Linguaggi di Programmazione con Laboratorio

Seminario di fine corso: Trasmissione di funzioni e procedure con Deep Binding in `Small21`

Francesco Caporali

Università di Pisa, Dip. di Matematica

28 luglio 2021

Trasmissione di funzioni/procedure in Small21

$$\texttt{fpars} = [\texttt{fp}]\ [\texttt{funproc}]\ \texttt{Type}\ \texttt{Ide}$$

- Come già fatto per le altre tipologie di trasmissione dei parametri presenti in Small21 si vuole introdurre la *trasmissione di funzioni e procedure con Deep Binding*.
- Questa estensione permette di introdurre funzioni e procedure come parametri formali ad ogni livello di annidamento di un programma.

Trasmissione di funzioni/procedure in Small21

$$\texttt{fpars} = [\texttt{fp}]\ [\texttt{funproc}]\ \texttt{Type}\ \texttt{Ide}$$

- Come già fatto per le altre tipologie di trasmissione dei parametri presenti in Small21 si vuole introdurre la *trasmissione di funzioni e procedure con Deep Binding*.
- Questa estensione permette di introdurre funzioni e procedure come parametri formali ad ogni livello di annidamento di un programma.

Trasmissione di funzioni/procedure in Small21

$$\texttt{fpars} = [\texttt{fp}]\ [\texttt{funproc}]\ \texttt{Type}\ \texttt{Ide}$$

La trasmissione di funzioni e procedure è strutturata come segue:

- il **parametro attuale** è un'espressione che calcola un identificatore di funzione/procedura;
- il **parametro formale** viene legato alla chiusura del parametro attuale che differirà in accordo con le regole di *Binding del linguaggio*.

Trasmissione di funzioni/procedure in Small21

$$\texttt{fpars} = [\texttt{fp}]\ [\texttt{funproc}]\ \texttt{Type}\ \texttt{Ide}$$

La trasmissione di funzioni e procedure è strutturata come segue:

- il **parametro attuale** è un'espressione che calcola un identificatore di funzione/procedura;
- il **parametro formale** viene legato alla chiusura del parametro attuale che differirà in accordo con le regole di *Binding del linguaggio*.

Trasmissione di funzioni/procedure in Small21

`fpars` = [`fp`] [`funproc`] `Type Ide`

La trasmissione di funzioni e procedure è strutturata come segue:

- il **parametro attuale** è un'espressione che calcola un identificatore di funzione/procedura;
- il **parametro formale** viene legato alla chiusura del parametro attuale che differirà in accordo con le regole di *Binding del linguaggio*.

Regola di Binding considerata

In questa estensione viene implementato il **Deep Binding**.

Il legame che viene a crearsi è tra il *parametro formale* e una *chiusura* del parametro attuale formata al momento della chiamata, costituita da:

- *definizione del parametro attuale*;
- *ambiente non locale* dello stesso.

Nota

Sebbene questa non sia l'unica regola di binding che viene comunemente adottata è bene osservare che essendo `Small21` un linguaggio con scoping statico questa è la scelta più naturale.

Regola di Binding considerata

In questa estensione viene implementato il **Deep Binding**.

Il legame che viene a crearsi è tra il *parametro formale* e una *chiusura* del parametro attuale formata al momento della chiamata, costituita da:

- *definizione del parametro attuale*;
- *ambiente non locale* dello stesso.

Nota

Sebbene questa non sia l'unica regola di binding che viene comunemente adottata è bene osservare che essendo `Small21` un linguaggio con scoping statico questa è la scelta più naturale.

Cambiamenti di `Small21` dopo l'estensione

I principali cambiamenti del linguaggio sono i seguenti:

- Dopo le modifiche è possibile avere parametri attuali e formali di tipo funzione/procedura, non più solo `Simple`;
- La trasmissioni di funzioni contribuisce all'**espressività** del linguaggio creando la possibilità di parametrizzare delle computazioni all'interno di altre computazioni;
- Non sono presenti ulteriori vincoli né limitazioni: con l'attuale struttura di `Small21` non sorgono difficoltà nell'introduzione della nuova tipologia di trasmissione.

Cambiamenti di `Small21` dopo l'estensione

I principali cambiamenti del linguaggio sono i seguenti:

- Dopo le modifiche è possibile avere parametri attuali e formali di tipo funzione/procedura, non più solo `Simple`;
- La trasmissioni di funzioni contribuisce all'**espressività** del linguaggio creando la possibilità di parametrizzare delle computazioni all'interno di altre computazioni;
- Non sono presenti ulteriori vincoli né limitazioni: con l'attuale struttura di `Small21` non sorgono difficoltà nell'introduzione della nuova tipologia di trasmissione.

Cambiamenti di `Small21` dopo l'estensione

I principali cambiamenti del linguaggio sono i seguenti:

- Dopo le modifiche è possibile avere parametri attuali e formali di tipo funzione/procedura, non più solo `Simple`;
- La trasmissioni di funzioni contribuisce all'**espressività** del linguaggio creando la possibilità di parametrizzare delle computazioni all'interno di altre computazioni;
- Non sono presenti ulteriori vincoli né limitazioni: con l'attuale struttura di `Small21` non sorgono difficoltà nell'introduzione della nuova tipologia di trasmissione.

Cambiamenti di `Small21` dopo l'estensione

I principali cambiamenti del linguaggio sono i seguenti:

- Dopo le modifiche è possibile avere parametri attuali e formali di tipo funzione/procedura, non più solo `Simple`;
- La trasmissioni di funzioni contribuisce all'**espressività** del linguaggio creando la possibilità di parametrizzare delle computazioni all'interno di altre computazioni;
- Non sono presenti ulteriori vincoli né limitazioni: con l'attuale struttura di `Small21` non sorgono difficoltà nell'introduzione della nuova tipologia di trasmissione.

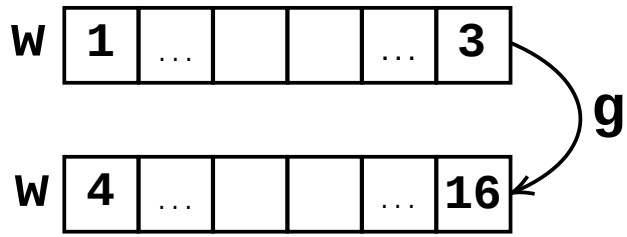
Linguaggi con trasmissione di funzioni/procedure: espressività

```
1  Program example_map{
2      void map(int(int) f, int[6] v){
3          for (i = 1 to 6){
4              v[i] = f(v[i]);
5          }
6      }
7      int g(int y){
8          return (y*y + 2*y + 1);
9      }
10     int[6] w;
11     w[0] = 1;
12     ...
13     w[5] = 3;
14     map(g,w);
15 }
```

`Codice:` codice di un programma che implementa la funzione `map`.

example_map

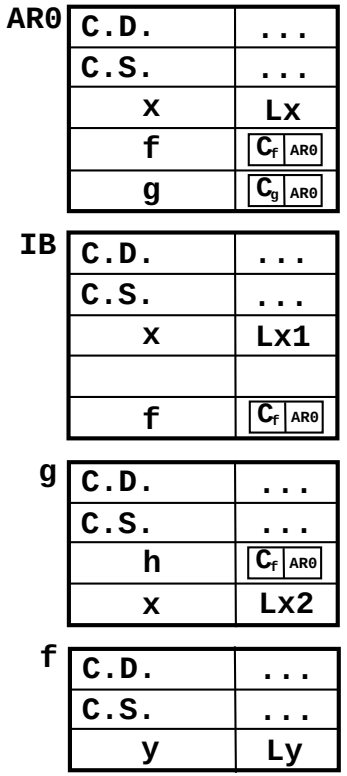
Tra gli esempi più comuni di estensione dell'espressività si ha la possibilità di scrivere una funzione che dato un array e una funzione permetta di applicare la funzione ad ogni elemento dell'array.



Trasmissione di funzioni/procedure: esempio

```
1  Program example_1{
2      int x = 1;
3      int f(value int y){
4          return (x + y);
5      }
6      int g(int(int) h){
7          int x = 2;
8          return (h(3) + x);
9      }
10         {
11         int x = 4;
12         int z = g(f);
13         x = (x + z);
14         }
15 }
```

Codice: codice di example_1 scritto in Small21 che fa uso di trasmissione di funzione con Deep Binding.



Stack di AR: rappresentazione degli AR nella prima parte dell'esecuzione del programma.

Trasmissione di funzioni/procedure: esempio

```
1  Program example_1{
2      int x = 1;
3      int f(value int y){
4          return (x + y);
5      }
6      int g(int(int) h){
7          int x = 2;
8          return (h(3) + x);
9      }
10         {
11         int x = 4;
12         int z = g(f);
13         x = (x + z);
14         }
15 }
```

Codice: codice di example_1 scritto in Small21 che fa uso di trasmissione di funzione.

```
1  #include <stdio.h>
2
3  int x = 1;
4  int f(int y){
5      return (x + y);
6  }
7  int g(int (*h)(int)){
8      int x = 2;
9      return ((*h)(3) + x);
10 }
11 int main(){
12     {
13         int x = 4;
14         int z = g(f);
15         x = (x + z);
16     }
17
18     return 0;
19 }
```

Codice: codice di example_1 scritto in C che fa uso di trasmissione di funzione (per mezzo di puntatori).

Trasmissione di funzioni/procedure: limitazioni del C

```
1  Program example_2{
2      int x = 0;
3      void g(int(int) h){
4          int l(value int y){
5              return ((z + y) + 1);
6          }
7          int z = 1;
8
9          x = h(z);
10         z = 0;
11         if (x == 1) g(l);
12     }
13     int f(value int y){
14         return 1;
15     }
16     g(f);
17 }
```

Codice: codice di `example_2` scritto in `Small21` che fa uso di trasmissione di funzione e dichiarazione di funzione dentro un blocco funzione.

Limitazioni di C

Il linguaggio C permette la trasmissione di funzioni tramite puntatori:

- trasmettendo una funzione si passa in realtà un puntatore al parametro attuale;
- è consentita la dichiarazione di funzioni soltanto nell'ambiente globale, ciò limita e semplifica il problema in questione;

Trasmissione di funzioni/procedure: limitazioni del C

```
1  Program example_2{
2      int x = 0;
3      void g(int(int) h){
4          int l(value int y){
5              return ((z + y) + 1);
6          }
7          int z = 1;
8
9          x = h(z);
10         z = 0;
11         if (x == 1) g(l);
12     }
13     int f(value int y){
14         return 1;
15     }
16     g(f);
17 }
```

Codice: codice di example_2 scritto in Small21 che fa uso di trasmissione di funzione e dichiarazione di funzione dentro un blocco funzione.

