Laboratory: Prolog (2)

Summary: We continue our exploration of the Prolog programming language.

Prerequisites: The first Prolog lab. Section 4.3 of Tate.

Contents:

- Preparation
- Exercises
  - Exercise 1: Exploring Ancestors
  - Exercise 2: Fun with Lists
  - Exercise 3: Adding to the End of a List
  - Exercise 4: Reversing Lists
  - Exercise 5: Smallest
- For Those with Extra Time

Preparation

Create a directory for the lab.

Exercises

Exercise 1: Exploring Ancestors

A common definition of the `ancestor/2` predicate is

\[
\text{ancestor}(X, Y) :- \text{parent}(X, Y).
\]
\[
\text{ancestor}(X, Y) :- \text{parent}(X, Z), \text{ancestor}(Z, Y).
\]

Here's a very simple database of family relations.

\[
\text{parent}(a, b).
\]
\[
\text{parent}(b, c).
\]
\[
\text{parent}(c, d).
\]
\[
\text{parent}(d, e).
\]

a. Verify that our Prolog interpreter can verify that \(a\) is an ancestor of \(d\).

b. Ask our Prolog interpreter to find the ancestors of \(e\).

c. Suppose we reversed the subgoals in the second rule for `ancestor/2`. 
ancestor(X,Y) :- ancestor(Z,Y), parent(X,Z).

What effect do you expect this change to have on the results of the two previous questions?
d. Check your answer experimentally.

e. Suppose we gave a different set of subgoals in the second rule for ancestor/2
ancestor(X,Y) :- parent(Z,Y), ancestor(X,Z).

What effect do you expect this change to have on the results of the questions in parts a and b?
f. Check your answer experimentally.

g. Suppose we rearranged the rules slightly, as in the following.
ancestor(X,Y) :- ancestor(Z,Y), parent(X,Z).
ancestor(X,Y) :- parent(X,Y).

What effect do you expect this change to have on the results of the questions in parts a and b?
h. Check your answer experimentally.

**Exercise 2: Fun with Lists**

Try the examples in Tate’s *Using Rules in Both Directions* to make sure that you understand lists and the multiple ways in which Prolog solves problems.

**Exercise 3: Adding to the End of a List**

Often, we need to add a single value to the end of a list. The append/3 and concatenate/3 predicates seem like overkill for this task.

a. Write a predicate, add_to_end(L,V,NewL), that holds when NewL is the result of adding V to the end of L.

b. Verify that add_to_end([],a,[a]).

c. Verify that add_to_end([a,b],c,[a,b,c]).

d. Use add_to_end to find what we get when we add d to the end of the list [a,b,c].

e. Use add_to_end to find what we need to add to the end of [a,b] to get [a,b,c].

f. Use add_to_end to find out what list we need to add c to to get [a,b,c].

g. Use add_to_end to find out how to decompose [a,b,c].
Exercise 4: Reversing Lists

a. Write a predicate, `reverse(L,R)`, the holds if R is the reverse of L.

Note that you might find it `concatenate` or `add_to_end` useful in this definition.

b. Check a simple fact (is one list the reverse of another)?

c. Use your predicate to reverse a list placed in the first parameter.

d. What do you expect to have happen if you use your predicate to reverse a list placed in the second parameter, as in `reverse(X, [a,b,c])`?

e. Check your answer experimentally.

Exercise 5: Smallest

Write and experiment with a `smallest(L,S)` predicate that holds when S is a smallest element of L.

For Those with Extra Time

Write a Prolog program to solve the Towers of Hanoi. (How’s that for a vague problem?)