PLP - Modulo II
Semantica Denotazionale

prof. Marco Bellia

march 30, 2011

1 Le Strutture di Base

1.1 Domini Sintattici

Introducono i termini (strutture) del linguaggio e possiamo considerarli alberi di sintassi astratta con i loro bravi costruttori che, applicati ad alberi argomento, generano un nuovo albero avente tali alberi come figli.

Table 1

Domini Sintattici

`D`	::=	`Proc I() C;` | `D D` | ...	(*Dichiarazioni*)
`C`	::=	`{D C}` | `I := E` | `Call I ()` | ...	(*Comandi*)
`E`	::=	`I` | `LV` | ...	(*Espressioni*)
`LV`	::=	...	(*Literals*)

Esercizio 1 *(a) Si elenchino i costruttori, indicandone la segnatura, utilizzati nel dominio* `C` *in aggiunta al costruttore binario* $\{\_\ \_\}$ *con segnatura* $\mathtt{D} \times \mathtt{C} \rightarrow \mathtt{C}$ *(ovvero, indicheremo il tutto con la scrittura:* $\{\_\ \_\} : \mathtt{D} \times \mathtt{C} \rightarrow \mathtt{C}$*).*
(b) Qual'e il costruttore della sequenza di dichiarazione.

Esercizio 2 *Sotto quali condizioni il linguaggio in Table 1 è un linguaggio Turing Completo (abbr. TC)?: Si dica quali caratteristiche devono o non devono avere i costrutti in tabella affichè il linguaggio sia o non sia TC.*

1.2 Domini Semantici

Introducono i valori che utilizzeremo per definire le funzioni semantiche.

Notazione 1 *Useremo i nomi dei domini (e i nomi posti a sinistra di '::='), eventualmente indiciati, come nomi di metavariabili che variano sul dominio. In tal modo, ad esempio, quando incontreremo il simbolo* `D` *sappiamo che esso indica, a seconda del contesto, l'intero dominio delle dichiarazioni oppure una dichiarazione tra quelle possibili. In tabella 2, nella'anticipare le funzioni* `bind`, `find`, `empty` *ricorriamo all'uso di meta termini la cui notazione e significato associato è presentata in Notazione 2*

Table 2

Domini Semantici

$\texttt{Env}, \rho, \delta$	$\equiv$	$\texttt{I} \to \texttt{Den}$	(*Ambienti*)
		operazioni di Env :	
		$\texttt{bind}: \ \texttt{I} \times \texttt{Den} \times \texttt{Env} \to \texttt{Env}$	
		$\texttt{bind}(\texttt{I}, \texttt{d}, \rho) \equiv \lambda \texttt{x}.\ \texttt{if}((\texttt{x}\ eq\ \texttt{I}), \texttt{d}, \rho(\texttt{I}))$	
		$\texttt{find}: \ \texttt{I} \times \texttt{Env} \to \texttt{Den}$	
		$\texttt{find}(\texttt{I}, \rho) \equiv \rho(\texttt{I})$	
		$\texttt{empty} : \texttt{Env}$	
		$\texttt{empty} \equiv \lambda \texttt{x}.\ \texttt{x}$	
$\texttt{Store}, \texttt{s}$	$\equiv$	...	(*Memoria*)
		operazioni di Store :	
		$\texttt{upd}: \ \texttt{Loc} \times \texttt{Mem} \times \texttt{Store} \to \texttt{Store}$	
		$\texttt{look}: \ \texttt{Loc} \times \texttt{Store} \to \texttt{Mem}$	
State	::=	...	(*Stato*)

Domini Ausiliari

$\texttt{Val}, \texttt{v}$	::=	...	(*Valori* :	*Esprimibili*)
$\texttt{Den}, \texttt{d}$	::=	...	(*Valori* :	*Denotabili*)
$\texttt{Mem}, \texttt{d}$	::=	...	(*Valori* :	*Memorizzabili*)
$\texttt{Loc}, \texttt{l}$	::=	...		(*Locazioni*)
Input	::=	...	(*I/O* :	*Input*)
Output	::=	...	(*I/O* :	*Output*)

