Linguaggi di programmazione e laboratorio

Seminario di fine corso:
Trasmissione per Result e per Value-Result in `Small21`

Alessandro Sferlazza

Università di Pisa, Dipartimento di Matematica

19 ottobre 2021

Scopi del progetto

Estendere le tipologie di trasmissione di parametri presenti in `Small21` con le seguenti:

1. Trasmissione per Result
2. Trasmissione per Value-Result

Scopi del progetto

Estendere le tipologie di trasmissione di parametri presenti in `Small21` con le seguenti:

1. Trasmissione per Result
2. Trasmissione per Value-Result

A questo scopo, utilizzeremo il meccanismo di Epilogo di cui è dotata `AM21`, attualmente non in uso con i meccanismi di trasmissione finora implementati nel linguaggio.

Trasmissione per Result: Presentazione

```
fps ::= [fp] [res] Type Ide
```

Nella trasmissione di parametri per Result,

- il parametro attuale è valutato nel chiamante come un'espressione che calcola un **valore modificabile**

Trasmissione per Result: Presentazione

```
fps ::= [fp] [res] Type Ide
```

Nella trasmissione di parametri per Result,

- il parametro attuale è valutato nel chiamante come un'espressione che calcola un **valore modificabile**
- il parametro formale è legato ad un diverso valore modificabile, utilizzato *one-way* in sola scrittura, collegato al parametro attuale tramite il meccanismo di Epilogo.

Trasmissione per Value-Result: Presentazione

```
fps ::= [fp] [valres] Type Ide
```

Nella trasmissione di parametri per Value-Result,

- il parametro attuale è contemporaneamente un'espressione che calcola:
  - un valore **memorizzabile** `val` (Value),
  - un valore **modificabile** `loc` (Result).

Trasmissione per Value-Result: Presentazione

```
fps ::= [fp] [valres] Type Ide
```

Nella trasmissione di parametri per Value-Result,

- il parametro attuale è contemporaneamente un'espressione che calcola:
  - un valore **memorizzabile** `val` (Value),
  - un valore **modificabile** `loc` (Result).
- il parametro formale è legato a tali valori *two-way* senza condivisione di memoria.

Trasmissione per Value-Result: Presentazione

```
fps ::= [fp] [valres] Type Ide
```

Più precisamente, il parametro formale è:

- legato ad un **nuovo** valore modificabile inizializzato al valore memorizzabile `val` del parametro attuale corrispondente, durante l'esecuzione del corpo della funzione (lettura)
- valutato e riassegnato al valore modificabile `loc` del parametro attuale, quando termina l'esecuzione della funzione (scrittura).

Trasmissione per Value-Result: Presentazione

```
fps ::= [fp] [valres] Type Ide
```

Più precisamente, il parametro formale è:

- legato ad un **nuovo** valore modificabile inizializzato al valore memorizzabile `val` del parametro attuale corrispondente, durante l'esecuzione del corpo della funzione (lettura)
- valutato e riassegnato al valore modificabile `loc` del parametro attuale, quando termina l'esecuzione della funzione (scrittura).

Per questo motivo, tale trasmissione viene anche detta *copy-restore* oppure *copy-in-copy-out*.

Trasmissione per Result: Motivazione

La trasmissione per Result risponde al principio del **privilegio minimo**: dichiarare ogni elemento usato esclusivamente per l'uso necessario.

In questo senso, un parametro usato solo in scrittura va segnalato come parametro di output.

Trasmissione per Result: Motivazione

La trasmissione per Result risponde al principio del **privilegio minimo**: dichiarare ogni elemento usato esclusivamente per l'uso necessario.

In questo senso, un parametro usato solo in scrittura va segnalato come parametro di output.

Inoltre, assumeremo nei nostri programmi che i parametri passati per Result siano davvero utilizzati solo in scrittura.

Trasmissione per Result: aumenta l'espressività?

Allo stato attuale di `Small21`:

- Nuova possibilità: restituire **due** valori al termine di una funzione

Trasmissione per Result: aumenta l'espressività?

Allo stato attuale di `Small21`:

- Nuova possibilità: restituire **due** valori al termine di una funzione
- Problema: utilizzando l'unico parametro disponibile come parametro *write-only*, la funzione è necessariamente priva di altri parametri

Trasmissione per Result: aumenta l'espressività?

Allo stato attuale di `Small21`:

- Nuova possibilità: restituire **due** valori al termine di una funzione
- Problema: utilizzando l'unico parametro disponibile come parametro *write-only*, la funzione è necessariamente priva di altri parametri
- Non è ancora possibile realizzare un programma come il seguente:

```
1   void minmax(int a, int b,
2               result int min, result int max) {
3       if (a <= b) {
4           min = a;
5           max = b;
6       }
7       else {
8           min = b;
9           max = a;
10      }
11  }
```

Trasmissione per Result: aumenta l'espressività?

In combinazione con la definizione di funzioni a più parametri:

- possibilità di restituire parametri multipli
- tali parametri possono essere di tipo arbitrario
- un programma come il precedente può essere eseguito.

Trasmissione per Result: aumenta l'espressività?

In combinazione con la definizione di funzioni a più parametri:

- possibilità di restituire parametri multipli
- tali parametri possono essere di tipo arbitrario
- un programma come il precedente può essere eseguito.

Grande aumento di espressività!

Trasmissione per Value-Result: aumenta l'espressività?

La trasmissione per Value-Result è attualmente simulabile tramite l'uso di parametri per Reference (effettuando manualmente una copia in entrata e in uscita).

In certi casi, è anche equivalente al Reference, eccetto che con fenomeni di *aliasing*.

Trasmissione per Value-Result: aumenta l'espressività?

La trasmissione per Value-Result è attualmente simulabile tramite l'uso di parametri per Reference (effettuando manualmente una copia in entrata e in uscita).

In certi casi, è anche equivalente al Reference, eccetto che con fenomeni di *aliasing*.

Figure: Le due esecuzioni seguenti danno risultati diversi.

```
int a = 1;

int fie(valueresult int a, valueresult int b) {
    if (b == a) {
        b = b+1;
    }
    return a+b;
}

fie(a,a);
```

(1) Value-result

```
int a = 1;

int fie(reference int a, reference int b) {
    if (b == a) {
        b = b+1;
    }
    return a+b;
}

fie(a,a);
```

(2) Reference

Trasmissione per Value-Result: Aumenta l'espressività?

