Evaluación (1ª instancia)

Duración: 3 horas
Solución

Ejercicio 0: Pregunta individual sobre el laboratorio

Ejercicio 1 [30 puntos]

Considere el problema del puente y la linterna. Hay cuatro personas intentando cruzar un puente sobre un río en la noche, y tienen una sola linterna. El puente soporta como máximo dos personas. Las personas demoran tiempos diferentes en cruzar el puente: la persona A demora 1 minuto, la persona B 2 minutos, la persona C 5 minutos y la persona D 8 minutos. Cuando dos personas cruzan el puente, demoran lo que demore la persona más lenta. La pregunta es: ¿existe una forma de cruzar el puente antes de que agote la batería de la linterna, que dura 15 minutos?

Implemente en Prolog los siguientes predicados:

a) `cruce(?Origen, ?Destino, ?T)`: existe un cruce posible (de una o dos personas, con la linterna) partiendo de `Origen` y llegando a `Destino`, que insume un tiempo T. Tanto `Origen` como `Destino` están representados por el término `estado(Izq,Der,Linterna)`, donde `Izq` y `Der` son listas ordenadas con las personas que están, respectivamente, a la izquierda y a la derecha del puente, y `Linterna` puede ser `left` o `right`, según el lado en el que se encuentra la linterna. Para que el cruce sea posible, la linterna debe estar del lado del lado del puente donde el cruce comienza.

```
cruce(estado([a,b,c],[d],left),estado([a,b],[c,d],right),5).

cruce(estado([a,b,c],[d],left),estado([a],[b,c,d],right),5).

cruce(estado([a,b],[c,d],right),estado([a,b,c,d],[],left),8).
```

Ayuda: para asegurar que la lista está siempre ordenada, puede utilizar el predicado `sort` de Prolog (`sort(+Lista,-ListaOrd)`: `ListaOrd` es la versión en orden lexicográfico de `Lista`).

b) `recorrido(?Camino,?T)`: existe una secuencia de cruces que resuelve el problema en un tiempo T, a partir de la configuración inicial. `Camino` es la lista de estados en la secuencia.

```
recorrido([estado([a,b,c,d],[],left),estado([a,b],[c,d],right),

estado([a,b,c],[d],left),estado([a],[b,c,d],right),

estado([a,b],[c,d],left),estado([],[a,b,c,d],right)],22).
```

c) `problema(?Camino)`: existe una secuencia de cruces que resuelve el problema del puente y la linterna (es decir, encuentra una secuencia de estados que resuelve el problema en un tiempo menor o igual a 15 minutos).

d) `recorrido2`: muestra en consola todos los caminos que resuelven el cruce (sin importar el tiempo), utilizando un mecanismo loop-fail.

Solución

a)

```
tiempo_cruce(a,1).
tiempo_cruce(b,2).
tiempo_cruce(c,5).
tiempo_cruce(d,8).


tiempo_cruce_dos(X,Y,T):-
    tiempo_cruce(X,T1),
```

```
    tiempo_cruce(Y,T2),
  T is max(T1,T2).

% Cruce de una persona de izquierda a derecha
% Asume que X está en Left
cruce(estado(Left,Right,left), estado(Left1,Right2,right), Time):-
    select(X,Left,Left1),
    append(Right,[X],Right1),
    tiempo_cruce(X,Time),
    sort(Right1,Right2).

% Cruce de derecha a izquierda
cruce(estado(Left,Right,right),estado(Left1,Right1,left),Time):-
    cruce(estado(Right,Left,left),estado(Right1,Left1,right),Time).

% Cruce de dos personas de izquierda a derecha
cruce(estado(Left,Right,left), estado(Left2,Right3,right),Time):-
    select(X,Left,Left1),
    select(Y,Left1,Left2),
    append(Right,[X],Right1),
    append(Right1,[Y],Right2),
    tiempo_cruce_dos(X,Y,Time),
    sort(Right2,Right3).

% Cruce de derecha a izquierda
cruce(estado(Left,Right,right),estado(Left1,Right1,left),Time):-
    cruce(estado(Right,Left,left),estado(Right1,Left1,right),Time).
```

