Definizione di "NUOVI TIPI" in CAML

Titolo nota 24/09/2015

int
bool
'a list
→

C'è la possibilità di definire nuovi tipi a partire da quelli predefiniti!

Enumerare i valori del nuovo tipo
(in CAML questi nuovi valori sono "NOMI")

alternativa

```
type giorno = Lun | Mar | Mer | Gio | Ven | Sab | Dom;;
type giorno defined
```

Lun, Mar, Mer, ... sono i valori del nuovo tipo

Definizione di tipo enumerato i cui valori sono nuovi, non basati su valori di altri tipi.

Titolo nota 24/09/2015

```
type giorno = Lun | .... | Dom ;;
```

```
# Lun ;;
- : giorno = Lun
# Lun < Mar ;;
- : bool = true
# Mer < Lun ;;
- : bool = false
```

```
# let feriale x = Lun =< x & x <= Sab ;;
feriale : giorno -> bool = <fun>
# type g = Lun | Mar ;;
errore di tipo
```

Definire nuovi tipi che usano valori di tipi predefiniti
(usano quei valori mediante nuovi "COSTRUTTORI DI VALORI")

Titolo nota 24/09/2015

Supponete di voler definire un nuovo tipo che ha come valori $\mathbb{Z} \cup \mathbb{B}$. È possibile in CAML?

In accordo all'inferenza dei tipi la risposta è NO!!

```
type tag = int | bool ;;
```

non c'è questa possibilità perché il valore 5 apparterrebbe sia al tipo "int" che al tipo "tag"

Si può definire "tag" mediante "costruttori di valori"

```
type tag = Tagn of int | Tagb of bool ;;
type tag defined
```

Titolo nota 24/09/2015

```
# type tag = Tagn of int | Tagb of bool ;;
```

costruttori di valori. A partire da valori di tipo int e bool, rispettivamente, COSTRUISCONO valori di tipo tag.

```
# Tagn ;;
- : int -> tag = <fun>
```

```
# Tagb ;;
- : bool -> tag = <fun>
```

```
# Tagn 5 ;;
- : tag = Tagn 5
```

```
# Tagb true ;;
- : tag = Tagb true
```

Titolo nota 24/09/2015

```
# type 'a foo = Foo of 'a ;;
  type 'a foo defined

# Foo 5 ;;
  -: int foo = Foo 5

# Foo true ;;
  -: bool foo = Foo true
```

```
# Foo [3;4];;
  -: int list foo = Foo [3;4]
```

Potremmo definire nuovi tipi per operare su "programmi"

Espressioni numeriche (sintassi semantica)

Titolo nota 24/09/2015

Exp → Num | Exp Op Exp

Op → Add | Mul

Num → Zero | Succ (Num)

2 + 3

Succ (Succ (Zero)) Add

Succ (Succ (Succ (Zero)))

Definire un nuovo tipo CAML per rappresentare queste espressioni

```
# type num = Zero | Succ of num ;;
  type num defined
# type op = Add | Mul ;;
  type op defined
```

definizione di tipo ricorsiva

```
type exp = num | ...
```

Titolo nota 24/09/2015

```
# type num = Zero | Succ of num ;;
# type op = Add | Mul ;;
# Zero ;;
- : num = Zero
# Succ (Succ Zero) ;;
- : num = Succ (Succ Zero)
```

~~type exp = num ;;~~

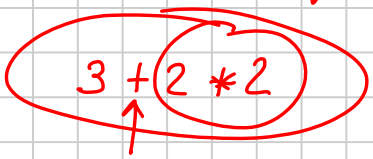
```
# Zero ;;
- : num =
- : exp =
```

```
type exp = Num of num |
           Exp of exp * op * exp ;;
                  Triple

# Zero ;;
- : num = Zero

# Num Zero ;;
- : exp = Num Zero
# Num ;;
- : num -> exp = <fun>
```

```
# Exp ;;
- : exp * op * exp
  -> exp = <fun>
```

Titolo nota 24/09/2015

```
type num = Zero | Succ of num ;;
type op = Add | Mul ;;
type exp = Num of num | Exp of exp * op * exp ;;
```




```
# Exp (Num Succ(Succ(Succ(Zero))), Add,
       Exp (Num Succ(Succ(Zero)), Mul, Num Succ(Succ(Zero))))
- : exp = ------
```

Titolo nota 24/09/2015

uso i costruttori di valori

```
let rec valn n = match n with
    Zero -> 0
  | Succ m -> valn m + 1 ;;
valn : nums -> int = <fun>
```

```
valn Succ (Succ Zero);;
- : int = 2
```

```
let valop o = match o with
    Add -> let f x y = x + y in f
  | Mul -> prefix * ;;
valop : op -> int -> int -> int = <fun>
```

(prefix +) la somma come funzione

```
(# prefix + ;;
- : int -> int -> int = <fun>
```

Titolo nota 24/09/2015

ML meta language (adatto a rappresentare altri linguaggi)

```
let rec valexp e = match e with
    Num n -> valn n
  | Exp (e1, o, e2) -> (valop o) (valexp e1) (valexp e2);;
```

valexp : exp -> int = <fun>

funzione int -> int -> int

3 + 2 * 2

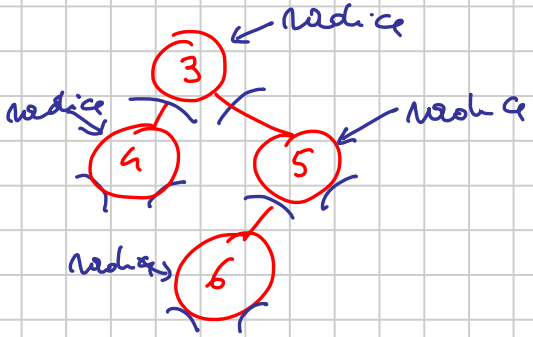
```
valexp (Exp (Num Succ(Succ(Succ Zero)), Add,
        Exp (Num Succ(Succ Zero), Mul, Num Succ(Succ Zero))));;
- : int = 7
```

