Lecture 12

Part 1

Correctness - Motivating Examples

Program Correctness: Example (1)

```
class FOO
  i: INTEGER
  increment_by_9
    require
      i > 3
    do
      i := i + 9
    ensure
      i > 13
    end
end
```

specification

4, 5, 6, 7, ...

↳ fix 1: i > 4 strengthen precondition (stronger)

14, 15, 16.

postcondition violation

precondition i > 3 is too weak (4 is allowed)

↳ this program is not correct.

Fix 2: weaken the postcondition : i > 12

Correctness of (relative) Program:

i > 3 ~~4~~, 5, 6, 7 → i > 4

implementation satisfies spec.

Given valid input (satisfying precond) implementation. executing the implementation will

(1) terminate

(2) upon termination, the postcondition is satisfied.

Program Correctness: Example (2)

```
class FOO
  i: INTEGER
  increment_by_9
    require
      i > 5
    do
      i := i + 9
    ensure
      i > 13
    end
end
```

6, 7, 8, 9, ...

Assume: fixed (not subject to change)

14 5

14

↳ Correct. (assuming that 5 should be excluded).

Is i > 5 too strong? ∵ 5 will not cause postcond violat.

Guiding Principle

Precondition cannot be too weak (i.e. it does not allow any input value that can cause a postcondition violation).

whether an input should be included is up to the req.

→ all values satisfying the precondition will guarantee the satisfaction of postcondition (after executing the implementation).

