

Tipi di Dato: Java vs. OCaml

Sommario: 8 Maggio, 2019

- **Il tipo di dato Iset in Java:**
 - > Uso di Package per ...
 - > Costruttori con Nomi Assegnati;
 - > Segnatura Data Centric in Java
 - > Additional: toString
- **Stato:**
 - > Vector<Integer>: AF, I
- **In OCaml**
 - > Uso di moduli
 - > Stato. int list: AF, I
 - > Tipizzazione esplicita delle operazioni
 - > Structure Sharing: List vs. Vector
 - > fold_left: High Order Programming
 - > Una diversa implementazione: Liste ordinate (AF ed I)
- **Esercizi**

Un Problema da risolvere: Un Programma da scrivere

- Esaminiamo il Testo per il Tipo di Dato Iset

```
package ISet2019;

import java.io.*;
import java.util.*;

/*
  Un insieme finito e' un tipo di dato strutturato, 'a set, della forma
  {x1,...,xn}, per n>=0, a componenti xi omogenei e contenente al piu'
  un'occorrenza di uno stesso valore, xi=xj iff i=j.
  I valori 'a set sono valori Immutable, sui quali possiamo operare me-
  diante le sole seguenti operazioni (in segnatura funzionale):
    isin: 'a -> 'a set -> bool
    union: 'a set -> 'a set -> 'a set
    inter: 'a set -> 'a set -> 'a set
    diff: 'a set -> 'a set -> 'a set
    card: 'a set -> int
    toString: 'a set -> string
  in aggiunta ai costruttori per:
    - insieme vuoto
    - insieme singoletto
  Si dia un'implementazione in Java di Iset per tipi di dato int set tale
  che:
  (1) Utilizzi una classe di nome Iset per l'implementazione
  (2) Lo stato della rappresentazione utilizzi Vector<Integer>
  (3) Non sovraccarichi la rappresentazione con duplicati di elementi.
*/
```

Uso di Package

- Uso di package per: Delimitare Scope delle Classi
 - non (dichiarate) `public`
- Per sola convenienza tipografica:
 - 1 package, 1 file, 1 classe dichiarata `public`
 - `main` appare come metodo della classe
- Per convenienza di progettazione e sviluppo:
 - 1 package, k file X k classi dichiarate `public`
 - `main` in una classe separata

```
package ISet;
import java.util.*;

public class Iset{
    ...
}

/* In file differenti per ogni class dichiarata public */
package ISet;
import java.io.*;

class Main{
    ...
}
```

Metodi: Segnatura Data Centric

- Programmazione OO è **Data Centric**
- I met. di istanza si applicano all'oggetto proprietario del met.
es.: `s1.union(s2).isin(3)`
 - `this` è un operando nel caso di metodi per operatori.

```
isin: 'a -> 'a set -> bool
union: 'a set -> 'a set -> 'a set
inter: 'a set -> 'a set -> 'a set
diff: 'a set -> 'a set -> 'a set
card: 'a set -> int
toString: 'a set -> string

...
*/
public class Iset{
    public boolean isin(int x);
    public Iset union(Iset y);
    public Iset diff(Iset y);
    public Iset inter(Iset y);
    public int card();
    public String toString();
}
```

Additional: toString

- **Additional:** Metodi che dovrebbe essere presenti in ogni class, dato il ruolo svolto nella programmazione con valori.
 - In effetti sono tutti (toString, equals, clone) definiti nella class Object ed ereditati oppure **overridden**
- **toString():** Deve fornire una presentazione concreta del valore descritto dall'oggetto.

```
Un insieme finito e' un tipo di dato strutturato, 'a set, della forma
{x1,...,xn}, per n>=0, a componenti xi omogenei e contenente al piu'
...
*/

public class Iset{
    ...
    public String toString(){
        String temp = rep.toString();
        temp = temp.substring(1,temp.length()-1);
        return "{" + temp + "}";
    }
}
```

