Constructive Logic (15-317), Fall 2017
Assignment 7: Practising Prolog Programming

Ryan Kavanagh[*]

Due Thursday, November 2, 2017

This assignment is due at the beginning of class on the above date and must be submitted electronically via Autolab. Submit your homework as a **tar** archive containing the files: `g4ip.pl`, `coloring.pl`, and `hw7.pdf`.

After submitting via Autolab, please check the submission's contents to ensure it contains what you expect. No points can be given to a submission that isn't there.

1 Implementing a theorem prover (one more time)

Now that you are experts in implementing **G4ip** in Standard ML, it is time to try doing so in Prolog.

Task 1 (15 points)**.** Implement a theorem prover for **G4ip** in Prolog. You must define the predicate predicate `prove/1` for proving a formula, and use the predefined logical operators (see accompanying `g4ip.pl` file). This means that, given a valid *ground* formula `a`, the query `prove`(a) should succeed (with *true* or *yes*).

For your convenience, we have provided you with a shell script to test your implementation. You can invoke it by going

```
$ ./test_g4ip.sh
```

2 Colouring maps

Graph colouring is an interesting problem in graph theory. A graph colouring is an assignment of colours to each vertex such that no two adjacent vertices have the same colour. Of particular interest is a colouring using a minimum number of colours; this number is called the *chromatic number* of the graph. The

[*]Based on an assignment by Giselle Reis.

four-colour theorem states that any planar graph[1] can be coloured using at most four colours. The theorem was proved in 1976 using a computer program, and has caused much controversy (is a computer proof really a proof?). It has since been formally verified using the Coq theorem prover in 2005.

As a consequence of this theorem, any map can be coloured with at most four colours such that no adjacent regions have the same colour. This is because every map can be represented by a planar graph, with one vertex for each region, and an edge between two vertices if and only if their corresponding regions are adjacent.

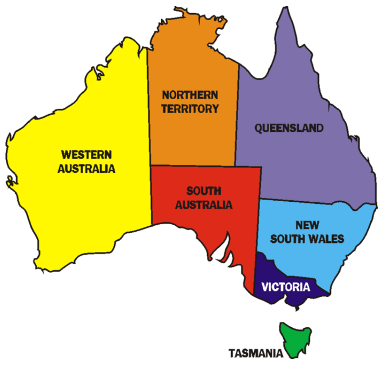

Figure 1: Australia (more colourful than necessary)

Consider, for example, Australia's map in Figure 1. Observe that this map uses more colours than necessary, although this might make it more visually appealing.

Task 2 (15 points)**.** Implement a predicate `color_graph`$(nodes, edges, colours)$ that associates with the graph $(nodes, edges)$ all of the valid 4-colourings of the graph. Submit your implementation in a file named `coloring.pl`.

The predicate `color_graph` should find all valid colourings via backtracking. For efficiency reasons, you may prefer to find all valid colourings without repetition, but we will not be checking this. Once all valid solutions have been found via backtracking, the predicate should fail. You may assume the graph is finite and planar, and your implementation should satisfy the following requirements:

1. You should define a `color/1` predicate with four colours.
2. Assume there are functions `node/1` and `edge/2`.
3. In `color_graph/3`, the first parameter is a list of `node/1` terms, the second

[1]A graph that can be drawn on the plane with no crossing edges.

parameter is a list of `edge/2` terms, and the third parameter is a list of pairs `(a,c)`, where `a` is a node and `c` is a colour.

4. In the terminology of Task 3, the predicate `color_graph` should be multisolution for the mode `color_graph`$(+nodes, +edges, -colouring)$. (Indeed, the four-colour theorem tells us that we will always be able to find a 4-colouring for a planar graph, and the graph's finiteness guarantees there are only finitely many such colourings.)

To clarify the terminology, consider the predicate `childOf`(P, Q), which we claim is multisolution for the mode `childOf`$(+person, -person)$:

```
person(alice).
person(bob).
person(eve).
person(mallory).
childOf(eve, alice).
childOf(eve, bob).
childOf(alice, eve). % Yes, this family tree has a cycle...
childOf(bob, eve).
childOf(mallory, alice).
childOf(mallory, bob).
% Repeated for the sake of contrasting findall and setof below.
childOf(mallory, bob).
```

We can ask Prolog to backtrack and find additional solutions by entering ";" when prompted:

```
| ?- childOf(eve, Parent).

Parent = alice ? ;

Parent = bob

yes
```

Observe that `childOf`$(+person, -person)$ is multisolution because it will always terminate with at least one solution. In contrast, `childOf`$(-person, +person)$ is *not* multisolution, because for no term P does `childOf`$(P, mallory)$ hold.

