LING 388: Language and Computers

Sandiway Fong

Lecture 8
Adminstrivia

• Review of grammar rules
• One worked exercise
• Homework 3
  – submit by next Wednesday midnight to TA Ben Martin
    • bamartin@email.arizona.edu
Last Time

• Grammars
  – what is a grammar?
    • informally
      – set of “rewrite” rules to parse or “diagram” a sentence
    • derivation
      – top-down (from the sentence symbol), bottom-up (from the words)

• Definite Clause Grammar (DCG) System
  – built-in Prolog support for writing grammars
    – --> is the rewrite symbol
    – [book], [took] terminals are enclosed in brackets
    – sentence, vp, np terminals and non-terminal symbols begin with lowercase letters
Last Time

- **example grammar** *(from last time)*
  - sentence --> np, vp.
  - vp --> verb, np.
  - verb --> [took].
  - np --> [the], [man].
  - np --> [the], [book].

**query format**

- \(?- \text{sentence}(S,[])\).
- \(S = \text{sentence}\) (as a list)
- \([] = \text{empty list}\)

  - i.e. *call the start symbol as a predicate and*
  - *supply two arguments, a list and an empty list*

Prolog translates grammar into underlying Prolog rules you can see the rules using the command
\(?- \text{listing}.\)
Prolog Summary (so far)

- Use a text editor to write your programs (extension: .pl)
- Prolog interpreter
  - /opt/local/bin/swipl (on Mac)
- Definite clause grammar rules
  - LHS --> RHS.
  - Non-terminal symbols (begin with lowercase letter), terminal symbols (same but in [..])
- Load in grammar at the prompt:
  - ?- [filename]. (no extension needed)
- Submit Prolog query at the prompt:
  - ?- nonterminal(List,[]).
  - List: [word1,word2,..,wordn] head-tail notation: [Head|Tail]
- Top-down, left-to-right derivation
- Variables begin with an uppercase letter
Prolog translates DCG rules into ordinary Prolog rules

example
- sentence --> np, vp.
- vp --> verb, np.
- verb --> [took].
- np --> det, [man].
- np --> det, [book].
- det --> [the].
- det --> [a].

variables \( A, B, C \) are all going to be lists
'\( C \)' (\( C, \text{man}, B \)) is true when
man is the head of \( C \)
and \( B \) is the tail of \( C \)
i.e. \( C = [\text{man}|B] \)
The computation rule for DCG rules

- each time we look for a matching rule, we pattern-match against the database from the 1st rule on down

example

- sentence --> np, vp.
- vp --> verb, np.
- verb --> [took].
- np --> det, [man].
- np --> det, [book].
- det --> [the].
- det --> [a].
Prolog Grammar Rules

- **derivation**
  - \( \text{?- sentence([the, man, took, the, book], []).} \)
  - Call: (7) sentence([the, man, took, the, book], []) ? creep
  - Call: (8) np([the, man, took, the, book], _G353) ? creep
  - Call: (9) det([the, man, took, the, book], _G353) ? creep
  - Exit: (9) det([the, man, took, the, book], [man, took, the, book]) ? creep
  - Call: (9) 'C'([man, took, the, book], man, _G357) ? creep
  - Exit: (9) 'C'([man, took, the, book], man, [took, the, book]) ? creep
  - Exit: (8) vp([took, the, book], []) ? creep
  - Call: (9) verb([took, the, book], _G353) ? creep
  - Exit: (9) verb([took, the, book], [took, the, book]) ? creep
  - Call: (9) np([the, book], []) ? creep
  - Call: (10) det([the, book], _G353) ? creep
  - Exit: (10) det([the, book], [book]) ? creep
  - Call: (10) 'C'([book], man, []) ? creep
  - Fail: (10) 'C'([book], man, []) ? creep
  - Redo: (10) det([the, book], _G353) ? creep
  - Fail: (10) det([the, book], _G353) ? creep
  - Redo: (9) np([the, book], []) ? creep
  - Call: (10) det([the, book], _G353) ? creep
  - Exit: (10) det([the, book], [book]) ? creep
  - Call: (10) 'C'([book], book, []) ? creep
  - Exit: (10) 'C'([book], book, []) ? creep
  - Exit: (9) np([the, book], []) ? creep
  - Exit: (8) vp([took, the, book], []) ? creep
  - Exit: (7) sentence([the, man, took, the, book], []) ? creep

- **listing.**
  - \( \text{vp(A, B) :-} \)
  - \( \text{verb(A, C),} \)
  - \( \text{np(C, B).} \)
  - \( \text{np(A, B) :-} \)
  - \( \text{det(A, C),} \)
  - \( \text{'}C'(C, man, B).} \)
  - \( \text{np(A, B) :-} \)
  - \( \text{det(A, C),} \)
  - \( \text{'}C'(C, book, B).} \)
  - \( \text{verb([took|A], A).} \)
  - \( \text{det([the|A], A).} \)
  - \( \text{det([a|A], A).} \)
  - \( \text{sentence(A, B) :-} \)
  - \( \text{np(A, C),} \)
  - \( \text{vp(C, B).} \)

