Introducción a ML

José de Jesús Lavalle Martínez
http://aleteya.cs.buap.mx/~jlavalle/

Benemérita Universidad Autónoma de Puebla
Facultad de Ciencias de la Computación
Licenciatura en Ciencias de la Computación
Fundamentos de Lenguajes de Programación
CCOS 255

Otoño 2020

Contenido

Principales características de ML I

- Fue desarrollado por Robin Milner y sus colaboradores en 1970s en la Universidad de Edinburgo, es un acrónimo para **M**eta **L**anguage.

Principales características de ML I

- Es un **lenguaje funcional**, todo cálculo se realiza evaluando una función.

Principales características de ML I

- **Fuertemente tipificado**, toda expresión del lenguaje tiene asociado un tipo y sólo uno.

Principales características de ML I

- Utiliza **polimorfismo paramétrico**, una función o un tipo de datos se pueden escribir de manera genérica, de tal manera que puedan manejar valores sin depender de su tipo. Por ejemplo, una lista tiene las mismas características sin importar si sus elementos son enteros, reales, cadenas o incluso otras estructuras de datos.

Principales características de ML I

- **Maneja excepciones**, se puede controlar en tiempo de ejecución si ocurre un valor incorrecto con el propósito de que el programa no colapse, por ejemplo en una división si el denominador es cero.

Principales características de ML II

- Es un lenguaje **interpretado**.

Principales características de ML II

- **Tipificado estático**, la asignación de un tipo a una expresión del lenguaje se hace en tiempo de interpretación.

Principales características de ML II

- **Inferencia de tipos**, no hace falta especificar el tipo exacto de una expresión, con la evidencia que hay en su entorno se infiere el tipo de dicha expresión.

Principales características de ML II

- **Alto orden**, el dominio y rango de las funciones pueden a su vez ser funciones.

Principales características de ML II

- ***Pattern Matching***, los parámetros reales con los que se evalúa una función tienen que empatar con los patrones especificados como parámetros formales de una función.

Ejecución del intérprete

```
iMac-de-Jose:code jlavallenator$ mosml
Moscow ML version 2.10
Enter `quit();' to quit.
-
```

Definición de funciones I

```
- fun sucesor x = x + 1;
> val sucesor = fn : int -> int
-
```

Definición de funciones II

```
- fun sucesor1 x :int = x + 1;
> val sucesor1 = fn : int -> int
-
```

Definición de funciones III

```
- fun sucesor2 x = x + 1:int;
> val sucesor2 = fn : int -> int
-
```

Definición de funciones IV

```
- fun sucesor3 x :int = x+1:int;
> val sucesor3 = fn : int -> int
-
```

Definición de funciones IV

```
- fun factorial1 0 = 1
|     factorial1 n = n * factorial1 (n-1);
> val factorial1 = fn : int -> int
-
```

Definición de funciones IV

```
- fun factorial1 0 = 1
|     factorial1 n = n * factorial1 (n-1);
> val factorial1 = fn : int -> int
-
```

```
factorial1 3 = 3 * factorial1 2 =
3 * 2 * factorial1 1 = 3 * 2 * 1 * factorial1 0 =
3 * 2 * 1 * 1 = 6 * 1 * 1 = 6 * 1 = 6
```

Definición de funciones IV

```
- fun factorial1 0 = 1
|     factorial1 n = n * factorial1 (n-1);
> val factorial1 = fn : int -> int
-
```

```
factorial1 3 = 3 * factorial1 2 =
3 * 2 * factorial1 1 = 3 * 2 * 1 * factorial1 0 =
3 * 2 * 1 * 1 = 6 * 1 * 1 = 6 * 1 = 6
```

```
- factorial1 3;
> val it = 6 : int
-
```

Definición de funciones V

Sería conveniente no tener que postergar la evalución del factorial hasta llegar al caso base. Es decir, antes de llegar al caso base ir acumulando los resultados parciales:

Definición de funciones V

Sería conveniente no tener que postergar la evalución del factorial hasta llegar al caso base. Es decir, antes de llegar al caso base ir acumulando los resultados parciales:

$$(1 * 3)2 \rightarrow (3 * 2)1 \rightarrow (6 * 1)0 \rightarrow 6\ 0 \rightarrow 6$$

Definición de funciones V

Sería conveniente no tener que postergar la evalución del factorial hasta llegar al caso base. Es decir, antes de llegar al caso base ir acumulando los resultados parciales:

$$(1 * 3)2 \rightarrow (3 * 2)1 \rightarrow (6 * 1)0 \rightarrow 6\ 0 \rightarrow 6$$

