

Linguaggi di Programmazione con Laboratorio

Seminario di fine corso:

`switch`

e

`break`

Simmaco Di Lillo

Università di Pisa, Dipartimento di Matematica

4 luglio 2021

Il comando `case` è una specializzazione del comando `if` con più rami. La sua sintassi è

Il comando **case** è una specializzazione del comando **if** con più rami. La sua sintassi è

```
Case Exp of
  label1: C1;
  label2: C2;
  ...
  labelN: CN;
```

Il comando **case** è una specializzazione del comando `if` con più rami. La sua sintassi è

```
Case Exp of
  label1: C1;
  label2: C2;
  ...
  labelN: CN;
```

dove

Il comando **case** è una specializzazione del comando `if` con più rami. La sua sintassi è

```
Case Exp of
  label1: C1;
  label2: C2;
  ...
  labelN: CN;
```

dove

- `Exp` è un'espressione

Il comando **case** è una specializzazione del comando `if` con più rami. La sua sintassi è

```
Case Exp of
  label1: C1;
  label2: C2;
  ...
  labelN: CN;
```

dove

- `Exp` è un'espressione
- `label1`, ... , `labelN` sono etichette con valore compatibile a quello calcolato da `Exp`

Il comando **case** è una specializzazione del comando `if` con più rami. La sua sintassi è

```
Case Exp of
  label1: C1;
  label2: C2;
  ...
  labelN: CN;
```

dove

- `Exp` è un'espressione
- `label1`, ... , `labelN` sono etichette con valore compatibile a quello calcolato da `Exp`
- `C1`, ..., `CN` sono comandi

Il comando **case** è una specializzazione del comando `if` con più rami. La sua sintassi è

```
Case Exp of
  label1: C1;
  label2: C2;
  ...
  labelN: CN;
```

Il significato di tale comando è il seguente:

Il comando **case** è una specializzazione del comando `if` con più rami. La sua sintassi è

```
Case Exp of
  label1: C1;
  label2: C2;
  ...
  labelN: CN;
```

Il significato di tale comando è il seguente:

- Viene valutata l'espressione `Exp`

Il comando **case** è una specializzazione del comando **if** con più rami. La sua sintassi è

```
Case Exp of
  label1: C1;
  label2: C2;
  ...
  labelN: CN;
```

Il significato di tale comando è il seguente:

- Viene valutata l'espressione **Exp**
- Viene eseguito il comando presente nell'unico ramo la cui etichetta include il valore ottenuto

Alcuni linguaggi esibiscono differenze significative nel comando case.

Alcuni linguaggi esibiscono differenze significative nel comando `case`.

Ad esempio, in C il comando `switch` ha la seguente sintassi

Alcuni linguaggi esibiscono differenze significative nel comando case.

Ad esempio, in C il comando `switch` ha la seguente sintassi

```
Switch (Exp) corpo
```

Alcuni linguaggi esibiscono differenze significative nel comando case.

Ad esempio, in C il comando `switch` ha la seguente sintassi

```
Switch (Exp) corpo
```

dove corpo può essere un qualunque comando.

Alcuni linguaggi esibiscono differenze significative nel comando case.

Ad esempio, in C il comando `switch` ha la seguente sintassi

```
Switch (Exp) corpo
```

dove `corpo` può essere un qualunque comando.

In generale il `corpo` è costituito da un blocco nel quale alcuni comandi sono etichettati, ovvero sono della forma

Alcuni linguaggi esibiscono differenze significative nel comando case.

Ad esempio, in C il comando `switch` ha la seguente sintassi

```
Switch (Exp) corpo
```

dove `corpo` può essere un qualunque comando.

In generale il `corpo` è costituito da un blocco nel quale alcuni comandi sono etichettati, ovvero sono della forma

```
case label : comando
```

Alcuni linguaggi esibiscono differenze significative nel comando case.

Ad esempio, in C il comando `switch` ha la seguente sintassi

```
Switch (Exp) corpo
```

dove `corpo` può essere un qualunque comando.

In generale il `corpo` è costituito da un blocco nel quale alcuni comandi sono etichettati, ovvero sono della forma

```
case label : comando
```

oppure

Alcuni linguaggi esibiscono differenze significative nel comando case.

Ad esempio, in C il comando `switch` ha la seguente sintassi

```
Switch (Exp) corpo
```

dove corpo può essere un qualunque comando.

In generale il corpo è costituito da un blocco nel quale alcuni comandi sono etichettati, ovvero sono della forma

```
case label : comando
```

oppure

```
default : comando
```

Al momento dell'esecuzione del comando:

Al momento dell'esecuzione del comando:

- l'espressione `Exp` viene valutata,

Al momento dell'esecuzione del comando:

- l'espressione `Exp` viene valutata,
- il controllo è trasferito al comando la cui etichetta coincide con il valore di `Exp`.

Al momento dell'esecuzione del comando:

- l'espressione `Exp` viene valutata,
- il controllo è trasferito al comando la cui etichetta coincide con il valore di `Exp`.
- Se non ci sono etichette con tale valore, viene eseguito il comando etichettato con `default` (se presente) e poi il controllo passa alla prima istruzione che segue lo `switch`.

Al momento dell'esecuzione del comando:

- l'espressione `Exp` viene valutata,
- il controllo è trasferito al comando la cui etichetta coincide con il valore di `Exp`.
- Se non ci sono etichette con tale valore, viene eseguito il comando etichettato con `default` (se presente) e poi il controllo passa alla prima istruzione che segue lo `switch`.

Si osservi che, una volta che è stato selezionato un ramo-case dello `switch`, il controllo fluisce ai comandi successivi senza alcuna valutazione sulle etichette

Switch e case a confronto

```
program case ;
var
    n , s : integer ;
begin
    n:=3;
    s:=0
    case n of
        3: begin
            s:=s+3;
            end ;
        2: begin
            s:=s+2;
            end ;
        1: begin
            s:=s+1;
            end ;
    end ;
end .
```

Switch e case a confronto

```
program case;
var
  n, s: integer;
begin
  n:=3;
  s:=0
  case n of
    3: begin
        s:=s+3;
      end;
    2: begin
        s:=s+2;
      end;
    1: begin
        s:=s+1;
      end;
  end;
end.
```

```
int main(){
  int n=3, s=0 ;
  switch (n)
  {
    case 3 : s = s + 3 ;
    case 2 : s = s + 2 ;
    case 1 : s = s + 1 ;
  }
}
```

Vediamo come possiamo simulare il comportamento del case usando lo `switch` e un nuovo comando `break`.

Vediamo come possiamo simulare il comportamento del case usando lo `switch` e un nuovo comando `break`.

