LING 388: Language and Computers

Sandiway Fong

Lecture 4
Administrivia

• Today’s Topics
  – want to make sure everyone is able to run SWI-Prolog
  – SWI-Prolog (6.6.6) on the mac
  – Introduction to Prolog contd.
    Lab class: Unification
  – Homework 2: due next Wednesday by midnight
    • email: Ben Martin bamartin@email.arizona.edu
SWI-Prolog app on Mac

• Two strikes against the SWI-Prolog app:
  1. Default directory is / (root, non-writeable), not your home directory – fixable
  2. Uses the X11 Window System (which runs under OSX) – very slow
SWI-Prolog App on Mac

?- working_directory(X,X).
   X = '/Users/sandiway/Desktop/'.

?- working_directory(X,'/Users/sandiway').
   X = '/Users/sandiway/Desktop/'.

?- working_directory(X,X).
   X = '/Users/sandiway/'.

?- |

Takes two parameters:
(1\textsuperscript{st}) current working directory,
(2\textsuperscript{nd}) the new working directory
SWI-Prolog App on Mac
SWI-Prolog App on Mac

Takes a very long time to fire up because it fires up PCE-EMACS running under the X11 Window System

Better options: aquamacs runs natively under OSX or use your favorite plain text editor
Prolog online resources

• Some background in logic or programming?
• Useful Online Tutorials
  
  — Learn Prolog Now!
  • Patrick Blackburn, Johan Bos & Kristina Striegnitz
  • [http://www.learnprolognow.org](http://www.learnprolognow.org)
  
  — An introduction to Prolog
  • Michel Loiseleur & Nicolas Vigier
  • [http://boklm.eu/prolog/page_0.html](http://boklm.eu/prolog/page_0.html)
Chapter 2
Unification and Proof Search

Chapter 3
Recursion

This chapter has two main goals:

1. To introduce recursive definitions in Prolog.
2. To show that there can be mismatches between procedural meaning.

3.1 Recursive Definitions
   Example 1: Eating
   Example 2: Descendant
   Example 3: Successor
   Example 4: Addition

3.2 Rule Ordering, Goal Ordering, and Termination
3.3 Exercises
3.4 Practical Session

2.1 Unification
   Examples
   The occurs check
   Programming with unification
2.2 Proof Search
2.3 Exercises
2.4 Practical Session

This chapter has two main goals:

1. To discuss unification in Prolog, and to explain how it works.
2. Along the way, we’ll introduce =/2, the built-in predicate for standard equality.
2. To explain the search strategy Prolog uses (Tableaux).

Unification

Data Types:
• Variables (begin with an uppercase letter):
  – e.g. X, Y3, Var...
• Atoms (numbers or begin with a lowercase letter):
  – e.g. x, var, 12.34
• (Complex) terms (functor + 1 or more arguments):
  – e.g. f(1,x), g(X), g(f(1,x))
• Lists:
  – e.g. [] (empty list)
  – [1,2,3] (list containing three numbers)
  – [f(1),x,Y3] (list containing a term, an atom and variable)
Unification

• Lists:
  – []
  – [1,2,3] also can be written as [1|[2,3]]
  – Notation:
    • [head|tail] vertical bar (|) separate the head of the list from the rest of the list, the tail.
  – So [1,2,3] is equal to (and unifiable with) [1|[2,3]]
  – i.e. [1,2,3] = [1|[2,3]]
Unification

– Variables (begin with an uppercase letter):
– e.g. X, Y3, Var...
– Atoms (numbers or begin with a lowercase letter):
– e.g. x, var, 12.34

• Single quotation marks:
  – ‘x’ = x
  – ’12.34’ – not a number, not unifiable with 12.34
  – ‘X’ – not a variable, ‘X’ is not equal to variable X but they are unifiable
  – single quote can be part of an atom: e.g. \’s’
    apostrophe s
Exercise 1

- Using the Prolog interpreter, evaluate the following queries:

1. \(X = \text{mia}, X = \text{vincent.}\) \((, = \text{logical and, conjunction})\)
2. \(X = \text{mia} ; X = \text{vincent.}\) \((; = \text{logical or, disjunction})\)
3. \(X = Y.\)
4. \([1,2,3] = [1|2,3]\)
5. \([1,2,3] = [X|Y]\)
6. \([1,2,3] = [X|3]\)
7. \([1,2,3] = [X|Y]\)

Note: if a list is too long, SWI Prolog uses ... instead of printing the whole list. Type w to get the full display.

