Università di Pisa

Dipartimento di Matematica
Corso di Laurea in Matematica

Linguaggi di Programmazione con Laboratorio

Progetto di fine corso

Estensione di Small21 con la trasmissione per costante

Docente:
Prof. Marco Bellia

Studente:
Ivan Bioli

Anno Accademico 2020/2021

1 Presentazione del problema

1.1 Trasmissione per reference e altri tipi di trasmissione

In Small21 sono state implementate, durante lo svolgimento del corso, la trasmissione per valore *(call by value)* e la trasmissione per reference *(call by reference)*. Un ulteriore tipo di trasmissione studiato durante il corso è la trasmissione per costante *(call by constant)*, caratterizzata da:

- i parametri attuali sono valutati nel chiamante come espressioni che devono calcolare valori non modificabili, cioè costanti
- i parametri formali sono legati a tali valori costanti (vedremo meglio nella implementazione come)
- la trasmissione è quindi one-way: l'invocato riceve solo i valori per i parametri, ma non è in grado di modificare i parametri attuali

Possiamo riassumere le principali differenze tra i tipi di trasmissione di cui sopra nella seguente tabella:

	Valore	**Reference**	**Costante**
I **parametri attuali** sono valutati come espressioni che devono calcolare	valori memorizzabili	valori modificabili	valori non modificabili
I **parametri formali** sono legati a	valori modificabili inizializzati al valore del corrispondente parametro attuale	tali valori modificabili	tali valori non modificabili
La **comunicazione con il chiamante** è	one-way: l'invocato riceve solo i valori per i parametri	a memoria condivisa: chiamante e invocato condividono una regione di memoria	one-way: l'invocato riceve solo i valori per i parametri

Dunque i parametri passati per costante sono soggetti al vincolo statico di non poter essere modificati nel corpo della funzione, sia direttamente (tramite assegnamento) sia indirettamente (tramite l'invocazione di una funzione che li modifichi). Se tale vincolo viene soddisfatto, il parametro formale è una costante locale alla procedura.

1.2 Alcuni semplici esempi

Ad esempio, potremmo pensare di voler definire una funzione che, preso in input un intero `x`, restituisce il suo doppio `2*x`. Dato che non è necessario che `x` venga modificato, possiamo usare una trasmissione per costante.

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

dove abbiamo esplicitato la trasmissione per costante tramite la key-word `constant`. Una rappresentazione grafica di quanto avviene la si può vedere in Figura 1, dove supponiamo di aver invocato la funzione `double` su una variabile intera `a`.

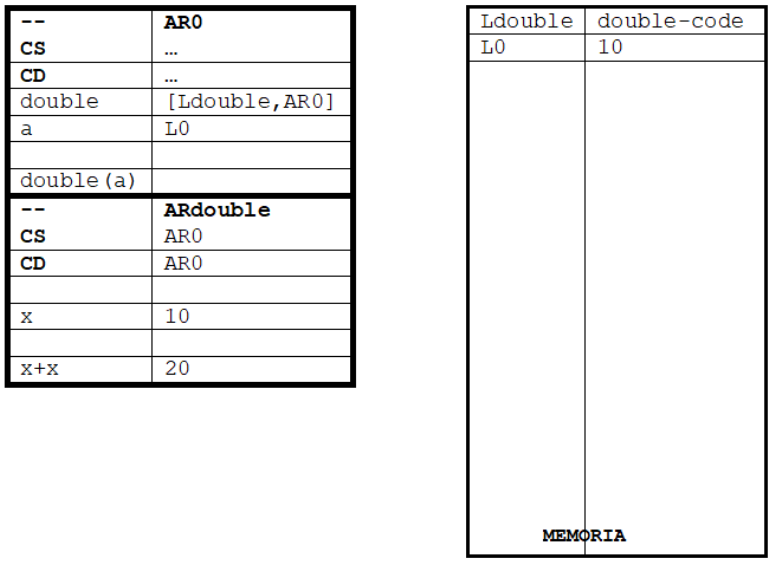

Figura 1

Non è invece legale una implementazione (abbastanza insensata) come la seguente, che memorizza il valore duplicato in `x` prima di restituirlo. In questo caso abbiamo una modifica diretta del parametro formale passato per costante.

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

Supponiamo di avere invece una funzione `f` come questa

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

Se la `g` ha come effetto laterale la modifica della `x` abbiamo una modifica indiretta del parametro e in tal caso il programma viene ritenuto non legale (controllo che, ricordiamo, in Small21 viene eseguito dinamicamente e non staticamente). Ad esempio, è non legale:

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

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

1.3 Trasmissione per costante degli array

Come in C, gli array in Small21 sono visti come valori non modificabili a componenti modificabili e non sono assegnabili (NOTA: in C sono assegnabili i puntatori, non gli array stessi). In Small21, gli identificatori di array hanno come binding nell'ambiente il valore array denotato, vale a dire la coppia ([arr] ([mut] t) N, loc) con N dimensione dell'array e loc prima locazione allocata per gli elementi dell'array stesso. Questo è del tutto analogo a quanto avviene per gli identificatori di costanti e pertanto questo rende gli array trasmissibili per costante.

