75.31 Teoría de Lenguaje
75.24 Teoría de la Programación

Primer Cuatrimestre de 2017

1. Procesamiento de listas

Desarrollar las siguientes funciones de manejo de listas

1. {Length Xs} devuelve el tamaño de la lista Xs.

2. {Take Xs N} Devuelve una lista que contiene los primeros N elementos de la lista Xs. Si el tamaño de Xs es menor que N, devuelve Xs.

3. {Drop Xs N} Devuelve la lista Xs pero sin los primeros N elementos. Si el tamaño de Xs es menor que N, devuelve nil.

4. {Append Xs Ys} Devuelve la lista Xs junto con los elemenos de Ys agregados.

5. {Member Xs Y} Prueba si Y es un elemento de Xs. Devuelve true o false.

6. {Position Xs Y} Devuelve la posición de Y dentro de la lista Xs. Se puede asumir que Y ya es un elemento de la lista. Por ejemplo, {Position [a b c] b} devuelve 2.

2. Referencias externas

Para cada una de las siguientes definiciones de procedimiento, liste las referencias externas

```
1. proc {P X Y} local Z in {Q Z U} end end
2. proc {P X Y} local Z in {Q Z Y} end end
3. proc {P X Y} local Z in {P Z Y} end end
```

3. Ejemplo de ejecución:

Ejecute el siguiente programa a mano usando la máquina abstracta vista en clase, y mostrando en cada paso el estado del stack y del store:

1)

```
local B in if B then skip else skip end end
```

2)

```
local B in
   B = false
   if B then
      skip
   else
      skip
   end
end
```

3)

```
local X Z A B P in
   proc {P X Y}
      Y = X+Z
   end
   Z=7
   X=4
   {P X A}
   {P A B}
end
```

4)

```
local X Z A B P in
   proc {P X Y}
      Y = X+Z
   end
   Z=10
   local Z in
       Z = 2
       X=4
       {P X A}
       {P A B}
    end
end
```

5)

```
local X Y Z P Q in
    X=6
    Y=4
    proc {P A B}
```

```
        proc {B U V}
            local F in
              F=A+1
              V=U+F
            end
        end
    end
    {P X Q}
    {Q Y Z}
end
```

6)

```
local X Y Z in
    X = Y
    try
        X = 1 Y = 2 Z = 3
    catch Exception then
        skip
    end
    {Browse X#Y#Z}
end
```

7)

```
local X Z A B P in
   proc {P X Y}
      Y = X+Z
   end
   Z=7
   local Z in
     Z = 2
     X=4
     {P X A}
     {P A B}
  end
end
```

4. Case

Si ejecutar el siguiente código, predecir que pasará. Utilizar la traducción al lenguaje Kernel y la semántica si es necesario. Luego ejecutar y comparar. Entregar el análisis

realizado.

```
local Test in
    proc {Test X}
        case X
        of a|Z then {Browse 'case'(1)}
        [] f(a) then {Browse 'case'(2)}
        [] Y|Z andthen Y==Z then {Browse 'case'(3)}
        [] Y|Z then {Browse 'case'(4)}
        [] f(Y) then {Browse 'case'(5)}
        else {Browse 'case'(6)} end
    end
    {Test [b c a]}
    {Test f(b(3))}
    {Test f(a)}
    {Test f(a(3))}
    {Test f(d)}
    {Test [a b c]}
    {Test [c a b]}
    {Test a|a}
    {Test '|'(v b)}
    {Test '|'(a a)}
    {Test '|'(b b)}
    {Test '|'(a b c)}
    {Test '|'(a [b c]}
end
```

5. Recursividad

Dada la siguiente función

```
fun {Length Xs}
    case Xs
        of nil then 0
        [] X|Xr then 1+{Length Xr}
    end
end
```

:

1)Traducirla al lenguaje Kernel visto en clase.

2)Reescribirla para que se ejecute como "tail-recursive". Explicar que ventaja tiene.