\(L = [[a, s, t, a, n, t, e], [c, o, b, a, l, t, o], [p, i, s, t, o, l|...], [a, s, t, o, r|...], [b, a, r, a|...], [s, t, a|...]] \) w
\(L = [[a, s, t, a, n, t, e], [c, o, b, a, l, t, o], [p, i, s, t, o, l, a], [a, s, t, o, r, i, a], [b, a, r, a, t, t, o], [s, t, a, t, a, l, e]]\)
Exercise 2

• Using the Prolog interpreter, evaluate the following queries with complex terms:

1. \( X = \text{father}(\text{mia}). \)
2. \( \text{vincent} = \text{father}(\text{mia}). \)
3. \( X = \text{father}(Y), Y = \text{father}(\text{mia}). \)
4. \( S = \text{s(np(john),vp(v(likes),np(mary)))), S = \text{s(np}(X),\text{vp(v}(Y),\text{np}(Z))). \)
Exercise 3

- According to learnprolognow.org, this shouldn’t work in standard Prologs, try it:

  The occurs check

  Unification is a well-known concept, used in several branches of computer science. It has many unification algorithms are known. But Prolog does not use a standard unification algorithm version of unification. Instead it takes a shortcut. You need to know about this shortcut.

  Consider the following query:

  ?- father(X) = X.

  Do these terms unify or not? A standard unification algorithm would say: “No, they don’t”.

- Try:

  2. father(X) = X, father(Y) = Y, X = Y.
  3. father(X) = Y, father(Y) = X.
Exercise 4

• From section 2.2 on learnprolognow.org, try the following query:

    loves(vincent,mia).
    loves(marcellus,mia).

    jealous(A,B):- loves(A,C), loves(B,C).

Now we pose the query

    ?- jealous(X,Y).

How many solutions are there?
For Homework Exercises

You’ll make use of the following:

1. \+ is the negation operator:
   - vincent = mia. \(false\)
   - \+ vincent = mia. \(true\)
   - \+ (vincent = mia). - same as above

2. there’s a special variable _ (underscore), known as the anonymous variable. It doesn’t do anything and Prolog doesn’t report its value.
   - it’s used when you don’t want to give a variable a name or don’t care what it’s called...

   - Try:
     * \(X = _, Y = _\).
Homework Exercise 1

• Consider Exercise 4 again.
• Assume one cannot be jealous of oneself.
• How would you change the rule or query so that this case is excluded from the answers?
• (show your code and query)
• A word that’s a palindrome is spelt the same way backwards and forwards, e.g. kayak, radar or noon. We can check for “palindrome-hood” using Prolog lists.

• Run the queries:
  – [k,a,y,a,k] = [X,Z,Y,Z,X]. (5 letter palindrome)
  – [c,a,n,o,e] = [X,Z,Y,Z,X].

1. Where can we use the anonymous variable (_) in the palindrome check above?

2. What’s the four letter version of the palindrome check?

• (Give examples)
Homework Exercise 3

• There’s a built-in predicate called reverse(List₁, List₂). Run the following queries:
  – reverse([1,2,3],[3,2,1]).
  – reverse([1,2,3],L).
  – reverse(L,[1,2,3]).

• Explain how you can use reverse/2 to check for palindromes of any length.

• (Give examples)
Homework Exercise 4

• Extra Credit. Write a rule and query (or query) that solves the following puzzle:

**Exercise 2.4** Here are six Italian words:
astante, astoria, baratto, cobalto, pistola, statale.
They are to be arranged, crossword puzzle fashion, in the following grid:

<table>
<thead>
<tr>
<th></th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Hint:** use the list representation

```prolog
word([a,s,t,a,n,t,e]).
word([a,s,t,o,r,i,a]).
word([b,a,r,a,t,t,o]).
word([c,o,b,a,l,t,o]).
word([p,i,s,t,o,l,a]).
word([s,t,a,t,a,l,e]).
```