Lezione 13-14

March 14, 2012

Astrazioni di controllo: Trasmissione dei parametri

- Parametri: dove e perchè
- Trasmissione per Valore: FUI
- Trasmissione per Nome (comando e espressione): FUI
- Trasmissione by Need: Variante Implementativa
- Trasmissione di Procedura/Funzione: Variante Implementativa
- Trasmissione per Riferimento: FUI
- Trasmissione per Costante: FUI
- Trasmissione per Risultato: FUI
- Trasmissione per Valore-Risultato: FUI

Parametri: Dove e Perchè

- Dove.
  - Ovunque si abbia un'astrazione (cioè, generalizzazione) si pone problema del suo uso in possibilmente, tanti contesti diversi del programma
  - l'uso dell'astrazione in un contesto richiede un meccanismo di *instanziazione* che colleghi l'astrazione con il contesto, integrandola con esso.
  - il collegamento è realizzato attraverso i parametri formali (con cui è stata creata la astrazione) e i parametri attuali con cui la vogliamo integrare l'astrazione nel contesto.
- Perchè
  - Per instanziare un'astrazione in un dato contesto occorre creare un collegamento tra astrazione e contesto.
  - il collegamento è realizzato attraverso i meccanismi di *trasmissione dei parametri* formali e i parametri attuali.

Convenzioni

- La trasmissione è presentata, per semplicità, in combinazione alla dichiarazione di procedura con un solo parametro, trasmesso con uno dei vari tipi di trasmissione.
- Useremo una funzione semantica $\mathcal{D}_E$ che non modifica lo stato, in aggiunta alla funzione $\mathcal{D}$ che, come sappiamo, modifica lo stato quando deve allocare un modificabile.
- Mostriamo sotto, come $\mathcal{D}_E$ sia estesa in funzione $\mathcal{D}$

Table11 : Convenzioni

Funzioni Semantiche

$$\mathcal{D}[\![\mathtt{D}]\!]_\rho : (\mathtt{Env} \times \mathtt{Store}) \to (\mathtt{Env} \times \mathtt{Store}_\perp)$$
$$\mathcal{D}_E[\![\mathtt{D}]\!]_\rho : (\mathtt{Env} \times \mathtt{Store}) \to \mathtt{Env}_\perp$$
$$\mathcal{D}[\![\mathtt{C}]\!]_\rho = \lambda \mathtt{s}.(\mathcal{D}_E[\![\mathtt{C}]\!]_\rho, \mathtt{s})$$

Trasmissione per Valore/1

Table11.1 – Procedure 2 : Trasmissione per Valore

Funzioni Semantiche

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

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

- il parametro formale ha come denotazione un valore modificabile (l_1)

Trasmissione per Valore/2

Table11.1 – Procedure 2 : Trasmissione per Valore

Funzioni Semantiche

$\mathcal{C}[\![\mathtt{C}]\!]_\rho : \mathtt{Env} \longrightarrow \mathtt{State} \longrightarrow \mathtt{State}_\perp$

$$\mathcal{C}[\![\mathtt{Call\ I(AMem(E))}]\!]_\rho(s) = \mathtt{Let}\{(v_1, s_1) =_{\perp_s} \mathcal{E}[\![\mathtt{E}]\!]_\rho(s),\ f = \rho(\mathtt{I})\}\ f(\mathtt{VM}(v_1))(s_1)$$

Funzioni Ausiliarie

$\mathtt{DV} : \mathtt{Den} \longrightarrow \mathtt{Val}$

$\mathtt{VM} : \mathtt{Val} \longrightarrow \mathtt{Mem}$

- il parametro attuale deve calcolare un valore memorizzabile (VM)

Trasmissione per Valore/3

- Uso.
  - Presente in tutti (o quasi) i linguaggi di programmazione
  - Collegamento one-way. L'invocato ha una copia dei valori memorizzabili del contesto
  - Impiegata in alcune metodologie per trasmettere valori all'invocante e utilizzare il parametro come una temporanea o accumulatore
  - No side effects
- Implementazione.
  - semplice realizzazione della s. denotazionale: legame tra formale e un valore modificabile inizializzato ad una copia del memorizzabile trasmesso

