Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-3661 - Laboratorio de Lenguajes de Programación
Enero-Marzo 2015

Programación Lógica

Estaciones

Cansados de vivir en Caracas y pensando en nuestro bolsillo, deseamos mudarnos a una ciudad pequeña y tranquila que no esté demasiado alejada de las grandes ciudades en las que podemos conseguir trabajo, ropa y alimentos. Nuestro único medio de transporte será el tren y hay solo una estación por ciudad, por lo que sería ideal movernos a una ciudad con una estación de tren que nos permita acceder rápidamente a las grandes ciudades.

Para esto definiremos un predicado **estacion/1** que unifica su argumento con la mejor estación (y por lo tanto con la mejor ciudad, ya que hay una estación por ciudad). El predicado solo tendrá éxito cuando encuentre la mejor estación, y deberá fallar cuando no pueda hallarla.

Una **ciudad pequeña**, es aquella que tiene a lo más dos conexiones directas con otras ciudades. Por lo tanto, una **ciudad grande** es aquella con más de dos conexiones directas con otras ciudades.

Una **buena ciudad** es una ciudad pequeña que está igualmente cerca a las dos ciudades grandes más cercanas, cada una en una linea de salida distinta. La **mejor ciudad** es aquella entre las todas la buenas ciudades que está más cerca a sus respectivas ciudades grandes.

La red de trenes será dada como una serie de hechos **carril/2**. Todos los carriles van en ambas direcciones y miden la misma distancia. La distancia entre dos ciudades es el largo del camino más corto entre ambas.

Ejemplos:

```
carril(brussel,mechelen).
carril(brussel,antwerpen).
carril(brussel,gent).
carril(antwerpen,mechelen).
carril(antwerpen,gent).
carril(gent,brugge).
```

```
?- estacion(X).
X = mechelen
```

```
carril(brussel,charleroi).
carril(brussel,haacht).
carril(haacht,mechelen).
carril(mechelen,berchem).
carril(berchem,antwerpen).
carril(brussel,boom).
carril(boom,antwerpen).
carril(antwerpen,turnhout).
```

```
?- estacion(X).
X = boom
```

Estrellas mágicas

Una estrella mágica de *N* puntas es una estrella polígono en la cual hay un número en cada vértice e intersección y la suma de estos en cada línea es igual a *4N + 2*. Una estrella mágica contiene números en el rango *1..2N* sin repeticiones.

Un ejemplo de una estrella mágica de 8 puntas es:

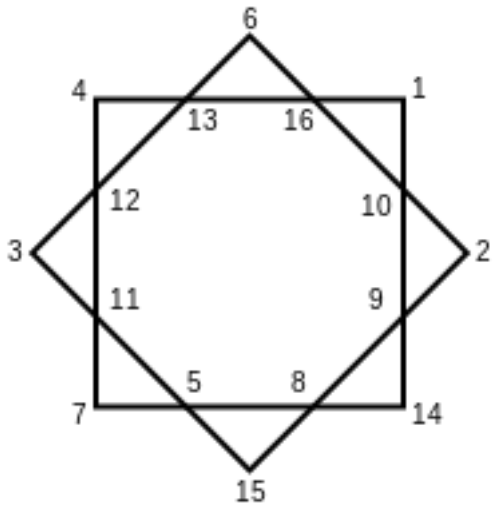

Figure 1: Estrella mágica de 8 puntas

En la cual podemos verificar la propiedad que define a una estrella mágica:

```
4 * 8 + 2 = 34

4 + 13 + 16 + 1  = 34
3 + 12 + 13 + 6  = 34
6 + 16 + 10 + 2  = 34
4 + 12 + 11 + 7  = 34
1 + 10 + 9  + 14 = 34
7 + 5  + 8  + 14 = 34
3 + 11 + 5  + 15 = 34
2 + 9  + 8  + 15 = 34
```

Escriba el predicado **estrella/1** que espera una lista y triunfa si la lista corresponde a una estrella mágica válida de 8 puntas. Los valores en la lista *[A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P]* deben ser organizados de la siguiente manera:

```
      A
    B C D E
     F   G
  H         I
     J   K
    L M N O
      P
```

Aplicando el predicado al ejemplo, obtendríamos

```
?-estrella([6, 4, 13, 16, 1, 12, 10, 3, 2, 11, 9, 7, 5, 8, 14, 15]).
yes
```

Su predicado debe ser puro, al ser consultado sin unificar debe poder enumerar todas las estrellas mágicas de tamaño 8. Por ejemplo:

```
?-estrella(Lista).
Lista = [6, 4, 13, 16, 1, 12, 10, 3, 2, 11, 9, 7, 5, 8, 14, 15] ? ;
...
```

Laberintos Rectangulares

Escriba tres predicados en prolog que le permitan leer, resolver y pintar laberintos de tamaño MxN.