Stato: Vector<Integer>

- **Stato:** Abbiamo implementato un tipo di dato strutturato mediante un tipo di dato strutturato già disponibile.
 - Pros: Uso di operazioni già definite sul tipo utilizzato
 - Cons: Uso di operazioni non "adeguate" (per costo o comportamento)
- In Alternativa. Una Struttura Multi-Componenti:

```
es.: int elem;  
     Iset otherOnes;  
     int size;
```

```
/*  
Un insieme finito e' un tipo di dato strutturato, 'a set, della forma  
{x1,...,xn}, per n>=0, a componenti xi omogenei e contenente al piu'  
un'occorrenza di uno stesso valore, xi=xj iff i=j.  
I valori 'a set sono valori Immutable, sui quali possiamo operare me-  
diante le sole seguenti operazioni (in segnatura funzionale):  
...  
*/  
public class Iset{  
    Vector<Integer> rep;  
    /*  
    AF(c) = {x1,...,xn} iff  
    ((n=c.rep.size())&&(forall 0<i<c.rep.size(), xi = c.rep.get(i-1)))  
    I(c) = (c.rep != null) && (forall i#j and 0<i,j<n+1, !xi.equals(xj))  
    */  
    Iset(){//costruttore  
        rep = new Vector<Integer>();  
    }  
    Iset(int x){//costruttore  
        rep = new Vector<Integer>();  
        rep.add(x);  
    }  
}
```

Sviluppo e Struttura del Programma

- **Abbiamo visto** l'API `Vector<E>` con le operazioni usabili per il tipo che useremo per l'implementazione.
- **Concentriamoci** sul Codice da scrivere
- **Poniamolo** su una sola Classe, un solo File.
- **Il file** in un'opportuna directory per la compilazione ed esecuzione.
- **Stato:** Codice per la definizione dello Stato dei valori Iset
- **Stato:** Funzioni AF ed I della rappresentazione
- **Costruttori:** Segnatura e Codice
- **Operazioni:** Conviene definire `toString` , come prima operazione
- **Definita `toString`:** Possiamo utilizzarla per verificare il codice dato per i costruttori
- **Test:** Richiede di introdurre il metodo statico `main`
- **Test:** Richiede anche, di definire il corpo di `main` con un codice che introduce anzitutto la costruzione dei valori più semplici.
- **Operazioni:** Possiamo procedere con le successive operazioni quali `isin`
- **Test:** Ad ogni operazione procedere con aggiunta di codice nel `main` per il Test del codice inserito.
- **Esaminiamo il Programma scritto nel Listing `Iset2019.java`**
- **Abbiamo terminato:** Tutte le operazioni sono state inserite, definite ed controllate con un Primo Test.
- **Possiamo:** Analizzare la Soluzione data e/o confrontarla con altre se disponibili
- **Alla fine:** dobbiamo riconsiderare la struttura data al programma e nel caso partizionarla in strutture autonome.

In OCaml: Uso di Moduli

```
module Iset =
struct
  type iset = int list
  (*
  AF(c) = {x1,...,xn} iff (c=[x1,...,xn] and n>=0)
  I(c) = (c = [x1,...,xn] and n>=0 and (forall i#j and 0<i,j<n+1, xi#xj))
  *)

  let (empty: unit -> iset) = fun () -> []
  let (sing: int -> iset) = fun n -> [n]
  let (isin: int -> iset -> bool) = fun x s -> List.exists (fun x -> x = n) s
  let rec (union: iset -> iset -> iset) = fun s1 s2 ->
    (match s1 with
     |[] -> s2
     |n::ns when isin n s2 -> union ns s2
     |n::ns -> n :: union ns s2)
  let rec (inter: iset -> iset -> iset) = fun s1 s2 ->
    (match s1 with
     |[] -> []
     |n::ns when isin n s2 -> n :: (inter ns s2)
     |_::ns -> inter ns s2)
  let rec (diff: iset -> iset -> iset) = fun s1 s2 ->
    (match s1 with
     |[] -> []
     |n::ns when isin n s2 -> diff ns s2
     |n::ns -> n :: diff ns s2)
  let (toString: iset -> string) = fun s ->
    (match s with
     |[] -> "{}"
     |n::[] -> "{" ^ (Printf.sprintf "%d" n) ^ "}"
     |n::ns -> let g x s = (Printf.sprintf ", %d" x)^s in
                let rest = List.fold_right g ns "}" in
                "{" ^ (Printf.sprintf "%d" n) ^ rest)
end;;
```