Inoltre la trasmissione per costante di array, nel caso di Small21, fornisce un comportamento assimilabile a quello della trasmissione per reference. Come detto in precedenza, il parametro formale viene legato al valore array (non modificabile), cioè alla coppia ([arr] ([mut] t) N, loc), non viene copiato nuovamente l'intero array. Pertanto il parametro formale è sì legato a una copia del valore array, ma le N locazioni di memoria allocate a partire da loc sono quelle dell'array originario. In particolare, eventuali modifiche alle componenti dell'array apportate nel corpo della procedura si ripercuotono sull'array originario, non soltanto su una sua copia.

Per quanto detto sopra, ci aspettiamo dunque che la funzione

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

effettivamente scambi i due elementi dell'array, cioè che lo stato finale di

```
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);
}
```

sia a[0] = 2, a[1] = 1. Per capire meglio quanto detto, si osservi l'illustrazione in Figura 2.

Il comportamento finale è molto simile a quanto avviene in C con la "trasmissione di array per valore", anche se cambiano l'implementazione e le motivazioni di tale comportamento. In C infatti, per definizione [5], il valore di una variabile o di un'espressione array è l'indirizzo dell'elemento zero del vettore stesso, cioè un puntatore. Dunque in C viene passato per valore il puntatore al primo elemento dell'array: il puntatore viene copiato, ma l'elemento puntato è quello originario, dunque anche in questo caso eventuali modifiche effettuate all'array puntato dal parametro formale si ripercuotono su quello orginario.

La principale differenza tra il comportamento di Small21 e quello del C, oltre al fatto che il puntatore in C viene solitamente passato per valore e non per costante, è nella flessibilità delle definizioni di procedure che coinvolgono un passaggio per costante di array. Infatti in C indipendentemente dalla dimensione dell'array il parametro formale è sempre lo stesso, vale a dire un puntatore a type dove type è il tipo di dato degli elementi dell'array. Questo permette di definire procedure in grado di operare su array di qualsiasi dimensione, mentre ciò non è possibile in Small21 poiché nel tipo array è inclusa anche la dimensione.

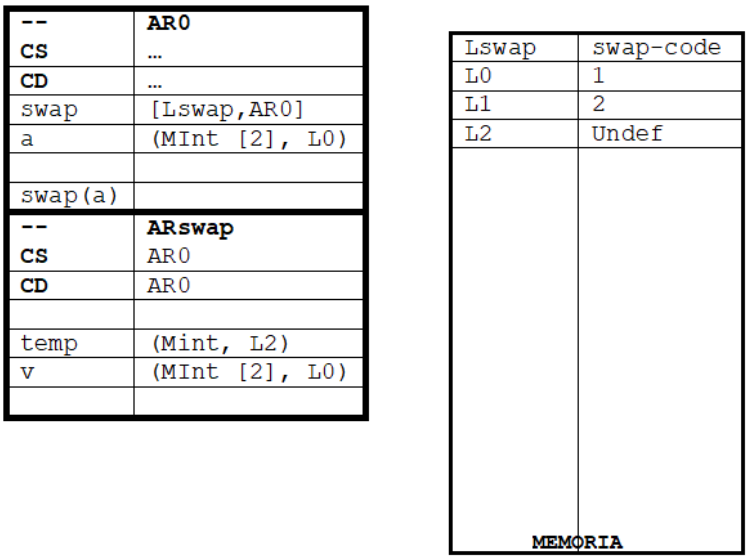

Figura 2

2 Modifiche alla definizione di Small21 e alla AM21

2.1 Sintassi Concreta: una CFG per Small21

...

PPF $\rightarrow \epsilon$ | ref | 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.

...

2.2 Sintassi Astratta

...

```
PPF ::= [value] | [ref] | [constant]  -- Parameter Passing Form
```

...

2.3 Sistema dei tipi Y

In primo luogo dobbiamo modificare le regole per la dichiarazione di funzioni e procedure in modo che sia supportata la trasmissione per costante di array. Aggiungiamo quindi la regola

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

e modifichiamo le regole per la gestione degli errori di tipo con

$$[\text{E13}]\frac{\mathtt{F} = [\mathtt{fp}]\ \mathtt{p}\ \mathtt{t}'\ \mathtt{I}' \quad \mathtt{p} \in \{[\mathtt{value}], [\mathtt{ref}]\} \quad \mathtt{t}' \notin \mathtt{Simple}}{\langle [\mathtt{pcd}]\ \mathtt{t}\ \mathtt{I}\ \mathtt{F}\ \mathtt{Bs}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} ([\mathtt{terr}], \mathtt{Y}_\rho)} \qquad [\text{E13.1}]\frac{\mathtt{F} = [\mathtt{fp}]\ \mathtt{p}\ \mathtt{t}'\ \mathtt{I}' \quad \mathtt{p} = [\mathtt{constant}] \quad \mathtt{t}' \notin \mathtt{Simple} \quad \mathtt{t}' \neq [\mathtt{arr}]\ \mathtt{t}''\ \mathtt{N}}{\langle [\mathtt{pcd}]\ \mathtt{t}\ \mathtt{I}\ \mathtt{F}\ \mathtt{Bs}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} ([\mathtt{terr}], \mathtt{Y}_\rho)}$$