Limitazioni di C

Il linguaggio C permette la trasmissione di funzioni tramite puntatori:

- trasmettendo una funzione si passa in realtà un puntatore al parametro attuale;
- è consentita la dichiarazione di funzioni soltanto nell'ambiente globale, ciò limita e semplifica il problema in questione;

Sintassi Concreta e Sintassi Astratta

- **Sintassi Concreta: una CFG per Small21**

  ...

  TypeFP → Type(TypeFP) | Type() | Type

  Fp → ... | TypeFP ide

  PPF → value | ref | funproc

  ...

  Vincoli Contestuali

  ...

  - parametri di tipo Simple per trasmissioni per valori e referenza
  - parametri di tipo funzione/procedura per trasmissioni di funzioni e procedure
  - le funzioni possono restituire solo tipi Simple

  ...

- **Sintassi Astratta:**

  ...

  PPF ::= [value] | [ref] | [funproc]

Sintassi Concreta e Sintassi Astratta

- **Sintassi Concreta: una CFG per Small21**

  ...

  TypeFP → Type(TypeFP) | Type() | Type

  Fp → ... | TypeFP ide

  PPF → value | ref | funproc

  ...

  Vincoli Contestuali

  ...

  - parametri di tipo Simple per trasmissioni per valori e referenza
  - parametri di tipo funzione/procedura per trasmissioni di funzioni e procedure
  - le funzioni possono restituire solo tipi Simple

  ...

- **Sintassi Astratta:**

  ...

  PPF ::= [value] | [ref] | [funproc]

Sintassi Concreta e Sintassi Astratta

- **Sintassi Concreta: una CFG per Small21**

  ...

  TypeFP → Type(TypeFP) | Type() | Type

  Fp → ... | TypeFP ide

  PPF → value | ref | funproc

  ...

  Vincoli Contestuali

  ...

  - parametri di tipo Simple per trasmissioni per valori e referenza
  - parametri di tipo funzione/procedura per trasmissioni di funzioni e procedure
  - le funzioni possono restituire solo tipi Simple

  ...

- **Sintassi Astratta:**

  ...

  PPF ::= [value] | [ref] | [funproc]

Sintassi Concreta e Sintassi Astratta

- **Sintassi Concreta: una CFG per Small21**
- **Sintassi Astratta:**

```
...
PPF ::= [value] | [ref] | [funproc]
...
```

Sintassi Concreta e Sintassi Astratta

- **Sintassi Concreta: una CFG per Small21**
- **Sintassi Astratta:**

  ...

  PPF ::= [value] | [ref] | [funproc]

  ...

Sistema Y: DCL

$$\texttt{Dcl} ::= \ldots \mid \texttt{[pcd] Type Ide FPars BlockS} \mid \ldots$$

$$\texttt{ABS} = \{\texttt{[abs] t},\ \texttt{[abs] t[::]t}' \mid \texttt{t} \in \texttt{Simple} \cup \texttt{[void]}, \texttt{t}' \in \texttt{ABS} \cup \texttt{Simple}\}$$

$$[\text{Y5}]\ \frac{\begin{array}{c}\texttt{t} \in \texttt{Simple} \cup \{\texttt{[void]}\} \quad \texttt{F} = \texttt{[fp] p t}' \texttt{ I}' \quad \texttt{p} \neq \texttt{[funproc]} \quad \texttt{t}' \in \texttt{Simple} \\ \texttt{Y}_\rho|_0(\texttt{I}) = \bot \quad > [\texttt{I}'/\texttt{t}'] \circ [] :: \texttt{Y}_\rho] = \texttt{Y}'_\rho \quad \langle \texttt{Bs}, \texttt{Y}'_\rho \rangle \rightarrow_\texttt{Y} (\texttt{[void]}, \texttt{Y}''_\rho)\end{array}}{\langle \texttt{[pcd] t I F Bs}, \texttt{Y}_\rho \rangle \rightarrow_\texttt{Y} (\texttt{[void]}, [\texttt{I}/\texttt{[abs] t[::]t}'] \otimes \texttt{Y}_\rho)}$$

$$[\text{Y5.1}]\ \frac{\begin{array}{c}\texttt{t} \in \texttt{Simple} \cup \{\texttt{[void]}\} \quad \texttt{F} = \texttt{[fp] p t}' \texttt{ I}' \quad \texttt{p} = \texttt{[funproc]} \quad \texttt{t}' \in \texttt{ABS} \\ \texttt{Y}_\rho|_0(\texttt{I}) = \bot \quad > [\texttt{I}'/\texttt{t}'] \circ [] :: \texttt{Y}_\rho] = \texttt{Y}'_\rho \quad \langle \texttt{Bs}, \texttt{Y}'_\rho \rangle \rightarrow_\texttt{Y} (\texttt{[void]}, \texttt{Y}''_\rho)\end{array}}{\langle \texttt{[pcd] t I F Bs}, \texttt{Y}_\rho \rangle \rightarrow_\texttt{Y} (\texttt{[void]}, [\texttt{I}/\texttt{[abs] t[::]t}'] \otimes \texttt{Y}_\rho)}$$

Nota:
l'insieme ABS e la regola Y5.1 sono stati introdotti per eseguire un controllo sul tipo del parametro formale nella trasmissione di funzione/procedura.

Sistema Y: DCL

$$\texttt{Dcl} ::= \ldots \mid \texttt{[pcd] Type Ide FPars BlockS} \mid \ldots$$

$$\texttt{ABS} \;=\; \{\texttt{[abs] t},\ \texttt{[abs] t[::]t}' \mid \texttt{t} \in \texttt{Simple} \cup \texttt{[void]}, \texttt{t}' \in \texttt{ABS} \cup \texttt{Simple}\}$$

$$[\text{Y5}]\ \frac{\begin{array}{c}\texttt{t} \in \texttt{Simple} \cup \{\texttt{[void]}\} \quad \texttt{F} = \texttt{[fp] p t}' \texttt{ I}' \quad \texttt{p} \neq \texttt{[funproc]} \quad \texttt{t}' \in \texttt{Simple} \\ \texttt{Y}_\rho|_0(\texttt{I}) = \bot \quad > [\texttt{I}'/\texttt{t}'] \circ [] :: \texttt{Y}_\rho] = \texttt{Y}'_\rho \quad \langle \texttt{Bs}, \texttt{Y}'_\rho\rangle \rightarrow_\texttt{Y} (\texttt{[void]}, \texttt{Y}''_\rho)\end{array}}{\langle \texttt{[pcd] t I F Bs}, \texttt{Y}_\rho\rangle \rightarrow_\texttt{Y} (\texttt{[void]}, [\texttt{I}/\texttt{[abs] t[::]t}'] \otimes \texttt{Y}_\rho)}$$

$$[\text{Y5.1}]\ \frac{\begin{array}{c}\texttt{t} \in \texttt{Simple} \cup \{\texttt{[void]}\} \quad \texttt{F} = \texttt{[fp] p t}' \texttt{ I}' \quad \texttt{p} = \texttt{[funproc]} \quad \texttt{t}' \in \texttt{ABS} \\ \texttt{Y}_\rho|_0(\texttt{I}) = \bot \quad > [\texttt{I}'/\texttt{t}'] \circ [] :: \texttt{Y}_\rho] = \texttt{Y}'_\rho \quad \langle \texttt{Bs}, \texttt{Y}'_\rho\rangle \rightarrow_\texttt{Y} (\texttt{[void]}, \texttt{Y}''_\rho)\end{array}}{\langle \texttt{[pcd] t I F Bs}, \texttt{Y}_\rho\rangle \rightarrow_\texttt{Y} (\texttt{[void]}, [\texttt{I}/\texttt{[abs] t[::]t}'] \otimes \texttt{Y}_\rho)}$$

Nota:
l'insieme ABS e la regola Y5.1 sono stati introdotti per eseguire un controllo sul tipo del parametro formale nella trasmissione di funzione/procedura.

Sistema Y: DCL

$$\texttt{Dcl} ::= \ldots \mid \texttt{[pcd] Type Ide FPars BlockS} \mid \ldots$$

$$\texttt{ABS} \;=\; \{\texttt{[abs] t},\ \texttt{[abs] t[::]t}' \mid \texttt{t} \in \texttt{Simple} \cup \texttt{[void]}, \texttt{t}' \in \texttt{ABS} \cup \texttt{Simple}\}$$

$$\text{[Y5]}\ \frac{\begin{array}{c}\texttt{t} \in \texttt{Simple} \cup \{\texttt{[void]}\} \quad \texttt{F} = \texttt{[fp] p t}' \texttt{ I}' \quad \texttt{p} \neq \texttt{[funproc]} \quad \texttt{t}' \in \texttt{Simple} \\ \texttt{Y}_\rho|_0(\texttt{I}) = \bot \quad > [\texttt{I}'/\texttt{t}'] \circ [] :: \texttt{Y}_\rho] = \texttt{Y}'_\rho \quad \langle \texttt{Bs}, \texttt{Y}'_\rho\rangle \to_\texttt{Y} (\texttt{[void]}, \texttt{Y}''_\rho)\end{array}}{\langle \texttt{[pcd] t I F Bs}, \texttt{Y}_\rho\rangle \to_\texttt{Y} (\texttt{[void]}, [\texttt{I}/\texttt{[abs] t[::]t}'] \otimes \texttt{Y}_\rho)}$$

$$\text{[Y5.1]}\ \frac{\begin{array}{c}\texttt{t} \in \texttt{Simple} \cup \{\texttt{[void]}\} \quad \texttt{F} = \texttt{[fp] p t}' \texttt{ I}' \quad \texttt{p} = \texttt{[funproc]} \quad \texttt{t}' \in \texttt{ABS} \\ \texttt{Y}_\rho|_0(\texttt{I}) = \bot \quad > [\texttt{I}'/\texttt{t}'] \circ [] :: \texttt{Y}_\rho] = \texttt{Y}'_\rho \quad \langle \texttt{Bs}, \texttt{Y}'_\rho\rangle \to_\texttt{Y} (\texttt{[void]}, \texttt{Y}''_\rho)\end{array}}{\langle \texttt{[pcd] t I F Bs}, \texttt{Y}_\rho\rangle \to_\texttt{Y} (\texttt{[void]}, [\texttt{I}/\texttt{[abs] t[::]t}'] \otimes \texttt{Y}_\rho)}$$

Nota:
l'insieme ABS e la regola Y5.1 sono stati introdotti per eseguire un controllo sul tipo del parametro formale nella trasmissione di funzione/procedura.