Lecture 12

Part 2

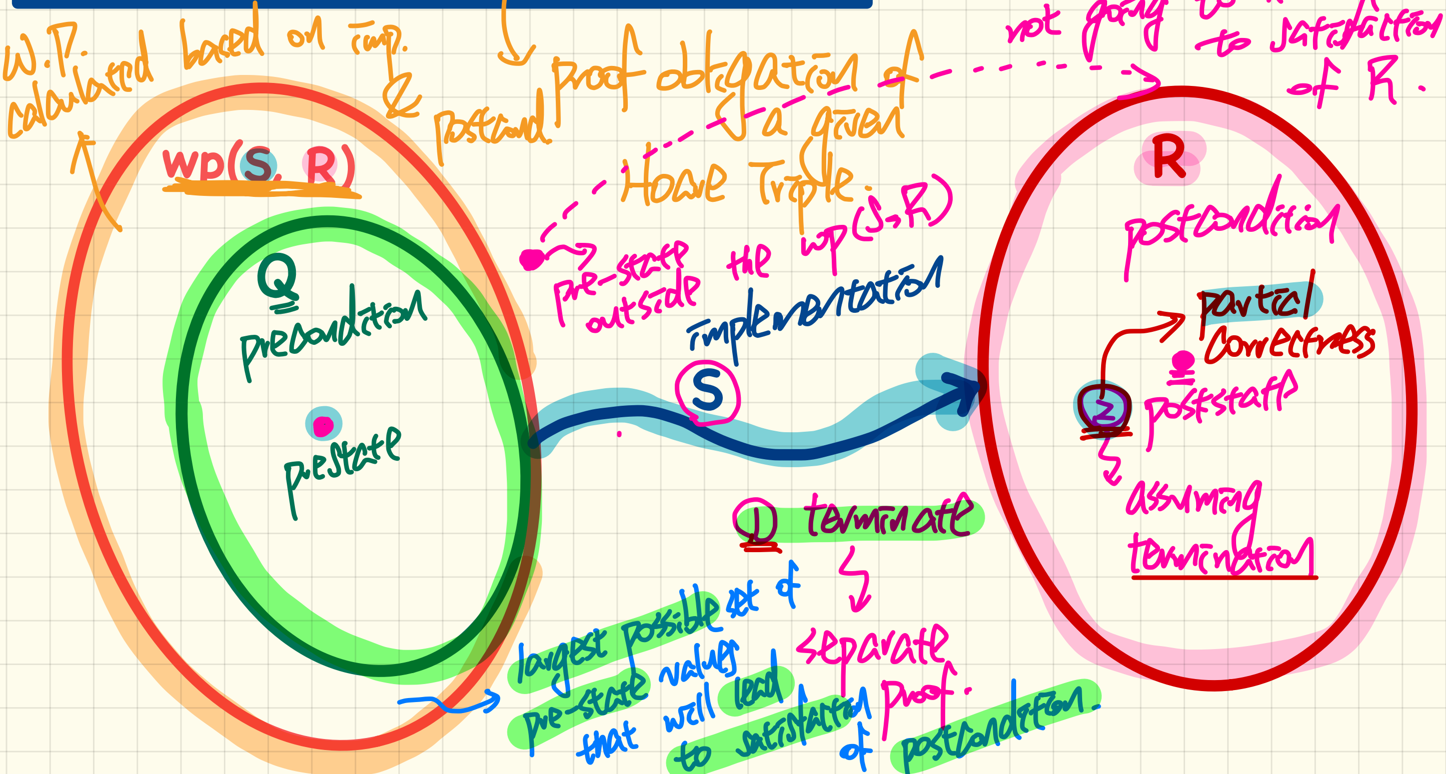
Hoare Triple and Weakest Precondition

Hoare Triple as a Predicate

② : partial correctness

① + ② : total correctness

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



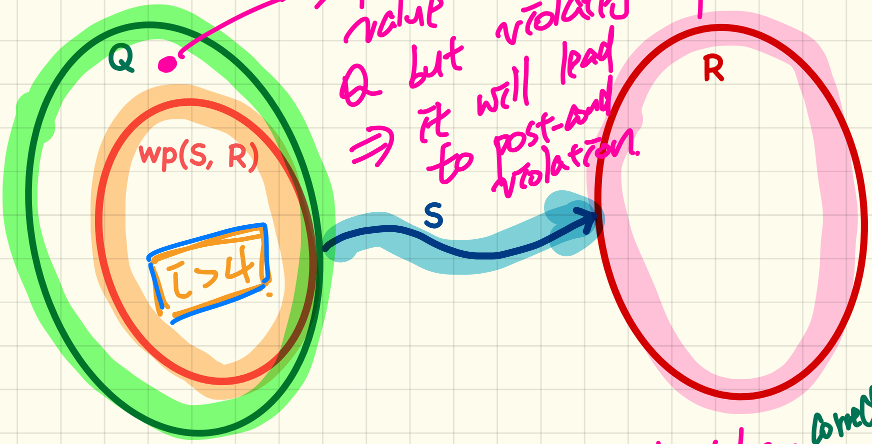
Program Correctness: Revisiting Example (1)

```
class FOO
  i: INTEGER
  increment_by_9
    require
        i > 3
    do
      i := i + 9
    ensure
      i > 13
    end
end
```

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

$\cancel{i > 3 \Rightarrow i > 4}$

$i = 4$: pre-state value that satisfies Q but violates wp(S, R) ⇒ it will lead to post-cond violation.



$\{i > 3\}\ i := i + 9\ \{i > 13\}$ → Hoare Triple ↳ predicate

→ tautology: correct.

→ counter-example: incorrect

Program Correctness: Revisiting Example (2)

```
class FOO
  i: INTEGER
  increment_by_9
    require
      i > 5
    do
      i := i + 9
    ensure
      i > 13
    end
end
```