$$[\text{E13.2}]\frac{\mathtt{F} = [\mathtt{fp}]\ \mathtt{p}\ \mathtt{t}'\ \mathtt{I}' \quad \mathtt{p} = [\mathtt{constant}] \quad \mathtt{t}' = [\mathtt{arr}]\ \mathtt{t}''\ \mathtt{N} \quad \mathtt{t}'' \notin \mathtt{Simple}}{\langle [\mathtt{pcd}]\ \mathtt{t}\ \mathtt{I}\ \mathtt{F}\ \mathtt{Bs}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} ([\mathtt{terr}], \mathtt{Y}_\rho)} \qquad [\text{E13.3}]\frac{\mathtt{F} = [\mathtt{fp}]\ \mathtt{p}\ \mathtt{t}'\ \mathtt{I}' \quad \mathtt{p} = [\mathtt{constant}] \quad \mathtt{t}' = [\mathtt{arr}]\ \mathtt{t}''\ \mathtt{N} \quad \mathtt{N} \leq 0}{\langle [\mathtt{pcd}]\ \mathtt{t}\ \mathtt{I}\ \mathtt{F}\ \mathtt{Bs}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} ([\mathtt{terr}], \mathtt{Y}_\rho)}$$

Occorre modificare le regole del Sistema Y per `Exp` in modo tale che si possa ottenere il valore array, cioè in modo da dare senso, coerentemente con quanto detto in precedenza, a `[val] I` con `I` identificatore di array. Aggiungiamo dunque:

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

Modifichiamo quindi anche le regole per gli errori di tipo, aggiungendo:

$$[\text{E16.1}]\frac{\mathtt{Y}_\rho(\mathtt{I}) = [\mathtt{arr}]\,([\mathtt{mut}]\,\mathtt{t})\,\mathtt{N} \quad \mathtt{t} \notin \mathtt{Simple}}{\langle [\mathtt{val}]\,\mathtt{I}, \mathtt{Y}_\rho\rangle \to_{\mathtt{DY}} ([\mathtt{terr}], \mathtt{Y}_\rho)} \qquad [\text{E16.2}]\frac{\mathtt{Y}_\rho(\mathtt{I}) = [\mathtt{arr}]\,([\mathtt{mut}]\,\mathtt{t})\,\mathtt{N} \quad \mathtt{N} \leq 0}{\langle [\mathtt{val}]\,\mathtt{I}, \mathtt{Y}_\rho\rangle \to_{\mathtt{DY}} ([\mathtt{terr}], \mathtt{Y}_\rho)}$$

e modificando

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

Si noti che i controlli su $\mathtt{t} \in \mathtt{Simple}$ e $\mathtt{N} > 0$ sono in realtà superflui se l'identificatore `I` è stato in precedenza definito secondo le regole per la dichiarazione di identificatori di array.

Inoltre, dobbiamo modificare anche la parte relativa alla trasmissione di parametri, aggiungendo le regole

$$[\text{Y301}]\frac{\begin{array}{c}\mathtt{fps} = [\mathtt{fp}][\mathtt{constant}]\,\mathtt{t}\,\mathtt{I} \\ \mathtt{Y}_\rho|_0(\mathtt{I}) = \bot \\ \mathtt{aps} = [\mathtt{ap}]\,\mathtt{exp} \\ \langle \mathtt{exp}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{ta}, \mathtt{Y}_\rho) \\ \mathtt{t} = \mathtt{ta} \quad \mathtt{t} \in \mathtt{Simple}\end{array}}{\langle \mathtt{fps} \lhd \mathtt{aps}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} ([\mathtt{void}], \mathtt{Y}_\rho)} \qquad [\text{Y302}]\frac{\begin{array}{c}\mathtt{fps} = [\mathtt{fp}][\mathtt{constant}]\,\mathtt{t}\,\mathtt{I} \\ \mathtt{t} = [\mathtt{arr}]\,\mathtt{t}'\,\mathtt{N} \\ \mathtt{Y}_\rho|_0(\mathtt{I}) = \bot \\ \mathtt{aps} = [\mathtt{ap}]\,\mathtt{exp} \\ \langle \mathtt{exp}, \mathtt{Y}_\rho\rangle \to_{\mathtt{DY}} (\mathtt{ta}, \mathtt{Y}_\rho) \\ \mathtt{ta} = [\mathtt{arr}]\,([\mathtt{mut}]\,\mathtt{ta}')\,\mathtt{N}_\mathtt{a} \\ \mathtt{t}' = \mathtt{ta}' \quad \mathtt{N} = \mathtt{N}_\mathtt{a} \quad \mathtt{t}' \in \mathtt{Simple}\end{array}}{\langle \mathtt{fps} \lhd \mathtt{aps}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} ([\mathtt{void}], \mathtt{Y}_\rho)}$$

e la regola per gli errori di tipo:

$$[\text{E61.3}]\frac{\mathtt{fps} = [\mathtt{fp}][\mathtt{constant}]\,\mathtt{t}\,\mathtt{I} \quad \mathtt{t} \notin \{[\mathtt{arr}]\,\mathtt{t}'\,\mathtt{N}\} \cup \mathtt{Simple}}{\langle \mathtt{fps} \lhd \mathtt{aps}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} ([\mathtt{terr}], \mathtt{Y}_\rho)}$$

Anche in questo caso, se la procedura è stata dichiarata secondo le regola del sistema Y, il tipo `t` sarà necessariamente tra quelli corretti.