Tratemos de encontrar un patrón:

$$(\overbrace{1}^{pa} * \overbrace{3}^{n})\overbrace{2}^{n-1} \rightarrow (\overbrace{3}^{pa} * \overbrace{2}^{n})\overbrace{1}^{n-1} \rightarrow (\overbrace{6}^{pa} * \overbrace{1}^{n})\overbrace{0}^{n-1} \rightarrow \overbrace{6}^{pa}\overbrace{0}^{n} \rightarrow 6$$

Definición de funciones V

Sería conveniente no tener que postergar la evalución del factorial hasta llegar al caso base. Es decir, antes de llegar al caso base ir acumulando los resultados parciales:

$$(1 * 3)2 \rightarrow (3 * 2)1 \rightarrow (6 * 1)0 \rightarrow 6\ 0 \rightarrow 6$$

Tratemos de encontrar un patrón:

$$(\overbrace{1}^{pa} * \overbrace{3}^{n})\overbrace{2}^{n-1} \rightarrow (\overbrace{3}^{pa} * \overbrace{2}^{n})\overbrace{1}^{n-1} \rightarrow (\overbrace{6}^{pa} * \overbrace{1}^{n})\overbrace{0}^{n-1} \rightarrow \overbrace{6}^{pa}\overbrace{0}^{n} \rightarrow 6$$

```
fun fact_pa pa 0 = pa
|   fact_pa pa n = fact_pa (pa * n) (n - 1);
```

Definición de funciones VI

```
- fun fact_pa pa 0 = pa
|     fact_pa pa n = fact_pa (pa * n) (n - 1);
> val fact_pa = fn : int -> int -> int
-
```

Definición de funciones VI

```
- fun fact_pa pa 0 = pa
|     fact_pa pa n = fact_pa (pa * n) (n - 1);
> val fact_pa = fn : int -> int -> int
-

- fact_pa 3;
> val it = fn : int -> int
-
```

Definición de funciones VI

```
- fun fact_pa pa 0 = pa
|     fact_pa pa n = fact_pa (pa * n) (n - 1);
> val fact_pa = fn : int -> int -> int
-

- fact_pa 3;
> val it = fn : int -> int
-

- it 3;
> val it = 18 : int
-
```

Definición de funciones VI

```
- fun fact_pa pa 0 = pa
|     fact_pa pa n = fact_pa (pa * n) (n - 1);
> val fact_pa = fn : int -> int -> int
-

- fact_pa 3;
> val it = fn : int -> int
-

- it 3;
> val it = 18 : int
-

- fact_pa 1 3;
> val it = 6 : int
-
```

Definición de funciones VII

```
- fun factorial x =
    let
        fun fact_pa pa 0 = pa
|           fact_pa pa n = fact_pa (pa * n) (n - 1)
    in
        fact_pa 1 x
    end;
> val factorial = fn : int -> int
```

Definición de funciones VII

```
- fun factorial x =
    let
        fun fact_pa pa 0 = pa
|           fact_pa pa n = fact_pa (pa * n) (n - 1)
    in
        fact_pa 1 x
    end;
> val factorial = fn : int -> int

- factorial;
> val it = fn : int -> int
```

Definición de funciones VII

```
- fun factorial x =
    let
        fun fact_pa pa 0 = pa
|           fact_pa pa n = fact_pa (pa * n) (n - 1)
    in
        fact_pa 1 x
    end;
> val factorial = fn : int -> int

- factorial;
> val it = fn : int -> int

- fact_pa;
! Toplevel input:
! fact_pa;
! ^^^^^^^
! Unbound value identifier: fact_pa
-
```

Definición de funciones VIII

```
- fun shift (x,y) = (y,x);
> val ('a, 'b) shift = fn : 'a * 'b -> 'b * 'a
-
```

Definición de funciones VIII

```
- fun shift (x,y) = (y,x);
> val ('a, 'b) shift = fn : 'a * 'b -> 'b * 'a
-

- shift(4, 5);
> val it = (5, 4) : int * int
-
```

Definición de funciones VIII

```
- fun shift (x,y) = (y,x);
> val ('a, 'b) shift = fn : 'a * 'b -> 'b * 'a
-

- shift(4, 5);
> val it = (5, 4) : int * int
-

- shift([1,2], [4,5]);
> val it = ([4, 5], [1, 2]) : int list * int list
-
```

Definición de funciones IX

```
- fun factorialbs n = if n = 0
                      then 1
                      else n * factorialbs (n-1);
> val factorialbs = fn : int -> int
-
```

Definición de funciones IX

```
- fun factorialbs n = if n = 0
                      then 1
                      else n * factorialbs (n-1);
> val factorialbs = fn : int -> int
-

- factorialbs 20;
> val it = 2432902008176640000 : int
-
```

Definición de funciones X

```
- fun fibo 0 = 1
|     fibo 1 = 1
|     fibo n = fibo(n-1) + fibo(n-2);
> val fibo = fn : int -> int
-
```