Stato: int list

- **Stato:** Abbiamo implementato un tipo di dato strutturato mediante un tipo di dato strutturato già disponibile.
 - Il tipo scelto: 'a list con 'a istanziata sul tipo int
 - Le liste sono un dato strutturato tipico dei Funzionali
 - Vector e Liste ruolo simile Ma comp. diverso (v. dopo)
 - In Alternativa. Una Struttura Multi-Componenti:

```
(*  
Un insieme finito e' un tipo di dato strutturato, 'a set, della forma  
{x1,...,xn}, per n>=0, a componenti xi omogenei e contenente al piu'  
un'occorrenza di uno stesso valore, xi=xj iff i=j.  
I valori 'a set sono valori Immutable, sui quali possiamo operare me-  
...  
)  
  
module Iset =  
struct  
  type iset = int list  
  (*  
   AF(c) = {x1,...,xn} iff (c=[x1,...,xn] and n>=0)  
   I(c) = (c = [x1,...,xn] and n>=0 and (forall i≠j and 0<i,j<n+1, xi≠xj))  
  *)  
end
```

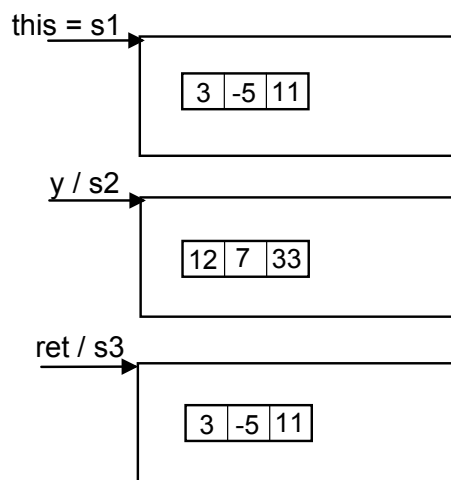
Operazioni: Dichiarazione esplicita del tipo voluto

- **Dichiarazione del Tipo:** OCaml inferisce il tipo più generale di ogni espressione "ben tipata"
 - OCaml usa equivalenza strutturale:
`iset sta per int list`
- Le operazioni (eccetto ...) se non esplicitamente tipate avrebbero tipo 'a list al posto di iset
- Quando esplicitamente tipate, OCaml controlla che il tipo dichiarato sia istanza del tipo inferito.

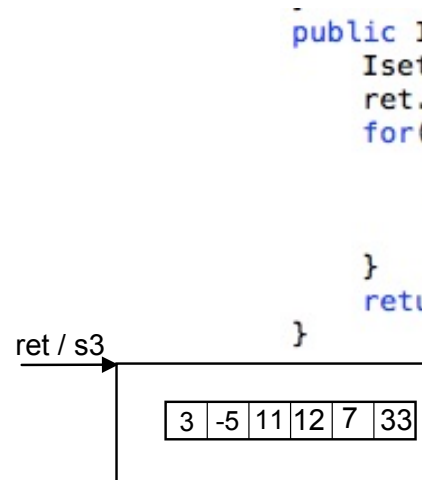
```
let (empty: unit -> iset) = fun () -> []
let (sing: int -> iset) = fun n -> [n]
let (isin: int -> iset -> bool) = fun n s -> List.exists (fun x -> x = n) s
let rec (union: iset -> iset -> iset) = fun s1 s2 ->
  (match s1 with
   | [] -> s2
   | n::ns when isin n s2 -> union ns s2
   | n::ns -> n :: union ns s2)
```

Vector e List hanno comportamento diverso: Vector<Integer>

- Vector sono Valori MUTABLE in struttura e nei componenti
- Quando usati come IMMUTABLE devono essere create copie indipendenti
- L'invocazione di `s1.union(s2)` conduce alla costruzione ed evoluzione del valore `s3`, binding di `ret`, sotto.



