Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI3661 - Taller de Lenguajes de Programación I
Septiembre-Diciembre 2016

Quiz Prolog

(10 puntos)

Nombre:

Carnet:

Atención: en todas sus respuestas debe utilizar *exclusivamente* predicados disponibles en el estándar ISO provisto por GNU Prolog. Si Ud. no sabe si un predicado está o no en el conjunto estándar ISO provisto por GNU Prolog, puede preguntarlo *públicamente* para recibir un **si** o **no** como respuesta y **nada** más.

1. **(4 puntos)** Escriba el predicado `min(Val,List)` que triunfa *exactamente una vez* si siendo `List` una lista de números, `Val` es el elemento con valor mínimo. Puede suponer que solamente se aplicará con listas de números.

```
?- min(V,[4,2,1,3]).
V = 1
yes
?- min(V,[]).
no
?- min(3,[1,2,4,0]).
no
?- min(2,[4,8,2,7]).
yes
```

La restricción «triunfa exactamente una vez» es natural, pues sólo hay un valor mínimo dentro de una lista, aunque ocurra más de una vez. Para asegurarla, es necesario cortar el *backtracking* tanto al terminar, como en cualquier paso intermedio en el cual se escoge el «mínimo hasta ahora». Por otro lado, el predicado debe fallar sobre listas vacías, de modo que se estructura la solución con dos predicados: `min/2` que obligue a la lista a tener al menos un elemento, que será tomado como el «mínimo hasta ahora», y `min/3` auxiliar que recorre la lista manteniendo el invariante tal que el «mínimo hasta ahora» sea el mínimo valor de la lista.

```
min(Min,[Base|Rest]) :- min(Base,Rest,Min).
min(Base,[],Base) :- !.
min(SoFar,[This|Rest],Min) :-
  SoFar =< This, !,
  min(SoFar,Rest,Min).
min(SoFar,[This|Rest],Min) :-
  SoFar >  This, !,
  min(This,Rest,Min).
```

2. **(6 puntos)** Diremos que una *secuencia* es una lista Prolog, posiblemente vacía, sin listas anidadas. Escriba el predicado `subseq(Sub,Seq)` que triunfe si `Sub` es una subsecuencia de `Seq`. Diremos que `Sub` es una subsecuencia de `Seq` si todos los elementos de `Sub` aparecen en el mismo orden, de izquierda a derecha, uno a continuación del otro, en la secuencia `Seq`. Por ejemplo, `[1,2]` es subsecuencia de `[3,1,2,4]` pero no es subsecuencia de `[1,3,2]`; así mismo, note que `[1,1]` es subsecuencia de `[1,1,1]` dos veces; sin embargo, la secuencia vacía `[]` es subsecuencia de cualquier otra solamente una vez. El predicado debe poder emplearse con *backtracking* para producir:

```
?- subseq([1,1],[1,1,1]).
true? ;
true? ;
no
?- subseq(Sub,[1,2,3]).
Sub = [] ? ;
Sub = [1] ? ;
Sub = [1,2] ? ;
Sub = [2] ? ;
Sub = [1,2,3] ? ;
Sub = [2,3] ? ;
Sub = [3] ? ;
no
?- subseq([1,2],Seq).
Seq = [1,2] ? ;
Seq = [1,2,_] ? ;
Seq = [_,1,2] ? ;
Seq = [1,2,_,_] ? ;
Seq = [_,1,2,_] ? ;
...
```

Si una secuencia es subsecuencia de otra, quiere decir que existe un prefijo izquierdo y un sufijo derecho, tales que al combinar los tres en orden natural (prefijo, secuencia y sufijo), se construye la lista que le contiene. Esto sugiere emplear el predicado `append/3` explotando *backtracking* para encontrar todas las soluciones.

```
subseq([],_).
subseq(Sub,Seq) :-
  length(Seq,Max),
  append(Left,_,Seq),
  append(_,Sub,Left),
  length(Sub,N),
  N > 0,
  N =< Max.
```