Definición de funciones X

```
- fun fibo 0 = 1
|     fibo 1 = 1
|     fibo n = fibo(n-1) + fibo(n-2);
> val fibo = fn : int -> int
-

- fibo 25;
> val it = 121393 : int
- fibo 30;
> val it = 1346269 : int
- fibo 35;
> val it = 14930352 : int
- fibo 40;
> val it = 165580141 : int
- fibo 45;
> val it = 1836311903 : int
-
```

Definición de funciones XI

```
- fun acker(0,y) = y+1
|     acker(x,0) = acker(x-1,1)
|     acker(x,y) = acker(x-1, acker(x,y-1));
> val acker = fn : int * int -> int
- acker(3,10);
> val it = 8189 : int
-
```

Definición de funciones XI

```
- fun acker(0,y) = y+1
|     acker(x,0) = acker(x-1,1)
|     acker(x,y) = acker(x-1, acker(x,y-1));
> val acker = fn : int * int -> int
- acker(3,10);
> val it = 8189 : int
-

- fun ack 0 y = y + 1
|     ack x 0 = ack (x-1) 1
|     ack x y = ack (x-1) (ack x (y-1));
> val ack = fn : int -> int -> int
- ack 3 10;
> val it = 8189 : int
-
```

Definición de funciones XII

```
- fun suma (0,n) = n
|     suma (m,n) = 1 + suma(m-1,n);
> val suma = fn : int * int -> int
```

Definición de funciones XII

```
- fun suma (0,n) = n
|     suma (m,n) = 1 + suma(m-1,n);
> val suma = fn : int * int -> int

- fun sumapa(0,n, pa) = pa + n
|     sumapa(m,n, pa) = sumapa(m - 1, n, pa + 1);
> val sumapa = fn : int * int * int -> int
-
```

Definición de funciones XII

```
- fun suma (0,n) = n
|     suma (m,n) = 1 + suma(m-1,n);
> val suma = fn : int * int -> int

- fun sumapa(0,n, pa) = pa + n
|     sumapa(m,n, pa) = sumapa(m - 1, n, pa + 1);
> val sumapa = fn : int * int * int -> int
-

- fun sumapa2(0,n) = n
|     sumapa2(m,n) = sumapa2(m-1,n+1);
> val sumapa2 = fn : int * int -> int
-
```

Ejercicios

1. Pruebe en ML todo el código de esta sesión.
2. Escriba una función recursiva en ML que realice lo siguiente $\sum_{i=1}^{n} i$.
3. Escriba la versión recursiva con parámetro acumulante para el ejercicio anterior.

Asesoría I

$$sum(n) = \begin{cases} 1 & \text{cuando } n = 1 \\ n + sum(n-1) & \text{cuando } n > 1 \end{cases}$$

$$sum'(n) = \begin{cases} 0 & \text{cuando } n = 0 \\ n + sum'(n-1) & \text{cuando } n > 0 \end{cases}$$

```
sum(4) = 4 + (sum(3)) = 4 + (3 + (sum(2))) =
4 + (3 + (2 + (sum(1)))) = 4 + (3 + (2 + (1))) =
4 + (3 + 3) = 4 + 6 = 10

sum'(4) = 4 + (sum'(3)) = 4 + (3 + (sum'(2))) =
4 + (3 + (2 + (sum'(1)))) = 4 + (3 + (2 + (1 + sum'(0)))) =
4 + (3 + (2 + (1 + (sum'(0)) = 4 + (3 + (2 + (1 + (0)))) =
4 + (3 + (2 + (1))) = 4 + (3 + (3)) = 4 + (6) = 10
```

Asesoría II

```
- fun suma (0,n) = n
|     suma (m,n) = 1 + suma(m-1,n);
> val suma = fn : int * int -> int
-
```

```
suma(3,5) = 1 + (suma(2,5)) = 1 + (1 + (suma(1,5))) =

1 + (1 + (1 + (suma(0,5)))) = 1 + (1 + (1 + (5))) =

1 + (1 + (6)) = 1 + (7) = 8
```

Asesoría III

```
- fun sumapa(0,n, pa) = pa + n
|     sumapa(m,n, pa) = sumapa(m - 1, n, pa + 1);
> val sumapa = fn : int * int * int -> int
-
```

```
sumapa(3,5,0) = sumapa(2,5,1) =

sumapa(1,5,2) = sumapa(0,5,3) = 8
```

Asesoría IV

```
- fun sumapa2(0,n) = n
|     sumapa2(m,n) = sumapa2(m-1,n+1);
> val sumapa2 = fn : int * int -> int
-
```

```
sumapa2(3,5) = sumapa2(2,6)

sumapa2(1,7) = sumapa2(0,8) = 8
```