Thursday Nov. 29

Lecture 23

wp Rules

$$wp(x := e, R) = R[x := e]$$

$$wp(\texttt{if } B \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ end}, R) = \begin{pmatrix} B \Rightarrow wp(S_1, R) \\ \wedge \\ \neg B \Rightarrow wp(S_2, R) \end{pmatrix}$$

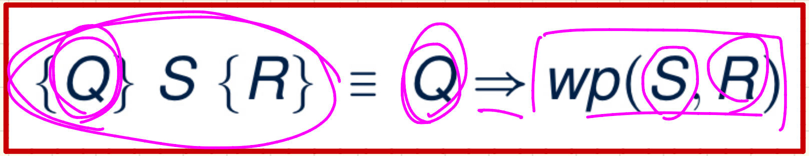
$$wp(S_1 \; ; \; S_2, R) = wp(S_1, wp(S_2, R))$$

$wp(S_1 ; \underline{wp(S_2 ; R)})$

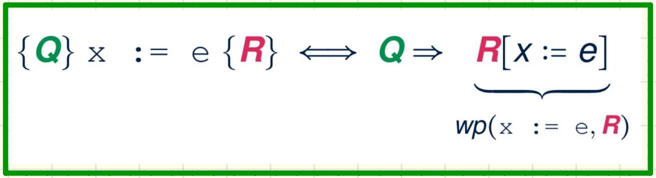
$\underline{S_1} \mid \underline{wp(S_2 ; R)} \; \underline{S_2} \; \underline{R}$

the post-cond of S_1

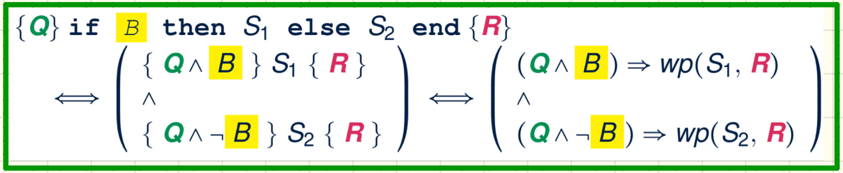
Proof Rules



$$\{Q\}\ S\ \{R\} \equiv Q \Rightarrow wp(S, R)$$

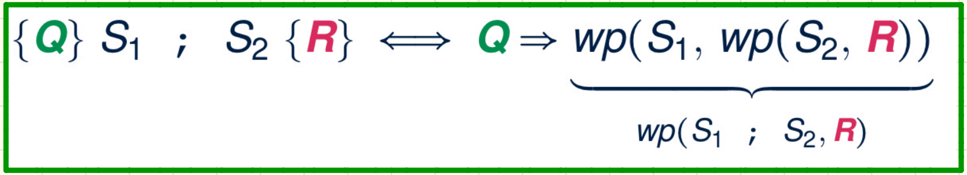


$$\{Q\}\ \texttt{x := e}\ \{R\} \iff Q \Rightarrow \underbrace{R[x := e]}_{wp(\texttt{x := e},\, R)}$$



$$\{Q\}\ \textbf{if}\ B\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{end}\ \{R\}$$

$$\iff \begin{pmatrix} \{\ Q \wedge B\ \}\ S_1\ \{\ R\ \} \\ \wedge \\ \{\ Q \wedge \neg B\ \}\ S_2\ \{\ R\ \} \end{pmatrix} \iff \begin{pmatrix} (Q \wedge B) \Rightarrow wp(S_1, R) \\ \wedge \\ (Q \wedge \neg B) \Rightarrow wp(S_2, R) \end{pmatrix}$$

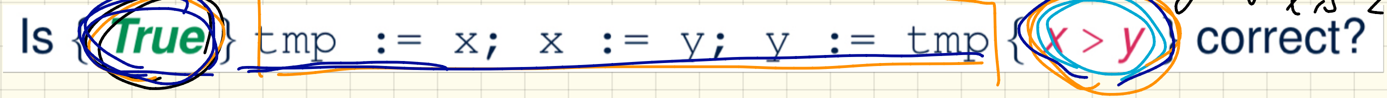


$$\{Q\}\ S_1\ ;\ S_2\ \{R\} \iff Q \Rightarrow \underbrace{wp(S_1, wp(S_2, R))}_{wp(S_1\ ;\ S_2,\, R)}$$

Correctness of Program : Sequential Composition

Step 2 : Q True $\Rightarrow$ $y > x$ wp(S;R)

↳ No e.g. y is 1, x is 2



Is {True} tmp := x; x := y; y := tmp {x > y} correct?

