Lezione 15-16

March 19, 2012

Astrazioni di controllo: Costrutti Particolari

- Procedure e Funzioni come parametri: Dove e Perchè
- Decomposition based Programming Methodology
- Problemi di Binding e di Scope: Un esempio
- Trasmissione deep binding: FUI
- Lambda Astrazioni
- Trasmissione shallow binding: FUI
- Funzioni come valori: Higher Order Programming Methodology
- Astrazioni per supportare Proc. e Funz. Ricorsive: FUI
- Divide and Conquer Programming Methodology
- Iterazione e Ricorsione: Modi diversi di risolvere problemi
- Tail Recursion: Trasformazioni Interessanti

Procedure e Funzioni come parametri: Dove e Perchè/1

Decomposition Based Programming Methodology

Dove

- Procedure possono avere come parametri altre procedure o delle funzioni. Lo stesso può essere previsto per le funzioni
- Possiamo astrarre, generalizzare, rispetto ad un'(altra) astrazione (*Vertical Dec.* vs. *Horizontal Dec.*). Ad esempio, in un programma decidiamo:
  - (a) prima di astrarre, in una procedura, P, un'algoritmo di ordinamento su collezioni di valori di un dominio D e
  - (b) dopo, di astrarre, in una funzione, F, l'algoritmo per la relazione di ordine di due arbitrari valori di D.
  - (a+b) è realizzato:

Procedure e Funzioni come parametri: Dove e Perchè/2

Decomposition Based Programming Methodology

Dove:

- (a+b) è realizzato:
  - Usando nel codice C_P della procedura, invocazioni $g(e_1, e_2)$ ogni volta che si abbia necessità di sapere in quale ordine siano i valori di D calcolati da e_1 ed e_2;
  - Estendendo l'intestazione di P in modo tale da includere un parametro funzione g dello stesso tipo T della funzione F;
  - Usando nel programma una definizione per la funzione F accessibile (che abbia nel proprio scope la procedura? -no) alla definizione di P
  - trasmettendo F come parametro funzione ad ogni invocazione di $P(..., T\ F, ...)$

Procedure e Funzioni come parametri: Dove e Perchè/3

Decomposition Based Programming Methodology

- (a+b) è realizzato:
  - Usando nel programma una definizione per la funzione F che sia accessibile all'invocazione della procedura P

Perchè

- F potrebbe essere già stata scritta quando definiamo il programma
- F può essere modificata senza necessità di guardare come sia definita P
- F può essere sostituita in alcune invocazione di P da un'altra funzione che realizza una differente relazione di ordine

Problemi di Binding e di Scope: Un esempio

$\{...$
$\quad \texttt{type}\ T = \texttt{int} \times \texttt{int} \rightarrow \texttt{bool};$
$\quad \texttt{int}\ u...$
$\quad \texttt{function bool}\ F_1(\texttt{int}\ x, \texttt{int}\ y)\{...u...\};$
$\quad \texttt{procedure}\ P(T\ g)\{....\}$
$\quad \texttt{procedure}\ Q()\{$
$\quad\quad \texttt{int}\ u...$
$\quad\quad \texttt{function bool}\ F_2(\texttt{int}\ x, \texttt{int}\ y)\{...u...\};$
$\quad\quad ...P(F_1); ...P(F_2); ....\}$
$\quad ...$
$\quad P(F_1); ...\}$

- Il valore denotabile di F_2 sopravvive al suo binding (se abbiamo scope statico)
- Chi è il binding di u? Nelle invocazioni dentro Q, potrebbe essere sempre quello di Q (shallow binding)

Trasmissione Deep Binding, DB (in scoping statico)/1

- DB estende in modo naturale lo scope della dichiarazioni alla trasmissione
- Vale anche per procedure ma più interessante su funzioni
- Prima però, vediamo una nuova classe di funzioni: Lambda Astrazioni. Ad esempio $\sharp(x)\ x<=5$

