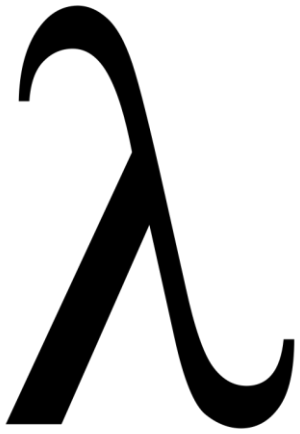


ML Lab 6

Programmazione Funzionale

2024/2025

Università di Trento

Chiara Di Francescomarino

Today

- Higher order functions in ML
- Old exercises
- New exercises

When you have time

Join this Wooclap event

1 Go to wooclap.com

2 Enter the event code in the top banner

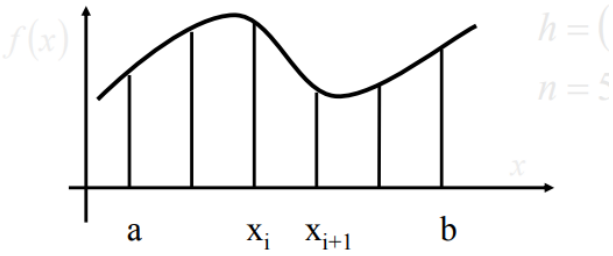
Event code
DQDQRX

Higher-order functions in ML

Higher-order functions

- Functions that take functions as arguments
- Example: Approximate numerical integration $\int_a^b f(x)dx$
  - Divide the interval from a to b into n equal parts
  - Sum the areas of the n trapezoids

$$\int_a^b f(x)dx \approx \frac{b-a}{n} \cdot \sum_{i=1}^{n} \frac{f(x_{i-1}) + f(x_i)}{2} = \sum_{i=1}^{n} \frac{b-a}{n} \cdot \frac{f(x_{i-1}) + f(x_i)}{2}$$



- We define a function `trap(a,b,n,F)` to do this, where the function `F` to be integrated is one of the parameters

Integration

$$\sum_{i=1}^{n} \frac{b-a}{n} \cdot \frac{f(x_{i-1}) + f(x_i)}{2}$$

```
> fun trap (a,b,n,F) =
    if n<=0 orelse b-a<=0.0 then 0.0
    else let
        val delta = (b-a)/real(n)
    in
        delta * (F(a)+F(a+delta))/2.0 + trap (a+delta,b,n-
        1,F)
    end;
val trap = fn: real * real * int * (real -> real) -> real
```

Example

```
> fun square(x:real) = x*x;
val square = fn: real -> real

> trap (0.0,1.0,8,square);
val it = 0.3359375: real
```

Some other higher-order functions

- We can write a function `simpleMap` that
  - takes a function F and a list $[a_1, \ldots, a_n]$ and produces the list $[F(a_1), \ldots, F(a_n)]$
  - we call it `simpleMap` to distinguish it from the built-in function `map` that we will see next time
- For instance
  - `simpleMap(square,[1.0,2.0,3.0]) = [1.0,4.0,9.0]`

simpleMap

```
> fun simpleMap (F,nil) = nil
    | simpleMap (F,x::xs) = F(x) :: simpleMap(F,xs);
val simpleMap = fn: ('a -> 'b) * 'a list -> 'b list

> simpleMap (square, [1.0,2.0,3.0]);
val it = [1.0, 4.0, 9.0]: real list
```

Further examples

- Using a unary operator

```
> simpleMap (~, [1,2,3]);
val it = [~1, ~2, ~3]: int list
```

- Using an anonymous function

```
> simpleMap ( fn x => x*x, [1.0,2.0,3.0]);
val it = [1.0, 4.0, 9.0]: real list
```

What would be possible definitions of `f(x,y,z)` where the argument has the following types?