Sistema Y: DCL

$$\texttt{Dcl} ::= \ldots \mid \texttt{[pcd] Type Ide FPars BlockS} \mid \ldots$$

$$\texttt{ABS} \;=\; \{\texttt{[abs] t},\ \texttt{[abs] t[::]t}' \mid \texttt{t} \in \texttt{Simple} \cup \texttt{[void]}, \texttt{t}' \in \texttt{ABS} \cup \texttt{Simple}\}$$

$$[\text{E13}]\ \frac{\texttt{F} = \texttt{[fp] p t}' \texttt{ I}' \quad \texttt{p} \neq \texttt{[funproc]} \quad \texttt{t}' \notin \texttt{Simple}}{\langle \texttt{[pcd] t I F Bs}, \texttt{Y}_\rho\rangle \rightarrow_{\texttt{Y}} (\texttt{[terr]}, \texttt{Y}_\rho)}$$

$$[\text{E13.1}]\ \frac{\texttt{F} = \texttt{[fp] p t}' \texttt{ I}' \quad \texttt{p} = \texttt{[funproc]} \quad \texttt{t}' \notin \texttt{ABS}}{\langle \texttt{[pcd] t I F Bs}, \texttt{Y}_\rho\rangle \rightarrow_{\texttt{Y}} (\texttt{[terr]}, \texttt{Y}_\rho)}$$

Sistema Y: DCL

$$\texttt{Dcl} ::= \ldots \mid \texttt{[pcd] Type Ide FPars BlockS} \mid \ldots$$

$$\texttt{ABS} \;=\; \{\texttt{[abs] t},\ \texttt{[abs] t[::]t}' \mid \texttt{t} \in \texttt{Simple} \cup \texttt{[void]}, \texttt{t}' \in \texttt{ABS} \cup \texttt{Simple}\}$$

$$\text{[E13]}\ \frac{\texttt{F} = \texttt{[fp] p t}' \texttt{ I}' \quad \texttt{p} \neq \texttt{[funproc]} \quad \texttt{t}' \notin \texttt{Simple}}{\langle \texttt{[pcd] t I F Bs}, \texttt{Y}_\rho \rangle \rightarrow_{\texttt{Y}} (\texttt{[terr]}, \texttt{Y}_\rho)}$$

$$\text{[E13.1]}\ \frac{\texttt{F} = \texttt{[fp] p t}' \texttt{ I}' \quad \texttt{p} = \texttt{[funproc]} \quad \texttt{t}' \notin \texttt{ABS}}{\langle \texttt{[pcd] t I F Bs}, \texttt{Y}_\rho \rangle \rightarrow_{\texttt{Y}} (\texttt{[terr]}, \texttt{Y}_\rho)}$$

Semantica: $\mathrm{SEM}_{\mathrm{DCL}}$

$$\texttt{Dcl} ::= \ldots \mid \texttt{[pcd] Type Ide FPars BlockS} \mid \ldots$$

$$[\text{D5}]\ \frac{\begin{array}{c}\texttt{t} \in \texttt{Simple} \cup \{\texttt{[void]}\} \quad \texttt{F} = \texttt{[fp] p t}' \texttt{ I}' \quad \texttt{p} \neq \texttt{[funproc]} \\ \texttt{t}' \in \texttt{Simple} \quad \Delta|_0(\texttt{I}) = \bot \quad \texttt{[abs] t[::]t}' = \texttt{t}_\texttt{n} \\ \texttt{Bs} = \texttt{[BlockS] d s} \quad \star\,\texttt{I}, \texttt{t}_\texttt{n}, \texttt{F}, \texttt{d}, \texttt{s}, \#\Delta\star = \texttt{v}_\texttt{n}\end{array}}{\langle \texttt{[pcd] t I F Bs}, (\Delta, \mu)\rangle \rightarrow (\texttt{[void]}, (\texttt{[I/(t}_\texttt{n}, \texttt{v}_\texttt{n}\texttt{)]} \otimes \Delta, \mu))}$$

$$[\text{D5.1}]\ \frac{\begin{array}{c}\texttt{t} \in \texttt{Simple} \cup \{\texttt{[void]}\} \quad \texttt{F} = \texttt{[fp] p t}' \texttt{ I}' \quad \texttt{p} = \texttt{[funproc]} \\ \texttt{t}' \in \texttt{ABS} \quad \Delta|_0(\texttt{I}) = \bot \quad \texttt{[abs] t[::]t}' = \texttt{t}_\texttt{n} \\ \texttt{Bs} = \texttt{[BlockS] d s} \quad \star\,\texttt{I}, \texttt{t}_\texttt{n}, \texttt{F}, \texttt{d}, \texttt{s}, \#\Delta\star = \texttt{v}_\texttt{n}\end{array}}{\langle \texttt{[pcd] t I F Bs}, (\Delta, \mu)\rangle \rightarrow (\texttt{[void]}, (\texttt{[I/(t}_\texttt{n}, \texttt{v}_\texttt{n}\texttt{)]} \otimes \Delta, \mu))}$$

Semantica: $\mathrm{SEM}_{\mathrm{EXP}}$

$$\texttt{Exp} ::= \ldots \mid \texttt{[apply] Ide APars} \mid \ldots$$

$$\text{[S15]}\ \frac{\begin{array}{c}\ldots \\ \mathtt{c_r} = \star\mathtt{ide}', \mathtt{t}, \mathtt{fps}, \mathtt{dcl}, \mathtt{sts}, \mathtt{k}\star \quad \ldots \quad \mathtt{ar} = \{\mathtt{ide}', \#\Delta - \mathtt{k} + 1, [], [], []\} \\ \ldots \\ \mathtt{ar}_2 = \{\mathtt{ide}', \mathtt{lc}, \mathtt{fr}, \mathtt{cnt}, \mathtt{v}\} \quad \ldots \quad \{\mathtt{ide}', \mathtt{lc}, \mathtt{fr}, \mathtt{sts}, \mathtt{v}\} = \mathtt{ar}_3 \\ \ldots \\ \mathtt{ar}'_3 = \{\mathtt{ide}', \mathtt{lc}, \mathtt{fr}, \mathtt{cnt}, \mathtt{v_r}\}\end{array}}{\langle \texttt{[apply] ide aps}, \sigma\rangle \to \lfloor \mathtt{t_r}, \mathtt{v_r}, (\Delta'_2, \mu_3)\rfloor}$$

Nota:
$\star\mathtt{ide}', \mathtt{t}, \mathtt{fps}, \mathtt{dcl}, \mathtt{sts}, \mathtt{k}\star$ chiusura per procedure e funzioni. Nel caso di funzioni e procedure passate come parametri ad altre funzioni/procedure:

- **`ide'`** = identificatore del parametro attuale;
- ... (il resto è come nella normale chiusura).

Semantica: $\mathrm{SEM}_{\mathrm{EXP}}$

$$\texttt{Exp} ::= \ldots \mid \texttt{[apply] Ide APars} \mid \ldots$$

$$[\text{S15}]\ \frac{\begin{array}{c}\ldots \\ \texttt{c}_\texttt{r} = \star\texttt{ide}', \texttt{t}, \texttt{fps}, \texttt{dcl}, \texttt{sts}, \texttt{k}\star \quad \ldots \quad \texttt{ar} = \{\texttt{ide}', \#\Delta - \texttt{k} + 1, [], [], []\} \\ \ldots \\ \texttt{ar}_2 = \{\texttt{ide}', \texttt{lc}, \texttt{fr}, \texttt{cnt}, \texttt{v}\} \quad \ldots \quad \{\texttt{ide}', \texttt{lc}, \texttt{fr}, \texttt{sts}, \texttt{v}\} = \texttt{ar}_3 \\ \ldots \\ \texttt{ar}_3' = \{\texttt{ide}', \texttt{lc}, \texttt{fr}, \texttt{cnt}, \texttt{v}_\texttt{r}\}\end{array}}{\langle [\texttt{apply}]\ \texttt{ide aps}, \sigma\rangle \rightarrow \lfloor \texttt{t}_\texttt{r}, \texttt{v}_\texttt{r}, (\Delta_2', \mu_3)\rfloor}$$

Nota:
$\star\texttt{ide}', \texttt{t}, \texttt{fps}, \texttt{dcl}, \texttt{sts}, \texttt{k}\star$ chiusura per procedure e funzioni. Nel caso di funzioni e procedure passate come parametri ad altre funzioni/procedure:

- `ide'` = identificatore del parametro attuale;
- $\ldots$ (il resto è come nella normale chiusura).

Semantica: SEM_{EXP}

$$\texttt{Exp} ::= \ldots \mid \texttt{[apply] Ide APars} \mid \ldots$$

$$[\text{S15}]\ \frac{\begin{array}{c}\ldots \\ \texttt{c}_\texttt{r} = \star\texttt{ide}', \texttt{t}, \texttt{fps}, \texttt{dcl}, \texttt{sts}, \texttt{k}\star \quad \ldots \quad \texttt{ar} = \{\texttt{ide}', \#\Delta - \texttt{k} + 1, [], [], []\} \\ \ldots \\ \texttt{ar}_2 = \{\texttt{ide}', \texttt{lc}, \texttt{fr}, \texttt{cnt}, \texttt{v}\} \quad \ldots \quad \{\texttt{ide}', \texttt{lc}, \texttt{fr}, \texttt{sts}, \texttt{v}\} = \texttt{ar}_3 \\ \ldots \\ \texttt{ar}_3' = \{\texttt{ide}', \texttt{lc}, \texttt{fr}, \texttt{cnt}, \texttt{v}_\texttt{r}\}\end{array}}{\langle [\texttt{apply}]\ \texttt{ide aps}, \sigma\rangle \rightarrow \lfloor \texttt{t}_\texttt{r}, \texttt{v}_\texttt{r}, (\Delta_2', \mu_3)\rfloor}$$

Nota:
$\star\texttt{ide}', \texttt{t}, \texttt{fps}, \texttt{dcl}, \texttt{sts}, \texttt{k}\star$ chiusura per procedure e funzioni. Nel caso di funzioni e procedure passate come parametri ad altre funzioni/procedure:

- **`ide'`** = identificatore del parametro attuale;
- ... (il resto è come nella normale chiusura).

