



UNIVERSITÀ DI PISA

Linguaggi di Programmazione con Laboratorio

Seminario di fine corso:
Trasmissione per costante in Small21

Ivan Bioli

Università di Pisa, Dipartimento di Matematica

15 Giugno 2021



$$\text{fps} = [\text{fp}] \text{ [constant] Type Ide}$$

La trasmissione per costante è caratterizzata da:

- i parametri attuali sono valutati nel chiamante come espressioni che devono calcolare valori non modificabili
- i parametri formali sono legati a tali valori costanti
- la trasmissione è one-way

$$\text{fps} = [\text{fp}] \text{ [constant] Type Ide}$$

La trasmissione per costante è caratterizzata da:

- i parametri attuali sono valutati nel chiamante come espressioni che devono calcolare valori non modificabili
- i parametri formali sono legati a tali valori costanti
- la trasmissione è one-way

$$\text{fps} = [\text{fp}] \text{ [constant] Type Ide}$$

La trasmissione per costante è caratterizzata da:

- i parametri attuali sono valutati nel chiamante come espressioni che devono calcolare valori non modificabili
- i parametri formali sono legati a tali valori costanti
- la trasmissione è one-way

$$\text{fps} = [\text{fp}] \text{ [constant] Type Ide}$$

La trasmissione per costante è caratterizzata da:

- i parametri attuali sono valutati nel chiamante come espressioni che devono calcolare valori non modificabili
- i parametri formali sono legati a tali valori costanti
- la trasmissione è one-way

Confronto con gli altri tipi di trasmissione

| | Valore | Reference | Costante |
|---|--|--------------------------|------------------------------|
| Parametri attuali valutati come espressioni che calcolano: | valori memorizzabili | valori modificabili | valori non modificabili |
| Parametri formali legati a: | valori modificabili inizializzati al valore del corrispondente parametro attuale | tali valori modificabili | tali valori non modificabili |
| Comunicazione con il chiamante | one-way | a memoria condizionale | one-way |

`fps = [fp] [constant] t ide`

Diamo una formalizzazione che permetta di applicare la trasmissione per costante a tipi:

- $t \in \text{Simple} = \{[\text{bool}], [\text{int}]\}$.
- $t = [\text{arr}] t' N$, $t' \in \text{Simple}$, $N > 0$.

`fps = [fp] [constant] t ide`

Diamo una formalizzazione che permetta di applicare la trasmissione per costante a tipi:

- $t \in \text{Simple} = \{[\text{bool}], [\text{int}]\}$. Esempio:

```
int double(int constant x){
    return (x+x);
}
```

- $t = [\text{arr}] t' N$, $t' \in \text{Simple}$, $N > 0$.

`fps = [fp] [constant] t ide`

Diamo una formalizzazione che permetta di applicare la trasmissione per costante a tipi:

- $t \in \text{Simple} = \{[\text{bool}], [\text{int}]\}$. Esempio:

```
int double(int constant x){
    return (x+x);
}
```

- $t = [\text{arr}] t' N$, $t' \in \text{Simple}$, $N > 0$. Esempio:

```
void swap(constant int[2] v){
    int temp;
    temp = v[0];
    v[0] = v[1];
    v[1] = temp;
}
```


I parametri passati per costante non possono essere modificati nel corpo della procedura né direttamente né indirettamente.

Trasmissione per costante di Simple

I parametri passati per costante non possono essere modificati nel corpo della procedura né direttamente né indirettamente. Non sono legali:

```
int double(int constant x){
    x = x+x;
    return x;
}
```

Figure: Modifica diretta del parametro formale passato per costante

```
int f(int constant x){
    int a;
    a = g(x);
    ...
}
```

Figure: Modifica indiretta del parametro formale passato per costante

Trasmissione per costante di Simple

I parametri passati per costante non possono essere modificati nel corpo della procedura né direttamente né indirettamente. Non sono legali:

```
int double(int constant x){
    x = x+x;
    return x;
}
```

Figure: Modifica diretta del parametro formale passato per costante

```
int doublev2(int constant x){
    int y;
    y = g(x);
    return (y+y);
}
```

```
int g(int reference z){
    z = z+1;
    return z;
}
```

Figure: Modifica indiretta del parametro formale passato per costante

In Small21:

