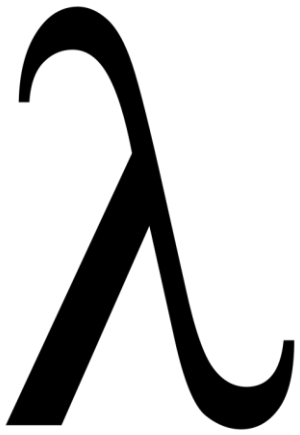

ML Lab 4

Programmazione Funzionale

2024/2025

Università di Trento

Chiara Di Francescomarino

Exercise 4.1

- Write a function `flip` that flips alternate elements of a list using patterns. $[a_1, a_2, \ldots, a_{n-1}\ a_n]$ should become $[a_2, a_1, \ldots, a_n, a_{n-1}]$. If n is odd, leave a_n at the end.
- For instance
  - `flip [1,2,3,4,5] = [2,1,4,3,5]`
  - `flip [1,2,3,4] = [2,1,4,3]`

Solution 4.1

```
> fun flip ([]) = []
    | flip ([x]) = [x]
    | flip (x::y::zs) = y::x::flip(zs);
val flip = fn: 'a list -> 'a list

> flip [1,2,3,4,5];
val it = [2, 1, 4, 3, 5]: int list
```

Exercise 4.2

- Write a function `remove` that, given a list L and an integer i, returns L with the i^{th} element deleted. If the length of L is lower than i, return L.
- For instance
  - `remove([1],1) = []`
  - `remove([1,2,3],3) = [1,2]`
  - `remove([1,2],3) = [1,2]`

Solution 4.2

```
> fun remove ([],m) = []
    | remove (x::xs,1) = xs
    | remove (x::ys,i) = x:: remove (ys,i-1);
> remove([1],5);
val it = [1]: int list
> remove([],4);
poly: : warning: The type of (it) contains a free type
variable. Setting it to a unique monotype.
val it = []: _a list
> remove([1],1);
val it = []: int list
```

Exercise 4.3

- Write a function `square` that takes as input an integer `n` and compute the square of `n`, using patterns according to the formula

$$n^2 = (n - 1)^2 + 2n - 1$$

- For instance
  - `square(2) = 4`
  - `square(5) = 25`
  - `square (0) = 0`

Solution 4.3

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

> square 0;
val it = 0: int
> square 6;
val it = 36: int
```

Exercise 4.4

- Write a function `flip` that takes as input a list of pairs of **integers** and orders each pair so that the smallest number is first, using patterns.
- For instance
  - `flip [(1,2),(4,3)] = [(1,2),(3,4)]`
  - `flip [(5,2),(4,3),(6,5),(1,2)] = [(2,5),(3,4),(5,6),(1,2)]`
  - `flip [(1,1),(1,2)] = [(1,1),(1,2)]`
  - `flip nil = nil`

Solution 4.4

```
> fun flip(nil) = nil
    | flip((x as (a:int,b))::xs) =
       if a<b then x::flip(xs) else
       (b,a)::flip(xs);
val flip = fn: (int * int) list -> (int * int)
list

> flip [(1,2),(4,3),(6,5)];
val it = [(1, 2), (3, 4), (5, 6)]: (int * int)
list
```

Exercise 4.5

- Write a function `vowel` that takes a list of characters and returns `true` if the first element is a vowel using patterns.
- For instance
  - `vowel [#"a",#"b"] = true`
  - `vowel [#"b",#"c"] = false`
  - `vowel [#"a"] = true`
  - `vowel nil = false`

Solution 4.5

```
> fun vowel(#"a"::ys) = true
    | vowel(#"e"::ys) = true
    | vowel(#"i"::ys) = true
    | vowel(#"o"::ys) = true
    | vowel(#"u"::ys) = true
    | vowel(_) = false;
val vowel = fn: char list -> bool

