Mini linguaggio funzionale

B.Pierce

Types and Programming Languages

R.Harper

Practical Foundations for Programming Languages

Teoria dei Linguaggi di Programmazione

- **Scopo** = descrivere il comportamento dei programmi
  - in modo **preciso – formale**

- Semantica operazionale
- Semantica denotazionle
- Semantica assiomatica

per **ragionare** sui programmi!

- Analisi statica e Verifica di programmi
  - sistemi di tipi, logiche temporali, logiche causali, teorie equazionali, interpretazione astratta, …

Linguaggi funzionali

Functional programming in **Java8**

- "with these facilities in Java we can write concise, elegant and expressive code with fewer error"

- "we can use this to easily enforce policies and implement common design patterns with fewer lines of code"

- "the new way of programming in Java has been around for decades in other languages"

Why then only now?? And what about OO?

The Free Lunch is over. H. Sutter 2005

" la grande rivoluzione del software degli ultimi 20 anni è stato il passaggio dalla programmazione *strutturata* alla programmazione *object-oriented*. "

OOP c'era già dal '60 con Simula, ma dal '90 che è il paradigma dominante, perché?

The Free Lunch is over. H. Sutter 2005

" la grande rivoluzione d[illegible] il passaggio dalla programmazione [illegible] *object-oriented*. "

ci vuole un catalizzatore

OOP c'era già dal '60 [illegible] dominante, perché?

- L'**industria** chiedeva di scrivere sistemi software sempre più grandi, che risolvessero problemi sempre più vasti usando risorse CPU e di memoria sempre maggiori.
- La capacità di **astrazione** e di **gestione delle dipendenze** dei OOL li hanno resi indispensabili per lo sviluppo software su larga scala, economico, affidabile riusabile.
  - e.g. si definisce una API senza definirne l'implementazione: un **numero fissato di operazioni con un numero illimitato di implementazioni** (es. GUI widgets)
  - OOP oggi e` essenziale per strutturare i sistemi software

The Free Lunch is over. H. Sutter 2005

è dagli anni '60 che si fa **concorrenza**
…ma la concorrenza è difficile….

ci vuole un catalizzatore

- ormai costruiamo solo hardware parallelo (multicore, cloud, cluster)
- *Big data analysis*
- ci vogliono ***modelli*** e ***linguaggi*** per la concorrenza e la distribuzione
  - high-level programming models (e.g. reactive manifesto, actor model, data streaming..)
  - **primitive** e **pattern** di programmazione (e.g. future/promises…)
  - devono **interoperare** con le altre feature del linguaggio!

W le funzioni !

From *How to do* to *What to do*

```
boolean found = false;
for(String city : cities) {
  if(city.equals("Chicago")){
    found=true;
    break;
  }
}
System.out.println("Found?" + found);
```

- No mutable variables
- No iteration
- Code closely trails the business intent
- Less error prone
- Easier to understand and maintain

```
System.out.println("Found Chicago?" +
                                cities.contains("Chicago"));
```

method ***contains*** *is not new in Java*

The *evolution* of iteration

```
Person[] people = ...
int maxAge=-1;
for(int i=0; i<people.length; i++)
  if(people[i].getGender()==MALE && people[i].getAge()>maxAge)
    maxAge=people[i].getAge();
```

```
Collection<Person> people = ...
Iterator<Person> it=people.iterator();
int maxAge=-1;
while(it.hasNext()){
  Person p=it.next();
  if(p.getGender()==MALE && p.getAge()>maxAge)
     maxAge=p.getAge();
}
```


The *evolution* of iteration

```
Collection<Person> people = …
int maxAge=-1;
for(Person p : people)
  if(p.getGender()==MALE && p.getAge()>maxAge)
    maxAge=p.getAge();
```

- No mutable **maxAge**
- Nicely composed
- Easier to enhance or change the logic
- Iteration controlled by library methods
- Efficient: lazy evaluation loops
- Easy to parallelize (no shared **maxAge**)

```
Collection<Person> people = ...
final int maxAge= people.stream()
                        .filter(p -> p.getGender()==MALE)
                        .mapToInt(p -> p.getAge())
                        .max();
```


Linguaggi funzionali

- no assegnamenti
- variabili
  - non rappresentano aree di memoria
  - ma rappresentano valori

valori **immutabili** che vengono ***trasformati***

IMPERATIVE program:
a set of **instructions** to be **executed**

FUNCTIONAL program:
a set of **expressions/terms** to be **evaluated**

strutture dati persistenti

`sort(list)` ottimizzazioni

Linguaggi funzionali

- no assegnamenti, valori **immutabili** che vengono ***trasformati***
- non ci sono side-effects
  - la chiamata di una funzione puo' essere sostituita con il suo risultato
    referenatial transparency

    la funzione ***comunica con il contesto*** solo tramite i parametri e i valori restituiti

    - "more reliable and more reusable"
    - eseguire **`f(exp)`** prima, dopo, o contemporaneamente a **`g(exp1)`** non cambia
    - tutto ciò che entra ed esce da una funzione è controllato dal type checker

      è più probabile che eventuali **errori logici** si riflettano in **errori di tipo**, quindi evidenziati dal compilatore!

Linguaggi funzionali

- no assegnamenti, valori **immutabili** che vengono ***trasformati***
- non ci sono side-effects
  - la chiamata di una funzione puo' essere sostituita con il suo risultato **referenatial transparency**
  - rifattorizzabile
- uso di ricorsione
- le funzioni sono valori *first-class*
  - ***comportamenti aggregabili*** con funzioni higher-order
  - un linguaggio con **buoni "combinatori" di funzioni** supporta la *decomposizione dei problemi* e il *riuso del codice*

funzione = comportamento

Linguaggi funzionali

- no assegnamenti, valori **immutabili** che vengono ***trasformati***
- non ci sono side-effects
  - la chiamata di una funzione puo' essere sostituita con il suo risultato **referenatial transparency**

```
people.stream()
      .filter(p -> p.getGender()==MALE)
      .mapToInt(p -> p.getAge())
      .max();
```

funzione = comportamento

- le funzioni sono valori *first-class*
  - ***comportamenti aggregabili*** con funzioni higher-order
  - un linguaggio con **buoni "combinatori" di funzioni** supporta la *decomposizione dei problemi* e il *riuso del codice*

S.Crafa AA 17/18

Sintassi di $\mathcal{L}$

$x \in Var \qquad n \in Num$

Termini M, N	::=	x	variabili
	|	n | true | false	costanti intere e booleane
	|	$M + M \mid M - M$	operazioni intere
	|	if M then M else M	condizionale
	|	fn $x.M$	dichiarazione di funzione
	|	MM	applicazione di funzione

Programma = un termine *chiuso* M

Esecuzione del programma = trovare il **valore** del termine (espressione) M

semantica

Sintassi di $\mathcal{L}$

Esempi di termini (anche programmi):

3+2	fn x.x
fn x.3	fn x.x+1
fn x.x+1 3	fn x.fn y.x+y
fn x.2+x	fn x.(fn y.y+x 2)
M=fn x.fn y.(x y)	(M fn z.z) 5
3+false	if 2 then fn x.x + x else 0

Sintassi di $\mathcal{L}$

((fn x.fn y. (fn z. z*z) x + (fn z. z*z) y) 3) 4

Scala

```
def square (z:Int):Int = z*z
def sumOfSquare(x:Int,y:Int):Int = square(x)+square(y)
sumOfSquare(3,4)
```

→ square(3) + square(4)
→ 3*3 + square (4)
→ 9 + square(4)
→ 9 + 4*4
→ 9+16
→ 25