Programmazione funzionale

Presentazione 2015-2016

Camillo Fiorentini
Alberto Momigliano

Dimenticate (quasi) tutto quello che sapete di programmazione

Organizzazione

- **Sito del corso**: ariel
- **Lezioni:**
  - Lunedì, 09:30 – 12.00, laboratorio tau.
  - Giovedì, 13:30 – 16.30, laboratorio sigma.
    - Non faremo sempre più di 2 ore, solo quando facciamo lab
- **Modalità di esame**:
  - Tutti gli esami su PC
  - 2 compitini in itinerere per chi segue oppure …
  - esame scritto in appello standard
    - Possibilità di un progetto per alzare il voto

Regole

- *Progetto* per alzare il voto: da consegnare entro **30** giorni dalla pubblicazione del voto
- *Esami*
  - Chi è *gravemente* ***insuff*** (<=12) nel primo compitino non può fare il secondo.
  - Gravemente *insuff* in appello standard deve fare **salto di appello**
  - Dopo 2 ritiri/no show in esame standard, si fa **salto**
  - Appelli con meno di 3 persone sono cancellati

Libro di Testo

- *Functional Programming using F#*, Michael R. Hansen and Hans Rischel, CUP (in inglese)
  - Potete portarvi il libro agli esami, quindi compratelo!
- Altri riferimenti su pagina web del corso, es:
  - Pagina wiki sulla programmazione in F#
  - Blog: F# for fun and profit
  - Bob Harper. Programming in Standard ML

Software

- **F# sotto Windows**: *Visual Studio 2013* su *UniCloud* – *registratevi!!*
- win@di.unimi.it da account studentesco oppure scaricatevi VS 2015 community version
- **F# sotto Linux**:
- *Mono:* open source version of .NET
  - MonoDevelop come IDE
    - Scaricate F# lang binding
  - Altre IDE: Tsunami e Xamarin
  - ottimo supporto emacs

Corsi collegati/propedeutici

- Ovviamente *programmazione*. Auspicabile *algoritmi*, meglio se avete anche qualche esperienza di Prolog, come da *Intelligenza Artificiale*
- Il corso di Cazzola *Linguaggi di Programmazione"* ha un certo overlap per le prime 3-4 lezioni
- Pubblicità: curriculum magistrale "*Metodi e modelli per la progettazione e sviluppo del software*"

Why FP?

Elevator pitch 1

- (Tratto da: “How to make money with FP”, by John Amstrong)
- You can write program *quicker*:
  - First to market
- Programs are *shorter* and with *less errors*
  - Reduced costs for maintainence

  - Why did Facebook bought WhatsApp for $19 billions?

Elevator pitch 2

- With only **55** engineers and a **billion** users, one **WhatsApp** developer supports ca **20 million** active users, a ratio unheard of in the industry.
  - WhatsApp's support team is even smaller
- "This crew has built a reliable, low-latency service that processes 50 billion messages every day across seven platforms using **Erlang**"
  - a (cuncurrent) FP language

IMP vs FP

- Un programma imperativo consiste essenzialmente di una sequenza di **comandi** che **assegnano variabili**
- Istruzioni come: *loop, jump, branching*, astratti come *for, while, switch* etc
- Vi sono altra astrazioni, come *procedure*, *oggetti* etc., ma alla fine si programma **modificando variabili in memoria**

IMP: problemi

- Visione esplicita della memoria (*aliasing*, *null pointers* etc.) prona all'errore e ostacolo alla concorrenza
- Ordine di esecuzione fisso (parallelismo?)
- Poca astrazione (es aritmetica dei puntatori)
- Semantica non sempre precisa: 191 comportamenti non definiti in C (stand c99)
- Difficile (ma non impossibile) applicare tecniche formali per la correttezza del software

FP 1

- Modello di computazione: *valutazione di espressioni,* non *esecuzione di comandi*
  - Focus: costruire *funzioni.*
    - Il programmatore dichiara *cosa* (**non come)** fa il programma definendo una funzione che mappa input a output.
  - Si costruiscono funzioni complesse a partire da più semplici *componendole* nel senso matematico del termine

FP 2

- **computazione** è passare argomenti a funzioni
- **Programmare** è dichiarare tipi di dati (la cui rappresentazione è lasciata al compilatore) e delle funzioni che li manipolano
  - Invece di loop, funzioni *ricorsive e higher order* (map)
  - Nozione matematica di variabile *immutabile*

FP: tipicamente tipato

- **Tipo** = insieme di **valori** + **operazioni**
  - int, list<T>, queue<T>
    - queue ops: enqueue, dequeue, is_empty …
  - Tipo è una "*predizione*" del valore che ha un'espressione (se converge)
  - Static vs. dynamic typing
    - *Phase distinction*
      - Errrori trovati a compile time
  - Type checking (tipi espliciti) vs. type inference (tipi impliciti)

FP: vantaggi

- Semantica precisa e semplice
- Ordine flessibile di esecuzione
  - Le funzioni operano isolatamente
    - Parallelismo naturale
  - Ricchi meccanismi di astrazione
    - Valori **non** sono celle in memoria, ma
    - Forme normali come *first class* funzioni, strutture infinite...
  - Garbage collection
  - Sistemi di tipi avanzati