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

The `reduce` function

- Define a function `reduce` as follows
  - List $[a_1]$ returns a_1
  - List $[a_1, \ldots, a_n]$. Reduce the tail applying a function F that takes a pair and returns b and then compute $F(a_1, b)$, e.g.,

    `reduce` $([a_1, \ldots, a_n]$, F) = F(a_1, F(a_2, … F(a_{n_1}, a_n)))
- For instance:
  - given `fun plus (x,y) = x+y;`
  - `reduce([1,3,5], plus) = 9;`

The `reduce` function

- This means that:
  - reducing a list with the addition function returns the sum of the elements of the list
  - reducing a list with the multiplication function returns the product of the elements of the list
  - reducing a list with the logical AND returns true if all the elements of a boolean list are true
  - reducing a list with max returns the largest of the elements in the list

Definition of `reduce`

```
> exception EmptyList;
exception EmptyList

> fun reduce (F,nil) = raise EmptyList
    | reduce (F,[a]) = a
    | reduce (F,x::xs) = F(x, reduce(F,xs));
val reduce = fn: ('a * 'a -> 'a) * 'a list -> 'a
```

Infix operators: op

- In order to apply `reduce` we have to declare a function called `plus`, since "+" is infix

```
> reduce (+, [1,2,3]);
poly: : warning: (+) has infix status but was not
preceded by op.
> fun plus (x,y) = x+y;
val plus = fn: int * int -> int
> reduce(plus, [1,2,3]);
val it = 6: int
```

- If we use op, we can convert an infix operator to a prefix one

```
> reduce (op +, [1,2,3]);
val it = 6: int
```

Using `reduce` to compute variance

- The variance of a list of reals $[a_1, \ldots, a_n]$ is defined as

$$\frac{\left(\sum_{i=1}^{n} {a_i}^2\right)}{n} - \left(\frac{\left(\sum_{i=1}^{n} a_i\right)}{n}\right)^2$$

- Let us define the following two functions:

```
> fun square (x:real) = x*x;
val square = fn: real -> real
> fun plus (x:real,y) = x+y;
val plus = fn: real * real -> real
```

The `variance` function

$$\frac{(\sum_{i=1}^{n} {a_i}^2)}{n} - \left(\frac{(\sum_{i=1}^{n} a_i)}{n}\right)^2$$

- The function

```
> fun variance (L) =
    let
        val n = real(length(L))
    in
        reduce (plus,simpleMap(square,L))/n - square
        (reduce(plus,L)/n)
    end;
val variance = fn: real list -> real

> variance ([1.0,2.0,5.0,8.0]);
val it = 7.5: real
```

What is the effect of reduce?

- What is the effect on a list of

  `reduce(op -,L)`

  $$a_1 - \left(a_2 - (a_3 - a_4)\right)$$

- Corresponding to alternating difference

  $$a_1 - a_2 + a_3 - a_4$$

```
> val L = [1,2,3,4];
val L = [1, 2, 3, 4]: int list
> reduce (op - , L);
val it = ~2: int
```

The `filter` function

- Write a function `filter` that takes as input a predicate, i.e., a boolean function and a list and selects from the list those elements that satisfy the boolean condition

```
> fun filter (P,nil) = nil
    | filter (P,x::xs) =
        if P(x) then x::filter(P,xs)
        else filter (P,xs);
val filter = fn: ('a -> bool) * 'a list -> 'a
list

> filter (fn x => x>=10, [1,10,23,5,16]);
val it = [10, 23, 16]: int list
```

Old exercises

Exercise 4.11

- Write a function `prodDiff` that given a list of reals $[a_1, \ldots, an]$ compute

$$\prod_{i<j}(ai - aj)$$

- You can use support functions, if needed
- For instance
  - `prodDiff([1.0,2.0,3.0])=(1.0-2.0)*(1.0-3.0)*(2.0-3.0) = -2.0`
  - `prodDiff (nil) = 1.0`

Solution 4.11

```
> fun prodDiff1(_,nil) = 1.0
    | prodDiff1(a,b::bs) = (a-b)*prodDiff1(a,bs);
> fun prodDiff(nil) = 1.0
    | prodDiff(b::bs) =
   prodDiff1(b,bs)*prodDiff(bs);
val prodDiff = fn: real list -> real

> prodDiff [1.0,1.1,1.2,1.3,1.4];
val it = 2.88E~8: real
```

