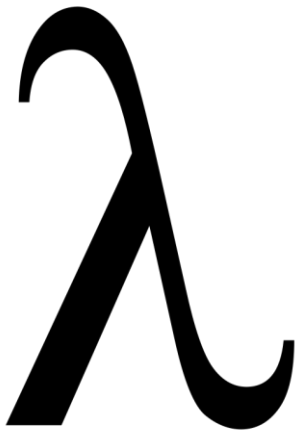


ML Lab 3

Programmazione Funzionale

2024/2025

Università di Trento

Chiara Di Francescomarino

Exercise 3.1

- Write a function `fact` that computes the factorial of n,

$$n! = 1 * 2 * \ldots * n$$

where $n \geq 1$. It does not need to work correctly for small n.

- For instance
  - `fact 1 = 1`
  - `fact 10 = 3628800`

Solution 3.1

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

> fact 1;
val it = 1: int
> fact 10;
val it = 3628800: int
> fact 100;
Exception- Overflow raised
```

It exceeds the maxInt

Exercise 3.2

- Write a function `cyclei` that, given an integer i and a list $L = [a_1, \ldots, an]$, cycles L i times, i.e., produce

$$[a_{i+1}, a_{i+2}, \ldots, a_n, a_1, \ldots, a_i]$$

- For instance
  - `cyclei([1,2,3,4],2) = [3,4,1,2]`
  - `cyclei(["a","b","c","d","e"],4) = ["e","a","b","c","d"]`

Solution 3.2

```
> fun cycle L = tl(L)@[hd(L)];

> fun cyclei (i,L) =
        if i=0 then L
        else cyclei(i-1, cycle(L));
val cyclei = fn: 'a list * int -> 'a list

> cyclei([1,2,3,4],2);
val it = [3, 4, 1, 2]: int list
```

Exercise 3.3

- Write a function `duplicate` that duplicates each element of a list, that is given the list $L = [a_1, \ldots, a_n]$, produce the list $[a_1, a_1, a_2, a_2, \ldots, a_n, a_n]$
- For instance
  - `duplicate([1,2,3,4]) = [1,1,2,2,3,3,4,4]`
  - `duplicate(["a","b","c"]) = ["a","a","b","b","c",“c"]`

Solution 3.3

```
> fun duplicate(L) =
    if L=[] then []
    else hd(L)::(hd(L)::duplicate(tl(L)));
val duplicate = fn: ''a list -> ''a list

> fun duplicate2(L) =
    if L=[] then []
    else [hd(L),(hd(L)]@duplicate(tl(L)));
val duplicate = fn: ''a list -> ''a list


> duplicate [1,2,3,4];
val it = [1, 1, 2, 2, 3, 3, 4, 4]: int list
```

Exercise 3.4

- Write a function `len` that computes the length of a list.
- For instance
  - `len([1,2,3,4]) = 4`
  - `len(["a","b","c"]) = 3`

Solution 3.4

```
> fun len(L) =
    if L=nil then 0
    else 1+len(tl(L));
val len = fn: ''a list -> int

> len [1,2,3,4];
val it = 4: int
```

Exercise 3.5

- Write a function `pow` that computes x^i where x is a real, and i a non-negative integer. It doesn't need to work for $i < 0$
- For instance
  - `pow(2.1,3) = 9.261`
  - `pow(2.0,3) = 9.0`

Solution 3.5

```
> fun pow(x:real, i:int) = if i=0 then 1.0 else
x*pow(x,i-1);
val pow = fn: real * int -> real

> pow(2.1,3);
val it = 9.261: real
> pow(2.0,3);
val it = 8.0: real
```

Exercise 3.6

- Write a function `maxList` that computes the largest (in a lexicographical sense) element of a list of strings, e.g., ([“a”,”abc”, “ab”] → “abc”). It doesn't need to work for empty lists.
- Note that in the lexicographical order:

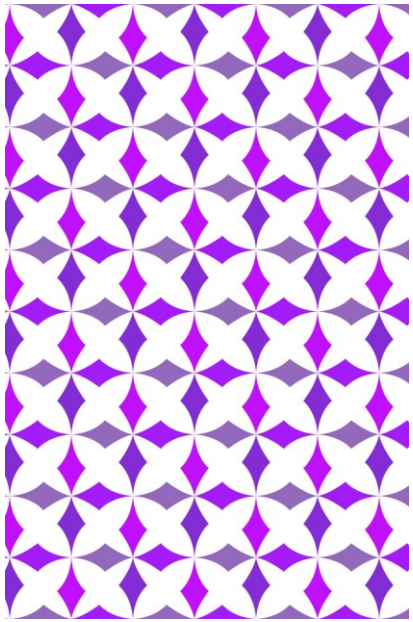
```
> "abc">"ab";
val it = true: bool
> "abc">"abb";
val it = true: bool
> "ab">"a";
val it = true: bool
```

- For instance
  - `maxList(["a","abc","ab"]) = "abc"`

Solution 3.6

```
> fun maxList(L: string list) =
    if tl(L)=nil then hd(L)
    else
    if hd(L)>hd(tl(L)) then maxList(hd(L)::tl(tl(L)))
    else maxList(tl(L));
val maxList = fn: string list -> string

