LA NOZIONE DI STATO

Value-oriented programming

- La visione del frammento di Ocaml che abbiamo considerato descrivere perfettamente la nozione di "value-oriented programming": una espressione Ocaml denota un valore.
- L'esecuzione di un programma OCaml puo' essere vista come una sequenza di passi di calcolo (semplificazioni di espressioni) che alla fine produce un valore.

Immutable data structures

- Abbiamo introdotto la nozione di ADT in un paradigma di programmazione *puro*.
- Una computazione produce nuovi valori a partire dai valori iniziali
- Programmi operano su valori immutabili: uno volta associato un valore a un idetificatore questo valore non viene mai alterato durante il flusso di esecusione del programma
- Potrebbe non essere visibile. Perche?

Immutable data structure

- Operare con strutture dati immutabili introduce alcuni vantaggi.
- Dato che i programmi possono solo *leggere* dati e non *modificarli* diventa facile programmare programmi che operara in parallelo su strutture dati immutabili (non interferiscono).
- Non e' letteratura, Esempi:
  - Immutable Collections nel linguaggio di programmazione Scala, o in CloJure.

Mutable Data Structures

- Linguaggi di programmazione imperativi (esempio che avete visto il C) sono basati sul paradigma di programmazione in cui lo stato di una struttura dati e' mutabile.
- La possibilita' di modificare lo stato introduce una nuova visione computazionale e un nuovo modo di descrivere il comportamento operazionale di programmi.

Esempio

- Supponiamo di voler introdurre una struttura dati per descrivere e operare sulle coordinate di uno spazio a due dimensioni.
- Sostanzialmente si deve definire il tipo di dato astratto punto cartesiano, tipo di dato astratto mutabile.

```
type point = {mutable x: int, mutable y: int}

(*shift di coordinate cartesiane*)
let shift (p: point) (dx:int) (dy:int): unit =
p.x <-p.x + dx;
p.Y  <- p.y + dy

:
let p1 = {x=0; y=0}
let p2 = {x=17; y=17}
;; shift p1 12 13
;; shift p2 2 4
(* spostiamo i punti p1 e p2 nello sopazio cartesiano*)
```

La parola chiave **mutable** indica che i valori del tipo sono valori modificabili

Analisi

```
let f (p1:point) (p2: point) : int =
  p1.x <- 17;
  p2.x <- 42;
  p1.x
```

Risposta ovvia: dato che p1 e' mutabile la funzione f restuisce sempre il valore intero 17.

E' vero?

```
let p = {x = 0; y = 0} in
  f p p
(*I parametri attuali sono identici *)
```

Nella esecuzione della chiamata della funzione f
i nomi dei paramentri
p1 e p2 denoteranno il medesimo punto
(fenomeno denominato alias)

Pertanto il valore restituito sara' il valore intero 42!!!

Aliasing

- Il fenomeno dell'aliasing come quello mostrato in precedenza rende maggiormente complicato l'analisi del comportamento di programmi.
- Bisogna comprendere il modo in cui vengono fatte le modifiche delle strutture dati.

Abstract Stack Machine

- Abstract Stack Machine: modello computazionale per Ocaml che permette di descrivere la nozione di stato modificabile
- Modello astratto: nella seconda parte del corso esamineremo nel dettaglio gli aspetti relativi alla realizzazione dei linguaggi di programmazione.

