Symbolic Programming

Thursday 21 May 2015  Goldsmith  9:30-11:30

Dr Tim Fernando

Instructions to Candidates:
Attempt two questions (out of the three given).
All questions carry equal marks. 50 marks per question.

You may not start this examination until you are instructed to do so by the Invigilator.

Materials permitted for this examination:
Non-programmable calculators are permitted for this examination — please indicate the make and model of your calculator on each answer book used.
1. (a) Three translations of the English sentence

Mary owns every lamb that is white

into Prolog are

(i) ‘Mary owns every lamb that is white’.

(ii) own(mary,’every lamb that is white’).

(iii) own(mary,X) :- lamb(X), white(X).

Which of (i)-(iii) are facts? Which are rules? Which translation is best and why?

[9 marks]

(b) How does the Prolog interpreter respond to the following queries:

(i) 3+2 = 5.
(ii) 3+2 = X.
(iii) 3+2 = 2+3.
(iv) 3+2 is X.
(v) X is 3+2.
(vi) [a|[b,c]] = [a,[b,c]].
(vii) [a,b|[c]] = [a|[b,c]].
(viii) [[a]] = [[a]|[]].

[16 marks]

(c) Define a 4-ary predicate split such that split(N,List,Small,Big) is true exactly when List is a list of numbers such that Small consists of all members of List less than N (occurring as many times in Small as in List), and Big consists of all members of List greater than or equal to N (occurring as many times in Big as in List).

For example,

\[
?- \text{split}(3,[5,1,3,4],\text{Small}, \text{Big}).
\]
\[
\text{Small} = [1], \text{Big} = [5,3,4].
\]

[10 marks]
(d) Consider the binary predicate `sumOfPowers` such that `sumOfPowers(N, SoP)` is true exactly if \( N \) is a non-negative integer and \( SoP \) is the sum

\[
\sum_{i=1}^{N} i^i = 1 + 2^2 + \cdots + N^N
\]

of powers \( i^i \) from \( i = 1 \) to \( N \).

For example, since \( 1 + 2^2 + 3^3 = 32 \),

\[
\begin{align*}
&\text{?- sumOfPowers(3, S).} \\
&S = 32.
\end{align*}
\]

Define `sumOfPowers(N, SoP)` in Prolog.

For full credit, make sure your definition is tail-recursive.

[15 marks]
2. (a) Define the binary predicate member such that member(X, List) is true precisely if X is a member of List.

(b) Is the cut below red or green? Explain.

\[ \text{memb}(X, [X|L]) :- !. \]
\[ \text{memb}(X, [Y|L]) :- \text{memb}(X, L). \]

(c) We can define first(X, List) to be true precisely if X is the first element of List as follows

\[ \text{first}(X, [X|_]). \]

Define last(X, List) to be true precisely if X is the last element of List.

(d) Define the binary predicate multiple such that multiple(X, List) is true precisely if X occurs at least twice in List. For example,

\[ \text{?- multiple}(1, [2,1]). \]
no

\[ \text{?- multiple}(1, [1,2,1]). \]
yes

\[ \text{?- multiple}(1, [1,2,1,1]). \]
yes

(e) Define the 3-ary predicate next such that next(A, B, List) is true precisely if List is a list with A and B appearing consecutively in it.

For example,

\[ \text{?- next}(A, B, [x,d,c]). \]
\( A = x, \) \( B = d ? ; \)
\( A = d, \) \( B = c ? ; \)
no.
(f) Let the 6-ary predicate mem3 be defined as follows

\[
\text{mem3}(X_1,X_2,X_3,L_1,L_2,L_3) :\rightleftharpoons \text{member}(X_1,L_1), \\
\quad \text{member}(X_2,L_2), \\
\quad \text{member}(X_3,L_3).
\]

Your task is to describe how Prolog backtracks on \text{mem3} by defining a 9-ary predicate

\[
\text{next3}(A_1,A_2,A_3,B_1,B_2,B_3,L_1,L_2,L_3)
\]

that serves the role for \text{mem3}(X_1,X_2,X_3,L_1,L_2,L_3) that the predicate \text{next}(A,B,List) from part (e) serves for \text{member}(X,List). More precisely, the requirement is that for all ground (i.e., variable-free) terms \(a_1, a_2, a_3, b_1, b_2, b_3, l_1, l_2, l_3\), we have \text{next3}(a_1,a_2,a_3,b_1,b_2,b_3,l_1,l_2,l_3) true precisely if the query

\[
| ?- \text{mem3}(X_1,X_2,X_3,l_1,l_2,l_3).
\]

eventually returns the instantiations

\[
X_1 = a_1, \; X_2 = a_2, \; X_3 = a_3
\]

followed (on backtracking) immediately by

\[
X_1 = b_1, \; X_2 = b_2, \; X_3 = b_3.
\]

For example, since

\[
| ?- \text{mem3}(X_1,X_2,X_3,1,2,[a],[x,y]).
X_1 = 1, \; X_2 = a, \; X_3 = x ? ; \\
X_1 = 1, \; X_2 = a, \; X_3 = y ? ; \\
X_1 = 2, \; X_2 = a, \; X_3 = x ? ; \\
X_1 = 2, \; X_2 = a, \; X_3 = y ? ;
\]

no

we have \text{next3}(A_1,A_2,A_3,B_1,B_2,B_3,1,2,[a],[x,y]) true for the following three instantiations:

\[
A_1 = 1, \; A_2 = a, \; A_3 = x, \; B_1 = 1, \; B_2 = a, \; B_3 = y
\]

and

\[
A_1 = 1, \; A_2 = a, \; A_3 = y, \; B_1 = 2, \; B_2 = a, \; B_3 = x
\]

and

\[
A_1 = 2, \; A_2 = a, \; A_3 = x, \; B_1 = 2, \; B_2 = a, \; B_3 = y.
\]

[20 marks]
3. (a) Define a Definite Clause Grammar (DCG) for strings \( u2v \) where \( u \) and \( v \) are strings over the alphabet \( \{0,1\} \) such that the number of 1's in \( u \) is twice the number of 0's in \( v \). For example,

\[
\text{\texttt{L}} = [1,0]; \\
\text{\texttt{no}}
\]

[15 marks]

(b) What are difference lists and how are they useful?

[5 marks]

(c) Write your DCG in part (a) using ordinary Prolog notation, making the difference lists explicit.

[10 marks]

(d) Write a DCG that given a non-negative integer \( \text{Half} \), accepts a list of integers \( \geq 1 \) that add up to twice \( \text{Half} \). For example,

\[
\text{\texttt{L}} = [4] ? ; \\
\text{\texttt{L}} = [3,1] ? ; \\
\text{\texttt{L}} = [2,2] ? ; \\
\text{\texttt{L}} = [1,3] ? ; \\
\text{\texttt{L}} = [2,1,1] ? ; \\
\text{\texttt{L}} = [1,2,1] ? ; \\
\text{\texttt{L}} = [1,1,2] ? ; \\
\text{\texttt{L}} = [1,1,1,1] ? ; \\
\text{\texttt{no}}
\]

It may be useful to write a predicate \( \text{mkList} (+\text{Num}, ?\text{List}) \) that returns a list of integers from \( \text{Num} \) down to 1. For example,

\[
\text{\texttt{L}} = [3,2,1] ? ; \\
\text{\texttt{no}}
\]

[20 marks]