Trasmissione per Nome/1

Table11.2 – Procedure 2 : Trasmissione per Nome

Funzioni Semantiche

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

$$\mathcal{D}_E[\![\mathtt{Proc\ I(byName\ I_1)\ C}]\!]_\rho =$$
$$\mathtt{Let}\{f = \lambda\, v\,.\mathtt{Let}\{\rho_1 = \mathtt{bind}(\mathtt{I}_1, \mathtt{Z}(v), \rho)\}$$
$$\mathcal{M}[\![\mathtt{C}]\!]_{\rho_1}\}$$
$$\mathtt{bind}(\mathtt{I}, f, \rho)$$

Domini Ausiliari

$\mathtt{Den} ::= \mathtt{Loc} + \mathtt{ProcFun} + \mathtt{VL} + \mathtt{Code} \quad (\textit{Valori Codice})$

$\mathtt{Code} ::= (\mathtt{State} \to \mathtt{State}) + (\mathtt{State} \to (\mathtt{Val} \times \mathtt{State}))$

Funzioni Ausiliarie

$\mathtt{Z} : \mathtt{Code} \to \mathtt{Den} \qquad (\textit{iniettiva, i.e. costruttore})$

Trasmissione per Nome/2

Table11.2 – Procedure 2 : Trasmissione per Nome

Domini Sintattici

$\texttt{C} ::= \ldots \mid \texttt{Call Proc I}(\texttt{A}_1 \ \ldots \ \texttt{A}_n)$ (*Invocazione*)

$\mid \texttt{Exec I} \mid \ldots$ (*Valutazione* : *uso del parametro*)

Funzioni Semantiche

$$\mathcal{C}[\![\texttt{Call I}(\texttt{ACodeC}(\texttt{C}))]\!]_\rho = \texttt{Let}\{f = \rho(\texttt{I})\} f(\mathcal{M}[\![\texttt{C}]\!]_\rho)$$

$$\mathcal{C}[\![\texttt{Call I}(\texttt{ACodeE}(\texttt{E}))]\!]_\rho = \texttt{Let}\{f = \rho(\texttt{I})\} f(\mathcal{E}[\![\texttt{E}]\!]_\rho)$$

- il parametro formale ha come denotazione un valore codice ($\texttt{Z}$)
- il parametro attuale deve calcolare un codice, comando ($\mathcal{M}$) o espressione ($\mathcal{E}$)
- il codice trasmesso è chiuso nell'ambiente dell'invocante

Trasmissione per Nome/3

Table11.2 – Procedure 2 : Trasmissione per Nome

Domini Sintattici

$\texttt{C} ::= \ldots\ \texttt{Exec I} \mid \ldots$ (*Valutazione* : *uso del parametro*)

$\texttt{E} ::= \ldots\ \texttt{Val(I)} \mid \texttt{Den(I)} \mid \ldots$

Funzioni Semantiche

$$\mathcal{C}[\![\texttt{Exec I}]\!]_\rho(\texttt{s}) = \texttt{Let}\{\texttt{Z}(v) = \rho(\texttt{I})\}v(\texttt{s})$$

- il codice trasmesso può essere un comando semplice o composto o un'astrazione procedurale (anche anonima) con scope nell'ambiente dell'invocane

Trasmissione per Nome/4

Table11.2 – Procedure 2 : Trasmissione per Nome

Domini Sintattici

$\mathtt{E} ::= \ldots\ \mathtt{Val(I)}\ \ldots$

Funzioni Semantiche

$$\mathcal{E}[\![\mathtt{Val(I)}]\!]_\rho(\mathtt{s}) = \begin{cases} (\mathtt{MV}(\mathtt{s}(l), \mathtt{s}) & \text{if } \rho(\mathtt{I}) \equiv l, \textit{for } l \in \mathtt{Loc} \\ v(\mathtt{s}) & \text{if } \rho(\mathtt{I}) \equiv \mathtt{Z}(v), \\ & \quad \text{for } v \in \mathtt{Store} \to (\mathtt{Val} \times \mathtt{Store})_\perp \\ (\mathtt{DV}(d), \mathtt{s}) & \text{if } \rho(\mathtt{I}) \equiv d, \textit{for } d \in \mathtt{VL} \end{cases}$$

