Trabajo Práctico 2
Programación Lógica

Paradigmas de Lenguajes de Programación — 1^o cuat. 2011

Fecha de entrega: 9 de Junio

Este trabajo consiste en implementar en Prolog un programa que resuelva tableros del juego llamado buscaminas. Uno de los objetivos es comparar distintas formas de plantear soluciones del tipo *Generate & test* y disminuir el espacio de búsqueda.

La solución debe realizarse en un solo archivo Prolog. Pueden utilizarse todos los predicados y metapredicados existentes.

El grupo que entregue la implementación más eficiente (en tiempo) para encontrar una solución cualquiera recibirá un premio.

Introducción

En un tablero de buscaminas, las celdas pueden tener dos valores. O bien tienen una mina, o bien un número entre 0 y 8 que indica cuántas minas hay en sus celdas contiguas.

A lo largo de la implementación los tableros estarán representados por listas de filas. Los valores posibles son `mina` y números de `0` a `8`. Un elemento no unificado del tablero, indica el desconocer qué valor le corresponde.

Por ejemplo el tablero

1		
1	*	
	2	2

Se representa como `[[1,_,_],[1,mina,_],[_,2, 2]]`.

Ejercicios

Ejercicio 1

valor$(-X)$ que enumera todos los posibles valores de una celda.

```
?- valor(X).
X = mina ;
X = 0 ;
X = 1 ;
...
X = 8 ;
false.
```

◇

Ejercicio 2

dim(+T,−Cols,−Fils) que es verdadero cuando `Cols` y `Fils` correspoden al tamaño del tablero `T`. Nota: asumir que `T` es un tablero bien formado, o sea que todas sus filas tiene el mismo tamaño.

?– dim([[_,_,_],[_,_,_]], Cols, Fils).
Cols = 3,
Fils = 2.

◇

Ejercicio 3

pos(+T, −C, −F) que enumera todas las posibles posiciones del tablero. Las casillas se enumeran desde 1 en adelante.

?– pos([[_,_,_],[_,_,_]], C, F).
C = 1, F = 1 ;
C = 2, F = 1 ;
C = 3, F = 1 ;
C = 1, F = 2 ;
C = 2, F = 2 ;
C = 3, F = 2.

◇

Ejercicio 4

nonvars(?L, −R) que es verdadero cuando `R` resulta de excluir todos los elementos no instanciados de `L`.

?– nonvars([A,mina, 2, B, 1], R).
R = [mina,2,1].

◇

Ejercicio 5

cant_minas(?L, −N) que es verdadero cuando `N` corresponde a la cantidad de elementos instaciados de `L` que son `mina`. Notar que `L` puede tener elementos no instanciados.

?– cant_minas([_,mina,2,1,_,mina], N).
N = 2.

◇

Ejercicio 6

vecinos(+T, +C, +F, −L) que es verdadero cuando `L` corresponde a los vecinos de la celda de posición `C`, `F` en el tablero `T`.

```
?– vecinos([[A,B,C],[D,E,F],[G,H,I]], 1, 1, L)
L = [B,D,E].

?– vecinos([[A,B,C],[D,E,F],[G,H,I]], 2, 2, L)
L = [A,B,C,D,F,G,H,I].
```

◇

Ejercicio 7

consistente(+T, +C, +F) que es verdadero si la celda en la posición `C`, `F` contiene un valor consistente con sus vecinos. O sea, si no es una `mina`, entonces contiene el número de cuántas minas hay en su vecindad. Se asume que la vecindad y la posición en cuestión están instanciadas. ◇

Ejercicio 8

consistente(+T) análoga a la anterior pero para todo el tablero. ◇

Toma 1

Ejercicio 9

completarA(?T) que enumera todas las posibles soluciones para problema representado por dicho tablero. Se busca hacer una implementación simple que complete las celdas del tablero con alguno de los valores posibles y al final se verifique si es una solución válida. ◇

En el archivo base cuentan con ejemplos de tablero y algunos que les permiten mostrar los tableros y cuánto tiempo se demora en encontrar una solución.

