# Linguaggi di Programmazione con Laboratorio

## Seminario di fine corso:

## `switch`

## e

## `break`

Simmaco Di Lillo

Università di Pisa, Dipartimento di Matematica

3 luglio 2021

# Comando case

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

# Comando case

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;
```

# Comando case

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

# Comando case

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

# Comando case

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`

# Comando case

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

# Comando case

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:

# Comando case

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`

# Comando case

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

# Comando `swith`

Alcuni linguaggi esibiscono differenze significative nel comando `case`.

# Comando `swith`

Alcuni linguaggi esibiscono differenze significative nel comando `case`.

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

# Comando `swith`

Alcuni linguaggi esibiscono differenze significative nel comando `case`.

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

```
Switch (Exp) corpo
```

# Comando `swith`

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.

# Comando `swith`

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

# Comando `swith`

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
```

# Comando `swith`

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

# Comando `swith`

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
```

# Comando Switch

Al momento dell'esecuzione del comando:

# Comando Switch

Al momento dell'esecuzione del comando:

- l'espressione `Exp` viene valutata,

# Comando 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`.

# Comando 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`.

# Comando 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 ;
    }
}
```

# Comando `Break`

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

# 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 e Sintassi Astratta

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

$$\texttt{Stm} \rightarrow \ldots \mid \texttt{switch}\ (\texttt{Exp})\ \texttt{Stm}$$

$$\begin{aligned}\texttt{OtherStm} \rightarrow \ldots &\mid \texttt{break;} \\ &\mid \texttt{case Exp : Stm} \\ &\mid \texttt{default : Stm}\end{aligned}$$

