λ

Introduction to ML

Programmazione Funzionale

2023/2024

Università di Trento

Chiara Di Francescomarino

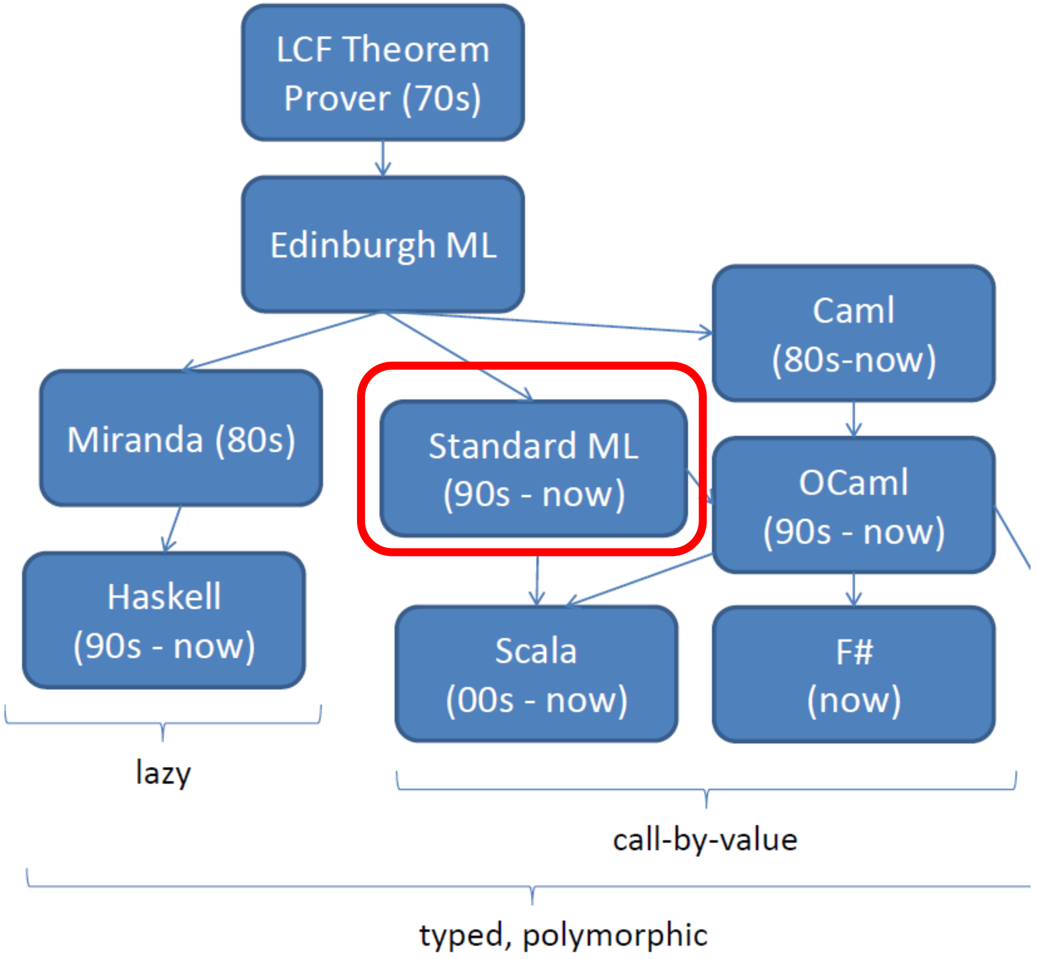
Today

- Introduction to ML
- Basic types and operators
- Type errors and conversions

Introduction to ML

The ML (Meta Language) Programming Language

- General purpose programming language
- It is a meta language for verification purposes
- Different dialects



Standard ML

Robin Milner, Mads Tofte, & Robert Harper
Standard ML
1980's

Why studying ML?

- Functional Programming will make you think differently about programming
    - Mainstream languages are all about states
    - Functional programming is about values
- ML is a practical (small) Programming Language
- New ideas can help make you a better programmer in any language

ML main FP characteristics

1. ML is a functional language:
   - Basic mode of computation: definition and application of functions
   - Functions can be considered as code
   - But they can also be considered as values, i.e., as parameters of other functions
   - Higher-order functions
     - Functions that take functions as values are supported
     - Other languages like C usually only have limited support for this

ML main FP characteristics

2. Recursion
   - Strongly encouraged in ML in preference for while-loops
   - Can be implemented efficiently
   - Iterative constructs are available in ML, when more appropriate
3. Rule-based programming
   - `If-then-else` rules implemented via pattern matching

