PROGRAMMAZIONE 2
15bisbis.
OCaML: un veloce ripasso

Funzioni generiche

Analizziamo la funzione *length* applicata a *int list* e *string list*

```
let rec length (l : int list) : int =
      match l with
            [] -> 0
          | _::tl -> 1 + length tl
```

Le funzioni sono identiche, eccetto l'annotazione di tipo

```
let rec length (l : string list) :int =
      match l with
            [] -> 0
          | _::tl -> 1 + length tl
```

Generici in OCaML

```
let rec length (l : 'a list) : int =
      match l with
            [] -> 0
          | _::tl -> 1 + (length tl)
```

La notazione 'a list indica una lista generica

length [1; 2; 3] applica la funzione a *int list*

length ["a"; "b"; "c"] applica la funzione a *string list*

Append generico

```
let rec append (l1 : 'a list) (l2 : 'a list):'a list=
      match l1 with
            [] -> l2
          | h::tl -> h::(append tl l2)
```

Il pattern matching permette di operare su tipi generici

h ha tipo 'a

tl ha tipo 'a list

Generic zip

```
let rec zip (l1 : 'a list) (l2 : 'b list):
('a * 'b) list=
      match (l1, l2) with
    (h1::t1, h2::t2) -> (h1, h2)::(zip t1 t2)
          | _ -> []
```

La funzione opera su tipi generici multipli
(da `'a list` e `'b list` verso `('a * 'b) list`)

`zip` [1;2;3] ["a";"b";"c"] = [(1,"a");(2,"b");(3,"c")] : (int * string) list

Generic tree


```
type 'a tree =
        Empty
      | Node of 'a tree * 'a * 'a tree
```

Si noti l'utilizzo del parametro di tipo **'a**

Generic BST

```
let rec insert (t : 'a tree) (n : 'a): 'a tree=
        match t with
            Empty-> Node (Empty, n, Empty)
          | Node (lt, x, rt) ->
                if x = n then t
                  else if n < x then
                   Node (insert  lt n, x, rt)
                          else
                   Node (lt, x, insert  rt n)
```

Gli operatori di `Node` < operano su ogni tipo di dato

Collection (Set)

- Un insieme è una collezione di dati omogenei con operazioni di unione, intersezione, etc.
- Un Set è sostanzialmente una lista nella quale
  - la struttura d'ordine non è importante
  - non sono presenti duplicati

  ma non è un **tipo primitivo** in OCaML
- Diversi modi per implementare Set
- **Come possiamo astrarre dall'implementazione?**

Set

- Un BST definisce una implementazione della struttura Set
  - *l'insieme vuoto (**bst empty**)*
  - *determinare tutti gli elementi che appartengono all'insieme (**visita dell'albero**)*
  - *definire una operazione per testare l'appartenenza di un elemento a un insieme (**lookup**)*
  - *definire unione e intersezione (tramite **insert e delete**)*

OCaML: Set Interface


```
module type Set = sig
  type 'a set
  val empty : 'a set
  val add : 'a -> 'a set -> 'a set
  val remove : 'a -> 'a set -> 'a set
    val list_to_set : 'a list -> 'a set
    val member : 'a -> 'a set -> bool
    val elements : 'a set -> 'a list
end
```