- gli array sono valori non modificabili, ma a componenti modificabili (come in C)
- gli identificatori di array hanno come binding nell'ambiente il valore array denotato, vale a dire la coppia $([\text{arr}]([\text{mut}]t)N, \text{loc})$ (come gli identificatori di costante)

In Small21:

- gli array sono valori non modificabili, ma a componenti modificabili (come in C)
- gli identificatori di array hanno come binding nell'ambiente il valore array denotato, vale a dire la coppia $([\text{arr}]([\text{mut}] \text{t}) N, \text{loc})$ (come gli identificatori di costante)

In Small21:

- gli array sono valori non modificabili, ma a componenti modificabili (come in C)
- gli identificatori di array hanno come binding nell'ambiente il valore array denotato, vale a dire la coppia $([\text{arr}]([\text{mut}] \text{t}) N, \text{loc})$ (come gli identificatori di costante)

In Small21:

- gli array sono valori non modificabili, ma a componenti modificabili (come in C)
- gli identificatori di array hanno come binding nell'ambiente il valore array denotato, vale a dire la coppia $([\text{arr}]([\text{mut}] \tau) N, \text{loc})$ (come gli identificatori di costante)

Gli array sono quindi trasmissibili per costante, ma con un comportamento assimilabile a quello della trasmissione per reference.

Trasmissione per costante di array

```
Program arraySwap{
  int[2] a;
  void swap(constant int[2] v){
    int temp;
    temp = v[0];
    v[0] = v[1];
    v[1] = temp;
  }
  a[0] = 1;
  a[1] = 2;
  swap(a);
}
```

| | |
|---------|----------------|
| -- | AR0 |
| CS | ... |
| CD | ... |
| swap | [Lswap, AR0] |
| a | (MInt [2], L0) |
| | |
| swap(a) | |
| -- | ARswap |
| CS | AR0 |
| CD | AR0 |
| | |
| temp | (Mint, L2) |
| v | (MInt [2], L0) |

| Lswap | swap-code |
|---------|-----------|
| L0 | 1 |
| L1 | 2 |
| L2 | Undef |
| | |
| MEMORIA | |

STATO FINALE ATTESO:
a[0] = 2, a[1] = 1

Figure: Rappresentazione dello stato durante l'invocazione di swap(a)

Comportamento finale simile a quello del "passaggio per valore di un array" in C, ma con delle differenze:

- in C il valore di una variabile o di un'espressione array è l'indirizzo dell'elemento zero del vettore stesso: viene trasmesso un puntatore
- flessibilità: il tipo array di Small21 include anche la taglia N

Comportamento finale simile a quello del "passaggio per valore di un array" in C, ma con delle differenze:

- in C il valore di una variabile o di un'espressione array è l'indirizzo dell'elemento zero del vettore stesso: viene trasmesso un puntatore
- flessibilità: il tipo array di Small21 include anche la taglia N

Comportamento finale simile a quello del "passaggio per valore di un array" in C, ma con delle differenze:

- in C il valore di una variabile o di un'espressione array è l'indirizzo dell'elemento zero del vettore stesso: viene trasmesso un puntatore
- flessibilità: il tipo array di Small21 include anche la taglia N

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

```
...  
PPF  $\rightarrow \epsilon \mid \text{ref} \mid \text{constant}$   
...
```

Vincoli contestuali

...

- Parametri solo di tipo `Simple` per trasmissione per valore e trasmissione per reference, anche `Simple [Num]` (cioè array) per la trasmissione per costante.

...

- **Sintassi Astratta**

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

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

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

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

```
...  
ppf =  
  Value  
  | Ref  
  | Constant  
...
```

- 1 Regole per la Dcl di procedure
- 2 Regole per Exp per $[val] I$ con I identificatore di array
- 3 Regole per la trasmissione di paramentri

- 1 **Regole per la Dcl di procedure** per supportare il passaggio per costante

$$\text{[Y5.1]} \frac{
 \begin{array}{l}
 t \in \text{Simple} \cup \{\text{[void]}\} \\
 p = \text{[constant]} \quad t' = \text{[arr]} t'' N \quad t'' \in \text{Simple} \quad N > 0 \\
 Y_\rho |_0(I) = \perp \quad > [I'/t'] \circ [] :: Y_\rho = Y'_\rho \quad \langle \text{Bs}, Y'_\rho \rangle \rightarrow_{\mathcal{Y}} ([\text{void}], Y''_\rho)
 \end{array}
 }{
 \langle [\text{pcd}] t I F \text{Bs}, Y_\rho \rangle \rightarrow_{\mathcal{Y}} ([\text{void}], [I/[\text{abs}] t [::] t'] \otimes Y_\rho)
 }$$