- **Sintassi Astratta in ocaml**

# Sintassi Concreta e Sintassi Astratta

- **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 e Sintassi Astratta

- **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  ;;
```

# Sistema dei tipi Y

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

# Sistema dei tipi Y

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

- `Case`

$$[\text{Y001}]\ \frac{\langle e, Y_\rho\rangle \to \langle \mathtt{t}, Y_\rho\rangle \qquad t \in \mathtt{Simple} \quad \langle \mathtt{stm}, Y_\rho\rangle \to_{\mathtt{Y}} \langle [\mathtt{void}], Y'_\rho\rangle}{\langle [\mathtt{case}]\ \mathtt{e}\ \mathtt{stm}, Y_\rho\rangle \to_{\mathtt{Y}} \langle [\mathtt{void}], Y'_\rho\rangle}$$

# Sistema dei tipi Y

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

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

$$[\text{E001}]\ \frac{\langle e, Y_\rho\rangle \to_{\mathtt{Y}} \langle \mathtt{t}, Y_\rho\rangle \qquad t \notin \mathtt{Simple}}{\langle [\mathtt{case}]\ \mathtt{e}\ \mathtt{stm}, Y_\rho\rangle \to_{\mathtt{Y}} \langle [\mathtt{terr}], Y_\rho\rangle}$$

$$[\text{E002}]\ \frac{\langle e, Y_\rho\rangle \to_{\mathtt{Y}} \langle \mathtt{t}, Y_\rho\rangle \qquad t \in \mathtt{Simple} \quad \langle \mathtt{stm}, Y_\rho\rangle \to_{\mathtt{Y}} \langle [\mathtt{terr}], Y'_\rho\rangle}{\langle [\mathtt{case}]\ \mathtt{e}\ \mathtt{stm}, Y_\rho\rangle \to_{\mathtt{Y}} \langle [\mathtt{terr}], Y'_\rho\rangle}$$

# Sistema dei tipi Y

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

- `switch`

$$[\text{Y002}]\ \frac{\begin{array}{c}\langle e, Y_\rho\rangle \rightarrow_{\text{Y}} \langle t, Y_\rho\rangle \qquad t \in \texttt{Simple} \\ \langle \texttt{stm}, Y_\rho\rangle \rightarrow_{\text{Y}} \langle [\texttt{void}], Y'_\rho\rangle\end{array}}{\langle [\texttt{switch}]\ \texttt{e}\ \texttt{stm}, Y_\rho\rangle \rightarrow_{\text{Y}} \langle [\texttt{void}], Y'_\rho\rangle}$$

# Sistema dei tipi Y

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

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

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

$$[\text{E004}]\ \frac{\langle e, Y_\rho\rangle \rightarrow_{\mathtt{Y}} \langle \mathtt{t}, Y_\rho\rangle \qquad t \in \mathtt{Simple} \quad \langle \mathtt{stm}, Y_\rho\rangle \rightarrow_{\mathtt{Y}} \langle [\mathtt{terr}], Y'_\rho\rangle}{\langle [\mathtt{switch}]\ \mathtt{e}\ \mathtt{stm}, Y_\rho\rangle \rightarrow_{\mathtt{Y}} \langle [\mathtt{terr}], Y'_\rho\rangle}$$

# Sistema dei tipi Y

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

- `default`

$$[\text{Y003}]\ \frac{\langle s, Y_\rho\rangle \rightarrow_{\text{Y}} \langle [\texttt{void}], Y'_\rho\rangle}{\langle [\texttt{default}]\ s, Y_\rho\rangle \rightarrow_{\text{Y}} \langle [\texttt{void}], Y'_\rho\rangle}$$

# Sistema dei tipi Y

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

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

$$[\text{E005}]\ \frac{\langle s, Y_\rho\rangle \rightarrow_{\text{Y}} \langle [\texttt{terr}], Y'_\rho\rangle}{\langle [\texttt{default}]\ s, Y_\rho\rangle \rightarrow_{\text{Y}} \langle [\texttt{terr}], Y'_\rho\rangle}$$

# Sistema dei tipi Y

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

- `break`

$$[\text{Y004}]\ \frac{}{\langle [\texttt{break}], Y_\rho \rangle \rightarrow_{\texttt{Y}} \langle [\texttt{void}], Y_\rho \rangle}$$

# Regole d'inferenza per $\mathtt{SEM}_{\mathtt{STM}}$

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

# Regole d'inferenza per $\mathtt{SEM}_{\mathtt{STM}}$

Mostriamolo per la regola $[S2]$

# Regole d'inferenza per $\text{SEM}_{\text{STM}}$

Mostriamolo per la regola $[S2]$

$$[\text{S2}]\ \frac{\begin{array}{c}\langle e, \sigma\rangle \rightarrow \lfloor[\texttt{bool}], \texttt{true}, \sigma_e\rfloor \\ \langle c, \sigma_e\rangle \rightarrow ([\texttt{void}], \sigma_1)\end{array}}{\langle[\texttt{ifT}]\ e\ c, \sigma\rangle \rightarrow ([\texttt{void}], \sigma_1)}$$