> vowel [#"a",#"b"];
val it = true: bool
> vowel [#"b",#"a"];
val it = false: bool
```

Exercise 4.6

- Let us represent sets by lists. We represent a set by a list: the elements can be in any order, but without repetitions.
- Write a function `member(x,S)` to test whether `x` is a member of set `S` using patterns.
- For instance
  - `member (1,[2,3]) = false`
  - `member (2,[2,3,1]) = true`
  - `member (5,nil) = false`
  - `member ("b",["aa","c"]) = false`

Solution 4.6

```
> fun member(_,nil) = false
    | member(x,y::ys) =
        (x=y orelse member(x,ys));
val member = fn: ''a * ''a list -> bool

> member (5,[6,7,5]);
val it = true: bool
> member (5,[6,7,8]);
val it = false: bool
```

Exercise 4.7

- Write a function `delete` that deletes an element from a set `delete(x,S)` using patterns – if the set contains the element. If the item is not contained in the set, the function returns the set itself.
- For instance
  - `delete(1,[2,3,4]) = [2,3,4]`
  - `delete (1,[2,1,3]) = [2,3]`
  - `delete (1,nil) = nil`
  - `delete (#"a",[#"c",#"b",#"a"]) = [#"c",#"b“]`

Solution 4.7

```
> fun delete (a,[]) = []
    | delete (b,c::ys) = if b=c then ys
       else c::delete(b,ys);
val delete = fn: ''a * ''a list -> ''a list

> delete (2,[3,4,2,5]);
val it = [3, 4, 5]: int list
> delete (2,[3,4,5]);
val it = [3, 4, 5]: int list
```

Exercise 4.8

- Write a function `insert` that inserts an element `x` into a set S (in whatever order) using patterns. Since S is a set, although a list is used for representing it, it has to be added only if it is not already present in the set.
- For instance
  - `insert (2,[3,4,5]) = [3,4,5,2]`
  - `insert (3,[3,4,5]) = [3,4,5]`
  - `insert (2, nil) = [2]`
  - `insert (#"a",[#"b",#"c"]) = [#"b",#"c", #"a"]`

Solution 4.8

```
> fun insert(x,nil) = [x]
    | insert(x,S as y::ys) =
       if x=y then S else y::insert(x,ys);
val insert = fn: ''a * ''a list -> ''a list

> insert (2,[3,4,5]);
val it = [3, 4, 5, 2]: int list
> insert (3,[3,4,5]);
val it = [3, 4, 5]: int list
```

Exercise 4.9

- Write a function `insertAll` that takes an element `a` and a list of lists `L` and inserts `a` at the front of each of these lists.
- For example
  - `insertAll (1,[[2,3],[],[3]]) = [[1,2,3],[1],[1,3]]`
  - `insertAll (1,nil) = nil`
  - `insertAll (#"c",[[#"a",#"t"],[#"a",#"r"],nil])= [[#"c", #"a", #"t"], [#"c", #"a", #"r"], [#"c"]]`

Solution 4.9

```
> fun insertAll(a,nil) = nil
    | insertAll(a,L::Ls) =
       (a::L)::insertAll(a,Ls);
val insertAll = fn: 'a * 'a list list -> 'a list
list

> insertAll (1,[[2,3],[4,5,6],nil]);
val it = [[1, 2, 3], [1, 4, 5, 6], [1]]: int
list list
```

Exercise 4.10

- Suppose that sets are represented by lists. Write a function `powerSet` that takes a list `L`, and produces as output the power set of the list
- If S is a set, the power set of S is the set of all subsets S' such that $S' \subseteq S$
- You can use support functions, if needed
- For instance
  - `powerSet([1,2,3])=[[],[1],[2],[3],[1,2],[1,3],[2,3],[1,2,3]]`
  - `powerSet [#"a",#"c"]=[[], [#"c"], [#"a"], [#"a", #"c"]]`
  - `powerSet nil = [[]]` (do not care about type warning)

Solution 4.10

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


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

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 ";

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

Solution 4.12

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

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

- This would be wrong …why?

```
val f = fn
_ => "anything else";
| 1 => "one"
```

It would always match anything else