- 2 **Regole per Exp per [val] I con I identificatore di array**
- 3 **Regole per la trasmissione di paramentri**

- ① **Regole per la Dcl di procedure** per supportare il passaggio per costante con relative regole per la gestione degli errori

$$[E13] \frac{F = [fp] p t' I' \quad p \in \{[value], [ref]\} \quad t' \notin \text{Simple}}{\langle [pcd] t I F Bs, Y_\rho \rangle \rightarrow_Y ([terr], Y_\rho)}$$

$$[E13.1] \frac{F = [fp] p t' I' \quad p = [constant] \quad t' \notin \text{Simple} \quad t' \neq [arr] t'' N}{\langle [pcd] t I F Bs, Y_\rho \rangle \rightarrow_Y ([terr], Y_\rho)}$$

$$[E13.2] \frac{F = [fp] p t' I' \quad p = [constant] \quad t' = [arr] t'' N \quad t'' \notin \text{Simple}}{\langle [pcd] t I F Bs, Y_\rho \rangle \rightarrow_Y ([terr], Y_\rho)}$$

$$[E13.3] \frac{F = [fp] p t' I' \quad p = [constant] \quad t' = [arr] t'' N \quad N \leq 0}{\langle [pcd] t I F Bs, Y_\rho \rangle \rightarrow_Y ([terr], Y_\rho)}$$

- ② **Regole per Exp per [val] I con I identificatore di array**
- ③ **Regole per la trasmissione di paramentri**

- 1 Regole per la Dcl di procedure
- 2 Regole per Exp per $[\text{val}] I$ con I identificatore di array

$$[\text{Y12.1}] \frac{Y_\rho(I) = [\text{arr}]([\text{mut}] t) N \quad t \in \text{Simple} \quad N > 0}{\langle [\text{val}] I, Y_\rho \rangle \rightarrow_{\text{DY}} \langle [\text{arr}]([\text{mut}] t) N, Y_\rho \rangle}$$

- 3 Regole per la trasmissione di paramentri

- 1 Regole per la Decl di procedure
- 2 Regole per Exp per [val] I con I identificatore di array con le relative regole per la gestione degli errori

$$[E16.1] \frac{Y_\rho(I) = [\text{arr}]([\text{mut}]t)N \quad t \notin \text{Simple}}{\langle [\text{val}] I, Y_\rho \rangle \rightarrow_{DY} \langle [\text{terr}], Y_\rho \rangle}$$

$$[E16.2] \frac{Y_\rho(I) = [\text{arr}]([\text{mut}]t)N \quad N \leq 0}{\langle [\text{val}] I, Y_\rho \rangle \rightarrow_{DY} \langle [\text{terr}], Y_\rho \rangle}$$

$$[E17] \frac{Y_\rho(I) = t \quad t \notin \{[\text{mut}]t', [\text{arr}]([\text{mut}]t')N\}}{\langle [\text{val}] I, Y_\rho \rangle \rightarrow_{DY} \langle [\text{terr}], Y_\rho \rangle}$$

- 3 Regole per la trasmissione di paramentri

- 1 Regole per la Decl di procedure
- 2 Regole per Exp per $[\text{val}] I$ con I identificatore di array
- 3 Regole per la trasmissione di paramentri

$$\begin{array}{c}
 \text{fps} = [\text{fp}][\text{constant}] t I \\
 Y_\rho \upharpoonright_0(I) = \perp \\
 \text{aps} = [\text{ap}] \text{exp} \\
 \langle \text{exp}, Y_\rho \rangle \rightarrow_{\mathcal{Y}} (ta, Y_\rho) \\
 t = ta \quad t \in \text{Simple} \\
 \hline
 [\text{Y301}] \frac{}{\langle \text{fps} \triangleleft \text{aps}, Y_\rho \rangle \rightarrow_{\mathcal{Y}} ([\text{void}], Y_\rho)}
 \end{array}$$

