Department of Informatics and Media
Information Systems

Declarative problem solving methods, 7.5 hp
2IS008
Written exam 2017-11-11

- Answer each question on a separate sheet.
- Write your code number on each paper.
- Don't answer any questions in this paper, it should not be handed in.
- The exam consists of 7 questions. Total score on the exam is 100 points.
- Last in the paper you find an Appendix with built-in predicates in Prolog.
- *Note, that if you use predicates that can be built-in in some Prolog systems, e.g. member, give the code for them in the written exam.*

Good luck!

Task 1 Theory (3 + 6 + 3 + 3 points)

a) A Prolog program consists of Horn clauses. In logic a Horn clause has the form
$\forall x1...\forall x_n (A \leftarrow B_1 \wedge \ldots \wedge B_i)$
The Horn clauses can have three different formats in Prolog. Present these three formats in Prolog and describe what the clauses denote.

b) The derivation in Prolog is based on the resolution rule and the sentences involved are the Horn clauses. Illustrate how the resolution rule works through deriving that
declarative_progr_language('Prolog') is true when the following Horn clauses are given:

declarative_progr_language (X) ← logic_based(X)
declarative_progr_language(X) ← functional_based(X)

logic_based(X) ← representation(X,'Horn clauses'), reasoning(X, 'resolution rule')

representation('Prolog','Horn clauses') ←
reasoning('Prolog','resolution rule') ←

c) When applying the resolution rule the terms *resolvent* and *parents to the resolvent* can be used. Explain these two terms and illustrate them utilising the derivation in **b)**.

d) The proof procedure in Prolog is an *indirect proof*. Describe what an indirect proof is and how the proof is organized. Use the given example in **b)** for your explanation.

Task 2 Unification (10 points)

Perform the unifications. If the expressions cannot be unified, describe why.

a) `a(X,4,Z,W) = a(5,Y,X,Y).`

b) `b(X,p,r(Y,Z)) = b(W,Y,r(2,4)).`

c) `c(p(p(X)),p(Z),4) = c(Z,X,Y).`

d) `d(f(X),s(Y)) = d(Y,Z).`

e) `e(p(X),q(Y,p(2,5))) = e(p(5),q(p(X),Z)).`

f) `f([10,12,14],L) = f([X,Y|L],[Z|R]).`

g) `g([1|List], List, [E,[2]]) = g(F,[2],F ).`

h) `h([X|[Y|Z]],[X,Y,Z]) = h([1|[2]],L).`

Task 3 Rewrite programs (6 + 6 points)

a) Rewrite the (incorrect) program `select(List, IntegerList, CharList)` below, according to the following definition:
The recursive program divides a list of numbers and characters `List` into two lists, one contains the numbers and the other one contains the characters.

Examples:

```
?- select([1,a,8,r,16,9,d], IntegerList, CharList).
IntegerList = [1,8,16,9],
CharList = [a,r,d];
no

?- select([x,y,5,z], IntegerList, CharList).
IntegerList = [5],
CharList = [x,y,z];
no

select([],L,L).
select([Head|Rest],[Head|Integers],Chars):-
      integer(Head),
      select(([Head|Rest],Integers,New_Chars).
select([Head|Rest],Integers,[Head|Chars]):-
      \+integer(Head),
      select(Rest,New_Integers,[Head|Chars]).
```

b) Rewrite the (incorrect) program `count_vowels(List, Vowels)` below, according to the following definition:
The recursive program counts the number of vowels "a,e,i,o,u,y" in a list of characters `List`, using `member` predicate which defines if an element is a member in a list.

Examples:

```
?- count_vowels([h,e,l,l,o], Vowels).
Vowels = 2;
no

?- count_vowels([u,m,b,r,e,l,l,a], Vowels).
Vowels = 3;
no

vowel(X):- member(X,[a,e,i,o,u,y]).

count_vowels([],1).
count_vowels([Head|Rest],N):-
      vowel(Head),
      count_vowels(Rest,N),
      N is N+1.
count_vowels([Head|Rest],N):-
      \+ vowel(Head),
      N is N1 + 0,
      count_vowels(Rest,N).
```

Task 4 Work with a database (2 + 3 + 3 + 4 + 5 + 7 + 7 points)

Given is a database with information about student nations in Uppsala. The predicate `nation(Abbreviation, Name, address(Street,Number))` defines a an abbreviation for the nation, the name of it and the structure `address` with the arguments street and number. The predicate `geography(Abbreviation, Geografic_regions)` defines the geographic regions where the members in the nations mainly came from historically. The predicate `members(Abbreviation, Number_of_members)` gives the number of members belonging to a nation.