Esercizio 3 *Per alcuni linguaggi il dominio* State *coincide con il dominio* Store*. Per tali linguaggi si dice che le operazioni di I/O non fanno parte del linguaggio ma il linguaggio ne permette l'uso attraverso la propria interfaccia con codice nativo di sistema (*language native interface*). Quando le operazioni di I/O fanno invece, parte del linguaggio quale potrebbe essere una definizione per* State*.*

Esercizio 4 *Osservando le definizioni in Table 2 e in Table 3 si dica cosa sono i valori* denotabili*, i valori* memorizzabili*, i valori* esprimibili*. Si giustifichi la risposta richiamando le definizioni, nelle due tabelle, a conferma di quanto asserito.*

Esercizio 5 *Si consideri il seguente frammento di programma C*

```
#define max 100
typedef enum colore {bianco, nero, giallo, rosso} colore;
typedef struct list{colore *head; struct list *next;}list;
void printColore(colore *j){...}
{
list *quad, *temp; colore x;
goto addr;
addr:   x = bianco;
quad = NULL;
temp = (struct list *)malloc(sizeof(struct list));
```

```
temp->head = &x;  temp->next = quad; quad = temp;  colore p = *quad->head;
if(quad == NULL) printf("puntatore nullo");
else printColore(&(*quad->head));
}}
```

Per ciascuno dei domini di valori (denotabili, esprimibili, memorizzabili) si indichino almeno 2 strutture, se esistono, che coinvolgono anche separatamente, valori del dominio: specificando quali valori sono coinvolti e perch.

Esercizio 6 *Si consideri il seguente frammento di programma Pascal*

```
label addr;
const max = 100;
type colore = (bianco, nero, verde, giallo, rosso); { nessun commento }
        colorePtr = ^colore;
        listPtr = ^list;
       list = record head : colorePtr; next : listPtr;  end;
var  quad, temp : listPtr;   x : colorePtr;
procedure printColore(var j: colorePtr); begin ... end
begin
       goto addr;
       addr : new(x);
       x^ := bianco; quad := nil;
       new(temp); temp^.head := x; temp^.next := quad; quad := temp;
       if (quad = nil) then Writeln("puntatore nullo") else printColore(quad^.head);
     end.
```

Per ciascuno dei domini di valori (denotabili, esprimibili, memorizzabili) si indichino almeno 2 strutture, se esistono, che coinvolgono anche separatamente, valori del dominio: specificando quali valori sono coinvolti e perch.

Esercizio 7 *Si elenchino quali sono i valori denotabili di Java fornendo i costruttori (nomi e segnatura) della loro sintassi astratta (si ricordi che la scelta dei nomi inessenziale per le caratteristiche della sintassi astratta risultante).*

Esercizio 8 *(a) Confrontando il programma C e il programma Pascal degli esercizi sopra, si dica perch possiamo asserire che i puntatori sono valori denotabili in C mentre lo stesso non si pu affermare per il Pascal: In particolare si indichi il costrutto del programma C che contiene tale valore e si discuta sull'impossibilit in Pascal di esprimere la stessa struttura. (b) Si dica poi perch, contrariamente a quanto avviene in generale, in C le locazioni di variabile sono puntatori.*

1.3 Funzioni Semantiche

Introducono le strutture con cui daremo significato alle strutture del linguaggio. In questo caso useremo tre funzioni con arità e segnatura come indicato in Table 3.

