{vp}(\text{VP}) \).
      - NB. variable naming is not strict: \( s(s(X,Y)) \rightarrow \text{np}(X), \text{vp}(Y) \).
    - **Rule of compositionality:** rules combine sub-phrases into larger phrases
      - *we will combine sub-trees to form larger trees, eventually resulting in the complete parse tree*
Computing Parse Trees

• use a recursive Prolog data structure to represent a parse
• examples
  1. **data structure:** verb(took)
     • functor = verb
     • argument = took
  2. **data structure:** det(the)
     • functor = det
     • argument = the
  3. **data structure:** np(det(the), man)
     • functor = np
     • 1st argument = det(the)
       – functor = det
       – argument = the
     • 2nd argument = man
Computing Parse Trees

- **original DCG**
  - sentence --> np, vp.
  - vp --> verb, np.
  - verb --> [took].
  - np --> det, [man].
  - np --> det, [book].
  - det --> [the].

- **revised DCG**
  - sentence(s(NP,VP)) --> np(NP), vp(VP).
  - vp(vp(V,NP)) --> verb(V), np(NP).
  - verb(v(took)) --> [took].
  - np(np(D,man)) --> det(D), [man].
  - np(np(D,book)) --> det(D), [book].
  - det(det(the)) --> [the].
Computing Parse Trees

• revised DCG
  - sentence(s(NP,VP)) --> np(NP), vp(VP).
  - vp(vp(V,NP)) --> verb(V), np(NP).
  - verb(v(took)) --> [took].
  - np(np(D,man)) --> det(D), [man].
  - np(np(D,book)) --> det(D), [book].
  - det(det(the)) --> [the].

• query (with an extra argument)
  - ?- sentence(P,List,[]).

  - P = parse tree
  - List = sentence

\[
\text{s(np(det(the),man),vp(v(took),np(det(the),man)))}
\]
Examples

• query
  – what parse $P$ corresponds to the sentence “the man took the book”?
  – ?- sentence($P$, [the, man, took, the, book], []).
  – $P = s(np(det(the), man), vp(v(took), np(det(the), book)))$ ? ;
  – no

• query
  – what are the possible parses $P$ and word $X$ for the sentence “the man took the $X$”?
  – ?- sentence($P$, [the, man, took, the, $X$], []).
  – $P = s(np(det(the), man), vp(v(took), np(det(the), man)))$,
  – $X = man$ ? ;
  – $P = s(np(det(the), man), vp(v(took), np(det(the), book)))$,
  – $X = book$ ? ;
  – no
Examples

- **query**
  - given a parse, what is the corresponding Sentence?
  - `?- sentence(s(np(det(the),man),vp(v(took),np(det(the),book))),Sentence,[]).`
  - Sentence = [the,man,took,man,book] ;
  - no

- **query**
  - supply no information, i.e. Parse and Sentence are both variables, what are the possible sentences and parses?
  - `?- sentence(Parse,Sentence,[]).`
  - Parse = s(np(det(the),man),vp(v(took),np(det(the),man))),
    Sentence = [the,man,took,man] ;
  - Parse = s(np(det(the),man),vp(v(took),np(det(the),book))),
    Sentence = [the,man,took,man,book] ;
  - Parse = s(np(det(the),book),vp(v(took),np(det(the),man))),
    Sentence = [the,book,took,man] ;
  - Parse = s(np(det(the),book),vp(v(took),np(det(the),book))),
    Sentence = [the,book,took,man,book] ;
  - no
Printing of Answers

- [http://www.swi-prolog.org/FAQ/AllOutput.html](http://www.swi-prolog.org/FAQ/AllOutput.html)

What to do?

If the system prints an answer that is abbreviated and you want to see it all, type `w` and the system will use plain `write/1` for printing the answer: (the user pressed `w` at the place the diagram says `[write]`). Note the `; true`. This is used to introduce _non-determinism_ that makes Prolog wait after the answer. If the answer is _deterministic_, Prolog prints it with the default settings and prompts for the next command.

```
?- atom_chars(goodbye_prolog, X) ; true.
X = [g, o, o, d, b, y, e, '_' , p|...] [write]
X = [g, o, o, d, b, y, e, '_' , p, r, o, l, o, g]
```
Worked Exercise

• Let's construct a grammar that produces parse tree representations for:
  – I saw the boy with a telescope (2 parses)
  – the boy with a telescope saw me (1 parse)

– **remember:**
  • our grammar should allow for PP attachment to VP and NP

– **note:**
  • let's use lowercase i to represent the first person pronoun (I) to sidestep problems with variables vs. symbols in SWI Prolog