The built-in Prolog predicates `findall/3` and `setof/3` may be useful in debugging your implementation. You can use the `findall/3` predicate to return a list of all solutions (including repetitions):

```
| ?- findall(P, childOf(mallory, P), Parents).
```

```
Parents = [alice,bob,bob]

yes
```

We can also ask Prolog to return a set of all solutions (in list form) using the `setof/3` predicate:

```
| ?- setof(P, childOf(mallory, P), Parents).

Parents = [alice,bob]

yes
```

For your convenience, we have provided you with a shell script to test your implementation. You can invoke it by going

```
$ ./test_coloring.sh
```

3 Mode and determinacy checking

A mode assignment for an n-ary predicate $\texttt{pred}(arg_1, \ldots, arg_n)$ is a tuple of pairs $((m_1, \tau_1), ..., (m_n, \tau_n))$, where each $m_i \in \{+, -\}$ and each τ_i is a type. We usually write $\texttt{pred}(m_1\tau_1, \ldots, m_n\tau_n)$ to mean $((m_1, \tau_1), \ldots, (m_n, \tau_n))$ is a mode assignment for `pred/n`. For example, we wrote $\texttt{childOf}(+person, -person)$ to describe the mode $((+, person), (-, person))$ for the predicate `childOf/2` above. We call the arg_i such that m_i is $+$ “input arguments”, and the arg_i such that m_i is $-$ “output arguments”. Using a predicate according to a mode assignment consists of instantiating all of its input arguments with terms of the right type, and all output terms with unbound variables.

Recall that a solution to `pred/n` is an instantiation $\texttt{pred}(t_1, ..., t_n)$ with terms $t_1, \ldots, t_n$ such that $\texttt{pred}(t_1, \ldots, t_n)$ can be shown via backwards inference on the rules defining `pred`.

A mode M for a predicate `pred` is *valid* if all solutions for all uses of `pred` according to M satisfy the typing constraints given by M. The *determinacy*[2] of a valid mode M for `pred` tells us how many times we can succeed in finding a solution before terminating for all possible uses `pred` according to M. If all possible uses of `pred` according to M terminate and

- find a solution exactly once, then M is *deterministic*;
- find a solution at most once, then M is *semideterministic*;

[2]This definition is based on Section 6.1 of the Mercury Language Reference Manual.

- find a solution at least once, then M is *multisolution*;
- find a solution at least zero times, then M is *nondeterministic*;
- always fail without producing any solutions, then M has the determinacy *failure*.

Observe that determinacies are not mutually exclusive: all deterministic modes are semideterministic, all semideterministic modes are nondeterministic, etc.

Task 3 (10 points)**.** Consider the following predicates. First, we define the type of binary integers in standard form:

$$\frac{}{\texttt{std}(\texttt{e})} \qquad \frac{\texttt{std}(N)}{\texttt{std}(\texttt{b1}(N))} \qquad \frac{\texttt{std}(\texttt{b0}(N))}{\texttt{std}(\texttt{b0}(\texttt{b0}(N)))} \qquad \frac{\texttt{std}(\texttt{b1}(N))}{\texttt{std}(\texttt{b0}(\texttt{b1}(N)))}$$

Recall the predicate for incrementing binary integers from class:

$$\frac{}{\texttt{binc}(\texttt{e},\texttt{b1}(\texttt{e}))} \qquad \frac{}{\texttt{binc}(\texttt{b0}(\texttt{b0}(N)),\texttt{b1}(\texttt{b0}(N)))}$$

$$\frac{}{\texttt{binc}(\texttt{b0}(\texttt{b1}(N)),\texttt{b1}(\texttt{b1}(N)))} \qquad \frac{\texttt{binc}(M,N)}{\texttt{binc}(\texttt{b1}(M),\texttt{b0}(N))}$$