The built-in predicates `setof, bagof` or `findall` could be useful in some of the subtasks. When asked for a list, you should not give the list but the question that can generate the list.

```
:- dynamic nation/3, geography/2, members/2.

% nation(Abbreviation,Name, Address)
nation('GH','Gästrike-Hälsinge nation', address('Trädgårdsgatan', 9)).
nation('V-Dala','Västmanlands-Dala nation', address('S:t Larsgatan', 13)).
nation('NN','Norrlands nation', address('Västra Ågatan', 14)).
nation('ÖG','Östgöta nation', address('Trädgårdsgatan', 15)).
nation('Stocken','Stockholms nation', address('Drottninggatan', 11)).
nation('UN','Uplands nation', address('S:Larsgatan', 11)).
nation('Guten','Gotland', address('Östra Ågatan', 13)).
nation('KN','Kalmar nation', address('Svartmangatan', 3)).
nation('GN','Göteborgs nation',address('S:t Larsgatan',7)).
nation('SNerikes','Södermanlands-Nerikes nation', address('S:t Larsgatan',4)).

% geography(Abbreviation, Geografic_regions).
geography('GH',[ 'Gästrikland', 'Hälsingland']).
geography('V-Dala',[ 'Västmanland', 'Dalarna']).
geography('NN',['Norrbotten', 'Västerbotten', 'Jämtland', 'Västernorrland',
       'Gästrikland', 'Hälsingland']).
geography('ÖG',['Östergötland']).
geography('Stocken',['Stockholm']).
geography('UN',['Uppland', 'Stockholm']).
geography('Guten',[ 'Gotland']).
geography('KN',[ 'Småland']).
geography('GN',['Göteborg','Bohuslän']).
geography('SNerikes',[ 'Södermanland', 'Stockholm', 'Närke']).

% members(Abbreviation,Number_of_members
members('GH', 1700).
members('V-Dala', 5500).
members('NN', 8000).
members('ÖG', 5500).
members('Stocken', 4400).
members('UN', 2000).
members('Guten', 600).
members('KN', 1500).
members('GN', 500).
members('SNerikes', 4500).
```

a) List all the abbreviations for the nations found in the database.

b) List the names on the nations that have 2000 or more members. If there are none, the answer should be an empty list.

c) List the names of all nations situated at Trädgårdsgatan. If there are none the answer should be no.

d) Make a list consisting of elements in form of a structure
`nation_info(Nation_name, Abbreviation)`
where the structure's arguments are the name of a nation and the abbreviation for it. The list should be sorted.

e) Write a program `update_address(Nation_name,New_street,New_number)` that can update the structure `address(Street, Number)` and store the new information in the database. The old information should be deleted. Then let the program write the new information utilising the built_in predicate `write` according to the format below.

Ex:
```
?- update_address('Uplands nation', 'S:t Johannesgatan',47).

Information after the address is updated:
nation('UN','Uplands nation', address('S:t Johannesgatan', 47)).
```

f) Historically, you were supposed to choose nation based on where you came from in Sweden. This is, of course, not the case anymore. Presuppose that you still do this. Write a program `choose_nation(Geographic_region, List_of_nations)` that lists the possible nations' names that you could choose between based on a geographic area. If there is no recommendation, the list should be empty.

Ex:
```
?- choose_nation('Stockholm', List_of_nations).
List_of_nations = ['Stockholms nation', 'Uplands nation', 'Södermanlands-
Nerikes nation']
```

g) Write a program that counts the sum of all members in the different nations.

Ex:
```
?- number_of_members(Num).
Num = 34200
```

Task 5 Working with text (7 points)

Write a program `decode(Text,Message)` that decode a text by taking away all characters in the code that are not letters, spaces or full stop. The order of the characters in the code should be kept.

The upper case letters have an ASCII code in the interval 65-90.
The lower case letters have an ASCII code in the interval 97-122.
A space has the ASCII code 32.
A full stop has the ASCII code 46.

Example:

```
?- decode('*2I3t i%s+ fu#56n 7t?8o s9tud!y.',Message).
Message = 'It is fun to study.'
```

Task 6 Check the search space (7 + 3 + 4 + 3 points)

The program `sort_together(List1,List2,SList)` defines a relation between three lists. The elements in `List1` and `List2` are sorted in increasing order. `SList` should consist of the all elements in the first two lists and it should also be ordered. If one specific element is found in both lists it should occur twice in `SList` .

```
sort_together(A,[],A).
sort_together([],B,B).
sort_together([E|A],[F|B],[E|C]):-
      E < F,
      sort_together(A,[F|B],C).
sort_together([E|A],[F|B],[F|C]):-
      sort_together([E|A],B,C).
```

a) Give all alternative answers that the program can give on the following question:

```
?- sort_together([1,4],[2,11],SList).
```

b) Rewrite the program in a declarative way so that only correct answers are generated when asking Prolog for alternative answers.

c) Another way to make sure that only correct answers are generated, than the solution in **b)**, is to use cut (i.e. !). Rewrite the program with cut and describe whether you have used a red or green cut.

d) Describe what it means to have a tail recursive program. Is the program tail recursive?

