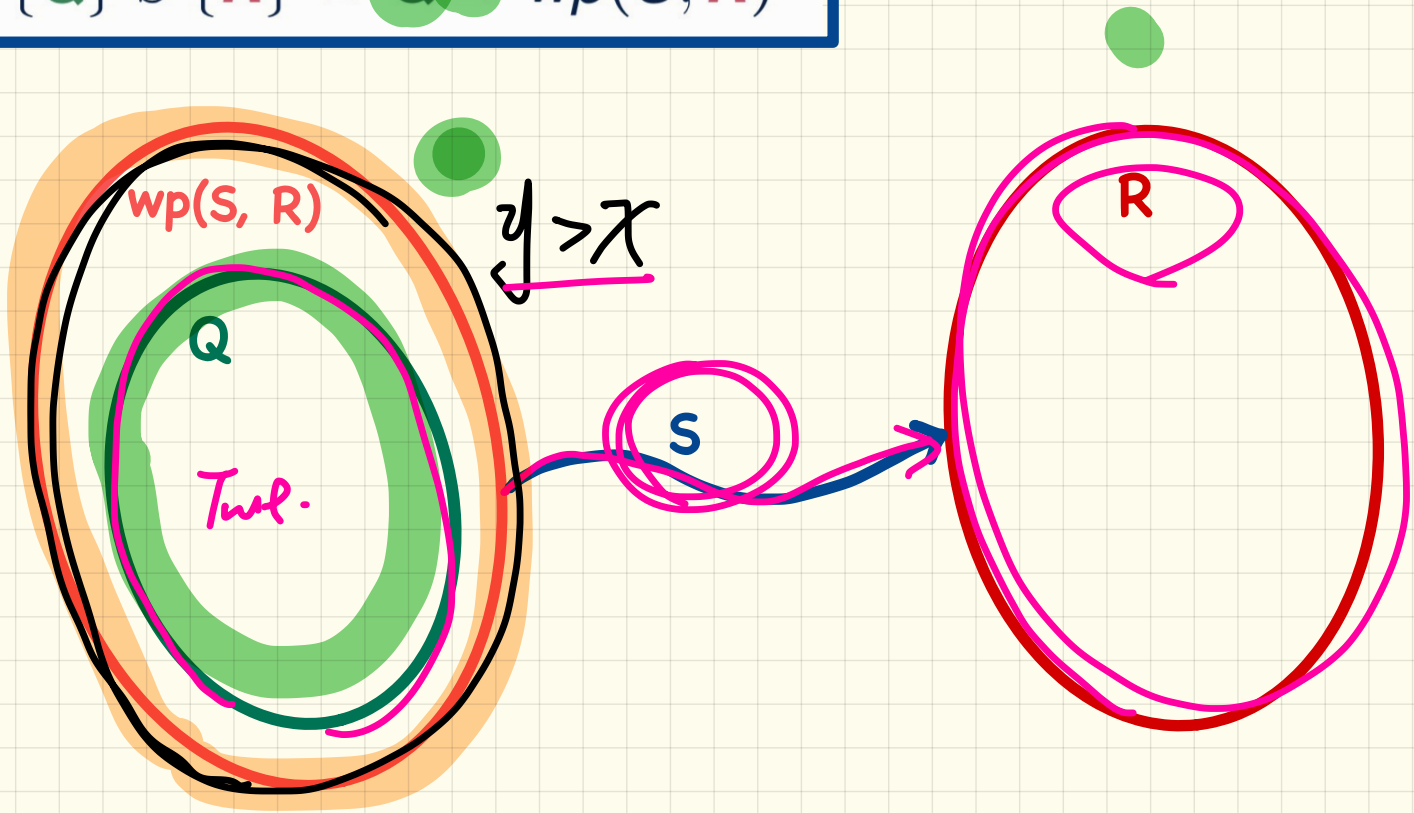
Lecture 24

Tuesday December 3

Hoare Triple as a Predicate

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



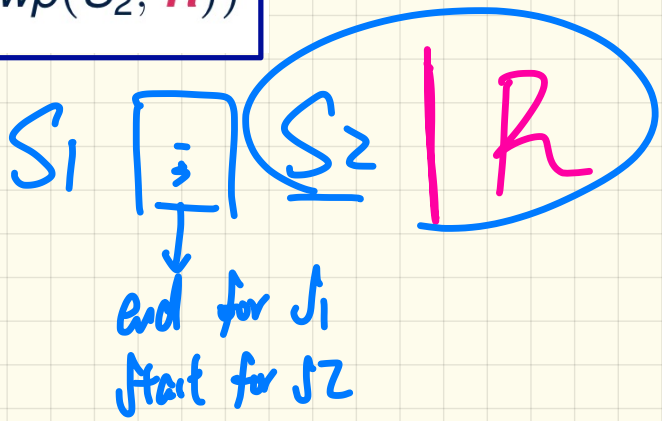
Rules of Weakest Precondition: Summary

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

$wp(S_1, wp(S_2, R))$

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

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



Correctness of Programs: Sequential Composition

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

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

$= \{$ wp rule of ; $\}$

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

$= \{$ wp rule of ; $\}$

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

$= \{$ wp rule of := $\}$

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

$= \{$ wp rule of := $\}$

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