ML main FP characteristics

4. No side effects
   - Computation is by evaluation of expressions, not by assigning values to variables
   - In C, `a=b+c` modifies the value of `a` (side effect)
   - In ML, `b+c` creates a new element associated with the result
   - If needed, side effects are allowed (printing output), but are not the main means of computation

ML characteristics

5. Strong typing
   - All values and variables have types that can be determined at "compile-time"
   - `4` value of integer type, `4.0` value of real type
   - Valuable debugging aid
   - Variable declarations usually not needed

ML characteristics

6. Polymorphism
    - A function can have arguments of different types
    - In C, we may have to create different programs for sorting arrays of strings or reals
    - In ML we can define one program that work for any type
7. Abstract Data Types (Structures)
    - Elegant type system
    - Ability to construct new types

Running ML

- Three ways:
  - Command line poly/interface poly
    - In order to enter through command line type `poly`
    - We see > once we are inside the Poly environment
  - If `foo` is a file name, there are two options

    `poly < foo`

    `use "foo";` (inside the Poly environment)

- Inside the Poly environment
  - `To quit: CTRL+D`
  - `To interrupt: CTRL+C`

+

x * 4

div

2+3

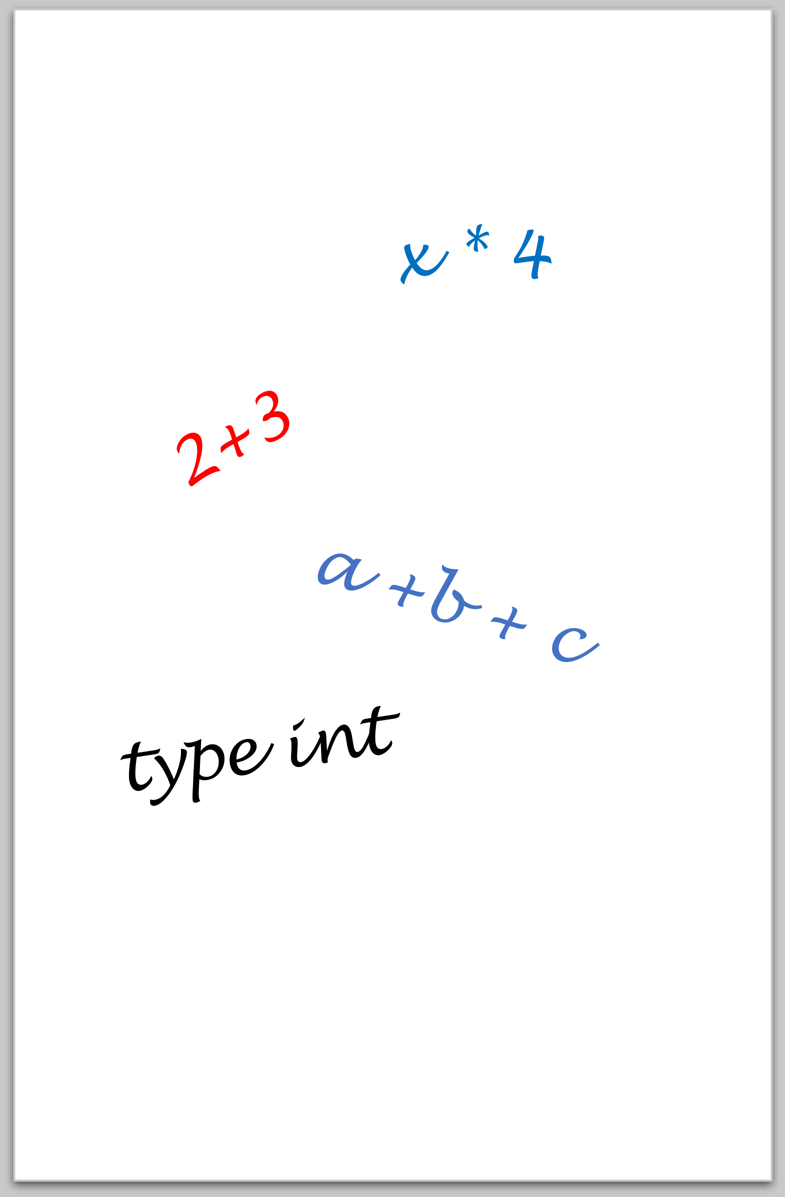
>=

a+b+c

type int

mod

Expressions, types and operators



Expressions and types

Expressions

- Basic notion in a functional language
- Expressions can be values or functions applied to values (to be evaluated)