Goal: True $\Rightarrow$ wp(tmp := x; x := y; y := tmp, $x > y$)

Step 1: wp(tmp := x; x := y; y := tmp, $x > y$)

= {wp rule for ;}

wp(tmp := x, wp(x := y; y := tmp, $x > y$))

= {wp rule for ;}

wp(tmp := x, wp(x := y, wp(y := tmp, $x > y$)))

= {wp for :=}

wp(tmp := x, wp(x := y, $x >$ tmp))

= {wp for :=}

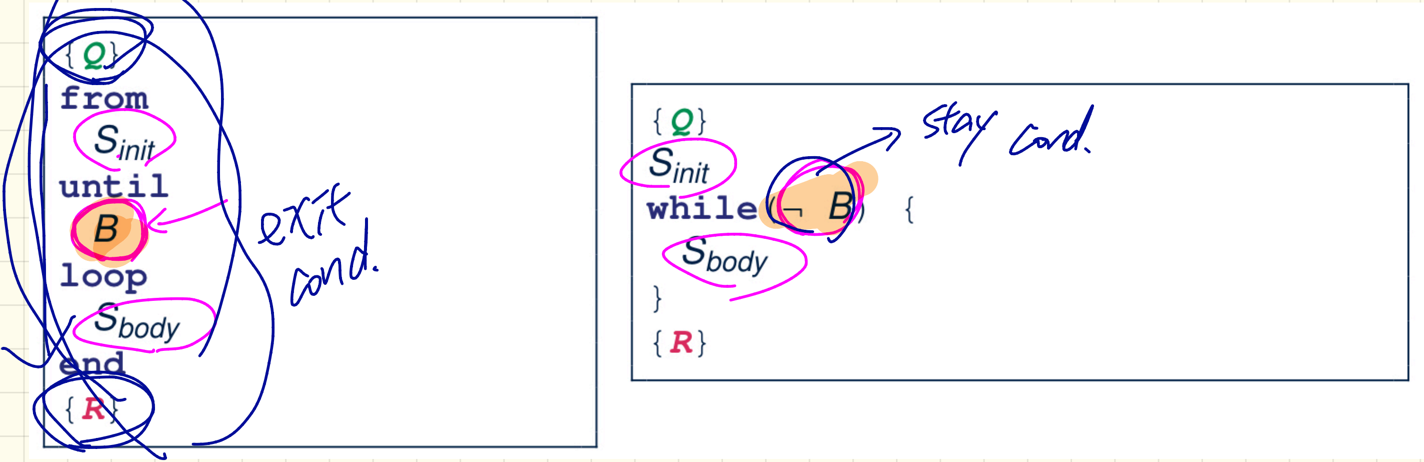
wp(tmp := x, $y >$ tmp) = {wp for :=} $y > x$

$$\boxed{x > 4 \Rightarrow x > 3}$$

it is obvious that

$$x > 4 \Rightarrow x > 3$$

Loops : Eiffel vs. Java



```
{$Q$}
from
  $S_{init}$
until
  $B$
loop
  $S_{body}$
end
{$R$}
```