Module type (in un file .mli) per dichiarare un **TdA**

sig ... end racchiudono una segnatura, che definisce il TdA e le operazioni

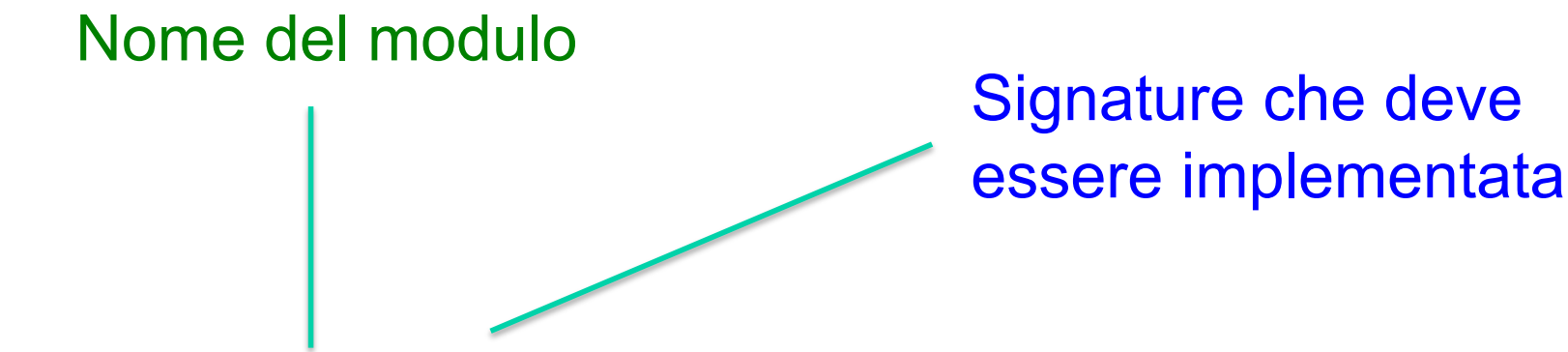
val: nome dei valori che devono essere definiti e dei loro tipi

OCaML: Set Interface


```
module type Set = sig
   type 'a set
    val empty : 'a set
    val add : 'a -> 'a set -> 'a set
    val remove : 'a -> 'a set -> 'a set
    val list_to_set : 'a list -> 'a set
    val member : 'a -> 'a set -> bool
    val elements : 'a set -> 'a list
end
```

Idea (solita): fornire diverse funzionalità nascondendo la loro implementazione

Moduli in OCaML




```
module Myset : Set = struct …
    (* implementations of all the operations *)
    :
end
```

“dot notation”

```
let s1 = Myset.add 3 Myset.empty
let s2 = Myset.add 4 Myset.empty
let s3 = Myset.add 4 s1
let test() : bool (Myset.member 3 s1) = true
;; run_test "Myset.member 3 s1" test
let test() : bool = (Myset.member 4 s3) = true
 ;; run_test "Myset.member 4 s3" test
```

Open module

Alternativa: aprire lo scope del modulo (**open**) per portare i nomi nell'ambiente del programma in esecuzione

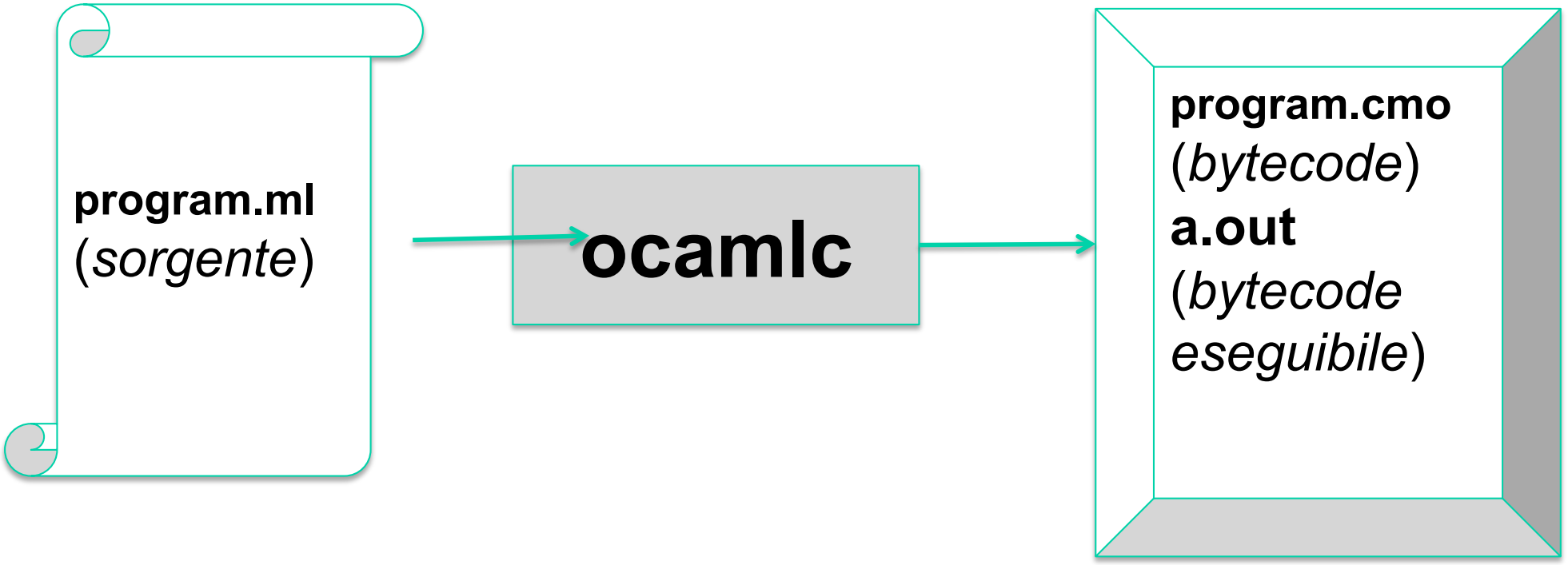
```
;; open Myset
let s1 = add 3 empty
let s2 = add 4 empty
let s3 = add 4 s1
let test() : bool = (member 3 s1) = true
;; run_test "Myset.member 3 s1" test
let test() : bool = (member 4 s3) = true
 ;; run_test "Myset.member 4 s3" test
```