Con questa estensione di `Small21` saremo capaci di eseguire un programma come questo:

```
Prog sum{
  int a = -1;
  int b = 0;
  int c = 4;
  bool sumfirst (valueresult int b) {
    if (b < 0) {
      return False;
    }
    if (b == 0) {
      b = 0;
      return True;
    }
    int c = b-1;
    bool a = sumfirst(c);
    b = b + c;
    return True;
  }
  bool a1 = sumfirst(a);
  bool b1 = sumfirst(b);
  bool c1 = sumfirst(c);
}
```

Trasmissione per Value-Result: Aumenta l'espressività?

Con questa estensione di `Small21` saremo capaci di eseguire un programma come questo:

```
Prog sum{
  int a = -1;
  int b = 0;
  int c = 4;
  bool sumfirst (valueresult int b) {
    if (b < 0) {
      return False;
    }
    if (b == 0) {
      b = 0;
      return True;
    }
    int c = b-1;
    bool a = sumfirst(c);
    b = b + c;
    return True;
  }
  bool a1 = sumfirst(a);
  bool b1 = sumfirst(b);
  bool c1 = sumfirst(c);
}
```

Nota: Come visto nel corso, il comportamento di funzioni come questa sarebbe da evitare in favore di un meccanismo di gestione delle eccezioni.

Trasmissione per Value-Result: Aumenta l'espressività?

Con questa estensione di `Small21` saremo capaci di eseguire programmi come i seguenti:

```
let p3 = Prog(
  "doubleP",
  Block(
    SeqD (
      Var( (Int, "a", N 2) ),
      Pcd (Void, "double",
        FP(ValRes, Int, "b"),
        BlockP(
          ED,
          Upd(Val "b", Plus(Val "b", Val "b"))
        )
      )
    ),
    UnL(
      Call(
        "double",
        AP(Val "a")
      )
    )
  )
);;
```

(3) funzione `double` che raddoppia l'argomento

```
let p10 = Prog(
  "Abs",
  Block(
    SeqD (
      Var( (Int, "a", N (-2)) ),
      Pcd (Void, "abs",
        FP(ValRes, Int, "b"),
        BlockP(
          ED,
          IfT (LT(Val "b", N 0),
            Upd(Val "b", Minus(N 0, Val "b"))
          )
        )
      )
    ),
    UnL(
      Call(
        "abs",
        AP(Val "a")
      )
    )
  )
);;
```

(4) funzione `abs` che trasforma l'argomento nel suo valore assoluto

Trasmissione per Value-Result: Motivazione

- Spesso più costosa rispetto al Reference, per i meccanismi di copia
- Non aggiunge molto all'espressività
- Protegge dai fenomeni di *aliasing*, mantenendo distinti parametri formali diversi anche se associati allo stesso parametro attuale

Il meccanismo di Epilogo

Con i metodi di trasmissione finora visti, al termine dell'esecuzione di una funzione non vi sono operazioni da eseguire prima di chiudere l'Activation Record.

Nel caso di Result e Value-Result, prima di restituire il controllo al chiamante la funzione esegue un Epilogo: ad ogni parametro Result o Value-Result viene assegnato il valore che il corrispondente parametro attuale ha al termine della funzione.

Il meccanismo di Epilogo: problemi

1. Come implementare l'epilogo in `AM21`?
2. Quando valutare le locazioni di memoria dei parametri attuali? Al momento della chiamata o alla chiusura?
3. In che ordine assegnare i valori ai parametri alla chiusura?

Il meccanismo di Epilogo: problemi

1. Come implementare l'epilogo in `AM21`?
   - La risposta dipende dalle due successive.
2. Quando valutare le locazioni di memoria dei parametri attuali? Al momento della chiamata o alla chiusura?
3. In che ordine assegnare i valori ai parametri alla chiusura?

Il meccanismo di Epilogo: problemi

1. Come implementare l'epilogo in `AM21`?
2. Quando valutare le locazioni di memoria dei parametri attuali? Al momento della chiamata o alla chiusura?
   - In questa estensione, valuteremo gli *l-values* degli argomenti al momento della chiamata.
3. In che ordine assegnare i valori ai parametri alla chiusura?

Il meccanismo di Epilogo: problemi

1. Come implementare l'epilogo in `AM21`?
2. Quando valutare le locazioni di memoria dei parametri attuali? Al momento della chiamata o alla chiusura?
3. In che ordine assegnare i valori ai parametri alla chiusura?
   - Se decidiamo di non ammettere argomenti con uno stesso *l-value* (controllabile con analisi statica), l'ordine è ininfluente.
   - Decidiamo di non considerare questo errore, e valutare i parametri in ordine inverso, in modo che il primo sia l'ultimo ad essere aggiornato.
   - Per questo, `epi` sarà implementato come una pila.

Sintassi concreta, Sintassi Astratta, vincoli posti

- Sintassi Concreta: una CFG per `Small21`

$$\ldots$$

$$\texttt{PPF} \rightarrow \epsilon \mid \texttt{ref} \mid \texttt{result} \mid \texttt{valueresult}$$

$$\ldots$$

Sintassi concreta, Sintassi Astratta, vincoli posti

- Sintassi Concreta: una CFG per `Small21`

  ...

  PPF $\rightarrow \epsilon \mid$ `ref` $\mid$ `result` $\mid$ `valueresult`

  ...

- Sintassi Astratta:

  ...

  PPF ::= `[value]` | `[ref]` | `[result]` | `[valueresult]`

  ...

Sintassi concreta, Sintassi Astratta, vincoli posti

- Sintassi Concreta: una CFG per `Small21`

  ...

  PPF $\rightarrow \epsilon \mid$ `ref` $\mid$ `result` $\mid$ `valueresult`

  ...

- Sintassi Astratta:

  ...

  PPF ::= `[value]` | `[ref]` | `[result]` | `[valueresult]`

  ...

- Vincoli contestuali
  - parametri soltanto di tipo `Simple`
  - come negli altri tipi di trasmissione, richiediamo che il parametri siano già inizializzati prima della chiamata

Sistema Y: Trasmissione dei parametri

Aggiorniamo la regola di trasmissione [Y36] relativa al passaggio per Reference: anche per un parametro passato per Result va calcolato un valore denotabile.

$$\text{[Y36']}\ \frac{\begin{array}{c}\texttt{fps = [fp] ppf t ide} \\ \texttt{ppf} \in \{\texttt{[ref]}, \texttt{[result]}\} \\ \texttt{aps} = \texttt{[ap]exp} \\ \mathtt{Y}_\rho|_0(\texttt{ide}) = \bot \\ \langle \texttt{exp}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{DY}} (\texttt{[mut]ta}, \mathtt{Y}_\rho) \\ \texttt{t} = \texttt{ta} \qquad \texttt{t} = \texttt{Simple}\end{array}}{\langle \texttt{fps} \lhd \texttt{aps}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} (\texttt{[void]}, \mathtt{Y}_\rho)}$$

$$\text{[E60']}\ \frac{\begin{array}{c}\texttt{fps = [fp] ppf t ide} \\ \texttt{ppf} \in \{\texttt{[ref]}, \texttt{[result]}\} \\ \texttt{aps} = \texttt{[ap]exp} \\ \mathtt{Y}_\rho|_0(\texttt{ide}) = \bot \\ \langle \texttt{exp}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{DY}} (\texttt{[terr]}, \mathtt{Y}_\rho)\end{array}}{\langle \texttt{fps} \lhd \texttt{aps}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} (\texttt{[terr]}, \mathtt{Y}_\rho)}$$

Modifiche analoghe per [E61], [E61.1], [E61.2].