FP: svantaggi

- Minor controllo della CPU e della memoria:
  - Programmi occasionalmente poco efficienti
    - Uso non ottimale della memoria (raro in linguaggi *strict* come F#)
  - Certi programmi sono essenzialmente procedurali
    - Come in Haskell, in F# la programmazione monadica *(workflows)* simula ciò, e comunque posso usare oggetti e parte imperativa di .NET

Perché F#

- Sviluppato da Microsoft Research Cambridge, ma **non** proprietario
  - Grazie a *Mono,* gira dappertutto (anche su Android)
  - Pienamente compatibile con .NET e i suoi linguaggi, in primis C#
  - **Non** è un linguaggio accademico, anzi è il competitor di **Scala** senza tradire FP
  - E' un linguaggio *multi-paradigma*, anche se noi vederemo soprattutto il lato FP

Caratteristiche di F# (ML family)

- *Strongly-typed:*
  - Tipi controllati ed inferiti **staticamente**
- E' *strict* o *call-by-value*
  - Argomenti di una funzione valutati prima del corpo della funzione
  - *Impuro*: funzioni possono avere *side effects*
  - Modificare vars in memoria, I/O, etc.
    - Effects usati *sporadicamente* e solo dove servono

Caratteristiche (cont.)

- *Polimorfismo*: funzioni si applicano a valori di più tipi
- *Higher order*: funzioni sono *first-class* e sono passate come ogni altro dato
- *Gestione automatica della memoria* – no puntatori, GC!
- *ADT, eccezioni*
- Specifico di F#: *workflows/monads* (stile Haskell) e oggetti (.NET)

Qualche esempio
C# vs. F#

- intro.fsx
- Intro.cs

Si, tutto bello, ma che mi serve?

"But let's be honest about this: you'll probably never see an employer advertising for someone with ML skills"
Webber, *Modern programming languages*, 2003

- 13 anni dopo: ICFP
- In particolare: CUFP
- Una lista di applicazioni industriali di Haskell

ICFP sponsors

“Industry day”

- E in Italia ?? … non sono 640 ma in aprile ci saranno presentazioni da aziende italiane che fanno FP (e assumono nel campo)
  - Workinvoice
  - Unicredit
  - Sinapsi
  - Onebip
  - …

Outcome didattico

- Esposizione a un paradigma di programmazione alternativo e in grande espansione
- Approfondimento di argomenti trattati poco nel corso di laurea come ADT, interpreti, compilatori, type-checkers
- Uso pervasivo di *property-based testing*
- Cenni su presupposti teorici di FP, in particolare *type inference/checking,* forse qualcosa su *FP parallelo*

Questo corso non è:

- Esposizione di tutte le caratteristiche di F#
  - Troppa roba, alcune particolarmente esotiche (type providers, query & computation expressions...)
- Carrellata de librerie .NET e della parte OO
  - Siete perfettamente in grado di studiarvele da voi
- Un corso sui fondamenti della FP (lambda-calcolo)
  - Scarso appetito studentesco …

Un po' di storia

Un po' di storia 1

- FP è il paradigma più antico in programmazione; dimenticatevi APL, FORTRAN, etc.
- Lambda-calculus, Church [1932-41]
  - computational notion of a function vs. Turing machines [1937],
  - fun vs. imp, Church's thesis
  - TM = lambda terms

Un po' di storia 2

- **LISP** [62] (McCarthy)
  - recursion and conditionals,
  - lists and ho functions (map)
  - garbage collection
  - S-expressions for both programs and data

- **ISWIM** (Landin [66]):
  - First fp coming explicitly from l-calculus,
  - first abstract machine
  - emphasis on equational reasoning
  - *let* clauses

Un po' di storia 3

- Backus's **FP** [77] (Turing lecture):
  - "Can Programming Be Liberated From the vonNeumann Style? A Functional Style and its Algebra of Programs"
  - **ML** (Gordon, Milner, Wadsworth [79]):
    - first full functional language
    - static types/type inference/ADT/polymorphism
  - **Miranda** (Turner [85]): first *lazy fpl*
    - Guards
    - list comprehension

Un po' di storia 4

- **Erlang** (Ericsson, late 1980s): fault-tolerant systems based on async message passing, no shared memory – "Concurrency Oriented Programming"
  - Remember WhatsApp?
- **Haskell** (Hudak, Wadler etc. 1988): first full *lazy* language & compiler

…

- **F#** (Syme 2005 – …)

Oggi

- FP è in enorme sviluppo
  - **Pratico**
    - Altri linguaggi multi-paradigmi *(Scala)*
    - Aspetti funzionali in linguaggi imperativi
      - Java 8
      - LINQ in C#
      - JavaScript
      - Swift, Go, Rust
  - **Teorico**
    - Generic programming, dependant types (*Agda, Idris, F**), certified programming (*Coq),* reactive FP etc...

Demo