Trabajo Práctico 2
Programación Lógica

Paradigmas de Lenguajes de Programación — 1er cuat. 2015

Fecha de entrega: 9 de junio

1. Introducción

Se quiere trabajar con un robot que debe moverse en un espacio modelado como una grilla o tablero, el que está conformado por una cantidad finita de celdas. Cada celda puede estar libre, en cuyo caso el robot puede transitarla, o puede estar ocupada y nuestro robot deberá eludirla. El robot puede moverse en forma ortogonal y de a una celda a la vez.

Se espera que se exprese en Prolog de forma elemental, declarativa y clara algunas variantes de este problema según la descripción de los ejercicios del presente trabajo.

El primer paso será armar algunos tableros para luego buscar posibles caminos dentro del mismo dadas las coordenadas iniciales y finales.

Representación utilizada

Un tablero consta de $F \times C$ celdas con F filas y C columnas. El mismo queda representado como una matriz almacenada por filas, o sea, una lista de listas donde las internas son las filas de la matriz.

Cada elemento de la matriz contendrá un átomo `ocupada`, para denotar una celda ocupada, o será una variable sin unificar, para denotar una celda libre.

Además se usarán términos de la forma `pos(F,C)` para denotar una posición determinada, con los índices comenzando en 0. Así, una lista de estos elementos denotará un camino.

2. Guía de trabajo

Tablero

1. `tablero(+Filas,+Columnas,-Tablero)` instancia una estructura de tablero en blanco de `Filas` × `Columnas`, con todas las celdas libres.

```
?- tablero(3, 2, T).
T = [[_G289, _G292], [_G298, _G301], [_G307, _G310]] ;
false.
```

2. `ocupar(+Pos,?Tablero)` será verdadero cuando la posición indicada esté ocupada.

```
?- tablero(3, 2, T), ocupar(pos(1, 0), T).
T = [[_G338, _G341], [ocupada, _G350], [_G356, _G359]] ;
false.
```

A fin de contar con tableros de ejemplo, se sugiere tener un predicado `tablero(+nombre,-Tablero)` que instancie una estructura de tablero determinada. Por ejemplo:

```
tablero(ej5x5, T) :-
  tablero(5, 5, T),
  ocupar(pos(1, 1), T),
  ocupar(pos(1, 2), T).

tablero(libre20, T) :-
  tablero(20, 20, T).

?- tablero(ej5x5,T).
T = [[_G287, _G290, _G293, _G296, _G299], [_G305, ocupada, ocupada, _G314, _G317],
[_G323, _G326, _G329, _G332, _G335], [_G341, _G344, _G347, _G350, _G353], [_G359,
_G362, _G365, _G368|...]]
```

El tablero `ej5x5` equivale a la Figura 1.

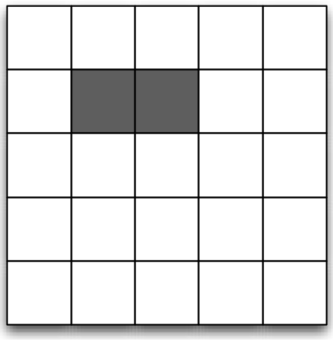

Figura 1: Tablero `ej5x5`

Vecinos

3. `vecino(+Pos, +Tablero, -PosVecino)` será verdadero cuando `PosVecino` sea una celda contigua a `Pos`. Las celdas contiguas pueden ser a lo sumo cuatro, dado que el robot se moverá en forma ortogonal.

```
?- tablero(ej5x5, T), vecino(pos(0,0), T, V).
V = pos(1, 0) ;
V = pos(0, 1) ;
false.

?- tablero(ej5x5, T), vecino(pos(2,2), T, V).
V = pos(1, 2) ;
```

```
V = pos(3, 2) ;
V = pos(2, 1) ;
V = pos(2, 3) ;
false.
```

4. `vecinoLibre(+Pos, +Tablero, -PosVecino)` ídem `vecino/3`, pero además `PosVecino` debe ser una celda transitable (no ocupada) en `Tablero`.

```
?- tablero(ej5x5, T), vecinoLibre(pos(0,0), T, V).
V = pos(1, 0) ;
V = pos(0, 1) ;
false.

?- tablero(ej5x5, T), vecinoLibre(pos(2,2), T, V).
V = pos(3, 2) ;
V = pos(2, 1) ;
V = pos(2, 3) ;
false.
```

Definición de caminos

5. `camino(+Inicio, +Fin, +Tablero, -Camino)` será verdadero cuando `Camino` sea una lista `[pos(F1,C1), pos(F2,C2),..., pos(Fn,Cn)]` que denote un camino desde `Inicio` hasta `Fin` pasando sólo por celdas transitables. Además, se espera que `Camino` no contenga ciclos.
   Notar que la cantidad de caminos es finita y por ende se tiene que poder recorrer todas las alternativas, eventualmente.
   Consejo: Utilizar una lista auxiliar con las posiciones visitadas.