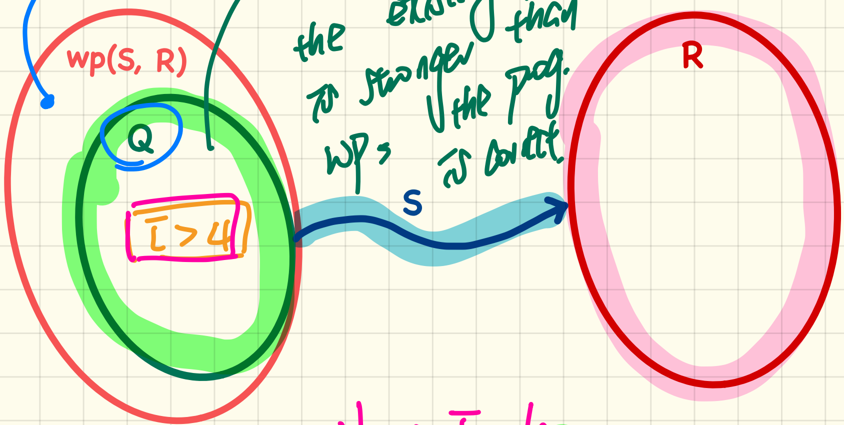
$$\{Q\}\ S\ \{R\} \equiv Q \Rightarrow wp(S, R)$$

all allowed input values will satisfy wp.

$i = 5$

as long as the existing precondition is stronger than wp, the prog. is correct

$i>5 \Rightarrow i>4$



$\{i>5\}\ i := i+9\ \{i>13\}$ → Hoare Triple

↳ can be proved as a tautology

↳ correct.

<u>Lecture 12</u>

<u>Part 3</u>

Rules of wp Calculus

Rules of Weakest Precondition: Assignment

$$\boxed{wp(x := e,\ R) = R[x := e]}$$

replace every free occurrence of x by e.

massage this post-state expression into an equivalent one for the w.p. → precondition

$x+1 > 0$

wp??

involves pre-state values only.

$x := e$

R

e.g. $x > 0$

post-state value of x

e.g. $x := x + 1$

pre-state value of x

$\{Q\}\ x := e\ \{R\} \equiv Q \Rightarrow \boxed{wp(x := e, R)}$

$R[x:e]$.

Correctness of Programs: Assignment (1)

What is the weakest precondition for a program `x := x + 1` to establish the postcondition $x > x_0$?

$$\{??\}\ \texttt{x := x + 1}\ \{x > x_0\}$$

post-state pre-state

$wp(x := x + 1, x > x_0)$

In order for the design of ?? to be correct:

$\boxed{?? \Rightarrow True}$

any precondition is ok.

= { wp rule of assignment }

$x > x_0 [x := x + 1]$

pre-state value: $x_0 + 1$

= { replace each free occurrence of x by $x_0 + 1$ }

w.p. for $x := x + 1$ to establish $x > x_0$.

$\cancel{x_0} + 1 > \cancel{x_0}$ = { arithmetic } = $\boxed{True}$ w.p.

Correctness of Programs: Assignment (2)

What is the weakest precondition for a program `x := x + 1` to establish the postcondition $x > x_0$?

$$\{??\}\ \texttt{x := x + 1}\ \{x = 23\}$$

Rules of Weakest Precondition: Conditionals

$$wp(\textbf{if } B \textbf{ then } S1 \textbf{ else } S2 \textbf{ end}, R)$$

wp(S: PROGRAM; r: PRED)

- :=
- if
- ;
- loop

arbitrarily complicated

$\land$?

$B \Rightarrow wp(S1, R)$ ✓

$\land$. $\lor$?

$\land$?

$\neg B \Rightarrow wp(S2, R)$ ✓

Rules of Weakest Precondition: Conditionals

wp(if B **then** S1 **else** S2 **end**, R)

$\boxed{\begin{array}{l} B \Rightarrow wp(S1, R) \\ \lor \\ \neg B \Rightarrow wp(S2, R) \end{array}}$ vs. $\boxed{\begin{array}{l} B \Rightarrow wp(S1, R) \\ \land \\ \neg B \Rightarrow wp(S2, R) \end{array}}$??

