Teoría de lenguajes

Resumen y apuntes
son personales con posibles errores*

Índice

1. Primera clase

Modelo computacional: sistema formal que define cémo se ejecutan los cálculos. Se define en términos de los conceptos que incluye.

Modelo imperativo: programación orientada a objetos, estructural... Te dicen como hacer las cosas, las seguis y el programa sale funcionando. Ejemplo: dar los pasos a seguir para hacer una torta.

Modelo declarativo: programación funcional, lógica... Una receta en declarativo es inseguible, es como una descripción del problema sin dar los pasos a seguir. Ejemplo ver una torta y describir que es lo que tiene y como se hizo.

Oz

Las **variables** viven solo en los scopes propios, no son literalmente como una variable de C, toman el valor que se les asigna en el momento. Son como un alias que toman los valores en ese scope. Un mismo alias puede en cada scope apuntar a cosas diferentes.
Las minúsculas son átomos y las mayúsculas son variables. La palabra **Funcion** se debe usuar para variable y la palabra **funcion** se podría por ejemplo usar como label de un árbol: funcion(key:.. value:.. left:.. right:..). También se usa para las caracteristicas (features).

Creación de variable global:

```
declare A
    A = 1
end
```

Creación de variable en scope:

```
local A in
    A = 1
end
```

Las variables tambien pueden ser alias de funciones:

```
local A Funcion in
    A = fun {Funcion B} % B es el parametro que recibe
        % cuerpo de funcion
    end
end
```

Redeclaración de variables:

```
local A C D F in
    C = 10
    D = 20
    F = C + D
    local F in      % en un nuevo scope se puede redefinir
        F = 100
        {Browse F}  % muestra 100 en pantalla
    end
    A = 5
    {Browse F+A} % muestra (C + D) + A
end
```

Doble dirección de igualdad:

```
local A B in
    A = 1
    2 = B
    {Browse A+B} % muestra 3 en pantalla
end
```

Función:

La función se termina creando como una variable más, porque es una variable más

```
local Mayor A B M in
    fun {Mayor X Y}
        if (X > Y) then X else Y end
    end

    A = 30
    B = 20
    M = {Mayor A B}
    {Browse M}
end
```

Otra manera de hacer esto

```
local Mayor A B M in
    Mayor = fun { X Y}
        if (X > Y) then X else Y end
    end

    A = 30
    B = 20
    M = {Mayor A B}
    {Browse M}
end
```

Los nombres de variables no tienen nada que ver con los scopes de funciones:

```
local A B F in
    F = fun {$ B A}
        B-A
    end

    A = 10
    B = 9
    {Browse {F A B}}    % imprime 1 (10 - 9)
end
```

Records: estructuras de datos para agrupar referencias.

```
    tree(key:I value:Y left:LT right:RT)
% etiqueta     caracteristicas
```

```
local Arbol in
    Arbol = miArbol(key:1 value:8 left:nil right:nil)

    {Browse Arbol}
    % imprime:
    % miArbol{
    %   key:1
    %   left:nil
    %   right:nil
    %   value:8}
```

```
11
12      {Browse Arbol.value}
13      % 8
```

Cómo cambia el valor dentro de un record:

```
1  local Alumno E in
2      Alumno = persona(nombre:'Roberto' apellido:'Sanchez' edad:E)
3
4      Alumno.edad = 78     % SI se puede hacer porque su valor es variable
5
6      Alumno.nombre = 'Diego'      % NO se puede hacer
7  end
```

Acceso a secciones de un record:

```
1  local Alumno E in
2      Alumno = persona(nombre:'Juan' apellido:'Perez' edad:E)
3
4                                   % en pantalla:
5      {Browse {Label Alumno}}      % persona
6      {Browse {Width Alumno}}      % 3
7      {Browse {Arity Alumno}}      % apellido edad nombre
8
9      % si Alumno no tuviese nombre en los features:
10     % Arity -> [1 2 3]
```

Tuplas: una tupla es un Record en el cual sus características son enteros empezando por el 1. No llevan nombres los features.

Binding: Asignarle valor a una variable.

Partial Value: estructura de datos que puede tener unbound variables.

Variable - Variable binding: cuando una variable se liga a otra variable.

Ejemplo de tuplas:

```
1  local X Y B U in
2                       % X es tupla
3      X = f(a B)       % f es label, B es una variable no ligada, a es atomo
4      Y = f(U v)       % f es label, U es una variable no ligada, v es atomo
5      X = Y
6      {Browse X}       % Lo que va a pasar es que se van a
7      {Browse Y}       % cargar las variables no ligadas
8  end                  % U y B, como U=a y B=v.
9
10     % si eran f(a B) y f2(U v) no compilaba porque son incompatibles, si los labels
        son diferentes no matchea.
```

Listas: una lista es una tupla de dos elementos, uno el primer elemento y el otro el resto de la lista. Se construye con [], |, ó '|'().

```
1  local L L2 L3 in
2      L = [20 340 132 5 132]
3      {Browse L}       % muestra todos los elementos
4      {Browse L.5}     % no compila, el feature 5 no existe, solo el 1 y el 2 (head y
       tale)
5      {Browse L.2}     % muestra todos los elementos despues del primero
6      {Browse L.2.2}   % muestra 132 5 132
7  end
```

```
8
9       % el funcionamiento para acceder es:
10      % siempre que sea .2 se va a mostrar todos los elementos de la tale
11      % siempre que sea .1 se toma el elemento del head
```

Pattern Matching: es una manera de acceder a los campos de una estructura de datos y obtener los valores.
Un patrón matchea sobre un record cuando coincide: cantidad de elementos, etiqueta y caracteristicas.

```
local Alumno E Alumno2 EdadDeAlumno in
    EdadDeAlumno = fun {$ Alumno}
        case Alumno of persona(nombre:N apellido:A edad:E) then
            E
        else
            0
        end
    end
    Alumno = persona(nombre:'Juan' apellido:'Perez' edad:E)
    Alumno.edad = 78

    % si el label no es "persona" no matchea

    {Browse {EdadDeAlumno Alumno}}      % muestra 78
    {Browse {EdadDeAlumno 33}}          % muestra 0
    {Browse Alumno}
end
```

Lo que intenta hacer pattern matching es con una especie de máscara obtener algo de la misma forma, pero sin necesidad de especificar un tipo de dato. En el ejemplo EdadDeAlumno tambien podia ser igual a un nombre. Lo unico que importa es que el case pregunta si lo que llegó tiene esa "forma", de no ser asi, en este caso, solo devuelve 0.

Ejemplo de **recursividad** con lista:

```
local L Largo ProcesarLista in
    ProcesarLista = proc {$ L}
                        case L of H|T then
                            {Browse T}
                            {ProcesarLista T}
                        else
                            {Browse 'Termine'}
                        end
                    end

    L = [30 540 nil 430 50]
    {ProcesarLista L}       % se muestra la tale en cada
                            % iteracion con un elemento menos
end
```