Il comando `break` provoca l'interruzione dell'istruzione `switch` più interna racchiusa tra graffe; il controllo passa all'istruzione che segue l'istruzione interrotta.

Switch e case a confronto

Siamo ora pronti ad emulare il case usando switch e break

```
program case;  
var  
    n, s: integer;  
begin  
    n:=3;  
    s:=0  
    case n of  
        3: begin  
            s:=s+3;  
            end;  
        2: begin  
            s:=s+2;  
            end;  
        1: begin  
            s:=s+1;  
            end;  
    end;  
end.
```

```
int main(){  
    int n=3, s=0 ;  
    switch (n)  
    {  
        case 3 : s = s + 3 ;  
        break ;  
        case 2 : s = s + 2 ;  
        break ;  
        case 1 : s = s + 1 ;  
        break ;  
    }  
}
```

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

`Stm` \rightarrow ... | `switch (Exp) Stm`

`OtherStm` \rightarrow ... | `break;`

| `case Exp : Stm`

| `default : Stm`

- **Sintassi Astratta in ocaml**

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

```
stm =  
  ...  
  | Case   of exp * stm  
  | Switch of exp * stm  
  | Break  
  | Default of stm  
  ...
```

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

```
stm =  
  ...  
  | Case   of exp * stm  
  | Switch of exp * stm  
  | Break  
  | Default of stm  
  ...
```

Inoltre occorre modificare

```
type head = Name of ide | Exp | Cmd | NoneH | IPB | SW of tye *  
↪ aval | S of tye ;;
```

Introduciamo le regole per il sistema dei tipi λ dei nuovi costrutti

Introduciamo le regole per il sistema dei tipi \mathcal{Y} dei nuovi costrutti

- Case

$$[\mathcal{Y}001] \frac{\begin{array}{l} \langle e, Y_\rho \rangle \rightarrow \langle t, Y_\rho \rangle \quad t \in \text{Simple} \\ \langle \text{stm}, Y_\rho \rangle \rightarrow_{\mathcal{Y}} \langle [\text{void}], Y'_\rho \rangle \end{array}}{\langle [\text{case}] \text{ e } \text{stm}, Y_\rho \rangle \rightarrow_{\mathcal{Y}} \langle [\text{void}], Y'_\rho \rangle}$$

Introduciamo le regole per il sistema dei tipi Υ dei nuovi costrutti

- Case con la relativa regola per la gestione degli errori

$$[E001] \frac{\langle e, Y_\rho \rangle \rightarrow_Y \langle t, Y_\rho \rangle \quad t \notin \text{Simple}}{\langle [\text{case}] e \text{ stm}, Y_\rho \rangle \rightarrow_Y \langle [\text{terr}], Y_\rho \rangle}$$

$$[E002] \frac{\begin{array}{l} \langle e, Y_\rho \rangle \rightarrow_Y \langle t, Y_\rho \rangle \quad t \in \text{Simple} \\ \langle \text{stm}, Y_\rho \rangle \rightarrow_Y \langle [\text{terr}], Y'_\rho \rangle \end{array}}{\langle [\text{case}] e \text{ stm}, Y_\rho \rangle \rightarrow_Y \langle [\text{terr}], Y'_\rho \rangle}$$

Introduciamo le regole per il sistema dei tipi Y dei nuovi costrutti

- switch

$$[Y002] \frac{\begin{array}{l} \langle e, Y_\rho \rangle \rightarrow_Y \langle t, Y_\rho \rangle \quad t \in \text{Simple} \\ \langle \text{stm}, Y_\rho \rangle \rightarrow_Y \langle [\text{void}], Y'_\rho \rangle \end{array}}{\langle [\text{switch}] e \text{ stm}, Y_\rho \rangle \rightarrow_Y \langle [\text{void}], Y'_\rho \rangle}$$

Introduciamo le regole per il sistema dei tipi \mathcal{Y} dei nuovi costrutti

- `switch` con le relative regole per la gestione degli errori

$$[\text{E003}] \frac{\langle e, Y_\rho \rangle \rightarrow_{\mathcal{Y}} \langle t, Y_\rho \rangle \quad t \notin \text{Simple}}{\langle [\text{switch}] e \text{ stm}, Y_\rho \rangle \rightarrow_{\mathcal{Y}} \langle [\text{terr}], Y_\rho \rangle}$$

$$[\text{E004}] \frac{\begin{array}{l} \langle e, Y_\rho \rangle \rightarrow_{\mathcal{Y}} \langle t, Y_\rho \rangle \quad t \in \text{Simple} \\ \langle \text{stm}, Y_\rho \rangle \rightarrow_{\mathcal{Y}} \langle [\text{terr}], Y'_\rho \rangle \end{array}}{\langle [\text{switch}] e \text{ stm}, Y_\rho \rangle \rightarrow_{\mathcal{Y}} \langle [\text{terr}], Y'_\rho \rangle}$$

Introduciamo le regole per il sistema dei tipi \mathcal{Y} dei nuovi costrutti

- default

$$[\mathcal{Y}003] \frac{\langle s, Y_\rho \rangle \rightarrow_{\mathcal{Y}} \langle [\text{void}], Y'_\rho \rangle}{\langle [\text{default}] s, Y_\rho \rangle \rightarrow_{\mathcal{Y}} \langle [\text{void}], Y'_\rho \rangle}$$

Introduciamo le regole per il sistema dei tipi Υ dei nuovi costrutti

- `default` con la relativa regola per la gestione degli errori

$$[E005] \frac{\langle s, Y_\rho \rangle \rightarrow_Y \langle [\mathbf{terr}], Y'_\rho \rangle}{\langle [\mathbf{default}] s, Y_\rho \rangle \rightarrow_Y \langle [\mathbf{terr}], Y'_\rho \rangle}$$

Introduciamo le regole per il sistema dei tipi \mathcal{Y} dei nuovi costrutti

- break

$$[\mathcal{Y}004] \frac{}{\langle [\text{break}], \mathcal{Y}_\rho \rangle \rightarrow_{\mathcal{Y}} \langle [\text{void}], \mathcal{Y}_\rho \rangle}$$

Sostituiamo ad ogni regola data per SEM_{STM} ad eccezione della regola $S13$ (BlockS) e $S19$ (SeqS) due nuove regole

Mostriamolo per la regola [S2]

Mostriamolo per la regola [S2]

$$[S2] \frac{\begin{array}{l} \langle e, \sigma \rangle \rightarrow \llbracket \text{bool} \rrbracket, \text{true}, \sigma_e \rrbracket \\ \langle c, \sigma_e \rangle \rightarrow \llbracket \text{void} \rrbracket, \sigma_1 \rrbracket \end{array}}{\langle \llbracket \text{ifT} \rrbracket e c, \sigma \rangle \rightarrow \llbracket \text{void} \rrbracket, \sigma_1 \rrbracket}$$

