Funzioni e Spezie

Luca Abeni

March 14, 2016

Spezie???

- Qual'è la relazione fra funzioni e spezie???
  - Non esiste...
  - Ma procediamo un po' alla volta
- Di cosa ci occupiamo oggi, allora?
  - Funzioni in più variabili
  - Abbiamo già visto qualcosa (usando `fun ... = ...`)
  - Ma ci sono aspetti che ancora ci sfuggono

```
> fun gcd a b = if b = 0
                then a
                else gcd b (a mod b);
val gcd = fn : int -> int -> int
```

- "`int -> int -> int`"???

Esempio

- Funzioni a più variabili: proviamo a capirle con un esempio
- Funzione `sommaquadrati` che implementa

$$f(x, y) = x^2 + y^2$$

- Matematicamente parlando, $f : \mathcal{N}^2 \to \mathcal{N}$
- Esercizio: implementarla in ML, usando `fun`
  - Sappiamo usare `fun sommaquadrati x y = ...`
  - Ma $\mathcal{N}^2 \to \mathcal{N}$ sembresebbe suggerire qualcosa di diverso:
    `fun sommaquadrati1 (x, y) = ...`

Soluzione

```
fun sommaquadrati x y = x * x + y * y;
```

- Ma abbiamo visto anche un'altra possibile soluzione:

```
fun sommaquadrati1 (x, y) = x * x + y * y;
```

- Qual'è la differenza fra queste due sintassi?
- Entrambe sembrano funzionare correttamente...
- Ma:

```
> sommaquadrati 1 4;
val it = 17 : int
> sommaquadrati1 1 4;
Error-Can't unify Int32.int/int with int * int (Incompatible types) Found near sommaquadrati1
(1)(4)
> sommaquadrati1 (1,4);
val it = 17 : int
```

- Per capire, bisogna guardare i tipi delle due funzioni

Funzioni di Coppie e Funzioni che Ritornano Funzioni

```
> fun sommaquadrati  x y   = x * x + y * y;
val sommaquadrati = fn : int -> int -> int
> fun sommaquadrati1 (x, y) = x * x + y * y;
val sommaquadrati1 = fn : int * int -> int
```

- Tipo di `sommaquadrati1`: `int * int -> int`
  - $\mathcal{N}^2 \rightarrow \mathcal{N}$
  - Un unico argomento, di tipo `int * int`
- Tipo di `sommaquadrati`: `int -> int -> int`
  - ???
  - Hint: per ML, `->` associa a destra...
  - ...Quindi, `int -> (int -> int)`
  - Tutto chiaro, no???

Funzioni che Ritornano Funzioni

- In pratica, `sommaquadrati` prende in ingresso un `int` e ritorna una funzione (da `int` a `int`)!!!
- "Trucco" standard per esprimere funzioni in più variabili
  - $f(x, y) \Rightarrow f'(x)$, dove $f()$ ritorna un numero, mentre $f'()$ ritorna una funzione!
  - Informalmente: è come se $f'()$ "fissasse" il primo parametro di $f()$, generando una funzione del secondo parametro
  - $f'(x) = f_x(y) : f_x(y) = f(x, y)$
- Ah... Ora è tutto più chiaro!
  - Se facciamo "**val** f = sommaquadrati 3;"...
  - ...Otteniamo una funzione $f : \mathcal{N} \to \mathcal{N}$ con $f(y) = 9 + y^2$!

E... Le Spezie?

- Currying: tecnica generica per trasformare una funzione $f()$ a più argomenti in una "catena di funzioni" ad un solo argomento
  - Da Haskell Curry, non da Masala Curry...
- Data $f(x, y) : \mathcal{A} \times \mathcal{B} \to \mathcal{C}$, tramite currying si ottiene $f'(x) = C(f) : \mathcal{A} \to (\mathcal{B} \to \mathcal{C})$ (spesso scritto $\mathcal{A} \to \mathcal{B} \to \mathcal{C}$)
  - $(f'(x))(y) = f(x, y)$
  - Notare che $f'(x)$ è una funzione in y (si può dire $g_x(y) = f'(x)$)
- Non limitato a solo due argomenti...
  - Funzione a n argomenti: gli argomenti sono "eliminabili" uno alla volta
  - Catena di n funzioni ad un argomento

Matematicamente Parlando...

