Programmazione Funzionale: Principi

Sommario: 24 Arile, 2019

- Programmazione Funzionale: Principi e Proprietà
  - > Modello di Calcolo
  - > Trasparenza Referenziale
- Strutture Fondamentali:
  - > Espressione, Applicazione, Strategia di Valutazione
  - > Sistema di Tipi e Polimorfismo Generico
  - > API-ADT: Modulo Segnature, Moduli Implementazione
- Higher Order:
  - > Principio di Astrazione e Riuso di Codice
  - > Funzioni Higher Order: List.fold_left
  - > List.fold_right
  - > List.map
  - > List.filter
  - > Definizioni Tail Recursive

Programmazione Funzionale: Principi e Proprietà

- **Modello di Calcolo.** Lambda-Calcolo è il modello ispirante questi linguaggi. Ne consegue:
  - Nessuna Memoria [1]: Valori Non Modificabili
  - Trasparenza Referenziale:[2]

Definition

Sia `P` un programma ed $\texttt{E}_\texttt{i}$ un suo termine (espressione) all'occorrenza `i`. Se la computazione di `P` riduce $\texttt{E}_\texttt{i}$ al valore $\texttt{V}_{\texttt{E}_\texttt{i}}$, allora il programma $\texttt{P}[\texttt{E}_\texttt{i} \leftarrow \texttt{V}_{\texttt{E}_\texttt{i}}]$ è equivalente a `P` e può rimpiazzarlo in ogni computazione.

Ad esempio. `f(x)+f(x)` $\equiv$ 2 * f(x), con + somma, e * prodotto.

  - Correttezza e Verifica: Trasparenza Referenziale permette tecniche di prova piuttosto semplici.

[1] intesa come stato

[2] $\texttt{P}[\texttt{E} \leftarrow \texttt{V}_\texttt{E}]$ indica il rimpiazzamento di ogni occorrenza di `E` in `P` con $\texttt{V}_\texttt{E}$

Programmazione Funzionale: Principi e Proprietà /2

- **Modello di Calcolo.** Lambda-Calcolo è il modello ispirante questi linguaggi. Ne consegue:
  - Nessuna Memoria
  - Trasparenza Referenziale
  - Correttezza e Verifica
  - Funzioni sono valori:
    - calcolabili da un'applicazione di funzione
    - passabili come argomenti a un'applicazione di funzione
  - Programmazione (astratta e) Higher Order
    - Astrazioni sui dati: Modellare valori con funzioni [3]
    - Astrazioni sul controllo: Modellare strutture di controllo con funzioni [4]
  - Computazione per Riduzione: Come per il Lambda-Calcolo (vedi Esercizio su SOS per Lambda-Calcolo)

[3] ad es. interi di Barendregt, env e store in Laboratorio LPL

[4] ad es. iteratori fold_left e fold_right in OCaml

Strutture Fondamentali: Funzioni, Applicazioni, Espressioni

- **Programmi** sono:
  - Una struttura (di Moduli) con definizioni di Tipi, Funzioni e Dati, che forma un ambiente (di bindings)
  - Una espressione da valutare in tale ambiente
- **Funzioni**:
  - Sono definite da espressioni:
    Ad es.: (OCaml) `fun x` $\rightarrow$ `let...in exp`
  - Possono contenere blocchi-espressione, `let_in_`
  - Sono Programming Units: Il corpo ha la stessa struttura del programma:
- **Applicazione**: Il Meccanismo di calcolo fondamentale
- **Trasmissione e Strategia di Valutazione**:
  - OCaml: Trasmissione per valore
  - Haskell: Valutazione esterna e Lazy costruttori

Strutture Fondamentali: Tipi

- Sistema dei Tipi (Pure OCaml).
  - **Tipi Basici Scalari**: int, float, bool, char, e (char)string
  - **T. Polimorfi e Variabili di tipo**. Ad es.: `'a`
  - **Tipi Basici Strutturati**: Tuples, List polimorfe (generiche).
    Ad es.: `('a * int) list`
  - **Tipi funzione**: $\rightarrow$.
    Ad es.: $'a$ `list` $\rightarrow$ ($'a \rightarrow' a \rightarrow$ `bool`) $\rightarrow'$ `a list`
  - **Tipi Concreti**: Record e Variant Types.
    - **Definiti** mediante costrutto:
      type typeName = typeExpression
    - **Record**. Ad es.:
      `type ratio = {num:int; denum:int}`
    - **Variant o Algebrici**. Ad es.:
      `type ('a,'b) myType = Either of 'a | Or of 'b;;`
    - **Variant Ricorsivi**. Ad es.:
      `type 'a myList = Nil | Cons of 'a * 'a myList;;`
  - **Tipi Astratti**