$$\begin{array}{c}
 \text{fps} = [\text{fp}][\text{constant}] t I \\
 t = [\text{arr}] t' N \\
 Y_\rho \upharpoonright_0(I) = \perp \\
 \text{aps} = [\text{ap}] \text{exp} \\
 \langle \text{exp}, Y_\rho \rangle \rightarrow_{D\mathcal{Y}} (ta, Y_\rho) \\
 ta = [\text{arr}] ([\text{mut}] ta') N_a \\
 t' = ta' \quad N = N_a \quad t' \in \text{Simple} \\
 \hline
 [\text{Y302}] \frac{}{\langle \text{fps} \triangleleft \text{aps}, Y_\rho \rangle \rightarrow_{\mathcal{Y}} ([\text{void}], Y_\rho)}
 \end{array}$$

- 1 Regole per la Dcl di procedure
- 2 Regole per Exp per $[val]$ I con I identificatore di array
- 3 Regole per la trasmissione di paramentri con le relative regole per la gestione degli errori

$$[E61.3] \frac{\begin{array}{l} fps = [fp][constant] t \text{ I} \\ t \notin \{[arr] t' N\} \cup \text{Simple} \end{array}}{\langle fps \triangleleft aps, Y_\rho \rangle \rightarrow_Y ([terr], Y_\rho)}$$

Occorre modificare la regola [D5] per la dichiarazione di procedure con un solo parametro formale:

$$\begin{array}{c}
 \begin{array}{l}
 t \in \text{Simple} \cup \{\text{[void]}\} \\
 (t' \in \text{Simple}) \vee (p = [\text{constant}] \quad t' = [\text{arr}] t'' \ N \quad t'' \in \text{Simple} \quad N > 0) \\
 \Delta|_0(I) = \perp \\
 Bs = [\text{BlockS}]ds
 \end{array}
 \quad
 \begin{array}{l}
 F = [\text{fp}] p \ t' \ I' \\
 [\text{abs}] t \ [::] t' = t_r \\
 \star I, t_r, F, d, s, \#\Delta\star = v_r
 \end{array}
 \\
 \hline
 \text{[D5]} \frac{}{\langle [\text{pcd}] t \ I \ F \ Bs, (\Delta, \mu) \rangle \rightarrow (\text{[void]}, ([I/(t_r, v_r)] \otimes \Delta, \mu))}
 \end{array}$$

Coerentemente con quanto visto per la regola [Y12.1], la regola X4 diventa

$$\begin{array}{c}
 \sigma = (\Delta, \mu) \\
 \Delta(I) = (t, \text{loc}_{t'}) \\
 t \in \{[\text{mut}] t', [\text{arr}]([\text{mut}] t') N\} \\
 t' \in \text{Simple} \\
 \text{X4:} \frac{}{\langle [\text{val}] I, \sigma \rangle \rightarrow_{\text{DEN}} [t, \text{loc}_{t'}, \sigma]}
 \end{array}$$

Occorre in particolare modificare l'inferenza per la trasmissione di parametri \rightarrow_{TR1} aggiungendo:

$$\begin{array}{l}
 \text{fps} = [\text{fp}][\text{constant}] \text{t I} \\
 \text{aps} = [\text{ap}] \text{exp} \\
 \langle \text{exp}, (\Delta, \mu) \rangle \rightarrow [\text{ta}, \text{va}, (\Delta_1, \mu_1)] \\
 \text{t} = \text{ta} \quad \text{t} \in \text{Simple} \\
 \Delta_C|_0(\text{I}) = \perp \\
 [\text{I}/(\text{ta}, \text{va})] \otimes \Delta_C = \Delta_C^F \\
 (\Delta_C^F, \mu_1) = \sigma_r \quad [] = \text{epi}_r \\
 \hline
 [\text{S302}] \frac{}{\langle \text{fps} \triangleleft \text{aps}, (\Delta, \Delta_C, \mu) \rangle \rightarrow_{TR1} (\sigma_r, \text{epi}_r)}
 \end{array}$$