2.4 Regole di inferenza

Per supportare la trasmissione per costante di array occorre modificare leggermente la regola di inferenza di $\mathtt{SEM}_{\mathtt{DCL}}$ per la dichiarazione di procedure con un solo parametro formale, cioè la [D5]

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

Occorre modificare la $\texttt{SEM}_{\texttt{EXP}}$ coerentemente con quanto visto per la regola [Y12.1]. In particolare la regola X4 diventa:

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

Analogamente, occorre modificare $\texttt{SEM}_{\texttt{CMD}}$ e in particolare l'inferenza per la trasmissione di parametri $\rightarrow_{\texttt{TR1}}$ aggiungendo:

$$[\text{S302}]\ \frac{\begin{array}{c}\texttt{fps} = [\texttt{fp}][\texttt{constant}]\,\texttt{t}\,\texttt{I} \\ \texttt{aps} = [\texttt{ap}]\,\texttt{exp} \\ \langle \texttt{exp}, (\Delta, \mu)\rangle \rightarrow \lfloor \texttt{ta}, \texttt{va}, (\Delta_1, \mu_1)\rfloor \\ \texttt{t} = \texttt{ta} \qquad \texttt{t} \in \texttt{Simple} \\ \Delta_{\texttt{C}}|_0(\texttt{I}) = \bot \\ [\texttt{I}/(\texttt{ta}, \texttt{va})] \otimes \Delta_{\texttt{C}} = \Delta^{\texttt{F}}_{\texttt{C}} \\ (\Delta^{\texttt{F}}_{\texttt{C}}, \mu_1) = \sigma_r \qquad [] = \texttt{epi}_{\texttt{r}}\end{array}}{\langle \texttt{fps} \lhd \texttt{aps}, (\Delta, \Delta_{\texttt{C}}, \mu)\rangle \rightarrow_{\texttt{TR1}} (\sigma_r, \texttt{epi}_{\texttt{r}})}$$

$$[\text{S303}]\ \frac{\begin{array}{c}\texttt{fps} = [\texttt{fp}][\texttt{constant}]\,\texttt{t}\,\texttt{I} \\ \texttt{aps} = [\texttt{ap}]\,\texttt{exp} \\ \langle \texttt{exp}, (\Delta, \mu)\rangle \rightarrow_{\texttt{DEN}} \lfloor \texttt{ta}, \texttt{loc}_{\texttt{a}}, (\Delta_1, \mu_1)\rfloor \\ \texttt{ta} = [\texttt{arr}]\,([\texttt{mut}]\,\texttt{ta}')\,\texttt{N}_{\texttt{a}} \qquad \texttt{t} = [\texttt{arr}]\,\texttt{t}'\,\texttt{N} \\ \texttt{t}' = \texttt{ta}' \qquad \texttt{N} = \texttt{N}_{\texttt{a}} \qquad \texttt{t}' \in \texttt{Simple} \\ \Delta_{\texttt{C}}|_0(\texttt{I}) = \bot \\ [\texttt{I}/([\texttt{arr}]\,([\texttt{mut}]\,\texttt{ta}')\,\texttt{N}_{\texttt{a}}, \texttt{loc}_{\texttt{a}})] \otimes \Delta_{\texttt{C}} = \Delta^{\texttt{F}}_{\texttt{C}} \\ (\Delta^{\texttt{F}}_{\texttt{C}}, \mu_1) = \sigma_r \qquad [] = \texttt{epi}_{\texttt{r}}\end{array}}{\langle \texttt{fps} \lhd \texttt{aps}, (\Delta, \Delta_{\texttt{C}}, \mu)\rangle \rightarrow_{\texttt{TR1}} (\sigma_r, \texttt{epi}_{\texttt{r}})}$$

3 Nuove strutture e forme di programma esprimibili

Alla luce delle modifiche apportate, abbiamo delle nuove strutture e delle nuove forme di programma esprimibili. La trasmissione per costante è un tipo di trasmissione aggiuntivo rispetto a quelli già presenti in Small21 (value e reference). Se ci limitassimo alla trasmissione per costante di valori soltanto di tipo `Simple`, questa sarebbe del tutto equivalente a una trasmissione per valore in cui il parametro formale non viene modificato, direttamente o indirettamente, nel blocco procedura. Questo uso non aumenterebbe pertanto l'espressività di Small21.

La formalizzazione data rende però la trasmissione per costante applicabile alla trasmissione di array, precedentemente non prevista e che invece aumenta la espressività di Small21.

In primo luogo possiamo trasmettere array per costante, con un comportamento assimilabile a quello della trasmissione per reference. Ad esempio, potremmo ordinare in ordine cresente un array di dimensione 2 tramite:

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

dove `swap` e la funzione definita in precedenza.