- il codice trasmesso può essere un'espressione o un'astrazione funzionale (anche anonima: *lambda astrazione*) con scope nell'ambiente dell'invocante

Trasmissione per Nome/5

Table11.2 – Procedure 2 : Trasmissione per Nome

Domini Sintattici

$\mathtt{E} ::= \ldots \mathtt{Den(I)} \mid \ldots$

Funzioni Semantiche

$$\mathcal{E}[\![\mathtt{Den(I)}]\!]_\rho(\mathtt{s}) = \begin{cases} (l, \mathtt{s}) & \text{if } (\rho(\mathtt{I}) \equiv l), l \in \mathtt{Loc} \\ (l, \mathtt{s}) & \text{if } (\rho(\mathtt{I}) \equiv \mathtt{Z}(l)), l \in \mathtt{Loc} \\ (\perp_D, \mathtt{s}) & \text{otherwise} \end{cases}$$

- il codice trasmesso può essere un'espressione denotabile ed usata da invocante/invocato per condividere un valore memorizzabile

Trasmissione per Nome/6

- Uso.
  - Presente in molti linguaggi di programmazione PASCAL, C♯, SCALA, Ocaml, Haskell, sebbene in forme più ristrette.
  - L'invocato ha una un codice dell'invocante (cioè nello scope dell'ambiente dell'invocante)
  - Side effects (attraverso l'ambiente accessibile dal codice trasmesso)
  - Aliasing (attraverso l'ambiente contemporaneamente accessibile dal codice trasmesso e dall'invocato)
  - Le astrazioni procedurali e/o funzionali possono essere fatte rispetto a codice e non solo dati
    - Call Back. Trasmettere codice diverso a seconda dello stato in cui eseguiamo l'invocazione (event programming)
    - Higher Order. Funzioni come valori di prima classe

Trasmissione per Nome/7

- Uso.
  - Impiegata anche in alcune metodologie come trasmissione base con la quale emulare vari tipi di trasmissione (incluso anche trasmissione quali quella per valore e per riferimento, in quest'ultimo caso occorre fare attenzione all'aliasing)
- Implementazione.
  - Estendere ambiente per un nuovo tipo di denotabili (Z).
  - **Thunk** per implementare Z: Si utilizza una struttura tipo *chiusura* in grado di contenere un codice e l'ambiente dei suoi bindings.
    - il codice nel thunk non è un'astrazione (procedura/funzione)
    - l'ambiente è quello dell'invocante (*shallow bindings*)
  - Valutare questi valori richiede l'uso di AR come per i blocchi

Trasmissione by Need/1

- Variante Implementativa della trasmissione by Name
  - by name: il thunk è eseguito ad ogni occorrenza (`Exec I`, `Val(I)`, `Den(I)`) del formale nel corpo dell'invocato.
    - Ogni esecuzione potendo risultare in un differente side effect e/o in un differente valore calcolato
    - Tutto ciò al costo di una nuova valutazione del codice trasmesso
  - by need: il thunk è eseguito sollo alla prima occorrenza (`Exec I`, `Val(I)`, `Den(I)`) del formale incontrata nella valutazione del corpo del chiamato
    - Ogni successiva occorrenza userà il valore giá calcolato
    - Tutto ciò al costo di limitare sostanzialmente il codice trasmesso alle sole espressioni
- Da un punto di vista formale la call by need è abbastanza *strana* giacchè conduce a valutazioni che modificano l'ambiente (di valutazione)

Trasmissione by Need/2

- Uso e Implementazione
  - Nei linguaggi con trasparenza referenziale delle espressioni (come Haskell) call by need coincide con call by name di espressioni (che non calcolano funzioni)
  - Nei linguaggi funzionali basati sulla nozione di *riduzione di termine* la call by need:
    - è la forma di trasmissione usata al posto del *by value* per espressioni che calcolano valori che non sono una funzione
    - è una naturale forma di trasmissione per valutazioni esterne: le funzioni sono tutte non strette
    - ha una naturale implementazione basato su *graph reduction*
  - In linguaggi senza trasparenza referenziale non si usa.