- Functions and values have types
- ML uses "eager evaluation": first, evaluate the arguments of a function, then the function itself
  - Except for few constructs (e.g., `andalso`, `orelse` and `if then else`)

Example of expressions

```
> 1+2*3;
val it = 7: int
```

Any expression ends with ;

- `val`: value of the expression (7)
- `it`: name of the result
- `int`: type (inferred automatically) of the result

Types

- Basic types

  `int, real, bool, char, string, unit`

  - `int`: Integers, positive and negative. Note that ~3 is -3. This is actually an operator, that negates the integer.
  - `unit`: Single value `()`, used for expressions that do not return a value
- Complex types: constructed starting from other types

Integers

- Integers
  - Positive integers, e.g.: 0, 1234
  - Negative integers, e.g.: ~1234 (not -1234)
  - Hexadecimals

```
> 0x124;
val it = 292: int
> ~0x124;
val it = ~292: int
```

Reals

- Reals
  - One of more digits
  - At least one of
    - Decimal point and more digits
    - The letter `E` or `e`, and one or more digits
- Examples

```
> ~123.0;
val it = ~123.0: real
> 3E~3;
val it = 0.003: real
> 3.14e12;
val it = 3.14E12: real
```

Booleans

- Booleans
  - `true` and `false` (only lower-case)
- Examples

```
> true;
val it = true: bool
> 1=3;
val it = false: bool
```

Characters

- Single character #"a"

```
> #"a";
val it = #"a": char
```

Strings

- Double quotes `"foo"`
- Special characters
  - `\n`: newline
  - `\t`: tab
  - `\\`: Backslash
  - `\"`: Double-quote
  - `\xyz`: ASCII character with this code

^

+

div >=

*

mod

Operators

Arithmetic operators

- Arithmetic operators
  - + and -
  - *, / (division of reals), div (division of integers, rounding down), mod (remainder of integer division)
  - ~ unary minus
- Usual precedence rules

```
> 3.0 - 4.5 + 6.7;
val it = 5.2: real
> 3 – 4 * 2;
val it = ~5: int
> 43 div (8 mod 3) * 5;
val it = 105: int
> 3 + 4.0;
poly: : error: Type error in function application
Function: + : int * int -> int
Argument: (3, 4.0) : int * real
Reason: Can't unify int (*In Basis*) with real (*In Basis*)
(Different type constructors)
Found near 3 + 4.0 Static Errors
```

We will be back to this later

Infix operators

- Note the line

```
Function: + : int * int -> int
```

- Operators in ML are prefix form (+(1,2)) with infix notation allowed for some operators for convenience
- + has only one argument and this argument is a tuple of two numbers

```
> (1,2);
val it = (1, 2): int * int
> (1.0,2.0);
val it = (1.0, 2.0): real * real
> (1,2.0);
val it = (1, 2.0): int * real
```

We will see tuples in the next lecture

- In the ML type system, * is used for tuples

String operators

- ^: Concatenation
- Example

```
> "house" ^ "cat";
val it = "housecat": string
> "house" ^ "";
val it = "house": string
```

Comparisons

- `=, <, >, <=, >=, <>`
- Can be used to compare integers, reals, characters, strings (lexicographical order), but:
  - Equality (and non-equality) comparisons of reals is not allowed

Examples

```
> 1 < 2;
val it = true: bool
> 1.0 < 2.0;
val it = true: bool
> 1 = 2;
val it = false: bool
> 1.0 = 2.0;
poly: : error: Type error in function application.
Function: = : ''a * ''a -> bool
Argument: (1.0, 2.0) : real * real
Reason: Can't unify ''a to real (Requires equality type)
Found near 1.0 = 2.0
```

We will be back to this later on

Examples

```
> #"Z" < #"a";
val it = true: bool
> "abc" <= "ab";
val it = false: bool
> "abc" <= "acc";
val it = true: bool
```

Lexicographical order: strings are compared lexicographically and, in case of the same prefix (e.g., “abc” and “ab”), the shorter string “precedes” the longer one.