Exercise 4.12

- Write a function `is_one` that returns “one” if the parameter is 1 and “anything else” otherwise, using the construct `case` and pattern matching with `fun` and `fn`.
- For instance
  - `is_one 1 = "one"`
  - `is_one 3 = "anything else"`

Solution 4.12

```
val is_one = fn x => case x of
      1 => "one"
      |_ => "anything else ";

fun is_one 1 = "one"
   | is_one _ = "anything else";

> is_one 1;
val it = "one": string
> is_one 3;
val it = "anything else": string
```

Exercise 5.12

- Write a function `getList(filename)` that reads a file, extracts the words (without space characters), and transforms the file in a list of words (without space characters).
- For instance, if the file `helloworld` contains

  Hello beautiful world!

  Bye!

  ```
  getList(helloworld) = ["Hello", "beautiful", "world",
  "Bye!"]
  ```

- Hint: first write a function `getWord(in)` that extracts a word (without spaces) from a `TextIO.instream in` and then put them in a list. You can use support functions

Solution 5.12

```
(* test if a character is white space *)
> fun white(" ") = true
    | white("\t") = true
    | white("\n") = true
    | white("\r") = true
    | white(_) = false;

(* read one word *)
> fun getWord(file) =
    if TextIO.endOfStream(file) then ""
    else
        let
                val c = TextIO.inputN(file,1)
        in
                if white(c) then ""
                else c^getWord(file)
    end;
```

Solution 5.12 (continue)

```
(*test if a string is empty*)
> fun is_empty("")=true
        |is_empty(_)=false;
> fun getList1(file) = (* read all words from an instream *)
    if TextIO.endOfStream(file) then nil
        else
                let
                        val w = getWord(file);
                        val tail = getList1(file)
                in
                        if is_empty(w) then tail
                        else w::tail
                end;
(* read all words from a file given the file name *)
> fun getList(filename) = getList1(TextIO.openIn(filename));
```

New exercises

Exercise 6.1

- Write a program `returnThird(L)` that, given a list of integers `L`, returns its third element. If the list is too short, it raises an exception `shortList` and handles it by printing "List too short.\n It only contains `n` elements.\n" where `n` is the number of elements in the list.

- For instance
  - `returnThird [1,2,3,4] = 3`
  - `returnThird [1,2]`

```
List too short.
It only contains 2 elements.
```

Solution 6.1

```
> exception shortList of int list;
> fun returnThird1 L =
      if length(L) < 3 then raise shortList (L)
      else hd(tl(tl(L)));
val returnThird1 = fn: int list -> int
> fun returnThird L = returnThird1 L handle
      shortList L => (
      print ("List too short.\n");
      print ("It only contains "); print(Int.toString(length(L))); print ("
      elements.\n");
      0
      );
val returnThird = fn: int list -> int

> returnThird [1,2,3,4];
val it = 3: int
> returnThird [1,2];
List too short.
It only contains 2 elements.
val it = 0: int
```

Solution 6.1

- Another possible solution

```
> exception shortList of int;
> fun thirdElement1 nil = raise shortList(0)
                        |thirdElement1[x] = raise shortList(1)
                        |thirdElement1[x,y] = raise shortList(2)
                        |thirdElement1 L = hd(tl(tl(L)));

> fun thirdElement L = thirdElement1 L handle
        shortList n => (
                print("List too short.\n");
                print("It only contains ");
                print(Int.toString(n));
                print(" elements.\n");
                0);
```

Exercise 6.2

- Write a factorial function `fact` that, given an integer `n`, returns `n!`, `1` when its argument is `0`, `0` for a negative argument with an error message “Negative argument `x` found” where `x` is the negative argument
- For instance
  - `fact 5 = 120`
  - `fact 0 = 1`
  - `fact ~2 = 0`

```
Negative argument ~2 found.
```