Yes
Worked Exercise

• Let's write grammar rules to handle:
  – I saw the boy with a telescope
    (ambiguous: 2 derivations)
  – the boy with a telescope saw me
    (unambiguous: 1 derivation)
Worked Exercise

• Step 1: decide on the parses you want your grammar to produce
• Unsure? *Use a parser... you already know one*
Worked Exercise

• Step 2: write the rules
  – **Remember**: *terminal and nonterminal symbols should begin with a lowercase letter*

1. s --> np, vp.
2. np --> dt, nn.
3. np --> prp.
4. ...
Worked Exercise

• Step 3: test and debug your grammar

- ?- s([i,saw,the,boy,with,a,telescope],[[]]).
  - true ;
  - true ;
  - false
  - two possible derivations
- ?- s([the,boy,with,a,telescope,saw,me],[[]]).
  - true ;
  - False
  - only one possible derivation
Prolog Terms

• LISP s-exp notation:

```
(ROOT
  (S
    (NP (PRP I))
    (VP (VBD saw)
      (NP (DT the) (NN boy))
      (PP (IN with)
        (NP (DT a) (NN telescope)))))
  (..))
```

Prolog term notation  
(introduced back in lecture 4)

• Simple terms
  – atoms: s, np, vp, dt ...
  – numbers: 1, 45.3 ...
  – variables: List, Sentence ...

• Complex terms:
  general template:
  – functor(argument₁,..,argumentₙ)
  – functor/n (n= arity)
  – functor must be an atom
  – arguments can be simple terms
    or (recursive) complex terms

Examples:
  – nn(boy)
  – dt(the)
  – np(nn(boy), dt(the))
Exercise 1

Part 1: Construct Prolog term representations for the two Stanford parses:
   — exclude the ROOT node

```
(Root
  (S
    (NP (PRP I))
    (VP (VBD saw)
      (NP (DT the) (NN boy))
      (PP (IN with)
        (NP (DT a) (NN telescope))))
  (..))))

(Root
  (S
    (NP (DT The) (NN boy))
    (PP (IN with)
      (NP (DT a) (NN telescope))))
  (VP (VBD saw)
    (NP (PRP me))
  (..)))
```
Exercise 2

4.20 Writing a program to analyse/construct complex terms

functor(?Term, ?Name, ?Arity) [ISO]
   True when Term is a term with functor Name/Arity. If Term is a variable it is unified with a new term whose arguments are all different variables (such a term is called a skeleton). If Term is atomic, Arity will be unified with the integer 0, and Name will be unified with Term. Raises instantiation_error() if Term is unbound and Name/Arity is insufficiently instantiated.

arg(?Arg, +Term, ?Value) [ISO]
   Term should be instantiated to a term, Arg to an integer between 1 and the arity of Term. Value is unified with the Arg-th argument of Term. Arg may also be unbound. In this case Value will be unified with the successive arguments of the term. On successful unification, Arg is unified with the argument number. Backtracking yields alternative solutions. The predicate arg/3 fails silently if Arg = 0 or Arg > arity and raises the exception domain_error(not_less_than_zero, Arg) if Arg < 0.

?Term =.. ?List [ISO]
   List is a list whose head is the functor of Term and the remaining arguments are the arguments of the term. Either side of the predicate may be a variable, but not both. This predicate is called `Univ`. Examples:

   ?- foo(hello, X) =.. List.
   List = [foo, hello, X]

   ?- Term =.. [baz, foo(1)]
   Term = baz(foo(1))
Exercise 2

• using functor and arg
  
  – Term = np(nn(boy),dt(the))
  
  `?- Term = np(nn(boy),dt(the)), functor(Term,Name,Aarity).
  Term = np(nn(boy), dt(the)),
  Name = np,
  Aarity = 2.`

  – What do the following Prolog queries do?
  
  `?- Term = np(nn(boy),dt(the)), arg(1,Term,Argument).
?- Term = np(nn(boy),dt(the)), arg(2,Term,Argument).
?- Term = np(nn(boy),dt(the)), arg(0,Term,Argument).
?- Term = np(nn(boy),dt(the)), arg(3,Term,Argument).`
Homework 3

• Question 1: what does this do?
?– Term = np(nn(boy), dt(the)), arg(N, Term, Argument).

• Question 2: what does this do?
?– Term = np(np(nn(boy), dt(the)), pp(in(with), np(dt(a), nn(telescope)))), arg(2, Term, Arg1), arg(2, Arg1, Arg2).

• Question 3: give a Prolog query that picks out the word *telescope* from the compound term

?– Term = np(np(nn(boy), dt(the)), pp(in(with), np(dt(a), nn(telescope))))
Exercise 3

• Univ predicate:

```
?- Term =.. [np,DT,NN].
Term = baz(foo(1))
```

• Try these Prolog queries:
  - `?- Term =.. [np,DT,NN].`
  - `?- Term =.. [dt, the].`

• What does this do?
  - `?- Term =.. [DT, the].`
Homework 3

• Question 4: Given

?- List1 = [dt, the], List2 = [nn, telescope], List3 = [in, with]

devise a query using =.. that builds the complex term NP:

NP = np(dt(the), nn(telescope))

• Question 5: devise a query using =.. that builds the complex term PP:

PP = pp(in(with), np(dt(the), nn(telescope))}. 