3)Mostrar la ejecución en máquina abstracta de una ejecución {Length [1 2 3 4]} en ambos casos

6. Alto orden con listas

6.1. Fold

Las funciones de tipo Fold son unas de las funciones de alto orden más comunes. Generalmente toman una lista (o secuencia), una funcion que recibe 2 elementos y un valor inicial. Se combina la secuencia aplicando la función recibida elemento por elemento. EL fold puede ser asociativo a izquieda o a derecha

La función {FoldL L F U} realiza lo siguiente:{F ... {F {F {F U X1} X2} X3} ... Xn}

La función {FoldR L F U} realiza lo siguiente:{F X1 {F X2 {F X3 ... {F Xn U} ...}}}

Desarrollar las funciones FoldR y FoldL y utilizarlas en un ejemplo.

6.2. Map

Una operación común sobre listas es Map, calcula una nueva lista aplicando una función a cada elemento de la lista de origen.

Por ejemplo,

```
{Map [1 2 3] fun {$ I} I*I end}
```

devuelve

```
[1 4 9].
```

El tipo es <fun {$ <List T> <fun {$ T}: U>}: <List U>>. Desarrollar la función Map y utilizarla en un ejemplo

6.3. Filter

Otra operación es Filter, que aplica una función booleana a cada elemento de la lista y la salida es otra lista con los elementos en los cuales la función dio verdadero.

Por ejemplo

```
{Filter [1 2 3 4] fun {$ A} A<3 end}
```

devuelve

```
[1 2]
```

El tipo es <fun {$ <List T > <fun {$ T T}: <bool>>}: <List T >>. Desarrollar la función Filter y utilizarla en un ejemplo

7. Currying

Explicar el concepto de Currying, dar un ejemplo.

8. Hilos

8.1. Wait

Un problema que ocurre con frecuencia en la práctica es tener que esperar hasta que al menos una de dos variables se ligue. Para esto, OZ provee el procedimiento {WaitOr X Y}, que suspende la ejecución del hilo que lo llama hasta que X o Y se liguen. Escribir un procedimiento {WaitSome Xs}, que suspenda la ejecución del hilo que lo llamó hasta que al menos una de las variables de la lista Xs se ligue.

Mostrar con un ejemplo su utilización.

8.2. Máquina abstracta

Realizar la ejecución del sisguiente programa en la máquina abstracta. Lo importante es entender como los threads se crean y se ejecutan

```
local A B C in
thread if A then B=true else B=false end end
thread if B then C=false else C=true end end
A=false
end
```

9. Evaluación perezosa

9.1. Máquina abstracta

Traducir a lenguaje Kernel y ejecutar en máquina abstracta el siguiente ejemplo

```
local Ints L A B C in

    fun lazy {Ints N}
        N|{Ints N+1}
    end
    L = {Ints 1}
    A = L.2.2.1
    B = L.1
    C = A+B
    {Browse C}

end
```

9.2. Reverse

Considere las siguientes definiciones de una funcion para revertir una lista

```
fun lazy {Reverse1 S}
```

```
    fun {Rev S R}
        case S of nil then R
        [] X|S2 then {Rev S2 X|R} end
    end
in {Rev S nil} end
fun lazy {Reverse2 S}
    fun lazy {Rev S R}
        case S of nil then R
        [] X|S2 then {Rev S2 X|R} end
    end
in {Rev S nil} end
```

¿Cúal es la diferencia en comportamiento entre {Reverse1 [a b c]} y {Reverse2 [a b c]}? ¿Ambos llamados devuelven el mismo resultado? Explique en cada caso.

10. Mensajes

Escribir un agente que muestre en el *Browser* cada uno de los mensajes recibidos.

11. Servidor de filtros

Escribir una agente que reciba un mensaje en un puerto, verifique si el mensaje es correcto y lo envie a otro agente. La verificación la debe realizar con una funcion unaria.

12. Celdas de memoria

¿Qué mostrará el siguiente programa? ¿Por qué?

```
declare
X={NewCell 0}
{Assign X 5}
Y=X
{Assign Y 10}
{Browse {Access X}==10}
{Browse X==Y}
Z={NewCell 10}
{Browse Z==Y}
{Browse @X==@Z}
```

CONDICIONES GENERALES

- El trabajo es individual.
- La fecha de entrega es a más tardar, el 12/06/2017. Se debe enviar el informe con las respuestas y los códigos fuentes a la dirección de correo: leanrafa+tdl20171c@gmail.com (SI, es con el "+")