AST per OCaml

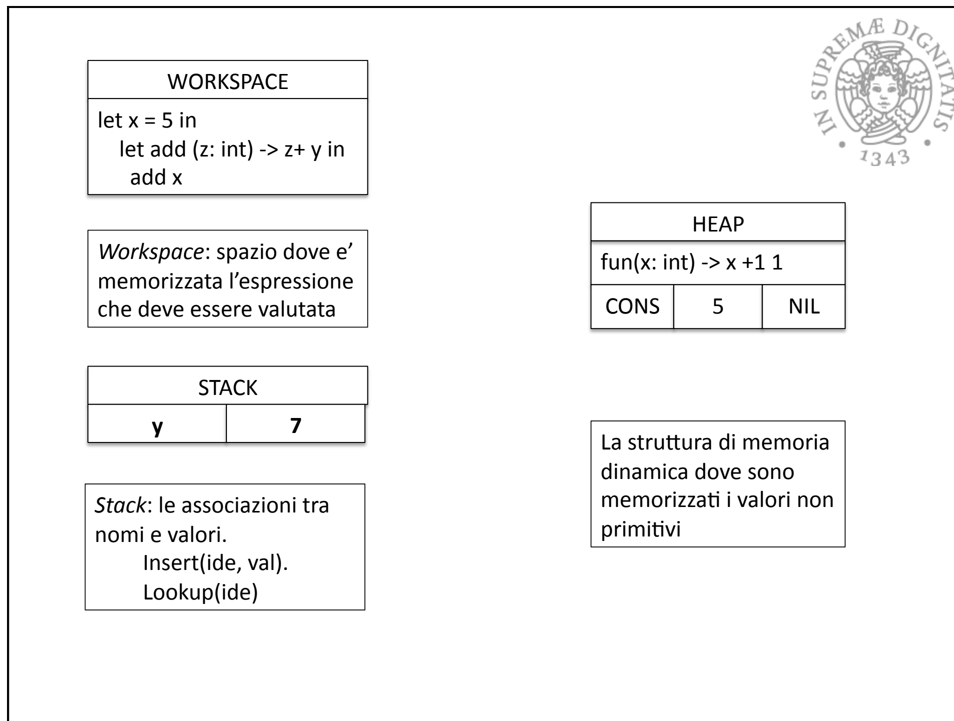
- Programmi sono espressioni
  - ✓E= let x = 5 in let s (y: int) -> y+1 in s x
- Una espressione calcola un valore.
  - ✓E ==> 6
- Valori *esprimibili*: valori che possono essere il risultato di una valutazione
  - ✓Interi: 0,1, …
  - ✓tuple di valori esprimibili: (1, (0,1))
  - ✓ funzioni: fun (x:int) -> x+1,
  - ✓costruttori applicati a valori 3:: [] = CONS(3 Nil)
- Binding: associazioni nome valore.

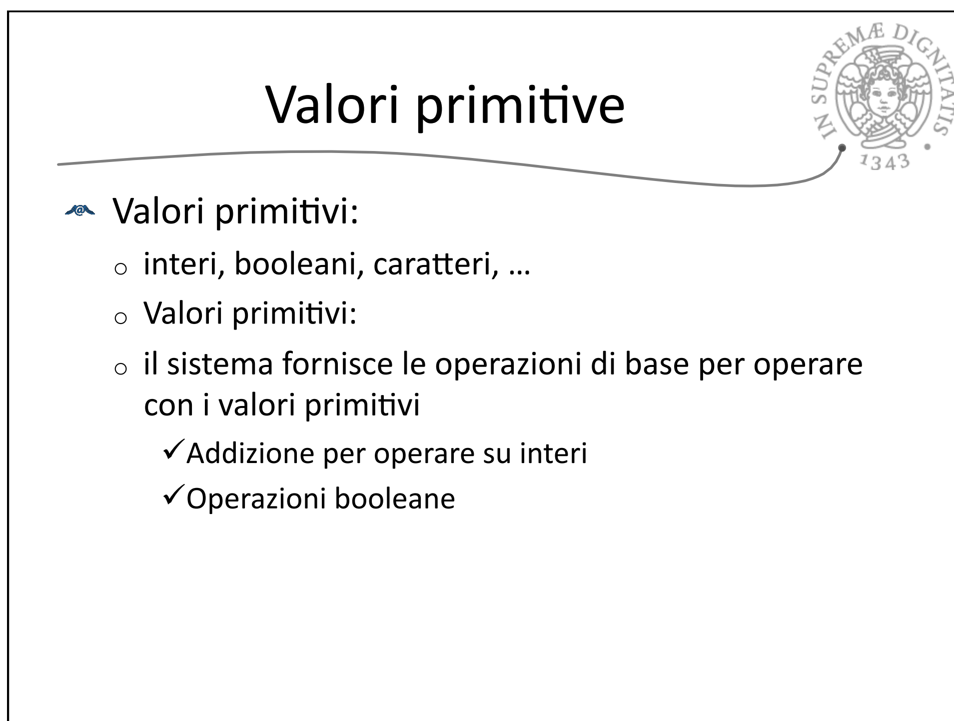
AST per OCaml

- Tre componenti
  - Lo spazio di lavoro (workspace) che contiene il programma che deve essere eseguito e le informazioni di controllo che guidano l'esecuzione
  - L'ambiente corrente (lo stack) con i legami attivi
  - La memoria dinamica (heap)
- Stato iniziale
  - Workspace contiene il programma completo
  - Heap e stack sono vuoti

AST

- Buon modello per comprendere come funzionano I programmi Ocaml
- Definisce in modo chiaro come sono gestiste le strutture dati
- Permette di trattare in modo omogeneo sia gli aspetti funzionali (strutture dati immutabili) che gli aspetti imperativi (dati mutabili)
- Visione semplificata della macchina stratta della realizzazione del linguaggio
  - Non trattiamo il caso di programmidi che restituiscono funzioni come valori: quale e' il problema.

