Evaluación (1ª instancia)

Duración: 3 horas

Ejercicio 0: Pregunta individual sobre el laboratorio

Describa brevemente los algoritmos utilizados por su jugador para la Fase 1 y para la fase 2 del Arabelog.

Ejercicio 1 [30 puntos]

Considere un puzzle unidimensional con piezas como las siguientes compuestas de secuencias de letras y puntos:

```
1) i..m.n..o      2) d....s....l      3) u...i......n      4) n....e..i..a
```

Los puntos representan espacios vacíos. Colocar una pieza sobre otra implica dos acciones: buscar el primer lugar vacío en la primera pieza, y a partir de allí ubicar las letras y espacios vacíos de la segunda pieza respetando los espacios y letras de la anterior.

El objetivo del puzzle es colocar todas las piezas de manera que armen un solo texto sin espacios. Por ejemplo, con las piezas anteriores se puede armar el siguiente texto:

`u...i......n`	Comenzando con la pieza 3.
`__n....e..i..a`	Colocamos la pieza 4 en el primer lugar vacío.
`un..i.e..i.na`	Esta pieza (pieza 5) es el resultado luego de colocar la pieza 4.
`__i..m.n..o__`	Colocamos la pieza 1 en el primer lugar vacío de la pieza 5.
`uni.imen.iona`	Esta pieza (pieza 6) es el resultado luego de colocar la pieza 1.
`__d....s....l__`	Colocamos la pieza 2 en el primer lugar vacío de la pieza 6.
`unidimensional`	Esta pieza final es el resultado luego de colocar la pieza 2.

En Prolog, representaremos el conjunto de piezas como una lista, donde cada pieza es una lista de átomos (las letras) o variables sin instanciar (en lugar de los puntos). El puzzle anterior se representa de la siguiente manera:

```
Piezas1 = [[i,_,_,m,_,n,_,_,o], [d,_,_,_,_,s,_,_,_,_,l],
           [u,_,_,_,i,_,_,_,_,_,n], [n,_,_,_,_,e,_,_,i,_,_,a]].
```

Implemente en Prolog los siguientes predicados para resolver el puzzle unidimensional:

a) colocar_pieza(Pieza1,Pieza2,?Resultado) ← Resultado es el resultado de colocar la Pieza2 sobre la Pieza1, comenzando en el primer lugar vacío de Pieza1. Por ejemplo:

```
colocar_pieza([a,_,_],[b],[a,b,_]).
colocar_pieza([c],[b,_,a,_],[c,b,_,a,_]).
colocar_pieza([c,e,_,d],[b,_,a,_],[c,e,b,d,a,_]).
colocar_pieza([c,e,_,d],[b,a,_],X).
```

← Este ejemplo debe fallar porque las piezas no son compatibles.

b) resolver_puzzle(+Piezas,?Solucion) ← Solucion es una solución al puzzle formado por Piezas. Por ejemplo:

```
resolver_puzzle(Piezas1,[u,n,i,d,i,m,e,n,s,i,o,n,a,l]).
```

c) puzzle_bien_formado(+Piezas) ← Piezas representa un puzzle bien formado, esto significa que para el conjunto de piezas existe una sola solución. Por ejemplo:

```
?- puzzle_bien_formado(Piezas1).
true
```

```
?- puzzle_bien_formado([[a,_,o,_,a],[r,_,m,_,n]]).
false
```

Notar que en el ejemplo de la derecha existen dos soluciones: “ramona” y “aroman”, por lo que no es un puzzle bien formado.

