Liste in SML

Luca Abeni

May 5, 2017

Liste Standard ML

- α list: liste costituite da una sequenza finita di elementi di tipo α
  - Nota: valutazione per valore (non per nome o lazy) $\rightarrow$ no liste infinite
- Tutto come già visto usando il tipo ricorsivo
  datatype 'a lista = vuota | cons **of** ('a $*$ 'a lista)
- Parte del linguaggio $\Rightarrow$ sintassi semplificata
  - `vuota` $\equiv$ `[]`
  - `cons` $\equiv$ operatore *infisso* `::`
  - Rappresentazione semplificata: `1::2::3::[]` $\equiv$ `[1,2,3]`
- Il nome "ufficiale" del tipo è `'a list`

Liste Standard ML: Esempi

- cons(1, cons(2, cons(3, vuota)))
  - tipo `int lista`
  - Equivalente a 1::2::3::[] (tipo `int list`)...
  - ...Che si scrive anche [1,2,3]
- cons(”Ciao,␣” , cons(”come␣”, cons(”stai?”, vuota)))
  - tipo `string lista`
  - Equivalente a ”Ciao,␣” :: ”come␣”::”stai?” ::[] (tipo `string list`)...
  - ...Che si scrive anche [”Ciao,␣”, ”come␣”, ”stai?”]
- Possibili anche cose più complesse:
  - [(1, ”Ciao”), (2, ”come”)]
  - [[1, 2], [3, 4, 5]]

Operatori su Liste

- Ricordate? 2 costruttori 2 funzioni di accesso
  - Costruttori: `cons` e `vuota`
  - Accesso: `car` e `cdr`
- Tutto uguale, ma cambiano nomi e sintassi!
  - Costruttori: `::` (infisso) e `[]` (anche chiamato `nil`)
  - Accesso: `hd` e `tl`
    - Funzioni non pure: eccezioni!
- `l` era ottenibile come `cons((car l), (cdr l))`
  - Diventa hd l :: tl l

Funzioni sulle Liste

- In teoria basterebbero i due costruttori, `hd` e `tl`...
  - ...Ma Standard ML fornisce anche altre funzioni di utilità
  - Tutte implementabili con costruttori ed operatori di base
- Concatenazione: `@`. `[1, 2] @ [3, 4, 5] = [1, 2, 3, 4, 5]`
  - Ricordare funzione `appendi`

```
val rec appendi = fn vuota => (fn b => b)
                  |cons(e,l) => (fn b => cons (e, appendi l b));
```

- Lunghezza: `length`. length [1,2,3] = 3
- Inverti lista: `rev`
- Conversione fra stringhe e liste di caratteri: `explode` e `implode`

Definizione di Funzioni su Liste

- Tipo di dato ricorsivo $\Rightarrow$ funzioni spesso ricorsive
  - Base induttiva (fine della ricorsione): lista vuota `[]` (o `nil`)
  - Passo induttivo: data lista non vuota `e::l1`, richiama la funzione su `l1`
- Spesso definite con *pattern matching*: pattern `[]` e `e::l1`

```
val rec f = fn []     => ...
            | e::l1 => ...
```

  - La prima clausola non invoca ricorsivamente `f`
  - La seconda clausola può invocare ricorsivamente `f`
- In caso di più parametri, generalmente ricorsione sul primo!
  - Vedi funzione `inserisci`

Esempio: Appartenenza ad una Lista

- Funzione che controlla se un elemento `e` appartiene ad una lista `l`
  - `e` non appartiene alla lista vuota
  - Se `l` non è vuota ed `e` = `hd l`, allora `e` appartiene ad `l`
  - Altrimenti, controlla se `e` appartiene alla coda di `l`

```
val rec appartiene =
      fn e =>
            (fn [] => false
              | (h::l) => if e = h then
                            true
            else
                            appartiene e l);
```

- Notare il tipo: `''a -> ''a list -> bool`
  - `''a`??? Type variable che accetta solo tipi “speciali”: tipi su cui è definita uguaglianza!

Esempio: Lista di Numeri fra n e m

- Ricorsione non solo sulla dimensione della lista...
- Esempio: generare una lista contenente i numeri fra n e m

```
val rec numeri =
      fn n =>
            fn m =>
                  if n > m then
                        []
                  else
                        n::(numeri (n + 1) m);
```

Esempio più Complesso: Massimo di una Lista

- Lista vuota: massimo 0
- Lista contenente solo n: massimo n
- Lista contenente due elementi n, m seguiti da un'altra lista
  - Se $n > m$, massimo fra n e la lista rimanente
  - Se $n \leq m$, massimo fra m e la lista rimanente

```
val rec massimo =
      fn []     => 0
    | [n]       => n
    | n::m::l => if n > m then
                   massimo (n::l)
                 else
                   massimo (m::l);
```