Sistema Y: trasmissione dei parametri

Nel caso della trasmissione per Value-Result, l'espressione in aps ha un doppio significato: valore memorizzabile, valore denotabile. Controlliamoli entrambi introducendo la regola [Y36.1].

$$[\text{Y36.1}]\ \frac{\begin{array}{c}\texttt{fps = [fp] [valueresult] t ide} \\ \texttt{aps} = \texttt{[ap] exp} \\ \mathtt{Y}_\rho|_0(\texttt{ide}) = \bot \\ \langle \texttt{exp}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{DY}} (\texttt{[mut]ta}, \mathtt{Y}_\rho) \\ \langle \texttt{exp}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} (\texttt{ta}, \mathtt{Y}_\rho) \\ \texttt{t} = \texttt{ta} \qquad \texttt{t} = \texttt{Simple}\end{array}}{\langle \texttt{fps} \triangleleft \texttt{aps}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} (\texttt{[void]}, \mathtt{Y}_\rho)}$$

Nota: In base alle regole di inferenza attuali, il doppio controllo risulta ridondante.

Sistema Y: trasmissione dei parametri

Introduciamo i corrispondenti errori di tipo: [E60.1], [E60.2], [E61.3], [E61.4], [E61.5]

$$[\text{E60.1}]\ \frac{\begin{array}{c}\texttt{fps = [fp] [valueresult] t ide} \\ \ldots \\ \langle \texttt{exp}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{DY}} (\texttt{[terr]}, \mathtt{Y}_\rho)\end{array}}{\langle \texttt{fps} \triangleleft \texttt{aps}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} (\texttt{[terr]}, \mathtt{Y}_\rho)}$$

$$[\text{E60.2}]\ \frac{\begin{array}{c}\texttt{fps = [fp] [valueresult] t ide} \\ \ldots \\ \langle \texttt{exp}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{DY}} (\texttt{[mut]ta}, \mathtt{Y}_\rho) \\ \langle \texttt{exp}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} (\texttt{[terr]}, \mathtt{Y}_\rho)\end{array}}{\langle \texttt{fps} \triangleleft \texttt{aps}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} (\texttt{[terr]}, \mathtt{Y}_\rho)}$$

Sistema Y: trasmissione dei parametri

$$[\text{E61.3}]\ \frac{\begin{array}{c}\texttt{fps = [fp] [valueresult] t ide}\\ \ldots \\ \mathtt{Y}_\rho|_0(\texttt{ide}) \neq \bot\end{array}}{\langle \texttt{fps} \triangleleft \texttt{aps}, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} (\texttt{[terr]}, \mathtt{Y}_\rho)}$$

$$[\text{E61.4}]\ \frac{\begin{array}{c}\texttt{fps = [fp] [valueresult] t ide}\\ \ldots \\ \texttt{t} \neq \texttt{ta}\end{array}}{\langle \texttt{fps} \triangleleft \texttt{aps}, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} (\texttt{[terr]}, \mathtt{Y}_\rho)}$$

$$[\text{E61.5}]\ \frac{\begin{array}{c}\texttt{fps = [fp] [valueresult] t ide}\\ \ldots \\ \texttt{t} \notin \texttt{Simple}\end{array}}{\langle \texttt{fps} \triangleleft \texttt{aps}, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} (\texttt{[terr]}, \mathtt{Y}_\rho)}$$

Sistema TR1: Trasmissione dei parametri

Aggiungiamo le regole [S17.2, S17.3] alla semantica di trasmissione dei parametri.

$$[\text{S17.2}]\ \frac{\begin{array}{c}\texttt{fps} = \texttt{[fp]}\ \texttt{[result]}\ \texttt{t}\ \texttt{I} \\ \texttt{aps} = \texttt{[ap]}\ \texttt{exp} \\ \langle \texttt{exp}, (\Delta, \mu)\rangle \rightarrow_{\texttt{DEN}} \lfloor \texttt{[Mut]ta}, \texttt{loc}_{\texttt{ta}}, (\Delta_{\texttt{d}}, \mu_{\texttt{d}})\rfloor \\ \texttt{[I/([Mut]ta, loc}_{\texttt{ta}}\texttt{)]} = \texttt{bind1} \\ > \texttt{bind1}] = \texttt{epi}_{\texttt{r}} \\ \texttt{t} \in \texttt{Simple} \qquad \texttt{t} = \texttt{ta} \qquad \Delta_c|_0(\texttt{I}) = \bot \\ \triangleright(\mu_{\texttt{d}}, \texttt{t}, 1) = (\texttt{loc}_{\texttt{t}}, \mu_2) \\ [\texttt{I}/(\texttt{[Mut]}\ \texttt{t},\ \texttt{loc}_{\texttt{t}})] \otimes \Delta_{\texttt{c}} = \Delta^{\texttt{F}}_{\texttt{c}} \\ (\Delta^{\texttt{F}}_{\texttt{c}}, \mu_2) = \sigma_{\texttt{r}}\end{array}}{\langle \texttt{fps} \triangleleft \texttt{aps}, (\Delta, \Delta_{\texttt{c}}, \mu)\rangle \rightarrow_{\texttt{TR1}} (\sigma_{\texttt{r}}, \texttt{epi}_{\texttt{r}})}$$

Sistema TR1: Trasmissione dei parametri

Aggiungiamo le regole [S17.2, S17.3] alla semantica di trasmissione dei parametri.