Semantica: $\mathtt{SEM}_{\mathtt{STM}}$

$$\mathtt{Stm} ::= \ldots \mid \mathtt{[call]\ Ide\ APars} \mid \ldots$$

$$[\mathrm{S14}]\ \frac{\begin{array}{c}\ldots \\ \mathtt{v_r} = \star\mathtt{ide}', \mathtt{t}, \mathtt{fps}, \mathtt{dcl}, \mathtt{sts}, \mathtt{k} \star \quad \ldots \quad \mathtt{ar} = \{\mathtt{ide}', \#\Delta - \mathtt{k} + 1, [], [], []\} \\ \ldots \\ \mathtt{ar}_2 = \{\mathtt{ide}', \mathtt{lc}, \mathtt{fr}, \mathtt{cnt}, \mathtt{v}\} \quad \ldots \quad \{\mathtt{ide}', \mathtt{lc}, \mathtt{fr}, \mathtt{sts}, \mathtt{v}\} = \mathtt{ar}_3 \\ \ldots \end{array}}{\langle [\mathtt{call}]\ \mathtt{ide\ aps}, \sigma\rangle \to ([\mathtt{void}], (\Delta_2', \mu_3))}$$

Semantica: $\mathrm{SEM}_{\mathrm{STM}}$

$$\texttt{Stm} ::= \ldots \mid \texttt{[call] Ide APars} \mid \ldots$$

$$\text{[S14]}\ \frac{\begin{array}{c}\ldots \\ \texttt{v}_\texttt{r} = \star\texttt{ide}', \texttt{t}, \texttt{fps}, \texttt{dcl}, \texttt{sts}, \texttt{k} \star \quad \texttt{ar} = \{\texttt{ide}', \#\Delta - \texttt{k} + 1, [], [], []\} \\ \ldots \\ \texttt{ar}_2 = \{\texttt{ide}', \texttt{lc}, \texttt{fr}, \texttt{cnt}, \texttt{v}\} \quad \{\texttt{ide}', \texttt{lc}, \texttt{fr}, \texttt{sts}, \texttt{v}\} = \texttt{ar}_3 \\ \ldots \end{array}}{\langle [\texttt{call}]\ \texttt{ide aps}, \sigma \rangle \rightarrow ([\texttt{void}], (\Delta_2', \mu_3))}$$

Sistema Y: Trasmissione dei parametri

$$\texttt{FPars} ::= [\texttt{fp}]\ \texttt{PPF Type Ide} \mid \ldots$$
$$\texttt{PPF} ::= [\texttt{value}] \mid [\texttt{ref}] \mid [\texttt{funproc}]$$
$$\texttt{APars} ::= [\texttt{ap}]\ \texttt{Exp} \mid \ldots$$

$$[\text{Y35.1}]\ \frac{\begin{array}{c}\texttt{fps} = [\texttt{fp}]\ [\texttt{funproc}]\ \texttt{t}\ \texttt{I} \quad \texttt{Y}_\rho|_0(\texttt{I}) = \bot \\ \texttt{aps} = [\texttt{ap}]\ [\texttt{val}]\ \texttt{I}' \quad \texttt{Y}_\rho(\texttt{I}') = \texttt{t}' \\ \texttt{t} = \texttt{t}' \quad \texttt{t} \in \texttt{ABS}\end{array}}{\langle \texttt{fps} \lhd \texttt{aps}, \texttt{Y}_\rho \rangle \rightarrow_{\texttt{Y}} ([\texttt{void}], \texttt{Y}_\rho)}$$

Nota:
nella Y35.1, nei suoi errori e nelle regole S16.1 e S16.2 alcune "premesse" sono colorate in verde per segnalarne la ridondanza.

Sistema Y: Trasmissione dei parametri

$$\texttt{FPars} ::= \texttt{[fp] PPF Type Ide} \mid \ldots$$
$$\texttt{PPF} ::= \texttt{[value]} \mid \texttt{[ref]} \mid \texttt{[funproc]}$$
$$\texttt{APars} ::= \texttt{[ap] Exp} \mid \ldots$$

$$[\text{Y35.1}]\ \frac{\begin{array}{c}\texttt{fps} = \texttt{[fp] [funproc] t I} \quad \texttt{Y}_\rho|_0(\texttt{I}) = \bot \\ \texttt{aps} = \texttt{[ap] [val] I}' \quad \texttt{Y}_\rho(\texttt{I}') = \texttt{t}' \\ \texttt{t} = \texttt{t}' \quad \texttt{t} \in \texttt{ABS}\end{array}}{\langle \texttt{fps} \lhd \texttt{aps}, \texttt{Y}_\rho \rangle \rightarrow_{\texttt{Y}} (\texttt{[void]}, \texttt{Y}_\rho)}$$

Nota:
nella Y35.1, nei suoi errori e nelle regole S16.1 e S16.2 alcune "premesse" sono colorate in verde per segnalarne la ridondanza.

Sistema Y: Trasmissione dei parametri

$$\texttt{FPars} ::= [\texttt{fp}]\ \texttt{PPF Type Ide} \mid \ldots$$
$$\texttt{PPF} ::= [\texttt{value}] \mid [\texttt{ref}] \mid [\texttt{funproc}]$$
$$\texttt{APars} ::= [\texttt{ap}]\ \texttt{Exp} \mid \ldots$$

$$[\text{Y35.1}]\ \frac{\begin{array}{c}\texttt{fps} = [\texttt{fp}]\ [\texttt{funproc}]\ \texttt{t}\ \texttt{I} \quad \texttt{Y}_\rho|_0(\texttt{I}) = \bot \\ \texttt{aps} = [\texttt{ap}]\ [\texttt{val}]\ \texttt{I}' \quad \texttt{Y}_\rho(\texttt{I}') = \texttt{t}' \\ \texttt{t} = \texttt{t}' \quad \texttt{t} \in \texttt{ABS}\end{array}}{\langle \texttt{fps} \lhd \texttt{aps}, \texttt{Y}_\rho \rangle \rightarrow_{\texttt{Y}} ([\texttt{void}], \texttt{Y}_\rho)}$$

Nota:
nella Y35.1, nei suoi errori e nelle regole S16.1 e S16.2 alcune "premesse" sono colorate in verde per segnalarne la ridondanza.

Sistema Y: Trasmissione dei parametri

$$\texttt{FPars} ::= \texttt{[fp]}\ \texttt{PPF}\ \texttt{Type}\ \texttt{Ide} \mid \ldots$$
$$\texttt{PPF} ::= \texttt{[value]} \mid \texttt{[ref]} \mid \texttt{[funproc]}$$
$$\texttt{APars} ::= \texttt{[ap]}\ \texttt{Exp} \mid \ldots$$

$$\text{[E61.01]}\ \frac{\begin{array}{c}\texttt{fps} = \texttt{[fp] [funproc] t I} \\ \texttt{Y}_\rho|_0(\texttt{I}) \neq \bot\end{array}}{\langle \texttt{fps} \lhd \texttt{aps}, \texttt{Y}_\rho\rangle \to_{\texttt{Y}} (\texttt{[terr]}, \texttt{Y}_\rho)}$$

$$\text{[E61.02]}\ \frac{\begin{array}{c}\texttt{fps} = \texttt{[fp] [funproc] t I} \\ \texttt{aps} \neq \texttt{[ap] [val] I}'\end{array}}{\langle \texttt{fps} \lhd \texttt{aps}, \texttt{Y}_\rho\rangle \to_{\texttt{Y}} (\texttt{[terr]}, \texttt{Y}_\rho)}$$

$$\text{[E61.11]}\ \frac{\begin{array}{c}\texttt{fps} = \texttt{[fp] [funproc] t I} \\ \texttt{aps} = \texttt{[ap] [val] I}' \\ \texttt{Y}_\rho(\texttt{I}') = \texttt{t}' \quad \texttt{t} \neq \texttt{t}'\end{array}}{\langle \texttt{fps} \lhd \texttt{aps}, \texttt{Y}_\rho\rangle \to_{\texttt{Y}} (\texttt{[terr]}, \texttt{Y}_\rho)}$$

$$\text{[E61.21]}\ \frac{\begin{array}{c}\texttt{fps} = \texttt{[fp] [funproc] t I} \\ \texttt{t} \notin \texttt{ABS}\end{array}}{\langle \texttt{fps} \lhd \texttt{aps}, \texttt{Y}_\rho\rangle \to_{\texttt{Y}} (\texttt{[terr]}, \texttt{Y}_\rho)}$$

Sistema Y: Trasmissione dei parametri

$$\texttt{FPars} ::= \texttt{[fp]}\ \texttt{PPF}\ \texttt{Type}\ \texttt{Ide} \mid \ldots$$
$$\texttt{PPF} ::= \texttt{[value]} \mid \texttt{[ref]} \mid \texttt{[funproc]}$$
$$\texttt{APars} ::= \texttt{[ap]}\ \texttt{Exp} \mid \ldots$$

$$\text{[E61.01]}\ \frac{\begin{array}{c}\texttt{fps} = \texttt{[fp] [funproc] t I} \\ \texttt{Y}_\rho|_0(\texttt{I}) \neq \bot\end{array}}{\langle \texttt{fps} \lhd \texttt{aps}, \texttt{Y}_\rho\rangle \to_{\texttt{Y}} (\texttt{[terr]}, \texttt{Y}_\rho)}$$

$$\text{[E61.02]}\ \frac{\begin{array}{c}\texttt{fps} = \texttt{[fp] [funproc] t I} \\ \texttt{aps} \neq \texttt{[ap] [val] I}'\end{array}}{\langle \texttt{fps} \lhd \texttt{aps}, \texttt{Y}_\rho\rangle \to_{\texttt{Y}} (\texttt{[terr]}, \texttt{Y}_\rho)}$$

$$\text{[E61.11]}\ \frac{\begin{array}{c}\texttt{fps} = \texttt{[fp] [funproc] t I} \\ \texttt{aps} = \texttt{[ap] [val] I}' \\ \texttt{Y}_\rho(\texttt{I}') = \texttt{t}' \quad \texttt{t} \neq \texttt{t}'\end{array}}{\langle \texttt{fps} \lhd \texttt{aps}, \texttt{Y}_\rho\rangle \to_{\texttt{Y}} (\texttt{[terr]}, \texttt{Y}_\rho)}$$

$$\text{[E61.21]}\ \frac{\begin{array}{c}\texttt{fps} = \texttt{[fp] [funproc] t I} \\ \texttt{t} \notin \texttt{ABS}\end{array}}{\langle \texttt{fps} \lhd \texttt{aps}, \texttt{Y}_\rho\rangle \to_{\texttt{Y}} (\texttt{[terr]}, \texttt{Y}_\rho)}$$

