UNIVERSIDAD DE MÁLAGA
E.T.S.I. INFORMÁTICA

Programación Declarativa

(3º de Ingeniería Técnica en Informática)
Septiembre de 2006

Nombre:
Especialidad: **Grupo:**

Prolog

Ejercicio 1.1 *(1 punto)*

Realizad el árbol de búsqueda con el objetivo `p([Z,Y])` y el siguiente programa *Prolog*.

```
p([b,X|Z]):-
    q(X,f(X,g(Y))).
q(g(b),f(g(Y),g([b,Y]))).
q(g(a),f(g(Y),g([b,Y]))):-
    p([b,T|Z]).
```

Ejercicio 1.2 *(0,25 puntos)*

¿Cuál/es será/n la/s respuesta/s de *Prolog* para el caso anterior?.

Ejercicio 2.1 *(1.5 puntos)*

Definir el predicado `purga(+Xs,-Ys)` que elimina los elementos repetidos de una lista. Por ejemplo

```
?- purga([2,3,3,4,2,3,5,2],Ys).
Ys = [4, 3, 5, 2] ;
No
```

Ejercicio 2.2 *(1 punto)*

Definir el predicado `diferencia(+Xs,+Ys,-Zs)` que elimina de `Xs` los elementos que se encuentran en `Ys`. Por ejemplo

```
?- diferencia([3,4,3,2,5],[3,5],Ys).
Ys = [4, 2] ;
No
```

Ejercicio 3.1 *(1.75 puntos)*

Una regla de Golumb es una regla que tiene marcas solo en ciertas distancias dadas. Por ejemplo, la de abajo es una regla con marcas en 2, 3 y 7 cm.

Estas reglas se pueden representar en Prolog mediante listas ordenadas; p.ej. [2,3,7]

Define el predicado `medir(+R,+D,-S)` que dadas una regla de Golumb `R` y una distancia `D`, devuelva en S todas las formas en que es posible medir D sumando medidas directas (a partir del origen) hechas con R. Por ejemplo:

```
?- medir([2,3,7],9,S).
S = [2, 2, 2, 3] ;
```

```
S = [2, 2, 3, 2] ;
S = [2, 3, 2, 2] ;
S = [2, 7] ;
S = [3, 2, 2, 2] ;
S = [3, 3, 3] ;
S = [7, 2] ;
No
```

Ejercicio 3.2 *(2 puntos)*

Define el predicado `distancias(+R,-Ds)` que dada una regla de Golumb `R` devuelve una lista `Ds` con todas las distancias distintas entre cada dos de sus marcas, sin repeticiones y sin incluir las marcas originales de la regla.

```
?- distancia([2,3,7],Xs).
Xs = [1,4,5];
No
```

Ejercicio 4 *(2.5 puntos)*

Definir el predicado `medirEx(+R,+D,-Solucion)` que genere todas las formas posible de alcanzar la distancia `D` sumando o restando medidas directas hechas con la regla `R`, con las siguientes restricciones: las mediciones acumuladas durante el proceso deben ser positivas y no mayores que `D` y además no se deben repetir medidas acumuladas. Por ejemplo:

```
?- medirEx([3,7],9,S).
S = [3, 6, 9] ;
S = [7, 4, 1, 8, 5, 2, 9] ;
```

La primera solución corresponde a las distancias que se van alcanzando sumando 3 tres veces seguidas (uso $3+3+3$). La segunda corresponde a las distancias que se van alcanzando sumando 7, restando 3, restando 3, sumando 7, etc. (uso $7-3-3+7-3-3+7$). Observa que en ninguna de las listas aparecen elementos repetidos ni números negativos o 0 ni mayores que 9.

UNIVERSIDAD DE MÁLAGA
E.T.S.I. INFORMÁTICA

Programación Declarativa

(3º de Ingeniería Técnica en Informática)
Septiembre de 2006

Nombre:
Especialidad: **Grupo:**

Haskell

Ejercicio 1

(*a*) (1 *pt.*) ¿ Cuál es el tipo polimórfico de la función `f` ?:

```
f g x y = foldl (g x) 0 (x:y)


f::_____________________________________________
```

(*b*) (1 *pt.*) ¿Cuál es el resultado de evaluar la expresión `foldr (\y x -> y (y x)) 1 [\x->x*x, \x->x-1, (+2)]`?