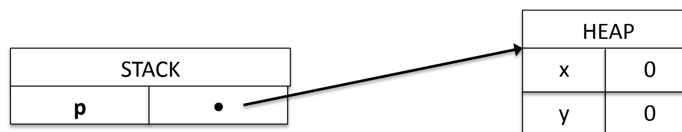

WORKSPACE
let x = 5 in
let add (z: int) -> z+ y in
add x
Workspace: spazio dove e' memorizzata l'espressione che deve essere valutata
STACK
y
7
Stack: le associazioni tra nomi e valori.
Insert(ide, val).
Lookup(ide)
HEAP
fun(x: int) -> x +1 1
CONS
5
NIL
La struttura di memoria dinamica dove sono memorizzati i valori non primitivi
IN SUPREMÆ DIGNITATIS
1343


Valori primitive
Valori primitivi:
interi, booleani, caratteri, …
Valori primitivi:
il sistema fornisce le operazioni di base per operare con i valori primitivi
✓Addizione per operare su interi
✓Operazioni booleane
IN SUPREMÆ DIGNITATIS
1343

Valori non primitivi

Valori non primitivi sono riferimenti a dati strutturati memorizzati nello heap. Il riferimento e' l'indirizzo della porzione di memoria dello heap in cui e' memorizzato la rappresentazione del valore

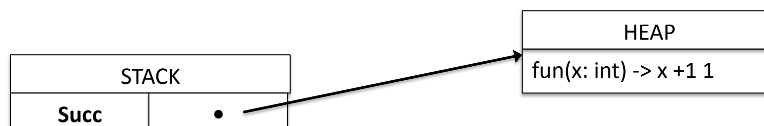
```
type point = {mutable x: int, mutable y: int}
let p  = {x=0; y=0}
```



Valori non primitivi

Valori non primitivi sono riferimenti a dati strutturati memorizzati nello heap. Il riferimento e' l'indirizzo della porzione di memoria dello heap in cui e' memorizzato la rappresentazione del valore

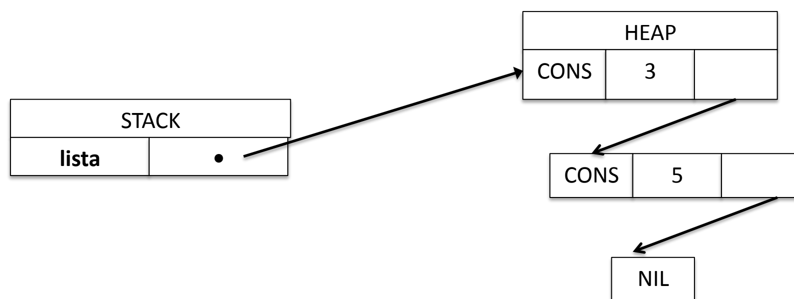
```
let Succ = fun(x:int) -> x + 1
```



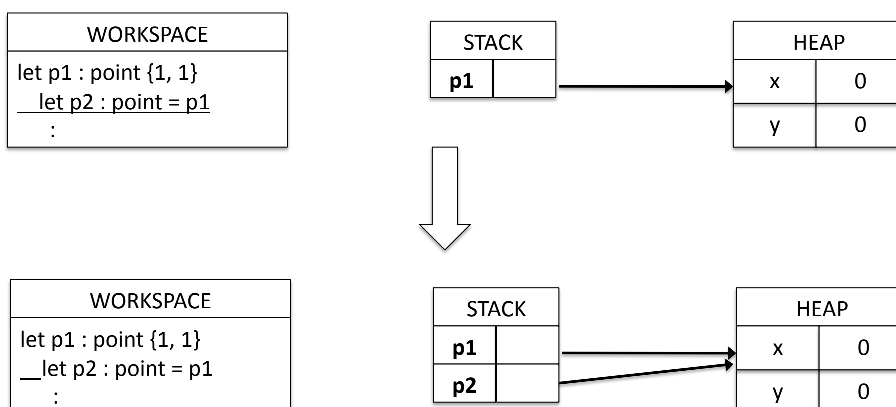
Valori non primitivi

Valori non primitivi sono riferimenti a dati strutturati memorizzati nello heap. Il riferimento e' l'indirizzo della porzione di memoria dello heap in cui e' memorizzato la rappresentazione del valore

Let lista = 3::[5]



ALIASING



Esercizi

Descrivere l’evoluzione della AST nel caso dei programmi

```
let x = 22 in
  let x = 2 + x in
   if x > 23 then 3 else 4


let rec append (l1: ‘a list) (l2: ‘a list): ‘a list =
  begin matcth l1 with
  | Nil --> l2
  | Cons(h,t) --> Cons(h, append t l2)
  end in

Let a = Cons(1, nil) in
 let b = Cons(2, Cons(3, Nil)) in
   append a b
```