exit cond.

```
{$Q$}
$S_{init}$
while ($\neg$ $B$) {
  $S_{body}$
}
{$R$}
```

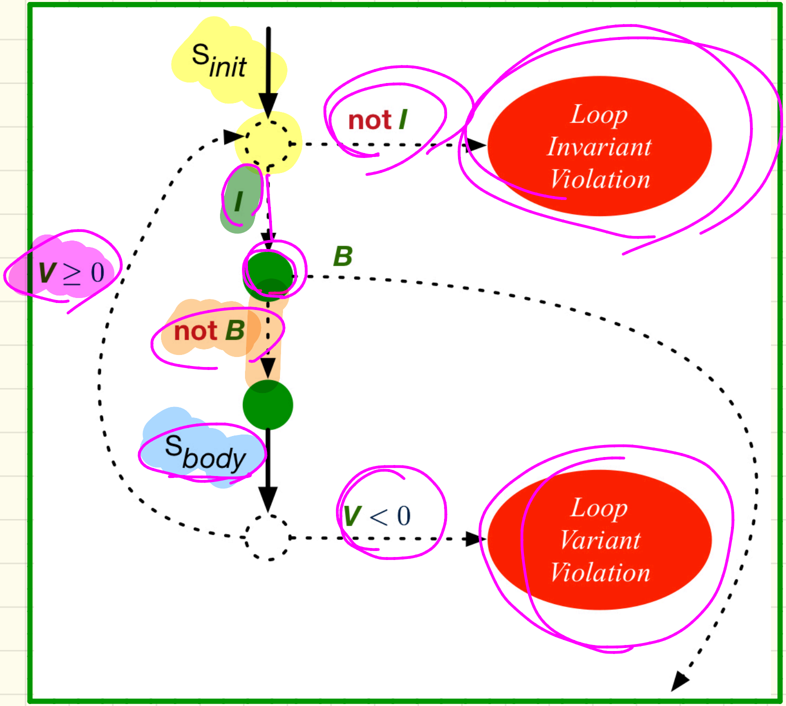
stay cond.

Contracts of Loops

Syntax

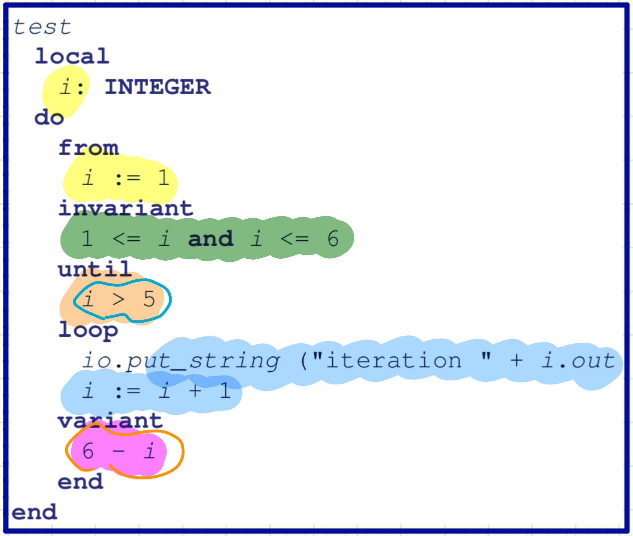
```
from
  S_init
invariant
  invariant_tag: I
until
  B
loop
  S_body
variant
  variant_tag: V
end
```

