Esercizio per casa: grado alcolico

Supponendo di avere le seguenti 6 variabili booleane:

super, alc, liq, forte, norm e leggero

classificare un vino in base ai gradi alcolici rispetto alla tabella

gradi alcolici g	messaggio
$40 < g$	superalcolico
$20 < g \leq 40$	alcolico
$15 < g \leq 20$	vino liquoroso
$12 < g \leq 15$	vino forte
$10.5 < g \leq 12$	vino normale
$g \leq 10.5$	vino leggero

Esercizio per casa: grado alcolico

Supponendo di avere le seguenti 6 variabili booleane:

super, alc, liq, forte, norm e leggero

classificare un vino in base ai gradi alcolici rispetto alla tabella

gradi alcolici g	messaggio
$40 < g$	superalcolico
$20 < g \leq 40$	alcolico
$15 < g \leq 20$	vino liquoroso
$12 < g \leq 15$	vino forte
$10.5 < g \leq 12$	vino normale
$g \leq 10.5$	vino leggero

```
bool super; bool alc; bool liq; bool forte; bool norm; bool leggero;

if ( gradi > 40) { super := true; }

else { if ( gradi > 20) { alc := true; }

       else { if ( gradi > 15) { liq := true; }

              else { if ( gradi > 12) { forte := true; }

                     else { if ( gradi > 10) { norm := true; }

                            else { leggero := true; } } } } }
```

si può fare meglio?

Esercizio per casa: grado alcolico

Supponendo di avere le seguenti 6 variabili booleane:

super, alc, liq, forte, norm e leggero

classificare un vino in base ai gradi alcolici rispetto alla tabella

gradi alcolici g	messaggio
$40 < g$	superalcolico
$20 < g \leq 40$	alcolico
$15 < g \leq 20$	vino liquoroso
$12 < g \leq 15$	vino forte
$10.5 < g \leq 12$	vino normale
$g \leq 10.5$	vino leggero

```
bool super; bool alc; bool liq; bool forte; bool norm; bool leggero;

if ( gradi <= 15) { if ( gradi > 12) { forte := true; }

                    else { if ( gradi > 10) { norm := true; }

                           else { leggero := true; } }

} else { if ( gradi <= 20) { liq := true; }

         else { if ( gradi <= 40) { alc := true; }

                else { super := true; } }

}
```

Semantica operazionale delle espressioni

$$\frac{}{\langle\, \mathsf{V}\,,\rho\,,\sigma\,\rangle \Downarrow v} \qquad \frac{\sigma(\rho(\mathsf{Id}))=v}{\langle\, \mathsf{Id}\,,\rho\,,\sigma\,\rangle \Downarrow v} \qquad \frac{\langle\, \mathsf{E}_1\,,\rho\,,\sigma\,\rangle \Downarrow v}{\langle\, (\,\mathsf{E}_1\,)\,,\rho\,,\sigma\,\rangle \Downarrow v}$$

$$\frac{\langle\, \mathsf{E}_1\,,\rho\,,\sigma\,\rangle \Downarrow v_1 \qquad \langle\, \mathsf{E}_2\,,\rho\,,\sigma\,\rangle \Downarrow v_2}{\langle\, \mathsf{E}_1\ \mathit{bop}\ \mathsf{E}_2\,,\rho\,,\sigma\,\rangle \Downarrow v_1\ \mathit{bop}\ v_2} \qquad \frac{\langle\, \mathsf{E}_1\,,\rho\,,\sigma\,\rangle \Downarrow v_1}{\langle\, \mathit{uop}\ \mathsf{E}_1\,,\rho\,,\sigma\,\rangle \Downarrow \mathit{uop}\ v_1}$$

Semantica operazionale dei comandi (parziale)

$$\frac{}{\langle\ \mathsf{skip;}\ ,\ \rho\ ,\ \sigma\ \rangle \rightarrow \langle\ \rho\ ,\ \sigma\ \rangle} \qquad \frac{\langle\ \mathsf{C}_1\ ,\ \rho\ ,\ \sigma\ \rangle \rightarrow \langle\ \rho_1\ ,\ \sigma_1\ \rangle \quad \langle\ \mathsf{C}_2\ ,\ \rho_1\ ,\ \sigma_1\ \rangle \rightarrow \langle\ \rho_2\ ,\ \sigma_2\ \rangle}{\langle\ \mathsf{C}_1\ \mathsf{C}_2\ ,\ \rho\ ,\ \sigma\ \rangle \rightarrow \langle\ \rho_2\ ,\ \sigma_2\ \rangle}$$

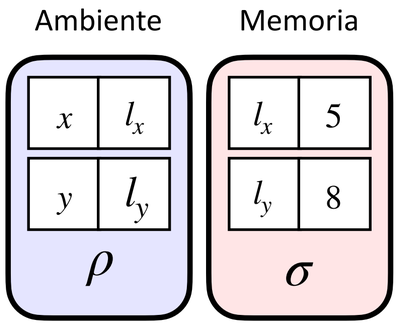
$$\frac{\langle\ \mathsf{E}\ ,\ \rho\ ,\ \sigma\ \rangle \Downarrow v \quad l \notin \sigma}{\langle\ \mathsf{T\ Id\ =\ E;}\ ,\ \rho\ ,\ \sigma\ \rangle \rightarrow \langle \rho[\mathsf{Id} \mapsto l]\ ,\ \sigma[l \mapsto v]\ \rangle} \qquad \frac{\langle\ \mathsf{E}\ ,\ \rho\ ,\ \sigma\ \rangle \Downarrow v \quad \rho(\mathsf{Id}) = l}{\langle\ \mathsf{Id\ :=\ E;}\ ,\ \rho\ ,\ \sigma\ \rangle \rightarrow \langle \rho\ ,\ \sigma[l \mapsto v]\ \rangle}$$

$$\frac{\langle\ \mathsf{C}\ ,\ \rho\ ,\ \sigma\ \rangle \rightarrow \langle\ \rho_1\ ,\ \sigma_1\ \rangle}{\langle\ \mathsf{\{\ C\ \}}\ ,\ \rho\ ,\ \sigma\ \rangle \rightarrow \langle\ \rho\ ,\ \sigma_1\ \rangle}$$

$$\frac{\langle\ \mathsf{E}\ ,\ \rho\ ,\ \sigma\ \rangle \Downarrow \mathit{true} \quad \langle\ \mathsf{C}_1\ ,\ \rho\ ,\ \sigma\ \rangle \rightarrow \langle\ \rho_1\ ,\ \sigma_1\ \rangle}{\langle\ \mathsf{if\ (\ E\ )\ \{\ C_1\ \}\ else\ \{\ C_2\ \}}\ ,\ \rho\ ,\ \sigma\ \rangle \rightarrow \langle\ \rho\ ,\ \sigma_1\ \rangle}$$