$$\begin{array}{l}
 \text{fps} = [\text{fp}][\text{constant}] \text{t I} \\
 \text{aps} = [\text{ap}] \text{exp} \\
 \langle \text{exp}, (\Delta, \mu) \rangle \rightarrow_{DEN} [\text{ta}, \text{loc}_a, (\Delta_1, \mu_1)] \\
 \text{ta} = [\text{arr}]([\text{mut}] \text{ta}') N_a \quad \text{t} = [\text{arr}] \text{t}' N \\
 \text{t}' = \text{ta}' \quad N = N_a \quad \text{t}' \in \text{Simple} \\
 \Delta_C|_0(\text{I}) = \perp \\
 [\text{I}/([\text{arr}]([\text{mut}] \text{ta}') N_a, \text{loc}_a)] \otimes \Delta_C = \Delta_C^F \\
 (\Delta_C^F, \mu_1) = \sigma_r \quad [] = \text{epi}_r \\
 \hline
 [\text{S303}] \frac{}{\langle \text{fps} \triangleleft \text{aps}, (\Delta, \Delta_C, \mu) \rangle \rightarrow_{TR1} (\sigma_r, \text{epi}_r)}
 \end{array}$$

- Trasmissione per costante di valori `Simple`: non aumenta l'espressività
- Trasmissione per costante di array
 - trasmissione di array con comportamento assimilabile a quello della trasmissione per reference
 - "emulazione" di procedure con più parametri di tipo omogeneo e `Simple`

- Trasmissione per costante di valori `Simple`: non aumenta l'espressività
- Trasmissione per costante di array
 - trasmissione di array con comportamento assimilabile a quello della trasmissione per reference
 - "emulazione" di procedure con più parametri di tipo omogeneo e `Simple`

- Trasmissione per costante di valori Simple: non aumenta l'espressività
- Trasmissione per costante di array
 - trasmissione di array con comportamento assimilabile a quello della trasmissione per reference

```
void sort2(const int[2] w){  
    if (w[0] > w[1])  
        swap(w);  
}
```

- "emulazione" di procedure con più parametri di tipo omogeneo e Simple

- Trasmissione per costante di valori Simple: non aumenta l'espressività
- Trasmissione per costante di array
 - trasmissione di array con comportamento assimilabile a quello della trasmissione per reference
 - "emulazione" di procedure con più parametri di tipo omogeneo e Simple

```
int minArray(constant int [2] v){
    if (v[0] < v[1])
        return v[0];
    return v[1];
}
...
int [2] aux;
aux[0] = x;
aux[1] = y;
min = minArray(aux);
...
```

Modifiche più rilevanti all'interprete di Small21

- 1 Modifica della SEM_{DCL} per supportare la trasmissione per costante di array e non solo di variabili di tipo Simple.

```
...
| Pcd(ty,ide,fpars,blockP) ->
  (match (declared sk ide,fpars,blockP) with
  ...
  | (_,FP(pf,t,ide),_)
    when not(isConstant pf) && not(isSimple t)
    -> raise(TypeErrorI("E13: dclSem",ide))
  | (_,FP(pf,t,ide),_)
    when isConstant pf && not(isSimple t || isArr t)
    -> (match t with
        | Arr(t1,_)
          when not(isSimple t1)
          -> raise(TypeErrorI("E13.2: dclSem",ide))
        | Arr(_,_)
          -> raise(TypeErrorI("E13.3: dclSem",ide))
        | _ -> raise(TypeErrorI("E13.1: dclSem",ide))
        ↵
      )
  | (_,FP(pf,t,ide),_)
    when isConstant pf && not(isSimple t || isArr t)
    -> raise(TypeErrorI("E13.1: dclSem",ide))
  ...
...

```

2 Modifica della SEM_{DEXP} per introdurre il valore array

```
...
dexpSem dexp (sk, (Store(d,g)as mu)) =
  match dexp with
  | Val ide -> (
    ...
    | DArry(Arr(Mut tr,n), loct)
      when (isSimple tr) && (n>0)
        -> (Arr(Mut tr,n), loct, (sk,mu))
    | DArry(Arr(Mut tr,n), loct)
      when not(isSimple tr)
        -> raise(TypeErrorE("E16.1: dexpSem - ",dexp))
    | DArry(Arr(Mut tr,n), loct)
      -> raise(TypeErrorE("E16.2: dexpSem - ",dexp))
    | -
      -> raise(TypeErrorE("E17: dexpSem - ",dexp))
    ...
  )
...
```