$$[\text{S17.2}]\ \frac{\begin{array}{c}\texttt{fps} = \texttt{[fp]}\ \texttt{[result]}\ \texttt{t}\ \texttt{I} \\ \texttt{aps} = \texttt{[ap]}\ \texttt{exp} \\ \langle \texttt{exp}, (\Delta, \mu)\rangle \rightarrow_{\texttt{DEN}} \lfloor \texttt{[Mut]ta}, \texttt{loc}_{\texttt{ta}}, (\Delta_{\texttt{d}}, \mu_{\texttt{d}})\rfloor \\ \texttt{[I/([Mut]ta, loc}_{\texttt{ta}}\texttt{)]} = \texttt{bind1} \\ > \texttt{bind1}] = \texttt{epi}_{\texttt{r}} \\ \texttt{t} \in \texttt{Simple} \qquad \texttt{t} = \texttt{ta} \qquad \Delta_c|_0(\texttt{I}) = \bot \\ \triangleright(\mu_{\texttt{d}}, \texttt{t}, 1) = (\texttt{loc}_{\texttt{t}}, \mu_2) \\ \texttt{[I/([Mut] t, loc}_{\texttt{t}}\texttt{)]} \otimes \Delta_{\texttt{c}} = \Delta_{\texttt{c}}^{\texttt{F}} \\ (\Delta_{\texttt{c}}^{\texttt{F}}, \mu_2) = \sigma_{\texttt{r}}\end{array}}{\langle \texttt{fps} \triangleleft \texttt{aps}, (\Delta, \Delta_{\texttt{c}}, \mu)\rangle \rightarrow_{\texttt{TR1}} (\sigma_{\texttt{r}}, \texttt{epi}_{\texttt{r}})}$$

Sistema TR1: Trasmissione dei parametri

Aggiungiamo le regole [S17.2, S17.3] alla semantica di trasmissione dei parametri.

$$\text{[S17.3]}\ \frac{\begin{array}{c}\texttt{fps} = \texttt{[fp] [valueresult] t I} \\ \texttt{aps} = \texttt{[ap] exp} \\ \langle \texttt{exp}, (\Delta, \mu)\rangle \rightarrow_{\texttt{DEN}} \lfloor \texttt{[Mut]ta}, \texttt{loc}_{\texttt{ta}}, (\Delta_{\texttt{d}}, \mu_{\texttt{d}})\rfloor \\ \texttt{t} \in \texttt{Simple} \quad \texttt{t} = \texttt{ta} \\ \texttt{[I/([Mut]ta, loc}_{\texttt{ta}}\texttt{)]} = \texttt{bind1} \\ \texttt{>bind1]} = \texttt{epi}_{\texttt{r}} \\ \mu_{\texttt{d}}(\texttt{loc}_{\texttt{ta}}) = \texttt{va} \\ \triangleright(\mu_{\texttt{d}}, \texttt{t}, 1) = (\texttt{loc}_{\texttt{t}}, \mu_2) \\ \mu_2[\texttt{loc}_{\texttt{t}} \leftarrow \texttt{va}] = \mu_3 \\ \Delta_c|_0(\texttt{I}) = \bot \\ [\texttt{I/([Mut] t, loc}_{\texttt{t}}\texttt{)}] \otimes \Delta_{\texttt{c}} = \Delta_{\texttt{c}}^{\texttt{F}} \\ (\Delta_{\texttt{c}}^{\texttt{F}}, \mu_3) = \sigma_{\texttt{r}}\end{array}}{\langle \texttt{fps} \triangleleft \texttt{aps}, (\Delta, \Delta_{\texttt{c}}, \mu)\rangle \rightarrow_{\texttt{TR1}} (\sigma_{\texttt{r}}, \texttt{epi}_{\texttt{r}})}$$

Sistema TR1: Trasmissione dei parametri

Aggiungiamo le regole [S17.2, S17.3] alla semantica di trasmissione dei parametri.

$$\text{[S17.3]}\ \frac{\begin{array}{c}\texttt{fps} = \texttt{[fp]}\ \texttt{[valueresult]}\ \texttt{t}\ \texttt{I} \\ \texttt{aps} = \texttt{[ap]}\ \texttt{exp} \\ \langle \texttt{exp}, (\Delta, \mu)\rangle \rightarrow_{\texttt{DEN}} \lfloor \texttt{[Mut]ta}, \texttt{loc}_{\texttt{ta}}, (\Delta_{\texttt{d}}, \mu_{\texttt{d}})\rfloor \\ \texttt{t} \in \texttt{Simple} \qquad \texttt{t} = \texttt{ta} \\ \texttt{[I/([Mut]ta,loc}_{\texttt{ta}}\texttt{)]} = \texttt{bind1} \\ > \texttt{bind1}] = \texttt{epi}_{\texttt{r}} \\ \mu_{\texttt{d}}(\texttt{loc}_{\texttt{ta}}) = \texttt{va} \\ \triangleright(\mu_{\texttt{d}}, \texttt{t}, 1) = (\texttt{loc}_{\texttt{t}}, \mu_2) \\ \mu_2[\texttt{loc}_{\texttt{t}} \leftarrow \texttt{va}] = \mu_3 \\ \Delta_c|_0(\texttt{I}) = \bot \\ \texttt{[I/([Mut] t, loc}_{\texttt{t}}\texttt{)]} \otimes \Delta_{\texttt{c}} = \Delta_{\texttt{c}}^{\texttt{F}} \\ (\Delta_{\texttt{c}}^{\texttt{F}}, \mu_3) = \sigma_{\texttt{r}} \end{array}}{\langle \texttt{fps} \triangleleft \texttt{aps}, (\Delta, \Delta_{\texttt{c}}, \mu)\rangle \rightarrow_{\texttt{TR1}} (\sigma_{\texttt{r}}, \texttt{epi}_{\texttt{r}})}$$

Sistema TR1: Trasmissione dei parametri

Come realizzare l'epilogo nella semantica di Small21?

1. Durante la trasmissione, per ogni parametro attuale, si pone in cima allo stack $\texttt{epi}_\texttt{r}$ un binding che lega l'identificatore del corrispondente parametro formale alla locazione di memoria del parametro attuale.

Sistema TR1: Trasmissione dei parametri

Come realizzare l'epilogo nella semantica di Small21?

1. Durante la trasmissione, per ogni parametro attuale, si pone in cima allo stack $\texttt{epi}_\texttt{r}$ un binding che lega l'identificatore del corrispondente parametro formale alla locazione di memoria del parametro attuale.
2. Al termine, partendo dalla cima dello stack, gli identificatori legati nei binding vengono valutati e riassegnati alla locazione associata.

Sistema TR1: Trasmissione dei parametri

Come realizzare l'epilogo nella semantica di Small21?

1. Durante la trasmissione, per ogni parametro attuale, si pone in cima allo stack $\texttt{epi}_\texttt{r}$ un binding che lega l'identificatore del corrispondente parametro formale alla locazione di memoria del parametro attuale.
2. Al termine, partendo dalla cima dello stack, gli identificatori legati nei binding vengono valutati e riassegnati alla locazione associata.
3. Perché siano valutati senza errori, vi dev'essere stato assegnato un valore nel corpo della funzione.