Inoltre, tramite la trasmissione per costante di array, possiamo "emulare" procedure e funzioni con più di un parametro, purché tali parametri siano omogenei e di tipo `Simple`. Basta infatti

memorizzare tali parametri nelle componenti di un array e trasmettere per costante tale array alla funzione. Ad esempio, possiamo pensare di definire una funzione che calcola il minimo tra due interi memorizzati in un array

```
int minArray(constant int[2] v){
    if (v[0] < v[1])
        return v[0];
    return v[1];
}
```

e poi applicarla al calcolo del minimo tra `x` e `y` tramite

```
...
int[2] aux;
aux[0] = x;
aux[1] = y;
min = minArray(aux);
...
```

4 Modifiche all'interprete di Small21

Abbiamo inoltre apportato le seguenti modifiche all'interprete di Small21:

1. Aggiunta della trasmissione per costante alle PPF (Parameter Passing Forms)

```
73  ppf =
74      Value
75      | Ref
76      | Constant
```

   e modificata di conseguenza anche la `toStringPPF`

```
224 toStringPPF = (function
225         | Value -> "value"
226         | Ref -> "ref"
227         | Constant -> "constant"
228         )
```

2. Funzione ausiliaria che dice se un tipo è $[\texttt{arr}]\,\texttt{t}'\,\texttt{N}$ con $\texttt{t}' \in \texttt{Simple}$, $\texttt{N} > 0$

```
169 isArr = (function
170         | Arr(t,n)
171             when (isSimple t) && (n>0) -> true
172         | _ -> false)
```

3. Modifica della $\texttt{SEM}_{\texttt{DCL}}$ 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
        | (_,_,_)
              when not(isSimple ty) && (ty <> Void)
              -> raise(TypeErrorI("E12: dclSem",ide))
        | (true,_,_) -> raise(TypeErrorI("E14: dclSem",ide))
        | (_,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))
        | (_,FP(pf,t,_),BlockP(d,s))                (* checked 21.05.21 *)
              -> (let tr = Abs(ty,[t]) in
                  let clos = ClosT(ide,tr,fpars,d,s,sizeS sk) in
                  let den = DAbs(tr,clos) in
                  let skF = bindS sk ide den in
                  (Void,(skF,mu)))
        | (_,EFP,BlockP(d,s))
              -> (let tr = Abs(ty,[]) in
                  let clos = ClosT(ide,ty,EFP,d,s,sizeS sk) in
                  let den = DAbs(tr,clos) in
                  let skF = bindS sk ide den in
                  (Void,(skF,mu))))
```

dove abbiamo definito

```
isConstant = (function
        | Constant -> true
        | _ -> false)
```

4. Modifica della $\texttt{SEM}_{\texttt{DEXP}}$ per introdurre il valore array

```
dexpSem dexp (sk,(Store(d,g)as mu)) =
    match dexp with
      | Val ide -> (
          match getS sk ide with
```

```
            | DVar(Mut t,loct)
                when isSimple t
                -> (Mut t,loct,(sk,mu))
            | DVar(Mut t,loct)
                -> raise(TypeErrorE("E16: dexpSem - ",dexp))
            | 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)))
```

5. Modifica della `tri1Fun` per trattare la trasmissione per costante di `Simple` e array

```
tr1Fun fps aps sk skc mu =
   match (fps,aps) with
     | (FP(Value,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 (loct,mu2) = allocate mu1 1 in
                      let mu3 = upd mu2 loct (eTOm va) in
                      let den = DVar(Mut t,loct) in
                      let skcF = bindS skc ide den in
                      let sgr = (skcF,mu3) and epir = [] in
                      (sgr,epir))
              | _ -> raise(TypeErrorT("E61.1: Trasmission: Type not
              ↪  expected")))
     | (FP(Value,_,ide),AP _)
         when declared skc ide
         -> raise(TypeErrorT("E61: Trasmission: Declared ide"))
     | (FP(Value,t,_),AP _)
         -> raise(TypeErrorT("E61.2: Trasmission: Type not expected"))
     | (FP(Ref,t,ide),AP exp)
         when (isSimple t) && not(declared skc ide)
         -> (match dexpSem exp (sk,mu) with
              | (Mut ta,loca,(sk1,mu1))
                  when ysame t ta
                  -> (let den = DVar(Mut ta,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(Ref,_,ide),AP _)
```

```
            when declared skc ide
            -> raise(TypeErrorT("E61: Trasmission: Declared ide"))
        | (FP(Ref,t,_),AP _)
            -> raise(TypeErrorT("E61.2: Trasmission: Type not expected"))
        | (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 = DArry(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"))
        | (EFP,EAP)
            -> let sgr =(skc,mu) and epir = [] in
               (sgr,epir)
        | (EFP,_)
            -> raise(TypeErrorT("E62: Trasmission: Too Many Params"))
        | (_,EAP)
            -> raise(TypeErrorT("E63: Trasmission: Too Few Params"))
```

5 Comportamento dell'interprete sugli esempi introdotti

5.1 double

La funzione `double` di cui nella Sezione 1.2 può essere testata tramite il seguente programma, che la definisce e calcola il doppio di `a` = 10.