Implementazione basata su liste

```
module MySet2: Set =
struct
    type 'a set = 'a list
    let empty : 'a set = []
    ...
end
```

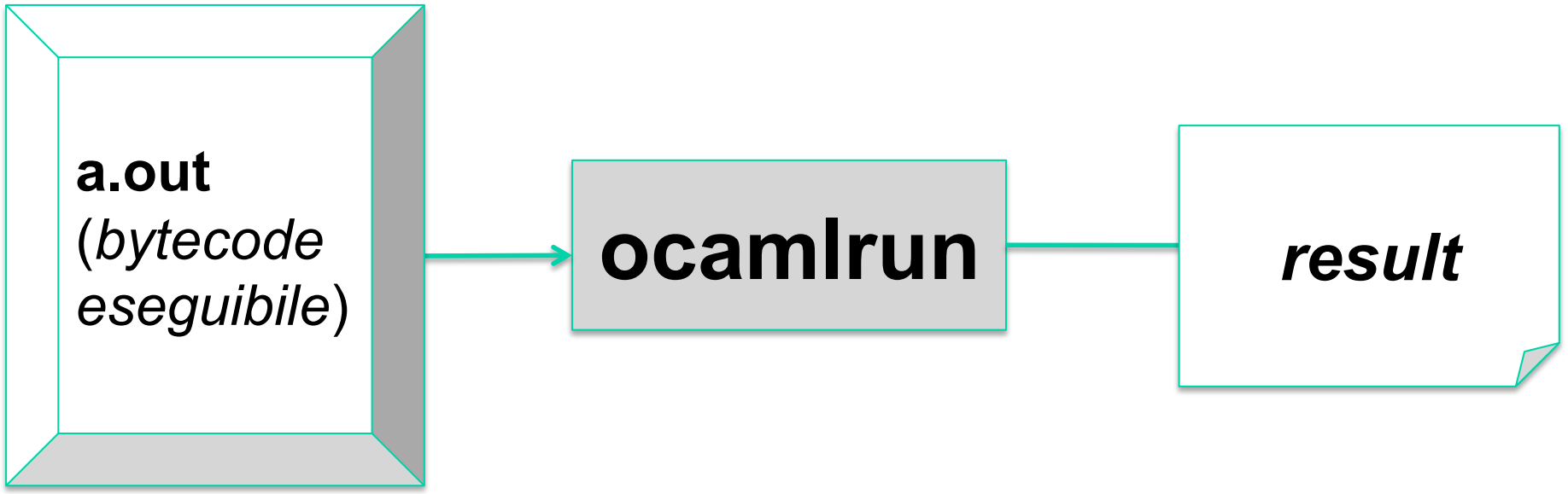
Una definizione concreta per il tipo Set

Compilare programmi OCaML



http://caml.inria.fr/pub/docs/manual-ocaml/comp.html

Eseguire bytecode OCaML



http://caml.inria.fr/pub/docs/manual-ocaml-400/manual024.html