Programación Lógica, Programación en ISO-Prolog

Raúl Munoz Dávila, C20M063

Se dispone de un tablero NxN donde cada casilla está etiquetada con un operador aritmético binario y un operando entero. Comenzando desde una casilla elegida de forma arbitraria, y con el valor inicial 0 (que pasa a ser el valor actual), se recorre el tablero visitando cada una de las casillas exactamente una vez. Por cada casilla visitada se debe realizar la operación indicada utilizando el valor actual como operando izquierdo, y el valor entero que aparece en la casilla como operando derecho. El resultado de esta operación se utilizará como operando izquierdo para la operación asociada a la siguiente casilla visitada, y así sucesivamente. El valor final obtenido (es decir, el que se obtiene tras completar el recorrido) depende evidentemente del recorrido realizado. Tableros: El tablero se representa mediante una lista de celdas definidas mediante estructuras cell/2 con cabecera cell(Pos,Op) . El argumento Pos debe tomar valores pos(Row,Col) donde las variables Row y Col identifican la fila y la columna del tablero en la que está ubicada la celda en cuestión. Estas variables únicamente pueden tomar valores entre 1 y N. Por otro lado, la variable Op representa a la operación matemática asociada a la celda. Estas operaciones se explicitan mediante la estructura op/2 con cabecera op(Op,Val) , donde Op es un operador aritmético escogido de entre los pertenecientes al conjunto predefinido, y Val es un número entero.

```
cell(pos(X,Y),op(Op,Value)) :-
    pos(X,Y),
    op(Op,Value).
```

```
pos(X,Y) :-
    integer(X),
    integer(Y).
```

```
op(Op,Value) :-
    member(Op,[*,-,+,//]),
    integer(Value).
```

```
board1([cell(pos (1 ,1),op(*,-3)),
        cell(pos (1 ,2),op(-,1)),
        cell(pos (1 ,3),op(-,4)),
        cell(pos (1 ,4),op(- ,555)),
        cell(pos (2 ,1),op(-,3)),
        cell(pos (2 ,2),op(+ ,2000)),
        cell(pos (2 ,3),op(* ,133)),
        cell(pos (2 ,4),op(- ,444)),
        cell(pos (3 ,1),op(*,0)),
        cell(pos (3 ,2),op(* ,155)),
```

```
        cell(pos (3 ,3),op(// ,2)),
        cell(pos (3 ,4),op(+ ,20)),
        cell(pos (4 ,1),op(-,2)),
        cell(pos (4 ,2),op(- ,1000)),
        cell(pos (4 ,3),op(-,9)),
        cell(pos (4 ,4),op(*,4))]).
```

Direcciones: las direcciones de tránsito de una casilla a otra se identifican por las siguientes constantes: n (norte), s (sur), e (este), o (oeste), no (noroeste), ne (noreste), so (suroeste) y se (sureste).

```
dir(N,X) :-
    member(N,[n,s,e,o,no,ne,se,so]),
    integer(X),
    X>0.
```

Direcciones Permitidas: Para realizar el tránsito de una casilla a la siguiente es necesario escoger una dirección de entre las disponibles en una lista denominada DireccionesPermitidas . Esta lista contiene una serie de estructuras dir/2 con cabecera dir(Dir,Num) , donde Dir es una de las direcciones anteriores y Num es el número máximo de veces que puede utilizarse dicha dirección durante el cálculo de un recorrido. Por ejemplo, con la lista: DireccionesPermitidas = [dir(n,3), dir(s,4), dir(o,2), dir(se,10)] solamente estaría permitido transitar desde la casilla actual a la casilla ubicada al norte, sur, oeste o sureste de esta siempre y cuando la casilla exista (es decir, que se encuentre dentro de los límites del tablero). Asimismo, durante todo el recorrido, como máximo se puede transitar a las direcciones n , s , o y se en 3, 4, 2 y 10 ocasiones respectivamente. Cabe también aclarar que podría suceder que no sea posible completar ningún recorrido con ciertas listas de direcciones permitidas independientemente de cual sea la casilla de comienzo.