3 Modifica della tri1Fun

```
tri1Fun fps aps sk skc mu =
  match (fps,aps) with
  ...
  | (FP(Constant,t,ide),AP exp)
    when (isSimple t) && not(declared skc ide)
      -> (match expSem exp (sk,mu) with
          | (ta,va,(sk1,mu1))
            when ysame t ta
              -> (let den = DConst(t,va) in
                  let skcF = bindS skc ide den in
                  let sgr = (skcF,mu1) and epir = [] in
                  (sgr,epir))
              | _ -> raise(TypeErrorT("E61.1: Trasmission: Types Mismatch")))
  | (FP(Constant,t,ide),AP exp)
    when (isArr t) && not(declared skc ide)
      -> (match dexpSem exp (sk,mu) with
          | (Arr(Mut tr,n),loca,(sk1,mu1))
            when ysame t (Arr(tr,n))
              -> (let den = DArray(Arr(Mut tr,n),loca) in
                  let skcF = bindS skc ide den in
                  let sgr = (skcF,mu1) and epir = [] in
                  (sgr,epir))
              | _ -> raise(TypeErrorT("E61.1: Trasmission: Types Mismatch")))
  | (FP(Constant,_,ide),AP _)
    when declared skc ide
      -> raise(TypeErrorT("E61: Trasmission: Declared ide"))
  | (FP(Constant,t,_) ,AP _)
    -> raise(TypeErrorT("E61.3: Trasmission: Type not expected"))
  ...
```

Comportamento sugli esempi - double

```
let d1 = Var(Int, "a", EE);;
let dpcd = Pcd(Int, "double", (FP(Constant, Int,
↪ "x")),BlockP(ED,Return (Plus (Val "x", Val
↪ "x"))));;
let dclseq = SeqD(d1,dpcd);;
let c1 = UnL(Upd(Val "a", N 10));;
let c2 = UnL(Upd(Val "a", Apply("double",AP (Val
↪ "a"))));;
let cmdseq = SeqC(c1,c2);;
let doubleTest = Prog("doubleTest",Block(dclseq,
↪ cmdseq));;

printProg doubleTest;;
progSem doubleTest;;
```

```
Program doubleTest{
  int a;
  int double(constant int x){
    return (x + x);
  }
  a = 10;
  a = double(a);    }
- : unit/2 = ()
```

- La funzione calcola il doppio di a
- Durante l'applicazione di double ad a non viene modificato il valore di a, la modifica è successiva all'assegnamento

Comportamento sugli esempi - double

```
let d1 = Var(Int, "a", EE);;
let dpcd = Pcd(Int, "double", (FP(Constant, Int,
↳ "x")),BlockP(ED,Return (Plus (Val "x", Val
↳ "x"))));;
let dclseq = SeqD(d1,dpcd);;
let c1 = UnL(Upd(Val "a", N 10));;
let c2 = UnL(Upd(Val "a", Apply("double",AP (Val
↳ "a"))));;
let cmdseq = SeqC(c1,c2);;
let doubleTest = Prog("doubleTest",Block(dclseq,
↳ cmdseq));;

printProg doubleTest;;
progSem doubleTest;;
```

```
Program doubleTest{
  int a;
  int double(constant int x){
    return (x + x);
  }
  a = 10;
  a = double(a);      }
- : unit/2 = ()
```

- La funzione calcola il doppio di a
- Durante l'applicazione di double ad a non viene modificato il valore di a, la modifica è successiva all'assegnamento