$y > x$

$x > y [y := tmp]$

$x > tmp [x := y]$

$y > tmp [tmp := x]$

② $Q \Rightarrow wp$

$Tmp \Rightarrow y > x$

C.P.

$y = x$

Proof Rules using Weakest Precondition

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

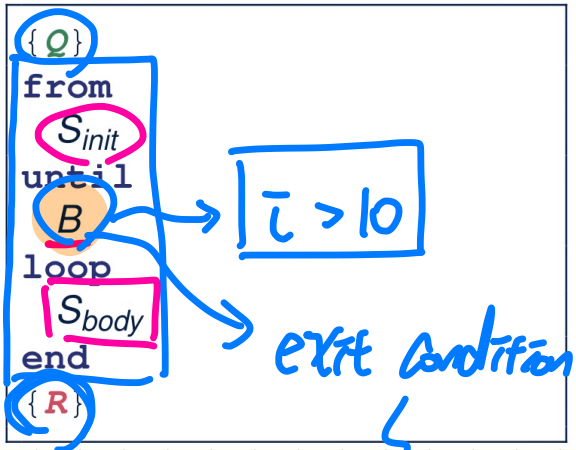
$$\{Q\}\ \texttt{x := e}\ \{R\} \iff Q \Rightarrow \underbrace{R[x := e]}_{wp(\texttt{x := e}, R)}$$

$$\{Q\}\ \texttt{if } B \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ end}\ \{R\}$$
$$\iff \left(\begin{array}{l} \{\, Q \wedge B \,\}\ S_1\ \{\, R \,\} \\ \wedge \\ \{\, Q \wedge \neg B \,\}\ S_2\ \{\, R \,\} \end{array} \right) \iff \left(\begin{array}{l} (Q \wedge B) \Rightarrow wp(S_1, R) \\ \wedge \\ (Q \wedge \neg B) \Rightarrow wp(S_2, R) \end{array} \right)$$

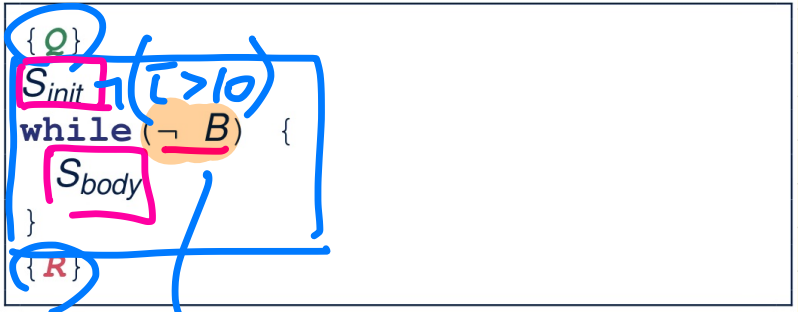
$$\{Q\}\ S_1\ ;\ S_2\ \{R\} \iff Q \Rightarrow \underbrace{wp(S_1, wp(S_2, R))}_{wp(S_1\ ;\ S_2, R)}$$

loop.