Para el tablero denominado `t1`, se deberían obtener las siguientes soluciones.

```
pre_r(N,T) :– get_time(T1), stamp_date_time(T1, X1, 'UTC'), write(X1), nl, problem(N,T).
post_r(T) :– mostrar(T), get_time(T2), stamp_date_time(T2, X2, 'UTC'), write(X2).
rA(N,T) :– pre_r(N,T), completarA(T), post_r(T).

?– rA(t1,T).
date(2011, 5, 23, 13, 20, 16.9439, 0, UTC, –)
*1
11
date(2011, 5, 23, 13, 20, 16.9459, 0, UTC, –)
T = [[mina, 1], [1, 1]] ;

1*
```

```
11
date(2011, 5, 23, 13, 20, 17.3009, 0, UTC, -)
T = [[1, mina], [1, 1]] ;

11
*1
date(2011, 5, 23, 13, 20, 17.4495, 0, UTC, -)
T = [[1, 1], [mina, 1]] ;
false.
```

En el archivo base cuentan con algunas instancias pequeñas: `t1` a `t4` y varios problemas medianos y grandes: `0` a `14`.

Toma 2

Ejercicio 10

completarB(?T) donde se intentará mejorar el recorrido del espacio de búsqueda. La motivación es hacer una verificación de consistencia por celda lo antes que se pueda. Por ejemplo, una vez determinado un valor para la celda en la posición `C`, `F` se puede verificar la consistencia de la celda `C-1`, `F-1`; adicionalmente si `C`, `F` es un borde derecho o inferior, se puede verificar consistencia en otras celdas. ◇

De forma similar al caso anterior, se puede ver el tiempo de la primera solución de la siguiente manera:

```
rB(N,T) :- pre_r(N,T), completarB(T), post_r(T).

?- rB(t1,T).
date(2011, 5, 23, 13, 33, 19.9863, 0, UTC, -)
*1
11
date(2011, 5, 23, 13, 33, 19.9959, 0, UTC, -)
T = [[mina, 1], [1, 1]] .
```

Toma 3

Ejercicio 11

completarC(?T) donde nuevamente se intentará mejorar el recorrido del espacio de búsqueda. Esta vez la motivación radica en no probar con todos los valores: `mina`, `0`, ..., `8` para cada celda. Se debe usar la información de los vecinos para reducir los posibles valores. Por ejemplo:

- Si un vecino instanciado vale `0` entonces se sabe que la celda central no puede tener una `mina`.
- La cantidad de minas en los vecinos está acotada por los vecinos que ya están instanciados en `mina` y adicionalmente cuántos vecinos aún no están instanciados.

◇

Resultados

Ejercicio 12

Incluir en el informe una tabla con los tiempos insumidos, para cada una de las tres alternativas, en encotrar una solución (la primera).

Es posible que algunos problemas requieran mucho tiempo. Pueden decidir un tiempo de corte. ◇

Toma 4 (opcional)

Ejercicio 13

Se invita a pensar una cuarta variante de `completar` e incluir sus resultados en la tabla del punto anterior. ◇

1. Predicados y metapredicados útiles

- var(?X) y nonvar(?X). Observar que nonvar(X) es equivalente a not(var(X)).

```
?- var(X).
true.

?- var(42).
fail.

?- X = 42, var(X).
fail.
```

- between(+Low, +High, −Value).

```
?- between(100, 102, X).
X = 100 ;
X = 101 ;
X = 102.
```

- member(−Elem, +List).

```
?- member(X, [cero, uno, dos, tres]).
X = cero ;
X = uno ;
X = dos ;
X = tres.
```

- append(?List1, ?List2, ?List3).

```
?- append([1,2,3],[4,5],A).
A = [1,2,3,4,5]

?- append([1,2,3],W,[1,2,3,4,5]).
W = [4,5]
```

- setof(+Template, +Goal, −Set).

```
?- setof(X,between(100, 102, X),L).
L = [100, 101, 102].
```

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 del mensaje 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** (puede adjuntarse un .zip o .tar.gz).

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.
- Reuso de predicados previamente definidos.
- Uso de unificación, backtracking y reversibilidad de los predicados que correspondan.
- 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.