Table12bis – Trasmissione di Valori Funzione
Domini Sintattici
$\texttt{D} ::= \ldots \mid \texttt{Function I(P}_1\ \texttt{I}_1\ \ldots\ \texttt{P}_n\ \texttt{I}_n\texttt{) E} \mid \ldots$
$\texttt{E} ::= \ldots \mid \texttt{A} \mid \texttt{Lambda(P}_1\ \texttt{I}_1\ \ldots\ \texttt{P}_n\ \texttt{I}_n\texttt{) E} \mid \ldots$

- Lambda Astrazioni non hanno naming: Valori esprimibili, denotabili solo con parametri (e raramente memorizzabili)
- raramente hanno meccanismi per esprimere definizioni ricorsive

Lambda Astrazioni

Table12bis – Trasmissione di Valori Funzione
Domini Sintattici
$\texttt{D} ::= \ldots \mid \texttt{Function I}(\texttt{P}_1\ \texttt{I}_1 \ldots \texttt{P}_n\ \texttt{I}_n)\ \texttt{E} \mid \ldots$
$\texttt{E} ::= \ldots \mid \texttt{A} \mid \texttt{Lambda}(\texttt{P}_1\ \texttt{I}_1 \ldots \texttt{P}_n\ \texttt{I}_n)\ \texttt{E} \mid \ldots$

$\{\ldots \texttt{type}\ C(t) = \ldots;\ T_f(t) = t \rightarrow \texttt{bool};\ T_o(t) = t \times t \rightarrow t; \ldots$
$\texttt{function}\ C(t)\ Filter(C(t)\ c, T_f(t)\ r)\{\ldots.\};$
$\ldots\{\ldots C(int) v = \ldots;$
$\ldots.Filter(v, \sharp(\mathbf{x})\ \mathbf{x} > \mathbf{5}) \ldots // maggiori\ di\ 5$
$\ldots.Filter(v, \sharp(\mathbf{x})\ \mathbf{x} <= \mathbf{5}) \ldots // minori\ o\ uguali\ a\ 5$
$\ldots\{\ldots \texttt{type}\ T(t) = C(t) \times T_f(t) \rightarrow C(t);$
$\texttt{function}\ C(t)\ QuickSort(C(t)\ c, T(t)\ f, T_o(t) o)\{\ldots.\};$
$\ldots.QuickSort(v, Filter, \sharp(\mathbf{x}, \mathbf{y})\ (\mathbf{x} > \mathbf{y})?\mathbf{y} : \mathbf{x})$

Trasmissione Deep Binding, DB (in scope statico)/2

Table12.1 – Deep Binding con Scope Statico

Funzioni Semantiche

$$\mathcal{D}_E[\![\mathtt{D}]\!]_\rho : (\mathtt{Env} \times \mathtt{Store}) \to \mathtt{Env}_\perp$$

$$\mathcal{D}_E[\![\mathtt{Function\ I(Fun\ I_1)\ E}]\!]_\rho =$$
$$\mathtt{Let}\{f = \lambda v\,.\mathtt{Let}\{\rho_1 = \mathtt{bind}(\mathtt{I}_1, \mathtt{F}(v), \rho)\}$$
$$\mathcal{E}[\![\mathtt{E}]\!]_{\rho_1}\}$$
$$\mathtt{bind}(\mathtt{I}, \mathtt{F}(f), \rho)$$

