BIBLIOGRAFÍA para ML:

- Autores: Jeffrey D. Ullman
  "Elements of ML Programming - ML97 EDITION"
  Editorial: PRENTICE HALL

- A. Wikstron - "Standard ML"

- Lawrence C. Paulson - "ML for the Working Programmer"
  Second edition. - Cambridge University Press 1996, ISBN 0-521-56543-X.

BIBLIOGRAFÍA para ALGORITMOS:

- Autores: Fathi Rabbi, Guy Lapalme
  "Algorithms a functional Programming Approach"
  Editorial: Addisson Wesley

- Autores: Alfred V. Aho, John E. HopCroft y Jeffrey D. Ullman
  "Estructuras de Datos y Algoritmos"
  Editorial: Addisson Wesley

- Sedgewick - "Algorithms in C++"

y por último "la biblia"...

- AUTOR: DONALD E. KNUTH -
  "THE ART OF COMPUTER PROGRAMMING"
  VOL. 1 – BASIC ALGORITHMS
  VOL. 2 – SEMINUMERICAL ALGORITHMS
  VOL. 3 – SORTING AND SEARCHING

ML: Significa "META LENGUAJE"

Historia:
Nace del **DEMOSTRADOR TÁCTICO DE TEOREMAS: LCF.**
En **1979 Luca Cardelli**, describe que puede ser un Lenguaje de propósito general. Escribe el primer compilador y lo rebautiza.
Actualmente el último estándar es el del año 1997.
Las implementaciones que hay son las siguientes:

- POLY ML – es Privado
- STANDARD ML OF NEW JERSEY – es Libre
- MLJ – es libre – genera clases en Lenguaje JAVA
- MLTON – es Libre
- **MOSCOW ML** – es Libre – es el que usaremos

MOSCOW ML está hecho por **Romanenko, Sestoft** de **Rusia.**
Para descargarlo:

- dirigirse a http://www.dina.kvl.dk/~sestoft/mosml.html

Están los fuentes y los binarios ejecutables como se muestra en la sección siguiente:

Moscow ML files

- Version 2.01 for **Unixes**:
  - source files with documentation
- an i386 RPM for Fedora Core 6 Linux, created by Arne Glenstrup, Copenhagen. There is a corresponding spec file and some patches applied prior to RPM build.
- a stand-alone toplevel (mosml) compiled for Linux, created by Peter Szabo, Budapest, Hungary.
- Version 2.01 for **MS Windows 95/98/ME/NT/2000/XP**. Unpack with JustZIPit (free) or Info-Zip (free) or Winzip (shareware):
  - executables with documentation
- installation instructions
- A very convenient self-extracting Moscow ML installer for Windows by Ken Friis Larsen.
- Version 2.00 .Net release 0.9.0 for **Microsoft .Net**:
  - executables and documentation.
- readme and installation instructions.
- Moscow ML .Net-specific documentation in PDF: Manual and some information about the internals.
- Moscow ML .Net was created by Niels Kokholm.
- Mosmake, Henning Makholm's tool to generate Makefiles for recompilation management
- Previous releases of Moscow ML.

NOTA: vigente al 08 de Septiembre de 2007

ML es un Lenguaje:

- FUNCIONAL
- FUERTEMENTE EQUIPADO
- EAGER, es decir, los arreglos son evaluados y los devuelve
- IMPURO

La implementación **MOSCOW ML** proporciona:

- un intérprete: **mosml**
- un compilador. **mosmlc**

Para instalarlo en GNU/Linux, lo más conveniente es:

1) Descargar el paquete binario ejecutable para GNU/Linux desde:
   ftp://ftp.dina.kvl.dk/pub/mosml/linux-mos20bin.tar.gz
   o desde
   ftp://ftp.csd.uu.se/pub/mirror/mosml/linux-mos20bin.tar.gz

2) Descargar las instrucciones de instalación desde:
   ftp://ftp.csd.uu.se/pub/mirror/mosml/install-linux.txt

3) Ubicar el paquete “**linux-mos20bin.tar.gz**” en algún directorio
   Supongamos que es el directorio: **/home/user**