Semantica: Trasmissione dei parametri TR1

$$\texttt{FPars} ::= [\texttt{fp}]\ \texttt{PPF}\ \texttt{Type}\ \texttt{Ide} \mid \ldots$$
$$\texttt{PPF} ::= [\texttt{value}] \mid [\texttt{ref}] \mid [\texttt{funproc}]$$
$$\texttt{APars} ::= [\texttt{ap}]\ \texttt{Exp} \mid \ldots$$

$$[\text{S16.1}]\ \frac{\begin{array}{c}\texttt{fps} = [\texttt{fp}]\ [\texttt{funproc}]\ \texttt{t}\ \texttt{I} \quad \texttt{aps} = [\texttt{ap}]\ [\texttt{val}]\ \texttt{I}_\texttt{n} \\ \Delta(\texttt{I}_\texttt{n}) = (\texttt{t}_\texttt{n}, \texttt{v}_\texttt{n}) \quad \texttt{t} = \texttt{t}_\texttt{n} \quad \texttt{t}_\texttt{n} \in \texttt{ABS} \\ \Delta_\texttt{c}|_0(\texttt{I}) = \bot \quad [\texttt{I}/(\texttt{t}_\texttt{n}, \texttt{v}_\texttt{n})] \otimes \Delta_\texttt{c} = \Delta_\texttt{c}^\texttt{F} \\ (\Delta_\texttt{c}^\texttt{F}, \mu) = \sigma_\texttt{n} \quad [] = \texttt{epi}_\texttt{n}\end{array}}{\langle \texttt{fps} \triangleleft \texttt{aps}, (\Delta, \Delta_\texttt{c}, \mu)\rangle \rightarrow_{\texttt{TR1}} (\sigma_\texttt{n}, \texttt{epi}_\texttt{n})}$$

Implementazione: modifiche all'interprete di Small21

Modifiche alla **Sintassi Astratta** e aggiunta del controllo di appartenenza al nuovo insieme di tipi: **ABS**.

```
73 ppf =
74     Value
75     | Ref
76     | FunProc
77     and
```

```
180 isABS = (function
181         | Abs(tye,[]) when (isSimpleorVoid tye) -> true
182         | Abs(tye,[tyef])
183             when (isSimpleorVoid tye) && ((isSimple tyef) || (isABS tyef)) -> true
184         | _ -> false)
185         and
```

Implementazione: modifiche all'interprete di Small21

Modifiche alla **Sintassi Astratta** e aggiunta del controllo di appartenenza al nuovo insieme di tipi: **ABS**.

```
73  ppf =
74      Value
75      | Ref
76      | FunProc
77      and
```

```
180 isABS = (function
181          | Abs(tye,[]) when (isSimpleorVoid tye) -> true
182          | Abs(tye,[tyef])
183              when (isSimpleorVoid tye) && ((isSimple tyef) || (isABS tyef)) -> true
184          | _ -> false)
185          and
```

Implementazione: modifiche all'interprete di Small21

Modifiche alla **Sintassi Concreta**.

```
toStringTyeFP = (function
        | Abs(ty1,[]) -> (toStringTye ty1) ^ "()"
        | Abs(ty1,[ty2]) -> (toStringTye ty1) ^ "(" ^ (toStringTyeFP ty2) ^ ")"
        | ty1 -> toStringTye ty1
        )
        and

toStringFPars = (function
        | EFP -> ""
        | FP(ppf,Abs(tyefp,tyseq),ide) ->
              (toStringTyeFP (Abs(tyefp,tyseq))) ^ " " ^ (toStringI ide)
        | FP(ppf,tye,ide) ->
              ((toStringPPF ppf) ^ " " ^ (toStringTye tye) ^ " " ^ (toStringI ide))
        )
        and
```

Implementazione: modifiche all'interprete di Small21

Modifiche alla **Sintassi Concreta**.

```
229 toStringTyeFP = (function
230         | Abs(ty1,[]) -> (toStringTye ty1) ^ "()"
231         | Abs(ty1,[ty2]) -> (toStringTye ty1) ^ "(" ^ (toStringTyeFP ty2) ^ ")"
232         | ty1 -> toStringTye ty1
233         )
234         and
235
236 toStringFPars = (function
237         | EFP -> ""
238         | FP(ppf,Abs(tyefp,tyseq),ide) ->
239                 (toStringTyeFP (Abs(tyefp,tyseq))) ^ " " ^ (toStringI ide)
240         | FP(ppf,tye,ide) ->
241                 ((toStringPPF ppf) ^ " " ^ (toStringTye tye) ^ " " ^ (toStringI ide))
242         )
243         and
```

Implementazione: modifiche all'interprete di Small21

SEM$_{\mathtt{DCL}}$: implementazione delle regole D5 e D5.1 e dei controlli di tipo Y5 e Y5.1.

```
877 | (_,FP(pf,t,ide),_)
878         when not(isSimple t) && not(isFunProc pf)
879         -> raise(TypeErrorI("E13: dclSem",ide))
880 | (_,FP(FunProc,t,ide),_)
881         when not(isABS t)
882         -> raise(TypeErrorI("E13.1: dclSem",ide))
883 | (_,FP(FunProc,t,_),BlockP(d,s))
884         -> (let tnew = Abs(ty,[t]) in
885             let clos = ClosT(ide,tnew,fpars,d,s,sizeS sk) in
886             let den = DAbs(tnew,clos) in
887             let sknew = bindS sk ide den in
888             (Void,(sknew,mu)))
```

Implementazione: modifiche all'interprete di Small21

SEM$_{\mathtt{DCL}}$: implementazione delle regole D5 e D5.1 e dei controlli di tipo Y5 e Y5.1.

```
877 | (_,FP(pf,t,ide),_)
878         when not(isSimple t) && not(isFunProc pf)
879         -> raise(TypeErrorI("E13: dclSem",ide))
880 | (_,FP(FunProc,t,ide),_)
881         when not(isABS t)
882         -> raise(TypeErrorI("E13.1: dclSem",ide))
883 | (_,FP(FunProc,t,_),BlockP(d,s))
884         -> (let tnew = Abs(ty,[t]) in
885             let clos = ClosT(ide,tnew,fpars,d,s,sizeS sk) in
886             let den = DAbs(tnew,clos) in
887             let sknew = bindS sk ide den in
888             (Void,(sknew,mu)))
```

Implementazione: modifiche all'interprete di Small21

SEM$_{EXP}$: implementazione della regola S15.

```
| Apply(ide,aps)
    ->(match getS sk ide with
        | DAbs(Abs(tr,aa),ClosT(i,_,fps,dcl,sts,k))
            when (isSimple tr)
        ->(let ar = mkAR5 (Name i) ((sizeS sk)-k+1)
                          (emptyEnv()) [] None in
        let skc = push sk ar in
        let (sgR,epiR) = tr1Fun fps aps sk skc mu in
        let (_,(ar2sk2,mu2)) = dclSem dcl sgR in
        let ar3sk2 = resetC ar2sk2 [UnL sts] in
        let (_,(sk4,mu4)) = nextCmd(ar3sk2,mu2) in
        let sgF = (pop sk4,mu4) in
        (tr,getR sk4,sgF))
```

Implementazione: modifiche all'interprete di Small21

SEM$_{EXP}$: implementazione della regola S15.

```
| Apply(ide,aps)
    ->(match getS sk ide with
        | DAbs(Abs(tr,aa),ClosT(i,_,fps,dcl,sts,k))
            when (isSimple tr)
        ->(let ar = mkAR5 (Name i) ((sizeS sk)-k+1)
                          (emptyEnv()) [] None in
        let skc = push sk ar in
        let (sgR,epiR) = tr1Fun fps aps sk skc mu in
        let (_,(ar2sk2,mu2)) = dclSem dcl sgR in
        let ar3sk2 = resetC ar2sk2 [UnL sts] in
        let (_,(sk4,mu4)) = nextCmd(ar3sk2,mu2) in
        let sgF = (pop sk4,mu4) in
        (tr,getR sk4,sgF))
```

Implementazione: modifiche all'interprete di Small21

SEM$_{\texttt{STM}}$: implementazione della regola S14.

```
1173 | Call (ide,aps)
1174     ->( match getS sk ide with
1175         | DAbs(Abs(Void,aa),ClosT(i,_,fps,dcl,sts,k))
1176             ->(let ar = mkAR5 (Name i) ((sizeS sk)-k+1)
1177                               (emptyEnv()) [] None in
1178         let skc = push sk ar in
1179         let (sgR,epiR) = tr1Fun fps aps sk skc mu in
1180         let (_,(ar2sk2,mu2)) = dclSem dcl sgR in
1181         let ar3sk2 = resetC ar2sk2 [UnL sts] in
1182         let (_,(sk4,mu4)) = nextCmd(ar3sk2,mu2) in
1183         let sgF = (pop sk4,mu4) in
1184         (Void,sgF))
```

Implementazione: modifiche all'interprete di Small21

SEM$_{\mathtt{STM}}$: implementazione della regola S14.

```
1173 | Call (ide,aps)
1174     ->( match getS sk ide with
1175         | DAbs(Abs(Void,aa),ClosT(i,_,fps,dcl,sts,k))
1176             ->(let ar = mkAR5 (Name i) ((sizeS sk)-k+1)
1177                               (emptyEnv()) [] None in
1178         let skc = push sk ar in
1179         let (sgR,epiR) = tr1Fun fps aps sk skc mu in
1180         let (_,(ar2sk2,mu2)) = dclSem dcl sgR in
1181         let ar3sk2 = resetC ar2sk2 [UnL sts] in
1182         let (_,(sk4,mu4)) = nextCmd(ar3sk2,mu2) in
1183         let sgF = (pop sk4,mu4) in
1184         (Void,sgF))
```

Implementazione: modifiche all'interprete di Small21

Semantica (Trasmissione dei parametri TR1):
implementazione della regola S16.1 e del controllo di tipo Y35.1.

```
1262 |(FP(FunProc,t,ide), AP exp) when (isABS t) && not(declared skc ide) ->
1263         (match exp with
1264             | Val(i) ->
1265                     (match getS sk i with
1266                         | DAbs(tin,clos)
1267                             when (ysame tin t) ->
1268                                 let vt = clos in
1269                                 let den = DAbs(t,vt) in
1270                                 let skcF = bindS skc ide den in
1271                                 let sgn = (skcF,mu) and epin = [] in
1272                                 (sgn,epin)
1273                         | DAbs(tin,_) when not(ysame tin t)
1274                             -> raise(TypeErrorT("E61.11: Trasmission: Types Mismatch"))
1275                         |_ -> raise(TypeErrorT("E61.21: Trasmission: Type not expected"))
1276                     )
1277             |_ -> raise(TypeErrorT("E61.02: Trasmission: Type not expected"))
1278         )
1279 |(FP(FunProc,_,ide), AP _) when declared skc ide ->
1280         raise(TypeErrorT("E61.21: Trasmission: Type not expected"))
1281 |(FP(FunProc,_,ide), AP _) ->
1282         raise(TypeErrorT("E61.01: Trasmission: Type not expected"))
```