- Haskell Curry era un matematico...
  - Vediamo la tecnica del currying dal punto di vista matematico!
- Insieme delle funzioni $f : \mathcal{D} \to \mathcal{C}$: $\mathcal{C}^{\mathcal{D}}$
- Funzioni in due variabili ($\mathcal{D} = \mathcal{A} \times \mathcal{B}$) $\Rightarrow$ insieme $\mathcal{F}$ delle funzioni $f(x, y) : \mathcal{A} \times \mathcal{B} \to \mathcal{C}$: $\mathcal{F} = \mathcal{C}^{\mathcal{A} \times \mathcal{B}}$
  - Alternativa: invece che $f : \mathcal{A} \times \mathcal{B} \to \mathcal{C}$ usiamo $f : \mathcal{A} \to (\mathcal{B} \to \mathcal{C})$
- Insieme $\mathcal{F}'$ delle funzioni da $\mathcal{A}$ a funzioni da $\mathcal{B}$ a $\mathcal{C}$: $\mathcal{F}' = (\mathcal{C}^{\mathcal{B}})^{\mathcal{A}}$
- Il meccanismo del currying può quindi essere visto come una funzione da $\mathcal{F}$ a $\mathcal{F}'$

Currying Come Funzione...

- Currying come funzione matematica
  - Dall'insieme delle funzioni $f(x, y) : \mathcal{A} \times \mathcal{B} \to \mathcal{C}$
  - All'insieme delle funzioni $f : \mathcal{A} \to (\mathcal{B} \to \mathcal{C})$

$$curry : \mathcal{C}^{\mathcal{A} \times \mathcal{B}} \to (\mathcal{C}^{\mathcal{B}})^{\mathcal{A}}$$

- Importanza matematica: nello studio matematico delle funzioni possiamo restringerci a considerare solo funzioni ad un unico argomento!

In Pratica...

- `fn` permette di definire funzioni ad un solo argomento...
  - ...Ma Haskell Curry ci ha insegnato che questo non è limitante!
  - Tramite currying possiamo codificare funzioni a n argomenti usando `fn`
- Impareremo poi che `fn` == λ
- Notare che in ML una funzione si applica ai suoi argomenti senza usare parentesi e che l'applicazione di funzione associa a sinistra
  - (**fn** x => **fn** y => x * x + y * y) a b = ((**fn** x => **fn** y => x * x + y * y) a) b
  - Prima si applica la funzione ad `a`, poi si applica il risultato (che è una funzione) a `b`

Currying in ML

- A volte, la sintassi di `fn` può risultare poco intuitiva:

```
fn x => fn y => x * x + y * y;
```

- Per questo, ML supporta il meccanismo del currying semplificandone la sintassi con `fun`

```
fun f p1 p2 p3 ... = exp;
```

è equivalente a

```
val rec f = fn p1 => fn p2 => fn p3 ... => exp;
```

Esercizi

- Scrivere (usando val rec ... = fn ...) le seguenti funzioni:
  - Fattoriale, con ricorsione in coda
  - Massimo Comun Divisore
  - Soluzione al problema della Torre di Hanoi

Derivata di una Funzione

- Esempio (forse) utile per capire meglio il meccanismo del currying
  - Calcolo della derivata di una funzione (meglio: rapporto incrementale sinistro)
  - $f'(x) = \lim_{\delta \to 0} \frac{f(x) - f(x-\delta)}{\delta}$
- Scrivere una funzione `calcoladerivata` che riceve come argomenti una funzione `f` ed un valore reale `x` e calcola il valore della derivata di `f` nel punto `x`
  - Dati $f()$ ed x, calcolare $f'(x)$
  - Approssimiamo $f'(x)$ con $\frac{f(x) - f(x-\delta)}{\delta}$ per piccoli valori di δ
  - Funzione $d/dx : \mathcal{R}^{\mathcal{R}} \times \mathcal{R} \to \mathcal{R}$

Soluzione

```
val calcoladerivata
    = fn (f, x) => (f(x) - f(x - 0.001)) / 0.001;
```

- Niente di sorprendente...
- Si poteva fare anche in C

```
double calcoladerivata(double (*f)(double x),
                       double x)
{
  return (f(x) - f(x - 0.001)) / 0.001;
}
```

- Puntatore a funzione invece che valore funzione
- Ma... Come implementare `derivata` che invece di ritornare il valore di $f'()$ nel punto x ritorna direttamente la funzione $f'()$?
- Questo non è proprio facile da fare in C... :)

Soluzione - 2

```
val derivata
    = fn f => fn x => (f(x) - f(x - 0.001)) / 0.001;
```

- Argomenti: $f : \mathcal{R} \to \mathcal{R}$
- Valore di ritorno: $f' : \mathcal{R} \to \mathcal{R}$
  - $(\mathcal{R} \to \mathcal{R}) \to (\mathcal{R} \to \mathcal{R})$
- Mentre `calcoladerivata` aveva argomenti $f : \mathcal{R} \to \mathcal{R}$ e $x \in \mathcal{R}$ e ritornava un valore $f'(x) \in \mathcal{R}$...
- Visto? `derivata` è ottenuta applicando in currying a `calcoladerivata`!