Alan Smaill
7th Oct 2013

Logic Programming (Programming)

Assessed Practical

(revised 14/10/13)

This coursework is due at 3:00 pm, Monday 21st October, 2013.
Submit your answers in a single file of Prolog source code, which will be tested using Sictus Prolog.

Use the following command on DICE:

```
=>  submit lp 1 <yourFile.pl>
```

Note that questions 2 and 4 make use of some material that will only be covered in the lecture of Thursday 10th October.

1. Write a predicate `prefixes(L,M)` that, given a list $\mathtt{L} = [\mathtt{X}_1, \mathtt{X}_2, \ldots, \mathtt{X}_n]$, calculates a list `M` of all of the prefixes

$$[[\mathtt{X}_1, \ldots, \mathtt{X}_n], [\mathtt{X}_1, \ldots, \mathtt{X}_{n-1}], \ldots, [\mathtt{X}_1, \mathtt{X}_2], [\mathtt{X}_1], [\,]]$$

of `L`.

For example:

```
?- prefixes([],M).
M = [[]].
?- prefixes([1,2],M)
M = [[1,2],[1],[]]
?- prefixes([1,1,1],M).
M = [[1,1,1],[1,1],[1],[]]
```

For full credit, your solution should work whether or not `M` is ground. You may assume that `L` is completely ground when `prefixes/2` is called.

[5 marks]

2. Consider the following example data about teams and goals scored:

```
goals(berwick,12).
goals(montrose,13).
goals(albion,11).
goals(clyde,7).
goals(elgin,14).
```

(a) Define a predicate `goalList/1` such that after solving `goalList(L)`, the variable `L` will be bound to the multiset (bag) of all goals recorded for each team in the knowledge base.

[5 marks]

(b) Define a predicate `belowAverage/1` such that after solving `belowAverage(L)`, the list `L` is bound to a list of all teams whose goal record is under the average.

[8 marks]

For example:

```
?- goals(L).
L = [12,13,11,7,14]
?- belowAverage(L).
L = [clyde,albion]
```

The solution should be independent of the particular example facts above. You may use predicates such as `setof/3`, `bagof/3` or `findall/3`.

3. Suppose we have some arithmetic simplification rules expressed via a Prolog predicate `simp/2`:

```
simp(X + 0, X).     simp(X - 0, X).
simp(X * 0, 0).     simp(X * 1, X).
```

Write a predicate `simplify/2` which takes *any* ground Prolog term whose only function symbols are `+`, `*`, `-`, and applies a simplification rule once somewhere inside it, if there is any applicable subterm. There may be more than one applicable rule, which may lead to multiple solutions. For example:

```
?- simplify(1+2*0,X).
X = 1+0
?- simplify(b+(a-0),X).
X = b+a
?- simplify(((a*1)+b*0)-1,X).
X = (a+b*0)-1;
X = (a*1+0)-1
?- simplify(a+b*c,X).
no
```

You may refer to the above predicate `simp/2` in the definition of `simplify/2`; your solution should not depend strongly on the form of the rules. The `simplify/2` predicate should fail if called when the first argument is a ground term in which no simplification rule applies to any subterm.

[12 marks]

4. Suppose we are given a weighted graph G whose nodes are Prolog atoms $a, b, c, \ldots$ and whose edges are represented by a Prolog relation `edge/3`. The edge weights are positive numbers and each edge has at most one weight. For example, we might represent approximate travel times between cities in Scotland as follows:

```
edge(edi,gla,50).  edge(edi,per,40).
edge(edi,dun,90).  edge(inv,abd,70).
edge(obn,gla,80).  edge(obn,inv,100).
edge(edi,str,30).  edge(gla,str,30).
edge(str,per,30).  edge(per,inv,120).
edge(str,dun,60).  edge(dun,abd,70).
```

In this problem your answers should work for any graph represented by an `edge/3` relation, not just the example above.

(a) The edges should be treated as *undirected*. Define a predicate `undirected_edge/3` such that `undirected_edge(a,b,c)` holds if c is the cost of getting from a to b in one step, in either direction. For example:

```
?- undirected_edge(gla,edi,X).
X = 50
```

[2 marks]

(b) A *simple path* is a list of nodes with no repeats such that each adjacent pair of nodes is linked by `undirected_edge/2`. Define a predicate `path/3` that if a and b are atoms then `path(a,b,P)` succeeds by repeatedly binding the third argument to all simple paths from a to b. For example:

```
?- path(edi,obn,P).
P = [edi,gla,str,per,inv,obn]
P = [edi,gla,str,dun,abd,inv,obn]
...
```

[8 marks]

(c) Define a predicate `pathcost/2` such that, if called with a path P in G as the first argument, succeeds with the second argument bound to the cost of travelling along path P in G. For example:

```
?- pathcost([edi,gla,str,dun],X).
X = 140
```

[4 marks]

(d) Define a predicate `shortest/3` such that if a and b are vertices then `shortest(a,b,P)` succeeds by binding P to a least-cost path from a to b. If there is more than one shortest path, you only need to find one. For example:

```
?- shortest(edi,obn,P).
P = [edi,gla,obn]
?- shortest(gla,abd,P)
P = [gla,str,dun,abd]
```

Hint: Build a list of pairs of all paths and their costs using `setof/3`, then find a least-cost one.

[6 marks]