Sistema TR1: Trasmissione dei parametri

Come realizzare l'epilogo nella semantica di Small21?

1. Durante la trasmissione, per ogni parametro attuale, si pone in cima allo stack $\texttt{epi}_\texttt{r}$ un binding che lega l'identificatore del corrispondente parametro formale alla locazione di memoria del parametro attuale.
2. Al termine, partendo dalla cima dello stack, gli identificatori legati nei binding vengono valutati e riassegnati alla locazione associata.
3. Perché siano valutati senza errori, vi dev'essere stato assegnato un valore nel corpo della funzione.
4. Si può controllare che ciò sia realizzato con un'analisi statica prima dell'esecuzione. Noi solleveremo un errore a run-time dato dal controllo `ResultAssignedCheck` e dal corrispondente errore `ResultAssignedCheck:Fail`.

Sistema $\mathrm{SEM}_{\mathrm{CMD}}$ e $\mathrm{SEM}_{\mathrm{EXP}}$: Regole [call] e [apply]

Uso del meccanismo di Epilogo

Lasciamo le regole [S14] ed [S15] quasi invariate, aggiungendo il controllo che $\mathtt{epi_r}$ sia vuoto, per mantenere i metodi di trasmissione precedenti.

$$[\text{S14'}]\ \frac{\begin{array}{c}\dots \\ \langle \mathtt{fps} \triangleleft \mathtt{aps}, (\Delta, \Delta_c, \mu)\rangle \rightarrow_{\mathtt{TR1}} (\sigma_r, \mathtt{epi_r}) \\ \mathtt{epi_r} = [] \\ \dots\end{array}}{\langle \mathtt{[call]\ ide\ aps}, \sigma\rangle \rightarrow (\mathtt{[void]}, (\Delta_2', \mu_3))}$$

$$[\text{S15'}]\ \frac{\begin{array}{c}\dots \\ \langle \mathtt{fps} \triangleleft \mathtt{aps}, (\Delta, \Delta_c, \mu)\rangle \rightarrow_{\mathtt{TR1}} (\sigma_r, \mathtt{epi_r}) \\ \mathtt{epi_r} = [] \\ \dots\end{array}}{\langle \mathtt{[apply]\ ide\ aps}, \sigma\rangle \rightarrow \lfloor \mathtt{t_r}, \mathtt{v_r}, (\Delta_2', \mu_3)\rfloor}$$

Sistema $\mathrm{SEM}_{\mathrm{CMD}}$ e $\mathrm{SEM}_{\mathrm{EXP}}$: Regole [call] e [apply]

Uso del meccanismo di Epilogo

Aggiungiamo la regola [S14.1] implementando il meccanismo di Epilogo nel costrutto [call].

$$
\text{[S14.1]}\ \frac{\begin{array}{c}\dots \\ \langle \mathtt{fps} \triangleleft \mathtt{aps}, (\Delta, \Delta_c, \mu)\rangle \rightarrow_{\mathtt{TR1}} (\sigma_r, \mathtt{epi}_r) \\ \mathtt{epi}_r = {>}\mathtt{bind1}] \\ \mathtt{bind1} = [\mathtt{I}/([\mathtt{Mut}]\mathtt{ta}, \mathtt{loc}_{\mathtt{ta}})] \\ \dots \\ (\mathtt{ar}_3 + \Delta_2, \mu_2) \rightarrow_{\mathtt{R}^*} (\mathtt{ar}'_3 + \Delta'_2, \mu_3) \\ \mathtt{ResultAssignedCheck} \\ (\mathtt{ar}'_3 + \Delta'_2, \mu_3) = \sigma' \\ \langle [\mathtt{val}]\ \mathtt{I}, \sigma'\rangle \rightarrow \lfloor \mathtt{ta}, \mathtt{v}_\mathtt{I}, \sigma' \rfloor \\ \mu_3[\mathtt{loc}_{\mathtt{ta}} \leftarrow \mathtt{v}_\mathtt{I}] = \mu_4 \end{array}}{\langle [\mathtt{call}]\ \ \mathtt{ide}\ \ \mathtt{aps}, \sigma\rangle \rightarrow ([\mathtt{void}], (\Delta'_2, \mu_4))}
$$

Si ottiene in maniera identica la regola [S15.1] per il costrutto [apply], a partire dall'attuale [S15].

Sistema SEM_{CMD} e SEM_{EXP}: Regole [call] e [apply]

Uso del meccanismo di Epilogo

Aggiungiamo la regola [S14.1] implementando il meccanismo di Epilogo nel costrutto [call].

$$\text{[S14.1]}\ \frac{\begin{array}{c}\ldots \\ \langle \texttt{fps} \triangleleft \texttt{aps}, (\Delta, \Delta_c, \mu)\rangle \rightarrow_{\texttt{TR1}} (\sigma_r, \texttt{epi}_r) \\ \texttt{epi}_r = {>}\texttt{bind1}] \\ \texttt{bind1} = [\texttt{I}/([\texttt{Mut}]\texttt{ta}, \texttt{loc}_{\texttt{ta}})] \\ \ldots \\ (\texttt{ar}_3 + \Delta_2, \mu_2) \rightarrow_{\texttt{R}^*} (\texttt{ar}'_3 + \Delta'_2, \mu_3) \\ \texttt{ResultAssignedCheck} \\ (\texttt{ar}'_3 + \Delta'_2, \mu_3) = \sigma' \\ \langle [\texttt{val}]\ \texttt{I}, \sigma'\rangle \rightarrow \lfloor \texttt{ta}, \texttt{v}_\texttt{I}, \sigma' \rfloor \\ \mu_3[\texttt{loc}_{\texttt{ta}} \leftarrow \texttt{v}_\texttt{I}] = \mu_4\end{array}}{\langle [\texttt{call}]\ \texttt{ide}\ \texttt{aps}, \sigma\rangle \rightarrow ([\texttt{void}], (\Delta'_2, \mu_4))}$$

Si ottiene in maniera identica la regola [S15.1] per il costrutto [apply], a partire dall'attuale [S15].

Sistema $\mathtt{SEM}_{\mathtt{CMD}}$ e $\mathtt{SEM}_{\mathtt{EXP}}$: Regole [call] e [apply]

Uso del meccanismo di Epilogo

Nel caso in cui a un parametro Result non sia stato ancora assegnato un valore al termine della funzione, l'interprete restituisce errore.

$$[\text{E46.1}]\ \frac{\begin{array}{c}\dots \\ (\mathtt{ar}_3 + \Delta_2, \mu_2) \rightarrow_{\mathtt{R}^*} (\mathtt{ar}'_3 + \Delta'_2, \mu_3) \\ \texttt{ResultAssignedCheck:Fail}\end{array}}{\langle \texttt{[call] ide aps}, \mathtt{Y}_\rho \rangle \rightarrow (\texttt{[terr]}, \mathtt{Y}_\rho)}$$

In modo identico si ottiene l'errore [E33.1] per [apply].