`ret.rep.addAll(rep);`

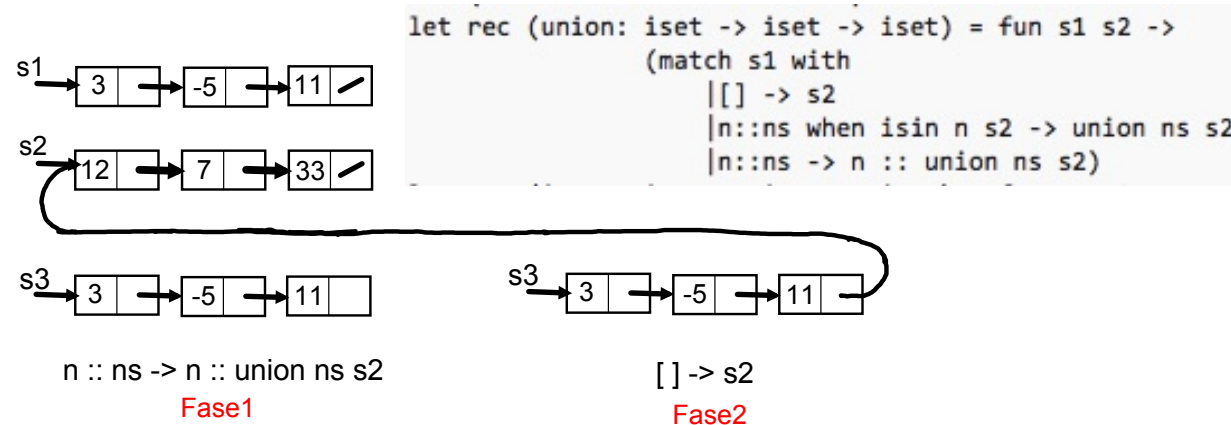


`for(int i=0;i<y.rep.size();i++)`

```
public Iset union(Iset y){
    Iset ret = new Iset();
    ret.rep.addAll(rep);
    for(int i=0;i<y.rep.size();i++){
        int e = y.rep.get(i);
        if (!ret.rep.contains(e))
            ret.rep.add(e);
    }
    return ret;
}
```

List usano Structure Sharing: int list

- List di OCaml sono Valori IMMUTABLE in struttura e nei componenti
- Quando usati come IMMUTABLE mostrano in pieno questa loro proprietà (vedi, operatori: ::, @)
- L'invocazione di `union(s1,s2)` conduce alla costruzione del valore `s3` sotto, mostrata nelle 2 fasi richieste dall'invocazione.
- **Structure Sharing** Alla fine `s3` condivide con `s2` la struttura di `s2`



List.fold_left: High Order, Iteration, Tail-recursion

```
let (toString: iset -> string) = fun s ->
  (match s with
   | [] -> "{}"
   | n::[] -> "{" ^ (Printf.sprintf "%d" n) ^ "}"
   | n::ns -> let g s x = s^(Printf.sprintf ", %d" x) in
               let rest = List.fold_left g "" ns in
               "{" ^ (Printf.sprintf "%d" n) ^ rest ^ "}")
```

- **High Order:** Nel primo argomento (operazione binaria)
`List.fold_left: ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a`
- **Iteration:** Ripete per ogni x in els l'applicazione di g al valore ret, calcolato al passo precedente, e ad x
`let rec fold_left (g: 'a -> 'b -> 'a) (ret: 'a) (els: 'b list): 'a =
 match els with
 | [] -> ret
 | x :: xs -> fold_left g (g ret x) xs`
- **Tail-Recursive:** Un solo AR dove rimpiazziamo, ad ogni invocazione successiva, il binding di ret con il valore di (g ret x) e il binding di els con il valore di xs.