Funzioni Ausiliarie

$$\mathtt{F} : \mathtt{Fun} \to \mathtt{Den}$$

$$\in \mathtt{Fun} : \mathtt{Val} \to \mathtt{TruthV}$$

Trasmissione Deep Binding, DB (in scope statico)/3

Table12.1 – Deep Binding con Scope Statico

Funzioni Semantiche

$\mathcal{E}[\![\mathtt{E}]\!]_\rho : \mathtt{Env} \to \mathtt{State} \to \mathtt{State}_\perp$

$$\mathcal{E}[\![\mathtt{Call\ I(Fun(E))}]\!]_\rho(\mathtt{s}) = \mathtt{Let}\{g = \mathcal{E}[\![\mathtt{E}]\!]_\rho, \mathtt{F}(f) = \rho(\mathtt{I})\} f(g)(\mathtt{s})$$

$$\mathcal{E}[\![\mathtt{Lambda(byValue\ I_1)E}]\!]_\rho(\mathtt{s}) = \lambda\, v\,.\lambda\, s\,.\mathtt{Let}\{(l_1, s_1) = \mathit{allocate}(s)\} \{s_2 = \mathtt{upd}(l_1, v, s_1),\ \rho_1 = \mathtt{bind}(\mathtt{I}_1, l_1, \rho)\} \mathcal{E}[\![\mathtt{E}]\!]_{\rho_1}(s_2)\}$$

Domini Ausiliari

$\mathtt{Den} ::= \mathtt{Loc} + \mathtt{ProcFun} + \mathtt{VL} + \mathtt{Code} + \mathtt{Fun}$

$\mathtt{Fun} ::= \mathtt{State} \to \mathtt{State}$

Trasmissione Shallow Binding, SB (in scope statico)/1

- SB non estende in modo naturale lo scope statico della dichiarazioni alla trasmissione
- Occorre ri-definire le dichiarazioni di funzione (e procedura)
- Quando invocate hanno scope statico, quando trasmesse dinamico
- Il valore denotabile di una funzione è una chiusura

Table12.1 – Shallow Binding con Scope Statico

$$\mathcal{D}_E[\![\texttt{Function I(Fun I}_1\texttt{) E}]\!]_\rho =$$
$$\texttt{Let}\{f = \lambda\delta.\lambda v.\texttt{Let}\{\rho_1 = \texttt{bind}(\texttt{I}_1, v, \delta)\}$$
$$\mathcal{E}[\![\texttt{E}]\!]_{\rho_1}\}$$
$$\texttt{bind}(\texttt{I}, [\texttt{F}(f), \rho], \rho)$$

Funzioni Ausiliarie

$$\texttt{Den} ::= \texttt{Loc} + \texttt{ProcFun} + \texttt{VL} + \texttt{Code} + \texttt{Fun} + [\_, \_]$$
$$\texttt{Fun} ::= \texttt{State} \rightarrow \texttt{State}$$

Trasmissione Shallow Binding, SB (in scope statico)/4 – wrong

Table12.1 – ShallowBinding con Scope Statico

Funzioni Semantiche

$\mathcal{E}[\![\mathtt{E}]\!]_\rho : \mathtt{Env} \to \mathtt{State} \to \mathtt{State}_\perp$

$\mathcal{E}[\![\mathtt{Call\ I(Fun(E))}]\!]_\rho(\mathtt{s}) =$
$\quad \mathtt{Let}\{[\mathtt{F}(g), \delta_g] = \mathcal{E}[\![\mathtt{E}]\!]_\rho, [\mathtt{F}(f), \delta_f] = \rho(\mathtt{I})\}\ f(\delta_f)(g(\rho))(\mathtt{s})$

$\mathcal{E}[\![\mathtt{Lambda(byValue\ I_1)E}]\!]_\rho(\mathtt{s}) =$
$\mathtt{Let}\{h = \lambda\delta.\lambda v.\lambda s.\mathtt{Let}\{(l_1, s_1) = \mathit{allocate}(s)\}$
$\{s_2 = \mathtt{upd}(l_1, v, s_1),\ \delta_1 = \mathtt{bind}(\mathtt{I}_1, l_1, \delta)\}$
$\mathcal{E}[\![\mathtt{E}]\!]_{\delta_1}(s_2)\}\}$
$[\mathtt{F}(h), \rho]$

manca un "pezzetto" di espressione: dove, come, perchè - correggere