Trasmissione by Need/3

- Meriti della call by need: Ormai quasi ovunque rimpiazzata da altri meccanismi di cui è stata:
  modello base per lo studio e lo sviluppo
  - Demand driven Evaluation (linguaggi per processi concorrenti/cooperanti)
  - Lazy Evaluation (linguaggi funzionali moderni)

Trasmissione Procedure-Funzione/1

- Variante Implementativa della trasmissione by Name
  - by name: qualunque codice può essere trasmesso
    - Richiede una nuova classe di valori denotabili (`Code`)
    - Richiede una nuova struttura per l'implementazione (Thunk)
    - Richiede vari accorgimenti nelle valutazione e gestione degli AR
    - Astrazioni non sempre chiare: Quale funzionalità dell'algoritmo implementato è realizzata dal codice che trasmettiamo?
  - Procedure-Funzione: solo procedure e/o solo funzioni possono essere trasmesse

Trasmissione Procedure-Funzione/2

Variante Implementativa della trasmissione by Name

- by name: qualunque codice può essere trasmesso
- Procedure-Funzione: solo procedure e/o solo funzioni possono essere trasmesse
  - astrazioni:
    - con naming legato in qualche ambiente ed avente l'invocante nel suo scope
    - anonime ma definite (tipicamente nell'invocante) con un meccanismo di astrazione funzionale
  - La classe dei valori denotabile (`ProcFun`) è già presente
  - È implementata con le chiusure già utilizzate per procedure e funzioni dichiarate
  - Usa AR identici a quelli usati per tutte le invocazioni in cui non è trasmesso come attuale
  - Astrazioni chiare: nella misura in cui le proedure e funzioni del linguaggio sono *unità di programmazione* e come tali esprimono una ben definita funzionalità dell'algoritmo.

Trasmissione Procedure-Funzione/3

- Uso
  - Le astrazioni procedurali e/o funzionali possono essere fatte rispetto a procedure/funzioni
    - Call Back. Trasmettere procedura/funzione diversa a seconda dello stato in cui eseguiamo l'invocazione (event programming)
    - Higher Order. Funzioni come valori di prima classe
- Implementazione.
  - Si utilizza la *chiusura* e la si realizza con gli AR (già studiati)
  - Non sempre usiamo la stessa chiusura della dichiarazione:
    - shallow binding: ambiente dell'invocante
    - deep binding: ambiente di dichiarazione

Trasmissione by Reference/1

Table11.3 – Procedure 2 : Trasmissione per Riferimento

Funzioni Semantiche

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

$$\mathcal{D}_E[\![\mathtt{Proc\ I(byReference\ I_1)\ C}]\!]_\rho =$$
$$\mathtt{Let}\{f = \lambda\ v\,.\mathtt{Let}\{\rho_1 = \mathtt{bind}(\mathtt{I}_1, v, \rho)\}\mathcal{M}[\![\mathtt{C}]\!]_{\rho_1}\}$$
$$\mathtt{bind}(\mathtt{I}, f, \rho)$$

- il parametro formale ha come denotazione un valore denotabile (apparentemente arbitrario)

Trasmissione by Reference/2

Table11.3 – Procedure 2 : Trasmissione per Riferimento

Funzioni Semantiche

$$\mathcal{C}[\![\texttt{Call I(Den(E))}]\!]_\rho(s) = \\ \texttt{Let}\{(l_1, s_1) =_{\perp_S} \mathcal{E}[\![\texttt{Den(E)}]\!]_\rho(s),\ f = \rho(\texttt{I})\} \\ \texttt{if}((l_1 \in \texttt{Loc}), f(l_1)(s_1), \perp_{\texttt{Store}})$$

Funzioni Ausiliarie

$\in \texttt{Loc} : \texttt{Den} \longrightarrow \texttt{TruthV}$

- il parametro attuale deve essere un valore modificabile e denotabile (Den)