```
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;;
```

Traccia della computazione:

```
Program doubleTest{
     int a;
     int double(constant int x){
            return (x + x);
            }
     a = 10;
     a = double(a);     }

- : unit/2 = ()

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


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


Stack:
>{double,1,[x/(int,10)],:cmdNext:,20}
{doubleTest,0,[double/([int::int],$double,[int::int],:fpar:,:cmd:,1$);
 a/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-10]


Stack:
>{doubleTest,0,[double/([int::int],$double,[int::int],:fpar:,:cmd:,1$);
 a/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-20]


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

Si può notare come durante l'applicazione di `double` ad `a` non venga modificato il valore di `a`, la modifica è successiva all'assegnamento.

5.2 arraySwap

Coerentemente con quanto detto in precedenza sulla trasmissione per costante di array, possiamo vedere che la computazione del programma `arraySwap` ha come stato finale quello atteso, vale a dire $\texttt{a}[0] = 2,\ \texttt{a}[1] = 1$.

```
let d1 = Array(Arr (Int, 2),"a");;
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 stmpcdseq = SeqS(SeqS(stmpcd1,stmpcd2),stmpcd3);;
let dpcd = Pcd(Void, "swap", (FP(Constant, Arr (Int, 2), "v")),
↪  BlockP(d2,stmpcdseq));;
let dclseq = SeqD(d1,dpcd);;
let c1 = UnL(Upd(Arrow1("a", N 0), N 1));;
let c2 = UnL(Upd(Arrow1("a", N 1), N 2));;
let c3 = UnL(Call ("swap", AP (Val "a")));;
let cmdseq = SeqC(SeqC(c1,c2),c3);;
let arraySwap = Prog("arraySwap",Block(dclseq,cmdseq));;

printProg arraySwap;;
progSem arraySwap;;
```

Traccia della computazione:

```
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);     }


Stack:
>{arraySwap,0,[swap/([void:::int[2]],$swap,[void:::int[2]],:fpar:,:cmd:,1$);
 a/(:Mint[2],L0)],:cmdNext:,[N]}
]
Store:
[L0<-Undef,L1<-Undef]


Stack:
>{arraySwap,0,[swap/([void:::int[2]],$swap,[void:::int[2]],:fpar:,:cmd:,1$);
 a/(:Mint[2],L0)],:cmdNext:,[N]}
]
Store:
[L0<-1,L1<-Undef]


Stack:
>{arraySwap,0,[swap/([void:::int[2]],$swap,[void:::int[2]],:fpar:,:cmd:,1$);
 a/(:Mint[2],L0)],:cmdNext:,[N]}
]
Store:
[L0<-1,L1<-2]


Stack:
>{swap,1,[temp/(Mint,L2);
 v/(:Mint[2],L0)],:cmdNext:,[N]}
{arraySwap,0,[swap/([void:::int[2]],$swap,[void:::int[2]],:fpar:,:cmd:,1$);
 a/(:Mint[2],L0)],:cmdNext:,[N]}
```

```
]
Store:
[L0<-1,L1<-2,L2<-1]


Stack:
>{swap,1,[temp/(Mint,L2);
 v/(:Mint[2],L0)],:cmdNext:,[N]}
{arraySwap,0,[swap/([void:::int[2]],$swap,[void:::int[2]],:fpar:,:cmd:,1$);
 a/(:Mint[2],L0)],:cmdNext:,[N]}
]
Store:
[L0<-2,L1<-2,L2<-1]


Stack:
>{swap,1,[temp/(Mint,L2);
 v/(:Mint[2],L0)],:cmdNext:,[N]}
{arraySwap,0,[swap/([void:::int[2]],$swap,[void:::int[2]],:fpar:,:cmd:,1$);
 a/(:Mint[2],L0)],:cmdNext:,[N]}
]
Store:
[L0<-2,L1<-1,L2<-1]


Stack:
>{arraySwap,0,[swap/([void:::int[2]],$swap,[void:::int[2]],:fpar:,:cmd:,1$);
 a/(:Mint[2],L0)],:cmdNext:,[N]}
]
Store:
[L0<-2,L1<-1,L2<-1]


SUCCESSFUL_TERMINATION
```

5.3 sort2

Applicando la `sort2` a due array $\mathtt{a} = [1;2]$ e $\mathtt{b} = [7;5]$ possiamo notare come il primo venga lasciato invariato mentre il secondo venga riordinato, coerentemente con quanto ci aspettavamo.

```
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;;
```

