Pattern Matching in Standard ML

Luca Abeni, Csaba Kiraly

April 11, 2016

Introduzione a Pattern Matching

- "Trucco" per funzioni a più argomenti senza usare currying:
  - Uso com unico argomento una coppia (o "tupla")
  - **val** sommaquadrati = **fn** (x, y) => x * x + y * y;
- Ma... Come funziona?
  - Coppia "(x, y)"
  - Invocando la funzione si crea un doppio binding?
  - sommaquadrati (3, 4) crea binding fra `x` e 3 e fra `y` e 4
- Standard ML usa un meccanismo di `pattern matching`
  - Informalmente: `(x, y)` è un pattern che viene matchato con la coppia $(3, 4)$
  - Perché ci sia match bisogna che `x` valga 3 ed `y` valga 4
- Vediamo di capire meglio queste cose...

Pattern

- Vari tipi di pattern:
  - Valore costante
  - Variable pattern: `<var>` o `<var>:<type>`
  - Tupla: `(<pattern1>, <pattern2>, ... <patternn>)`
  - Wildcard pattern: `_`
- Pattern tupla: composti da n pattern più semplici
- Wildcard pattern: matcha qualsiasi cosa
- Esempi:
  - `1`, `false`, `"Ciao"`, ...: pattern costanti
  - `(1, x)`: pattern tupla (di 2 elementi) composto da un pattern costante (`1`) ed un variable pattern (`x`)

Pattern Matching

- Operazione di confronto fra un valore ed un pattern (meglio: fra un pattern costante ed un pattern)
  - Pattern costanti matchano solo col valore del pattern
    - Pattern costante: matcha solo con un pattern costante uguale
  - Pattern variabili matchano se il valore è del tipo giusto; creano legame fra valore e nome
  - Pattern tupla matchano solo se tutte le componenti della tupla matchano
  - `_` matcha con tutto. Si usa spesso per la clausola “default” (o simile)
- Esempi:
  - `(true, true)` matcha con `(true, x)` legando il valore `true` al simbolo `x`
  - `(5, 4)` matcha con `(_, x)` legando il valore `4` al simbolo `x`. Notare che qualsiasi coppia di numeri matcha questo pattern
  - `(5, 4)` non matcha `(_, x:real)`

Pattern ed Invocazione di Funzioni

- Quando una funzione viene invocata, pattern matching fra parametro attuale e parametro formale!
  - **val** succ = **fn** n => n + 1;, invocata come succ 2
    - Pattern matching fra `n` e 2
    - Match: crea legame fra `n` e 2 nel corpo di `succ`
- Se il parametro formale è un pattern tupla, match solo se anche il parametro attuale è un pattern tupla (stesso numero di elementi)
- **val** sommaquadrati = **fn** (x, y) => x * x + y * y;
  - sommaquadrati 3;: pattern matching fra `(x, y)` e 3
    - Fallisce!
  - sommaquadrati (3, 4);: pattern matching fra `(x, y)` e $(3, 4)$
    - Ha successo se `x` matcha con 3 e `y` matcha con 4
    - Crea binding $\texttt{x} \leftarrow 3$ e $\texttt{y} \leftarrow 4$

Pattern in Altre Situazioni

- In quali altri casi Standard ML usa pattern matching?
- Generica creazione di binding (`val`)

```
> val (a,b) = (1, true);
val a = 1 : int
val b = true : bool
```

- Definizione di funzioni “per casi”
  - Utile il costrutto `case`
  - Sorta di “`if` on steroids”

Costrutto case

- "case":

```
case x of
  pattern_1 => exp_1 | ... | pattern_N => exp_N;
```

  - prima valuta l'espressione `x`
  - poi confronta il valore ottenuto con i pattern specificati (pattern_1, pattern_2, ecc.)
  - valuta l'espressione alla *prima* corrispondenza

- una funzione può anche essere definita per casi

```
fn x => case x of
  <pattern_1> => <expression_1>
| <pattern_2> => <expression_2>
...
| <pattern_N> => <expression_N>;
```

Pattern: fn ... case

- esempio:

```
val giorno = fn n => case n of
   1 => "Lunedi"
 | 2 => "Martedi"
 | 3 => "Mercoledi"
 | 4 => "Giovedi"
 | 5 => "Venerdi"
 | 6 => "Sabato"
 | 7 => "Domenica"
 | _ => "Giorno non valido";
```

- il pattern _ cattura tutti i casi non precedentemente enumerati (numeri interi minori di 1 o maggiori di 7)
- come nel caso di if .. then .. else .. :
  - tutti gli `exp` devono avere lo stesso tipo
  - i `pattern` devono coprire tutti i possibili valori

Pattern: fn, Sintassi Semplificata

- sintassi semplificata

```
val giorno = fn n => case n of
  1 => "Lunedi"
| 2 => "Martedi"
...
| _ => "Giorno non valido";
```

```
val giorno = fn
  1 => "Lunedi"
| 2 => "Martedi"
...
| _ => "Giorno non valido";
```

Pattern Complessi

- Pattern: non solo costanti e "`default`" come in C, ma essere anche:
  - nomi: crea binding con scope locale fra valore della espressione e nome nel pattern

```
val f = fn a => case a of
    0 => 1000.0
  | x => 1.0 / (real x);
```

  - tuple: associa nomi ad elementi di un tuple

```
val somma = fn (a, b) => a + b;
```

```
val (x, y) = (4, 5);
```