$False[x := x+1] \equiv False$

Consider:

$\boxed{wp(\textbf{if } y > 0 \textbf{ then } \underline{x := x + 1} \textbf{ else } \underline{x := x - 1} \textbf{ end}, \underline{x >= 0})}$

$T \Rightarrow F \equiv F$

$\boxed{y>0} \Rightarrow wp(x := x+1, x >= 0)$ F

$\land$ $\cancel{\lor}$ F

$\boxed{\neg(y>0) \Rightarrow wp(x := x-1, x >= 0)}$ T

F → correct result by using ∧

T → by using disjunction, misleading result

$y=1, x=-4$

Correctness of Programs: Conditionals

Is this program correct?

$\{x > 0 \wedge y > 0\}$

```
if x > y then
  bigger := x ; smaller := y
else
  bigger := y ; smaller := x
end
```

$\{bigger \geq smaller\}$

intermediate state

$S_1 \;\; ; \;\; S_2 \;\; R$

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

$\|$

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

$\boxed{wp(S_2, R)}$

↳ starting condition for S_2 to establish R

↳ ending condition for S_1 to establish

Correctness of Programs: Sequential Composition

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

Step 1: calculate wp

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

$= \{ wp \text{ rule for seq. comp} \}$

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

$= \{ wp \text{ rule for seq. comp.} \}$

$x > tmp$

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

$y > tmp$

$= \{ wp \text{ rule for } ; \} \; wp(temp := x, wp(x := y, x > tmp))$

$= \{ wp \text{ rule for } ; \} \; wp(temp := x, y > tmp) = \{ wp \text{ rule for } ; \} \; y > x$

Step 2

Prove or disprove:

$True \Rightarrow y > x$

$\equiv \; y > x$

not tautology. Counter-example.

x	4
y	3

prog not correct.

Rules of Weakest Precondition: Summary

$$wp(\mathrm{x} \;\mathbf{:=}\; \mathrm{e},\ \boldsymbol{R}) = \boldsymbol{R}[x := e]$$

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

$$wp(S_1\ \mathbf{;}\ S_2,\ \boldsymbol{R}) = wp(S_1,\ wp(S_2,\ \boldsymbol{R}))$$