Table 3

Funzioni Semantiche

$\mathcal{D}$:	$\texttt{D} \rightarrow \texttt{Env} \rightarrow \texttt{Env}$	(*Significato delle Dichiarazioni*)
$\mathcal{M}$:	$\texttt{C} \rightarrow \texttt{Env} \rightarrow \texttt{State} \rightarrow \texttt{State}$	(*Significato dei Comandi*)
$\mathcal{E}$:	$\texttt{E} \rightarrow \texttt{Env} \rightarrow \texttt{State} \rightarrow \texttt{Val}$	(*Significato delle Espressioni*)

Funzioni Ausiliarie

IntoVal :	$\texttt{LV} \rightarrow \texttt{Val}$
MemToVal :	$\texttt{Mem} \rightarrow \texttt{Val}$

Esercizio 9 *(a) Si elenchino le classi di literals del linguaggio C mostrato una definizione del dominio* LV*per il linguaggio C. (b) Si faccia la stessa cosa per il linguaggio Java; (c) Si faccia la stessa cosa per ciascun ulteriore linguaggio studiato.*

Esercizio 10 *(a) Si mostri la definizione del dominio* Val *del linguaggio C limitatamente ai valori coinvolti nell'esercizio 5; (b) Si mostri una definizione per* IntoVal *del linguaggio C.*

Esercizio 11 *(a) Si faccia la stessa cosa dell'esercizio precedente per* Val *di Pascal facendo riferimento al programma dell'esercizio 6; (b) Si mostri una definizione per* IntoVal *del linguaggio Pascal.*

2 Semantica: Linguaggi con Naming ma Senza Blocchi o con solo Blocchi In-Line

Definiamo in Table 4 le funzioni semantiche nel caso di Linguaggi Imperativi con strutture estremamente semplici.

Notazione 2 *Per esprimere le funzioni semantiche useremo la λnotazione, pertanto* $\lambda\ \texttt{x}_1...\texttt{x}_n\texttt{.}\ \texttt{F}$ *esprime la funzione nelle variabili* $\texttt{x}_1...\texttt{x}_n$ *che, applicata a* $\texttt{F}_1...\texttt{F}_n$*, calcola il valore espresso da* $[\texttt{F}_1...\texttt{F}_n/\texttt{x}_1...\texttt{x}_n]\texttt{F}$*, ovvero il valore espresso da* F *dove le occorrenze delle variabili* $\texttt{x}_1...\texttt{x}_n$ *sono simultaneamente sostituite dai corrispondenti argomenti* $\texttt{F}_1...\texttt{F}_n$*. Ovviamente,* $\texttt{F}, \texttt{F}_1, ..., \texttt{F}_n$ *sono tutti termini con cui esprimiamo i significati definiti dalla semantica.*
Usiamo la composizione di funzione $\circ$ nella seguente forma: $\texttt{f} \circ \texttt{g}\ (\texttt{F}) \equiv \texttt{g}(\texttt{f}(\texttt{F}))$

Table4

Linguaggi Imperativi : solo Naming e Blocchi In – Line

$\mathcal{M}[\![\mathtt{C}]\!]_\rho : \mathtt{State} \to \mathtt{State}$ (*Significato dei Dichiarazioni*)

$\mathcal{M}[\![\mathtt{I} := \mathtt{E}]\!]_\rho = \lambda\ \mathtt{s.}\ \mathtt{upd}(\mathtt{find}(\mathtt{I}, \rho), \mathcal{E}[\![\mathtt{E}]\!]_\rho(\mathtt{s}), \mathtt{s})$
$\mathcal{M}[\![\mathtt{C}_1; \mathtt{C}_2]\!]_\rho = \mathcal{M}[\![\mathtt{C}_1]\!]_\rho \circ \mathcal{M}[\![\mathtt{C}_2]\!]_\rho$

$\mathcal{E}[\![\mathtt{E}]\!]_\rho : \mathtt{State} \to \mathtt{Val}$ (*Significato delle Espressioni*)

$\mathcal{E}[\![\mathtt{I}]\!]_\rho = \lambda \mathtt{s.}\ \mathtt{MemToVal}(\mathtt{look}(\mathtt{find}(\mathtt{I}, \rho), \mathtt{s}))$
$\mathcal{E}[\![\mathtt{LV}]\!]_\rho = \lambda \mathtt{s.}\ \mathtt{IntoVal}(\mathtt{LV})$

$\mathtt{State} ::= \mathtt{Store}$ (*Stato*)

Esercizio 12

3 Semantica: Linguaggi a Blocchi e/o Blochi Procedura

Definiamo le funzioni semantiche nel caso di Linguaggi Imperativi con strutture a blocchi includenti blocchi procedura/funzioni, oltre a blocchi in-line. Queste estendono le definizioni di Table 4, mantenendole integralmente. Distinguiamo i due tipi di Scoping.