Implementazione: modifiche all'interprete di Small21

Sintassi astratta e concreta:

```
ppf =
  Value
  | Ref
  | Res    (*** Ale ***)
  | ValRes  (*** Ale ***)
  and
```

(5) Sintassi Astratta

```
toStringPPF = (function
    | Value -> "value"
    | Ref -> "ref"
    | Res -> "result"
    | ValRes -> "valueresult"
    )
```

(6) Sintassi Concreta

Implementazione: modifiche all'interprete di Small21

Sistema TR1:

```
| (FP(Res,t,ide),AP exp)
      when (isSimple t) && not(declared skc ide)
      -> (match dexpSem exp (sk,mu) with
            | (Mut ta,loca,(_,mud))
              when ysame t ta
              -> (let (loct,mu2) = allocate mud 1 in
                        let den = DVar(Mut t,loct) in
                        let skcF = bindS skc ide den in
                        let sgr = (skcF,mu2)
                          and
                        den2 = DVar(Mut ta, loca) in
                        let bind1 = (ide, den2) in
                        let epir = pushEpi (Epi []) bind1 in
                                          (sgr,epir))
            | _ -> raise(TypeErrorT("E61.1: Trasmission: Type not expected")))
  | (FP(Res,_,ide),AP _)
      when declared skc ide
      -> raise(TypeErrorT("E61: Trasmission: Type not expected"))
  | (FP(Res,t,_),AP _)
      -> raise(TypeErrorT("E61.2: Trasmission: Type not expected"))
```

(7) Result

Implementazione: modifiche all'interprete di Small21

Sistema TR1:

```
| (FP(ValRes,t,ide),AP exp)
      when (isSimple t) && not(declared skc ide)
      -> (match dexpSem exp (sk,mu) with
            | (Mut ta,loca,(skd,mud))
              when ysame t ta
              ->  (let va = getStore mud loca in
                   let (loct,mu2) = allocate mud 1 in
                   let den = DVar(Mut t,loct) in
                   let skcF = bindS skc ide den and
                   mu3 = upd mu2 loct va in
                   let sgr = (skcF,mu3)
                     and
                   den2 = DVar(Mut ta, loca) in
                     let bind1 = (ide, den2) in
                     let epir = pushEpi (Epi []) bind1 in
                                  (sgr,epir))
            | _ -> raise(TypeErrorT("E61.4: Trasmission: Type not expected")))
  | (FP(ValRes,_,ide),AP _)
      when declared skc ide
      -> raise(TypeErrorT("E61.3: Trasmission: Type not expected"))
  | (FP(ValRes,t,_),AP _)
      -> raise(TypeErrorT("E61.5: Trasmission: Type not expected"))
```

(8) Value-Result

Implementazione: modifiche all'interprete di Small21

Meccanismo di Epilogo: tipo, presentazione, operazioni

```
type epi = Epi of (ide * dval) list;;

let toStringEpi (Epi epilogue) =
  let rec endingAux (Epi epilogue) = (match epilogue with
                    | [] -> "]"
                    | bind1::[] ->
                      let (i, dv) = bind1 in
                      let ide = toStringI i and
                      den = toStringDval dv in
                        ide^"/"^den^"]"
                    | bind1::epiRest ->
                        let (i, dv) = bind1 in
                        let ide = toStringI i and
                        den = toStringDval dv in
                          ide^"/"^den ^ "; " ^ endingAux (Epi epiRest)
  ) in
  "[" ^ endingAux (Epi epilogue)
;;
```

Implementazione: modifiche all'interprete di Small21

Meccanismo di Epilogo: tipo, presentazione, operazioni

```
exception EmptyEpi;;

let emptyEpi () =
    Epi []
    ;;
let pushEpi (Epi epilogue) binding =
    Epi (binding::epilogue)
        ;;
let topEpi (Epi epilogue) = match epilogue with
    | [] -> raise EmptyEpi
    | binding::_ -> binding
    ;;
let popEpi (Epi epilogue) = match epilogue with
    | [] -> raise EmptyEpi
    | _::rest -> Epi rest
    ;;
let sizeEpi (Epi epilogue) =
    List.length epilogue
    ;;
```

(10) Implementazione epilogo

Implementazione: modifiche all'interprete di Small21

Sistema $\mathrm{SEM}_{\mathtt{STM}}$, regola `[call]`:

```
| Call (ide,aps)
   ->( match getS sk ide with
         | (* include check for E46 *)
       | DAbs(Abs(Void,aa),ClosT(_,_,fps,dcl,sts,k)) (* no overloading *)
          -> (let ar = mkAR5 (Name ide) ((sizeS sk)-k+1)
                                 (emptyEnv()) [] None in
              let skc = push sk ar in
              let (sgR,epiR) = tr1Fun fps aps sk skc mu in
                                 (* include check for E47 *)
              let (_,(ar2sk2,mu2)) = dclSem dcl sgR in
              let ar3sk2 = resetC ar2sk2 [UnL sts] in
              let (_,(sk4,mu4)) = nextCmd(ar3sk2,mu2) in
              (*** include check for E46.1 ***)
              let muF =
                 (let rec restoreArgs epiE skE muE =
                 (match epiE with
                 | Epi [] -> muE
                 | Epi ((ideI, DVar(_, locI))::epiRest) ->
                  (let (_, vI, _) =
                    (try expSem (Val ideI) (skE, muE) with
                    | _ -> raise ResultNotAssignedError)
                  in
                  let muE2 = upd muE locI (eTOm vI) in
                  restoreArgs (Epi epiRest) skE muE2)
                 | _ -> raise(SystemErrorS ("stmSem: Unexpected binding in epilogue", stm)))
                 in
               restoreArgs epiR sk4 mu4)
              in
              let sgF = (pop sk4,muF) in
              (Void,sgF))
       |  _ -> raise(TypeErrorS("E45: stmSem",stm)))
```

Implementazione: modifiche all'interprete di Small21

Sistema SEM_{STM}, regola [call]:

```
let muF =
  (let rec restoreArgs epiE skE muE =
  (match epiE with
  | Epi [] -> muE
  | Epi ((ideI, DVar(_, locI))::epiRest) ->
    (let (_, vI, _) =
      (try expSem (Val ideI) (skE, muE) with
      | _ -> raise ResultNotAssignedError)
    in
    let muE2 = upd muE locI (eTOm vI) in
    restoreArgs (Epi epiRest) skE muE2)
  | _ -> raise(SystemErrorS ("stmSem: Unexpected binding in epilogue", stm)))
  in
 restoreArgs epiR sk4 mu4)
in
let sgF = (pop sk4,muF) in
(Void,sgF))
```