Trasmissione by Reference/3

- Uso.
  - Presente in molti linguaggi di programmazione Imperativa: Algol, Pascal,
  - Emulata in altri attraverso l'uso di puntatori: C, C++
  - Re-introdotta in varianti di linguaggi che ne erano privi: C♯
  - Collegamento two-way. L'invocato e l'invocante condividono l'accesso ad un valore modificabile
  - Impiegata in alcune metodologie per trasmettere valori all'invocante e ottenere i valori calcolati dalla sua valutazione
  - side effects
  - Aliasing. Possibile se invocato è nello scope di un binding per il valore modificabile trasmesso.
- Implementazione.
  - semplice realizzazione della s. denotazionale: legame tra formale e denotabile (modificabile) trasmesso

Trasmissione by Constant/1

Table11.4 – Procedure 2 : Trasmissione per Costante

Funzioni Semantiche

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

$\mathcal{D}_E[\![\texttt{Proc I(byConst I}_1\texttt{) C}]\!]_\rho =$
$\quad \texttt{Let}\{f = \lambda\ v\,.\texttt{Let}\{\rho_1 = \texttt{bind}(\texttt{I}_1, \texttt{VD}(v), \rho)\}\mathcal{M}[\![\texttt{C}]\!]_{\rho_1}\}$
$\quad \texttt{bind}(\texttt{I}, f, \rho)$

Funzioni Ausiliarie

$\texttt{VD} : \texttt{Val} \longrightarrow \texttt{Den}$

- il parametro formale ha come denotazione un valore non modificabile (VD)

Trasmissione by Constant/2

Table11.4 – Procedure 2 : Trasmissione per Costante

Funzioni Semantiche

$\mathcal{C}[\![\mathtt{C}]\!]_\rho : \mathtt{Env} \longrightarrow \mathtt{State} \longrightarrow \mathtt{State}_\perp$

$$\mathcal{C}[\![\mathtt{Call\ I(AConst(E))}]\!]_\rho(s) = \mathtt{Let}\{(v_1, s_1) =_{\perp_S} \mathcal{E}[\![\mathtt{E}]\!]_\rho(s),\ f = \rho(\mathtt{I})\} \ \mathtt{in\ if}((v_1 \in \mathtt{Val}), f(v_1)(s_1), \perp_{\mathtt{Store}})$$

Funzioni Ausiliarie

$\in \mathtt{Den} : \mathtt{Val} \longrightarrow \mathtt{TruthV}$

- il parametro attuale deve calcolare un denotabile, non modificabile (VD)

Trasmissione by Constant/3

- Uso.
  - Presente in pochi linguaggi di programmazione a causa dei forti limiti sui valori trasmessi
  - Collegamento one-way. L'invocato ha una copia dei valori denotabili non modificabili
  - Impiegata in alcune metodologie per trasmettere valori all'invocante e
  - Garantire l'assoluta separazione tra invocante e invocato (correttezza)
  - No side effects
- Implementazione.
  - semplice realizzazione della s. denotazionale: legame tra formale e il valore non modificabile trasmesso

Trasmissione by Result/1

Table11.5 – Procedure 2 : Trasmissione per Risultato

Funzioni Semantiche

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

$$\begin{aligned}
&\mathcal{D}_E[\![\mathtt{Proc\ I(byResult\ I_1)\ C}]\!]_\rho = \\
&\quad \mathtt{Let}\{f = \lambda\, v\,.\lambda\, s\,. \\
&\qquad \mathtt{Let}\{(l_1, s_1) = \mathit{allocate}(s)\} \\
&\qquad\quad \{s_2 = \mathtt{upd}(l_1, \perp_{\mathtt{Mem}}, s_1), \rho_1 = \mathtt{bind}(\mathtt{I}_1, l_1, \rho)\} \\
&\qquad\quad (\mathcal{M}[\![\mathtt{C}]\!]_{\rho_1} \circ (\lambda u\,.\mathtt{upd}(v, \mathtt{look}(l_1, u), u)))(s_2) \\
&\quad \mathtt{bind}(\mathtt{I}, f, \rho)
\end{aligned}$$

- il parametro formale ha come denotazione un valore modificabile con valore, memorizzabile associato, indefinito.