Comportamento sugli esempi: traccia di doubleTest

Comportamento sugli esempi - sort2

```
let d0 = Array(Arr (Int, 2), "a");
let d1 = Array(Arr (Int, 2), "b");
let d2 = SeqD(Var(Int, "temp", EE), ED);;
let stmpcd1 = Upd(Val "temp", Arrow1("v", N 0));;
let stmpcd2 = Upd(Arrow1("v", N 0), Arrow1("v", N 1));;
let stmpcd3 = Upd(Arrow1("v", N 1), Val "temp");;
let stmpcdseq1 = SeqS(SeqS(stmpcd1, stmpcd2), stmpcd3);;
let dpcd1 = Pcd(Void, "swap", (FP(Constant, Arr (Int, 2),
  ↪ "v")), BlockP(d2, stmpcdseq1));;
let stmpcd4 = IfT(GT(Arrow1("w", N 0), Arrow1("w", N
  ↪ 1)), Call("swap", AP(Val "w")));;
let dpcd2 = Pcd(Void, "sort2", (FP(Constant, Arr (Int,
  ↪ 2), "w")), BlockP(ED, stmpcd4));;
let dclseq = SeqD(SeqD(SeqD(d0, d1), dpcd1), dpcd2);;
let c1 = UnL(Upd(Arrow1("a", N 0), N 1));;
let c2 = UnL(Upd(Arrow1("a", N 1), N 2));;
let c3 = UnL(Call ("sort2", AP (Val "a")));;
let c4 = UnL(Upd(Arrow1("b", N 0), N 7));;
let c5 = UnL(Upd(Arrow1("b", N 1), N 5));;
let c6 = UnL(Call ("sort2", AP (Val "b")));;
let cmdseq =
  ↪ SeqC(SeqC(SeqC(SeqC(SeqC(c1, c2), c3), c4), c5), c6));;
let sort2Test = Prog("sort2Test", Block(dclseq, cmdseq));;
```

```
printProg sort2Test;;
progSem sort2Test;;
```

```
Program sort2Test{
  int[2] a;
  int[2] b;
  void swap(constant int[2] v){
    int temp;
    temp = v[0];
    v[0] = v[1];
    v[1] = temp;
  }
  void sort2(constant int[2] w){
    if (w[0] > w[1]) swap(w);
  }
  a[0] = 1;
  a[1] = 2;
  sort2(a);
  b[0] = 7;
  b[1] = 5;
  sort2(b);
}
- : unit/2 = ()
```

Stato finale atteso: a = [1,2], b = [5,7]

Comportamento sugli esempi: traccia di sort2Test

Comportamento sugli esempi - min

```
let d1 = Var(Int, "x", N 23);;
let d2 = Var(Int, "y", N 15);;
let d3 = Var(Int, "min", EE);;
let stmpcd1 = If(LT(Arrow1("v", N
→ 0), Arrow1("v", N 1)), Return(Arrow1("v", N
→ 0)));;
let stmpcd2 = Return(Arrow1("v", N 1));;
let dpcd = Pcd(Int, "minArray", (FP(Constant,
→ Arr (Int, 2), "v")),
→ BlockP(ED, SeqS(stmpcd1, stmpcd2)));;
let d4 = Array(Arr (Int, 2), "aux");;
let dclseq =
→ SeqD(SeqD(SeqD(SeqD(d1, d2), d3), dpcd), d4));;
let c1 = UnL(Upd(Arrow1("aux", N 0), Val "x"));;
let c2 = UnL(Upd(Arrow1("aux", N 1), Val "y"));;
let c3 = UnL(Upd(Val
→ "min", Apply("minArray", AP(Val "aux"))));;
let cmdseq = SeqC(SeqC(c1, c2), c3);;
let minTest =
→ Prog("minTest", Block(dclseq, cmdseq));;
```

```
printProg minTest;;
progSem minTest;;
```

```
Program minTest{
  int x = 23;
  int y = 15;
  int min;
  int minArray(constant int [2] v){
    if (v[0] < v[1]) return v[0]
    return v[1];
  }
  int [2] aux;
  aux[0] = x;
  aux[1] = y;
  min = minArray(aux);
}
- : unit/2 = ()
```

Viene effettivamente calcolato il minimo tra x e y : emuliamo la trasmissione di più parametri.

Comportamento sugli esempi: traccia di minTest

Errori di tipo - Modifica diretta del parametro

```
let d1 = Var(Int, "a", EE);;
let stmpcd1 = Upd(Val "x", Plus(Val "x", Val
↳ "x"));;
let stmpcd2 = Return (Val "x");;
let stmpcdseq = SeqS(stmpcd1,stmpcd2);;
let dpcd = Pcd(Int, "double", (FP(Constant, Int,
↳ "x")),BlockP(ED,stmpcdseq));;
let dclseq = SeqD(d1,dpcd);;
let c1 = UnL(Upd(Val "a", N 10));;
let c2 = UnL(Upd(Val "a", Apply("double",AP (Val
↳ "a"))));;
let cmdseq = SeqC(c1,c2);;
let doubleERRTest =
↳ Prog("doubleERRTest",Block(dclseq,
↳ cmdseq));;
```

```
printProg doubleERRTest;;
progSem doubleERRTest;;
```

```
Program doubleERRTest{
  int a;
  int double(constant int x){
    x = (x + x);
    return x;

  }
  a = 10;
  a = double(a); }
- : unit/2 = ()
```