Loops: Eiffel vs. Java



as soon as $i > 10$ is true, exit.



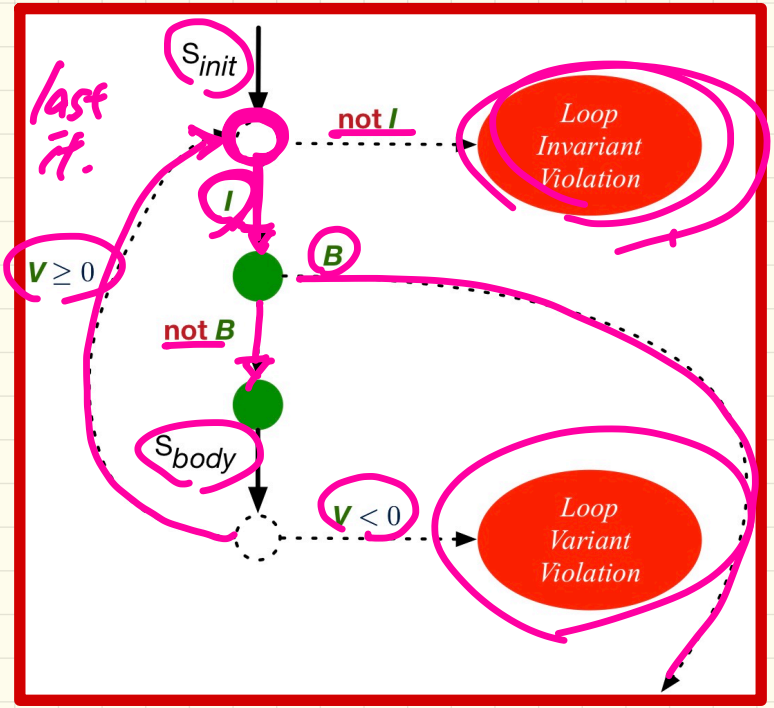
as long as $\neg(i > 10)$ is true, stay.

