Progetto per il Laboratorio di Programmazione
Un interprete per il linguaggio PINO

Descrizione del progetto

Stefano Guerrini

A.A. 2002/03 – Canale P-Z

Versione del 22 luglio 2003

1 Esempi di programmi PINO

In questa sezione diamo alcuni esempi di funzioni ricorsive definibili in PINO. Questi esempi saranno utilizzati per verificare i moduli.

La prima funzione che prendiamo in considerazione è quella che verifica se due liste sono uguali. La funzione ritorna $[*]$ se le due liste di input sono uguali e $*$ se non lo sono.

$$\begin{array}{l}\textbf{deffun } \mathit{eq}(l_1, l_2) = \\ \quad \textbf{ite}(l_1, \\ \quad\quad \textbf{ite}(l_2, \\ \quad\quad\quad \textbf{ite}(\mathit{eq}(\textbf{hd}(l_1), \textbf{hd}(l_2)), \\ \quad\quad\quad\quad \textbf{ite}(\mathit{eq}(\textbf{tl}(l_1), \textbf{tl}(l_2)), [*], *), \\ \quad\quad\quad\quad *), \\ \quad\quad\quad *), \\ \quad\quad \textbf{ite}(l_2, *, [*]));\end{array}$$

La funzione *append* concatena due liste, appendendo la seconda in fondo alla prima.

$$\textbf{deffun } \mathit{append}(l_1, l_2) = \textbf{ite}(l_1, \langle \textbf{hd}(l_1) : \mathit{append}(\textbf{tl}(l_1), l_2)\rangle, l_2);$$

La prossima funzione esegue un appiattimento della lista di input. Ad esempio, data una lista in cui il simbolo $*$ appare cinque volte (a qualsiasi livello di profondità), il risultato della funzione *flat* su tale lista è la lista che si ottiene concatenando cinque volte $*$, quindi, $\mathit{flat}([*, [*, [*, *], *]]) = [*, *, *, *, *]$. In modo equivalente, possiamo dire che la funzione *flat* cancella tutte le parentesi quadre all'interno di una lista (ovviamente, le parentesi più esterne rimangono).

$$\textbf{deffun } \mathit{flat}(l) = \textbf{ite}(l, \mathit{append}(\textbf{ite}(\textbf{hd}(l), \mathit{flat}(\textbf{hd}(l)), [*]), \mathit{flat}(\textbf{tl}(l))), *);$$

Definiamo infine una funzione utile per creare liste di prova.

$$\textbf{deffun } \mathit{dupnil}(l) = \textbf{ite}(l, \langle \textbf{ite}(\textbf{hd}(l), \mathit{dupnil}(\textbf{hd}(l)), [*, *]) : \mathit{dupnil}(\textbf{tl}(l))\rangle, *);$$

La funzione *dupnil* sostituisce tutti i simboli $*$contenuti all'interno di una lista (a qualsiasi livello di profondità) sono rimpiazzati dalla lista $[*, *]$. Per mezzo della funzione *dupnil* si possono facilmente costruire liste molto lunghe, basta tener conto che ad ogni applicazione della *dupnil*, la dimensione del risultato è circa il doppio.

1.1 Un esempio di sessione

Riportiamo qui di seguito un esempio di sessione con l'interprete PINO senza garbage collection (modulo VI).

Dopo aver definito le funzioni descritte precedentemente

```
# deffun append(l1, l2) = ite(l1, <hd(l1) : append(tl(l1), l2)>, l2);
deffun append(l1, l2) =
  ite(l1, <hd(l1) : append(tl(l1), l2)>, l2)
# deffun eq(l1, l2) =
      ite(l1,
          ite(l2,
              ite(eq(hd(l1), hd(l2)),
                  ite(eq(tl(l1), tl(l2)), [*], *),
                  *),
              *),
          ite(l2, *, [*]));
deffun eq(l1, l2) =
  ite(l1, ite(l2, ite(eq(hd(l1), hd(l2)), ite(eq(tl(l1), tl(l2)), [*], *), *), *),
ite(l2, *, [*]))
# deffun dupnil(l) =
       ite(l, < ite(hd(l), dupnil(hd(l)), [*, *]) : dupnil(tl(l))>, *);
deffun dupnil(l) =
  ite(l, <ite(hd(l), dupnil(hd(l)), [*, *]) : dupnil(tl(l))>, *)
# deffun flat(l) =
  ite(l, append(ite(hd(l), flat(hd(l)), [*]), flat(tl(l))), *);
deffun flat(l) =
  ite(l, append(ite(hd(l), flat(hd(l)), [*]), flat(tl(l))), *)
```

engono definite alcune variabili usando la funzione *append*.

```
# defvar a = [[*], *];
defvar a =
  [[*], *]
-> [[*], *]
# defvar b = [*, [*]];
defvar b =
  [*, [*]]
-> [*, [*]]
# defvar c = append(a, b);
defvar c =
  append(a, b)
-> [[*], *, *, [*]]
# defvar d = append(b, a);
defvar d =
  append(b, a)
-> [*, [*], [*], *]
```

Si osservi che le variabili c e d contengono valori diversi, dato che sono ottenute concatenando le liste a a b con un diverso ordine. Quindi $eq(c, d)$ deve dare $*$.

```
# eval eq(c, d);
eq(c, d)
-> *
# eval eq(flat(dupnil(c)), flat(dupnil(d)));
eq(flat(dupnil(c)), flat(dupnil(d)))
-> [*]
```

In ogni caso però che c e d contengono lo stesso numero di simboli $*$ e che tale proprietà rimane vera se applichiamo *dupnil* ad entrambe le liste. Quindi, $eq(flat(dupnil(c)), flat(dupnil(d)))$ da correttamente $[*]$.

I successivi esempi sono simili a quelli precedentemente visti, ma eseguiti su liste più lunghe.

```
# defvar e = <dupnil(c), dupnil(d), dupnil(c) : dupnil(d)>;
defvar e =
  <dupnil(c), dupnil(d), dupnil(c) : dupnil(d)>
-> [[[[*, *]], [*, *], [*, *], [[*, *]]], [[*, *], [[*, *]], [[*, *]], [*, *]],
[[[*, *]], [*, *], [*, *], [[*, *]]], [*, *], [[*, *]], [[*, *]], [*, *]]
# defvar f = <dupnil(d), dupnil(c), dupnil(c) : dupnil(c)>;
defvar f =
  <dupnil(d), dupnil(c), dupnil(c) : dupnil(c)>
-> [[[*, *], [[*, *]], [[*, *]], [*, *]], [[[*, *]], [*, *], [*, *], [[*, *]]],
[[[*, *]], [*, *], [*, *], [[*, *]]], [[*, *]], [*, *], [*, *], [[*, *]]]
# eval eq(flat(e), flat(f));
eq(flat(e), flat(f))
-> [*]
```