   La Figura 2 ilustra la primera solución que podría ofrecer
   `?- tablero(ej5x5, T), camino(pos(0,0), pos(2,3), T, C).`

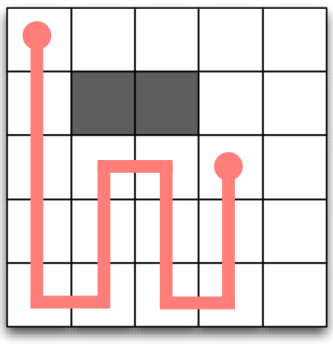

Figura 2: Ejemplo camino/4

6. `cantidadDeCaminos(+Inicio, +Fin, +Tablero, ?N)` que indique la cantidad de caminos posibles sin ciclos entre `Inicio` y `Fin`. Ejemplo.

```
?- tablero(ej5x5, T), cantidadDeCaminos(pos(0,0), pos(2,3), T, N).
N = 287 ;
```

7. `camino2(+Inicio, +Fin, +Tablero, -Camino)` ídem `camino/4`, pero se espera una aplicar una heurística que mejore las soluciones iniciales. No se espera que la primera solución sea necesariamente la mejor.
   Una solución es mejor mientras menos pasos requiera para llegar a destino (distancia Manhattan).

   La Figura 3 ilustra la primera solución que podría ofrecer
   `?- tablero(ej5x5, T), camino2(pos(0,0), pos(2,3), T, C).` en rojo y en azul otro posible camino que se ofrecerá como solución, eventualmente.

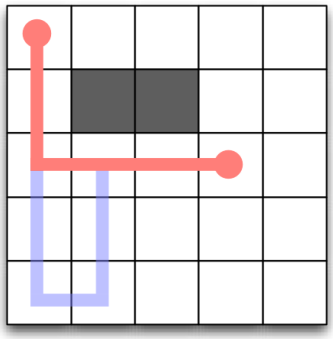

Figura 3: Ejemplo camino2/4

   Como dato adicional, el total de caminos posibles es 287 según `camino2/4`. No se filtran soluciones con respecto a `camino/4`.

8. `camino3(+Inicio, +Fin, +Tablero, -Camino)` ídem `camino2/4`, pero se espera que se reduzca drásticamente el espacio de búsqueda.

   En el proceso de generar los potenciales caminos se pueden ir sacando algunas conclusiones. Por ejemplo, si se está en la celda `(3,4)` y ya se dieron 6 pasos desde `Inicio`, entonces no tiene sentido seguir evaluando cualquier camino en otra rama de ejecución que implique llegar a la celda `(3,4)` desde `Inicio` en más de 6 pasos.

   Notar que dos ejecuciones de `camino3/4` con los mismos argumentos deben dar los mismos resultados.

   En este ejercicio se permiten el uso de predicados: `dynamic/1`, `asserta/1`, `assertz/1` y `retractall/1`.

   La Figura 4 ilustra la primera solución que podría ofrecer
   `?- tablero(ej5x5, T), camino2(pos(0,0), pos(2,3), T, C).`

A diferencia de `camino/2`, en la Figura 3, el camino azul no es una solución factible ya se conoció uno mejor a él (que llegó a la celda `(2,1)` en 3 pasos).

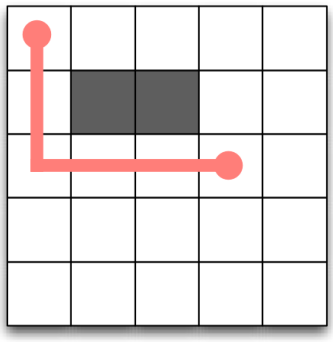

Figura 4: Ejemplo camino3/4

En `camino/3` el total de caminos posibles será menor. A modo de ejemplo, en la implementación de la cátedra se obtuvieron 2 caminos posibles.

Tableros simultáneos

9. `caminoDual(+Inicio, +Fin, +Tablero1, +Tablero2, -Camino)` será verdadero cuando `Camino` sea un camino desde `Inicio` hasta `Fin` pasando al mismo tiempo sólo por celdas transitables de ambos tableros.
   Nota: Es posible una implementación que resuelva en forma inmediata casos en los que trivialmente no existe camino dual posible.