(12) Implementazione epilogo

Implementazione: modifiche all'interprete di Small21

Sistema $\mathrm{SEM}_{\mathrm{EXP}}$, regola [apply]:

```
| Apply(ide,aps)
    ->(match getS sk ide with
          (* include check for E33 *)
        | DAbs(Abs(tr,aa),ClosT(_,_,fps,dcl,sts,k)) (* no overloading *)
          when (isSimple tr)
            -> (let ar = mkAR5 (Name ide) ((sizeS sk)-k+1)
                                (emptyEnv()) [] None in
                let skc = push sk ar in
                let (sgR,epiR) = tr1Fun fps aps sk skc mu in
                                 (* include check for E34 *)
                let (_,(ar2sk2,mu2)) = dclSem dcl sgR in
                let ar3sk2 = resetC ar2sk2 [UnL sts] in
                let (_,(sk4,mu4)) = nextCmd(ar3sk2,mu2) in
                (*** include check for E33.1 ***)
                let muF =
                 (let rec restoreArgs epiE skE muE =
                 (match epiE with
                 | Epi [] -> muE
                 | Epi ((ideI, DVar(_, locI))::epiRest) ->
                   (let (_, vI, _) =
                     (try expSem (Val ideI) (skE, muE) with
                      | _ -> raise ResultNotAssignedError)
                    in
                    let muE2 = upd muE locI (eTOm vI) in
                    restoreArgs (Epi epiRest) skE muE2)
                 | _ -> raise(SystemErrorE ("expSem: Unexpected binding in epilogue")))
                 in
                 restoreArgs epiR sk4 mu4)
                in
                let sgF = (pop sk4,muF) in
                (tr,getR sk4,sgF))
        | DAbs(Abs(tr,aa),ClosT(_,_,fps,dcl,sts,k))
            -> raise(TypeErrorE("E34.1: expSem",Apply(ide,aps)))
        |  _ -> raise(TypeErrorE("E32: expSem",Apply(ide,aps))))
```

Implementazione: modifiche all'interprete di Small21

Sistema SEM_{EXP}, regola [apply]:

```
let muF =
 (let rec restoreArgs epiE skE muE =
 (match epiE with
 | Epi [] -> muE
 | Epi ((ideI, DVar(_, locI))::epiRest) ->
   (let (_, vI, _) =
     (try expSem (Val ideI) (skE, muE) with
       | _ -> raise ResultNotAssignedError)
   in
   let muE2 = upd muE locI (eTOm vI) in
   restoreArgs (Epi epiRest) skE muE2)
 | _ -> raise(SystemErrorE ("expSem: Unexpected binding in epilogue")))
 in
 restoreArgs epiR sk4 mu4)
in
let sgF = (pop sk4,muF) in
(tr,getR sk4,sgF))
```

(14) Implementazione epilogo

Esecuzione degli esempi

Trasmissione per Result: programma corretto

```
let p1 = Prog(
  "test1",
  Block(
    SeqD (
      Var( (Int, "a", N 1) ),
      Pcd (Void, "fun1",
        FP(Res, Int, "b"),
        BlockP(
          ED,
          Upd(Val "b", N 2)
        )
      )
    ),
    UnL(
      Call(
        "fun1",
        AP(Val "a")
      )
    )
  )
);;
```

(15) Programma

Esecuzione degli esempi

Trasmissione per Result: programma corretto

```
# progSem p1;;

Stack:
>{test1,0,[fun1/([void::int],$fun1,[void::int],:fpar:,:cmd:,1$);
 a/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-1]


Stack:
>{fun1,1,[b/(Mint,L1)],:cmdNext:,[N]}
{test1,0,[fun1/([void::int],$fun1,[void::int],:fpar:,:cmd:,1$);
 a/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-1,L1<-2]


Stack:
>{test1,0,[fun1/([void::int],$fun1,[void::int],:fpar:,:cmd:,1$);
 a/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-2,L1<-2]


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

(16) Esecuzione

Esecuzione degli esempi

Trasmissione per Result: Errore 46.1

```
let p2 = Prog(
  "test2",
  Block(
    SeqD (
      Var( (Int, "a", N 1) ),
      Pcd (Void, "fun2",
        FP(Res, Int, "b"),
        BlockP(
          ED,
          ES
        )
      )
    ),
    UnL(
      Call(
        "fun2",
        AP(Val "a")
      )
    )
  )
);;
```

(17) Programma

Esecuzione degli esempi

Trasmissione per Result: Errore 46.1

```
# progSem p2;;

Stack:
>{test2,0,[fun2/([void::int],$fun2,[void::int],:fpar:,:cmd:,1$);
 a/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-1]


Stack:
>{fun2,1,[b/(Mint,L1)],:cmdNext:,[N]}
{test2,0,[fun2/([void::int],$fun2,[void::int],:fpar:,:cmd:,1$);
 a/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-1,L1<-Undef]

Exception: ResultNotAssignedError.
```

(18) Esecuzione

Esecuzione degli esempi

Trasmissione per Value-Result: funzione `double`

```
let p3 = Prog(
  "doubleP",
  Block(
    SeqD (
      Var( (Int, "a", N 2) ),
      Pcd (Void, "double",
        FP(ValRes, Int, "b"),
        BlockP(
          ED,
          Upd(Val "b", Plus(Val "b", Val "b"))
        )
      )
    ),
    UnL(
      Call(
        "double",
        AP(Val "a")
      )
    )
  )
);;
```

(19) Programma

Esecuzione degli esempi

Trasmissione per Value-Result: funzione `double`

```
# progSem p3;;

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


Stack:
>{double,1,[b/(Mint,L1)],:cmdNext:,[N]}
{doubleP,0,[double/([void::int],$double,[void::int],:fpar:,:cmd:,1$);
 a/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-2,L1<-4]


Stack:
>{doubleP,0,[double/([void::int],$double,[void::int],:fpar:,:cmd:,1$);
 a/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-4,L1<-4]


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

(20) Esecuzione

Esecuzione degli esempi

Trasmissione per Value-Result: programma corretto, nonostante al parametro non venga assegnato alcun valore nel corpo della funzione

```
let p4 = Prog(
  "double2",
  Block(
    SeqD (
      Var( (Int, "a", N 2) ),
      Pcd (Void, "double2",
        FP(ValRes, Int, "b"),
        BlockP(
          ED,
          ES
        )
      )
    ),
    UnL(
      Call(
        "double2",
        AP(Val "a")
      )
    )
  )
);;
```

(21) Programma

Esecuzione degli esempi

Trasmissione per Value-Result: programma corretto, nonostante al parametro non venga assegnato alcun valore nel corpo della funzione

```
# progSem p4;;

Stack:
>{double2,0,[double2/([void::int],$double2,[void::int],:fpar:,:cmd:,1$);
 a/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-2]


Stack:
>{double2,1,[b/(Mint,L1)],:cmdNext:,[N]}
{double2,0,[double2/([void::int],$double2,[void::int],:fpar:,:cmd:,1$);
 a/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-2,L1<-2]


Stack:
>{double2,0,[double2/([void::int],$double2,[void::int],:fpar:,:cmd:,1$);
 a/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-2,L1<-2]


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

(22) Esecuzione

Esecuzione degli esempi

Trasmissione per Value-Result: Esempio iniziale

```
Prog sum{
  int a = -1;
  int b = 0;
  int c = 4;
  bool sumfirst (valueresult int b) {
    if (b < 0) {
      return False;
    }
    if (b == 0) {
      b = 0;
      return True;
    }
    int c = b-1;
    bool a = sumfirst(c);
    b = b + c;
    return True;
  }
  bool a1 = sumfirst(a);
  bool b1 = sumfirst(b);
  bool c1 = sumfirst(c);
}
```

(23) Programma

Esecuzione degli esempi

Trasmissione per Value-Result: Esempio iniziale

```
Stack:
>{SumFirstN,0,[c1/(Mbool,L20);
 b1/(Mbool,L6);
 a1/(Mbool,L4);
 sumfirst/([bool::int],$sumfirst,[bool::int],:fpar:,:cmd:,1$);
 c/(Mint,L2);
 b/(Mint,L1);
 a/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<--1,L1<-0,L2<-10,L3<--1,L4<-false,L5<-0,L6<-true,L7<-10,L8<-6,L9<-6,L10<-3,L11<-3
,L12<-1,L13<-1,L14<-0,L15<-0,L16<-true,L17<-true,L18<-true,L19<-true,L20<-true]


Stack:
>{SumFirstN,0,[c1/(Mbool,L20);
 b1/(Mbool,L6);
 a1/(Mbool,L4);
 sumfirst/([bool::int],$sumfirst,[bool::int],:fpar:,:cmd:,1$);
 c/(Mint,L2);
 b/(Mint,L1);
 a/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<--1,L1<-0,L2<-10,L3<--1,L4<-false,L5<-0,L6<-true,L7<-10,L8<-6,L9<-6,L10<-3,L11<-3
,L12<-1,L13<-1,L14<-0,L15<-0,L16<-true,L17<-true,L18<-true,L19<-true,L20<-true]


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

(24) Esecuzione

Analisi errori

Trasmissione per Value-Result: Errore di trasmissione 60.1

```
let p5 = Prog(
  "doubleErr",
  Block(
    SeqD (
      Var( (Int, "a", N 2) ),
      Pcd (Void, "doubleNO",
        FP(ValRes, Int, "b"),
        BlockP(
          ED,
          ES
        )
      )
    ),
    UnL(
      Call(
        "doubleNO",
        AP(Minus(Val "a", N 1))
      )
    )
  )
);;
```

(25) Programma

Analisi errori

Trasmissione per Value-Result: Errore di trasmissione 60.1

```
# progSem p5;;

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

Exception: SystemErrorE "dexp: (a - 1) is Mutable? ".
```

(26) Esecuzione

Analisi errori

Trasmissione per Result: Lettura parametro Result non inizializzato

```
let p6 = Prog(
  "ReadResultParam",
  Block(
    SeqD(
      Var(Int, "a", N 0),
      Pcd (Void, "readResultError",
        FP(Res, Int, "b"),
        BlockP(
          ED,
          Upd(Val "b", Plus(Val "b", N 1))
        )
      )
    ),
    SeqC(
      UnL (Call("readResultError", AP(Val "a"))),
      UnL ES
    )
  )
)
```

(27) Programma

Analisi errori

Trasmissione per Result: Lettura parametro Result non inizializzato

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

Exception: UndefinedLoc ("getStore", Loc 1).
```

(28) Esecuzione

Analisi errori

Trasmissione per Value-Result: Errore 61.4

```
let p7 = Prog(
  "Err61.4",
  Block(
    SeqD (
      Var( (Bool, "a", B True) ),
      Pcd (Void, "err614fun",
        FP(ValRes, Int, "b"),
        BlockP(
          ED,
          Upd(Val "b", Plus(Val "b", N 1))
        )
      )
    ),
    UnL(
      Call(
        "err614fun",
        AP(Val "a")
      )
    )
  )
);;
```

(29) Programma

Analisi errori

Trasmissione per Value-Result: Errore 61.4

```
# progSem p7;;

Stack:
>{Err61.4,0,[err614fun/([void::int],$err614fun,[void::int],:fpar:,:cmd:,1$);
 a/(Mbool,L0)],:cmdNext:,[N]}
]
Store:
[L0<-true]

Exception: TypeErrorT "E61.4: Trasmission: Type not expected".
```

(30) Esecuzione

Analisi errori

Trasmissione per Value-Result: Errore 61.2

```
let p8 = Prog(
  "Err61.2",
  Block(
    SeqD (
      Array( Arr (Int, 1), "a"),
      Pcd (Void, "err612fun",
        FP(Res, Arr(Int, 1), "b"),
        BlockP(
          ED,
          Upd(Arrow1("b", N 0), (N 3))
        )
      )
    ),
    UnL(
      Call(
        "err612fun",
        AP(Val "a")
      )
    )
  )
);;
```

(31) Programma

Analisi errori

Trasmissione per Value-Result: Errore 61.2

```
# progSem p8;;
Exception: TypeErrorI ("E13: dclSem", "b").
```

(32) Esecuzione

Notiamo che l'errore 61.2 è intercettato dal controllo in [E13].


M. Gabbrielli, S. Martini, *Programming Languages: Principles and Paradigms.* eng. Undergraduate topics in computer science. London: Springer London, 2010. isbn: 9781848829138.


Marco Bellia, *Materiale delle lezioni del corso di LPL: Small21-Definizione7*, Settembre 2021.


Marco Bellia, *Materiale delle lezioni del corso di LPL: Laboratori 2,3,4,5,6,7*, Maggio 2021.


Marco Bellia, *Materiale delle lezioni del corso di LPL: Lezione 8*, Aprile 2021.


Michael L. Scott, *Programming Languages Pragmatics* Burlington: Morgan Kaufmann Publishers, 2009. isbn: 97801237451498.

Grazie per l'attenzione!