Trasmissione by Result/2

Table11.4 – Procedure 2 : Trasmissione per Risultato

Funzioni Semantiche

$\mathcal{C}[\![\mathtt{C}]\!]_\rho : \mathtt{Env} \longrightarrow \mathtt{State} \longrightarrow \mathtt{State}_\perp$

$$\mathcal{C}[\![\mathtt{Call\ I(Den(E))}]\!]_\rho(s) = \mathtt{Let}\{(l_1, s_1) =_{\perp_S} \mathcal{E}[\![\mathtt{Den(E)}]\!]_\rho(s),\ f = \rho(\mathtt{I})\}\ \mathtt{if}((l_1 \in \mathtt{Loc}), f(l_1)(s_1), \perp_{\mathtt{Store}})$$

Funzioni Ausiliarie

$\in \mathtt{Loc} : \mathtt{Den} \longrightarrow \mathtt{TruthV}$

- il parametro attuale deve calcolare un valore, denotabile, modificabile.

Trasmissione by Result/3

- Uso.
  - Presente oggi, in pochi linguaggi di programmazione.
  - Collegamento one-way. L'invocante riceve una copia del valore, denotabile, modificabile, associato dall'invocato al formale, al termine dell'invocazione
  - Impiegata in alcune metodologie per ricevere i valori calcolati dall'invocato e
  - Garantire l'assoluta separazione tra invocante e invocato (correttezza)
  - no side effects
- Implementazione.
  - semplice realizzazione della s. denotazionale: legame tra formale e denotabile modificabile trasmesso (all'indietro) ma non accesibile

Trasmissione by Value-Result/1

Table11.5 – Procedure 2 : Trasmissione per Valore – Risultato

Funzioni Semantiche

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

$$\mathcal{D}_E[\![\mathtt{Proc\ I(byValueResult\ I_1)\ C}]\!]_\rho =$$
$$\mathtt{Let}\{f = \lambda v\,.\,\lambda \mathtt{s}\,.$$
$$\mathtt{Let}\{(l_1, s_1) = \mathit{allocate}(s)\}$$
$$\{s_2 = \mathtt{upd}(l_1, \mathtt{look}(v, s_1), s_1)$$
$$, \rho_1 = \mathtt{bind}(\mathtt{I}_1, l_1, \rho)\}$$
$$(\mathcal{M}[\![\mathtt{C}]\!]_{\rho_1} \circ (\lambda u\,.\,\mathtt{upd}(v, \mathtt{look}(l_1, u), u)))(s_2)$$
$$\mathtt{bind}(\mathtt{I}, f, \rho)$$

- il formale ha come denotabile un modificabile l_1 con associato una copia del valore associato al modificabile v, trasmesso ($\mathtt{look}(v,s_1)$)
- l'attuale riceve una copia del valore associato a l_1.

Trasmissione by Value-Result/2

Table11.4 – Procedure 2 : Trasmissione per Risultato

Funzioni Semantiche

$\mathcal{C}[\![\mathtt{C}]\!]_\rho : \mathtt{Env} \longrightarrow \mathtt{State} \longrightarrow \mathtt{State}_\perp$

$$\mathcal{C}[\![\mathtt{Call\ I(Den(E))}]\!]_\rho(s) = \mathtt{Let}\{(l_1, s_1) =_{\perp_S} \mathcal{E}[\![\mathtt{Den(E)}]\!]_\rho(s),\ f = \rho(\mathtt{I})\} \mathtt{\ in\ if}((l_1 \in \mathtt{Loc}), f(l_1)(s_1), \perp_{\mathtt{Store}})$$

- il parametro attuale deve calcolare un valore, denotabile, modificabile.

Trasmissione by Value-Result/3

- Uso.
  - Presente oggi, in pochi linguaggi di programmazione.
  - Collegamento Two-way. L'invocante trasmette una copia del valore associato ad un suo modificabile e riceve su tale invocante, come valore associato, una copia del valore associato, al termine dell'invocazione, dall'invocato al modificabile del formale.
  - Impiegata in alcune metodologie per tramettere valori per l'invocato e ricevere i valori calcolati da esso
  - Garantire l'assoluta separazione tra invocante e invocato (correttezza)
  - no side effects
- Implementazione.
  - semplice realizzazione della s. denotazionale