Contracts of Loops

Syntax

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

Runtime Checks

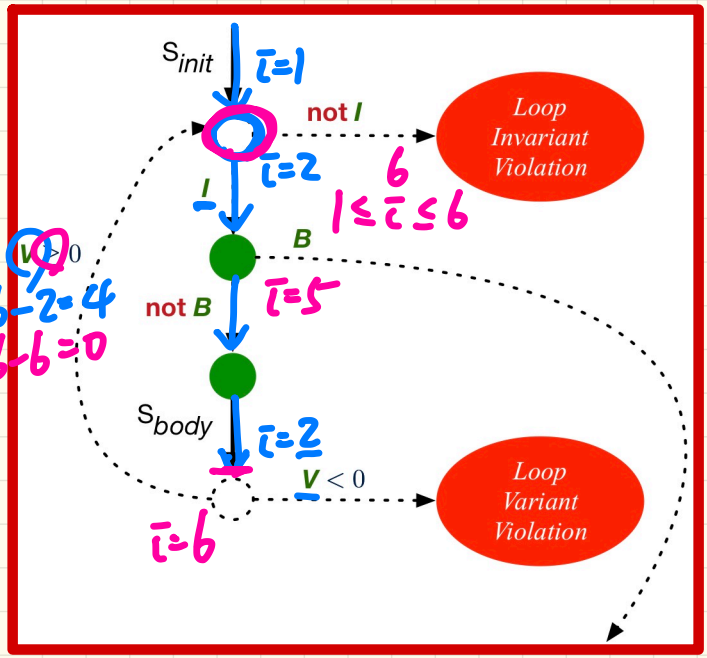


Contracts of Loops: Example

Syntax

```
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



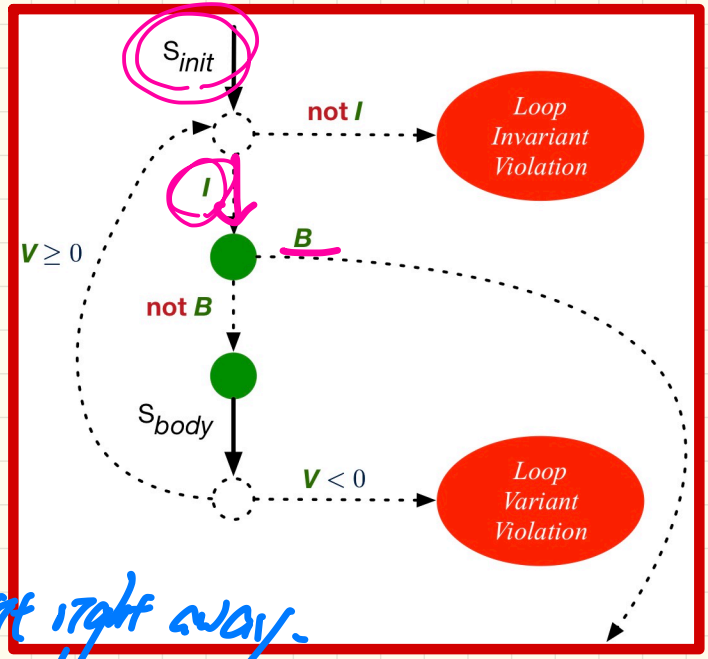
LI when exiting the loop should imply postcondition.

Contracts of Loops: Violations

Syntax

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

Runtime Checks

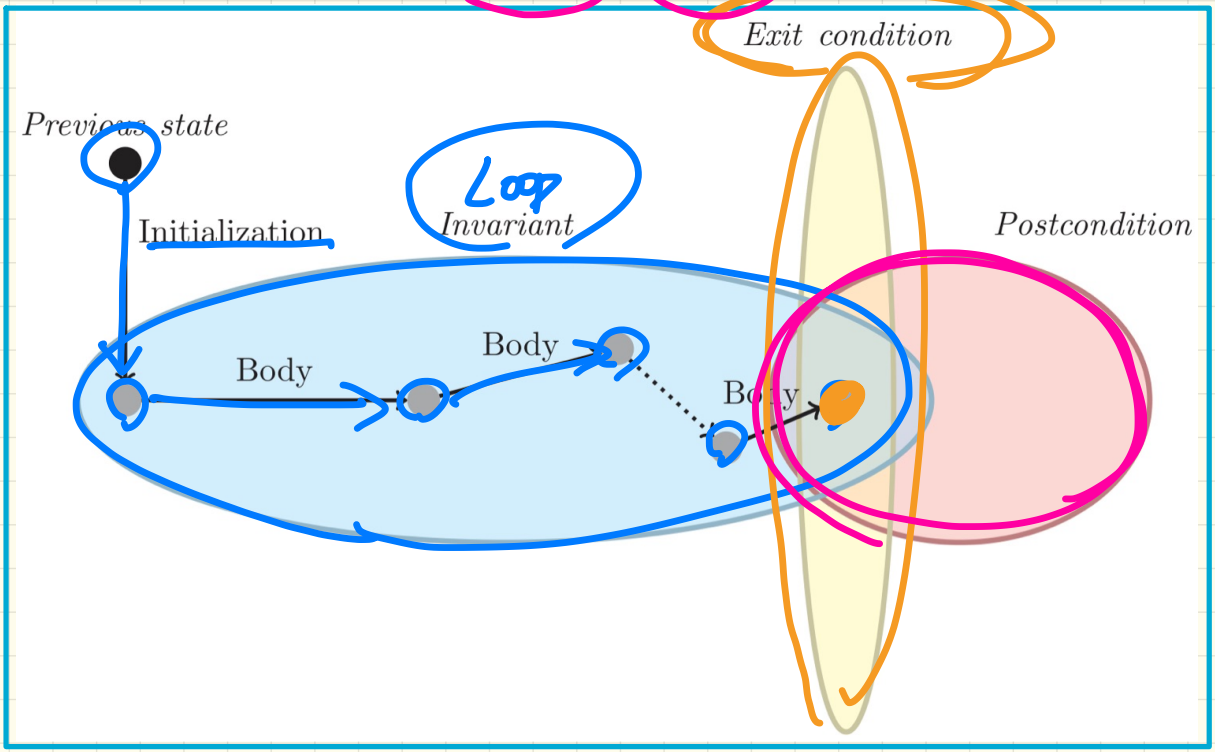


exit condition: i > 0

invariant: 1 <= i <= 5

variant: 5 - i

Contracts of Loops: Visualization



Contracts of Loops: Loop Invariant

```
find_max (a: ARRAY [INTEGER]): INTEGER
  local i: INTEGER
  do
    from
      i := a.lower; Result := a[i]
    invariant
      loop_invariant: ∀j | a.lower ≤ j ≤ i • Result ≥ a[j]
      across a.lower |..| i as j all Result >= a [j.item] end
    until
      i > a.upper
    loop
      if a [i] > Result then Result := a [i] end
      i := i + 1
    variant
      loop_variant: a.upper - i + 1
    end
  ensure
    correct_result: -- ∀j | a.lower ≤ j ≤ a.upper • Result ≥ a[j]
      across a.lower |..| a.upper as j all Result >= a [j.item]
  end
end
```

$\forall j \mid a.lower \le j \le i-1 \bullet Result \ge a[j]$

LI:

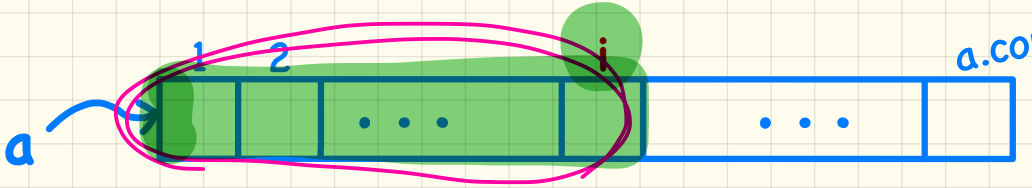
across a.lower |..| i-1 as j

all

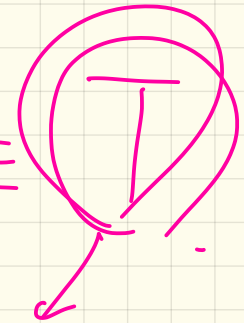
Result ≥ a[j]

end

Invariant: **Result** stores the max of the array scanned **so far**.



$\forall x \mid F \cdot P(x) \equiv \top$



∵ no violation witness can be found.

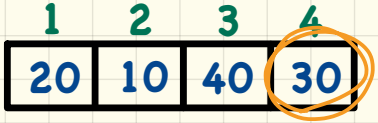
Finding Max: Version 1

```
find_max (a: ARRAY [INTEGER]): INTEGER
  local i: INTEGER
  do
    from
      i := a.lower ; Result := a[i]
    invariant
      loop_invariant: -- ∀j | a.lower ≤ j ≤ i • Result ≥ a[j]
        across a.lower |..| i as j all Result >= a [j.item] end
    until
      i > a.upper
    loop
      if a [i] > Result then Result := a [i] end
      i := i + 1
    variant
      loop_variant: a.upper - i + 1
    end
  ensure
    correct_result: -- ∀j | a.lower ≤ j ≤ a.upper • Result ≥ a[j]
      across a.lower |..| a.upper as j all Result >= a [j.item]
  end
end
```

wrong.

AFTER ITERATION	i	Result	LI	EXIT (*i* > *a.upper*)?	LV
Initialization					
1st					
2nd					

Finding Max: Version 2




```
find_max (a: ARRAY [INTEGER]): INTEGER
  local i: INTEGER
  do
    from
      i := a.lower ; Result := a[i]
    invariant
      loop_invariant: -- ∀j | a.lower ≤ j < i • Result ≥ a[j]
        across a.lower |..| (i - 1) as j all Result >= a [j.item] end
    until
      i > a.upper
    loop
      if a [i] > Result then Result := a [i] end
      i := i + 1
    variant
      loop_variant: a.upper - i
    end
  ensure
    correct_result: -- ∀j | a.lower ≤ j ≤ a.upper • Result ≥ a[j]
      across a.lower |..| a.upper as j all Result >= a [j.item]
  end
end
```

last iteration:

i = a.upper + 1

After Iteration	i	Result	*LI*	Exit (*i* > *a.upper*)?	*LV*
Initialization	1	20	✓	×	–
1st	2	20	✓	×	2
2nd	3	20	✓	×	1
3rd	4	40	✓	×	0
4th	●	●	●	●	●

Correct Loops: Proof Obligations

```
{Q}     from
          S_init
        invariant
          I
        until
          B
        loop
          S_body
        variant
          V
        end          {R}
```

- A loop is ***partially*** *correct* if:
  - Given precondition **Q**, the initialization step S_{init} establishes ***LI*** I.

    $$\{Q\}\ S_{init}\ \{I\}$$

  - At the end of S_{body}, if not yet to exit, ***LI*** I is maintained.

    $$\{I \wedge \neg B\}\ S_{body}\ \{I\}$$

  - if ready to exit and ***LI*** I maintained, postcondition **R** is established.

    $$I \wedge B \Rightarrow R$$

- A loop *terminates* if:
  - Given ***LI*** I, and not yet to exit, S_{body} maintains ***LV*** V as non-negative.

    $$\{I \wedge \neg B\}\ S_{body}\ \{V \geq 0\}$$

  - Given ***LI*** I, and not yet to exit, S_{body} decrements ***LV*** V.

    [redacted]

Correct Loops: Proof Obligations

```
find_max (a: ARRAY [INTEGER]): INTEGER
  local i: INTEGER
  do
    from
      i := a.lower ; Result := a[i]
    invariant
      loop_invariant: ∀j | a.lower ≤ j < i • Result ≥ a[j]
    until
      i > a.upper
    loop
      if a [i] > Result then Result := a [i] end
      i := i + 1
    variant
      loop_variant: a.upper − i + 1
    end
  ensure
    correct_result: ∀j | a.lower ≤ j ≤ a.upper • Result ≥ a[j]
  end
end
```

maintaining I

Initialization:

Before Termination:

Upon Termination:

Non-Negative Variant:

Decreasing Variant:

Prove

Establishment of Loop Invariant:

{ *True* }

```
i := a.lower
Result := a[i]
```

{ $\forall j \mid a.lower \leq j < i \bullet Result \geq a[j]$ }

Prove

Establishment of Postcondition upon Termination:

$$(\forall j \mid a.lower \leq j < i \bullet Result \geq a[j]) \wedge i > a.upper \Rightarrow \forall j \mid a.lower \leq j \leq a.upper \bullet Result \geq a[j]$$

Hint: Rewrite j < i and i > a.upper using >=

Hint: Identify i, j, a.upper on the number line.

Prove

Loop Variant Stays Non-Negative Before Exit:

$\{\ (\forall j \mid a.lower \le j < i \bullet Result \ge a[j]) \land \neg(i > a.upper)\ \}$

```
if a [i] > Result then Result := a [i] end
i := i + 1
```

$\{\ a.upper - i + 1 \ge 0\ \}$