Introduzione alla Programmazione Funzionale

Luca Abeni

April 27, 2017

Paradigmi di Programmazione

- Programmi sviluppabili usando diversi paradigmi
  - Imperativo: computazione come modifica di stato
  - Funzionale: computazione come riduzione
  - ...
- Paradigma imperativo
  - Variabili modificabili: ambiente associa nome a variabile, memoria associa variabile a valore
  - Costrutto fondamentale: assegnamento
    - Modifica associazione fra variabile e valore
    - R-Value “contenuto” nella variabile
  - Direttamente mappabile su macchina di Von Neumann

Paradigma di Programmazione Funzionale

- Paradigma funzionale $\rightarrow$ no stato / variabili modificabili
  - No variabili modificabili $\Rightarrow$ no assegnamento
    - Ambiente, ma no memoria
  - Solo espressioni e funzioni (no comandi)
  - Computazione come riduzione / riscrittura di espressioni
    - Invece che modifica dello stato...
- Riduzione??? Riscrittura di espressioni???

Computazione Funzionale

- No stato modificabile $\rightarrow$ no iterazione (loop)!
  - Basata su ripetere operazioni fino a che un predicato è vero
  - Predicato: funzione booleana dello stato... Stato non modificabile $\Rightarrow$ predicato sempre vero o sempre falso $\Rightarrow$ iterazione non ha senso!
- Invece che iterazione, Ricorsione!
- Esempi di linguaggi funzionali: ML (varie incarnazioni), Lisp (varie incarnazioni - Scheme), Haskell, ...
  - Fondamento teorico / matematico: λ-calcolo!

Funzioni: Definizione ed Applicazione

- Funzione: relazione fra dominio e codominio che associa ad un elemento del dominio un solo elemento del codominio
  - $f : \mathcal{X} \to \mathcal{Y}$
  - $f \subset \mathcal{X} \times \mathcal{Y} : (x, y_1) \in f \wedge (x, y_2) \in f \Rightarrow y_1 = y_2$
  - $(x, y) \in f \to y = f(x)$
- $f(x)$: significato ambiguo
  - $f(x) = x^2$: definizione di $f()$
  - $f(3)$: applicazione di $f()$ al numero 3
  - Stessa sintassi ($f(x)$) per esprimere la definizione e l'applicazione di una funzione?

Definizione o Applicazione?

- In matematica, il significato di "$f(x)$" dipende dal contesto...
  - Vedi "$f(x) = x^2$" vs "$f(3)$"
- ...In un linguaggio di programmazione, vorremmo una semantica univoca!
  - La distinzione fra definizione ed applicazione deve essere chiarita dalla sintassi...
  - ...Non dal contesto!!!
- Distinguere definizione da applicazione:
  - In ML, `fn` per definizione
  - In C, definizione con prototipo seguito da blocco di codice
  - ...

Definizione di Funzioni

- In linguaggi di programmazione, sintassi per distinguere definizione da applicazione
  - Esempio: in ML, **fn** x => x * x; definisce una funzione che calcola x^2
- In linguaggi funzionali, le funzioni sono *valori esprimibili*
  - Funzioni come risultato di espressioni (esempio stupido: espressione `fn`)
  - Nota: vere funzioni (== chiusure), non puntatori a funzione (in cui manca ambiente!)
  - Esempio: funzione derivata per capire meglio

Funzioni Anonime

- **val** f = **fn** x => x * x; definisce un simbolo `f` associato alla funzione che calcola x^2
  - Nota: sul libro, definizione leggermente imprecisa di `fun` (è `val rec`, non `val`!)
- Conseguenza dell'esistenza di un costrutto per definire funzioni (`fn` in Standard ML) : si possono definire anche *funzioni anonime*
  - "**fn** x => x * x;", senza "**val** f =" davanti
- Curiosità: esiste qualcosa di simile anche in C++:

```
[](int x) {
  return x * x;
};
```

- Vedere funzione derivata in C++

Applicazione di Funzioni

- Differenti linguaggi definiscono differenti sintassi
  - `f(2);`
  - `(f 2)`
  - `f 2;`
  - Nota: in ML tutte le espressioni di qui sopra sono valide...
- A differenza di matematica, chiara differenza fra definizione ed applicazione di una funzione!

Applicazione di Funzioni Anonime!

- Interessante conseguenza: si possono applicare anche funzioni anonime
  - (**fn** x => x * x) 3;
- Ancora, anche in C++:

```
[](int x) {
  return x * x;
}(3)
```

Esecuzione Come Valutazione

- Programma funzionale: composto da espressioni
- "Eseguito" valutando le espressioni
- Esempio: fattoriale!

```
unsigned int fact(unsigned int n)
{
  return n == 0 ? 1: n * fact(n - 1);
}
```

  - Notare "if aritmetico" (`p ? a : b`)

- `fact(4)` = ?

Esempio di Valutazione

```
fact(4) = ...
fact(4) = (4 == 0) ?  1 :  4 * fact(3) =
4 * fact(3) =
4 * ((3 == 0) ?  1 :  3 * fact(2)) = 4 *
3 * fact(2) =
4 * 3 * ((2 == 0) ?  1 :  2 * fact(1)) =
4 * 3 * 2 * fact(1) =
4 * 3 * 2 * ((1 == 0) ?  1 :  1 *
fact(0)) =
4 * 3 * 2 * 1 * 1 = 24
```

Alcune Osservazioni

- Per essere puramente funzionale, if aritmetico!

```
unsigned int fact(unsigned int n)
{
  return n == 0 ? 1: n * fact(n - 1);
}
```

vs

```
unsigned int fact(unsigned int n)
{
  if (n == 0) {
    return 1;
  } else {
    return n * fact(n - 1);
  }
}
```

- Esecuzione come valutazione/sostituzione possibile solo in assenza di effetti collaterali