Traccia della computazione:

```
Stack:
>{doubleERRTest,0,[double/([int::int],$double,[int::int],:fpar,:cmd:,1$);
  a/(Mint,L0)],:cmdNext:[N]}
]
Store:
[L0<-Undef]

Stack:
>{doubleERRTest,0,[double/([int::int],$double,[int::int],:fpar,:cmd:,1$);
  a/(Mint,L0)],:cmdNext:[N]}
]
Store:
[L0<-10]

Exception: TypeErrorE ("E17: dexpSem - ", Val "x").
```

Errori di tipo - Modifica indiretta del parametro

```
let d1 = Var(Int, "a", EE);;
let stmpcd1 = Upd(Val "z", Plus(Val "z", N 1));;
let g = Pcd(Int, "g", (FP(Ref, Int,
  ↪ "z")),BlockP(ED,SeqS(stmpcd1,Return (Val
  ↪ "z"))));;
let dclpcd = Var(Int, "y",EE);;
let stmpcd2 = Upd(Val "y", Apply("g", AP(Val
  ↪ "x")));;
let stmpcd3 = Return(Plus(Val "y", Val "y"));;
let doublev2 = Pcd(Int, "doublev2",
  ↪ (FP(Constant,
  ↪ Int,"x")),BlockP(SeqD(dclpcd,ED),SeqS(stmpcd2,stmp
let dclseq = SeqD(SeqD(d1,g),doublev2));;
let c1 = UnL(Upd(Val "a", N 10));;
let c2 = UnL(Upd(Val "a", Apply("doublev2",AP
  ↪ (Val "a"))));;
let cmdseq = SeqC(c1,c2));;
let doublev2ERR =
  ↪ Prog("doublev2ERR",Block(dclseq, cmdseq));;

printProg doublev2ERR;;
progSem doublev2ERR;;
```

```
Program doublev2ERR{
  int a;
  int g(ref int z){
    z = (z + 1);
    return z;
  }
  int doublev2(constant int x){
    int y;
    y = g(x);
    return (y + y);
  }
  a = 10;
  a = doublev2(a);  }
- : unit/2 = ()
```

Errori di tipo - Modifica indiretta del parametro

Traccia della computazione:

```
Stack:
>{doublev2ERR,0,[doublev2/([int::int],$doublev2,[int::int],:fpar::,cmd:,1$);
  g/([int::int],$g,[int::int],:fpar::,cmd:,1$);
  a/(Mint,L0)],:cmdNext:[N]}
]
Store:
[L0<-Undef]






Stack:
>{doublev2ERR,0,[doublev2/([int::int],$doublev2,[int::int],:fpar::,cmd:,1$);
  g/([int::int],$g,[int::int],:fpar::,cmd:,1$);
  a/(Mint,L0)],:cmdNext:[N]}
]
Store:
[L0<-10]

Exception: TypeErrorE ("E17: dexpSem - ", Val "x").
```

```
let dpcd = Pcd(Int, "double", (FP(Constant,  
↪ Arr(Int, 0), "x")),BlockP(ED,Return (Val  
↪ "x"))));;  
let err133 = Prog("err131", Block(dpcd,  
↪ UnL(ES))));;  
  
printProg err133;;  
progSem err133;;
```

```
Program err131{  
  int double(constant int[0] x){  
    return x;  
  }  
}  
  
- : unit/2 = ()  
Exception: TypeErrorI ("E13.3: declSem", "x").
```

Si possono scrivere analoghi esempi in cui l'errore riportato è [E13],[E13.1] o [E13.2]

-  Marco Bellia. *Materiale delle lezioni del corso di LPL: Laboratori 2,3,4,5,6,7.* May 2020.
-  Marco Bellia. *Materiale delle lezioni del corso di LPL: Lezione 8.* Apr. 2020.
-  Marco Bellia. *Materiale delle lezioni del corso di LPL: Small21-Definizione4.* June 2020.
-  Maurizio Gabbrielli and Simone Martini. *Programming Languages: Principles and Paradigms.* eng. Undergraduate topics in computer science. London: Springer London, Limited, 2010. ISBN: 9781848829138.
-  Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language.* 2nd. Prentice Hall Professional Technical Reference, 1988. ISBN: 0131103709.