b)

```
% Hacemos un camino
% Cada cruce es un arco

camino(X,Y,Path,Time) :- camino(X,Y,[X],Path,Time).

camino(X,Y,Visitados,[X,Y],Time):-
    cruce(X,Y,Time),
    \+member(Y,Visitados).

camino(X,Y,Visitados,[X|Path1],Time) :-
    cruce(X,Z,TimeCruce),
    \+ member(Z,Visitados),
    camino(Z,Y,[X|Visitados],Path1,Time1),
    Time is Time1+TimeCruce.

% Estado final
final(estado([],Final,_)):-
    sort(Final,[a,b,c,d]).

% Estado inicial
inicial(estado([a,b,c,d],[],left)).

% Recorrido es un camino entre un estado inicial y uno final
recorrido(Path,Time):-
```

```
    inicial(X),
    camino(X,Y,Path,Time),
    final(Y).
```

c)

```
problema(Camino):-
    recorrido(Camino,Time),
    Time =< 15.
```

d)

```
recorrido2 :-
       inicial(X),
       camino(X,Y,Path,Time),
       final(Y),
       write(Path),writeln(Time),nl,
       fail.
```

Ejercicio 2 [15 puntos]

a) Indique si son verdaderas o falsas las siguientes afirmaciones. Fundamente.

i) Una cláusula de Horn contiene uno y solo un literal positivo.
ii) Un conjunto de fórmulas cerradas es insatisfactible si no hay ninguna interpretación posible que sea modelo del conjunto.
iii) Dado un programa lógico y un objetivo, el árbol SLD resultante solo puede contener ramas de éxito y ramas infinitas.
iv) Existe al menos un m.g.u. para todo par de expresiones.

b) Aplique el algoritmo para hallar el m.g.u. de las siguientes dos expresiones, mostrando su aplicación paso a paso:

```
siguiente(X,g(X,Y),a(W))          siguiente(a(Z),T,Z)
```

Solución

a)
i) Falso. Las cláusulas de Horn pueden tener cero o un literal positivos, son la unión entre las cláusulas definidas y los objetivos definidos.
ii) Verdadero. Por definición de insatisfactibilidad.
iii) Falso. Puede tener ramas de éxito, ramas de fallo, y ramas infinitas.
iv) Falso. Existen pares de expresiones que no unifican, por lo tanto no tienen ningún m.g.u. por ejemplo f(X) y g(X).

b)

```
stack={ siguiente(X,g(X,Y),a(W)) = siguiente(a(Z),T,Z) }          mu={}
caso 4
stack={ X = a(Z), g(X,Y) = T, a(W) = Z }                          mu={}
caso 1: X/a(Z)
stack={ g(a(Z),Y) = T, a(W) = Z }                        mu={ X/a(Z) }
caso 2: T/g(a(Z),Y)
stack={ a(W) = Z }                            mu={ X/a(Z), T/g(a(Z),Y) }
caso 2: Z/a(W)
stack={}                       mu={ X/a(a(W)), T/g(a(a(W)),Y), Z/a(W) }
fin, resultado: mu={ X/a(a(W)), T/g(a(a(W)),Y), Z/a(W) }
```

Ejercicio 3 [25 puntos]

Considere el siguiente programa Prolog:

```
a(1).
b(2).
b(3).

pq(f(X),f(Y),f(Z)) :- a(X),a(Y),b(Z).
```

```
pq(X,Y,Z) :- pq(f(Z),Z,Y).
pq(1,1,1).
```