# Regole d'inferenza per $\mathrm{SEM}_{\mathrm{STM}}$

Mostriamolo per la regola $[S2]$

$$[\mathrm{S2}]\ \frac{\langle e, \sigma\rangle \to \lfloor[\mathtt{bool}], \mathtt{true}, \sigma_e\rfloor \quad \langle c, \sigma_e\rangle \to ([\mathtt{void}], \sigma_1)}{\langle[\mathtt{ifT}]\ e\ c, \sigma\rangle \to ([\mathtt{void}], \sigma_1)}$$

viene sostituita dalle due regole

# Regole d'inferenza per $\mathrm{SEM}_{\mathrm{STM}}$

Mostriamolo per la regola $[S2]$

$$[\mathrm{S2}]\ \frac{\langle e, \sigma\rangle \to \lfloor [\mathtt{bool}], \mathtt{true}, \sigma_e \rfloor \quad \langle c, \sigma_e\rangle \to ([\mathtt{void}], \sigma_1)}{\langle [\mathtt{ifT}]\ e\ c, \sigma\rangle \to ([\mathtt{void}], \sigma_1)}$$

viene sostituita dalle due regole

$$[\mathrm{S2.1}]\ \frac{\sigma = (ar + \Delta, \mu) \quad ar = \{h,\ l,\ f,\ c,\ vv\ \} \quad h \neq [\mathtt{SW}]\ t\ v \quad t \in \mathtt{Simple} \quad \langle e, \sigma\rangle \to \lfloor [\mathtt{bool}], \mathtt{true}, \sigma_e \rfloor \quad \langle c, \sigma_e\rangle \to ([\mathtt{void}], \sigma_1)}{\langle [\mathtt{ifT}]\ e\ c, \sigma\rangle \to ([\mathtt{void}], \sigma_1)}$$

$$[\mathrm{S2.2}]\ \frac{\sigma = (ar + \Delta, \mu) \quad ar = \{h,\ l,\ f,\ c,\ vv\ \} \quad h = [\mathtt{SW}]\ t\ v \quad t \in \mathtt{Simple} \quad \langle e, \sigma\rangle \to \lfloor [\mathtt{bool}], \mathtt{true}, \sigma_e \rfloor \quad \langle c, \sigma_e\rangle \to ([\mathtt{void}], \sigma_1)}{\langle [\mathtt{ifT}]\ e\ c, \sigma\rangle \to ([\mathtt{void}], \sigma)}$$

# Regole d'inferenza per $\text{SEM}_{\text{STM}}$ - switch

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

# Regole d'inferenza per $\text{SEM}_{\text{STM}}$ - switch

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

# Regole d'inferenza per $\mathrm{SEM}_{\mathrm{STM}}$ - `switch`

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

# Regole d'inferenza per $\mathrm{SEM}_{\mathrm{STM}}$ - case

$$[\text{S002A}]\ \frac{\begin{array}{c}\sigma = (ar + \Delta, \mu) \qquad ar = \{h,\ l,\ f,\ c,\ vv\ \} \\ h = [S]\ t \qquad\qquad t \in \mathit{Simple} \\ \langle e, \sigma\rangle \to \lfloor t_e, v_e, \sigma_e \rfloor \qquad\qquad t_e = t \\ \langle s, \sigma_e\rangle \to \langle [\texttt{void}], \sigma_F\rangle\end{array}}{\langle [\texttt{case}]\ e\ s, \sigma\rangle \to ([\texttt{void}], \sigma_F)}$$

# Regole d'inferenza per $\mathrm{SEM}_{\mathtt{STM}}$ - case

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

# Regole d'inferenza per $\mathrm{SEM}_{\mathrm{STM}}$ - case

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

# Regole d'inferenza per $\mathrm{SEM}_{\mathtt{STM}}$ - case

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

dove la funzione $k$

$$k([\mathtt{Int}], N) = [\mathtt{num}]\ N$$
$$k([\mathtt{Bool}], [\mathtt{true}]) = [\mathtt{true}]$$
$$k([\mathtt{Bool}], \mathtt{false}) = [\mathtt{false}]$$

# Regole d'inferenza per $\mathrm{SEM}_{\mathrm{STM}}$ - case

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

# Regole d'inferenza per $\text{SEM}_{\text{STM}}$ - `default`

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

Giunti allo statement `default` $s$

# Regole d'inferenza per $\text{SEM}_{\text{STM}}$ - `default`

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 $[S003A]$)

# Regole d'inferenza per $\text{SEM}_{\text{STM}}$ - `default`

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 $[S004C]$)

# Regole d'inferenza per $\text{SEM}_{\text{STM}}$ - `default`

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.

# Regole d'inferenza per $\mathrm{SEM}_{\mathtt{STM}}$ - default

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

# Regole d'inferenza per $\mathrm{SEM}_{\mathtt{STM}}$ - default

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

# Regole d'inferenza per $\mathrm{SEM}_{\mathtt{STM}}$ - default

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

# Regole d'inferenza per $\mathtt{SEM}_{\mathtt{STM}}$ - blockS

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

# Regole d'inferenza per $\text{SEM}_{\text{STM}}$ - blockS

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

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

# Regole d'inferenza per $\text{SEM}_{\text{STM}}$ - blockS

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

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

dove la funzione g

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

# Regole d'inferenza per $\mathtt{SEM}_{\mathtt{STM}}$ - blockS

Aggiungiamo anche la regola

# Regole d'inferenza per $\mathrm{SEM}_{\mathrm{STM}}$ - blockS

Aggiungiamo anche la regola

$$[\text{S13.2}]\ \frac{\begin{array}{c}\sigma = (\Delta, \mu) \\ \texttt{ar} = \{\texttt{h},\ \texttt{l},\ \texttt{f},\ \texttt{c},\ \texttt{vv}\} \qquad h = [SW]\ t\ v \\ \{g(h),\ 1,\ [],\ sts,\ []\} = ar_{top} \\ (ar_{top} + \Delta, \mu) \rightarrow_{R^\star} (ar' + \Delta', \mu') \\ (\Delta', \mu') = \sigma_F\end{array}}{\langle [\texttt{blockS}]\ ds\ sts, \sigma\rangle \rightarrow \langle [\texttt{void}], \sigma_F\rangle}$$

# Regole d'inferenza per $\mathrm{SEM}_{\mathrm{STM}}$ - break

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

# Regole d'inferenza per $\text{SEM}_{\text{STM}}$ - break

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

# Regole d'inferenza per $\mathrm{SEM}_{\mathrm{STM}}$ - break

$$[\text{S005C}]\ \frac{\begin{array}{c}\sigma = (\mathtt{ar} + \mathtt{ar}_1 + \Delta_r, \mu) \\ \mathtt{ar} = \{\mathtt{h},\ \mathtt{l},\ \mathtt{f},\ \mathtt{c},\ \mathtt{vv}\} \quad \mathtt{h} = [\mathrm{S}]\ t \quad t \in \mathit{Simple} \cup \{[\mathtt{Void}]\} \\ \mathtt{ar}_1 = \{\mathtt{hh},\ \mathtt{ll},\ \mathtt{ff},\ \mathtt{cc},\ \mathtt{vv}\} \quad \mathtt{hh} \neq [\mathrm{S}]\ t_2 \quad t_2 \in \mathit{Simple} \\ \{\mathtt{h},\ \mathtt{l},\ \mathtt{f},\ [],\ \mathtt{vv}\} = \mathtt{ar}' \\ (\mathtt{ar}' + \mathtt{ar}_1 + \Delta_r, \mu) = \sigma_F\end{array}}{\langle [\mathtt{break}], \sigma\rangle \rightarrow \langle [\mathtt{void}], \sigma_F\rangle}$$

# Modifiche al codice "small21.ml"

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 $\mathrm{SEM}_{\mathrm{STM}}$ 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))))
       ...
```

# Modifiche al codice "small21.ml"

Modifica della $\mathtt{SEM}_{\mathtt{STM}}$ 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))))
```

# Modifiche al codice "small21.ml"

## Modifica della $\mathrm{SEM}_{\mathrm{STM}}$ 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 $\mathtt{SEM}_{\mathtt{STM}}$ 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))))
```

# Modifiche al codice "small21.ml"

## Modifica della $\mathrm{SEM}_{\mathrm{STM}}$ 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 stm) in
                let sk4 =push sk3 ar in stmSem s (sk4,mu))
            | _ -> stmSem s (sk1,mu)))
      | _ ->raise(StaticErrorS("Invalid use of default",stm))))
```

# Modifiche al codice "small21.ml"

## Modifica della $\text{SEM}_{\text{STM}}$ 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)))))
```

# Modifiche al codice "small21.ml"

Modifica delle regole per tutti gli statement già presente ad eccezione di SeqS e BlockS.
Tale modifica serve affince stmSem stm restituisca

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

se:

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

# Modifiche al codice "small21.ml"

Modifica delle regole per tutti gli statement già presente ad eccezzione di SeqS e BlockS.
Tale modifica serve affince stmSem stm restituisca

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

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.