Alberi binari

Titolo nota 24/09/2015

Albero binario è:

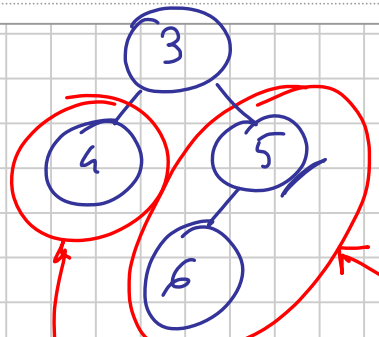
- un albero vuoto,
- una radice con due sottoalberi (sinistro e destro) che sono alberi binari,




```
type 'a btree = Void
              Node of 'a * 'a btree * 'a btree ;;
type 'a btree defined
```

Titolo nota 24/09/2015

valore di tipo int btree

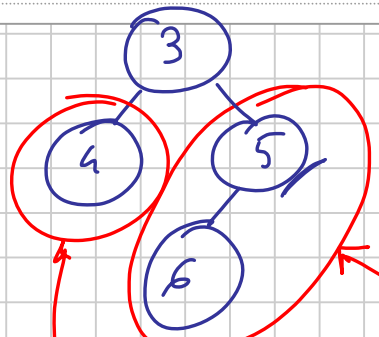


Node (3, Node (4, Void, Void),
Node (5, Node (6, Void, Void), Void))

```
# let rec member n bt = match bt with
        Void -> false
  | Node (x, lbt, rbt) when x = n -> true
  | Node (x, lbt, rbt) when x <> n -> (member n lbt) or (member n rbt);;
member : 'a -> 'a btree -> bool = <fun>
```

Titolo nota 24/09/2015

valore di tipo int btree



Node (3, Node (4, Void, Void),
Node (5, Node (6, Void, Void), Void))

```
# let rec member n bt = match bt with
      Void -> false
    | Node (x, lbt, rbt) when x = n -> true
    | Node (x, lbt, rbt) when x <> n -> if member x lbt then true
                                        else member x rbt;;
member : 'a -> 'a btree -> bool = <fun>
```

Titolo nota 24/09/2015

```
# let rec ins n bt = match bt with
      Void -> Node (n, Void, Void)
    | Node (x, lbt, rbt) ->
                Node (x, lbt, ins n rbt) ;;
```

ins : 'a -> 'a btree -> 'a btree = <fun>

ins 10 N(3, N(4,V,V), N(5, N(6,V,V), V))

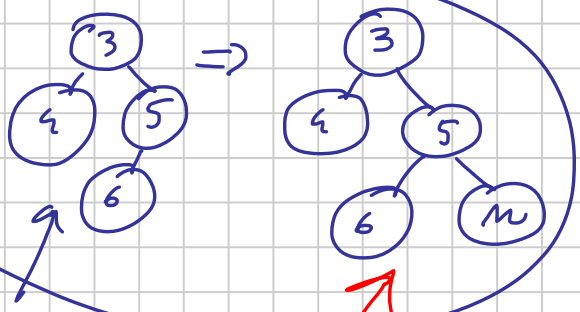
= N(3, N(4,V,V), ins 10 N(5, N(6,V,V), V))

= N(3, N(4,V,V), N(5, N(6,V,V), ins 10 V))

= N(3, N(4,V,V), N(5, N(6,V,V), N(10,V,V)))

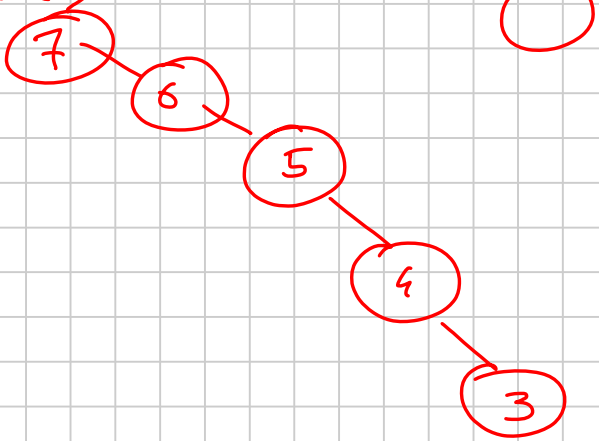
inserisce n come ultima foglia a destra

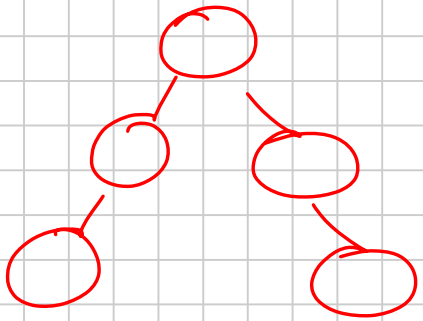
ins n



Titolo nota 24/09/2015

```
# let rec build l = match l with
      [] -> Void
    | x :: xs -> ms x (build xs);;

build : 'a list -> 'a btree = <fun>

# build [3;4;5;6;7]
```





```
let rec split l = match l with
      [] -> ([], [])
    | [x] -> ([x], [])
    | x :: y :: ys -> let (l1, l2) = split l
                      in (x :: l1, y :: l2);;
```

split : 'a list -> 'a list * 'a list = <fun>

```
let rec build l = match l with
      [] -> Void
    | x :: xs -> let (l1, l2) = split xs in Node(x, build l1, build l2);;
```

build : 'a list -> 'a btree = <fun>