We define addition of binary integers:

$$\frac{}{\texttt{bplus}(\texttt{e},N,N)} \qquad \frac{}{\texttt{bplus}(\texttt{b0}(N),\texttt{e},\texttt{b0}(N))}$$

$$\frac{\texttt{bplus}(N,M,O)}{\texttt{bplus}(\texttt{b0}(N),\texttt{b0}(M),\texttt{b0}(O))} \qquad \frac{\texttt{bplus}(N,M,O)}{\texttt{bplus}(\texttt{b0}(N),\texttt{b1}(M),\texttt{b1}(O))}$$

$$\frac{\texttt{binc}(M,M') \quad \texttt{bplus}(\texttt{b0}(N),M',O)}{\texttt{bplus}(\texttt{b1}(N),M,O)}$$

Squaring of binary integers, based on the equations

$$(2m)^2 = 4m^2$$
$$(2m+1)^2 = 4m^2 + 4m + 1$$

$$\frac{}{\texttt{bsq}(\texttt{e},\texttt{e})} \qquad \frac{}{\texttt{bsq}(\texttt{b1}(\texttt{e}),\texttt{b1}(\texttt{e}))} \qquad \frac{\texttt{bsq}(M,S)}{\texttt{bsq}(\texttt{b0}(M),\texttt{b0}(\texttt{b0}(S)))}$$

$$\frac{M \neq \texttt{e} \quad \texttt{bsq}(M,S) \quad \texttt{bplus}(\texttt{b0}(\texttt{b0}(S)),\texttt{b1}(\texttt{b0}(M)),T)}{\texttt{bsq}(\texttt{b1}(M),T)}$$

Taking the OR of binary integers:

$$\frac{}{\texttt{bor}(\texttt{e},M,M)} \qquad \frac{}{\texttt{bor}(M,\texttt{e},M)}$$

$$\frac{\texttt{bor}(M, N, O)}{\texttt{bor}(\texttt{b0}(M), \texttt{b0}(N), \texttt{b0}(O))} \qquad \frac{\texttt{bor}(M, N, O)}{\texttt{bor}(\texttt{b0}(M), \texttt{b1}(N), \texttt{b1}(O))}$$

$$\frac{\texttt{bor}(M, N, O)}{\texttt{bor}(\texttt{b1}(M), \texttt{b0}(N), \texttt{b1}(O))} \qquad \frac{\texttt{bor}(M, N, O)}{\texttt{bor}(\texttt{b1}(M), \texttt{b1}(N), \texttt{b1}(O))}$$

Taking the AND of binary integers:

$$\frac{}{\texttt{band}(\texttt{e}, M, \texttt{e})} \qquad \frac{}{\texttt{band}(M, \texttt{e}, \texttt{e})}$$

$$\frac{\texttt{band}(M, N, A)}{\texttt{band}(\texttt{b0}(M), \texttt{b0}(N), \texttt{b0}(A))} \qquad \frac{\texttt{band}(M, N, A)}{\texttt{band}(\texttt{b0}(M), \texttt{b1}(N), \texttt{b0}(A))}$$

$$\frac{\texttt{band}(M, N, A)}{\texttt{band}(\texttt{b1}(M), \texttt{b0}(N), \texttt{b0}(A))} \qquad \frac{\texttt{band}(M, N, A)}{\texttt{band}(\texttt{b1}(M), \texttt{b1}(N), \texttt{b1}(A))}$$

In this question, assume the types are binary integers in standard form, i.e., given by `std`. For each of the predicates listed below:

1. list all of the deterministic and semideterministic modes and state if each of these mode is deterministic or only semideterministic;
2. if the predicate has no such modes, state this.

Additionally, if one of the predicates has a multisolution mode, give it (you only need to give one such example); otherwise, state that none of the predicates have a multisolution mode.

1. `bplus`(M, N, O)
2. `bsq`(M, S)
3. `bor`(M, N, O)
4. `band`(M, N, A)

Can we use the predicates `bplus`(M, N, O) and `bsq`(M, S) to deterministically implement subtraction and taking the square root of binary numbers in standard form, respectively?

Submitting your assignment

Please generate a tarball containing your solution files by running

```
$ tar cf hw7.tar hw7.pdf coloring.pl g4ip.pl
```

and submit the resulting `hw7.tar` file to Autolab.