Lab 1. Prolog (1)

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

Prerequisites: Sections 4.1 and 4.2 of Tate. Experience with Linux.

Contents:

- Preparation
- Exercises
  - Exercise 1: Fun with Food
  - Exercise 2: Coloring States
  - Exercise 3: Watching Prolog Compute
  - Exercise 4: Relatives
- For Those with Extra Time

Preparation

Create a directory for the lab.

Exercises

Exercise 1: Fun with Food

a. Try Tate’s food example and see if it behaves as advertised.

b. Add the following two rules.

flavor(salty, cracker).
flavor(crunchy, cracker).

Then, determine what flavors a ritz has.

c. Try a few more experiments that you design.

Exercise 2: Coloring States

a. Try Tate’s map-coloring example and see if it behaves as advertised.

b. What happens if you don’t accept the first result from Prolog? (Keep rejecting results and see what happens.)
Exercise 3: Watching Prolog Compute

Consider the following modification of Tate’s food program, which has been extended to print out some information about the rule being applied.

```prolog
food_type(velveeta, cheese) :- write(rule1), nl.
food_type(ritz, cracker) :- write(rule2), nl.
food_type(spam, meat) :- write(rule3), nl.
food_type(sausage, meat) :- write(rule4), nl.
food_type(jolt, soda) :- write(rule5), nl.
food_type(twinkie, dessert) :- write(rule6), nl.

flavor(sweet, dessert) :- write(rule7), nl.
flavor(savory, meat) :- write(rule8), nl.
flavor(savory, cheese) :- write(rule9), nl.
flavor(sweet, soda) :- write(rule10), nl.

food_flavor(X, Y) :- write(rule11), food_type(X, Z), flavor(Y, Z).
```

a. Load this file and then try a few queries to see the order in which the Prolog interpreter applies rules.

b. Reverse the order of the two preconditions of `food_flavor` and see what effect this has on the order in which the Prolog interpreter applies rules.

Exercise 4: Relatives

Here’s the start of some rules about a family tree.

```prolog
related(X,Y) :- ancestor(X,Y).
related(X,Y) :- ancestor(Y,X).
related(X,Y) :- ancestor(Z,Y), ancestor(Z,X).

ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).

parent(alice,bob).
parent(alice,bill).
parent(andrew,bob).
parent(andrew,bill).
parent(andrew,brianna).
parent(amy,bo).
parent(bob,charles).
parent(billie,charles).
parent(charles,david).
parent(charisma,david).
parent(bo,carlene).
parent(carlene,darlene).
```

a. Determine all of David’s parents.
b. Determine all of Darlene’s ancestors.

c. Write some queries of your own.

d. Extend the database.

**For Those with Extra Time**

Add `write` and `nl` predicates to our family database and trace some interesting queries.