Mostriamolo per la regola [S2]

$$[S2] \frac{\begin{array}{l} \langle e, \sigma \rangle \rightarrow \llbracket \text{bool} \rrbracket, \text{true}, \sigma_e \\ \langle c, \sigma_e \rangle \rightarrow \llbracket \text{void} \rrbracket, \sigma_1 \end{array}}{\langle \text{ifT } e \text{ c}, \sigma \rangle \rightarrow \llbracket \text{void} \rrbracket, \sigma_1}$$

viene sostituita dalle due regole

Mostriamolo per la regola [S2]

$$[S2] \frac{\langle e, \sigma \rangle \rightarrow \llbracket [\text{bool}], \text{true}, \sigma_e \rrbracket \quad \langle c, \sigma_e \rangle \rightarrow (\llbracket \text{void} \rrbracket, \sigma_1)}{\langle [\text{ifT}] e c, \sigma \rangle \rightarrow (\llbracket \text{void} \rrbracket, \sigma_1)}$$

viene sostituita dalle due regole

$$[S2.1] \frac{\begin{array}{l} \sigma = (ar + \Delta, \mu) \quad ar = \{h, l, f, c, vv\} \\ h \neq [\text{SW}] t v \quad t \in \text{Simple} \\ \langle e, \sigma \rangle \rightarrow \llbracket [\text{bool}], \text{true}, \sigma_e \rrbracket \\ \langle c, \sigma_e \rangle \rightarrow (\llbracket \text{void} \rrbracket, \sigma_1) \end{array}}{\langle [\text{ifT}] e c, \sigma \rangle \rightarrow (\llbracket \text{void} \rrbracket, \sigma_1)}$$

$$[S2.2] \frac{\begin{array}{l} \sigma = (ar + \Delta, \mu) \quad ar = \{h, l, f, c, vv\} \\ h = [\text{SW}] t v \quad t \in \text{Simple} \\ \langle e, \sigma \rangle \rightarrow \llbracket [\text{bool}], \text{true}, \sigma_e \rrbracket \\ \langle c, \sigma_e \rangle \rightarrow (\llbracket \text{void} \rrbracket, \sigma_1) \end{array}}{\langle [\text{ifT}] e c, \sigma \rangle \rightarrow (\llbracket \text{void} \rrbracket, \sigma)}$$

$$\begin{array}{c}
 \sigma = (ar + \Delta, \mu) \quad ar = \{h, l, f, c, vv\} \\
 h \neq [SW] t v \quad t \in Simple \\
 s \neq [BlockS] d_1 s_1 \\
 [BlockS] emptyD s = s_2 \\
 \langle [switch] e s_2, \sigma \rangle \rightarrow \langle [void], \sigma_F \rangle \\
 \hline
 [S001A] \quad \langle [switch] e s, \sigma \rangle \rightarrow \langle [void], \sigma_F \rangle
 \end{array}$$

$$\begin{array}{l}
 \sigma = (ar + \Delta, \mu) \quad ar = \{h, l, f, c, vv\} \\
 h \neq [SW] \quad t_w \quad v_w \quad t_w \in Simple \\
 s = [BlockS] \quad d \quad s_1 \\
 \langle e, \sigma \rangle \rightarrow [t, v, \sigma_e] \quad t \in Simple \\
 \sigma_e = (\Delta_e, \mu_e) \\
 \{[SW] \quad t \quad v, 1, [], s_1, []\} = ar_{top} \\
 (ar_{top} + \Delta_e, \mu_e) \rightarrow_{R^*} (ar' + \Delta', \mu') \\
 (\Delta', \mu') = \sigma_F \\
 \hline
 [S001B] \quad \langle [switch] \quad e \quad s, \sigma \rangle \rightarrow ([void], \sigma_F)
 \end{array}$$

$$\begin{array}{l}
 \sigma = (ar + \Delta, \mu) \quad ar = \{h, l, f, c, vv\} \\
 h = [SW] t v \quad t \in Simple \\
 \langle e, \sigma \rangle \rightarrow [t_e, v_e, \sigma_e] \quad t_e \in Simple \\
 \langle s, \sigma_e \rangle \rightarrow \langle [Void], \sigma_F \rangle \\
 \hline
 [S001C] \quad \langle [switch] e s, \sigma \rangle \rightarrow \langle [void], \sigma \rangle
 \end{array}$$

$$\begin{array}{c}
 \sigma = (ar + \Delta, \mu) \quad ar = \{h, l, f, c, vv\} \\
 h = [S] t \quad t \in Simple \\
 \langle e, \sigma \rangle \rightarrow [t_e, v_e, \sigma_e] \quad t_e = t \\
 \langle s, \sigma_e \rangle \rightarrow \langle [void], \sigma_F \rangle \\
 \hline
 [S002A] \quad \langle [case] e s, \sigma \rangle \rightarrow \langle [void], \sigma_F \rangle
 \end{array}$$

$$\begin{array}{c}
 \sigma = (ar + \Delta, \mu)ar = \{h, l, f, c, vv\} \\
 h = [SW] t v \qquad \qquad \qquad t \in Simple \\
 \langle e, \sigma \rangle \rightarrow [t_e, v_e, (ar_e + ar_1 + \Delta_e, \mu_e)] \quad t_e = t \quad v_e = v \\
 \qquad \qquad \qquad ar_e = \{h_e, l_e, f_e, c_e, vv_e\} \\
 ar_1 = \{h_1, l_1, f_1, c_1, vv_1\} \quad h_1 \neq [SW] t_2 v_2 \\
 \qquad \qquad \qquad \{[S] t, l_e, f_e, c_e, vv_e\} = ar' \\
 \qquad \qquad \qquad (ar' + ar_1 + \Delta, \mu) = \sigma' \\
 \qquad \qquad \qquad \langle s, \sigma' \rangle \rightarrow \langle [void], \sigma_F \rangle \\
 \hline
 [S002B] \quad \langle [case] e s, \sigma \rangle \rightarrow \langle [void], \sigma_F \rangle
 \end{array}$$