SOLUCION:

(*c*) (1 *pt.*) Sea `e` una expresión Haskell. ¿Cuál de las siguientes situaciones es posible?:

a. Al evaluar `e` por evaluación perezosa se obtiene como resultado 3, y de modo impaciente el cómputo no termina.
b. Al evaluar `e` de modo impaciente se obtiene como resultado 3, y por evaluación perezosa 2.
c. Al evaluar `e` de modo impaciente se obtiene como resultado 3, y por evaluación perezosa el cómputo no termina.

SOLUCION:

(*d*) (1 *pt.*) La expresión `map (zip [1..4] . (:[])) [11,12]`

a. Tiene como valor [[(1,11)],[(1,12)]]
b. Tiene como valor [(1,11),(2,11),(3,11),(4,11),(1,12),(2,12),(3,12),(4,12)]
c. No tiene valor, ya que está mal tipada

SOLUCION:

(*e*) (1 *pt.*) Sean las siguientes definiciones de la funciones `curry` y `uncurry`:

```
curry f x y  = f (x,y)
uncurry f (x,y)  = f x y
```

¿Cuál es el tipo de la expresión `curry . uncurry`?

a. `curry . uncurry :: :: ((a,b) -> c) -> (a,b) -> c`
b. `curry . uncurry :: (a -> b -> c) -> a -> b -> c`
c. La expresión `curry . uncurry` está mal tipada

SOLUCION:

Ejercicio 2

Las planchas rectangulares de madera y de vidrio se suelen cortar con máquinas que realizan cortes perpendiculares, de lado a lado, de la plancha dividiéndola en dos partes. Las partes resultantes se vuelven a cortar de la misma forma y así sucesivamente hasta obtener una serie de piezas rectangulares. El dispositivo de corte de la máquina sólo admite dos orientaciones: horizontal y vertical, por lo que sólo se producen dos tipos de cortes.

Las secuencias de cortes que conducen a una determinada serie de piezas se pueden recoger en una estructura, que llamaremos *patrón de corte*, parecida a la de un árbol binario en el que los nodos figuran los cortes junto con la proporción del lado de la pieza que se corta (desde la izquierda para los cortes verticales y desde arriba para los cortes horizontales), y cada rama la secuencia de cortes que conduce a una pieza (tal como se ve en la figura).

Sean los siguientes tipos para representar rectángulos (dadas sus dimensiones vertical y horizontal) y patrones de corte:

```
data Rectángulo = R Float Float deriving Show
data Formato = V Float | H Float deriving Show
data PatrónCorte = C Formato PatrónCorte PatrónCorte
                 | P deriving Show
```

A partir de estos datos se deberá hacer lo siguiente:

a. *(1.5 puntos)* Definir una función `cortar :: Rectángulo -> PatrónCorte -> [Rectángulo]` que produzca la lista de rectángulos resultante de la aplicación de un patrón de corte (segundo argumento) a un rectángulo (primer argumento).

b. *(1 puntos)* Definir una función `numCortes :: PatrónCorte -> (Int,Int)` que cuente el número de cortes verticales (primera componente del resultado) y de cortes horizontales (segunda componente) que aparecen en un patrón.

c. *(1.5 puntos)* Definir una función de plegado adecuada al tipo `PatrónCorte` y expresar la función `numCortes` como una aplicación de dicha función de plegado. Indica también, el tipo polimórfico de la función de plegado.

d. *(1 punto)* Sea la siguiente función de orden superior:

```
plegado2 h g f = fun
 where
   fun r P = h r
   fun r (C t p' p'') = fun r' p' `g` fun r'' p''
    where
     (r',r'') = f r t
```

Expresa la función `cortar` como una aplicación de `plegado2`