3.1 Semantica: Linguaggi con Scoping Statico

Definiamo il comportamento della dichiarazione di procedura (e funzione) e dell'invocazione rispetto all'ambiente quando la portata degli identificatori definiti statica.

Linguaggi Imperativi : Scoping Statico

Funzioni Semantiche

$\mathcal{D}[\![\mathtt{D}]\!]_\rho : Env$ (*Significato delle Dichiarazioni*)

$\mathcal{D}[\![\mathtt{Proc\ I()\ C}]\!]_\rho = \mathtt{bind}(\mathtt{I}, \mathcal{M}[\![\mathtt{C}]\!]_\rho)$

$\mathcal{M}[\![\mathtt{C}]\!]_\rho : \mathtt{State} \to \mathtt{State}$ (*Significato dei Dichiarazioni*)

$\mathcal{M}[\![\mathtt{Call\ I()}]\!]_\rho = \mathtt{find}(\mathtt{I}, \rho)$

Esercizio 13

3.2 Semantica: Linguaggi con Scoping Dinamico

Definiamo il comportamento della dichiarazione di procedura (e funzione) e dell'invocazione rispetto all'ambiente quando la portata degli identificatori definiti dinamica.

Linguaggi Imperativi : Scoping Dinamico	
Funzioni Semantiche	
$\mathcal{D}[\![\texttt{D}]\!]_\rho : Env$	(*Significato delle Dichiarazioni*)
$\mathcal{D}[\![\texttt{Proc I() C}]\!]_\rho = \texttt{bind}(\texttt{I}, \lambda\ \delta . \mathcal{M}[\![\texttt{C}]\!]_\delta)$	
$\mathcal{M}[\![\texttt{C}]\!]_\rho : \texttt{State} \to \texttt{State}$	(*Significato dei Dichiarazioni*)
$\mathcal{M}[\![\texttt{Call I()}]\!]_\rho = \texttt{find}(\texttt{I}, \rho)(\rho)$	

Esercizio 14

3.3 Blocchi: Dichiarazioni Sequenziali, Parallele (simmetriche, mutuamente ricorsive o di punto fisso), Miste

Definiamo lo scope di una dichiarazione all'interno del blocco in cui dichiarata. Sia Iun identificatore dichiarato in un blocco (indifferentemente, In-line o procedura) il suo scope contiene l'intero blocco? Tre possibili risposte:

- Sequenziale: NO - solo quanto nel blocco segue la sua dichiarazione;
- Parallela: SI - anche tutto quello che precede la sua dichiarazione;
- Mista: NO - dipende da cosa stiamo denotando (ad es. no per variabili, si per procedure, si per tipi astratti).

Finora abbiamo omesso di dare significato alla composizione di dichiarazione. arrivato il momento di farlo. Potremmo farlo attribuendo 3 semantiche differenti per le differenti 3 risposte date dai vari linguaggi. Invece, introduciamo un nuovo costrutto per la dichiarazione che pu essere utilizzato tanto per la dichiarazione parallela quanto per quella mista.