$$\begin{array}{l}
 \sigma = (ar + \Delta, \mu)ar = \{h, l, f, c, vv\} \\
 h = [SW] t v \qquad \qquad \qquad t \in Simple \\
 \langle e, \sigma \rangle \rightarrow [t_e, v_e, (ar_e + ar_1 + \Delta_e, \mu_e)] \quad t_e = t \quad v_e = v \\
 \qquad \qquad \qquad ar_e = \{h_e, l_e, f_e, c_e, vv_e\} \\
 \qquad \qquad \qquad ar_1 = \{h_1, l_1, f_1, c_1, vv_1\} \\
 h_1 = [SW] t_2 v_2 \qquad \qquad \qquad t_2 = t_1 \\
 \qquad \qquad \qquad \{[S] t, l_e, f_e, c_e, vv_e\} = ar' \\
 \qquad \qquad \qquad k(t_e, v_e) = e_1 \\
 [case] e_1 \text{ emptyS} = s_1 \qquad [SeqS] s_1 \text{ cc} = cnt \\
 \qquad \qquad \qquad \{hh, ll, ff, cnt, vv\} = ar'_1 \\
 \qquad \qquad \qquad (ar' + ar'_1 + \Delta_e, \mu_e) = \sigma' \\
 \langle s, \sigma' \rangle \rightarrow \langle [void], \sigma_F \rangle \\
 \hline
 [S002C] \quad \langle [case] e s, \sigma \rangle \rightarrow \langle [void], \sigma_F \rangle
 \end{array}$$

$$\begin{array}{l}
 \sigma = (ar + \Delta, \mu)ar = \{h, l, f, c, vv\} \\
 h = [SW] t v \qquad \qquad \qquad t \in Simple \\
 \langle e, \sigma \rangle \rightarrow [t_e, v_e, (ar_e + ar_1 + \Delta_e, \mu_e)] \quad t_e = t \quad v_e = v \\
 \qquad \qquad \qquad ar_e = \{h_e, l_e, f_e, c_e, vv_e\} \\
 \qquad \qquad \qquad ar_1 = \{h_1, l_1, f_1, c_1, vv_1\} \\
 h_1 = [SW] t_2 v_2 \qquad \qquad \qquad t_2 = t_1 \\
 \qquad \qquad \qquad \{[S] t, l_e, f_e, c_e, vv_e\} = ar' \\
 \qquad \qquad \qquad k(t_e, v_e) = e_1 \\
 [case] e_1 \text{ emptyS} = s_1 \qquad [SeqS] s_1 \text{ cc} = cnt \\
 \qquad \qquad \qquad \{hh, ll, ff, cnt, vv\} = ar'_1 \\
 \qquad \qquad \qquad (ar' + ar'_1 + \Delta_e, \mu_e) = \sigma' \\
 [S002C] \frac{\langle s, \sigma' \rangle \rightarrow \langle [void], \sigma_F \rangle}{\langle [case] e s, \sigma \rangle \rightarrow \langle [void], \sigma_F \rangle}
 \end{array}$$

dove la funzione k

$$\begin{array}{l}
 k([Int], N) = [num] N \\
 k([Bool], [true]) = [true] \\
 k([Bool], [false]) = [false]
 \end{array}$$

$$\begin{array}{c}
 \sigma = (ar + \Delta, \mu) \quad ar = \{h, l, f, c, vv\} \\
 h = [SW] t v \quad t \in Simple \\
 \langle e, \sigma \rangle \rightarrow [t_e, v_e, \sigma_e] \quad t_e = t \quad v_e \neq v \\
 \langle s, \sigma_e \rangle \rightarrow \langle [void], \sigma_F \rangle \\
 \hline
 [S002D] \quad \langle [case] e s, \sigma \rangle \rightarrow \langle [void], \sigma_F \rangle
 \end{array}$$

Il default è stato implementato in maniera diversa rispetto al C.

Giunti allo statement `default s`

Il default è stato implementato in maniera diversa rispetto al C.

Giunti allo statement `default s`

- Se abbiamo già trovato un case corretto tale statement viene saltato ed è come se non ci fosse (Regola [S003A])

Il default è stato implementato in maniera diversa rispetto al C.

Giunti allo statement `default s`

- Se abbiamo già trovato un case corretto tale statement viene saltato ed è come se non ci fosse (Regola [S003A])
- Se non abbiamo trovato un case corretto viene eseguito il comando e lo `switch` precede come se avessimo trovato un case corretto (Regole [S003B] e [S004C])

Il default è stato implementato in maniera diversa rispetto al C.

Giunti allo statement `default s`

- Se abbiamo già trovato un case corretto tale statement viene saltato ed è come se non ci fosse (Regola [S003A])
- Se non abbiamo trovato un case corretto viene eseguito il comando e lo `switch` precede come se avessimo trovato un case corretto (Regole [S003B] e [S004C])

Per rispettare la semantica data all'inizio, controlleremo staticamente che nessun case statement segua un `default` statement.

$$[S003A] \frac{\begin{array}{l} \sigma = (ar + \Delta, \mu) \quad ar = \{h, l, f, c, vv\} \\ h = [S] t \quad t \in Simple \cup \{[void]\} \\ \langle s, \sigma \rangle \rightarrow \langle [void], \sigma_F \rangle \end{array}}{\langle [default] s, \sigma \rangle \rightarrow ([void], \sigma)}$$

Regole d'inferenza per SEM_{STM} - default

$$\begin{array}{l} \sigma = (ar + ar_1 + \Delta, \mu) \quad ar = \{h, l, f, c, vv\} \\ h = [SW] t v \quad t \in Simple \\ ar_1 = \{h_1, l_1, f_1, c_1, vv_1\} \quad h_1 \neq [SW] t_2 v_2 \\ \{[S] [Void], l, f, c, vv\} = ar' \\ (ar' + ar_1 + \Delta, \mu) = \sigma' \\ \langle s, \sigma' \rangle \rightarrow \langle [void], \sigma_F \rangle \\ \hline [S003B] \quad \langle [default] s, \sigma \rangle \rightarrow \langle [void], \sigma_F \rangle \end{array}$$

Regole d'inferenza per SEM_{STM} - default

$$\begin{array}{l} \sigma = (ar + ar_1 + \Delta, \mu) \quad ar = \{h, l, f, c, vv\} \\ \quad h = [SW] t v \quad \quad \quad t \in Simple \\ ar_1 = \{h_1, l_1, f_1, c_1, vv_1\} \quad h_1 = [SW] t_2 v_2 \quad t_2 = t \\ \quad \{[S] [Void], l, f, c, vv\} = ar' \\ \quad \quad defaultES = s_1 \\ \quad \quad [SeqS]s_1 c_1 = cnt \\ \quad \quad \{h_1, l_1, f_1, cnt, vv_1\} = ar'_1 \\ \quad \quad (ar' + ar'_1 + \Delta, \mu) = \sigma' \\ \quad \quad \langle s, \sigma' \rangle \rightarrow \langle [void], \sigma_F \rangle \\ [S003C] \frac{}{\langle [default] s, \sigma \rangle \rightarrow \langle [void], \sigma_F \rangle} \end{array}$$

Modifichiamo la regola [S13] aggiungendo il controllo sull'head dell'activation record

Modifichiamo la regola [S13] aggiungendo il controllo sull'head dell'activation record

$$\begin{array}{c}
 \sigma = (ar + \Delta, \mu) \\
 ar = \{h, l, f, c, vv\} \quad h \neq [SW] \quad t \quad v \\
 \{g(h), 1, [], [], []\} = ar_{top} \\
 ar_{top} + ar + \Delta = \Delta_1 \\
 (\Delta_1, \mu) = \sigma_1 \\
 \langle ds, \sigma_1 \rangle \rightarrow \langle [\text{void}], (ar_2 + \Delta_2, \mu_2) \rangle \\
 ar_2 = \{hh, ll, ff, cc, vvv\} \\
 \{hh, ll, ff, sts, vvv\} = ar_3 \\
 (ar_3 + \Delta_2, \mu_2) \rightarrow_{R^*} (ar'_3 + \Delta'_2, \mu_3) \\
 (\Delta'_2, \mu_3) = \sigma_F \\
 \text{[S13.1]} \frac{}{\langle [\text{blockS}] ds sts, \sigma \rangle \rightarrow \langle [\text{void}], \sigma_F \rangle}
 \end{array}$$

Modifichiamo la regola [S13] aggiungendo il controllo sull'head dell'activation record

$$\begin{array}{c}
 \sigma = (ar + \Delta, \mu) \\
 ar = \{h, l, f, c, vv\} \quad h \neq [SW] \ t \ v \\
 \{g(h), l, [], [], []\} = ar_{top} \\
 ar_{top} + ar + \Delta = \Delta_1 \\
 (\Delta_1, \mu) = \sigma_1 \\
 \langle ds, \sigma_1 \rangle \rightarrow \langle [void], (ar_2 + \Delta_2, \mu_2) \rangle \\
 ar_2 = \{hh, ll, ff, cc, vvv\} \\
 \{hh, ll, ff, sts, vvv\} = ar_3 \\
 (ar_3 + \Delta_2, \mu_2) \rightarrow_{R^*} (ar'_3 + \Delta'_2, \mu_3) \\
 (\Delta'_2, \mu_3) = \sigma_F \\
 \text{[S13.1]} \frac{}{\langle [blockS] \ ds \ sts, \sigma \rangle \rightarrow \langle [void], \sigma_F \rangle}
 \end{array}$$

dove la funzione g

$$\begin{aligned}
 g(\text{Ide}) &= [\text{IPB}] \\
 g([\text{IPB}]) &= [\text{IPB}] \\
 g([\text{SW}] \ t \ v) &= [\text{SW}] \ t \ v \\
 g([\text{S}] \ t) &= [\text{S}] \ t
 \end{aligned}$$

Aggiungiamo anche la regola

Aggiungiamo anche la regola

$$\begin{array}{c}
 \sigma = (\Delta, \mu) \\
 ar = \{h, l, f, c, vv\} \quad h = [SW] \ t \ v \\
 \{g(h), l, [], sts, []\} = ar_{top} \\
 (ar_{top} + \Delta, \mu) \rightarrow_{R^*} (ar' + \Delta', \mu') \\
 (\Delta', \mu') = \sigma_F \\
 \hline
 [S13.2] \quad \langle [blockS] \ ds \ sts, \sigma \rangle \rightarrow \langle [void], \sigma_F \rangle
 \end{array}$$

$$[S005A] \frac{\sigma = (\text{ar} + \text{ar}_1 + \Delta_r, \mu) \quad \text{ar} = \{\text{h}, \text{l}, \text{f}, \text{c}, \text{vv}\} \quad \text{h} = [\text{SW}] \text{t v} \quad t \in \text{Simple}}{\langle [\text{break}], \sigma \rangle \rightarrow \langle [\text{void}], \sigma \rangle}$$

$$\begin{array}{c}
 \sigma = (\text{ar} + \text{ar}_1 + \Delta_r, \mu) \\
 \text{ar} = \{\text{h}, \text{l}, \text{f}, \text{c}, \text{vv}\} \quad \text{h} = [\text{S}] \quad t \in \text{Simple} \cup \{\text{[Void]}\} \\
 \text{ar}_1 = \{\text{hh}, \text{ll}, \text{ff}, \text{cc}, \text{vv}\} \quad \text{hh} = \text{h} \\
 \{\text{h}, \text{l}, \text{f}, [], \text{vv}\} = \text{ar}' \\
 \{\text{hh}, \text{ll}, \text{ff}, [\text{Break}], \text{vv}\} = \text{ar}_1' \\
 (\text{ar}' + \text{ar}_1' + \Delta_r, \mu) = \sigma_F \\
 \hline
 \text{[S005B]} \quad \langle [\text{break}], \sigma \rangle \rightarrow \langle [\text{void}], \sigma_F \rangle
 \end{array}$$

$$\begin{array}{c}
 \sigma = (\mathbf{ar} + \mathbf{ar}_1 + \Delta_r, \mu) \\
 \mathbf{ar} = \{\mathbf{h}, \mathbf{l}, \mathbf{f}, \mathbf{c}, \mathbf{vv}\} \quad \mathbf{h} = [\mathbf{S}] t \quad t \in \mathit{Simple} \cup \{[\mathbf{Void}]\} \\
 \mathbf{ar}_1 = \{\mathbf{hh}, \mathbf{ll}, \mathbf{ff}, \mathbf{cc}, \mathbf{vv}\} \quad \mathbf{hh} \neq [\mathbf{S}] t_2 \quad t_2 \in \mathit{Simple} \\
 \{\mathbf{h}, \mathbf{l}, \mathbf{f}, [], \mathbf{vv}\} = \mathbf{ar}' \\
 (\mathbf{ar}' + \mathbf{ar}_1 + \Delta_r, \mu) = \sigma_F \\
 \hline
 \langle [\mathbf{break}], \sigma \rangle \rightarrow \langle [\mathbf{void}], \sigma_F \rangle
 \end{array}$$

[S005C]

Operazione per la presentazione in sintassi concreta dei nuovi statement

```
toStringStm tab = function
  | Switch (exp,stm) -> (let expString = (if isAtomic exp then "(" ^
  ↪ (toStringExp exp) ^ ")" else toStringExp exp) and stmString =
  ↪ (toStringStm 1 stm)
    in (indent tab) ^ "switch " ^ expString ^ stmString )
  | Case (exp,stm) -> (let expString= toStringExp exp and stmString =
  ↪ (toStringStm 1 stm)
    in (indent tab) ^ "case " ^ expString ^ " : " ^ stmString )
  | Default(stm) -> (indent tab)^"default : " ^ (toStringStm 1 stm)
  | Break -> (indent tab)^"break;"

...

```

Modifiche al codice "small21.ml"

Modifica della SEM_{STM} per introdurre switch

```
| Switch(e,s) ->
  ( match s with
    | SeqS(_,_) -> raise(StaticErrorS("Invalid use of SeqS",stm))
    | BlockS(d,s1) ->
      ( match (getH sk) with
        | SW(t,_) when (isSimple t) ->( (Void, (sk,mu)))
        | SW(a,b) -> raise(HeadNotValid(SW(a,b)))
        | _ ->
          ( match expSem e (sk,mu) with
            | (t,v,(ske,mue)) when (isSimple t) ->
              (let artop = mkAR5 (SW(t,v)) 1 (emptyEnv()) []
               ↪ None in
              let sk1 = push ske artop in
              let sk3 = resetC sk1 [UnL s1] in
              try(let(_, (sk4,mu4)) = nextCmd(sk3,mue) in
              let sgF = (pop sk4,mu4) in (Void,sgF))
              with | TypeErrorC("104:nextCmd",_) ->
               ↪ raise(TypeErrorS("E004:stmSem",stm)))
               | _
               ↪ ->raise(TypeErrorS("E003:stmSem",stm))))))
    ...
```

Modifica della SEM_{STM} per introdurre switch

```
| Switch(e,s) ->
  ( match s with
    ...
  | _ ->
    (let s2= BlockS( ED,s ) in
      ( match (getH sk) with
        | SW(t,_) when (isSimple t) ->
          let (te,ve,(ske,mue)) = expSem e (sk,mu) in
            (if (isSimple te) then
              ( if (ysame t te) then (Void,(sk,mu))
                else ( let msg = "expected"^(toStringTye
                  ↪ t)^ "expression" in
                  ↪ raise(StaticErrorS(msg,stm))))
              else raise(StaticErrorS("E003: stmSem", stm))))
        | SW(a,b) -> raise(HeadNotValid(SW(a,b)))
        | _ -> stmSem (Switch(e,s2)) (sk,mu)))
```

Modifica della SEM_{STM} per introdurre case

```
| Case(e,s) ->(match s with
  | SeqS(_,_) -> raise(StaticErrorS("Invalid use of SeqS",stm))
  | _ ->(match getH sk with
    | S t when (isSimple t) ->(match expSem e (sk,mu) with
      |(te,_,(ske,mue)) when (ysame t te) -> stmSem s (ske,mue)
      | _ -> (let msg = "expected"^(toStringTye t)^ "expression" in
        ↪ raise(StaticErrorS(msg,stm)))
    | S t -> raise(StaticErrorS("default statement before case
    ↪ statement", stm))
    ...
```

Modifiche al codice "small21.ml"

Modifica della SEM_{STM} per introdurre case

```
...
| SW(t,v)-> (match expSem e (sk,mu) with
  |(te,_,_) when not (isSimple te) -> raise(TypeErrorS("E001:
  ↪ stmSem",stm) )
  |(te,ve,(ske,mue)) when (ysame t te) -> (if (ve=v) then
    (let sk1=resetH ske (S t) in
     (match getH (pop sk1) with
      | SW(t1,v1) -> (let k = (function
        | (Int, Ival n) -> N n
        | (Bool, Bval b) -> B b
        | _-> raise(TypeErrorS("E001: stmSem",stm))) in
       let sk2 = pop sk1 in let ar = top sk1 in let e1 =
        ↪ k(te,ve) in let ss= Case(e1, ES) in let sk3 =
        ↪ addCode sk2 (UnL ss) in let sk4 =push sk3 ar in
        ↪ stmSem s (sk4,mue))
      | _ -> stmSem s (sk1,mue)))
    else ( match stmSem s (sk,mu) with
      | (Void,(ske,mue))-> (Void,(ske,mue))
      | _-> raise(TypeErrorS("E002: stmSem ",stm)))
      | _-> (let msg = "expected"^(toStringTye t)^
        ↪ "expression" in raise(StaticErrorS(msg,stm))))
  | _ -> raise(StaticErrorS("Invalid use of case",stm)))
```

Modifica della SEM_{STM} per introdurre default

```
|Default(s)-> (match s with
  |SeqS(_,_)>raise(StaticErrorS("Invalid use of SeqS",stm))
  |_> (match getH sk with
    | S t -> ( match stmSem s (sk,mu) with
      | (Void,_)-> (Void,(sk,mu))
      | _-> raise(TypeErrorS("E005: stmSem",stm)))
    | SW(_,_)> ( let sk1=resetH sk (S Void) in
      (match getH (pop sk1) with
        | SW(t1,v1) -> (let sk2 = pop sk1 in
          let ar = top sk1 in let sk3 = addCode sk2 (UnL
            ↪ Default(ES) in
          let sk4 =push sk3 ar in stmSem s (sk4,mu))
        | _ -> stmSem s (sk1,mu)))
    | _ ->raise(StaticErrorS("Invalid use of default",stm))))
```

Modifica della SEM_{STM} per introdurre break

```
| Break ->( match (getH sk) with
  | SW(t,v)-> (Void,(sk,mu))
  | S t  -> ( let ar1 = top(resetC sk []) in
    (match getH (pop sk) with
      | S(tt) when -> (let ar2 = top(resetC (pop sk) [UnL
        ↪ Break]) in let skr1= pop(pop sk) in let sgF = (push
        ↪ (push skr1 ar2) ar1,mu) in (Void,sgF))
      | _ -> (let skr1 = pop sk in let sgF = ((push skr1 ar1),mu)
        ↪ in (Void,sgF))))
  | _ -> raise(StaticErrors("Invalid use of break",stm)))
```

Modifiche al codice "small21.ml"

Modifica della SEM_{STM} per blockS

```
| BlockS (dcl,stm) -> (let g = (function
  |Name _ -> IPB      |IPB -> IPB      |SW(a,b)-> SW(a,b) |S t -> S t
  |_ -> NoneH) in
let h = getH sk in (match h with
  |SW(t,v)->( let artop = mkAR5 (g h) 1 (emptyEnv()) [] None in
  ↪ let sk1 = push sk artop in
    match dclSem ED (sk1,mu) with
      | (Void,(sk2,mu2)) ->(let sk3 = resetC sk2 [UnL stm] in
        try(let(_,sk4,mu4) = nextCmd(sk3,mu2) in let sgF =
          ↪ (pop sk4,mu4) in (Void,sgF))
        with | TypeErrorC("104:nextCmd",_) ->
          ↪ raise(TypeErrorS("E52: stmSem",stm))
           |_ -> raise(TypeErrorS("E51: stmSem",stm)))
  |_ ->(let artop = mkAR5 (g h) 1 (emptyEnv()) [] None in let sk1 =
  ↪ push sk artop in
    match dclSem dcl (sk1,mu) with
      | (Void,(sk2,mu2)) ->(let sk3 = resetC sk2 [UnL stm] in
        try(let(_,sk4,mu4) = nextCmd(sk3,mu2) in let sgF =
          ↪ (pop sk4,mu4) in Void,sgF))
        with | TypeErrorC("104:nextCmd",_) ->
          ↪ raise(TypeErrorS("E52: stmSem",stm))
           |_ -> raise(TypeErrorS("E51: stmSem",stm))))))
```

Modifica delle regole per tutti gli statement già presente ad eccezione di SeqS e BlockS.

Tale modifica serve affinché `stmSem stm` restituisca

$$(\text{Void}, (\text{sk}, \text{mu}))$$

se:

- `stm` è uno statement diverso da BlockS e non introdotto in questo seminario e

Modifica delle regole per tutti gli statement già presente ad eccezione di SeqS e BlockS.

Tale modifica serve affinché `stmSem stm` restituisca

$$(\text{Void}, (\text{sk}, \mu))$$

se:

- `stm` è uno statement diverso da BlockS e non introdotto in questo seminario e
- `getH sk` è della forma `SW(t, v)`

Esempio : Switch senza break

```
let d1 = Var(Int, "x", N 3);;
let d2= Var(Int, "s", N 0 );;
let c4 = Case( N 4, Upd ( Val "s", Plus( Val "s",
↳ N 4)));;
let c3 = Case( N 3, Upd ( Val "s", Plus( Val
↳ "s", N 3)));;
let c2 =Case( N 2, Upd ( Val "s", Plus( Val "s",
↳ N 2)));;
let c1=Case( N 1, Upd ( Val "s", Plus( Val "s", N
↳ 1)));;
let c = BlockS(
↳ ED, SeqS(c4, SeqS(c3, SeqS(c2, c1))));;
let a1 = Switch( Val "x", c) ;;
let p1 = Prog("senzaBreak", Block (SeqD( d1, d2),
↳ UnL a1));;

printProg p1;;
progSem p1;;
```

```
Program senzaBreak{
    int x = 3;
    int s = 0;

    switch (x)    {
        case : 4 s = (s + 4);
        case : 3 s = (s + 3);
        case : 2 s = (s + 2);
        case : 1 s = (s + 1);
    }    }

- : unit/2 = ()

Stack:
>{senzaBreak, 0, [s/(Mint, L1);
  x/(Mint, L0)], :cmdNext:, [N]}
]
Store:
[L0<-3, L1<-6]

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

Esempio : Switch con break

```
let d1 = Var(Int, "x", N 3 );;  
let d2= Var(Int, "s", N 0 );;  
let c4 = Case( N 4,Upd ( Val "s", Plus( Val "s",  
↳ N 4))));;  
let c3 = Case( N 3,Upd ( Val "s", Plus( Val "s",  
↳ N 3))));;  
let c2 =Case( N 2,Upd ( Val "s", Plus( Val "s",  
↳ N 2))));;  
let c1=Case( N 1, Upd ( Val "s", Plus( Val "s",  
↳ N 1))));;  
let c=BlockS(ED,SeqS(c4,SeqS(Break,SeqS(c3,  
SeqS(Break,SeqS(c2,SeqS(Break,c1))))))));;  
let a1 = Switch( Val "x", c) ;;  
let p2=Prog("conBreak", Block (SeqD( d1,d2), UnL  
↳ a1));;  
  
printProg p2;;  
progSem p2;;
```

```
Program conBreak{  
  int x = 3;  
  int s = 0;  
  
  switch (x)    {  
  
    case : 4 s = (s + 4);  
    break;  
    case : 3 s = (s + 3);  
    break;  
    case : 2 s = (s + 2);  
    break;  
    case : 1 s = (s + 1);  
  
  }    }  
  
- : unit/2 = ()
```

```
Stack:  
>{conBreak,0,[s/(Mint,L1);  
x/(Mint,L0)],:cmdNext:[N]}  
]  
Store:  
[L0<-3,L1<-3]
```

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

Esempio : Caso non trovato

```
let d1 = Var(Int, "x", N 5 );;  
let d2= Var(Int, "s", N 0 );;  
let c4 = Case( N 4,Upd ( Val "s", Plus( Val "s",  
↳ N 4))));;  
let c3 = Case( N 3,Upd ( Val "s", Plus( Val "s",  
↳ N 3))));;  
let c2 =Case( N 2,Upd ( Val "s", Plus( Val "s",  
↳ N 2))));;  
let c1=Case( N 1, Upd ( Val "s", Plus( Val "s",  
↳ N 1))));;  
let c= BlockS(ED,SeqS(c4,SeqS(Break,SeqS(c3,  
SeqS(Break,SeqS(c2,SeqS(Break,c1))))))));;  
let a1 = Switch( Val "x", c) ;;  
let p5=Prog("caseNotFound", Block (SeqD( d1,d2),  
↳ UnL a1));;  
  
printProg p5;;  
progSem p5;;
```

```
Program caseNotFound{  
  int x = 5;  
  int s = 0;  
  
  switch (x)    {  
  
    case : 4 s = (s + 4);  
    break;  
    case : 3 s = (s + 3);  
    break;  
    case : 2 s = (s + 2);  
    break;  
    case : 1 s = (s + 1);  
  
  }    }
```

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

Stack:

```
>{caseNotFound,0,[s/(Mint,L1);  
x/(Mint,L0)],:cmdNext:[N]}  
]
```

Store:

```
[L0<-5,L1<-0]
```

SUCCESSFUL_TERMINATION

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

Esempio : default

```
let d1 = Var(Int, "x", N 5 );;  
let d2= Var(Int, "s", N 0 );;  
let c4 = Case( N 4,Upd ( Val "s", Plus( Val "s",  
↳ N 4)));;  
let c3 = Case( N 3,Upd ( Val "s", Plus( Val "s",  
↳ N 3)));;  
let c2 =Case( N 2,Upd ( Val "s", Plus( Val "s",  
↳ N 2)));;  
let c1=Case( N 1, Upd ( Val "s", Plus( Val "s",  
↳ N 1)));;  
let c0=Default(Upd(Val "s", N 1000));;  
let c=  
↳ BlockS(ED,SeqS(c4,SeqS(c3,SeqS(c2,SeqS(c1,c0)))));  
let a1 = Switch( Val "x", c) ;;  
let def1=Prog("default1", Block (SeqD( d1,d2),  
↳ UnL a1));;  
printProg def1;;  
progSem def1;;
```

```
Program default1{  
  int x = 5;  
  int s = 0;  
  
  switch (x)    {  
  
    case : 4 s = (s + 4);  
    case : 3 s = (s + 3);  
    case : 2 s = (s + 2);  
    case : 1 s = (s + 1);  
    default : s = 1000;  
  
  }    }  
  
- : unit/2 = ()  
  
Stack:  
>{default1,0,[s/(Mint,L1);  
  x/(Mint,L0)],:cmdNext:[N]}  
]  
Store:  
[L0<-5,L1<-1000]
```

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

Esempio : dichiarazioni in blocchi switch (1/2)

```
let d1 = Var(Int, "x", N 4);;
let d2= Var(Int, "s", N 10 );;
let d3 = Var(Int, "z", N 37) ;;
let c1= Case( N 1, Case(N 2,Upd ( Val "s", Plus(
↳ Val "s", N 5)))));;
let c2= Case( N 3, Case(N 4,Upd ( Val "s", Val
↳ "z")));;
let c = Switch(Val "x", BlockS( d3,
↳ SeqS(c1,c2))));;
let p15= Prog("break7",Block ( SeqD(d1,d2), UnL
↳ c));;

printProg p15;;
progSem p15;;
```

```
Program break7{
    int x = 4;
    int s = 10;

    switch (x)    {
        int z = 37;
        case : 1 case : 2 s = (s + 5);
        case : 3 case : 4 s = z;

    }    }

- : unit/2 = ()

Exception: UnboundIde ("staticGetS", "z").
```

Esempio : dichiarazioni in blocchi switch (2/2)

```
let d1 = Var(Int, "x", N 4);;
let d2= Var(Int, "s", N 10 );;
let d3 = Var(Int, "z", N 37) ;;
let c1= Case( N 1, Case(N 2,Upd ( Val "s", Plus(
↳ Val "s", N 5)))));;
let c2= Case( N 3, Case(N 4,BlockS(d3,Upd ( Val
↳ "s", Val "z"))));;
let c = Switch(Val "x", BlockS( d3,
↳ SeqS(c1,c2))));;
let p16= Prog("break7.1",Block ( SeqD(d1,d2),
↳ UnL c));;

printProg p16;;
progSem p16;;
```

```
Program break7.1{
  int x = 4;
  int s = 10;

  switch (x)    {
    int z = 37;
    case : 1 case : 2 s = (s + 5);
    case : 3 case : 4    {
      int z = 37;
      s = z;
    }
  }

- : unit/2 = ()
```

```
Stack:
>{break7.1,0,[s/(Mint,L1);
  x/(Mint,L0)],:cmdNext:, [N]}
]
Store:
[L0<-4,L1<-37,L2<-37]
```

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

Esempio : case dopo un default

```
let d1 = Var(Int, "x", N 4 );;
let d2= Var(Int, "s", N 0 );;
let c4 = Case( N 4,Upd ( Val "s", Plus( Val "s",
↳ N 4)));;
let c3 = Case( N 3,Upd ( Val "s", Plus( Val "s",
↳ N 3)));;
let c2 =Case( N 2,Upd ( Val "s", Plus( Val "s",
↳ N 2)));;
let c1=Case( N 1, Upd ( Val "s", Plus( Val "s",
↳ N 1)));;
let c0=Default(Upd(Val "s", N 1000));;
let c=
↳ BlockS(ED,SeqS(c0,SeqS(c3,SeqS(c2,SeqS(c1,c4)))));;
let a1 = Switch( Val "x", c) ;;
let p5=Prog("default2", Block (SeqD( d1,d2), UnL
↳ a1));;

printProg p5;;
progSem p5;;
```

```
Program default2{
    int x = 4;
    int s = 0;

    switch (x)    {

        default : s = 1000;
        case : 3 s = (s + 3);
        case : 2 s = (s + 2);
        case : 1 s = (s + 1);
        case : 4 s = (s + 4);

    }    }

- : unit/2 = ()
```

```
Exception:
StaticErrorS ("default statement before case
statement",
    Case (N 3, Upd (Val "s", Plus (Val "s", N 3)
```

Esempio : case fuori dal blocco switch

```
let d1 = Var(Int, "x", N 4 );;  
let d2= Var(Int, "s", N 0 );;  
let c1=Switch( Val "x", Case( N 1, Case(N 2,Upd  
↳ ( Val "s", Plus( Val "s", N 4)))));;  
let c2= Case( N 3, Case(N 4,Upd ( Val "s", Plus(  
↳ Val "s", N 4)))));;  
let p12= Prog("break5",Block ( SeqD(d1,d2), UnL  
↳ (SeqS(c1,c2)))));;  
printProg p12;;  
progSem p12;;
```

```
Program break5{  
  int x = 4;  
  int s = 0;  
  
  switch (x) case : 1 case : 2 s = (s + 4);  
  case : 3 case : 4 s = (s + 4);  
  }  
  
- : unit/2 = ()
```

```
Exception:  
StaticErrorS ("Invalid use of case",  
  Case (N 3, Case (N 4, Upd (Val "s",  
    Plus (Val "s", N 4))))).
```

Esenpio: break fuori dal blocco switch

```
let pc = Pcd( Int, "foo", FP( Value,Int, "b"),
↳ BlockP ( ED, SeqS( Break, Return (Val
↳ "b"))));;
let d3 = SeqD( pc, Var (Int , "a", EE));;
let s4 = Upd ( Val "a", Apply( "foo", AP ( N
↳ 10));;
let p5= Prog ( "BreakPcd", Block ( d3, UnL s4));;
```

```
Program BreakPcd{
    int foo(value int b){
        break ;
        return b;
    }
    int a;

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

```
Exception: StaticErrors
("Wrong use of break", Break).
```

Riferimenti bibliografici

-  Marco Bellia. *Materiale delle lezioni del corso di LPL: Laboratori 2,3,4,5,6,7.* Mag. 2020.
-  Marco Bellia. *Materiale delle lezioni del corso di LPL: Small21-Definizione4.* Giu. 2020.
-  Marco Bellia. *Materiale delle lezioni del corso di LPL:Lezione 4,6,7.* Mar. 2020.
-  Maurizio Gabbrielli e Simone Martini. *Programming Languages: Principles and Paradigms.* eng. Undergraduate topics in computer science. London: Springer London, Limited, 2010. ISBN: 9781848829138.
-  Brian W Kernighan e Dennis M Ritchie. *Il linguaggio C: principi di programmazione e manuale di riferimento.* Pearson Italia Spa, 2004.
-  Niklaus Wirth. «The programming language Pascal». In: *Acta informatica* 1.1 (1971), pp. 35–63.