Solution 6.2

```
> exception Negative of int;
> fun fact1(0) = 1
        | fact1(n) =
            if n>0 then n*fact1(n-1)
            else raise Negative(n);
val fact1 = fn: int -> int
> fun fact(n) = fact1(n) handle Negative(n) => (
        print("Negative argument ");
        print(Int.toString(n));
        print(" found.\n");
        0
    );
val fact = fn: int -> int
```

Solution 6.2

```
> fact 5;
val it = 120: int

> fact 0;
val it = 1: int

> fact ~2;
Negative argument ~2 found
val it = 0: int
```

Exercise 6.3

- Write a function `tabulate` that takes an initial value a, an increment δ, a number of points n, and a function F from reals to reals and print a table with columns corresponding to x and $F(x)$, where $x = a, a + \delta, a + 2\delta, \ldots, a + (n-1)\delta$
- For instance
  - `tabulate (1.0,0.1,9,fn x => x*x);`

```
1.0  1.0
1.1  1.21
1.2  1.44
1.3  1.69
1.4  1.96
1.5  2.25
1.6  2.56
1.7  2.89
1.8  3.24
```

Solution 6.3

```
> fun tabulate(x,delta,0,F) = ()
    | tabulate(x,delta,n,F) = (
        print(Real.toString(x));
        print("\t");
        print(Real.toString(F(x)));
        print("\n");
        tabulate(x+delta,delta,n-1,F)
    );
val tabulate = fn: real * real * int * (real -> real) -> unit
```

Solution 6.3

```
> tabulate (1.0,0.1,9,fn x => x*x);
1.01.0
1.11.21
1.21.44
1.31.69
1.41.96
1.52.25
1.62.56
1.72.89
1.83.24
val it = (): unit
```

Exercise 6.4

- Use the function `simpleMap` to replace every negative element of a list of reals with 0.
- For instance applied to a list of reals such as `L = [0.0,1.0,~2.1,~2.3]`, it should return `[0.0,1.0,0.0,0.0]`

Solution 6.4

```
> val L = [0.0,1.0,~2.1,~2.3];
val L = [0.0, 1.0, ~2.1, ~2.3]: real list

simpleMap(fn(x)=>if x<0.0 then 0.0 else x, L);
val it = [0.0, 1.0, 0.0, 0.0]: real list
```

Exercise 6.5

- Use the function `reduce` to find the maximum of a list of reals.
- For instance, applied to a list of reals such as `L = [1.1,2.2,4.4,3.3]`, it should return `4.4`.

Solution 6.5

```
> val L = [1.1,2.2,4.4,3.3];
val L = [1.1, 2.2, 4.4, 3.3]: real list

> reduce(fn(x,y)=> if x<y then y else x, L) ;
val it = 4.4: real
```

Exercise 6.6

- Use `filter` to find the elements of a list of reals that are greater than 0.
- For instance, applied to a list of reals such as `L = [1.1,~1.2,~1.3,1.4]`, it should return `[1.1,1.4]`

Solution 6.6

```
> val L = [1.1,~1.2,~1.3,1.4];
val L = [1.1, ~1.2, ~1.3, 1.4]: real list

> filter(fn(x)=>x>0.0, L);
val it = [1.1, 1.4]: real list
```

Exercise 6.7

- Write a function `readAndSum` that, given a text file containing a number per line, transforms each number into an integer, sums them and returns the sum.
- You do not have to handle exceptions.
- In ML there exists a function `Int.fromString (fn: string -> int option)`
- You can use support functions, if needed.
- For instance , given a file “numbers” containing

```
5
8
10
```

```
readAndSum (“numbers”) = 23
```

Solution 6.7

```
> fun readAndSum1 file =
      if TextIO.endOfStream(file) then 0
      else
   valOf(Int.fromString(valOf(TextIO.inputLine(file)))
   ) + readAndSum1(file);
> fun readAndSum filename =
      readAndSum1(TextIO.openIn(filename));
```