Linguaggi : Diachiarazione Sequenziale	
Funzioni Semantiche	
$\mathcal{D}[\![\texttt{D}]\!]_\rho : Env$	(*Significato delle Dichiarazioni*)
$\mathcal{D}[\![\texttt{D}_1\ \texttt{D}_2]\!]_\rho = \mathcal{D}[\![\texttt{D}_2]\!]_{\mathcal{D}[\![\texttt{D}_1]\!]_\rho}$	

Esercizio 15 ***Esercizio 16*** *Si applichi la definizione alla dichiarazione* ricorsione semplice – mostrare fallimento

Notazione 3 *Indichiamo l'operatore di punto fisso con* `Y`*. Pertanto, dato un funzionale* $\mathtt{H} \equiv \lambda\ \mathtt{f.F}$ *nella variabile* `f`*, ovvero* `f` *è una variabile che assume valori sul dominio delle funzioni calcolabili,* `Y f.H` *esprime la più piccola (i.e. meno definita) funzione calcolabile che rende identica la seguente equazione:* $\mathtt{g} = \mathtt{H(g)}$*. Un'abbreviazione molto diffusa con punti fissi di funzionali è quella di contrarre l'operatore* `Y`*con il qualificatore* λ *limitatamente alla variabile funzionale, così da scrivere* `Y f.F` *invece di* `Y f`λ `f.F`*. Ovviamente, solo la seconda è una forma con senso la prima la usiamo solo come abbreviazione. Anche noi faremo così e questo può essere notato nella tabella sulla semantica delle* Dichiarazioni Mutuamente Ricorsive

Osservazione 1 (Calcolo del punto fisso) *Come sappiamo tale funzione pu essere ottenuta come limite di un processo di approssimazioni finite descritto dalla formula di Tarski introdotta nel suo teorema sul punto fisso di equazioni tra funzioni:* $\mathtt{Y\ f.H} = \bigcup_{i\in\aleph} \mathtt{H}^i_\perp$*. La formula qui espressa in forma non dissimile da quella utilizzata nel corso di compilatori per punti fissi di equazioni tra linguaggi. In particolare, l'approssimazione* $\mathtt{H}^{i+1}_\perp \equiv \mathtt{H}(\mathtt{H}^i_\perp)$*, mentre* $\mathtt{H}^0_\perp \equiv \perp$ *(dove* $\perp$*, ovviamente, esprime la funzione ovunque indefinita).*

Esempio 1 *Sia* `if` *l'usuale operatore condizionale a due vie,* `=`*,* `*`*,* `-` *gli usuali, rispettivamente, predicato di uguaglianza, operatore prodotto, operatore sottrazione, tutti su interi e tutti infissi, ed infine 0 ed 1 i neutri di somma e prodotto. Allora:*

$$\mathtt{H} \equiv \lambda\ \mathtt{f}.\lambda\ \mathtt{x.if}(\mathtt{x} = 0, 1, \mathtt{x}*\mathtt{f}(\mathtt{x}-1))$$

è un funzionale che esprime funzioni diverse al variare della variabile `f` *sul dominio delle funzioni. Ad esempio provate ad applicarla alla funzione identit,* $\mathtt{I} \equiv \lambda\ \mathtt{y.\ y}$*, oppure alla funzione successore,* $\mathtt{succ} \equiv \lambda\ \mathtt{y.\ y}+1$*: Dite la funzione espressa da* $\mathtt{H(I)}$*; Dite la funzione espressa da* $\mathtt{H(succ)}$*.*
Torniamo al funzionale `H` *e calcoliamo le prime approssimazioni del suo minimo punto fisso, ovvero di quella funzione* $\mathtt{g} \equiv \mathtt{Y\ f.H}$ *che come abbiamo detto è tale che* $\mathtt{g} = \mathtt{H(g)}$*.*