Solución

a)

```
colocar_pieza([],Pieza,Pieza).
colocar_pieza([X|Resto],Pieza,[X|Resultado]):-
   nonvar(X),!,
   colocar_pieza(Resto,Pieza,Resultado).
colocar_pieza([X|Resto],Pieza,Resultado):-
   var(X),!,
   colocar_pieza_lugar([X|Resto],Pieza,Resultado).


colocar_pieza_lugar(X,[],X).
colocar_pieza_lugar([],Y,Y).
colocar_pieza_lugar([X|Xs],[X|Ys],[X|Zs]):-
   colocar_pieza_lugar(Xs,Ys,Zs).
```

b)

```
resolver_puzzle([],[]).
resolver_puzzle(Piezas,Solucion):-
   select(Pieza,Piezas,Resto),
   resolver_puzzle(Resto,SolucionParcial),
   colocar_pieza(Pieza,SolucionParcial,Solucion).
```

c)

```
puzzle_bien_formado(Piezas):-
   findall(S,resolver_puzzle(Piezas,S),Soluciones),
   length(Soluciones,1).
```

Ejercicio 2 [15 puntos]

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

i) Es posible encontrar el mgu para cualquier par de expresiones.
ii) Una cláusula definida puede contener cualquier cantidad de literales positivos y negativos.
iii) Un árbol SLD contiene todas las respuestas computadas para una consulta.
iv) Todas las ramas finitas de un árbol SLD son ramas exitosas (la hoja es un objetivo vacío).

b) Sean C1 y C2 dos cláusulas de primer orden:
C1 : {L1, ... , Ln}
C2 : {M1, ... , Mk}
y θ un mgu de Li y ¬Mj
Escriba la resolvente binaria de C1 y C2 y diga qué restricción deben cumplir las variables de las cláusulas.

Solución

a)
i) Falso: solo es posible si las expresiones unifican, por ejemplo p(a) y p(b) no unifican.
ii) Falso: por definición, una cláusula definida puede tener uno y solo un literal positivo.
iii) Verdadero: por definición de respuesta computada.
iv) Falso: las ramas finitas del árbol SLD pueden ser ramas exitosas o ramas de falla.

b)
La resolvente es: C : {L1, . Li-1,Li+1, . . , Ln, M1, ..., Mj-1, Mj+1, ..., Mk}θ
Las cláusulas C1 y C2 no pueden compartir variables (lo cual se resuelve con un renombre de variables).

Ejercicio 3 [25 puntos]

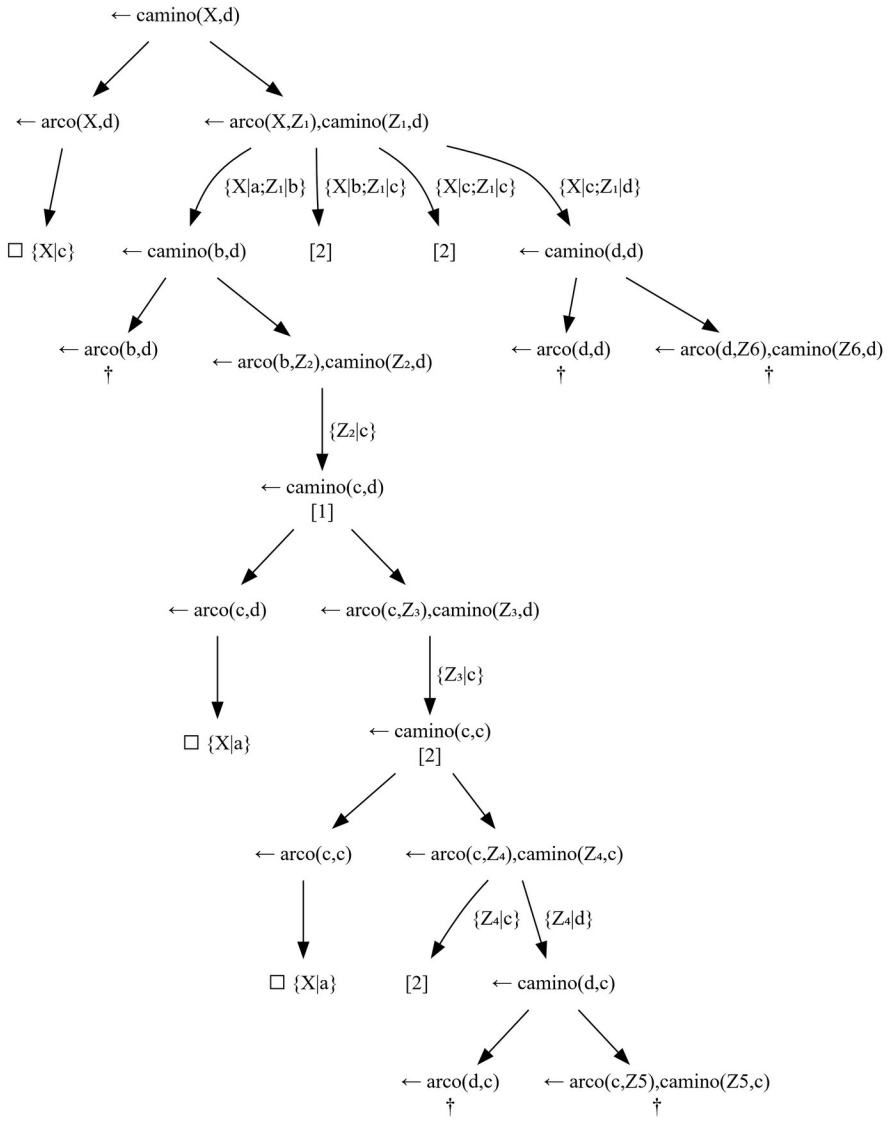
Considere el siguiente programa Prolog:

```
arco(a,b).
arco(b,c).
arco(c,c).
arco(c,d).

camino(X,Y) :- arco(X,Y).
camino(X,Y) :- arco(X,Z),camino(Z,Y).
```

