Universidad Nacional Autónoma de México
Facultad de Ciencias
Lenguajes de Programación

Tarea 04

Karla Ramírez Pulido karla@ciencias.unam.mx
J. Ricardo Rodríguez Abreu ricardo_rodab@ciencias.unam.mx
Manuel Soto Romero manu@ciencias.unam.mx

Fecha de inicio: 14 de mayo
Fecha de entrega: 21 de mayo
Semestre 2018-2

1. Responder correctamente cada inciso

1.1 (3 ptos) Demostrar formalmente la complejidad en tiempo y espacio del Algoritmo de Unificación visto en clase.

1.2 (2 ptos) Utilizar el algoritmo de unificación paso a paso visto en clase para cada una de las siguientes expresiones:

```
( (lambda(x)
    (* x (+ 2 x)))
  (/ 10 (* y 0)) )
```

Expresión 1.1

```
( (lambda(y) (ncons 1 y))
  (ncons x nempty) )
```

Expresión 1.2

1.3 (2 ptos) Leer cuidadosamente y resolver el siguiente ejercicio:

El programa en el Cuadro 1.1 contiene un error asociado a los tipos. Se deberá aislar el conjunto más pequeño de restricciones, identificando el error de tipo.

```
(+ 1 (first (cons true empty)))
```

Cuadro 1.1

1.4 (3 ptos) Usando el intérprete de `Prolog SWI PROLOG` realizar el siguiente ejercicio.

1. Implementar un intéprete del lenguaje de programación `WAE` visto al principio del semestre. Se propone una gramática ligeramente diferente a `WAE` la cual llamaremos `PWAE` para usar listas, hechos y predicados en PROLOG.

<WAE> ::= <num>

| {+ <WAE> <WAE>}

| {- <WAE> <WAE>}

| {* <WAE> <WAE>}

| {/ <WAE> <WAE>}

| <id>

| {with {<id> <WAE>} <WAE>}

<PWAE> ::= <num>

| [+, <PWAE>, <PWAE>]

| [-, <PWAE>, <PWAE>]

| [*, <PWAE>, <PWAE>]

| [/, <PWAE>, <PWAE>]

| [id, <id>]

| [with, <id>, <PWAE>, <PWAE>]

- Se recomienda usar el esqueleto del Cuadro 1.2, sin embargo pueden implementar su propias funciones. Ya que sólo se evaluará que todas las pruebas unitarias regresen el valor de `true`.

```
/* Facultad de Ciencias UNAM - Lenguajes de programacion 2018-2
Profesora: Karla Ramirez Pulido
Ayudante: Jose Ricardo Rodriguez Abreu
Ayudante de laboratorio: Manuel Soto Rometo
Alumno: --NOMBRE--*/

:- dynamic wae/3.
% Nos ayuda a agregar convenientemente a una lista ambiente.
agrega_env(ASUB,ENV,[ASUB|ENV]).

% Revisar igualdad y es mejor que comparar con "==".
igualdad(X,X).

% Sobrecargamos el predicado number(X).
num(X) :- number(X).

% Manejo de error si llegamos a encontrar un error de sintaxis.
error(_) :- writeln("Error de sintaxis"),false.
% Manejo de error si buscamos un id que no existe en el ambiente.
error_id(_) :- writeln("El id no existe"),false.

% Son las operaciones que debes implementar:
% X :- puede ser un objeto tan complejo como WAE lo permita.
% Y :- puede ser un objeto tan complejo como WAE lo permita.
% ENV :- Es la lista que usamos como ambiente.
% R :- Es el resultado de la operacion:
sum(X,Y,ENV,R) :- false. %Implementar suma.
res(X,Y,ENV,R) :- false. %Implementar resta.
prod(X,Y,ENV,R) :- false. %Implementar producto.
div(X,Y,ENV,R) :- false. %Implementar division.

% En este predicado revisamos que tipo de operacion es.
% Ejemplo: Si es operacion([+, 2 3], ENV, R) -> sum(2, 3, ENV, R).
operacion(L,ENV,R) :- false. %Implementar.

% Necesitamos un caso base para cuando el ambiente sea vacio.
lookup(_,[],_) :- false. %Implementar error.
% Buscamos en el ambiente que es una lista de tupletas.
lookup(ID,[X|XS],VAL) :- false. %Implementar la busqueda.

% En este predicado revisamos que la sintais de (id x) sea
% la correcta y buscamos el valor de la variable.
id(I,ENV,R) :- false. %Implementar.

% Con el predicado with aumentamos el ambiente.
with(L,ENV,R) :- false. %Implementar.

% En lugar de interp yo le llame wae a mi predicado que manda
% a llamar a cada una de las posibles evaluaciones de wae.
% esta es el predicado principal:
wae(X,ENV,R) :- flase. %Implementar.

% PRUEBAS QUE DEBE PASAR SU IMPLEMENTACION:
% NOTA: Al calificar ademas de estas pruebas usare otras para asegurar
% el buen funcionamiento de sus predicados.
prueba0() :- wae(1729,[],1729).
prueba1() :- wae(3.1415926535,[x|2],3.1415926535).
prueba2() :- wae([+,0,0],[],0).
prueba3() :- wae([+,25,75],[a|3],100).
prueba4() :- wae([-,65,98],[],-33).
prueba5() :- wae([*,[+,3,7],[-,[/,121,11],1]],[],100).
prueba6() :- wae([with,a,20,-1],[],-1).
prueba7() :- wae([with,lenguajes_rules,10,[-,[id, lenguajes_rules],10]],[],0).
prueba8() :- wae([with,z,1729,[/,[id,z],[id,y]]],[[y|-1729]],0).
prueba9() :- wae([with,x,10,[with,y,5,[+,[id,x],[id,y]]]],[],15).
prueba10():- wae([with,x,10,
                    [with,y,5,
                      [with,z,2,
                    [+,[*,[id,a],[id,x]],[-,[id,z],[id,y]]]]]],[[b|3]|[[a|2]]],17).
```

Cuadro 1.2

Bibliografía

[1] Shriram Krishnamurthi, David Tucker, Programming Languages: Application and Interpretation, Primavera-2017 Universidad de Brown, http://cs.brown.edu/courses/cs173/2012/book/book.pdf .