$$\begin{aligned}
\mathtt{H}^0_\perp &\equiv \perp \\
\mathtt{H}^1_\perp &\equiv \lambda\ \mathtt{x.if}(\mathtt{x} = 0, 1, \perp) \\
\mathtt{H}^2_\perp &\equiv \lambda\ \mathtt{x.if}(\mathtt{x} = 0, 1, \mathtt{x}*\mathtt{H}^1_\perp(\mathtt{x}-1)) \\
&= \lambda\ \mathtt{x.if}(\mathtt{x} = 0, 1, \mathtt{x}*\mathtt{if}((\mathtt{x}-1) = 0, 1, \perp)) \\
&= \lambda\ \mathtt{x.if}(\mathtt{x} = 0, 1, \mathtt{if}((\mathtt{x}-1) = 0, \mathtt{x}*1, \mathtt{x}*\perp)) \\
&= \lambda\ \mathtt{x.if}(\mathtt{x} = 0, 1, \mathtt{if}(\mathtt{x} = 1, 1*1, \perp)) \\
&= \lambda\ \mathtt{x.if}(\mathtt{x} = 0, 1, \mathtt{if}(\mathtt{x} = 1, 1, \perp)) \\
\mathtt{H}^3_\perp &\equiv \lambda\ \mathtt{x.if}(\mathtt{x} = 0, 1, \mathtt{if}(\mathtt{x} = 1, 1, \mathtt{if}(\mathtt{x} = 2, 2, \perp)))
\end{aligned}$$

passo induttivo:

$$\mathtt{H}^{n+1}_\perp \equiv \lambda\ \mathtt{x.if}(\mathtt{x} = 0, 0!, \mathtt{if}(\mathtt{x} = 1, 1!, \mathtt{if}(\mathtt{x} = 2, 2!, ...\mathtt{if}(\mathtt{x} = n, n!, \perp)))...)$$ □

Osservazione 2 (Trasparenza referenziale) *Nel calcolo di* $\mathtt{H}^2_\perp$ *(e, in generale in questo tipo di calcolo algebrico) abbiamo fatto ricorso a varie forme di semplificazioni quali:*

- *Rimpiazzamento di* $(\mathtt{x}-1) = 0$ *con* $\mathtt{x} = 1$*.*
- *Rimpiazzamento di* $\mathtt{x}*\mathtt{if}((\mathtt{x}-1) = 0, 1, \perp))$ *con* $\mathtt{if}((\mathtt{x}-1) = 0, \mathtt{x}*1, \mathtt{x}*\perp)$

Queste semplificazioni sono lecite perchè nel calcolo con cui stiamo esprimendo la semantica denotazionale vale la proprietà nota come Trasparenza Referenziale *che afferma che il significato di un'espressione dipende unicamente dall'espressione stessa e non dal*

contesto in cui essa può occorrere: Pertanto possiamo sempre rimpiazzare un'espressione con un'altra espressione di stesso valore senza che alterare il significato della forma in cui tale espressione compare. Questa proprietà è presente anche in alcuni linguaggi di programmazione quali i Linguaggi Funzionali Puri *(i.e. Haskell, FOL).* □

Esercizio 17 *In che senso il minimo punto fisso,* g, *del funzionale* H *in Esempio 1 è l'unione delle sue approssimazioni* $\bigcup_{i\in\aleph} \mathtt{H}^i_\perp$ *piuttosto che limite di esse? Ovvero perchè volendo il valore* $\mathtt{g}(n)$ *invece di usare tale limite consideriamo un'opportuna approssimazione quale* $\mathtt{H}^{n+1}_\perp$*, nel caso dell'esempio, e diciamo* $\mathtt{g}(n) = \mathtt{H}^{n+1}_\perp(n)$ □

A questo punto siamo in grado di estendere il linguaggio con un costrutto per la dichiarazione parallela o di punto fisso, e di esprimere precisamente il significato relativo. Questo è mostrato nella tabella sotto.