Implementazione: modifiche all'interprete di Small21

Semantica (Trasmissione dei parametri TR1): implementazione della regola S16.1 e del controllo di tipo Y35.1.

```
1262 |(FP(FunProc,t,ide), AP exp) when (isABS t) && not(declared skc ide) ->
1263         (match exp with
1264             | Val(i) ->
1265                     (match getS sk i with
1266                         | DAbs(tin,clos)
1267                             when (ysame tin t) ->
1268                                     let vt = clos in
1269                                     let den = DAbs(t,vt) in
1270                                     let skcF = bindS skc ide den in
1271                                     let sgn = (skcF,mu) and epin = [] in
1272                                     (sgn,epin)
1273                         | DAbs(tin,_) when not(ysame tin t)
1274                             -> raise(TypeErrorT("E61.11: Trasmission: Types Mismatch"))
1275                         |_ -> raise(TypeErrorT("E61.21: Trasmission: Type not expected"))
1276                     )
1277             |_ -> raise(TypeErrorT("E61.02: Trasmission: Type not expected"))
1278         )
1279 |(FP(FunProc,_,ide), AP _) when declared skc ide ->
1280         raise(TypeErrorT("E61.21: Trasmission: Type not expected"))
1281 |(FP(FunProc,_,ide), AP _) ->
1282         raise(TypeErrorT("E61.01: Trasmission: Type not expected"))
```

Esecuzione degli esempi in Small21: example_1

```
let dcl1 = Var(Int, "x", N(1));;
let blkp1 =  BlockP(ED, Return(Plus(Val("x"), Val("y"))));;
let dcl2 = Pcd(Int, "f", FP(Value, Int, "y"), blkp1);;
let fpars1 = FP(FunProc, Abs(Int, [Int]), "h");;
let stm1 = Return(Plus(Apply("h", AP(N(3))), Val("x")));;
let blkp2 = BlockP(Var(Int, "x", N(2)), stm1);;
let dcl3 = Pcd(Int, "g", fpars1, blkp2);;
let dcl = SeqD(dcl1, SeqD(dcl2, dcl3));;
let dcl5 = Var(Int, "x", N(4));;
let dcl6 = Var(Int, "z", Apply("g", AP(Val("f"))));;
let stm2 = Upd(Val("x"), Plus(Val("x"),Val("z")));;
let blk = BlockS(SeqD(dcl5, dcl6), stm2);;
let cmd = UnL(blk);;
let b = Block(dcl, cmd);;
let example_1 = Prog("example_1", b);;

printProg example_1;;
progSem example_1;;
```

```
Program example_1{
    int x = 1;
    int f(value int y){
        return (x + y);
    }
    int g(int(int) h){
        int x = 2;
        return (h(3) + x);
    }
        {
        int x = 4;
        int z = g(f);
        x = (x + z);
        }
}
```

Primo esempio su cui è stata testata la nuova trasmissione:

1. la funzione `f` viene usata come parametro attuale per `g`;
2. viene costruita la chiusura di `f` con l'ambiente globale (di dichiarazione);
3. tale chiusura è associata al parametro formale `h`;
4. al momento della chiamata di `h` (`h(3)`) viene usata la chiusura per valutare le variabili;

Esecuzione degli esempi in Small21: example_1

```
let dcl1 = Var(Int, "x", N(1));;
let blkp1 =  BlockP(ED, Return(Plus(Val("x"), Val("y"))));;
let dcl2 = Pcd(Int, "f", FP(Value, Int, "y"), blkp1);;
let fpars1 = FP(FunProc, Abs(Int, [Int]), "h");;
let stm1 = Return(Plus(Apply("h", AP(N(3))), Val("x")));;
let blkp2 = BlockP(Var(Int, "x", N(2)), stm1);;
let dcl3 = Pcd(Int, "g", fpars1, blkp2);;
let dcl = SeqD(dcl1, SeqD(dcl2, dcl3));;
let dcl5 = Var(Int, "x", N(4));;
let dcl6 = Var(Int, "z", Apply("g", AP(Val("f"))));;
let stm2 = Upd(Val("x"), Plus(Val("x"),Val("z")));;
let blk = BlockS(SeqD(dcl5, dcl6), stm2);;
let cmd = UnL(blk);;
let b = Block(dcl, cmd);;
let example_1 = Prog("example_1", b);;

printProg example_1;;
progSem example_1;;
```

```
Program example_1{
    int x = 1;
    int f(value int y){
        return (x + y);
    }
    int g(int(int) h){
        int x = 2;
        return (h(3) + x);
    }
        {
        int x = 4;
        int z = g(f);
        x = (x + z);
        }
}
```

Primo esempio su cui è stata testata la nuova trasmissione:

1. la funzione `f` viene usata come parametro attuale per g;
2. viene costruita la chiusura di `f` con l'ambiente globale (di dichiarazione);
3. tale chiusura è associata al parametro formale `h`;
4. al momento della chiamata di `h` (`h(3)`) viene usata la chiusura per valutare le variabili;

Esecuzione degli esempi in Small21: traccia di example_1

```
Stack:
>{example_1,0,[g/([int::[int::int]],$g,[int::[int::int]],:fpar:,:cmd:,1$);
 f/([int::int],$f,[int::int],:fpar:,:cmd:,1$);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-1]


Stack:
>{f,3,[y/(Mint,L3)],:cmdNext:,4}
{g,2,[x/(Mint,L2);
 h/([int::int],$f,[int::int],:fpar:,:cmd:,1$)],:cmdNext:,[N]}
{[IB],1,[x/(Mint,L1)],:cmdNext:,[N]}
{example_1,0,[g/([int::[int::int]],$g,[int::[int::int]],:fpar:,:cmd:,1$);
 f/([int::int],$f,[int::int],:fpar:,:cmd:,1$);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-1,L1<-4,L2<-2,L3<-3]


Stack:
>{g,2,[x/(Mint,L2);
 h/([int::int],$f,[int::int],:fpar:,:cmd:,1$)],:cmdNext:,6}
{[IB],1,[x/(Mint,L1)],:cmdNext:,[N]}
{example_1,0,[g/([int::[int::int]],$g,[int::[int::int]],:fpar:,:cmd:,1$);
 f/([int::int],$f,[int::int],:fpar:,:cmd:,1$);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-1,L1<-4,L2<-2,L3<-3]
```

```
Stack:
>{[IB],1,[z/(Mint,L4);
 x/(Mint,L1)],:cmdNext:,[N]}
{example_1,0,[g/([int::[int::int]],$g,[int::[int::int]],:fpar:,:cmd:,1$);
 f/([int::int],$f,[int::int],:fpar:,:cmd:,1$);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-1,L1<-10,L2<-2,L3<-3,L4<-6]


Stack:
>{example_1,0,[g/([int::[int::int]],$g,[int::[int::int]],:fpar:,:cmd:,1$);
 f/([int::int],$f,[int::int],:fpar:,:cmd:,1$);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-1,L1<-10,L2<-2,L3<-3,L4<-6]


SUCCESSFUL_TERMINATION
- : unit/2 = ()
```

Esecuzione degli esempi in Small21: example_2

```
let dcl1 = Var(Int, "x", N(0));;
let fpars1 = FP(FunProc, Abs(Int, [Int]), "h");;
let fpars2 = FP(Value, Int, "y");;
let blk2 = BlockP(ED, Return(Plus(Plus(Val "z", Val "y"), N 1)));;
let dclb1 = Pcd(Int, "l", fpars2, blk2);;
let dclsb = SeqD(dclb1, Var(Int, "z", N 1));;
let apars1 = AP(Val "l");;
let stsb11 = Upd(Val "x", Apply("h", AP(Val "z")));;
let stsb1 = SeqS(stsb11, Upd(Val "z", N 0));;
let stssb =  SeqS(stsb1, IfT(Eq(Val "x", N 1), Call("g", apars1)));;
let blk1 = BlockP(dclsb, stssb);;
let dcl2 = Pcd(Void, "g", fpars1, blk1);;
let blk3 = BlockP(ED, Return(N 1));;
let dcl3 = Pcd(Int, "f", FP(Value, Int, "y"), blk3);;
let dcl = SeqD(dcl1, SeqD(dcl2, dcl3));;
let cmd = UnL(Call("g", AP(Val "f")));;
let b = Block(dcl, cmd);;
let example_2 = Prog("example_2", b);;

printProg example_2;;
progSem example_2;;
```

```
Program example_2{
    int x = 0;
    void g(int(int) h){
        int l(value int y){
            return ((z + y) + 1);
        }
        int z = 1;

        x = h(z);
        z = 0;
        if (x == 1) g(l);
    }
    int f(value int y){
        return 1;
    }
    g(f);
}
```

Secondo esempio di trasmissione di funzione:

1. nella funzione g viene dichiarata una funzione l;
2. l viene trasmessa come parametro a g stessa;
3. la prima chiamata di g ha per parametro formale f mentre la seconda ha l;
4. nella fase della chiamata di h con chiusura associata ad l si ha y legato a z come parametro attuale e z presente nella chiusura di l;

Esecuzione degli esempi in Small21: example_2

```
let dcl1 = Var(Int, "x", N(0));;
let fpars1 = FP(FunProc, Abs(Int, [Int]), "h");;
let fpars2 = FP(Value, Int, "y");;
let blk2 = BlockP(ED, Return(Plus(Plus(Val "z", Val "y"), N 1)));;
let dclb1 = Pcd(Int, "l", fpars2, blk2);;
let dclsb = SeqD(dclb1, Var(Int, "z", N 1));;
let apars1 = AP(Val "l");;
let stsb11 = Upd(Val "x", Apply("h", AP(Val "z")));;
let stsb1 = SeqS(stsb11, Upd(Val "z", N 0));;
let stssb =  SeqS(stsb1, IfT(Eq(Val "x", N 1), Call("g", apars1)));;
let blk1 = BlockP(dclsb, stssb);;
let dcl2 = Pcd(Void, "g", fpars1, blk1);;
let blk3 = BlockP(ED, Return(N 1));;
let dcl3 = Pcd(Int, "f", FP(Value, Int, "y"), blk3);;
let dcl = SeqD(dcl1, SeqD(dcl2, dcl3));;
let cmd = UnL(Call("g", AP(Val "f")));;
let b = Block(dcl, cmd);;
let example_2 = Prog("example_2", b);;

printProg example_2;;
progSem example_2;;
```

```
Program example_2{
    int x = 0;
    void g(int(int) h){
        int l(value int y){
            return ((z + y) + 1);
        }
        int z = 1;

        x = h(z);
        z = 0;
        if (x == 1) g(l);
    }
    int f(value int y){
        return 1;
    }
    g(f);
}
```