Logical operations

- Operations on booleans:
  - `not`
  - `andalso`: if the first operand is `false`, then it will even not evaluate the second operator
  - `orelse`: if the first operand is `true`, then it will even not evaluate the second operator

Lazy evaluation

- Examples

```
> 1<2 andalso 3>4;
val it = false: bool
> 1<2 orelse 3>4;
val it = true: bool
```

- Take care of the precedence rules

```
> not 1<2;
poly: : error: Type error in function application.
 Function: < : 'a * 'a -> bool
 Argument: (not 1, 2) : bool * ‘a
> not (1<2);
val it = false: bool
```

If-then-else

- Syntax

```
if <p> then <exp1> else <exp2>;
```

- This (like everything in ML) is an expression. Therefore
  - `else` is required
  - Both parts must have values and the resulting expression a well-defined type
- Example

```
> if 1<2 then 3+4 else 5+6;
val it = 7: int
```

Examples

```
> if 5<6 then 5 else 6;
val it = 5: int

> if 5<6 then 5;
poly: : error: else expected but ; was found
poly: : error: Expression expected but ; was found
Static Errors
```

Both then and else parts should be there

```
> if 5<6 then 5 else 6.0;
poly: : error: Type mismatch between then-part and else-part.
   Then: 5 : int
   Else: 6.0 : real
   Reason: Can't unify int to real (Incompatible overloadings)
Found near if 5 < 6 then 5 else 6.0
Static Errors
```

Well-defined type required

What happens if … (without trying it ☺)

- we ask for

```
> 1;
val it = 1: int
```

- we ask for

```
> 1=8;
val it = false: bool
```

- we ask for

```
> 1.0;
val it = 1.0: real
```

What is the result of... (without trying it ☺)

```
> 1+2*4;
val it = 9: int

> 5.3-4.0/2.0;
val it = 3.3: real
```

What is the result of… (without trying it ☺)

```
> 11 div 2 mod 3;
val it = 2: int

> "hello"^" "^"world"^"";
val it = "hello world": string
```

What is the result of... (without trying it ☺)

```
> 3>4 orelse 5<6 andalso not (7<>8);
val it = false: bool
```

`not` precedence over the others
`andalso` precedence over `orelse`

```
> 3<4 orelse 6<5 andalso not (7<>8);
val it = true: bool
```

```
> if 6<10 then 6.0 else 10.6;
val it = 6.0: real
```

What is the result of... (without trying it ☺)

```
> 0xab + 123;
val it = 294: int
```

```
0xab = 11 + 10*16 = 171
```

```
> 0xab < 123;
val it = false: bool
```

What is the error in the following expression (without trying it ☺)?

```
> 8/4;
poly: : error: Type error in function application.
   Function: / : real * real -> real
   Argument: (8, 4) : int * int
   Reason: Can't unify real to int (Incompatible
overloadings)
Found near 8 / 4
Static Errors

> if 2<3 then 4;
poly: : error: else expected but ; was found
poly: : error: Expression expected but ; was found
Static Errors
```

What is the error in the following expression (without trying it ☺)?

```
> 1<2 and 5>3;
poly: : error: ; expected but and was found
Static Errors


> 6+7 DIV 2;
poly: : error: Value or constructor (DIV) has not been
declared Found near 6 + 7 DIV 2
poly: : error: Type error in function application.
   Function: 7 : int
   Argument: DIV : bad
   Reason: Value being applied does not have a function type
Found near 6 + 7 DIV 2
Static Errors
```

What is the error in the following expression (without trying it ☺)?

```
> #"a"^#"b";
poly: : error: Value or constructor (^#) has not been
declared Found near #"a" ^# "b"
poly: : error: Type error in function application.
   Function: #"a" : char
   Argument: ^# : bad
   Reason: Value being applied does not have a function
type
Found near #"a" ^# "b"
Static Errors
> 123.;
poly: : error: malformed real number: 123.;
Static Errors
```

Type errors and conversion

Type errors

Type errors

```
> 1 + 2;
val it = 3: int

> 1.0 + 2.0;
val it = 3.0: real

> 1 + 2.0;
poly: : error: Type error in function application.
Function: + : int * int -> int
Argument: (1, 2.0) : int * real
Reason:
Can't unify int (*In Basis*) with real (*In Basis*)
(Different type constructors)
Found near 1 + 2.0
Static Errors
```

What does the error message mean?

Type errors

- Type of `+` is `int * int -> int`
- Actually, + can have different types, but, based on the first argument, ML decides on the integer version
- What if we have a real as “first argument”?

```
> 1.0 + 2;
poly: : error: Type error in function application.
Function: + : real * real -> real
Argument: (1.0, 2) : real * int
Reason:
Can’t unify int (*In Basis*) with real (*In Basis*)
(Different type constructors)
Found near 1.0 + 2
Static Errors
```

- In this case, ML decides that we want real addition: `real * real -> real`
- In both cases, the second argument does not match the first

Other type errors

```
> #"a" ^ "bc";
poly: : error: Type error in function application.
Function: ^ : string * string -> string
Argument: (#"a", "bc") : char * string

> 1/2;
poly: : error: Type error in function application.
Function: / : real * real -> real
Argument: (1, 2) : int * int

> if 1<2 then #"a" else "bc";
poly: : error: Type mismatch between then-part and else-part.
Then: #"a" : char
Else: "bc" : string
```

Type conversion

How to deal with these issues?