Runtime Checks



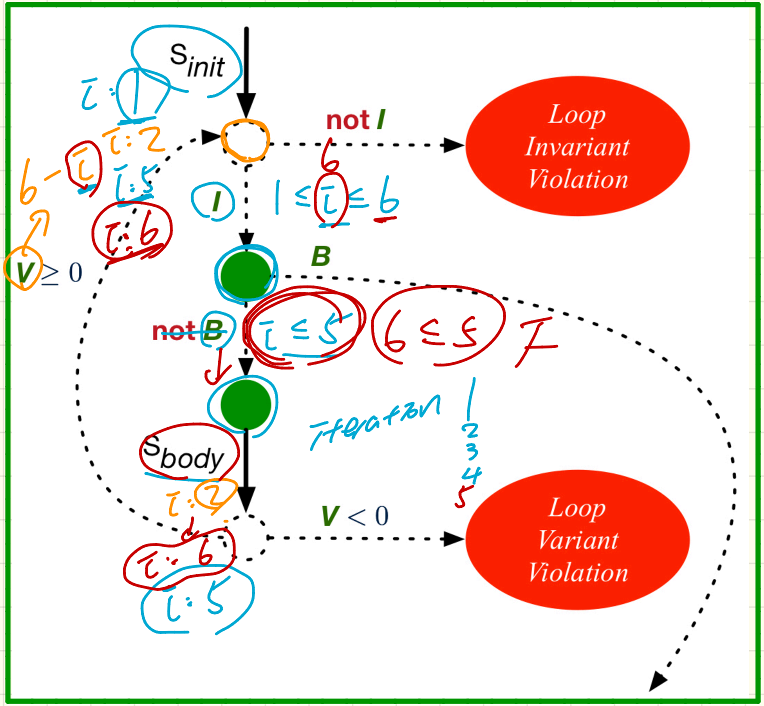
Contracts of Loops: Example

Example

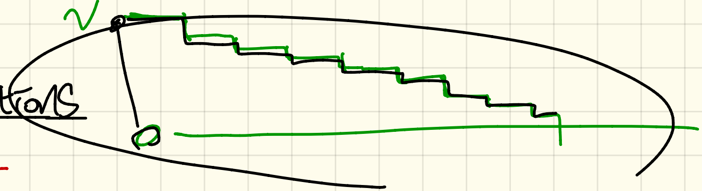


```
test
  local
    i: INTEGER
  do
    from
      i := 1
    invariant
      1 <= i and i <= 6
    until
      i > 5
    loop
      io.put_string ("iteration " + i.out
      i := i + 1
    variant
      6 - i
    end
end
```

Runtime Checks



Contracts of Loops: Violations



iteration	i	6 - i
	1	
1st	3	3
2nd	5	1
3rd	7	-1

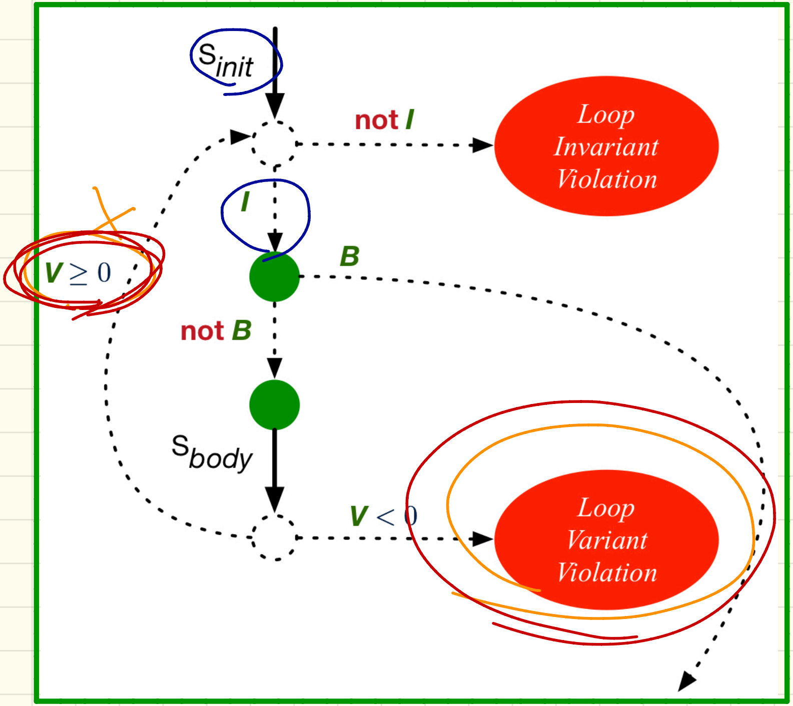
Example

```
test
  local
    i: INTEGER
  do
    from
      i := 1
    invariant
      1 <= i and i <= 6
    until
      i > 5
    loop
      io.put_string ("iteration " + i.out
      ~~i := i + 1~~
    variant
      6 - i
    end
end
```