3. Condiciones de aprobación

El principal objetivo de este trabajo es evaluar el correcto uso del lenguaje PROLOG de forma declarativa para resolver el problema planteado. El código debe estar *adecuadamente* comentado (es decir, los comentarios que escriban deben facilitar la lectura de los predicados, y no ser una traducción al castellano del código). También se debe explicitar cuáles de los argumentos de los predicados auxiliares deben estar instanciados usando `+`, `-` y `?`.

La entrega debe incluir casos de prueba en los que se ejecute al menos una vez cada predicado definido.

En el caso de los predicados que utilicen la técnica de *generate and test*, deberán indicarlo en los comentarios.

4. Algunas consideraciones sobre *generate and test*

Al generar y testear se toma un candidato y se verifica si sirve o no como solución. Para eso hay que decidir cuál es el universo de posibles soluciones. Una mala decisión de este universo puede impactar muy fuertemente en la eficiencia.

Por ejemplo, si se quieren calcular los factores primos de un número N, una forma es probar número por número, en este caso el universo de posibles soluciones son los naturales. Otra forma sería tomar como universo de soluciones posibles los números primos entre 2 y N.

En ambos casos el algoritmo de verificación es el mismo: tomar un candidato y ver si divide o no al número en cuestión.

Si el universo de soluciones posibles son todos los naturales, se tardará mucho en encontrar las soluciones y además, al no estar acotado el conjunto de partida, no se terminará nunca. Si en cambio contamos con una forma eficiente de enumerar los números primos y probar con estos, entonces encontraremos las soluciones de manera mucho más rápida.

En muchos otros casos se pueden generar las soluciones de manera directa, sin necesidad de testear posteriormente. Esto es deseable siempre que la generación sea un procedimiento sencillo y relativamente eficiente.

Además de la elección del universo de candidatos es importante tener en cuenta que, por como funciona el algoritmo detrás de PROLOG, **cuanto antes** podemos descartar una solución tanto mejor. Es decir que si al ir construyendo un candidato a solución descubrimos por el camino que no va a servir, podemos evitarnos una o varias ramas de ejecución potencialmente muy largas.

En algunos problemas se puede partir el proceso de generación y testeo en dos o más etapas que se aplican intercaladas. Por ejemplo:

```
esSolucion(X) :- generarParteUno(X1), testearParteUno(X1),
                 generarParteDos(X1,X), testearParteDos(X).
```

Donde `generarParteDos(+X1,-X2)` parte de una solución parcial ya testeada `X1` y la aumenta para conseguir un candidato a solución `X2` que a su vez deberá ser testeado, ya asumiendo que una parte es correcta.

5. Pautas de Entrega

Se debe entregar el código impreso con la implementación de los predicados pedidos. Cada predicado debe contar con un comentario donde se explique su funcionamiento. Cada predicado asociado a los ejercicios debe contar con ejemplos que muestren que exhibe la funcionalidad solicitada. Además, se debe enviar un e-mail conteniendo el código fuente en Prolog a la dirección plp-docentes@dc.uba.ar. Dicho mail debe cumplir con el siguiente formato:

- El título debe ser `[PLP;TP-PL]` seguido inmediatamente del nombre del grupo.
- El código Prolog debe acompañar el e-mail y lo debe hacer en forma de archivo adjunto con nombre `tp2.pl`.

El código debe poder ser ejecutado en SWI-Prolog. No es necesario entregar un informe sobre el trabajo, alcanza con que el código esté adecuadamente comentado. Los objetivos a evaluar en la implementación de los predicados son:

- corrección,
- declaratividad,
- reutilización de predicados previamente definidos
- Uso de unificación, backtracking, generate and test y reversibilidad de los predicados que correspondan.

- **Importante**: salvo donde se indique lo contrario, los predicados no deben instanciar soluciones repetidas. Vale aclarar que no es necesario filtrar las soluciones repetidas si la repetición proviene de las características de la entrada.

Importante: se admitirá un único envío, sin excepción alguna. Por favor planifiquen el trabajo para llegar a tiempo con la entrega.

6. Referencias y sugerencias

Como referencia se recomienda la bibliografía incluída en el sitio de la materia (ver sección *Bibliografía* → *Programación Lógica*).

Se recomienda que, siempre que sea posible, utilicen los predicados ISO y los de SWI-Prolog ya disponibles. Recomendamos especialmente examinar los predicados y metapredicados que figuran en la sección *Cosas útiles* de la página de la materia. Pueden hallar la descripción de los mismos en la ayuda de `SWI-Prolog` (a la que acceden con el predicado `help`). También se puede acceder a la documentación online de SWI-Prolog.