Proof Rules using
Weakest Precondition

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

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

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

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

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

Lecture 12

Part 4

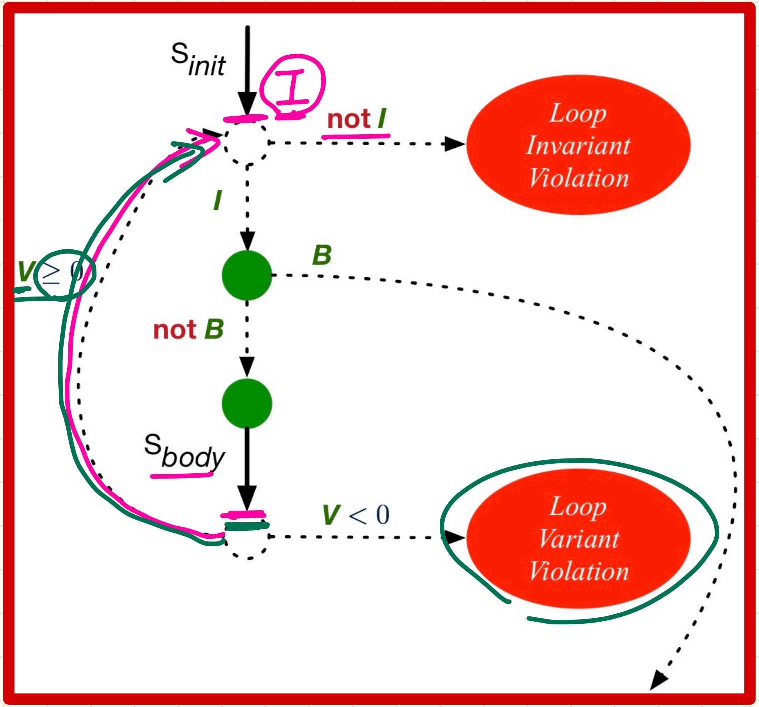
Contracts of Loops

Contracts of Loops

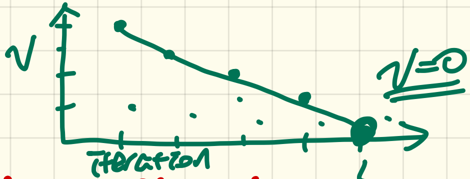
Syntax

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

Runtime Checks



Contracts of Loops: Example



at the end of 5th iteration

Syntax

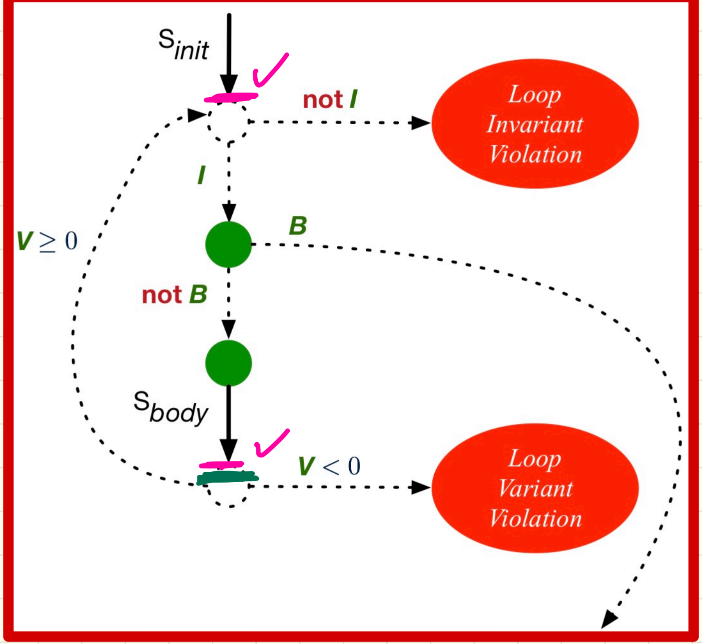
```
test
  local
    i: INTEGER
  do
    from
      i := 1
    invariant
      1 <= i and i <= 6
    until
      i > 5
    loop
      io.put_string ("iteration " + i.out)
      i := i + 1
    variant
      6 - i
    end
end
```

6 − 2 4 ≥ 0
6 − 3 3 ≥ 0
6 − 6 0 ≥ 0

iteration 1
2
3
4
5

6 → not output.

Runtime Checks

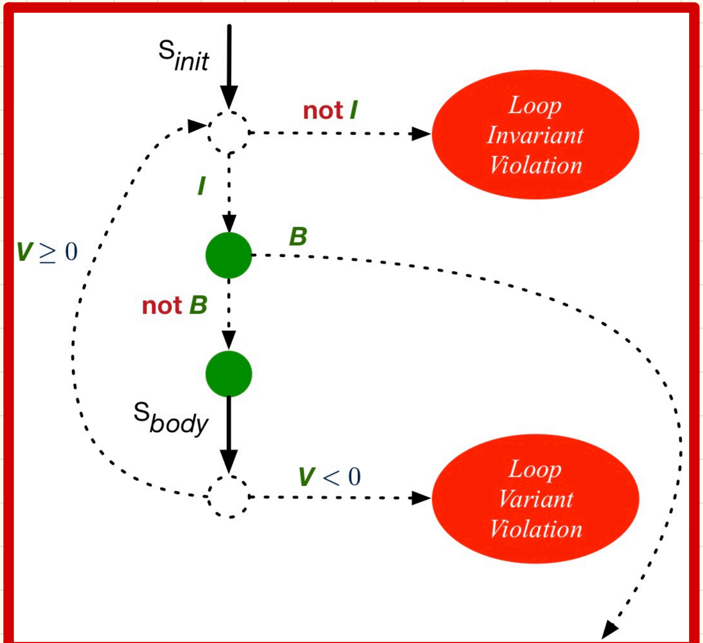


Contracts of Loops: Violations

Syntax

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

Runtime Checks



invariant: 1 <= i <= 5

↳ At the end of 5th iteration:

