Examen de *Programación declarativa* del 21 de Enero de 2008

José A. Alonso Jiménez

Grupo de Lógica Computacional
Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, 28 de enero de 2008

Ejercicio 1 (1 punto) *Escribir lo que devuelve Haskell al evaluar las siguientes expresiones:*

- `:type map`
- `filter (>3) [1,3,5,4,2,6,1]`
- `filter (3>) [1..7]`
- `zip [1,2,3] "abc"`

Solución:

```
Hugs> :type map
map :: (a -> b) -> [a] -> [b]
Hugs> filter (>3) [1,3,5,4,2,6,1]
[5,4,6]
Hugs> filter (3>) [1..7]
[1,2]
Main> zip [1,2,3] "abc"
[(1,'a'),(2,'b'),(3,'c')]
```

Ejercicio 2 (2 puntos) *Definir en Haskell la función*

```
subconjunto :: Eq a => [a] -> [a] -> Bool
```

tal que `(subconjunto xs ys)` *se verifica si todos los elementos de* `xs` *pertenecen a* `ys`*. Por ejemplo,*

```
subconjunto [3,2] [1..5]   ⇝ True
subconjunto [3,2,6] [1..5] ⇝ False
```

Escribir una definición por recursión y otra por comprensión.

Solución: La definición recursiva es

```
subconjunto [] _      = True
subconjunto (x:xs) ys = elem x ys && subconjunto xs ys
```

La definición por comprensión es

```
subconjunto xs ys = and [elem x ys | x <- xs]
```

Ejercicio 3 (2 puntos) *Se define el tipo de datos* `Arbol` *como*

```
data Arbol a = Hoja | Nodo a (Arbol a) (Arbol a)
```

Definir en Haskell la función

```
aplana :: Arbol a -> [a]
```

tal que `(aplana x)` *es la lista obtenida aplanando el árbol* `x`*. Por ejemplo,*

```
aplana (Nodo 6 (Nodo 3 Hoja Hoja) (Nodo 7 Hoja Hoja)) ⇝ [6,3,7]
```

Solución:

```
aplana Hoja             = []
aplana (Nodo x izq der) = x:(aplana izq ++ aplana der)
```

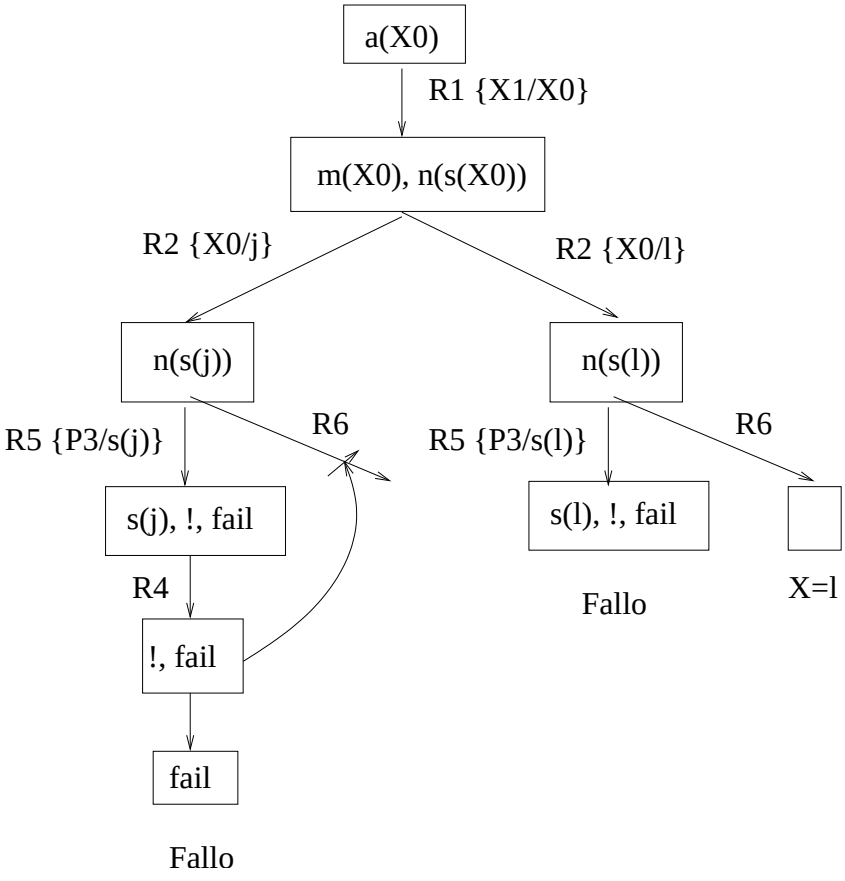
Ejercicio 4 (1 punto) *Se considera el siguiente programa lógico*

```
a(X) :- m(X), n(s(X)).   % R1
m(j).                    % R2
m(l).                    % R3
s(j).                    % R4
n(P) :- P, !, fail.      % R5
n(P).                    % R6
```

Construir el árbol de resolución correspondiente al programa anterior con la pregunta

```
?- a(X).
```

Solución:



Ejercicio 5 (2 puntos) *Definir en Prolog la relación* `cuenta(+X,+L,-N)` *que se verifique si* `N` *es el número de ocurrencias de* `X` *en la lista* `L`*. Por ejemplo,*

```
?- cuenta(a,[a,b,a,a],N).
N=3
```

Solución:

```
cuenta(_,[],0).
cuenta(A,[A|L],N) :-
   !,
   cuenta(A,L,M),
   N is M+1.
cuenta(A,[_|L],N) :-
   cuenta(A,L,N).
```

Ejercicio 6 (2 puntos) *Definir la relación* `lista_mayor(+L1,-L2)` *que se verifique si* `L2` *es una lista de la lista de listas* `L1` *de máxima longitud. Por ejemplo,*

```
?- lista_mayor([[a,b,c],[d,e],[1,2,3]],L).
L = [a, b, c] ;
L = [1, 2, 3] ;
No
```

Solución: Una solución es

```
lista_mayor_1(L1,L2) :-
   member(L2,L1),
   length(L2,N2),
   \+ (member(L,L1), length(L,N), N2 < N).
```

Otra solución más eficiente es

```
lista_mayor_4(L1,L2) :-
   máxima_longitud(L1,N),
   length(L2,N),
   member(L2,L1).
```

donde `máxima_longitud(+L,-N)` se verifica si `N` es la máxima longitud de la lista `L`. Por ejemplo,

```
?- máxima_longitud([[a],[a,b,c],[b,a]],N).
N = 3
```

```
máxima_longitud([L],N) :-
   length(L,N).
máxima_longitud([L|R],N) :-
   length(L,N1),
   máxima_longitud(R,N2),
   N is max(N1,N2).
```