Traccia della computazione

```
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 = ()

Stack:
>{sort2Test,0,[sort2/([void:::int[2]],$sort2,[void:::int[2]],:fpar:,:cmd:,1$);
 swap/([void:::int[2]],$swap,[void:::int[2]],:fpar:,:cmd:,1$);
 b/(:Mint[2],L2);
 a/(:Mint[2],L0)],:cmdNext:,[N]}
]
Store:
[L0<-Undef,L1<-Undef,L2<-Undef,L3<-Undef]


Stack:
>{sort2Test,0,[sort2/([void:::int[2]],$sort2,[void:::int[2]],:fpar:,:cmd:,1$);
 swap/([void:::int[2]],$swap,[void:::int[2]],:fpar:,:cmd:,1$);
 b/(:Mint[2],L2);
 a/(:Mint[2],L0)],:cmdNext:,[N]}
]
Store:
[L0<-1,L1<-Undef,L2<-Undef,L3<-Undef]


Stack:
>{sort2Test,0,[sort2/([void:::int[2]],$sort2,[void:::int[2]],:fpar:,:cmd:,1$);
 swap/([void:::int[2]],$swap,[void:::int[2]],:fpar:,:cmd:,1$);
 b/(:Mint[2],L2);
 a/(:Mint[2],L0)],:cmdNext:,[N]}
]
Store:
[L0<-1,L1<-2,L2<-Undef,L3<-Undef]


Stack:
>{sort2,1,[w/(:Mint[2],L0)],:cmdNext:,[N]}
{sort2Test,0,[sort2/([void:::int[2]],$sort2,[void:::int[2]],:fpar:,:cmd:,1$);
 swap/([void:::int[2]],$swap,[void:::int[2]],:fpar:,:cmd:,1$);
 b/(:Mint[2],L2);
 a/(:Mint[2],L0)],:cmdNext:,[N]}
]
Store:
```

```
[L0<-1,L1<-2,L2<-Undef,L3<-Undef]


Stack:
>{sort2Test,0,[sort2/([void:::int[2]],$sort2,[void:::int[2]],:fpar:,:cmd:,1$);
 swap/([void:::int[2]],$swap,[void:::int[2]],:fpar:,:cmd:,1$);
 b/(:Mint[2],L2);
 a/(:Mint[2],L0)],:cmdNext:,[N]}
]
Store:
[L0<-1,L1<-2,L2<-Undef,L3<-Undef]


Stack:
>{sort2Test,0,[sort2/([void:::int[2]],$sort2,[void:::int[2]],:fpar:,:cmd:,1$);
 swap/([void:::int[2]],$swap,[void:::int[2]],:fpar:,:cmd:,1$);
 b/(:Mint[2],L2);
 a/(:Mint[2],L0)],:cmdNext:,[N]}
]
Store:
[L0<-1,L1<-2,L2<-7,L3<-Undef]


Stack:
>{sort2Test,0,[sort2/([void:::int[2]],$sort2,[void:::int[2]],:fpar:,:cmd:,1$);
 swap/([void:::int[2]],$swap,[void:::int[2]],:fpar:,:cmd:,1$);
 b/(:Mint[2],L2);
 a/(:Mint[2],L0)],:cmdNext:,[N]}
]
Store:
[L0<-1,L1<-2,L2<-7,L3<-5]


Stack:
>{swap,2,[temp/(Mint,L4);
 v/(:Mint[2],L2)],:cmdNext:,[N]}
{sort2,1,[w/(:Mint[2],L2)],:cmdNext:,[N]}
{sort2Test,0,[sort2/([void:::int[2]],$sort2,[void:::int[2]],:fpar:,:cmd:,1$);
 swap/([void:::int[2]],$swap,[void:::int[2]],:fpar:,:cmd:,1$);
 b/(:Mint[2],L2);
 a/(:Mint[2],L0)],:cmdNext:,[N]}
]
Store:
[L0<-1,L1<-2,L2<-7,L3<-5,L4<-7]


Stack:
>{swap,2,[temp/(Mint,L4);
 v/(:Mint[2],L2)],:cmdNext:,[N]}
{sort2,1,[w/(:Mint[2],L2)],:cmdNext:,[N]}
{sort2Test,0,[sort2/([void:::int[2]],$sort2,[void:::int[2]],:fpar:,:cmd:,1$);
 swap/([void:::int[2]],$swap,[void:::int[2]],:fpar:,:cmd:,1$);
 b/(:Mint[2],L2);
 a/(:Mint[2],L0)],:cmdNext:,[N]}
]
Store:
[L0<-1,L1<-2,L2<-5,L3<-5,L4<-7]


Stack:
>{swap,2,[temp/(Mint,L4);
 v/(:Mint[2],L2)],:cmdNext:,[N]}
{sort2,1,[w/(:Mint[2],L2)],:cmdNext:,[N]}
{sort2Test,0,[sort2/([void:::int[2]],$sort2,[void:::int[2]],:fpar:,:cmd:,1$);
 swap/([void:::int[2]],$swap,[void:::int[2]],:fpar:,:cmd:,1$);
 b/(:Mint[2],L2);
 a/(:Mint[2],L0)],:cmdNext:,[N]}
]
Store:
[L0<-1,L1<-2,L2<-5,L3<-7,L4<-7]
```

```
Stack:
>{sort2,1,[w/(:Mint[2],L2)],:cmdNext:,[N]}
{sort2Test,0,[sort2/([void:::int[2]],$sort2,[void:::int[2]],:fpar:,:cmd:,1$);
 swap/([void:::int[2]],$swap,[void:::int[2]],:fpar:,:cmd:,1$);
 b/(:Mint[2],L2);
 a/(:Mint[2],L0)],:cmdNext:,[N]}
]
Store:
[L0<-1,L1<-2,L2<-5,L3<-7,L4<-7]


Stack:
>{sort2Test,0,[sort2/([void:::int[2]],$sort2,[void:::int[2]],:fpar:,:cmd:,1$);
 swap/([void:::int[2]],$swap,[void:::int[2]],:fpar:,:cmd:,1$);
 b/(:Mint[2],L2);
 a/(:Mint[2],L0)],:cmdNext:,[N]}
]
Store:
[L0<-1,L1<-2,L2<-5,L3<-7,L4<-7]


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

5.4 Emulazione della trasmissione di più parametri omogenei