Linguaggi : Dichiarazioni Mutuamente Ricorsive
Domini Sintattici
$\mathtt{D} ::= \ldots \mid \mathtt{Mut\ D_1\ D_2\ Ally} \mid \ldots$ $\quad$ (*Dichiarazioni*)
Funzioni Semantiche
$\mathcal{D}[\![\mathtt{D}]\!]_\rho : Env$ $\quad$ (*Significato delle Dichiarazioni*)
$\mathcal{D}[\![\mathtt{Mut\ D_1\ D_2\ Ally}]\!]_\rho = \mathtt{Y}\ \delta\ .\ (\mathcal{D}[\![\mathtt{D_1}]\!]_\delta \circ \mathcal{D}[\![\mathtt{D_2}]\!]_\delta \circ \rho)$

Esempio 2 *Siano* A *e* B *due identificatori. Consideriamo il seguente frammento di programma:*

```
...
{...
 Mut
   Proc A() Call B();
   Proc B() Call A();
 Ally
...
```

Applichiamo alla sua dichiarazione la semantica definita per il costrutto Mut_Ally*. Otteniamo la seguente funzione:*

$$\mathtt{g} \equiv \mathtt{Y}\ \delta.\ \mathtt{bind}(\mathtt{A}, \mathcal{M}[\![\mathtt{Call\ B()}]\!]_\delta) \circ \mathtt{bind}(\mathtt{B}, \mathcal{M}[\![\mathtt{Call\ A()}]\!]_\delta) \circ \rho$$

Calcoliamo le prime tre approssimazioni. Quindi, consideriamo il funzionale:

$$\mathtt{H} \equiv \lambda\ \delta.\mathtt{bind}(\mathtt{A}, \mathcal{M}[\![\mathtt{Call\ B()}]\!]_\delta) \circ \mathtt{bind}(\mathtt{B}, \mathcal{M}[\![\mathtt{Call\ A()}]\!]_\delta) \circ \rho$$

otteniamo:

$$\mathtt{H}^0_\perp \equiv \perp$$
$$\mathtt{H}^1_\perp \equiv \mathtt{bind}(\mathtt{A}, \mathcal{M}[\![\mathtt{Call\ B()}]\!]_\perp) \circ \mathtt{bind}(\mathtt{B}, \mathcal{M}[\![\mathtt{Call\ A()}]\!]_\perp) \circ \rho$$
$$\mathtt{H}^2_\perp \equiv \mathtt{bind}(\mathtt{A}, \mathcal{M}[\![\mathtt{Call\ B()}]\!]_{\mathtt{H}^1_\perp}) \circ \mathtt{bind}(\mathtt{B}, \mathcal{M}[\![\mathtt{Call\ A()}]\!]_{\mathtt{H}^1_\perp}) \circ \rho$$

Sostituiamo e vediamo cosa otteniamo. □

Esercizio 18 *Si utilizzi il costrutto* Mut_Ally *per scrivere, nel linguaggio finora definito, una procedura che, calcolato il fattoriale del valore associato ad una variable non locale* x*, assegna tale valore ad una variabile non locale* y*.*

Esercizio 19 *Si applichi la semantica della dichiarazione per mostrare che la procedura scritta per l'esercizio precedente calcola la funzione fattoriale*

Esercizio 20 *Si consideri la seguente definizione:*
$\mathcal{D}[\![\texttt{Mut D}_1\ \texttt{D}_2\ \texttt{Ally}]\!]_\rho = \texttt{Y}\ \delta\ .\ (\rho\ \circ\ \mathcal{D}[\![\texttt{D}_1]\!]_\delta\ \circ\ \mathcal{D}[\![\texttt{D}_2]\!]_\delta)$

Esercizio 21 *Si consideri la seguente definizione:*
$\mathcal{D}[\![\texttt{Mut D}_1\ \texttt{D}_2\ \texttt{Ally}]\!]_\rho = \texttt{Y}\ \delta\ .\ (\mathcal{D}[\![\texttt{D}_1]\!]_{\delta\ \circ\ \rho}\ \circ\ \mathcal{D}[\![\texttt{D}_2]\!]_{\delta\ \circ\ \rho})$

Combinando opportunamente il costrutto per la dichiarazione sequenziale con quello per la dichiarazione parallela possiamo riscrivere i programmi di quei linguaggi che utilizzano la dichiarazione mista.