Apartado 1 (6 puntos)

El objetivo de este primer apartado es implementar un predicado que genere recorridos en tableros. Se pide resolver el problema de forma gradual. Primero se deben implementar los siguientes predicados:

- (0.5 puntos) efectuar_movimiento(Pos,Dir,Pos2) : Pos2 es la posición resultante de moverse desde Pos en la dirección indicada por Dir .

```
efectuar_movimiento(pos(X,Y),Dir,pos(X2,Y2)) :-
    dir(Dir,1),
    mover_en_dir(pos(X,Y),Dir,pos(X2,Y2)).
```

```
mover_en_dir(pos(X,Y),o,pos(X,Y2)) :-
    Y2 is Y-1.
mover_en_dir(pos(X,Y),e,pos(X,Y2)) :-
    Y2 is Y+1.
mover_en_dir(pos(X,Y),n,pos(X2,Y)) :-
    X2 is X-1.
mover_en_dir(pos(X,Y),s,pos(X2,Y)) :-
    X2 is X+1.
mover_en_dir(pos(X,Y),no,pos(X2,Y2)) :-
    X2 is X-1,
    Y2 is Y-1.
mover_en_dir(pos(X,Y),so,pos(X2,Y2)) :-
    X2 is X+1,
```

```
    Y2 is Y-1.
mover_en_dir(pos(X,Y),ne,pos(X2,Y2)) :-
    X2 is X-1,
    Y2 is Y+1.
mover_en_dir(pos(X,Y),se,pos(X2,Y2)) :-
    X2 is X+1,
    Y2 is Y+1.
```

- (0.5 puntos) movimiento_valido(N,Pos,Dir) : debe tener éxito si la posición resultado de moverse desde Pos en la dirección indicada por Dir representa una posición válida en un tablero de NxN.

```
movimiento_valido(N,Pos,Dir) :-
    efectuar_movimiento(Pos,Dir,pos(Fila,Columna)),
    between(1,N,Fila),
    between(1,N,Columna).
```

- (0.5 puntos) select_cell(IPos,Op,Board,NewBoard) : extrae (se puede consultar el predicado Prolog select/3 como inspiración) la celda con posición IPos de la lista Board obteniendo NewBoard (sin dicha celda) y unificando Op con la operación asociada a la respectiva celda.

```
select_cell(IPos,Op,Board,NewBoard) :-
    select(cell(IPos,Op),Board,NewBoard).
```

- (0.5 puntos) select_dir(Dir,Dirs,NewDirs) : extrae (se puede consultar el predicado Prolog select/3 como inspiración) una dirección Dir de las permitidas en Dirs , obteniendo en NewDirs la lista de direcciones permitidas que restan. Esta puede ser la misma lista Dirs pero con el número de aplicaciones de la dirección seleccionada disminuido en uno, o, si esta fuera la última aplicación permitida, sin ese elemento.

```
select_dir(D,Dirs,NewDirs) :-
    select(dir(Dir,X),Dirs,RestDirs),
    ( X>1 ->
        Y is X-1,
        NewDirs=[dir(Dir,Y)|RestDirs]
    ; NewDirs=RestDirs
    ).
```

- (0.5 puntos) aplicar_op(Op, Valor, Valor2) : dado Op=op(Operador,Operando) , aplica la operación indicada en Valor para obtener Valor2 .

```
aplicar_op(op(+,X),Y,Z) :-
    Z is X+Y.
aplicar_op(op(-,X),Y,Z) :-
    Z is Y-X.
aplicar_op(op(*,X),Y,Z) :-
    Z is X*Y.
aplicar_op(op(//,X),Y,Z) :-
    Z is Y//X.
```

Finalmente, para terminar este apartado se debe escribir el predicado generar_recorrido/6 (3.5 puntos). Este predicado está formalizado mediante la cabecera: generar_recorrido (Ipos ,N,Board , DireccionesPermitidas , Recorrido ,Valor) donde Ipos representa posición (columna y fila en forma de término pos(Irow,Icol)) de la celda desde la que se inicia el recorrido del tablero Board de dimensión N teniendo en cuenta las direcciones permitidas indicadas por

DireccionesPermitidas . Por otro lado, Recorrido es una lista de tuplas (Pos, ValorActual) , donde la variable Pos hace referencia a las estructuras pos/2 con cabecera pos(Row,Col) que representan la posición de las celdas que forman parte del recorrido generado cuyo valor final es Valor . Asimismo, la variable ValorActual representa al valor actual del recorrido tras la realización de la operación indicada en la celda correspondiente. Nótese que este predicado debe devolver por vuelta hacia atrás todos los recorridos que parten desde la casilla señalada por la posición Ipos usando únicamente las direcciones reseñadas en la lista DireccionesPermitidas . A modo de ejemplo, se muestra el resultado de la ejecución de una consulta que busca obtener por vuelta hacia atrás todos los recorridos (en este caso un total de 27) posibles que pueden construirse partiendo desde la casilla (2, 2) del tablero de ejemplo mostrado anteriormente, usando para ello la lista de direcciones permitidas [dir(n,5),dir(s,6),dir(e,7),dir(o,4)] .

```
?- board1(_Board),
_Dirs = [dir(n ,5) ,dir(s ,6) ,dir(e ,7) ,dir(o ,4)],
generar_recorrido (pos (2 ,2) ,4,_Board ,_Dirs ,Recorrido ,Valor).
Esta que se muestra a continuaci@'{o}n es una de las 27 respuestas que devuelve Prolog
tras la ejecuci@'{o}n de la pregunta anterior:
Recorrido = [(pos (2 ,2) ,2000) ,
(pos (1 ,2) ,1999) ,
(pos (1 ,1),-5997) ,
(pos (2 ,1),-6000) ,
(pos (3 ,1) ,0),
(pos (4 ,1),-2),
(pos (4 ,2),-1002) ,
(pos (4 ,3),-1011) ,
(pos (4 ,4),-4044) ,
(pos (3 ,4),-4024) ,
(pos (2 ,4),-4468) ,
(pos (1 ,4),-5023) ,
(pos (1 ,3),-5027) ,
(pos (2 ,3),-668591) ,
(pos (3 ,3),-334295) ,
(pos (3 ,2),-51815725)],
Valor = -51815725 ?
```

```
generar_recorrido(Ipos,N,Board,DireccionesPermitidas,Recorrido,Valor) :-
    valida(Ipos,N),
    select_cell(Ipos,op(_1,Ivalue),Board,NewBoard),
    generar_recorrido_aux(Ipos,N,NewBoard,DireccionesPermitidas,NewRecorrido,Ivalue),
    Recorrido=[(Ipos,Ivalue)|NewRecorrido],
    last(Recorrido,(_2,Valor)).
```

```
generar_recorrido_aux(Ipos,N,Board,DireccionesPermitidas,Recorrido,Ivalue) :-
    valida(Ipos,N),
    select_dir(Dir,DireccionesPermitidas,NewDirs),
    movimiento_valido(N,Ipos,Dir),
    efectuar_movimiento(Ipos,Dir,Pos),
    select_cell(Pos,op(Op,Num),Board,NewBoard),
    aplicar_op(op(Op,Num),Ivalue,Value),
    ( NewBoard=[] ->
        Recorrido=[(Pos,Value)]
    ; append([(Pos,Value)],NewRecorrido,Recorrido),
```

```
      generar_recorrido_aux(Pos,N,NewBoard,NewDirs,NewRecorrido,Value)
    ).
```

```
valida(pos(X,Y),N) :-
    between(1,N,X),
    between(1,N,Y).
```

Apartado 2 (1.25 puntos)

A continuación, el alumno debe escribir el predicado generar_recorridos/5 con cabecera generar_recorridos(N, Board, DireccionesPermitidas, Recorrido, Valor) . Este predicado genera por vuelta hacia atrás el valor Valor del recorrido Recorrido correspondiente a todos los recorridos posibles en el tablero Board de dimensión N partiendo desde cualquier casilla, y teniendo en cuenta las direcciones permitidas en la lista DireccionesPermitidas . Este predicado facilitará al alumno el averiguar en el siguiente apartado cual es el valor mínimo de entre los proporcionados por todos los recorridos posibles. El recorrido devuelto en la variable Recorrido debe seguir el mismo formato que en el predicado anterior. A continuación se muestra un ejemplo de uso del citado predicado.

```
?- board1(_Board),
_Dirs = [dir(n ,5) ,dir(s ,6) ,dir(e ,7) ,dir(o ,4)],
generar_recorridos (4,_Board ,_Dirs ,R,V).
R = [(pos (1 ,1) ,0),

(pos (2 ,1),-3),
(pos (3 ,1) ,0),
(pos (4 ,1),-2),
(pos (4 ,2),-1002) ,
(pos (4 ,3),-1011) ,
(pos (4 ,4),-4044) ,
(pos (3 ,4),-4024) ,
(pos (2 ,4),-4468) ,
(pos (1 ,4),-5023) ,
(pos (1 ,3),-5027) ,
(pos (1 ,2),-5028) ,
(pos (2 ,2),-3028) ,
(pos (3 ,2),-469340) ,
(pos (3 ,3),-234670) ,
(pos (2 ,3),-31211110)],
V = -31211110 ?
```

Al igual que en el ejemplo anterior, aquí solamente se muestra una única respuesta. No obstante, el alumno debe tener en cuenta que este ejemplo devuelve un total de 362 soluciones por vuelta hacia atrás. El predicado implementado por el alumno debe devolver por vuelta hacia atrás todas las soluciones posibles.

```
generar_recorridos(N,Board,DireccionesPermitidas,Recorrido,Valor) :-
    generar_recorrido(_1,N,Board,DireccionesPermitidas,Recorrido,Valor).
```

Apartado 3 (1.25 puntos)

Para finalizar, se pide al alumno que escriba un predicado tablero/5 descrito mediante la cabecera: tablero(N, Tablero , DireccionesPermitidas , ValorMinimo ,

NumeroDeRutasConValorMinimo) que se hace cierto si ValorMinimo unifica con el mínimo valor final que es posible obtener teniendo en cuenta todas las rutas posibles comenzando en una casilla cualquiera del tablero de dimensión N , y utilizando únicamente los movimientos indicados en la lista DireccionesPermitidas para transitar entre las diferentes casillas del tablero. Además, la variable NumeroDeRutasConValorMinimo debe unificarse con el número de rutas existentes que permiten obtener el valor mínimo indicado por ValorMinimo . Si no existiese ninguna ruta posible, el programa debe fallar. Se recomienda al alumno que repase los predicados de agregación estudiados en las clases de teoría porque pueden serle de mucha utilidad en la implementación de este predicado. A continuación se muestra un ejemplo de uso de este predicado.

```
?- board1(_Board),
_Dirs = [dir(n ,5) ,dir(s ,6) ,dir(e ,7) ,dir(o ,4)],
tablero (4,_Board ,_Dirs ,VM ,NR).
NR = 1,
VM = -246992940 ? ;
no
```

```
tablero(N,Tablero,DireccionesPermitidas,ValorMinimo,NumeroDeRutasConValorMinimo) :-
    findall(Valor,generar_recorridos(N,Tablero,DireccionesPermitidas,_1,Valor),Valores)
    min_list(Valores,ValorMinimo),
    findall(MinRecorrido,generar_recorridos(N,Tablero,DireccionesPermitidas,MinRecorrid
    length(MinRecorridos,NumeroDeRutasConValorMinimo).
```

```
min_list(Lista,Min) :-
    member(Min,Lista),
    \+ (member(E,Lista),E<Min).
```

Usage and interface

Library usage:

```
:-
use_module(/mnt/c/Users/UsuarioPC/Desktop/GITHUB_REP/RAUL/prolog/practica2/code.pl).
```

Exports:

- *Predicates:*
  `author_data/4` , `efectuar_movimiento/3` , `mover_en_dir/3` , `movimiento_valido/3` , `select_cell/4` , `select_dir/3` , `aplicar_op/3` , `generar_recorrido/6` , `generar_recorrido_aux/6` , `generar_recorridos/5` , `min_list/2` , `tablero/5` .
- *Properties:*
  `cell/2` , `pos/2` , `op/2` , `dir/2` , `valida/2` .

Documentation on exports

PREDICATE author_data/4

No further documentation available for this predicate.

PROPERTY cell/2

Usage: `cell(Pos,Op)`

Estructura que representa una celda del tablero. El primer argumento es la posición de la celda y el segundo es la operación.

```
cell(pos(X,Y),op(Op,Value)) :-
    pos(X,Y),
    op(Op,Value).
```

PROPERTY pos/2

Usage: `pos(X,Y)`

Estructura que representa una posición en el tablero. El primer argumento es la fila y el segundo es la columna.

```
pos(X,Y) :-
    integer(X),
    integer(Y).
```

PROPERTY op/2

Usage: `op(Op,Value)`

Estructura que representa una operación. El primer argumento es el operador y el segundo es el operando.

```
op(Op,Value) :-
    member(Op,[*,-,+,//]),
    integer(Value).
```

PROPERTY dir/2

Usage:

Estructura que representa una dirección. El primer argumento es la dirección y el segundo es el número de veces que se puede utilizar dicha dirección.

```
dir(N,X) :-
    member(N,[n,s,e,o,no,ne,se,so]),
    integer(X),
    X>0.
```

PREDICATE efectuar_movimiento/3

Usage: `efectuar_movimiento(+Pos,+Dir,-Pos2)`

Predicado que calcula la posición resultante de moverse desde Pos en la dirección indicada por Dir.

```
efectuar_movimiento(pos(X,Y),Dir,pos(X2,Y2)) :-
    dir(Dir,1),
    mover_en_dir(pos(X,Y),Dir,pos(X2,Y2)).
```

PREDICATE mover_en_dir/3

Usage: `mover_en_dir(+Pos,+Dir,-Pos2)`

Predicado auxiliar que calcula la posición resultante de moverse desde Pos en la dirección indicada por Dir.

```
mover_en_dir(pos(X,Y),o,pos(X,Y2)) :-
    Y2 is Y-1.
mover_en_dir(pos(X,Y),e,pos(X,Y2)) :-
    Y2 is Y+1.
mover_en_dir(pos(X,Y),n,pos(X2,Y)) :-
    X2 is X-1.
mover_en_dir(pos(X,Y),s,pos(X2,Y)) :-
    X2 is X+1.
mover_en_dir(pos(X,Y),no,pos(X2,Y2)) :-
    X2 is X-1,
    Y2 is Y-1.
mover_en_dir(pos(X,Y),so,pos(X2,Y2)) :-
    X2 is X+1,
    Y2 is Y-1.
mover_en_dir(pos(X,Y),ne,pos(X2,Y2)) :-
    X2 is X-1,
    Y2 is Y+1.
mover_en_dir(pos(X,Y),se,pos(X2,Y2)) :-
    X2 is X+1,
    Y2 is Y+1.
```

PREDICATE movimiento_valido/3

Usage: `movimiento_valido(+N,+Pos,+Dir)`

Predicado que verifica si la posición resultante de moverse desde Pos en la dirección indicada por Dir representa una posición válida en un tablero de NxN.

```
movimiento_valido(N,Pos,Dir) :-
    efectuar_movimiento(Pos,Dir,pos(Fila,Columna)),
    between(1,N,Fila),
    between(1,N,Columna).
```

PREDICATE select_cell/4

Usage: `select_cell(+IPos,-Op,+Board,-NewBoard)`

Predicado que extrae la celda con posición IPos de la lista Board obteniendo NewBoard (sin dicha celda) y unificando Op con la operación asociada a la respectiva celda.

```
select_cell(IPos,Op,Board,NewBoard) :-
    select(cell(IPos,Op),Board,NewBoard).
```

PREDICATE select_dir/3

Usage: `select_dir(+Dir,+Dirs,-NewDirs)`

Predicado que extrae una dirección Dir de las permitidas en Dirs , obteniendo en NewDirs la lista de direcciones permitidas que restan. Esta puede ser la misma lista Dirs pero con el número de aplicaciones de la dirección seleccionada disminuido en uno, o, si esta fuera la última aplicación permitida, sin ese elemento.

```
select_dir(D,Dirs,NewDirs) :-
    select(dir(Dir,X),Dirs,RestDirs),
    ( X>1 ->
        Y is X-1,
        NewDirs=[dir(Dir,Y)|RestDirs]
    ; NewDirs=RestDirs
    ).
```

PREDICATE aplicar_op/3

Usage: `aplicar_op(+Op,+X,-Z)`

Predicado que aplica la operación indicada en X para obtener Z.

```
aplicar_op(op(+,X),Y,Z) :-
    Z is X+Y.
aplicar_op(op(-,X),Y,Z) :-
    Z is Y-X.
aplicar_op(op(*,X),Y,Z) :-
    Z is X*Y.
aplicar_op(op(//,X),Y,Z) :-
    Z is Y//X.
```

PROPERTY valida/2

Usage:

Verifica si una posición es válida en un tablero de NxN.

```
valida(pos(X,Y),N) :-
    between(1,N,X),
    between(1,N,Y).
```

PREDICATE generar_recorrido/6

Usage: `generar_recorrido(+Ipos,+N,+Board,+DireccionesPermitidas,-Recorrido,+Valor)`

Predicado que genera el recorrido y calcula el valor.

```
generar_recorrido(Ipos,N,Board,DireccionesPermitidas,Recorrido,Valor) :-
    valida(Ipos,N),
    select_cell(Ipos,op(_1,Ivalue),Board,NewBoard),
    generar_recorrido_aux(Ipos,N,NewBoard,DireccionesPermitidas,NewRecorrido,Ivalue),
    Recorrido=[(Ipos,Ivalue)|NewRecorrido],
    last(Recorrido,(_2,Valor)).
```

PREDICATE generar_recorrido_aux/6

Usage: `generar_recorrido_aux(+Ipos,+N,+Board,+DireccionesPermitidas,-Recorrido,+Ivalue)`

redicado auxiliar que genera el recorrido.

```
generar_recorrido_aux(Ipos,N,Board,DireccionesPermitidas,Recorrido,Ivalue) :-
    valida(Ipos,N),
    select_dir(Dir,DireccionesPermitidas,NewDirs),
    movimiento_valido(N,Ipos,Dir),
    efectuar_movimiento(Ipos,Dir,Pos),
    select_cell(Pos,op(Op,Num),Board,NewBoard),
    aplicar_op(op(Op,Num),Ivalue,Value),
    ( NewBoard=[] ->
        Recorrido=[(Pos,Value)]
    ; append([(Pos,Value)],NewRecorrido,Recorrido),
      generar_recorrido_aux(Pos,N,NewBoard,NewDirs,NewRecorrido,Value)
    ).
```

PREDICATE generar_recorridos/5

Usage: `generar_recorridos(+N,+Board,+DireccionesPermitidas,-Recorrido,-Valor)`

Predicado que genera todos los recorridos posibles en el tablero Board de dimensión N partiendo desde cualquier casilla, y teniendo en cuenta las direcciones permitidas en la lista DireccionesPermitidas .

```
generar_recorridos(N,Board,DireccionesPermitidas,Recorrido,Valor) :-
    generar_recorrido(_1,N,Board,DireccionesPermitidas,Recorrido,Valor).
```

PREDICATE min_list/2

Usage: `min_list(+Lista,-Min)`

Predicado que calcula el mínimo de una lista de enteros.

```
min_list(Lista,Min) :-
    member(Min,Lista),
    \+ (member(E,Lista),E<Min).
```

PREDICATE tablero/5

Usage: `tablero(+N,+Tablero,+DireccionesPermitidas,-ValorMinimo,-NumeroDeRutasConValorMinimo)`

Predicado que calcula el valor mínimo de entre los proporcionados por todos los recorridos posibles. El recorrido devuelto en la variable Recorrido debe seguir el mismo formato que en el predicado anterior.

```
tablero(N,Tablero,DireccionesPermitidas,ValorMinimo,NumeroDeRutasConValorMinimo) :-
    findall(Valor,generar_recorridos(N,Tablero,DireccionesPermitidas,_1,Valor),Valore
    min_list(Valores,ValorMinimo),
    findall(MinRecorrido,generar_recorridos(N,Tablero,DireccionesPermitidas,MinRecorr
    length(MinRecorridos,NumeroDeRutasConValorMinimo).
```

Documentation on imports

This module has the following direct dependencies:

- *Internal (engine) modules:*
  `term_basic` , `arithmetic` , `atomic_basic` , `basiccontrol` , `exceptions` , `term_compare` , `term_typing` , `debugger_support` , `basic_props` .
- *Packages:*
  `prelude` , `initial` , `condcomp` , `assertions` , `assertions/assertions_basic` , `regtypes` .