Considere el árbol SLD construido a partir del objetivo ← `camino(X,d)`, utilizando la regla de computación que selecciona siempre el átomo de más a la izquierda.

El árbol es el siguiente (no se pide dibujarlo):



a) Indique las soluciones que se obtienen si se selecciona primero la rama correspondiente a la regla que aparece en primer lugar. Indique los objetivos para cada uno de los nodos de la rama que lleva a las dos primeras soluciones. Justifique brevemente.

Las soluciones son:

X=c;
X=a;
X=a;
...
sigue infininitamente encontrando soluciones X=a

Los objetivos para la primera solución son:

← camino(X,d)
← arco(X,d)
□ {X|c}"

Para la segunda son:

← camino(X,d)
← arco(X,Z_1),camino(Z_1,d)
← camino(b,d)
← arco(b,Z_2),camino(Z_2,d)
← camino(c,d)
← arco(c,d)
□ {X|a}"

Al recorrer el árbol, encontramos las dos ramas recién descritas, y luego buscando un camino entre c y d encontramos un camino entre c y c, que nos devuelve una solución X=c y luego vuelve a repetir, debido al lazo en el grafo.

b) Se entraría en una rama infinita, sin devolver soluciones.

c) Véase el teórico

d) Se obtiene solamente la solución X=c.

e) Se obtendrían las mismas soluciones que en la parte a)

e) Se obtienen las soluciones {X=c,X=a}

Ejercicio 4 [15 puntos]

a) Implemente el siguiente predicado sobre listas de diferencias en Prolog:

not_member_ld(+X,?L) ← X es un elemento que no está presente en la lista de diferencias L utilizada con la notación L-LR. Por ejemplo:

not_member_ld(6,[1,2,3,4|LR]-LR). ← Devuelve “true”.

not_member_ld(4,[1,2,3,4|LR]-LR). ← Devuelve “false”.

b) Utilizando DCG, define un programa Prolog para reconocer el lenguaje:
L = {a^n $b^{n\hat{}2}$ / n ≥ 0}

Solución

a)

```
not_member_ld(_,L-L):-var(L),!.
not_member_ld(X,[Y|L]-LR):-
    X \= Y,
    not_member_ld(X,L-LR).
```

b)

```
l --> a(N),{N2 is N*N},b(N2).
a(0) --> [].
a(N) --> [a],a(N1),{N is N1 + 1}.
b(0) --> [].
b(N) --> [b],b(N1),{N is N1 + 1}.
```