4) Descomprimirlo mediante la órden:
   tar zxvf linux-mos20bin.tar.gz

5) Dirigirse al directorio base de la distribución de **mosml**
   cd /home/user/mosml

6) Al listar el contenido de esa carpeta con:
   ls -F | less
   Obtendrán:
   bin/
   config@
   copyrght/
   doc/
   examples/
   export-LD_LIBRARY_PATH.sh*
   include/
   install.txt
   install.txt.w32
   lib/
   mosml.spec
   README
   readme.w32
   roadmap
   tools/
   utility/

7) Hay que editar los archivos scripts:
   /home/user/mosml/mosml
   /home/user/mosml/mosmlc
   /home/user/mosml/mosmllex

   Para que las variables:
   `**stdlib**'
   `**mosmlbin**'
   Hagan referencia a la ubicación que corresponde, en este caso:
   ...
   stdlib=/home/user/mosml/lib
   mosmlbin=/home/user/mosml/bin
   ...

8) Si se intenta usar librerías que usan enlace dinámico, tales como, **Gdbm**, **Mysql**, **Polygdbm**, **Postgres**, **Regex**, **Socket**, o **Unix** hay que **definir** la **variable de ambiente**, que **en bash** se hace de la siguiente manera:
   export LD_LIBRARY_PATH='/home/user/mosml/lib'
   en cada uno de los 3 archivos scripts anteriores:
   home/user/mosml/mosml
   /home/user/mosml/mosmlc
   /home/user/mosml/mosmllex

9) Para invocar al intéprete: mosml
   cd home/user/mosml/
   ./mosml

10) Para chequear que el enlace dinámico funciona escribir la siguiente función en el intérprete **mosml**:

```
fun myload u =
(load u; print ("\nLoaded " ^ u ^ "\n")) handle Fail _ => ();
app myload ["Gdbm", "Mysql", "Postgres", "Regex", "Socket", "Unix"];
```

11) Para salir de **MOSCOW ML introducir la órden**:
quit();
o presionar la combinación de teclas:
Ctrl + D

Comencemos con alguna práctica básica con **mosml:**

Al introducir en mosml...	**... se obtiene en mosml**
2;	> val it = 2 : int
2 + 3.5;	! Toplevel input: ! 2 + 3.5; ! ^^^ ! Type clash: expression of type ! real ! cannot have type ! int
real(2) + 3.5;	> val it = 5.5 : real
2 + 3;	> val it = 5 : int
"hola";	> val it = "hola" : string
#"A";	> val it = #"A" : char
#"\"";	val it = #"\"" : char

Operaciones en ML:

Aritméticas:

+ **suma**

– **resta**

div **división entera**, es decir entre números enteros

/ **división real**, es decir entre números reales

mod **módulo**: resto de la división entera

~ **opuesto de un número**.
Nota: el **ARCO CIRCUNFLEJO** se obtiene mediante la combinación de teclas: **Alt + 126**

Algunos ejemplos:

Al introducir en mosml...	**... se obtiene en mosml**
1 - 3;	> val it = ~2 : int
4 div 2;	> val it = 2 : int
5 div 2;	> val it = 2 : int
1 div 2;	> val it = 0 : int
4 / 2 ;	! Toplevel input: ! 4 / 2 ; ! ^ ! Type clash: expression of type ! int ! cannot have type ! real
4.0 / 2.0 ;	> val it = 2.0 : real
2.0 / 4.0 ;	> val it = 0.5 : real
~4.0 / 3.0 ;	> val it = ~1.33333333333 : real
4 mod 2;	> val it = 0 : int
4.0 mod 2.0 ;	! Toplevel input: ! 4.0 mod 2.0 ; ! ^^^ ! Overloaded mod cannot be applied to argument(s) of type real

Operaciones de comparación:

< **... menor estricto que ...**

<= **... menor o igual que ...**

> **... mayor estricto que ...**

> **... mayor estricto que ...**

>= **... mayor o igual que ...**

= **... igual que ...**

<> **... distinto que ...**

Operaciones Lógicas:

not **negación** lógica.

andelse **AND** lógico**.** Similar a **&** del Lenguaje C.

orelse **OR** lógico. Similar a **|** del Lenguaje C.

