Departamento de Ciência de Computadores - FCUP

Logic Programming Exam — PART 2

(Duration: 1h 30min)

Date: January 7th, 2017

1) Given a list of sublists, where each sublist has two elements representing an interval $[X, Y], X \leq Y$, for example, `[[5,7],[1,2],[6,10],[12,15]]`, write a program that returns a pair with the smallest value for the beginning of the interval, and the largest value for the end of the interval. In the example the result should be: `[1,15]`. You can assume that each sublist is in ascending order.

```
:- use_module(library(lists)).

smallestInterval(List,[Min,Max]) :-
   flatten(List,FlattenedList),
   sort(FlattenedList,[Min|Sorted]),
   last([Min|Sorted],Max).
```

Inefficient, but I accepted solutions along these lines.

2) Write a program to remove a sublist of elements indicated by their interval indice. For example:

```
?- drop([a,b,c,d,e,f,g,h,i,k],3-6,X).
X = [a,b,g,h,i,k]
```

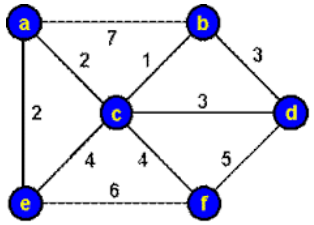
```
drop(List,B-E,Result) :-
   drop(List,1,B-E,Result).

drop([E|Es],N,I-N,Es) :- !. % end of the interval, return remaining list
drop([E|Es],I,I-N,L) :- !.  % beginning of the interval, discard E
   I1 is I + 1,
   drop(Es,I1,I-N,L).
drop([E|Es1],J,I-N,[E|Es2]) :- % beginning not reached yet, keep E
   I1 is I + 1.
   drop(Es1,I1,I-N,Es2).
```

3) A positive integer number is called Armstrong of order n if $abcd\ldots = a^n + b^n + c^n + d^n + \ldots$. For example, $1634 = 1^4 + 6^4 + 3^4 + 4^4$. In the case of Armstrong numbers of three digits, the sum of the cubes of each digit is the number itself. For example, $153 = 1 * 1 * 1 + 5 * 5 * 5 + 3 * 3 * 3$. Write a Prolog program that checks if a number is Armstrong.

```
:- use_module(library(lists)).
:- use_module(library(apply)).

armstrong(N) :-
   number_codes(N,M),
   maplist(plus(48),List,M),
   length(List,P),
   sum(List,P,Sum),
```

```
    N = Sum.

sum(List,P,Sum) :-
    sum(List,1,P,0,Sum).

sum([],_,_,Sum,Sum).
sum([H|T],I,P,Acc0,Sum) :-
    Acc is Acc0 + H^P,
    I1 is I + 1,
    sum(T,I1,P,Acc,Sum).
```

4) Given the graph below with costs in its edges, write a Prolog program that finds all paths with cost less than or equal to a given limit. For example, in the graph below, if the limit is 6, there are two possible paths: `[a,c,d]` e `[a,c,b,d]`.




```
s(a,b,7).
s(a,c,2).
s(a,e,2).
s(b,c,1).
s(b,d,3).
s(c,d,3).
s(c,e,4).
s(c,f,5).
s(d,f,5).

dfs(N,N,[N],C,C) :- !.
dfs(A,Z,[A|P],C1,C2) :-
    (s(A,B,C); s(B,A,C)),
    C3 is C1 + C,
    C3 =< 6,
    dfs(B,Z,P,C3,C2).
```

5) Show an example of CLP (Constraint Logic Programming) code that gives an error when executed in Prolog.

```
x-3 = y+5.
```

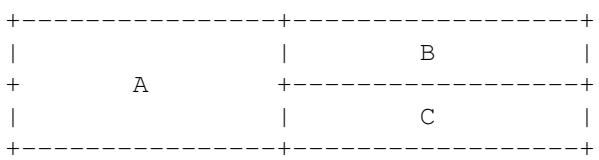
This piece of code fails in Prolog, while it is accepted in CLP.

```
light_meal(A,M,D) :-
    I > 0, J > 0, K > 0,
```

```
   I + J + K =< 10,
   starter(A,I),
   main_ourse(M,J),
   dessert(D,K).
```

This second piece of code gives an error in Proloh, because variables I, J and K are not bound when tests $I > 0, J > 0, K > 0$ are executed. In CLP, this does not give an error.

6) The problem of map coloring can be represented as a Constraint Satisfaction Problem (CSP), where each region in the map is painted with one color and adjacent regions can not have the same color. For example, in the map below, 3 distinct colors are necessary to paint the three regions A, B e C. Write a program using clp(fd) that can solve this problem in a generic way, given the map and the number of colors.



Solution specific to this problem:

```
mapColouring(Vars) :-
   Vars = [A,B,C],
   Vars ins 1..3, % assuming 3 colors
   alldifferent(Vars),
   label(Vars).
```

Generic solution:

```
mapColouring(Vars,ConstraintList,NumberOfColors) :-
   Vars ins 1..NumberOfColors,
   exec(ConstraintList),
   label(Vars).

exec([]).
exec([C|Cs]) :-
   call(C),
   exec(Cs).

% Possible query:
?- Vars=[A,B,C], mapColouring(Vars,[alldifferent(Vars)],3).

% Other example:
?- Vars=[A,B,C,D,E], mapColouring(Vars,[\+A#=C,\+A#=D,\+B#=E,...],3).
```

7) What is the difference between the `get` and `unify` instructions in the Prolog abstract machine?
get instructions are used to fetch arguments pointed by the Ai registers and unify their content with the actual parameter defined by the current instruction. Unify instructions are used to unify arguments pointed by the S register (arguments of level greater than 1 - the ones that appear in arguments of predicates defined by functors - Prolog nested structures).