El primer predicado será **leer/3**. Éste pedirá un nombre de archivo del que leerá un laberinto y unificará su primer argumento con una lista de tuplas que representará los **puntos abiertos**. El segundo y tercero unificarán con el tamaño de filas y columnas del laberinto, respectivamente.

La primera línea del archivo tendrá el número de columnas **N**, mientras que el resto de las **M** líneas representarán el laberinto. Un **punto cerrado** será representado con un '#', mientras que uno abierto con ''. Puede representar los puntos con un predicado **punto/2** con la coordenada **X** y **Y** del punto en el archivo. También puede suponer que el archivo no tiene errores de entrada.

laberinto.txt:

```
10
  ########
# ### ####
# ###    #
#     ####
# ##     #
# ########
```

```
#        #
## ##### #
## ##### #
##  ####
```

Al ejecutarlo, obtendríamos:

```
?- leer(L, M, N).

"Por favor, introduzca el nombre del archivo".

"laberinto.txt".

L = [punto(0,0), punto(1,0), punto(1,1),
     punto(1,2), punto(1,3), punto(1,4),
     punto(1,5), punto(1,6), punto(2,6),
     punto(3,6), punto(4,6), punto(5,6),
     punto(6,6), punto(7,6), punto(8,6),
     punto(8,7), punto(8,8), punto(9,8),
     punto(9,9),
     **Resto de espacios abiertos***
    ]
M = 10
N = 10 ? .
```

Posteriormente deberá escribir el predicado **resolver/4** en el cual su primer argumento es una lista de puntos que representa los puntos abiertos de un laberinto, el segundo y tercero **M** y **N** y unificará el cuarto con una lista de puntos que representará el camino de salida del laberinto. El predicado debe ser capaz de dar más de una solución en caso que el laberinto lo permita. En caso de no encontrar salida, deberá fallar.

Puede suponer que la entrada es el punto **(0,0)** y la salida en alguna fila de la columna **N-1**.

Por último el predicado **escribir/4** que tomará una lista de puntos que representa una solución, los puntos que representan el laberinto original, **M** y **N** y dibuja por salida estándar la solución en el laberinto, usando el mismo formato de entrada pero usando el caracter '.' para mostrar el camino.

Ejemplo:

```
?- escribir([punto(0,0), punto(1,0), punto(1,1),
             punto(1,2), punto(1,3), punto(1,4),
             punto(1,5), punto(1,6), punto(2,6),
             punto(3,6), punto(4,6), punto(5,6),
```

```
            punto(6,6), punto(7,6), punto(8,6),
            punto(8,7), punto(8,8), punto(9,8),
            punto(9,9)],
            [**Todos los espacios abiertos**],
            10, 10).
```

```
..########
#.### ####
#.###    #
#.    ####
#.##     #
#.########
#........#
## ######.#
## ######.#
##  ####..
```

Unificación

Por último, implante el Algoritmo de Unificación de Prolog2 con "Occurs Check"1 para garantizar terminación.

Escriba el predicado **unifica/2** que tendrá éxito en el caso que sus argumentos de entrada unifiquen y falle en caso contrario.

Detalles de Entrega

- Debe entregar un archivo **tar.gz** con el nombre **tp-carne1-carne2.tar.gz** dónde **carne1** y **carne2** corresponden al carné de cada uno de los estudiantes que pertenecen al grupo. El comprimido deberá contener los archivos **laberinto.pro**, **estaciones.pro**, **estrellas.pro** y **unificacion.pro**.
- La tarea deberá ser entregada a todos los representantes del curso a sus direcciones de correo oficiales. El asunto del correo deberá comenzar con el texto [ci3661].
- **Fecha de entrega**: Viernes, 20 de Febrero de 2015.
- **Valor de la evaluación**: 25%.