Secondo esempio di trasmissione di funzione:

1. nella funzione g viene dichiarata una funzione `l`;
2. `l` viene trasmessa come parametro a g stessa;
3. la prima chiamata di g ha per parametro formale `f` mentre la seconda ha `l`;
4. nella fase della chiamata di `h` con chiusura associata ad `l` si ha `y` legato a `z` come parametro attuale e `z` presente nella chiusura di `l`;

Esecuzione degli esempi in Small21: traccia di example_2

```
Stack:
>{example_2,0,[f/([int::int],$f,[int::int],:fpar:,:cmd:,1$);
 g/([void::[int::int]],$g,[void::[int::int]],:fpar:,:cmd:,1$);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-0]


Stack:
>{f,2,[y/(Mint,L2)],:cmdNext:,1}
{g,1,[z/(Mint,L1);
 l/([int::int],$l,[int::int],:fpar:,:cmd:,2$);
 h/([int::int],$f,[int::int],:fpar:,:cmd:,1$)],:cmdNext:,[N]}
{example_2,0,[f/([int::int],$f,[int::int],:fpar:,:cmd:,1$);
 g/([void::[int::int]],$g,[void::[int::int]],:fpar:,:cmd:,1$);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-0,L1<-1,L2<-1]


Stack:
>{g,1,[z/(Mint,L1);
 l/([int::int],$l,[int::int],:fpar:,:cmd:,2$);
 h/([int::int],$f,[int::int],:fpar:,:cmd:,1$)],:cmdNext:,[N]}
{example_2,0,[f/([int::int],$f,[int::int],:fpar:,:cmd:,1$);
 g/([void::[int::int]],$g,[void::[int::int]],:fpar:,:cmd:,1$);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-1,L1<-1,L2<-1]


Stack:
>{g,1,[z/(Mint,L1);
 l/([int::int],$l,[int::int],:fpar:,:cmd:,2$);
 h/([int::int],$f,[int::int],:fpar:,:cmd:,1$)],:cmdNext:,[N]}
{example_2,0,[f/([int::int],$f,[int::int],:fpar:,:cmd:,1$);
 g/([void::[int::int]],$g,[void::[int::int]],:fpar:,:cmd:,1$);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-1,L1<-0,L2<-1]


Stack:
>{l,2,[y/(Mint,L4)],:cmdNext:,2}
{g,2,[z/(Mint,L3);
 l/([int::int],$l,[int::int],:fpar:,:cmd:,3$);
 h/([int::int],$l,[int::int],:fpar:,:cmd:,2$)],:cmdNext:,[N]}
{g,1,[z/(Mint,L1);
 l/([int::int],$l,[int::int],:fpar:,:cmd:,2$);
 h/([int::int],$f,[int::int],:fpar:,:cmd:,1$)],:cmdNext:,[N]}
{example_2,0,[f/([int::int],$f,[int::int],:fpar:,:cmd:,1$);
 g/([void::[int::int]],$g,[void::[int::int]],:fpar:,:cmd:,1$);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-2,L1<-0,L2<-1,L3<-1,L4<-1]


Stack:
>{g,2,[z/(Mint,L3);
 l/([int::int],$l,[int::int],:fpar:,:cmd:,3$);
 h/([int::int],$l,[int::int],:fpar:,:cmd:,2$)],:cmdNext:,[N]}
{g,1,[z/(Mint,L1);
 l/([int::int],$l,[int::int],:fpar:,:cmd:,2$);
 h/([int::int],$f,[int::int],:fpar:,:cmd:,1$)],:cmdNext:,[N]}
{example_2,0,[f/([int::int],$f,[int::int],:fpar:,:cmd:,1$);
 g/([void::[int::int]],$g,[void::[int::int]],:fpar:,:cmd:,1$);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-2,L1<-0,L2<-1,L3<-0,L4<-1]


Stack:
>{g,2,[z/(Mint,L3);
 l/([int::int],$l,[int::int],:fpar:,:cmd:,3$);
 h/([int::int],$l,[int::int],:fpar:,:cmd:,2$)],:cmdNext:,[N]}
{g,1,[z/(Mint,L1);
 l/([int::int],$l,[int::int],:fpar:,:cmd:,2$);
 h/([int::int],$f,[int::int],:fpar:,:cmd:,1$)],:cmdNext:,[N]}
{example_2,0,[f/([int::int],$f,[int::int],:fpar:,:cmd:,1$);
 g/([void::[int::int]],$g,[void::[int::int]],:fpar:,:cmd:,1$);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-2,L1<-0,L2<-1,L3<-0,L4<-1]


Stack:
>{g,1,[z/(Mint,L1);
 l/([int::int],$l,[int::int],:fpar:,:cmd:,2$);
 h/([int::int],$f,[int::int],:fpar:,:cmd:,1$)],:cmdNext:,[N]}
{example_2,0,[f/([int::int],$f,[int::int],:fpar:,:cmd:,1$);
 g/([void::[int::int]],$g,[void::[int::int]],:fpar:,:cmd:,1$);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-2,L1<-0,L2<-1,L3<-0,L4<-1]


Stack:
>{example_2,0,[f/([int::int],$f,[int::int],:fpar:,:cmd:,1$);
 g/([void::[int::int]],$g,[void::[int::int]],:fpar:,:cmd:,1$);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-2,L1<-0,L2<-1,L3<-0,L4<-1]


SUCCESSFUL_TERMINATION
- : unit/2 = ()
```

Analisi degli errori di tipo: example_err_1

```
let dcl1 = Var(Int, "x", N(1));;
let dcl2 = Var(Int, "f", N 3);;
let fpars1 = FP(FunProc, Int, "h");;
let stm1 = Return(Plus(Val "h", Val("x")));;
let blkp2 = BlockP(Var(Int, "x", N(2)), stm1);;
let dcl3 = Pcd(Int, "g", fpars1, blkp2);;
let dcl = SeqD(dcl1, SeqD(dcl2, dcl3));;
let dcl5 = Var(Int, "x", N(4));;
let dcl6 = Var(Int, "z", Apply("g", AP(Val("f"))));;
let stm2 = Upd(Val("x"), Plus(Val("x"),Val("z")));;
let blk = BlockS(SeqD(dcl5, dcl6), stm2);;
let cmd = UnL(blk);;
let b = Block(dcl, cmd);;
let example_err_1 = Prog("example_err_1", b);;

printProg example_err_1;;
progSem example_err_1;;
```

```
Program example_err_1{
    int x = 1;
    int f = 3;
    int g(funproc int h){
        int x = 2;
        return (h + x);
    }
        {
        int x = 4;
        int z = g(f);
        x = (x + z);
        }
}
```

Il codice presenta un errore di tipo nella dichiarazione della funzione g:

1. il parametro formale ha tipologia di trasmissione `[funproc]` ma non è una funzione o procedura;
2. ci aspettiamo un errore nel `Sistema Y: DCL: E13.1`;

```
Exception: TypeErrorI ("E13.1: dclSem", "h").
```

Analisi degli errori di tipo: example_err_1

```
1  let dcl1 = Var(Int, "x", N(1));;
2  let dcl2 = Var(Int, "f", N 3);;
3  let fpars1 = FP(FunProc, Int, "h");;
4  let stm1 = Return(Plus(Val "h", Val("x")));;
5  let blkp2 = BlockP(Var(Int, "x", N(2)), stm1);;
6  let dcl3 = Pcd(Int, "g", fpars1, blkp2);;
7  let dcl = SeqD(dcl1, SeqD(dcl2, dcl3));;
8  let dcl5 = Var(Int, "x", N(4));;
9  let dcl6 = Var(Int, "z", Apply("g", AP(Val("f"))));;
10 let stm2 = Upd(Val("x"), Plus(Val("x"),Val("z")));;
11 let blk = BlockS(SeqD(dcl5, dcl6), stm2);;
12 let cmd = UnL(blk);;
13 let b = Block(dcl, cmd);;
14 let example_err_1 = Prog("example_err_1", b);;
15
16 printProg example_err_1;;
17 progSem example_err_1;;
```

```
Program example_err_1{
    int x = 1;
    int f = 3;
    int g(funproc int h){
        int x = 2;
        return (h + x);
    }
        {
        int x = 4;
        int z = g(f);
        x = (x + z);
        }
}
```

Il codice presenta un errore di tipo nella dichiarazione della funzione g:

1. il parametro formale ha tipologia di trasmissione `[funproc]` ma non è una funzione o procedura;
2. ci aspettiamo un errore nel `Sistema Y: DCL`: E13.1;

```
Exception: TypeErrorI ("E13.1: dclSem", "h").
```

Analisi degli errori di tipo: example_err_2

```
1  let dcl1 = Var(Int, "x", N(1));;
2  let s01 = Return(Plus(Val "x", N 1));;
3  let blkp1 =  BlockP(ED, s01);;
4  let t1 = Abs(Void,[Arr(Int, 1)]);;
5  let fpars0 = FP(FunProc, t1, "l");;
6  let dcl2 = Pcd(Int, "f", fpars0, blkp1);;
7  let fpars1 = FP(FunProc, Abs(Int, [t1]), "h");;
8  let stm1 = Return(Plus(Val("x"), N 1));;
9  let blkp2 = BlockP(ED, stm1);;
10 let dcl3 = Pcd(Int, "g", fpars1, blkp2);;
11 let dcl = SeqD(dcl1, SeqD(dcl3, dcl2));;
12 let dcl5 = Var(Int, "x", N(4));;
13 let dcl6 = Var(Int, "z", Apply("g", AP(Val("f"))));;
14 let stm2 = Upd(Val("x"), Plus(Val("x"),Val("z")));;
15 let blk = BlockS(SeqD(dcl5, dcl6), stm2);;
16 let cmd = UnL(blk);;
17 let b = Block(dcl, cmd);;
18 let example_err_2 = Prog("example_err_2", b);;
19
20 printProg example_err_2;;
21 progSem example_err_2;;
```

```
Program example_err_2{
    int x = 1;
    int g(int(void(int[1])) h){
        return (x + 1);
    }
    int f(void(int[1]) l){
        return (x + 1);
    }
        {
        int x = 4;
        int z = g(f);
        x = (x + z);
        }
}
```

Il codice presenta un errore di tipo nella dichiarazione della funzione g:

1. il parametro formale ha tipo funzione con input di tipo procedura con input di tipo array;
2. ci aspettiamo un errore nel `Sistema Y: DCL: E13.1`:
   rispetto all'esempio precedente abbiamo un parametro formale di tipo funzione $\mathtt{t} = \mathtt{Abs}(\mathtt{Int}, [\mathtt{Abs}(\mathtt{Void}, [\mathtt{Arr}(\mathtt{Int}, 1)])])$ ma tale che $\mathtt{t} \notin \mathtt{ABS}$.

```
Exception: TypeErrorI ("E13.1: dclSem", "h").
```

