Universidad Nacional de Costa Rica (UNA) .
Escuela de Informática. Curso EIF400. Paradigmas de Programación II-2017
Dr. Carlos Loría-Sáenz

PREGUNTAS

1) Enumere 2 funcionalidades nativas (propias) centrales y Prolog que no están disponibles como nativas (propias) en un lenguaje como Java. Explique ventajas y desventajas de tales funcionalidades con relación a Java. Sea claro y conciso.

2) Demuestre que `[X=a, Y=h(b)]` es el resultado de unificar `t1=f(g(X,a)`, h(b)) con `t2=f(g(a,a), Y)`? Hágalo paso a paso con el método visto en clase. Verifique su respuesta en Prolog

3) Resuelva en Prolog cualquiera de los ejercicios de programación `FP` de la práctica del Profesor Jose A. Jiménez.

4) Considere la siguiente estructura de datos para árboles de expresiones `{functor:"…", args:[…]}` y `{var:"…"}` para variables y `{functor:"…", args:[]}` para átomos. Escriba JS reglas funciones que calculen el unificador entre dos árboles t1, y t2 así modelados retornando en un objeto de la forma `{mgu:[….], exists:..}` donde mgu es la lista de bindings de la unificación y exists un booleano que es true si hay mgu y false en otro caso

5) Explique qué es el problema "*occur check*" del algoritmo de unificación.

6) Representando un árbol como un AST de Prolog `tree(info, left, right)` y el átomo `null` para árbol vacío escriba `bb(K+, +T, -P)` que dado un árbol binario de búsqueda `T` y una llave `K` retorne en la lista `P` la ruta (como lista de unos y ceros) que lleva desde la raíz de `T` hasta `K` en el árbol si `K` está en `T` siendo 0 izquierda y 1 derecha. `P` sería `[]` si no está.
Por ejemplo

```
:-bb(6, tree(10,
                tree(7,
                     tree(5,null,null),
                     tree(8,
                          tree(6, null, null))),
                tree(20, null, null)), P).
P=[0, 1, 0]
```

7) Usando la misma estructura `tree` anterior escriba un predicado `pretty_bb(+T)` que imprima el árbol T en forma indentada (es decir un "beautifier").

8) Escriba un predicado en Prolog `ocurre(+X, +E, -Z)` retorne en `Z` el número de veces que un átomo `X` ocurre en una expresión (AST) `E` de Prolog. Asuma sólo expresiones aritméticas.
Por ejemplo

```
:- ocurre(x, y+x*x)/2*x, Z) %Z=3 (x ocurre 3 veces).
:- ocurre(y, y+x*x)/2*x, Z) %Z=1  (y ocurre 1 vez).
:- ocurre(z, y+x*x)/2*x, Z) %Z= 0 (z ocurre 0 veces).
```

9) ¿Qué hace el siguiente predicado `foo(+L, +N)`? Resuelva sin usar Prolog, trate de hacerlo sin correrlo.

```
foo(L, N) :- member(X, L), X=N, write(X), nl, fail.
```

10) Escriba un predicado `solve(+N, -S)` que en `S` retorne todas las soluciones `x`, `y` de la ecuación `x+y=N` donde `N` es un entero no negativo. No use recursión use backtracking. Ejemplo de uso:

```
:-solve(3, S)
S=0+3
S=1+2
S=2+1
S=3+0
```

Haga `solveList(+N, -L)` que recolecte en L todas esas soluciones.

11) Dibuje el FA (autómata finito) asociado a la siguiente RE (expresión regular) : (010)*|(101)+.

12) Modelemos un FA como "facts"

`fa_start(FA, S)`: `S` es el estado de arranque del `FA`.
`fa_final(FA, S)`: `S` es estado final del `FA`.
`fa_move(FA, S1, S2, X)`: dado el input `I` estando en estado `S1` el `FA` se mueve a estado viendo símbolo `X`.

Modelamos el input I como un átomo (que se convierte en una lista de símbolos) la respuesta que da el FA con un átomo que puede ser '`acept`' ó '`reject`'.

El siguiente programa Prolog que lo corre:

```
fa_init(FA, IA, R) :- fa_start(FA, S),
                      atomic_list_concat(I, split, IA),
                      fa_run(FA, S, I, R).

fa_run(FA, SF, [], accept) :- fa_final(FA, SF), !.
fa_run(FA, S1, [X|I], R) :- fa_move(FA, S1, S2, X),
                            fa_run(FA, S2, I, R), !.
fa_run(_, _, _, reject).
```

Modele el FA asociado con la `RE 1(00)*1`. Pruebe que funciona, por ejemplo

```
:- fa_init(fa_1dos01, 10000001, R).
R=accept

:- fa_init(fa_1dos01, 1000001, R).
R=reject
```