```
let d1 = Var(Int,"x",N 23);;
let d2 = Var(Int,"y",N 15);;
let d3 = Var(Int,"min",EE);;
let stmpcd1 = IfT(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;;
```

Traccia della computazione

```
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 = ()

Stack:
>{minTest,0,[aux/(:Mint[2],L3);
 minArray/([int:::int[2]],$minArray,[int:::int[2]],:fpar:,:cmd:,1$);
 min/(Mint,L2);
 y/(Mint,L1);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-23,L1<-15,L2<-Undef,L3<-Undef,L4<-Undef]


Stack:
>{minTest,0,[aux/(:Mint[2],L3);
 minArray/([int:::int[2]],$minArray,[int:::int[2]],:fpar:,:cmd:,1$);
 min/(Mint,L2);
 y/(Mint,L1);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-23,L1<-15,L2<-Undef,L3<-23,L4<-Undef]


Stack:
>{minTest,0,[aux/(:Mint[2],L3);
 minArray/([int:::int[2]],$minArray,[int:::int[2]],:fpar:,:cmd:,1$);
 min/(Mint,L2);
 y/(Mint,L1);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-23,L1<-15,L2<-Undef,L3<-23,L4<-15]


Stack:
>{minArray,1,[v/(:Mint[2],L3)],:cmdNext:,[N]}
{minTest,0,[aux/(:Mint[2],L3);
 minArray/([int:::int[2]],$minArray,[int:::int[2]],:fpar:,:cmd:,1$);
 min/(Mint,L2);
 y/(Mint,L1);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-23,L1<-15,L2<-Undef,L3<-23,L4<-15]


Stack:
>{minArray,1,[v/(:Mint[2],L3)],:cmdNext:,15}
{minTest,0,[aux/(:Mint[2],L3);
 minArray/([int:::int[2]],$minArray,[int:::int[2]],:fpar:,:cmd:,1$);
 min/(Mint,L2);
 y/(Mint,L1);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-23,L1<-15,L2<-Undef,L3<-23,L4<-15]


Stack:
>{minTest,0,[aux/(:Mint[2],L3);
 minArray/([int:::int[2]],$minArray,[int:::int[2]],:fpar:,:cmd:,1$);
 min/(Mint,L2);
 y/(Mint,L1);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-23,L1<-15,L2<-15,L3<-23,L4<-15]


SUCCESSFUL_TERMINATION
```

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

5.5 Errori di tipo

5.5.1 Modifica diretta del parametro formale nella trasmissione per costante

```
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;;
```

Traccia della computazione:

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

            }
     a = 10;
     a = double(a);     }

- : unit/2 = ()

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").
```

5.5.2 Modifica indiretta del parametro formale nella trasmissione per costante

```
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,stmpcd3)));;
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;;
```

Traccia della computazione:

```
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 = ()

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").
```

5.6 E13*

```
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;;
```

Traccia della computazione

```
Program err131{
     int double(constant int[0] x){
            return x;
            }     }

- : unit/2 = ()
Exception: TypeErrorI ("E13.3: dclSem", "x").
```

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

5.7 E17

```
let d1 = Var(Int, "x", N 1);;
let d2 = Const(Int, "y", N 2);;
let stm1 = Upd(Val "y", Val "x");;
let err17 = Prog("err17", Block(SeqD(d1,d2), UnL(stm1)));;

printProg err17;;
progSem err17;;
```

Traccia della compuazione

```
Program err17{
     int x = 1;
     const int y = 2;
     y = x;     }

- : unit/2 = ()

Stack:
>{err17,0,[y/(int,2);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-1]

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

Riferimenti bibliografici

[1] Marco Bellia. *Materiale delle lezioni del corso di LPL: Laboratori 2,3,4,5,6,7*. Apr. 2020.

[2] Marco Bellia. *Materiale delle lezioni del corso di LPL: Lezione 8*. Apr. 2020.

[3] Marco Bellia. *Materiale delle lezioni del corso di LPL: Small21-Definizione4*. Giu. 2020.

[4] Maurizio Gabbrielli e Simone Martini. *Programming Languages: Principles and Paradigms*. eng. Undergraduate topics in computer science. London: Springer London, Limited, 2010. ISBN: 9781848829138.

[5] Brian W. Kernighan e Dennis M. Ritchie. *The C Programming Language*. 2nd. Prentice Hall Professional Technical Reference, 1988. ISBN: 0131103709.