> maxList(["a","abc","ab"]);
val it = "abc": string
```



Patterns in ML

Function definition: patterns

- Very powerful mechanism for defining functions
- A bit like a generalization of “case” or “switch” statements in procedural languages
- Example

  `x::xs`

  matches any non-empty list, with `x` set to the head and `xs` to the tail
- Function definition uses a sequence of patterns. The first that matches the argument determines the produced value

```
fun <identifier> (<first pattern>) = <first expression>
  | <identifier> (<second pattern>) = <second expression>
  ...
  | <identifier> (<last pattern>) = <last expression>;
```

Example: reverse a list

- Using patterns

```
> fun reverse (nil) = nil
    | reverse (x::xs) = reverse(xs) @ [x];
val reverse = fn: 'a list -> 'a list
```

- Without patterns

```
> fun reverse L =
    if L = nil then nil
    else reverse (tl L) @ [hd L];
val reverse = fn: ''a list -> ''a list
```

List of alternatives

- Note: The list of alternatives must be exhaustive, as with `if-then` clauses
  - If the list is not exhaustive, many implementations of ML only give a warning, with an error only if we actually use a parameter that does not match any of the possibilities

```
> fun reverse (nil) = nil;
poly: : warning: Matches are not exhaustive. Found near fun
reverse (nil) = nil
val reverse = fn: 'a list -> 'b list
> reverse([3]);
poly: : warning: The type of (it) contains a free type variable.
Setting it to a unique monotype.
Exception- Match raised
```

Reverse a list with patterns

```
> fun reverse (nil) =
nil
    | reverse (x::xs) =
    reverse(xs) @ [x];
val reverse = fn: 'a
list -> 'a list

> reverse([1,2,3])
```

We even do not try to match the second pattern

xs	nil
x	3
xs	[3]
x	2
xs	[2,3]
x	1
reverse	Definition of reverse

Added in call to `reverse (nil)`

Added in call to `reverse ([3])`

Added in call to `reverse ([2,3])`

Added in call to `reverse ([1,2,3])`

Environment Before the call

Exercise 3.7

- Write the factorial function `fact` using patterns.
- For instance
  - `fact 1= 1`
  - `Fact 10 = 3628800`

Solution 3.7

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

> fact 1;
val it = 1: int
> fact 10;
val it = 3628800: int
```

Exercise 3.8

- Write a function `cycle1` that cycles a list by one position using patterns. If the list is empty, return the empty list.
- For instance
  - `cycle1 [1,2,3,4,5] = [2,3,4,5,1]`
  - `cycle1 [1] = [1]`
  - `cycle1 nil = nil`

Solution 3.8

```
> fun cycle1 (nil) = nil
    | cycle (x::xs) = xs @ [x];

> cycle1 [1];
val it = [1]: int list

> cycle1 [1,2,3];
val it = [2, 3, 1]: int list
```

Exercise 3.9

- Write a function `cycleip` that, given an integer i and a list $L = [a_1, \ldots, a_n]$, cycles L i times, i.e., produce

$$[a_{i+1}, a_{i+2}, \ldots, a_n, a_1, \ldots, a_i]$$

  using patterns

- For instance
  - `cycleip([1,2,3,4],2) = [3,4,1,2]`
  - `cycleip(["a","b","c","d","e"],4) = ["e","a","b","c","d"]`

Solution 3.9

```
> fun cycleip (L,0) = L
    | cycleip (L,i) = cycleip (cycle(L),i-1);
val cycleip = fn: 'a list * int -> 'a list

> cycleip([1,2,3,4],2);
val it = [3, 4, 1, 2]: int list
```

Exercise 3.10

- Write a function `duplicatep` that duplicates each element of a list using patterns.
- For instance
  - `duplicatep [1,2,3,4] =[1,1,2,2,3,3,4,4]`
  - `duplicatep [2.0] =[2.0,2.0]`

Solution 3.10

```
> fun duplicatep(nil) = nil
    | duplicatep(x::xs) = x::x::duplicatep(xs);
val duplicatep = fn: 'a list -> 'a list

> duplicatep [1,2,3,4];
val it = [1, 1, 2, 2, 3, 3, 4, 4]: int list
```

Exercise 3.11

- Write a function `power` that computes x^i where x is an integer, and i a non-negative integer. It doesn't need to work for $i < 0$
- For instance
  - `power(2.1,3) = 9.261`
  - `power(2.0,3) = 9.0`

Solution 3.11

```
> fun power (x,0) = 1
    | power (x,i) = x * power (x,i-1);
val power = fn: int * int -> int

> power (4,0);
val it = 1: int
> power (4,3);
val it = 64: int
```

Exercise 3.12

- Write a function `maxList` that computes the largest of a list of reals, assuming that the list is not empty, using patterns.
- For instance
  - `maxList [2] = 2`
  - `maxList [2,5,4] = 5`

Solution 3.12

```
> fun maxList([x:real]) = x
    | maxList(x::y::zs) =
        if x<y then maxList(y::zs)
        else maxList(x::zs);
poly: : warning: Matches are not exhaustive.
val maxList = fn: real list -> real

> maxList [2.0];
val it = 2.0: real
> maxList [2.0,3.1,2.7];
val it = 3.1: real
```