Task 7 Problem solving methods (8 points)

In the last lecture we studied a program that solves the problem of moving a farmer, a wolf, a goat, and a cabbage to the other side of a river. The farmer, the wolf, the goat, and the cabbage are all on the north bank of a river and the problem is how to move them to the south bank. The farmer has a rowing boat and he can take one passenger at a time. The goat cannot be left with the wolf unless the farmer is with them. The cabbage also counts as a passenger and cannot be left with the goat if the farmer is not there.

In the course we have mainly described the problem solving methods "divide-and-conquer" and "generate-and-test". Describe the two problem solving methods and how these methods are utilised in the given program.

```
problem_solving(Solution):-
      state(n,n,n,n,[st(n,n,n,n)],Solution).

state(s,s,s,s,TList,TList).
state(F,W,G,C,TList1,TList):-
      opposite(F,F1),
      passenger(F1,W,G,C,W1,G1,C1),
      safe(F1,W1,G1,C1),
      \+ member(st(F1,W1,G1,C1),TList1),
      state(F1,W1,G1,C1,[st(F1,W1,G1,C1)|TList1],TList).

% opposite side to south is north and vice versa
opposite(s,n).
opposite(n,s).

% passenger(Farmer,Wolf,Goat,Cabbage,Wolf_new,Goat_new,Cabbage_new)
% The farmer and the wolf change side
passenger(X1,X,Y,Z,X1,Y,Z):-
      opposite(X,X1).
% The farmer and the goat change side
passenger(Y1,X,Y,Z,X,Y1,Z):-
      opposite(Y,Y1).
% The farmer and the cabbage change side
passenger(Z1,X,Y,Z,X,Y,Z1):-
      opposite(Z,Z1).
% Only the farmer change side
passenger(X1,X,Y,Z,X,Y,Z).

safe(X,Y,X,Z). % Farmer and goat are together
% The goat is on the other side than the other three
safe(X,X,Y,X):-
      opposite(X,Y).

member(X,[X|_]).
member(X,[Y|Rest]):-
      member(X,Rest).
```

Example of a question:

```
?- problem_solving(Solution).
```

Appendix with built-in predicates in Prolog

`<`	less than
`=<`	less than or equal to
`>`	greater than
`>=`	greater than or equal to
`=`	the two arguments can be unified
`=/=`	the two arguments cannot be unified
`X is Y`	`X` gets the value that the arithmetical expression `Y` has, `Y` must contain instantiated variables
`==`	the two arguments are identical
`\==`	the two arguments are not identical
`X+Y`	addition
`X-Y`	subtraction
`X*Y`	multiplication
`X/Y`	division
`X//Y`	integer division
`X mod Y`	gives the rest from the division between `X` and `Y`
`integer(X)`	is true if `X` is an integer
`atom(X)`	is true if `X` is an atom
`atomic(X)`	is true if `X` is a number or an atom
`read(X)`	reads a term
`write(X)`	writes a term
`nl`	gives a new line
`fail`	the predicate always fails, backtracking is started
`true`	the predicate always succeeds
`see(File)`	opens a file for reading or switches the input stream to `File`
`seen`	closes file
`tell(File)`	opens file for writing or switches the output stream to `File`
`told`	closes file
`listing(P)`	lists the predicate `P`
`assert(P)`	stores a predicate `P`
`retract(P)`	deletes a stored predicate `P`
`name(X,AsciiList)`	the relation between the term `X` and a list of Ascii-codes for the term.

Ex: `?-name(abö,L) ger L=[97,98,124]`
`?-name(X,[97,46,99,33,63,44]) gives X=a.c!?,`

`!`	cut
`\+`	negation as failure

`bagof(X,P(X),L)`
The first argument decides the elements in the output list, the second is the question, and the third is the list with the answers
`findall(X,P(X),L)`
The predicate works like bagof except that all variables in P are existentially quantified
`setof(X,P(X),L)`
The predicate works like bagof with the addition that the elements in L are sorted and duplicate items are deleted.