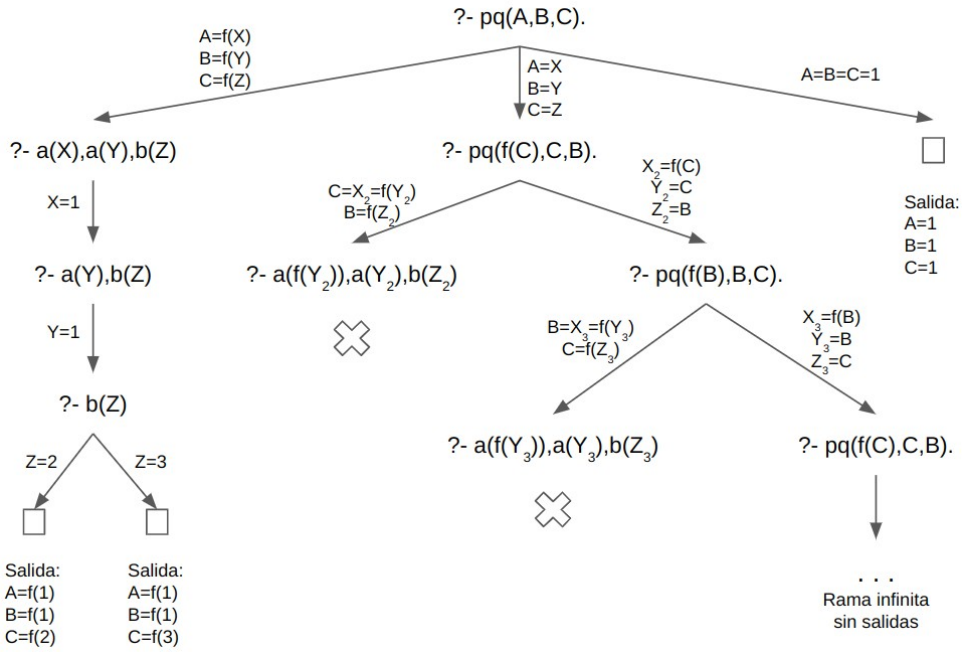
a) Dibuje el árbol SLD correspondiente a la consulta **?- p(A,B,C)** suponiendo que la regla de computación toma el átomo de más a la izquierda para la resolución.
b) ¿Qué respuestas dará el intérprete Prolog para el objetivo anterior?
c) ¿Cuáles son las respuestas si cambiamos la primera cláusula del predicado `pq/3` por la siguiente? Justifique.

```
pq(f(X),f(Y),f(Z)) :- a(X),a(Y),!,b(Z).
```

d) ¿Qué respuestas devolvería la consulta de parte a) si el intérprete realizara un recorrida en amplitud del árbol SLD?

Solución

a)



b) Las respuestas son las primeras dos salidas marcadas en el árbol:
A=f(1), B=f(1), C=f(2)
A=f(1), B=f(1), C=f(3)
Luego cae por una rama infinita.

c) Se mantienen las dos salidas, al llegar al cut se podan la segunda y tercera ramas del primer objetivo, pero mantiene las dos posibles instanciaciones para b(Z).

d) Si se recorriera en amplitud, se obtendrían las tres salidas:
A=1, B=1, C=1
A=f(1), B=f(1), C=f(2)
A=f(1), B=f(1), C=f(3)
Luego caería por una rama infinita.

Ejercicio 4 [15 puntos]

a) Considere un escenario donde se lanzan 3 dados, todos ellos perfectamente balanceados. Construya programas en Problog que permitan calcular:

i) La probabilidad de que los tres dados salgan iguales.
ii) La probabilidad de que salga una escalera (es decir, tres valores seguidos en los dados)
iii) La probabilidad de que salga una escalera, sabiendo que el primer dado salió 5 (utilice la capacidad de Problog de especificar evidencia).

b) Utilizando DCG, defina un programa Prolog para reconocer el siguiente lenguaje:

$L = \{a^n b^m c^n d^m / n,m \geq 0\}$

Solución

a)

```
dado(1).
dado(2).
dado(3).

1/6::elegir(X,1) ; 1/6::elegir(X,2) ; 1/6::elegir(X,3) ; 1/6::elegir(X,4) ;
1/6::elegir(X,5) ; 1/6::elegir(X,6).

dado(X,Y) :- dado(X),elegir(X,Y).

todos_iguales :- dado(1,Y), dado(2,Y), dado(3,Y).

escalera :- dado(1,X),dado(2,Y),dado(3,Z),Y is X+1, Z is Y+1. % 1 2 3
escalera :- dado(1,X),dado(2,Y),dado(3,Z),Y is X+2, Z is X+1. % 1 3 2
escalera :- dado(1,X),dado(2,Y),dado(3,Z),X is Y+1, Z is Y+2. % 2 1 3
escalera :- dado(1,X),dado(2,Y),dado(3,Z),X is Z+2, Y is Z+1. % 2 3 1
escalera :- dado(1,X),dado(2,Y),dado(3,Z),X is Y+2, Z is Y+1. % 3 1 2
escalera :- dado(1,X),dado(2,Y),dado(3,Z),X is Z+2, Y is Z+1. % 3 2 1

% Evidencia de que el primer dado salió 5
% evidence(dado(1,5)).

query(todos_iguales).
query(escalera).
```

b)

```
s --> a(N),b(M),c(N),d(M).

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}.
```