Analisi degli errori di tipo: example_err_2

```
let dcl1 = Var(Int, "x", N(1));;
let s01 = Return(Plus(Val "x", N 1));;
let blkp1 =  BlockP(ED, s01);;
let t1 = Abs(Void,[Arr(Int, 1)]);;
let fpars0 = FP(FunProc, t1, "l");;
let dcl2 = Pcd(Int, "f", fpars0, blkp1);;
let fpars1 = FP(FunProc, Abs(Int, [t1]), "h");;
let stm1 = Return(Plus(Val("x"), N 1));;
let blkp2 = BlockP(ED, stm1);;
let dcl3 = Pcd(Int, "g", fpars1, blkp2);;
let dcl = SeqD(dcl1, SeqD(dcl3, dcl2));;
let dcl5 = Var(Int, "x", N(4));;
let dcl6 = Var(Int, "z", Apply("g", AP(Val("f"))));;
let stm2 = Upd(Val("x"), Plus(Val("x"),Val("z")));;
let blk = BlockS(SeqD(dcl5, dcl6), stm2);;
let cmd = UnL(blk);;
let b = Block(dcl, cmd);;
let example_err_2 = Prog("example_err_2", b);;

printProg example_err_2;;
progSem example_err_2;;
```

```
Program example_err_2{
    int x = 1;
    int g(int(void(int[1])) h){
        return (x + 1);
    }
    int f(void(int[1]) l){
        return (x + 1);
    }
        {
        int x = 4;
        int z = g(f);
        x = (x + z);
        }
}
```

Il codice presenta un errore di tipo nella dichiarazione della funzione g:

1. il parametro formale ha tipo funzione con input di tipo procedura con input di tipo array;
2. ci aspettiamo un errore nel `Sistema Y: DCL: E13.1`:
   rispetto all'esempio precedente abbiamo un parametro formale di tipo funzione $\texttt{t} = \texttt{Abs}(\texttt{Int}, [\texttt{Abs}(\texttt{Void}, [\texttt{Arr}(\texttt{Int}, 1)])])$ ma tale che $\texttt{t} \notin \texttt{ABS}$.

```
Exception: TypeErrorI ("E13.1: dclSem", "h").
```

Analisi degli errori di tipo: example_err_3

```
let dcl1 = Var(Int, "x", N(1));;
let stmb1 = Upd(Val "x", Plus(Val("x"), Val("y")));;
let blkp1 =  BlockP(ED, stmb1);;
let dcl2 = Pcd(Void, "f", FP(Value, Int, "y"), blkp1);;
let fpars1 = FP(FunProc, Abs(Int, [Int]), "h");;
let exp1 = Apply("h", AP(N(3)));;
let stm1 = Return(Plus(exp1, Val("x")));;
let blkp2 = BlockP(Var(Int, "x", N(2)), stm1);;
let dcl3 = Pcd(Int, "g", fpars1, blkp2);;
let dcl = SeqD(dcl1, SeqD(dcl2, dcl3));;
let dcl5 = Var(Int, "x", N(4));;
let dcl6 = Var(Int, "z", Apply("g", AP(Val("f"))));;
let stm2 = Upd(Val("x"), Plus(Val("x"),Val("z")));;
let blk = BlockS(SeqD(dcl5, dcl6), stm2);;
let cmd = UnL(blk);;
let b = Block(dcl, cmd);;
let example_err_3 = Prog("example_err_3", b);;

printProg example_err_3;;
progSem example_err_3;;
```

```
Program example_err_3{
    int x = 1;
    void f(value int y){
        x = (x + y);
    }
    int g(int(int) h){
        int x = 2;
        return (h(3) + x);
    }
        {
        int x = 4;
        int z = g(f);
        x = (x + z);
        }
}
```

Il codice presenta un errore di tipo nell'applicazione della funzione g:

1. il tipo del parametro formale non corrisponde al tipo del parametro attuale;
2. ci aspettiamo un errore nel `Sistema Y: Trasmissione dei parametri`: E61.11;

Analisi degli errori di tipo: example_err_3

```
let dcl1 = Var(Int, "x", N(1));;
let stmb1 = Upd(Val "x", Plus(Val("x"), Val("y")));;
let blkp1 =  BlockP(ED, stmb1);;
let dcl2 = Pcd(Void, "f", FP(Value, Int, "y"), blkp1);;
let fpars1 = FP(FunProc, Abs(Int, [Int]), "h");;
let exp1 = Apply("h", AP(N(3)));;
let stm1 = Return(Plus(exp1, Val("x")));;
let blkp2 = BlockP(Var(Int, "x", N(2)), stm1);;
let dcl3 = Pcd(Int, "g", fpars1, blkp2);;
let dcl = SeqD(dcl1, SeqD(dcl2, dcl3));;
let dcl5 = Var(Int, "x", N(4));;
let dcl6 = Var(Int, "z", Apply("g", AP(Val("f"))));;
let stm2 = Upd(Val("x"), Plus(Val("x"),Val("z")));;
let blk = BlockS(SeqD(dcl5, dcl6), stm2);;
let cmd = UnL(blk);;
let b = Block(dcl, cmd);;
let example_err_3 = Prog("example_err_3", b);;

printProg example_err_3;;
progSem example_err_3;;
```

```
Program example_err_3{
    int x = 1;
    void f(value int y){
        x = (x + y);
    }
    int g(int(int) h){
        int x = 2;
        return (h(3) + x);
    }
        {
        int x = 4;
        int z = g(f);
        x = (x + z);
        }
}
```

Il codice presenta un errore di tipo nell'applicazione della funzione g:

1. il tipo del parametro formale non corrisponde al tipo del parametro attuale;
2. ci aspettiamo un errore nel `Sistema Y: Trasmissione dei parametri`: E61.11;

Analisi degli errori di tipo: `example_err_3`

Il codice presenta un errore di tipo nell'applicazione della funzione g:

1. il tipo del parametro formale non corrisponde al tipo del parametro attuale;
2. ci aspettiamo un errore nel `Sistema Y: Trasmissione dei parametri`: E61.11;

```
Stack:
>{example_err_3,0,[g/([int::[int::int]],$g,[int::[int::int]],:fpar:,:cmd:,1$);
 f/([void::int],$f,[void::int],:fpar:,:cmd:,1$);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-1]

Exception: TypeErrorT "E61.11: Trasmission: Types Mismatch".
```

Analisi degli errori di tipo: example_err_4

```
let dcl1 = Var(Int, "x", N(1));;
let blkp1 =  BlockP(ED, Return(Plus(Val("x"), Val("y"))));;
let dcl2 = Pcd(Int, "f", FP(Value, Int, "y"), blkp1);;
let fpars1 = FP(FunProc, Abs(Int, [Int]), "h");;
let stm1 = Return(Plus(Apply("h", AP(N(3))), Val("x")));;
let blkp2 = BlockP(Var(Int, "x", N(2)), stm1);;
let dcl3 = Pcd(Int, "g", fpars1, blkp2);;
let dcl = SeqD(dcl1, SeqD(dcl2, dcl3));;
let dcl5 = Var(Int, "x", N(4));;
let apars1 = AP(Plus(Val("x"), N 3));;
let dcl6 = Var(Int, "z", Apply("g", apars1));;
let stm2 = Upd(Val("x"), Plus(Val("x"),Val("z")));;
let blk = BlockS(SeqD(dcl5, dcl6), stm2);;
let cmd = UnL(blk);;
let b = Block(dcl, cmd);;
let example_err_4 = Prog("example_err_4", b);;

printProg example_err_4;;
progSem example_err_4;;
```

```
Program example_err_4{
    int x = 1;
    int f(value int y){
        return (x + y);
    }
    int g(int(int) h){
        int x = 2;
        return (h(3) + x);
    }
        {
        int x = 4;
        int z = g((x + 3));
        x = (x + z);
        }
}
```

Il codice presenta un errore di tipo nell'applicazione della funzione g:

1. il parametro attuale è un'espressione di forma diversa da `Val "ide"`;
2. ci aspettiamo un errore nel `Sistema Y: Trasmissione dei parametri`: E61.02;

Analisi degli errori di tipo: example_err_4

```
let dcl1 = Var(Int, "x", N(1));;
let blkp1 =  BlockP(ED, Return(Plus(Val("x"), Val("y"))));;
let dcl2 = Pcd(Int, "f", FP(Value, Int, "y"), blkp1);;
let fpars1 = FP(FunProc, Abs(Int, [Int]), "h");;
let stm1 = Return(Plus(Apply("h", AP(N(3))), Val("x")));;
let blkp2 = BlockP(Var(Int, "x", N(2)), stm1);;
let dcl3 = Pcd(Int, "g", fpars1, blkp2);;
let dcl = SeqD(dcl1, SeqD(dcl2, dcl3));;
let dcl5 = Var(Int, "x", N(4));;
let apars1 = AP(Plus(Val("x"), N 3));;
let dcl6 = Var(Int, "z", Apply("g", apars1));;
let stm2 = Upd(Val("x"), Plus(Val("x"),Val("z")));;
let blk = BlockS(SeqD(dcl5, dcl6), stm2);;
let cmd = UnL(blk);;
let b = Block(dcl, cmd);;
let example_err_4 = Prog("example_err_4", b);;

printProg example_err_4;;
progSem example_err_4;;
```

```
Program example_err_4{
    int x = 1;
    int f(value int y){
        return (x + y);
    }
    int g(int(int) h){
        int x = 2;
        return (h(3) + x);
    }
        {
        int x = 4;
        int z = g((x + 3));
        x = (x + z);
        }
}
```

Il codice presenta un errore di tipo nell'applicazione della funzione g:

1. il parametro attuale è un'espressione di forma diversa da `Val "ide"`;
2. ci aspettiamo un errore nel `Sistema Y: Trasmissione dei parametri`: E61.02;

Analisi degli errori di tipo: `example_err_4`

Il codice presenta un errore di tipo nell'applicazione della funzione g:

1. il parametro attuale è un'espressione di forma diversa da `Val "ide"`;
2. ci aspettiamo un errore nel `Sistema Y: Trasmissione dei parametri`: E61.02;

```
Stack:
>{example_err_4,0,[g/([int::[int::int]],$g,[int::[int::int]],:fpar:,:cmd:,1$);
 f/([int::int],$f,[int::int],:fpar:,:cmd:,1$);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-1]

Exception: TypeErrorT "E61.02: Trasmission: Type not expected".
```