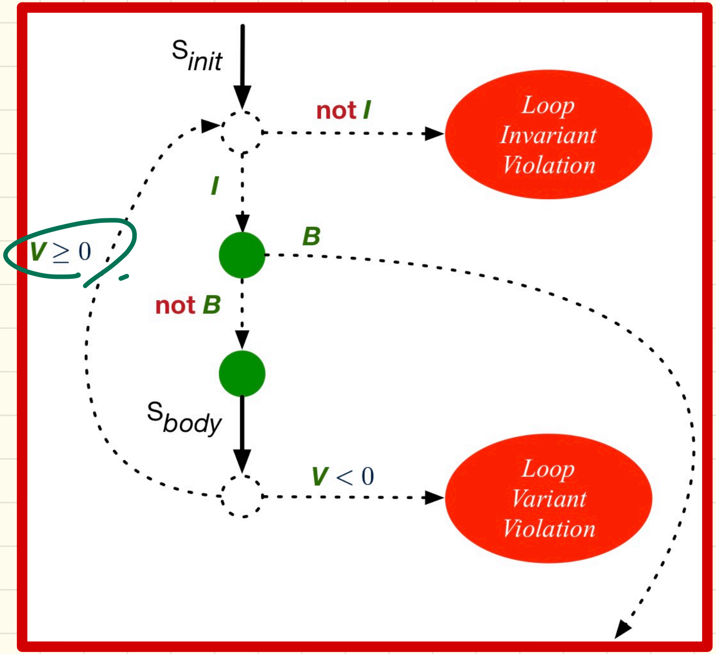
$1 \leq \overset{6}{i} \leq 5 \Rightarrow$ LI violation.

Contracts of Loops: Violations

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
  5 ~~6~~ - i
    end
end
```

Runtime Checks

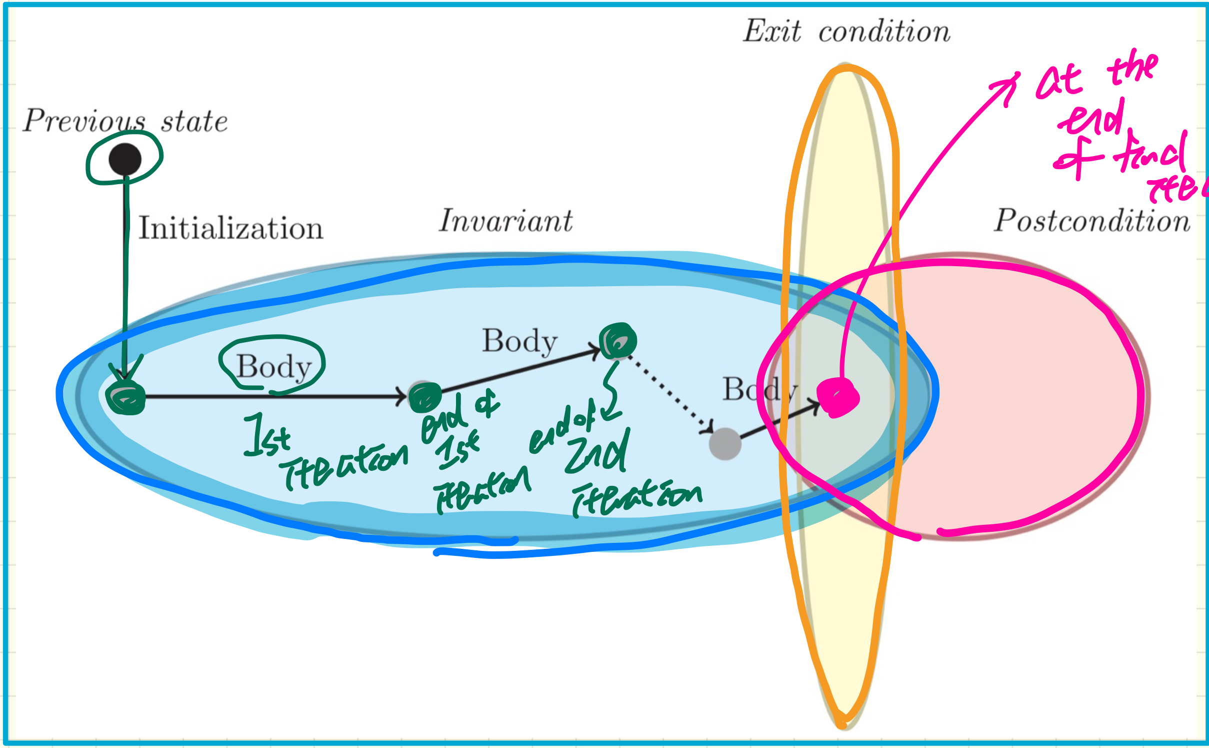


variant: 5 - i

↳ At the end of 5th iteration

LV: $5 - 6 = -1 \geq 0 \equiv F \Rightarrow$ LV violation

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 - 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
```

① After init before 1st iteration.

$\forall j : \boxed{1 \leq j \leq 0} \bullet$ ____

F

T

② At the end of 1st iteration: $i = 2$

$\forall j : 1 \leq j \leq 1 \bullet$ ____

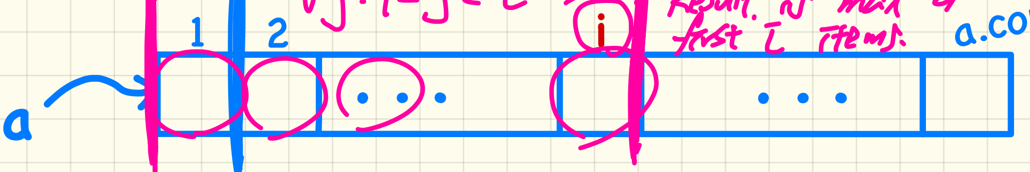
At the end of ith iteration, what's the value of i?

$i + 1$

③ At the end of ith iteration: loop counter $i+1$

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

$\forall j : 1 \leq j \leq i \Rightarrow$ Result is max of first i items.



Finding Max: Version 1

1	2	3	4
20	10	40	30

```
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
```

Iteration	Result	i	LI
Init	20	1	1≤j≤1
1	20	2	1≤j≤2
2	20	3	1≤j≤3

LI violation
∵ Result 20 is not max of the first 3 items.

After Iteration	i	Result	*LI*	Exit ($i > a.upper$)?	*LV*
Initialization	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
1st	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
2nd	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

Finding Max: Version 2

1	2	3	4
20	10	40	30

```
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
```

$4 - 5 = -1 \geq 0 \equiv F$

⇓

LV violation

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	5	●	●	●	-1

Lecture 12

Part 5

Correctness Proofs of Loops

Correct Loops: Proof Obligations

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

$V < V_0$

V: LV value at the end of current iteration

V_0: LV value in the beginning of current iteration

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

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

    $\{I \land \neg B\}\ S_{body}\ \{I\}$ — $\{I \land \neg B\}\ S_{body}\ \{I\}$
  - If ready to exit and *LI* I maintained, postcondition R is established.

    $I \land B \Rightarrow R$ — $B \land I \Rightarrow R$
- A loop ***terminates*** if:
  - Given *LI* I, and not yet to exit, S_{body} maintains *LV* V as non-negative.

    $\{I \land \neg B\}\ S_{body}\ \{V \geq 0\}$ — $\{I \land \neg B\}\ S_{body}\ \{V \geq 0\}$
  - Given *LI* I, and not yet to exit, S_{body} decrements *LV* V.

    $\{I \land \neg B\}\ S_{body}\ \{V < V_0\}$ — $\{I \land \neg B\}\ S_{body}\ \{V < V_0\}$

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
```

Initialization:

$\{True\}\ i := a.lower ;$

$Result := a[i]$

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

Before Termination:

Upon Termination:

Non-Negative Variant:

Decreasing Variant: