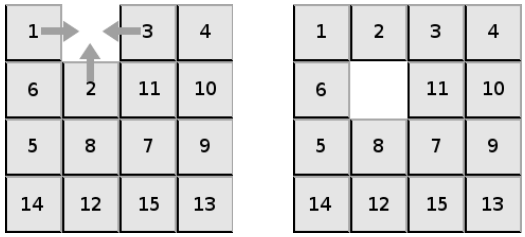
Evaluación (1ª instancia)

Duración: 3 horas

Solución

Ejercicio 1 [40 puntos]

Un *puzle de deslizamiento* (*sliding puzzle*) es un juego donde se distribuye un conjunto de piezas desordenadas sobre una grilla dejando una celda libre. El objetivo es ubicar todas las piezas en orden dejando la celda libre en último lugar (abajo a la derecha), con la limitante de que solamente se puede mover una pieza por vez, deslizándola hacia el lugar vacío desde una posición adyacente.



Por ejemplo, en el puzle del lado izquierdo de la figura se ven tres movimientos posibles: **arriba** (mover la pieza 2 hacia arriba), **izquierda** (mover la pieza 3 hacia la izquierda) o **derecha** (mover la pieza 1 a la derecha). Del lado derecho de la figura se ve el resultado luego de aplicar el movimiento **arriba**.

En el caso general puede haber hasta cuatro movimientos posibles: **arriba**, **abajo**, **izquierda** y **derecha**. Una solución del juego es una secuencia de movimientos posibles que lleva del puzle actual hasta su resolución.

Un puzle se representa como una matriz de tamaño MxN (puede no ser cuadrada) que contiene los números del 1 al (M*N-1) y una celda con el átomo **vacio**. Considere que los siguientes predicados para manejo de matrices ya están implementados:

- `dim(+M,?F,?C)` ← F y C son las cantidades de filas y columnas de la matriz M
- `celda(+M,?F,?C,?V)` ← V es el valor de la celda en la fila F y la columna C de la matriz M
- nueva_celda(+M,+F,+C,+V,?M2) ← M2 es una matriz igual a M donde la celda de la fila F y la columna C tienen el nuevo valor V
- `serializar(+M,?S)` ← S es una lista con todas las celdas de M serializadas por fila, por ejemplo:
  ```
  serializar([[4,5,vacio],[7,6,3],[8,2,1]],[4,5,vacio,7,6,3,8,2,1]).
  ```

Implemente en Prolog los siguientes predicados:

a) `buscar_vacio(+P,?F,?C)` ← F y C son la fila y la columna del elemento vacío en el puzle P.

b) `posicion_final(+F,+C,?P)` ← P es la posición final de un puzle de tamaño FxC, o sea, las piezas están ubicadas en orden por fila y la última es el vacío. Por ejemplo:

```
posicion_final(3,3,[[1,2,3],[4,5,6],[7,8,vacio]).
```

c) `mover(+P,+F,+C,+Mov,?P2,?F2,?C2)` ← P es un puzle, (F,C) son las coordenadas del elemento vacío, Mov puede ser el átomo **arriba**, **abajo**, **izquierda** o **derecha**; P2 es el resultado de aplicar el movimiento Mov al puzle P y (F2,C2) son las coordenadas del elemento vacío luego del movimiento.

d) `resolver_puzle(P,S)` ← S es una secuencia de movimientos que resuelven el puzle P. Por ejemplo:

```
?- resolver_puzle([[vacio,1],[3,2]],S).
S = [izquierda,arriba]
```

Solución

a)
```
buscar_vacio(P,F,C):-
    celda(P,F,C,vacio),!.
```

b)
```
posicion_final(F,C,P):-
    L is F*C - 1,
    findall(X,between(1,L,X),Nums),
    append(Nums,[vacio],Piezas),
    matriz(F,C,P),
    serializar(P,Piezas).
```

c)
```
mover(P,F,C,arriba,P3,F1,C):-
    F1 is F + 1,
    celda(P,F1,C,V),
    nueva_celda(P,F1,C,vacio,P2),
    nueva_celda(P2,F,C,V,P3).
mover(P,F,C,izquierda,P3,F,C1):-
    C1 is C + 1,
    celda(P,F,C1,V),
    nueva_celda(P,F,C1,vacio,P2),
    nueva_celda(P2,F,C,V,P3).
mover(P,F,C,abajo,P3,F1,C):-
    F1 is F - 1,
    celda(P,F1,C,V),
    nueva_celda(P,F1,C,vacio,P2),
    nueva_celda(P2,F,C,V,P3).
mover(P,F,C,derecha,P3,F,C1):-
    C1 is C - 1,
    celda(P,F,C1,V),
    nueva_celda(P,F,C1,vacio,P2),
    nueva_celda(P2,F,C,V,P3).
```

d)
```
resolver_puzle(P,S):-
    buscar_vacio(P,F,C),
    dim(P,DF,DC),
    posicion_final(DF,DC,Final),
    resolver_puzle(P,Final,F,C,[P],S).

resolver_puzle(P,P,_,_,_,[]):-!.
resolver_puzle(P,Final,F,C,Visto,[Mov|Resto]):-
    member(Mov,[arriba,abajo,izquierda,derecha]),
    mover(P,F,C,Mov,P2,F2,C2),
    \+ member(P2,Visto),
    resolver_puzle(P2,Final,F2,C2,[P2|Visto],Resto).
```

Ejercicio 2 [15 puntos]

Considere el siguiente programa lógico P, con predicados p de aridad 2 y q de aridad 1:

```
p(Z,X) ← p(X,Y) , P(Y,Z)
p(X,X)

q(f(f(X))) ← q(X)
q(a)
q(f(b))
```

a) Defina una interpretación para P que sea modelo y una que no lo sea.

b) Considere el objetivo ← p(a,X),q(X).

i. Indique dos respuestas computadas
ii. ¿Existe alguna respuesta correcta que no sea computada? Justifique.
iii. ¿Existe alguna respuesta computada que no sea correcta? Justifique.

Solución

a) Sea una pre-interpretacion I con dominio en los naturales, y donde la función f mapea en la función sucesor, la constante a mapea en 0 y la constante b mapea en 1

a1) La siguiente interpretación I1 es modelo, siendo I1 tal que agrega a I una interpretación para los predicados tal que p mapea en un predicado binario verdadero para todo par de naturales y q mapea en un predicado unario verdadero para todo natural.

a2) Considere I2 idéntica a I1 excepto en en que q mapea en la propiedad de ser mayor que 0. I2 no es modelo.

b)

i. Existe una sola respuesta computada, X= a, que se obtiene múltiples vece.s

ii. No, la respuesta computada que se obtiene está completamente instanciada, no podría haber más respuestas correctas.

iii. Todas las respuestas computadas son correctas. Esta es una propiedad que se demuestra.

Ejercicio 3 [20 puntos]

Considere el siguiente programa Prolog:

```
p([A|B],[_C|D],[A|X]):-
        p(B,D,X).
p([_|B],[_|C],Y):-
        p(B,[_|C],Y).
p(_,[],[]).
p(X,[_A,B|C],Y):-
        p(X,[B|C],Y).
```

a) Dibuje el árbol SLD correspondiente a la consulta `p([a,b,c],[48,101],X)`. Enumere los valores de X obtenidos como solución.

b) Considere 3 variantes del programa anterior, consistentes en poner un cut (!) como primer elemento en el cuerpo de las cláusulas 1 (Variante 1), 2 (Variante 2) y 4 (Variante 3).
Indique, para cada uno de esos cuts, si son rojos o verdes y cuáles son las soluciones para la consulta de la parte a.

Solución

a) Se muestran las soluciones

?- p([a,b,c],[48,101],X).
{X=[a,b],[b,c],[a,c],[c],[b],[a]}

b)
1- Cut rojo, soluciones {X=[a,b]}
2- Cut rojo, soluciones {X=[a,b],[b,c],[a,c]}
3- Cut verde, {X=[a,b],[b,c],[a,c],[c],[b],[a]}

Ejercicio 4 [10 puntos]

a) Implemente el siguiente predicado utilizando listas de diferencias:
`append_all_dl(+DLs,?DL)` ← DLs es una lista (común) de listas de diferencia, DL es la lista de diferencias resultante de concatenar todas las listas de diferencia de DLs.

Por ejemplo:

```
?- append_all_dl([[1,2,3|X]-X,[4|Y]-Y,[5,6|Z]-Z],A-AR).
A = [1,2,3,4,5,6|AR].
```

b) Utilizando DCG, define un programa Prolog para reconocer el lenguaje:
$L = \{a^n b^n c^n d^n / n \geq 0\}$

Solución

a)

```
append_all_dl(Ls,A-AR):-
    append_all_dl(X-X,Ls,A-AR).

append_all_dl(A-AR,[],A-AR).
append_all_dl(A-B,[B-BR|Resto],C-CR):-
    append_all_dl(A-BR,Resto,C-CR).
```

b)

```
s --> a(N),b(N),c(N),d(N).
a(0) --> [].
a(N) --> [a], a(N1), {N is N1 + 1}.
b(0) --> [].
b(N) --> [b], b(N1), {N is N1 + 1}.
c(0) --> [].
c(N) --> [c], c(N1), {N is N1 + 1}.
d(0) --> [].
d(N) --> [d], d(N1), {N is N1 + 1}.
```