Una diversa implementazione: Liste ordinate (AF ed I)

```
module Iset2 =
struct
  type iset = int list
  (*
  AF(c) = {x1,...,xn} iff (c=[x1,...,xn] and n>=0)
  I(c) = (c = [x1,...,xn] and n>=0 and (forall 0<i<n, xi<xi+1))
  *)

  let (empty: unit -> iset) = fun () -> []
  let (sing: int -> iset) = fun n -> [n]
  let (isin: int -> iset -> bool) = fun n s -> List.exists (fun x -> x = n) s
  let rec (union: iset -> iset -> iset) = fun s1 s2 ->
    (match (s1,s2) with
     |(_,[]) -> s1
     |([],_) -> s2
     |(n1::ns1, n2::ns2) when n1<n2 -> n1 :: union ns1 s2
     |(n1::ns1, n2::ns2) when n1=n2 -> n1 :: union ns1 ns2
     |(n1::ns1, n2::ns2) -> n2 :: union s1 ns2)

  let rec (inter: iset -> iset -> iset) = fun s1 s2 ->
    (match (s1,s2) with
     |(_,[]) -> s1
     |([],_) -> s2
     |(n1::ns1, n2::ns2) when n1<n2 -> inter ns1 s2
     |(n1::ns1, n2::ns2) when n1=n2 -> n1 :: inter ns1 ns2
     |(n1::ns1, n2::ns2) -> inter s1 ns2)

  let rec (diff: iset -> iset -> iset) = fun s1 s2 ->
    (match (s1,s2) with
     |(_,[]) -> s1
     |([],_) -> s2
     |(n1::ns1, n2::ns2) when n1<n2 -> n1 :: diff ns1 s2
     |(n1::ns1, n2::ns2) when n1=n2 -> diff ns1 ns2
     |(n1::ns1, n2::ns2) -> diff s1 ns2)

  let (toString: iset -> string) = fun s ->
    (match s with
     |[] -> "{}"
     |n::[] -> "{" ^ (Printf.sprintf "%d" n) ^ "}"
     |n::ns -> let g s x = s^(Printf.sprintf ", %d" x) in
      let rest = List.fold_left g "" ns in
      "{" ^ (Printf.sprintf "%d" n) ^ rest ^ "}")

end;;
```

Esercizi

- Nella slide "Stato: Vector<Integer>", abbiamo menzionato una struttura Multi-Componenti a 3 componenti per lo stato del tipo di dato Iset in java.
Si forniscano le funzioni AF ed I per tale stato sempre nella richiesta di "non sovraccaricamento" della rappresentazione. Si mostri come andrebbe riformulata la definizione della classe Iset per la parte mostrata nella figura inclusa in tale slide.
- Nella slide "Stato: Vector<Integer>", abbiamo menzionato l'uso di Strutture Multi-Componenti per l'implementazione di nuovi tipo di dato. Si proponga una di tali strutture per lo stato del tipo Iset in Java e si mostri come andrebbe riformulata la definizione della classe Iset nella parte mostrata nella figura inclusa in tale slide.
- Si svolga l'esercizio precedente nel caso di OCaml e limitatamente alla riformulazione dl modulo Iset di slide "In OCaml: Uso di Moduli nella definizione di: Tipo, funzioni AF ed I, costruttori empty e sing.
- Nella slide "Operazioni: Dichiarazione esplicita del tipo voluto abbiamo visto come imporre un tipo nella dichiarazione delle operazioni del modulo Iset.
 - (1) Si carichi il modulo Iset e si verifichi che i tipi assegnati siano quelli attesi e dichiarati.
 - (2) Si definisca un modulo IsetS ottenuto da Iset rimuovendo tutte le signature introdotte per i tipi delle operazioni. Si carichi il modulo IsetS e si commentino le signature ottenute per le nuove operazioni confrontandole con quelle precedentemente ottenute.
 - (3) Si considerino le seguenti espressioni:

```
let s1 = Iset.sing(5);;  
let l1 = [1;5;1;5];;  
Iset.union s1 s1;;  
Iset.union s1 l1;;
```

e si dica quale comportamento ci aspettiamo dalla loro valutazione, commentando adeguatamente le ragioni della risposta data.
 - (4) Si verifichi (3) eseguendo la valutazione delle espressioni.
- Si definisca in OCaml un tipo astratto per il tipo di dato iset. In particolare:
 - (1) Si fornisca un modulo signature, ISET, per la relativa API;
 - (2) Si mostri come deve essere modificata la definizione del modulo Iset (in slide "In OCaml: Uso di Moduli") per usarlo come un ADT per ISET.
 - (3) Si valutino ora le espressioni in esercizio precedente e si confrontino i comportamenti ottenuti nei due casi.

Altri Esercizi in EserciziL14.pdf

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ 🔍 ↺ 14/14