Type conversion

Conversion between integers and reals

- From integers to reals
  - `real`: convert from integer to real

- From reals to integers → four possibilities:
  - `floor`: round down
  - `ceil`: round up
  - `round`: nearest integer
  - `trunc`: truncate

Values halfway between two integers go towards the closest even value.
For example:

```
> round(3.5);
val it = 4: int
> round(4.5);
val it = 4: int
```

Examples

```
> real(4);
val it = 4.0: real
> real 4;
val it = 4.0: real

> floor(3.5);
val it = 3: int
> ceil(3.5);
val it = 4: int
> round(3.5);
val it = 4: int
> trunc(3.5);
val it = 3: int

> floor(~3.5);
val it = ~4: int
> trunc(~3.5);
val it = ~3: int
```

`trunc` and `ceil` are different on positive values, while `trunc` and `floor` on negative values.

Conversion between characters and integers

- ord: from character to integer
- chr: from integer to character

```
> ord #"a";
val it = 97: int
> ord #"a" - ord #"A";
val it = 32: int

> chr 97;
val it = #"a": char
```

Integers in the range 0 to 255

Conversion from characters to strings

- `str`: from character to string

```
> str #"a";
val it = "a": string
```

In other words …

- No automatic conversion of types
  - 5+7 and 5.0+7.0 are correct, resulting in `int` and `real`
  - 5+7.0 is wrong
- Conversion between types
  - `real`: integer to real
  - `ceil`, `floor`, `round` and `trunc`: real to integer
  - `ord`: character to integer
  - `chr`: reverse direction, i.e., integer to character
  - `str`: character to string

Exercise L1.1

- Write the expression to convert
  - 123.45 to the next lower integer
  - -123.45 to the next lower integer
  - 123.45 to the next higher integer
  - -123.45 to the next higher integer

Solutions L1.1

```
> floor 123.45;
val it = 123: int
> floor ~123.45;
val it = ~124: int
> ceil 123.45;
val it = 124: int
> ceil ~123.45;
val it = ~123: int
```

Exercise L1.2

- Write the expression to convert
  - #"Y" to an integer
  - 120 to a character
  - 97.0 to a character
  - #"N" to a real
  - #"Z" to a string

Solutions L1.2

```
> ord #"Y";
val it = 89: int
> chr 120;
val it = #"x": char
> chr(round(97.0));
val it = #"a": char
> real(ord(#"N"));
val it = 78.0: real
> str #"Z";
val it = "Z": string
```

Exercise L1.3

- What are the type errors in the following expressions and how can we fix them?

  - `ceil(4);`
  - `if true then 5+6 else 7.0;`

Solutions L1.3

```
> ceil(4);
poly: : error: Type error in function application.
   Function: ceil : real -> int
Static Errors
> ceil (4.0);
val it = 4: int

> if true then 5+6 else 7.0;
poly: : error: Type mismatch between then-part and else-part.
   Then: 5 + 6 : int
   Else: 7.0 : real
> if true then 5+6 else 7;
val it = 11: int
```

Exercise L1.4

- What are the type errors in the following expressions?
  - `chr(256);`
  - `chr(~1);`

Solutions L1.4

```
> chr(256);
Exception- Chr raised
> chr(~1);
Exception- Chr raised
```

Exercise L1.5

- What are the type errors in the following expressions and how can we fix them?
  - `if 0 then 1 else 2;`
  - `ord(“a”)`

Solutions L1.5

```
> if 0 then 1 else 2;
poly: : error: Condition in
if-statement must have type
bool.
   If: 0 : int
   Reason:
      Can’t unify int (*In
Basis*) with bool (*In Basis*)
         (Different type
constructors)
Found near if 0 then 1 else 2
Static Errors
> if false then 1 else 2;
val it = 2: int
```

```
> ord("a");
poly: : error: Type error in
function application.
   Function: ord : char -> int
   Argument: ("a") : string
   Reason:
      Can’t unify char (*In
Basis*) with string (*In
Basis*)
         (Different type
constructors)
Found near ord ("a")
Static Errors
> ord(#"a");
val it = 97: int
```

Summary

- Introduction to ML
- Basic types and operations
- Type errors and conversions

Next time

- Names and environments