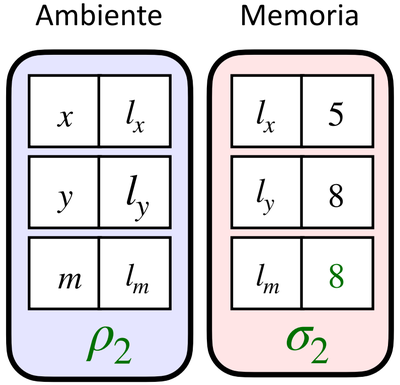
$$\frac{\langle\ \mathsf{E}\ ,\ \rho\ ,\ \sigma\ \rangle \Downarrow \mathit{false} \quad \langle\ \mathsf{C}_2\ ,\ \rho\ ,\ \sigma\ \rangle \rightarrow \langle\ \rho_2\ ,\ \sigma_2\ \rangle}{\langle\ \mathsf{if\ (\ E\ )\ \{\ C_1\ \}\ else\ \{\ C_2\ \}}\ ,\ \rho\ ,\ \sigma\ \rangle \rightarrow \langle\ \rho\ ,\ \sigma_2\ \rangle}$$

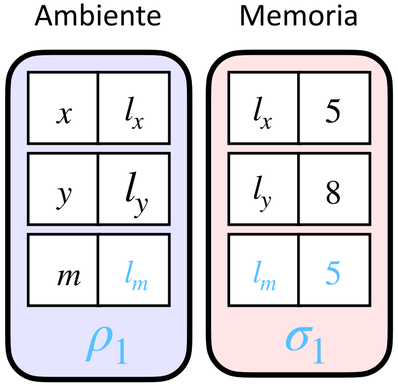
Esempi di esecuzione di comandi



C_1 = `int m=x;`

C_2 = `if (y>m) { m:=y;}`





$\langle C_1\ C_2 , \rho , \sigma \rangle \rightarrow \langle \rho_2 , \sigma_2 \rangle$

$\langle \text{int } m = x; , \rho , \sigma \rangle \rightarrow \langle \rho_1 , \sigma_1 \rangle$

$$\frac{\langle E , \rho , \sigma \rangle \Downarrow v \quad l \notin \sigma}{\langle T\ Id\ = E; , \rho , \sigma \rangle \rightarrow \langle \rho[Id \mapsto l] , \sigma[l \mapsto v] \rangle}$$

$\langle x , \rho , \sigma \rangle \Downarrow 5 \quad l_m \notin \sigma \quad \rho_1 = \rho[m \mapsto l_m] \quad \sigma_1 = \sigma[l_m \mapsto 5]$

$\langle \text{if } (y>m) \{ m:=y;\} , \rho_1 , \sigma_1 \rangle \rightarrow \langle \rho_2 , \sigma_2 \rangle \quad \rho_2 = \rho_1$

$$\frac{\langle E , \rho , \sigma \rangle \Downarrow true \quad \langle C_1 , \rho , \sigma \rangle \rightarrow \langle \rho_1 , \sigma_1 \rangle}{\langle \text{if } (E) \{ C_1 \} \text{ else } \{ C_2 \} , \rho , \sigma \rangle \rightarrow \langle \rho , \sigma_1 \rangle}$$

$\langle y > m; , \rho_1 , \sigma_1 \rangle \Downarrow true$ // omettiamo i dettagli!

$\langle m := y; , \rho_1 , \sigma_1 \rangle \rightarrow \langle \rho_1 , \sigma_2 \rangle$

$$\frac{\langle E , \rho , \sigma \rangle \Downarrow v \quad \rho(Id) = l}{\langle Id\ := E; , \rho , \sigma \rangle \rightarrow \langle \rho , \sigma[l \mapsto v] \rangle}$$

$\langle y , \rho_1 , \sigma_1 \rangle \Downarrow 8 \quad \sigma_2 = \sigma_1[l_m \mapsto 8]$

MiniMao: Semantica Operazionale

Cicli

Comando iterativo

I comandi iterativi servono per ripetere azioni nei programmi

Ad esempio “Mangiare cioccolatini da una scatola finché non è vuota”

In linguaggio naturale:

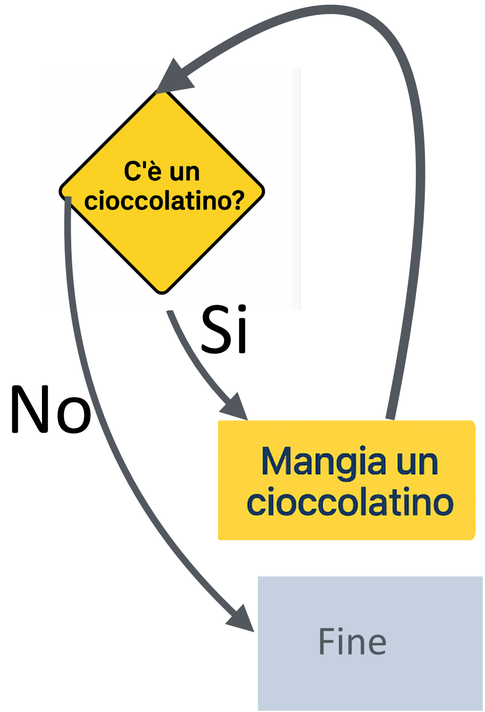
- Condizione: *Ci sono cioccolatini nella scatola?*
- Azioni se si: *Mangia un cioccolatino e guarda ancora nella scatola*
- Azione se no: *Finisci*

Comando iterativo in Mao

while (E) { C }

E è un'espressione booleana:

se vera si esegue C e si prova ancora, altrimenti si termina



Guardia

Corpo

```
while ( cioccolatini > 0 ) {
    cioccolatini := cioccolatini - 1;
}
```

Esempio: numeri triangolari

Scrivere un programma che calcola la somma di tutti i numeri naturali minori o uguali a n

```
int somma = 0;
int i = 0;
while ( i <= n ) {
    somma := somma + i;
    i := i + 1;
}
```

Che succede se parto con $n < 0$?

Non eseguo nessuna istruzione

Che succede se parto con $n \geq 0$?

L'assegnamento somma:=somma+i; viene ripetuto fino a quando i non supera n

Il blocco dei comandi iterativi in MAO

Come avevamo visto per i rami then/else dei comandi condizionali, anche Il corpo C del ciclo in while (E) {C} è racchiuso in un blocco

Qual è la conseguenza?

```
int somma = 0;
int i = 0;
while ( i <= n ) {
    somma := somma + i;
    int i = i + 1;
}
```

una volta entrati,

potremo mai uscire dal ciclo?

Esempio

Scrivere un programma che dati due numeri positivi x e y calcola x^y

```
int potenza = 1;
while (y > 0) {
    potenza := potenza * x;
    y := y - 1;
}
```

problema?

perdiamo il valore iniziale di y

Esempio

Scrivere un programma che dati due numeri positivi x e y calcola x^y

```
int potenza = 1;
int k = y;
while (k > 0) {
    potenza := potenza * x;
    k := k - 1;
}
```

Esempio: cerchiamo un divisore di n

Dato un numero n maggiore di 1, cerchiamo il suo più piccolo divisore non unitario (cioè diverso da 1)

```
int i = 2;
while (i <= n) {
    if (n%i == 0) { // i divide n?
        int div = i;
    }
    i := i + 1;
}
```

problema #1:

div non visibile fuori dal ciclo!

Esempio: cerchiamo un divisore di n

Dato un numero n maggiore di 1, cerchiamo il suo più piccolo divisore non unitario (cioè diverso da 1)

```
int i = 2;
while (i <= n) {
    int div;
    if (n%i == 0) { // i divide n?
        div := i;
    }
    i := i + 1;
}
```

problema #2:

div non visibile fuori dal ciclo!

Esempio: cerchiamo un divisore di n

Dato un numero n maggiore di 1, cerchiamo il suo più piccolo divisore non unitario (cioè diverso da 1)

```
int i = 2;
int div;
while (i <= n) {
    if (n%i == 0) { // i divide n?
        div := i;
    }
    i := i + 1;
}
```

problema #3:

alla fine del ciclo div=n

Esempio: cerchiamo un divisore di n

Dato un numero n maggiore di 1, cerchiamo il suo più piccolo divisore non unitario (cioè diverso da 1)

```
int i = 2;
int div;
while (i <= n) {
    if (n%i == 0) { // i divide n?
        div := i;
        i := n;     // forzo uscita dal ciclo
    }
    i := i + 1;
}
```

problema #4:

inguardabile!

Esempio: cerchiamo un divisore di n

Dato un numero n maggiore di 1, cerchiamo il suo più piccolo divisore non unitario (cioè diverso da 1)

```
int i = 2;
int div;
while (i<=n && div==0) { // trovato divisore?
    if (n%i == 0) { // i divide n?
        div := i;
    }
    i := i + 1;
}
```

problema #5:

si può fare meglio?

Esempio: cerchiamo un divisore di n

Dato un numero n maggiore di 1, cerchiamo il suo più piccolo divisore non unitario (cioè diverso da 1)

```
int div = 2; // usiamo solo div
while (div<=n && (n%div == 0)) {
    div := div + 1;
}
```

problema #6:

attenzione alla condizione logica

Esempio: cerchiamo un divisore di n

Dato un numero n maggiore di 1, cerchiamo il suo più piccolo divisore non unitario (cioè diverso da 1)

```
int div = 2;
while (div<=n && (n%div != 0)) {
    div := div + 1;
}
```

problema #8:

possiamo fermarci prima?

Esempio: cerchiamo un divisore di n

Dato un numero n maggiore di 1, cerchiamo il suo più piccolo divisore non unitario (cioè diverso da 1)

```
int div = 2;
while (div<=(n/2) && (n%div != 0)) {
    div := div + 1;
}
```

problema #9:

e se n fosse un numero primo?

Esempio: cerchiamo un divisore di n

Dato un numero n maggiore di 1, cerchiamo il suo più piccolo divisore non unitario (cioè diverso da 1)

```
int div = 2;
while (div<=(n/2) && (n%div != 0)) {
    div := div + 1;
}
if (div > (n/2)) { div := n; }
```

se entro nel ciclo:

i numeri da 2 a div non dividono n

incrementando div,

prima o poi esco dal ciclo

quando esco dal ciclo:

div divide n, oppure div>(n/2)

come convincersi che

"funzioni" correttamente?

Esercizio

Scrivere un programma che dato un numero positivo n ne calcola il fattoriale

$$n! = 1 \times 2 \times \cdots \times n$$

```
int fattoriale = 1;
while (n > 0) {
    fattoriale := fattoriale * n;
    n := n - 1;
}
```

Esercizio

Scrivere un programma che calcoli il quoziente q e il resto r della divisione tra due numeri interi positivi x e y.

In altre parole vogliamo calcolare q e r tali che $x = qy + r$ con $0 \leq r < y$.

Da questa formulazione possiamo progettare l'algoritmo:
per ottenere il quoziente q dobbiamo contare quante volte si può togliere y da x, poi il numero rimasto sarà il resto r.

senza usare

gli operatori / e %

Esercizio

Scrivere un programma che calcoli il quoziente q e il resto r della divisione tra due numeri interi positivi x e y, ovvero q e r tali che $x = qy + r$ con $0 \leq r < y$.

```
int q = 0;
int r = x;
while (r >= y) {
    q := q + 1;
    r := r - y;
}
```

all'inizio vale $x = qy + r$

a furia di sottrarre un valore positivo si otterrà $r < y$ (terminazione)

dopo ogni iterazione vale ancora $x = qy + r = (q + 1)y + (r - y)$

come convincersi che "funzioni" correttamente?

MiniMao: Semantica Operazionale

Cicli

Semantica dei comandi (seconda parte)

C ::= ... | while (E) { C }

Per eseguire il ciclo while (E) {C} nello stato (ρ , σ):

- valutiamo la guardia E nello stato (ρ , σ) ottenendo v
- se v è false lo stato finale sarà proprio (ρ , σ)
- se v è true bisogna:
  - eseguire il corpo C in (ρ , σ) ottenendo un nuovo stato (ρ_1 , σ_1)
  - restituire la memoria finale σ_2 ottenuta eseguendo while (E) {C} nello stato (ρ , σ_1)
  - trattandosi di blocchi, si ripristina sempre l'ambiente iniziale ρ

Semantica dei comandi (seconda parte)

C ::= ... | while (E) { C } Come per il comando condizionale usiamo due regole:

- una per il caso in cui la guardia E è falsa (terminazione)
- una per il caso in cui la guardia E è vera (iterazione)

Per eseguire il ciclo nello stato (ρ , σ) valutiamo la guardia E nello stato (ρ , σ) ottenendo v

- se v è false lo stato finale sarà proprio (ρ , σ)
- se v è true bisogna:
  - eseguire il corpo C in (ρ , σ) ottenendo un nuovo stato (ρ_1 , σ_1)
  - restituire la memoria finale σ_2 ottenuta eseguendo while (E) {C} nello stato (ρ_1 , σ_1)
  - trattandosi di blocchi, si ripristina sempre l'ambiente iniziale ρ

$$\frac{\langle \text{E} , \rho , \sigma \rangle \Downarrow \mathit{false}}{\langle \text{while (E) \{ C \}} , \rho , \sigma \rangle \rightarrow \langle \rho , \sigma \rangle}$$

$$\frac{\langle \text{E} , \rho , \sigma \rangle \Downarrow \mathit{true} \quad \langle \text{C} , \rho , \sigma \rangle \rightarrow \langle \rho_1 , \sigma_1 \rangle \quad \langle \text{while (E) \{ C \}} , \rho , \sigma_1 \rangle \rightarrow \langle \rho_2 , \sigma_2 \rangle}{\langle \text{while (E) \{ C \}} , \rho , \sigma \rangle \rightarrow \langle \rho , \sigma_2 \rangle}$$

$$\frac{}{\langle\ \text{skip;}\ ,\rho\ ,\sigma\ \rangle \to \langle\ \rho\ ,\sigma\ \rangle}$$

$$\frac{\langle\ \mathsf{C}_1\ ,\rho\ ,\sigma\ \rangle \to \langle\ \rho_1\ ,\sigma_1\ \rangle \quad \langle\ \mathsf{C}_2\ ,\rho_1\ ,\sigma_1\ \rangle \to \langle\ \rho_2\ ,\sigma_2\ \rangle}{\langle\ \mathsf{C}_1\ \mathsf{C}_2\ ,\rho\ ,\sigma\ \rangle \to \langle\ \rho_2\ ,\sigma_2\ \rangle}$$

$$\frac{\langle\ \mathsf{E}\ ,\rho\ ,\sigma\ \rangle \Downarrow v \quad l \notin \sigma}{\langle\ \mathsf{T\ Id\ =\ E;}\ ,\rho\ ,\sigma\ \rangle \to \langle \rho[\mathsf{Id} \mapsto l]\ ,\sigma[l \mapsto v]\ \rangle}$$

$$\frac{\langle\ \mathsf{E}\ ,\rho\ ,\sigma\ \rangle \Downarrow v \quad \rho(\mathsf{Id}){=}l}{\langle\ \mathsf{Id\ :=\ E;}\ ,\rho\ ,\sigma\ \rangle \to \langle \rho\ ,\sigma[l \mapsto v]\ \rangle}$$

$$\frac{\langle\ \mathsf{C}\ ,\rho\ ,\sigma\ \rangle \to \langle\ \rho_1\ ,\sigma_1\ \rangle}{\langle\ \{\ \mathsf{C}\ \}\ ,\rho\ ,\sigma\ \rangle \to \langle\ \rho\ ,\sigma_1\ \rangle}$$

$$\frac{\langle\ \mathsf{E}\ ,\rho\ ,\sigma\ \rangle \Downarrow \mathit{true} \quad \langle\ \mathsf{C}_1\ ,\rho\ ,\sigma\ \rangle \to \langle\ \rho_1\ ,\sigma_1\ \rangle}{\langle\ \mathsf{if\ (\ E\ )\ \{\ C_1\ \}\ else\ \{\ C_2\ \}}\ ,\rho\ ,\sigma\ \rangle \to \langle\ \rho\ ,\sigma_1\ \rangle}$$

$$\frac{\langle\ \mathsf{E}\ ,\rho\ ,\sigma\ \rangle \Downarrow \mathit{false}}{\langle\ \mathsf{while\ (\ E\ )\ \{\ C\ \}}\ ,\rho\ ,\sigma\ \rangle \to \langle\ \rho\ ,\sigma\ \rangle}$$

$$\frac{\langle\ \mathsf{E}\ ,\rho\ ,\sigma\ \rangle \Downarrow \mathit{false} \quad \langle\ \mathsf{C}_2\ ,\rho\ ,\sigma\ \rangle \to \langle\ \rho_2\ ,\sigma_2\ \rangle}{\langle\ \mathsf{if\ (\ E\ )\ \{\ C_1\ \}\ else\ \{\ C_2\ \}}\ ,\rho\ ,\sigma\ \rangle \to \langle\ \rho\ ,\sigma_2\ \rangle}$$

$$\frac{\langle\ \mathsf{E}\ ,\rho\ ,\sigma\ \rangle \Downarrow \mathit{true} \quad \langle\ \mathsf{C}\ ,\rho\ ,\sigma\ \rangle \to \langle\ \rho_1\ ,\sigma_1\ \rangle \quad \langle\ \mathsf{while\ (\ E\ )\ \{\ C\ \}}\ ,\rho\ ,\sigma_1\ \rangle \to \langle\ \rho_2\ ,\sigma_2\ \rangle}{\langle\ \mathsf{while\ (\ E\ )\ \{\ C\ \}}\ ,\rho\ ,\sigma\ \rangle \to \langle\ \rho\ ,\sigma_2\ \rangle}$$

Semantica operazionale dei comandi MiniMao

$$\frac{}{\langle\, \texttt{skip;} , \rho , \sigma \,\rangle \to \langle\, \rho , \sigma \,\rangle} \qquad \frac{\langle\, \texttt{C}_1 , \rho , \sigma \,\rangle \to \langle\, \rho_1 , \sigma_1 \,\rangle \quad \langle\, \texttt{C}_2 , \rho_1 , \sigma_1 \,\rangle \to \langle\, \rho_2 , \sigma_2 \,\rangle}{\langle\, \texttt{C}_1\ \texttt{C}_2 , \rho , \sigma \,\rangle \to \langle\, \rho_2 , \sigma_2 \,\rangle}$$

$$\frac{\langle\, \texttt{E} , \rho , \sigma \,\rangle \Downarrow v \quad l \notin \sigma}{\langle\, \texttt{T Id = E;} , \rho , \sigma \,\rangle \to \langle \rho[\texttt{Id} \mapsto l] , \sigma[l \mapsto v] \,\rangle} \qquad \frac{\langle\, \texttt{E} , \rho , \sigma \,\rangle \Downarrow v \quad \rho(\texttt{Id}) = l}{\langle\, \texttt{Id := E;} , \rho , \sigma \,\rangle \to \langle \rho , \sigma[l \mapsto v] \,\rangle}$$

$$\frac{\langle\, \texttt{C} , \rho , \sigma \,\rangle \to \langle\, \rho_1 , \sigma_1 \,\rangle}{\langle\, \texttt{\{ C \}} , \rho , \sigma \,\rangle \to \langle\, \rho , \sigma_1 \,\rangle} \qquad \frac{\langle\, \texttt{E} , \rho , \sigma \,\rangle \Downarrow \mathit{true} \quad \langle\, \texttt{C}_1 , \rho , \sigma \,\rangle \to \langle\, \rho_1 , \sigma_1 \,\rangle}{\langle\, \texttt{if (E) \{ C}_1 \texttt{ \} else \{ C}_2 \texttt{ \}} , \rho , \sigma \,\rangle \to \langle\, \rho , \sigma_1 \,\rangle}$$

$$\frac{\langle\, \texttt{E} , \rho , \sigma \,\rangle \Downarrow \mathit{false}}{\langle\, \texttt{while (E) \{ C \}} , \rho , \sigma \,\rangle \to \langle\, \rho , \sigma \,\rangle} \qquad \frac{\langle\, \texttt{E} , \rho , \sigma \,\rangle \Downarrow \mathit{false} \quad \langle\, \texttt{C}_2 , \rho , \sigma \,\rangle \to \langle\, \rho_2 , \sigma_2 \,\rangle}{\langle\, \texttt{if (E) \{ C}_1 \texttt{ \} else \{ C}_2 \texttt{ \}} , \rho , \sigma \,\rangle \to \langle\, \rho , \sigma_2 \,\rangle}$$

$$\frac{\langle\, \texttt{E} , \rho , \sigma \,\rangle \Downarrow \mathit{true} \quad \langle\, \texttt{C} , \rho , \sigma \,\rangle \to \langle\, \rho_1 , \sigma_1 \,\rangle \quad \langle\, \texttt{while (E) \{ C \}} , \rho , \sigma_1 \,\rangle \to \langle\, \rho_2 , \sigma_2 \,\rangle}{\langle\, \texttt{while (E) \{ C \}} , \rho , \sigma \,\rangle \to \langle\, \rho , \sigma_2 \,\rangle}$$

Esempio

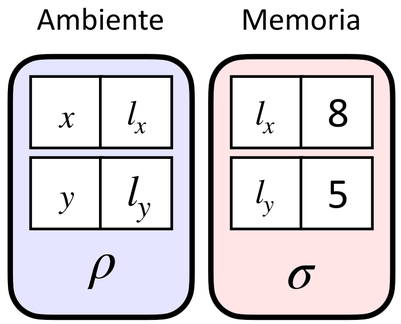
```
C1 = int q = 0;
C2 = int r = x;
C3 = while (r >= y) { q := q + 1; r := r - y; }
```

$$\langle C_1\ C_2\ C_3, \rho, \sigma \rangle \rightarrow \langle \rho_3, \sigma_3 \rangle$$

$$\langle \text{int } q = 0;, \rho, \sigma \rangle \rightarrow \langle \rho_1, \sigma_1 \rangle$$

$$\langle \text{int } r = x;, \rho_1, \sigma_1 \rangle \rightarrow \langle \rho_2, \sigma_2 \rangle$$

$$\langle C_3, \rho_2, \sigma_2 \rangle \rightarrow \langle \rho_3, \sigma_3 \rangle \qquad \rho_3 = \rho_2$$



$$\frac{}{\langle \text{skip};, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma \rangle} \qquad \frac{\langle C_1, \rho, \sigma \rangle \rightarrow \langle \rho_1, \sigma_1 \rangle \quad \langle C_2, \rho_1, \sigma_1 \rangle \rightarrow \langle \rho_2, \sigma_2 \rangle}{\langle C_1\ C_2, \rho, \sigma \rangle \rightarrow \langle \rho_2, \sigma_2 \rangle}$$

$$\frac{\langle E, \rho, \sigma \rangle \Downarrow v \quad l \notin \sigma}{\langle T\ Id\ = E;, \rho, \sigma \rangle \rightarrow \langle \rho[Id \mapsto l], \sigma[l \mapsto v] \rangle} \qquad \frac{\langle E, \rho, \sigma \rangle \Downarrow v \quad \rho(Id) = l}{\langle Id\ := E;, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma[l \mapsto v] \rangle}$$

$$\frac{\langle C, \rho, \sigma \rangle \rightarrow \langle \rho_1, \sigma_1 \rangle}{\langle \{ C \}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma_1 \rangle} \qquad \frac{\langle E, \rho, \sigma \rangle \Downarrow true \quad \langle C_1, \rho, \sigma \rangle \rightarrow \langle \rho_1, \sigma_1 \rangle}{\langle \text{if } (E) \{ C_1 \} \text{ else } \{ C_2 \}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma_1 \rangle}$$

$$\frac{\langle E, \rho, \sigma \rangle \Downarrow false}{\langle \text{while } (E) \{ C \}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma \rangle} \qquad \frac{\langle E, \rho, \sigma \rangle \Downarrow false \quad \langle C_2, \rho, \sigma \rangle \rightarrow \langle \rho_2, \sigma_2 \rangle}{\langle \text{if } (E) \{ C_1 \} \text{ else } \{ C_2 \}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma_2 \rangle}$$

$$\frac{\langle E, \rho, \sigma \rangle \Downarrow true \quad \langle C, \rho, \sigma \rangle \rightarrow \langle \rho_1, \sigma_1 \rangle \quad \langle \text{while } (E) \{ C \}, \rho, \sigma_1 \rangle \rightarrow \langle \rho_2, \sigma_2 \rangle}{\langle \text{while } (E) \{ C \}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma_2 \rangle}$$

Applicando la regola (associativa) per la sequenza di comandi,

per determinare lo stato finale $\langle \rho_3, \sigma_3 \rangle$

devo prima determinare gli stati intermedi $\langle \rho_1, \sigma_1 \rangle$ e $\langle \rho_2, \sigma_2 \rangle$

Dato che le regole dei cicli preservano l'ambiente di partenza: $\rho_3 = \rho_2$

Esempio

```
C1 = int q = 0;
C2 = int r = x;
C3 = while (r >= y) { q := q + 1; r := r - y; }
```

$$\langle C_1\ C_2\ C_3, \rho, \sigma \rangle \rightarrow \langle \rho_3, \sigma_3 \rangle$$

$$\langle \text{int } q = 0;, \rho, \sigma \rangle \rightarrow \langle \rho_1, \sigma_1 \rangle$$

$$\langle \text{int } r = x;, \rho_1, \sigma_1 \rangle \rightarrow \langle \rho_2, \sigma_2 \rangle$$

$$\langle C_3, \rho_2, \sigma_2 \rangle \rightarrow \langle \rho_3, \sigma_3 \rangle \qquad \rho_3 = \rho_2$$

$$\langle r \geq y, \rho_3, \sigma_2 \rangle \Downarrow \mathit{true}$$

Dato che la guardia è vera
devo eseguire il corpo del ciclo

$$\frac{\langle E, \rho, \sigma \rangle \Downarrow \mathit{true} \quad \langle C, \rho, \sigma \rangle \rightarrow \langle \rho_1, \sigma_1 \rangle \quad \langle \text{while } (E)\ \{C\}, \rho, \sigma_1 \rangle \rightarrow \langle \rho_2, \sigma_2 \rangle}{\langle \text{while } (E)\ \{C\}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma_2 \rangle}$$

Ambiente

x	l_x
y	l_y

ρ

Memoria

l_x	8
l_y	5

σ

Ambiente

x	l_x
y	l_y
q	l_q

ρ_1

Memoria

l_x	8
l_y	5
l_q	0

σ_1

Ambiente

x	l_x
y	l_y
q	l_q
r	l_r

ρ_3 ρ_2

Memoria

l_x	8
l_y	5
l_q	0
l_r	8

σ_2

$$\frac{}{\langle \text{skip;}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma \rangle} \qquad \frac{\langle C_1, \rho, \sigma \rangle \rightarrow \langle \rho_1, \sigma_1 \rangle \quad \langle C_2, \rho_1, \sigma_1 \rangle \rightarrow \langle \rho_2, \sigma_2 \rangle}{\langle C_1\ C_2, \rho, \sigma \rangle \rightarrow \langle \rho_2, \sigma_2 \rangle}$$

$$\frac{\langle E, \rho, \sigma \rangle \Downarrow v \quad l \notin \sigma}{\langle T\ \text{Id} = E;, \rho, \sigma \rangle \rightarrow \langle \rho[\text{Id} \mapsto l], \sigma[l \mapsto v] \rangle} \qquad \frac{\langle E, \rho, \sigma \rangle \Downarrow v \quad \rho(\text{Id}) = l}{\langle \text{Id} := E;, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma[l \mapsto v] \rangle}$$

$$\frac{\langle C, \rho, \sigma \rangle \rightarrow \langle \rho_1, \sigma_1 \rangle}{\langle \{C\}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma_1 \rangle} \qquad \frac{\langle E, \rho, \sigma \rangle \Downarrow \mathit{true} \quad \langle C_1, \rho, \sigma \rangle \rightarrow \langle \rho_1, \sigma_1 \rangle}{\langle \text{if } (E)\ \{C_1\}\ \text{else}\ \{C_2\}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma_1 \rangle}$$

$$\frac{\langle E, \rho, \sigma \rangle \Downarrow \mathit{false}}{\langle \text{while } (E)\ \{C\}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma \rangle} \qquad \frac{\langle E, \rho, \sigma \rangle \Downarrow \mathit{false} \quad \langle C_2, \rho, \sigma \rangle \rightarrow \langle \rho_2, \sigma_2 \rangle}{\langle \text{if } (E)\ \{C_1\}\ \text{else}\ \{C_2\}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma_2 \rangle}$$

$$\frac{\langle E, \rho, \sigma \rangle \Downarrow \mathit{true} \quad \langle C, \rho, \sigma \rangle \rightarrow \langle \rho_1, \sigma_1 \rangle \quad \langle \text{while } (E)\ \{C\}, \rho, \sigma_1 \rangle \rightarrow \langle \rho_2, \sigma_2 \rangle}{\langle \text{while } (E)\ \{C\}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma_2 \rangle}$$

Esempio

```
C1 = int q = 0;
C2 = int r = x;
C3 = while (r >= y) { q := q + 1; r := r - y; }
```

$$\langle C_1\ C_2\ C_3, \rho, \sigma \rangle \rightarrow \langle \rho_3, \sigma_3 \rangle$$

$$\langle \text{int } q = 0;, \rho, \sigma \rangle \rightarrow \langle \rho_1, \sigma_1 \rangle$$

$$\langle \text{int } r = x;, \rho_1, \sigma_1 \rangle \rightarrow \langle \rho_2, \sigma_2 \rangle$$

$$\langle C_3, \rho_2, \sigma_2 \rangle \rightarrow \langle \rho_3, \sigma_3 \rangle \qquad \rho_3 = \rho_2$$

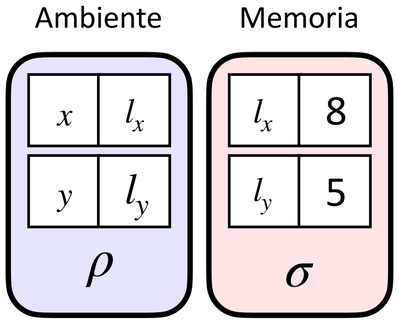
$$\langle r \geq y, \rho_3, \sigma_2 \rangle \Downarrow \mathit{true}$$

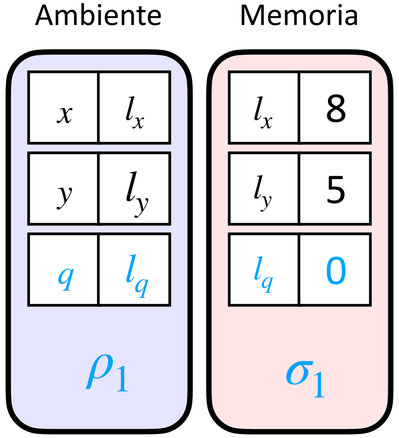
$$\langle q := q+1;, \rho_3, \sigma_2 \rangle \rightarrow \langle \rho_3, \sigma_4 \rangle$$

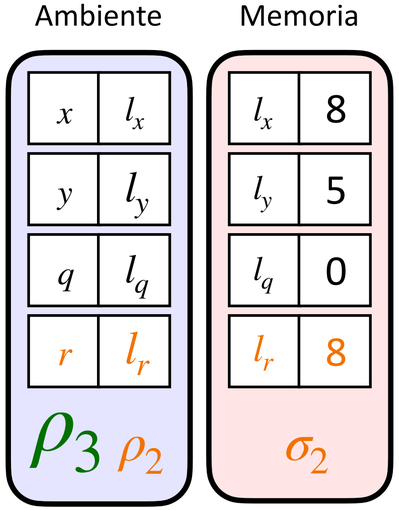
$$\langle r := r-y;, \rho_3, \sigma_4 \rangle \rightarrow \langle \rho_3, \sigma_5 \rangle$$

$$\langle C_3, \rho_3, \sigma_5 \rangle \rightarrow \langle \rho_3, \sigma_3 \rangle$$

Devo valutare nuovamente la guardia







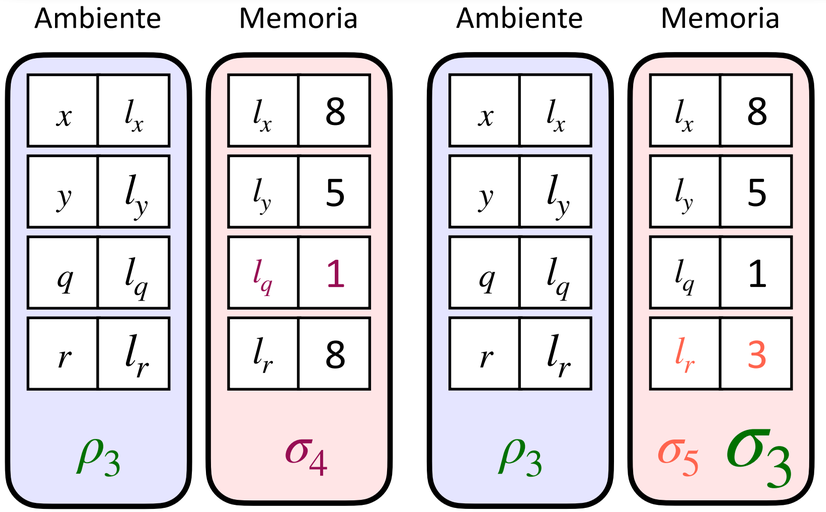
$$\frac{}{\langle \text{skip;}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma \rangle} \qquad \frac{\langle C_1, \rho, \sigma \rangle \rightarrow \langle \rho_1, \sigma_1 \rangle \quad \langle C_2, \rho_1, \sigma_1 \rangle \rightarrow \langle \rho_2, \sigma_2 \rangle}{\langle C_1\ C_2, \rho, \sigma \rangle \rightarrow \langle \rho_2, \sigma_2 \rangle}$$

$$\frac{\langle E, \rho, \sigma \rangle \Downarrow v \quad l \notin \sigma}{\langle T\ Id\ = E;, \rho, \sigma \rangle \rightarrow \langle \rho[Id \mapsto l], \sigma[l \mapsto v] \rangle} \qquad \frac{\langle E, \rho, \sigma \rangle \Downarrow v \quad \rho(Id) = l}{\langle Id\ := E;, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma[l \mapsto v] \rangle}$$

$$\frac{\langle C, \rho, \sigma \rangle \rightarrow \langle \rho_1, \sigma_1 \rangle}{\langle \{ C \}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma_1 \rangle} \qquad \frac{\langle E, \rho, \sigma \rangle \Downarrow \mathit{true} \quad \langle C_1, \rho, \sigma \rangle \rightarrow \langle \rho_1, \sigma_1 \rangle}{\langle \text{if } (E) \{ C_1 \} \text{ else } \{ C_2 \}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma_1 \rangle}$$

$$\frac{\langle E, \rho, \sigma \rangle \Downarrow \mathit{false}}{\langle \text{while } (E) \{ C \}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma \rangle} \qquad \frac{\langle E, \rho, \sigma \rangle \Downarrow \mathit{false} \quad \langle C_2, \rho, \sigma \rangle \rightarrow \langle \rho_2, \sigma_2 \rangle}{\langle \text{if } (E) \{ C_1 \} \text{ else } \{ C_2 \}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma_2 \rangle}$$

$$\frac{\langle E, \rho, \sigma \rangle \Downarrow \mathit{true} \quad \langle C, \rho, \sigma \rangle \rightarrow \langle \rho_1, \sigma_1 \rangle \quad \langle \text{while } (E) \{ C \}, \rho, \sigma_1 \rangle \rightarrow \langle \rho_2, \sigma_2 \rangle}{\langle \text{while } (E) \{ C \}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma_2 \rangle}$$



Esempio

```
C1 = int q = 0;
C2 = int r = x;
C3 = while (r >= y) { q := q + 1; r := r - y; }
```

$$\langle C_1\ C_2\ C_3, \rho, \sigma \rangle \rightarrow \langle \rho_3, \sigma_3 \rangle$$

$$\langle \text{int } q = 0;, \rho, \sigma \rangle \rightarrow \langle \rho_1, \sigma_1 \rangle$$

$$\langle \text{int } r = x;, \rho_1, \sigma_1 \rangle \rightarrow \langle \rho_2, \sigma_2 \rangle$$

$$\langle C_3, \rho_2, \sigma_2 \rangle \rightarrow \langle \rho_3, \sigma_3 \rangle \quad \rho_3 = \rho_2$$

$$\langle r \geq y, \rho_3, \sigma_2 \rangle \Downarrow \mathit{true}$$

$$\langle q := q+1;, \rho_3, \sigma_2 \rangle \rightarrow \langle \rho_3, \sigma_4 \rangle$$

$$\langle r := r-y;, \rho_3, \sigma_4 \rangle \rightarrow \langle \rho_3, \sigma_5 \rangle$$

$$\langle C_3, \rho_3, \sigma_5 \rangle \rightarrow \langle \rho_3, \sigma_3 \rangle$$

$$\langle r \geq y, \rho_3, \sigma_5 \rangle \Downarrow \mathit{false} \quad \sigma_3 = \sigma_5$$

$$\frac{\langle E, \rho, \sigma \rangle \Downarrow \mathit{false}}{\langle \text{while } (E) \{ C \}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma \rangle}$$

Ambiente

x	l_x
y	l_y

ρ

Memoria

l_x	8
l_y	5

σ

Ambiente

x	l_x
y	l_y
q	l_q

ρ_1

Memoria

l_x	8
l_y	5
l_q	0

σ_1

Ambiente

x	l_x
y	l_y
q	l_q
r	l_r

ρ_3 ρ_2

Memoria

l_x	8
l_y	5
l_q	0
l_r	8

σ_2

$$\frac{}{\langle \text{skip;}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma \rangle} \qquad \frac{\langle C_1, \rho, \sigma \rangle \rightarrow \langle \rho_1, \sigma_1 \rangle \quad \langle C_2, \rho_1, \sigma_1 \rangle \rightarrow \langle \rho_2, \sigma_2 \rangle}{\langle C_1\ C_2, \rho, \sigma \rangle \rightarrow \langle \rho_2, \sigma_2 \rangle}$$

$$\frac{\langle E, \rho, \sigma \rangle \Downarrow v \quad l \notin \sigma}{\langle T\ \text{Id} = E;, \rho, \sigma \rangle \rightarrow \langle \rho[\text{Id} \mapsto l], \sigma[l \mapsto v] \rangle} \qquad \frac{\langle E, \rho, \sigma \rangle \Downarrow v \quad \rho(\text{Id}) = l}{\langle \text{Id} := E;, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma[l \mapsto v] \rangle}$$

$$\frac{\langle C, \rho, \sigma \rangle \rightarrow \langle \rho_1, \sigma_1 \rangle}{\langle \{ C \}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma_1 \rangle} \qquad \frac{\langle E, \rho, \sigma \rangle \Downarrow \mathit{true} \quad \langle C_1, \rho, \sigma \rangle \rightarrow \langle \rho_1, \sigma_1 \rangle}{\langle \text{if } (E) \{ C_1 \} \text{ else } \{ C_2 \}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma_1 \rangle}$$

$$\frac{\langle E, \rho, \sigma \rangle \Downarrow \mathit{false}}{\langle \text{while } (E) \{ C \}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma \rangle} \qquad \frac{\langle E, \rho, \sigma \rangle \Downarrow \mathit{false} \quad \langle C_2, \rho, \sigma \rangle \rightarrow \langle \rho_2, \sigma_2 \rangle}{\langle \text{if } (E) \{ C_1 \} \text{ else } \{ C_2 \}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma_2 \rangle}$$

$$\frac{\langle E, \rho, \sigma \rangle \Downarrow \mathit{true} \quad \langle C, \rho, \sigma \rangle \rightarrow \langle \rho_1, \sigma_1 \rangle \quad \langle \text{while } (E) \{ C \}, \rho, \sigma_1 \rangle \rightarrow \langle \rho_2, \sigma_2 \rangle}{\langle \text{while } (E) \{ C \}, \rho, \sigma \rangle \rightarrow \langle \rho, \sigma_2 \rangle}$$

Ambiente

x	l_x
y	l_y
q	l_q
r	l_r

ρ_3

Memoria

l_x	8
l_y	5
l_q	1
l_r	8

σ_4

Ambiente

x	l_x
y	l_y
q	l_q
r	l_r

ρ_3

Memoria

l_x	8
l_y	5
l_q	1
l_r	3

σ_5 σ_3

Stato finale!

Terminazione vs Divergenza

L'introduzione dei cicli nei nostri programmi introduce anche la possibilità che i nostri programmi divergano, senza produrre alcun risultato finale

Che cosa è la divergenza?

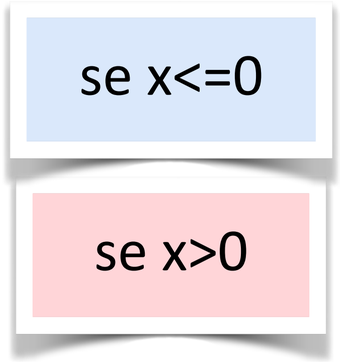
La divergenza, si riferisce alla situazione in cui un programma o un ciclo non termina mai e continua a eseguire istruzioni indefinitamente

All'opposto, la terminazione è una proprietà desiderabile nei programmi, poiché garantisce che il programma completi la sua esecuzione e produca un risultato in un tempo finito

Terminazione vs Divergenza

Troviamo qualche argomento per decidere sotto quali condizioni i seguenti comandi terminano o divergono

```
while (x > 0) {
    y := y + 1;
}
```



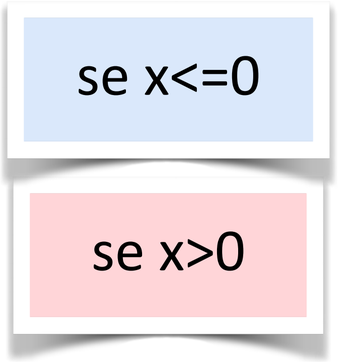
$$\langle C, \rho, \sigma \rangle \to \langle \rho, \sigma' \rangle$$

$$\langle x > 0, \rho, \sigma \rangle \Downarrow true$$

$$\langle y := y+1;, \rho, \sigma \rangle \to \langle \rho, \sigma_1 \rangle$$

$$\langle C, \rho, \sigma_1 \rangle \to \langle \rho, \sigma' \rangle$$

```
while (x > 0) {
    x := x + 1;
}
```



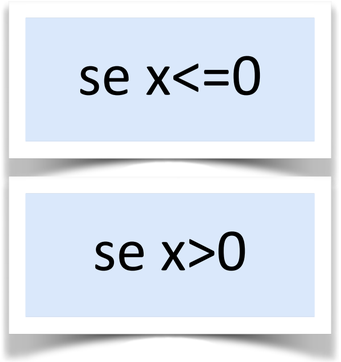
$$\langle C, \rho, \sigma \rangle \to \langle \rho, \sigma' \rangle$$

$$\langle x > 0, \rho, \sigma \rangle \Downarrow true$$

$$\langle x := x+1;, \rho, \sigma \rangle \to \langle \rho, \sigma_1 \rangle$$

$$\langle C, \rho, \sigma_1 \rangle \to \langle \rho, \sigma' \rangle$$

```
while (x > 0) {
    x := x - 1;
}
```



$$\langle C, \rho, \sigma \rangle \to \langle \rho, \sigma' \rangle$$

$$\langle x > 0, \rho, \sigma \rangle \Downarrow true$$

$$\langle x := x-1;, \rho, \sigma \rangle \to \langle \rho, \sigma_1 \rangle$$

$$\langle C, \rho, \sigma_1 \rangle \to \langle \rho, \sigma' \rangle$$

Condizione necessaria per la terminazione

- La guardia contenga almeno una variabile
- e il corpo contenga un assegnamento per quella variabile!

Purtroppo è solo una condizione necessaria, non sufficiente

```
while (x > 0) {
    x := x + 1;
}
```

```
while (x > 0) {
    x := x - 1;
}
```

RISULTATO DI IMPOSSIBILITÀ:

non esiste un algoritmo che per ogni programma sia in grado di decidere se termina o no

Altri costrutti iterativi

In Mao vediamo solo il costrutto while (E) {C} ma ci sono diversi costrutti iterativi che i linguaggi di programmazione mettono a disposizione

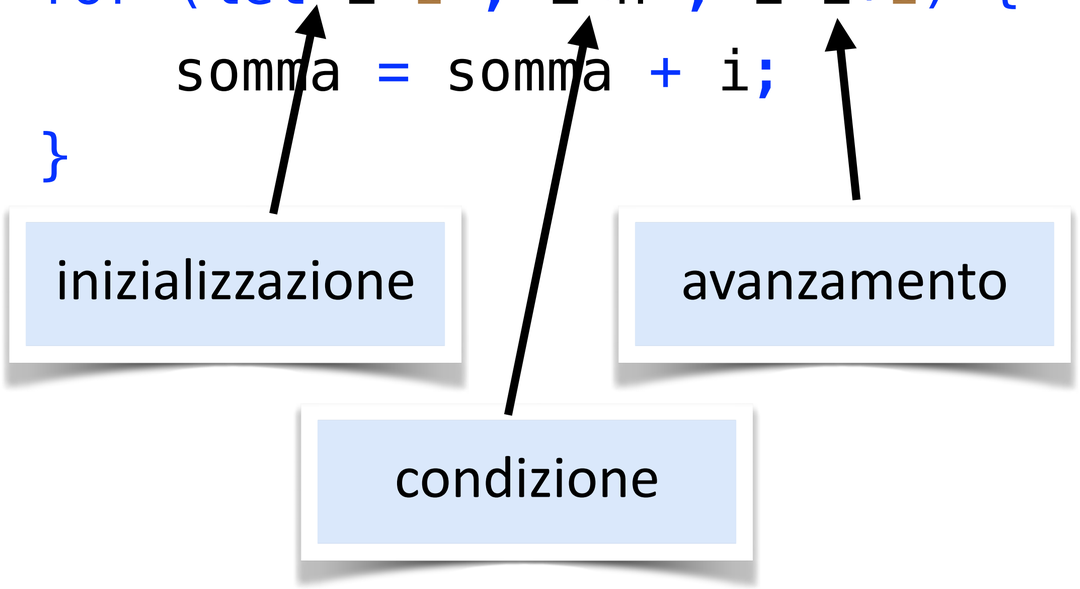
Ne vediamo brevemente alcuni perchè saranno utili per scrivere programmi compatti in pseudocodice

Ognuno di questi costrutti può essere espresso usando cicli while e quindi non ne definiamo esplicitamente la semantica

Altri costrutti iterativi: il ciclo for

Il costrutto for è particolarmente compatto, di immediata lettura e informativo:
in una riga sappiamo quante iterazioni sono necessarie
(infatti ne raccomandiamo l'uso proprio in queste situazioni)

```
// in JavaScript
let somma=0;
for (let i=1 ; i<n ; i=i+1) {
    somma = somma + i;
}
```




```
// in MiniMao
int somma=0;
int i=1;
while (i<n) {
    somma := somma + i;
    i := i + 1;
}
```

Altri costrutti iterativi: il ciclo do-while

Il costrutto do-while si usa quando il corpo del ciclo deve essere eseguito almeno una volta: prima eseguiamo il corpo e solo dopo valutiamo la guardia

```
// in JavaScript, col do-while
let età;
do {
    età = parseInt(prompt("Anni?"));
} while (età < 0)
```

```
// in JavaScript, col while
let età = parseInt(prompt("Anni?"));
while (età < 0) {
    età = parseInt(prompt("Anni?"));
}
```