invariant → correctness

variant → termination

Runtime Checks

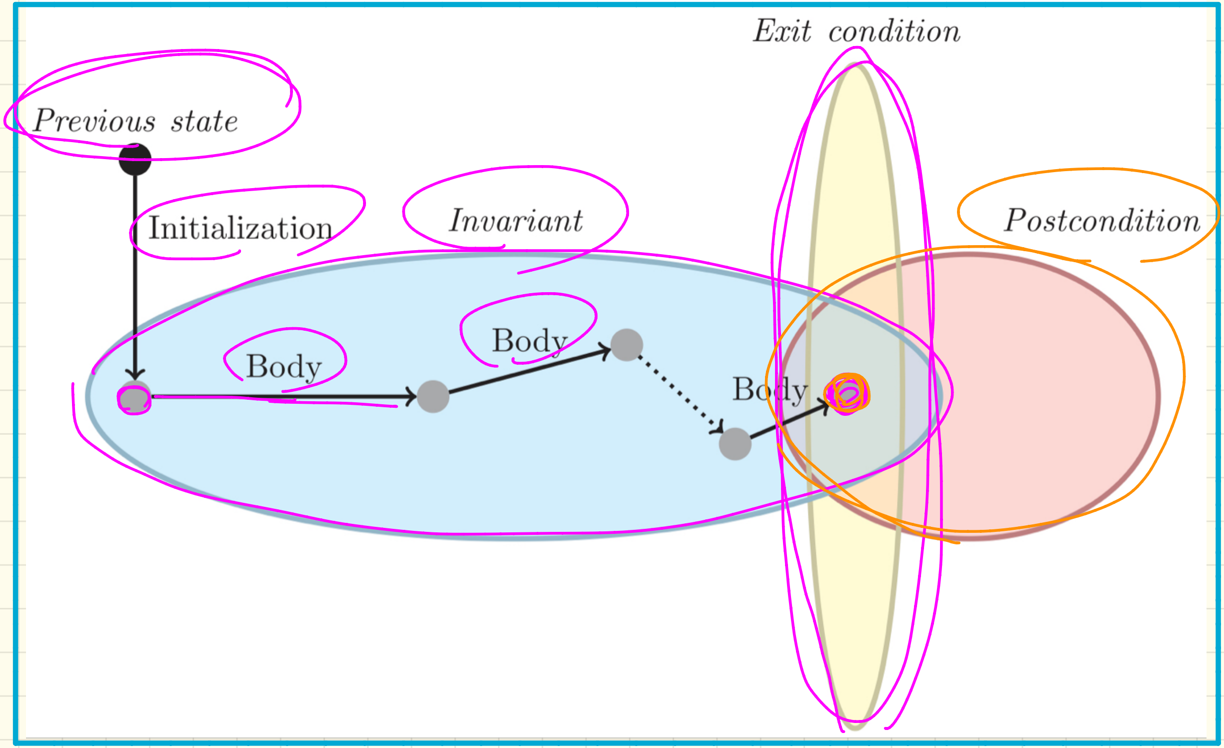


Invariant Violation: $1 \leq i \leq 5$

Variant Violation: $5 - i$

Skipping Loop Body: $i > 0$

Contracts of Loops: Visualization



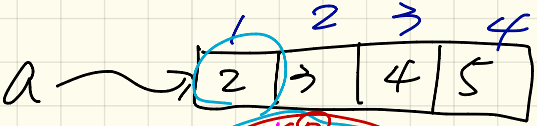
```
sum (a: ARRAY[INT]): INT
  local
    i : INT            Result
  from
    i := a.lower
    Result := 0
  invariant
    ?
  until
    i > a.upper
  loop
    sum := sum + a[i]
    i := i + 1
  end
ensure
```

$$\text{Result} = \sum_{j=a.lower}^{a.upper} a[j]$$

i : 1

Result := 0

i : a.upper + 1



$$\text{Result} = \sum_{j=1}^{\cancel{\times}\,0} a[j] \quad 0 \quad [1, 0]$$

0 $\quad i-1 \quad$ 2

$$\text{Result} = \sum_{j=a.lower}^{i-1} a[j] \qquad a.upper + 1$$

$$\text{Result} = \sum_{j=a.lower}^{i-1} a[j]$$

a.upper