Sentencia de comparación if – then – else:

A diferencia del **Lenguaje C**, en **mosml** un **if** siempre debe ir acompañado por el **then** y a continuación **else**

Algunos ejemplos:

Al introducir en mosml...	**... se obtiene en mosml**
1 + (**if** 3 > 4 **then** 10 **else** 11);	> val it = 12 : int

Se lee:
" El valor de esto es 12 de tipo int"

Operaciones de modificación de valores mediante:

trunc **truncamiento del valor de** ...
Elimina cualquier cifra posterior a la coma (que es el punto)

round **redondeo del valor de** ...
Redondea cualquier valor real, de acuerdo al criterio:
SI cifras_decimales **>=** [x]**.**5 **ENTONCES** [x+1]
SINO SI cifras_decimales **<** [x].5 **ENTONCES** [x]

floor **siguiente valor entero menor del valor de ...**
"El piso siempre está abajo"

ceil **siguiente valor entero mayor del valor de ...**
"El cielo siempre está arriba"

Algunos ejemplos de estas operaciones:

Al introducir en mosml...	**... se obtiene en mosml**
trunc 1.4;	> val it = 1 : int
trunc ~1.4;	> val it = ~1 : int
trunc 1.7;	> val it = 1 : int
trunc ~1.7;	> val it = ~1 : int
trunc 1.5;	> val it = 1 : int
trunc ~1.5;	> val it = ~1 : int
round 1.4;	> val it = 1 : int
round ~1.4;	> val it = ~1 : int
round 1.7;	> val it = 2 : int
round ~1.7;	> val it = ~2 : int
round 1.5;	> val it = 2 : int
round ~1.5;	> val it = ~2 : int

Al introducir en mosml...	**... se obtiene en mosml**
floor 1.4;	> val it = 1 : int
floor ~1.4;	> val it = ~2 : int
floor 1.7;	> val it = 1 : int
floor ~1.7;	> val it = ~2 : int
floor 1.5;	> val it = 1 : int
floor ~1.5;	> val it = ~2 : int
ceil 1.4;	> val it = 2 : int
ceil ~1.4;	> val it = ~1 : int
ceil 1.5;	> val it = 2 : int
ceil ~1.5;	> val it = ~1 : int
ceil 1.7;	> val it = 2 : int
ceil ~1.7;	> val it = ~1 : int

Operaciones de conversión entre distintos tipos de datos:

Algunos ejemplos:

Al introducir en mosml...	**... se obtiene en mosml**
real 3;	> val it = 3.0 : real
int 4.5;	! Toplevel input: ! int 4.5; ! ^^^ ! Unbound value identifier: int
integer 4.5;	! Toplevel input: ! integer 4.5; ! ^^^^^^^ ! Unbound value identifier: integer
string 3;	! Toplevel input: ! string 3; ! ^^^^^^ ! Unbound value identifier: string
double 2;	! Toplevel input: ! double 2; ! ^^^^^^ ! Unbound value identifier: double

Conversión de caracter a entero:

ord entrega el código ASCII de ...

Al introducir en mosml...	**... se obtiene en mosml**
ord(#"A");	> val it = 65 : int
ord(#"c");	> val it = 99 : int

Conversión de entero a caracter :

chr entrega el caracter asociado al valor de código ASCII dado

Al introducir en mosml...	**... se obtiene en mosml**
chr 65;	> val it = #"A" : char
chr 99;	> val it = #"c" : char
chr 1902;	! Uncaught exception: ! Chr

Conversión de caracter a cadena de caracteres:

str entrega una cadena de caracteres en base a un caracter

Al introducir en mosml...	**... se obtiene en mosml**
str #"A";	> val it = "A" : string
str #"z" ;	> val it = "z" : string
str #"1";	> val it = "1" : string

Operaciones que se pueden hacer con cadenas de caracteres:

... ^ ... **concatena** la **cadena de caracteres** que está **a su izquierda con la** cadena de caracteres **que está a su derecha** y **entrega** la **cadena de caracteres resultante**.

Algunos ejemplos:

Al introducir en mosml...	**... se obtiene en mosml**
"Ho" ^ "la" ;	> val it = "Hola" : string
"H" ^ "abía " ^ "una " ^ "vez..." ;	> val it = "Había una vez..." : string
"MOSCOW " ^ "ML";	> val it = "MOSCOW ML" : string

size ... **entrega** la **cantidad de caracteres** que tiene la cadena que está a su derecha.

Algunos ejemplos:

Al introducir en mosml...	**... se obtiene en mosml**
size "Hola";	> val it = 4 : int
size "Había una vez... ";	> val it = 18 : int
size "MOSCOW ML";	> val it = 9 : int

Tipo de dato: unit

unit en **mosml**, así como **en cualquier Lenguaje de Programación**, un **TIPO DE DATO** denota un **CONJUNTO DE VALORES.**

Algunos ejemplos:

Al introducir en mosml...	**... se obtiene en mosml**
();	> val it = () : unit
print "Hola mundo! \n";	Hola mundo! > val it = () : unit

Tipos de datos derivados:

Tupla es una **secuencia** de **valores de diversos tipos** **de datos introducidos entre paréntesis** y **separados entre ellos por comas simples**.

Algunos ejemplos:

Al introducir en mosml...	**... se obtiene en mosml**
(1 , 2 , true);	> val it = **(**1**,** 2**,** true**) : int** * **int** * **bool**
(#"A" **,** "e" **,** 2.0 **,** 45 **);**	> val it = **(**#"A"**,** "e"**,** 2.0**,** 45**) : char** * **string** * **real** * **int**
(1 **,** true **,** **(** false **,** 3.5 **)** **,** #"J" **,** "$" **);**	> val it = **(**1**, true, (**false**,** 3.5**),** #"J"**,** "$"**) :** int * bool * **(**bool * real**)** * char * string

Nota:
Aquí el **ASTERIZCO:** * indica que es una **Tupla**

Para extraer un elemento de la Tupla:

#[posición] (... , ... , ...) Extrae el elemento que está en la posición: **[posición]**

Algunos ejemplos:

Al introducir en mosml...	**... se obtiene en mosml**
#1**(** 1 **,** 2 **,** true **);**	> val it = 1 : int
#3**(** #"A" **,** "e" **,** 2.0 **,** 45 **);**	> val it = 2.0 : real
#4 **(** 1 **,** true **,** **(** false **,** 3.5 **)** **,** #"J" **,** "$" **);**	> val it = #"J" : char

Tipo de dato derivado: Registros (del Inglés: Records)

Records Es **similar a una Tupla**, salvo en que **cada valor se lo identifica con un nombre**, es decir, se trata de una **sucesión de pares:**
nombre_del_elemento = valor_del_elemento
introducidos entre llaves y **separados entre ellos mediante comas simples**.

Al introducir en mosml...	**... se obtiene en mosml**
{ nombre **=** "Carlos" **,** edad **=** 23 **,** DNI **=** "24.364.509" **,** ciudad **=** "Santa Fe" **}**;	> val it = {DNI = "24.364.509", ciudad = "Santa Fe", edad = 23, nombre = "Carlos"} : {DNI : string, ciudad : string, edad : int, nombre : string}
{ uno **=** "one" **, {** two **=** 2 **} }** ;	! Toplevel input: ! { uno = "one" , { two = 2 } } ; ! ^ ! Syntax error.
{ l **=** "Lun" **,** t **=** true **,** h **=** 2.0 **,** c **=** #"r" **}**;	> val it = {c = #"r", h = 2.0, l = "Lun", t = true} : {c : char, h : real, l : string, t : bool}

Para extraer un elemento de un Registro:

#[nombre_de_elemento] { ... , ... , ... } Extrae el **valor_de_elemento** que corresponde a **nombre_de_elemento**

Algunos ejemplos:

Al introducir en mosml...	**... se obtiene en mosml**
#3**{** nombre **=** "Carlos" **,** edad **=** 23 **,** DNI **=** "24.364.509" **,** ciudad **=** "Santa Fe" **}**;	! Toplevel input: ! #3{ nombre = "Carlos" , edad = 23 , DNI = "24.364.509" , ciudad = "Santa Fe" }; ! ^^^ ! Type clash: expression of type ! {DNI : 'a, ciudad : 'b, edad : 'c, nombre : 'd} ! cannot have type ! {DNI : 'e, ciudad : 'f, edad : 'g, nombre : 'h, 3 : 'i, ...} ! because record label 3 is missing
#uno**{** uno **=** 1 **,** dos **=** 2.0 **}** ;	> val it = 1 : int
Al introducir en mosml...	**... se obtiene en mosml**
#h{ l = "Lun" , t = true , h = 2.0 , c = #"r" };	> val it = 2.0 : real
#e{ a = #"a" , b = 2 , c = 3.0 , d = "cuatro" , e = true };	> val it = true : bool

Tipo de dato derivado: Lista

Lista es una **secuencia de cero o más valores DE UN MISMO TIPO DE DATO separados entre ellos por comas simples y encerrados entre corchetes**

Algunos ejemplos:

Al introducir en mosml...	**... se obtiene en mosml**
[1 **,** 2 **,** 3 **]**;	> val it = [1, 2, 3] : **int list**
[1 **,** 2 **,** true **]**;	! Toplevel input: ! [1 , 2 , true]; ! ^^^^ ! Type clash: expression of type ! bool ! cannot have type ! int
[#"A" **,** #"B" **,** #"C" **]**;	> val it = [#"A", #"B", #"C"] : **char list**
[1.0 **,** 2.3 **,** 3.4 **, [** 4.0 **]]**;	! Toplevel input: ! [1.0 , 2.3 , 3.4 , [4.0]]; ! ^^^^^ ! Type clash: expression of type ! 'a list ! cannot have type ! real
[1 **, [** 2 **,** 3 **] ,** 4 **]**;	! Toplevel input: ! [1 , [2 , 3] , 4]; ! ^^^^^^^ ! Type clash: expression of type ! 'a list ! cannot have type ! int
[];	> val 'a it = [] : 'a list

Operaciones que se pueden hacer con Listas:

- **Extraer el primer elemento** de la Lista: cabeza (del Inglés: **head**)
  hd[... , ... , ...]

Algunos ejemplos:

Al introducir en mosml...	**... se obtiene en mosml**
hd[1**,**2**,**3**,**4**];**	> val it = **1** : **int**
hd [~1.0 **,** 3.0 **,** ~2.5 **]**;	> val it = **~1.0** : **real**
hd [#"s" **,** #"%" **,** #"&" **,** #"|" **]**;	> val it = **#"s"** : **char**
hd ["ola" **,** "ajo" **,** "luz" **]**;	> val it = **"ola"** : **string**

Operaciones que se pueden hacer con Listas:

- **Extraer todos** los elementos de la Lista **excepto el primero**: cola (del Inglés: **tail**)
  tl [... , ... , ...]

Algunos ejemplos:

Al introducir en mosml...	... se obtiene en mosml
tl [1 **,** 2 **,** 3 **]**;	> val it = **[2, 3]** : **int list**
tl [~1.0 **,** 3.0 **,** ~2.5 **]**;	> val it = **[3.0, ~2.5]** : **real list**
tl [#"s" **,** #"%" **,** #"&" **,** #"|" **]**;	> val it = **[#"%", #"&", #"|"]** : **char list**
tl ["ola" **,** "ajo" **,** "luz" **]**;	> val it = **["ajo", "luz"] : string list**

- **Introducir un elemento al principio de la Lista**:
  Al operador **::** se le llama **CONSTRUCTOR**
  [elemento] :: [... , ... , ...]

Algunos ejemplos:

Al introducir en mosml...	... se obtiene en mosml
2 :: [3 , ~4];	> val it = [2, 3, ~4] : int list
0.5 :: [0.75 , 1.87 , 6.9];	> val it = [0.5, 0.75, 1.87, 6.9] : real list
#"a" :: [#"e" , #"i" , #"o" , #"u"];	> val it = [#"a", #"e", #"i", #"o", #"u"] : char list
"ua" :: ["ue" , "ui" , "uo" , "uu"];	> val it = ["ua", "ue", "ui", "uo", "uu"] : string list

- Una **combinación de estas operaciones** sobre una **Lista** que **no** está **vacía** devuelve esa misma Lista:
  hd[... , ... , ...] :: tl[... , ... , ...]

Algunos ejemplos:

Al introducir en mosml...	... se obtiene en mosml
hd[1 , 2 , 3] :: tl [1 , 2 , 3];	> val it = [1, 2, 3] : int list
hd[0.5 , 1.6 , 9.8] :: tl[0.5 , 1.6 , 9.8];	> val it = [0.5, 1.6, 9.8] : real list
hd[#"a",#"e",#"i"]::tl[#"a",#"e",#"i"];	> val it = [#"a", #"e", #"i"] : char list
hd["el","la","eso"]::tl["el","la","eso"];	> val it = ["el", "la", "eso"] : string list
hd[]::tl[];	! Uncaught exception: ! Empty

Otras operaciones con Listas:

- **Operador de nulidad**: **null**

  null[... , ... , ...] entrega

  · verdadero, **true**, si la **Lista** esta **vacía**

  · falso, **false**, si la lista posee elementos

Algunos ejemplos:

Al introducir en mosml...	**... se obtiene en mosml**
null [ ];	> val it = true : bool
null[1 , 2];	> val it = false : bool
null [#"x" , #"y"];	> val it = false : bool
null["osa" , "uno"];	> val it = false : bool
null[0.1 , 2.3];	> val it = false : bool

- **Operador de concatenación de Listas: @**

  [... , ... , ...] @ [... , ... , ...]

Algunos ejemplos:

Al introducir en mosml...	**... se obtiene en mosml**
[1,2]@[3,4];	> val it = [1, 2, 3, 4] : int list
[1.2,3.4]@[5.6,7.8];	> val it = [1.2, 3.4, 5.6, 7.8] : real list
[#"a",#"b"]@[#"c",#"d"]@[#"e"];	> val it = [#"a", #"b", #"c", #"d", #"e"] : char list
[]@["uau","y","ah!"];	> val it = ["uau", "y", "ah!"] : string list

Pattern matching (Coincidencia de patrones):

En **ML** hay un mecanismo para **vincular, binding**, **nombres** a **valores:**

val [nombre] = [valor]

De esta manera, en usos posteriores, si el intérprete **mosml** puede encontrar un binding único, tal que el patrón quede igual que el valor, se dice que hay coincidencia, en caso contrario estamos ante un error.

Algunos ejemplos:

Al introducir en mosml...	**... se obtiene en mosml**
val x = 10;	> val x = 10 : int
2 * 10;	> val it = 20 : int
val (x,y) = (13,14);	> val x = 13 : int val y = 14 : int
val (13,y) = (13,14);	> val y = 14 : int
val (15,y) = (13,14);	! Uncaught exception: ! Bind
val (a,b,c) = (13,14) ;	! Toplevel input: ! val (a,b,c) = (13,14) ; ! ^^^^^^^^ ! Type clash: expression of type ! 'a * 'b ! cannot have type ! 'c * 'd * 'e ! because the tuple has the wrong number of components
val [] = [];	NOTA: NO MUESTRA NADA, MATCH SIN BINDING!
val [a,b] = [1,2];	> val a = 1 : int val b = 2 : int
val [a,b,c] = [13,14];	! Uncaught exception: ! Bind
val (h::t) = [1,2,3];	> val h = 1 : int val t = [2, 3] : int list
val (h::t) = [];	! Uncaught exception: ! Bind
val PI = 3.14;	> val PI = 3.14 : real
val R = 2.75;	> val R = 2.75 : real
val AREA = PI * R * R;	> val AREA = 23.74625 : real
AREA - 2.66;	> val it = 21.08625 : real

Funciones en ML: