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
```

Program Correctness: Example (2)

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

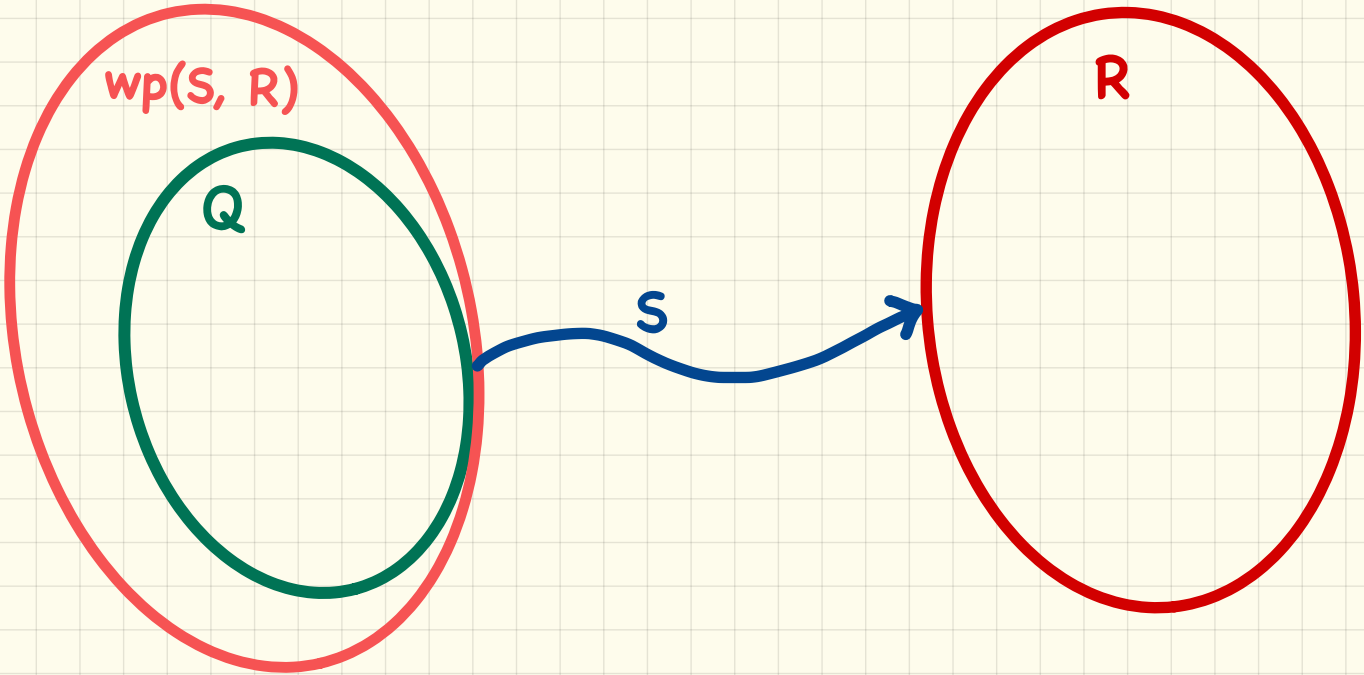
Lecture 12

Part 2

Hoare Triple and Weakest Precondition

Hoare Triple as a Predicate

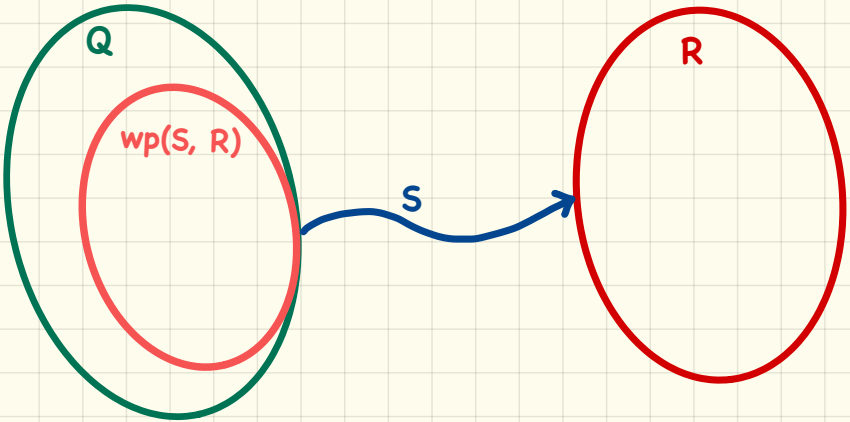
$$\{Q\}\ S\ \{R\} \equiv Q \Rightarrow wp(S, R)$$



Program Correctness: Revisiting Example (1)

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

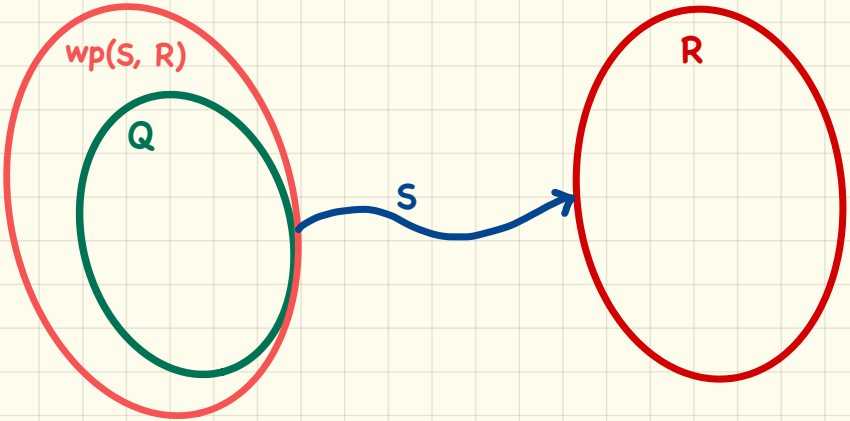
$$\{Q\}\ S\ \{R\} \equiv Q \Rightarrow wp(S, R)$$



Program Correctness: Revisiting Example (2)

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

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



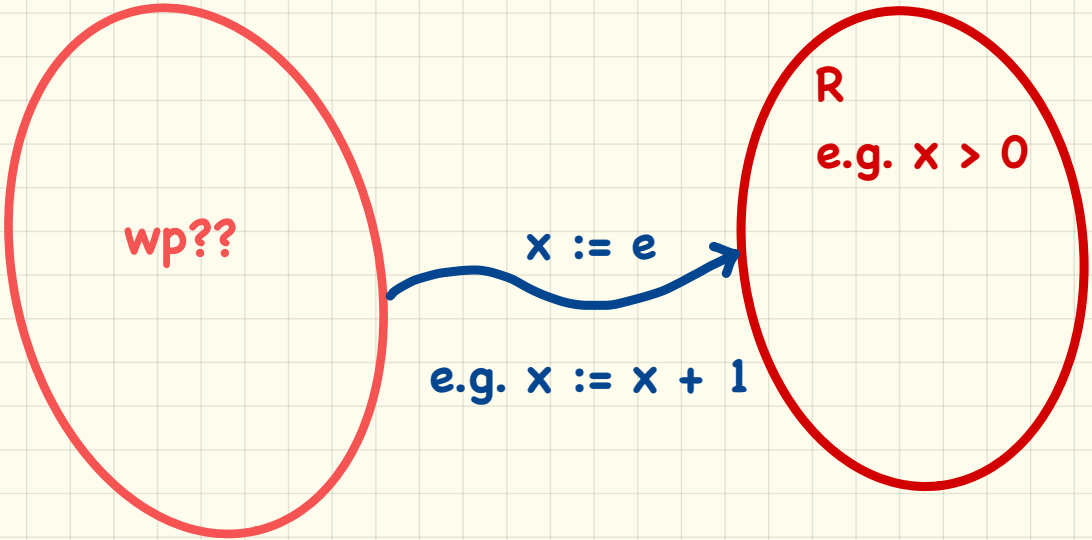
Lecture 12

Part 3

Rules of wp Calculus

Rules of Weakest Precondition: Assignment

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



Correctness of Programs: Assignment (1)

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

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

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

Rules of Weakest Precondition: Conditionals

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

$$B \Rightarrow wp(S1, R) \quad \vee \quad \neg B \Rightarrow wp(S2, R)$$

vs.

$$B \Rightarrow wp(S1, R) \quad \wedge \quad \neg B \Rightarrow wp(S2, R)$$

??

Consider:

wp(**if** y > 0 **then** x := x + 1 **else** x := x - 1 **end**, x >= 0)

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
```

$\{bigger \geq smaller\}$

Correctness of Programs: Sequential Composition

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

Rules of Weakest Precondition: Summary

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

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

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

Proof Rules using Weakest Precondition

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

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

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

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

Lecture 12

Part 4

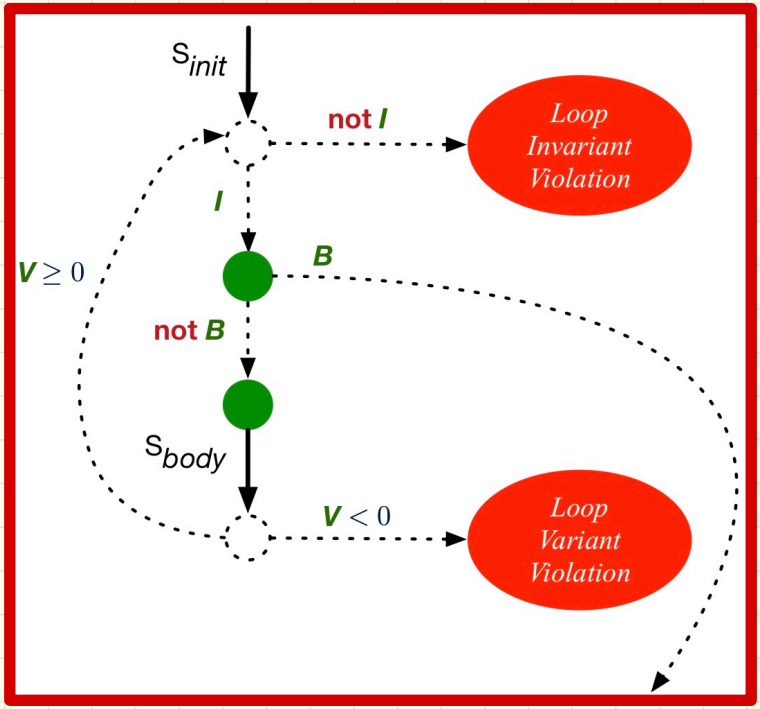
Contracts of Loops

Contracts of Loops

Syntax

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

Runtime Checks

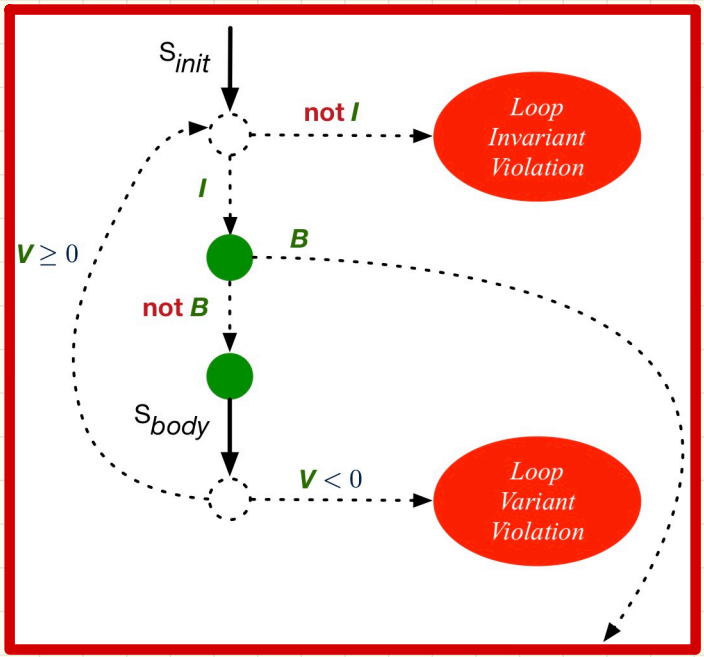


Contracts of Loops: Example

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

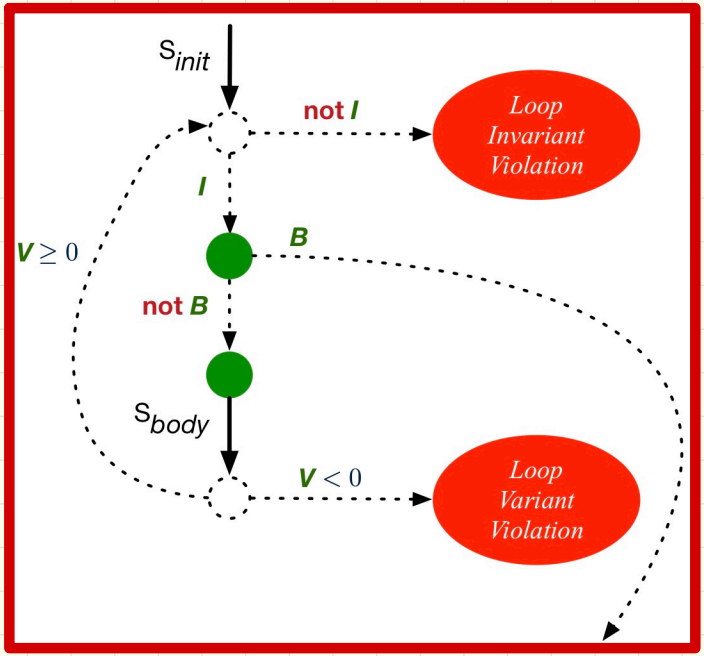


Contracts of Loops: Violations

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
      6 - i
    end
end
```

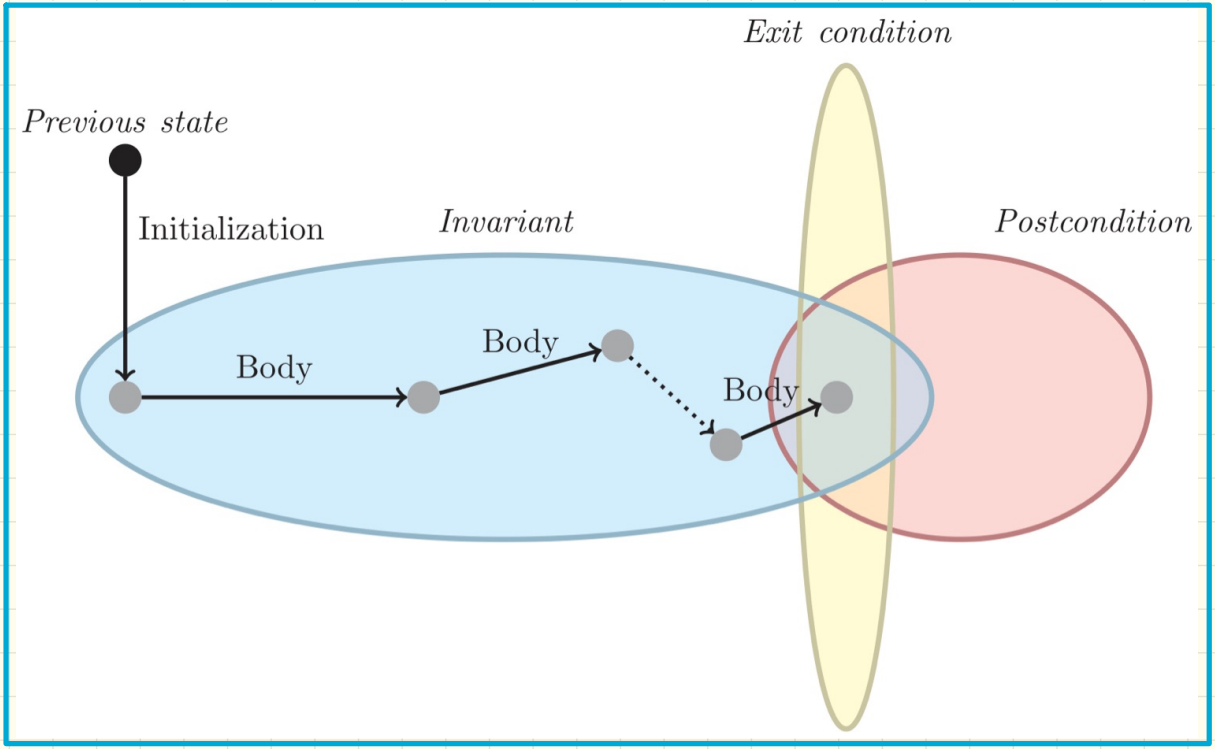
Runtime Checks



invariant: 1 <= i <= 5

variant: 5 - i

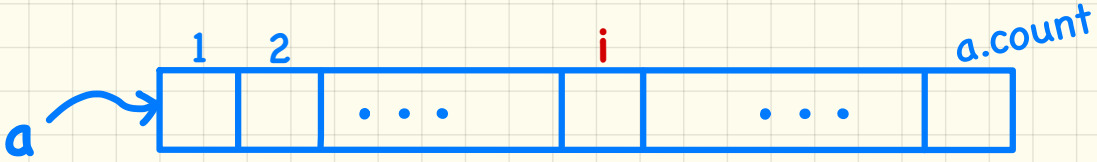
Contracts of Loops: Visualization



Contracts of Loops: Loop Invariant

```
find_max (a: ARRAY [INTEGER]): INTEGER
  local i: INTEGER
  do
    from
      i := a.lower ; Result := a[i]
    invariant
      [redacted]
    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
```

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



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
```

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

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
```

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	●	●	●	●	●

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}
```

- A loop is ***partially*** *correct* if:
  - Given precondition ***Q***, the initialization step S_{init} establishes ***LI*** *I*.
  - At the end of S_{body}, if not yet to exit, ***LI*** *I* is maintained.
  - If ready to exit and ***LI*** *I* maintained, postcondition ***R*** is established.
- A loop *terminates* if:
  - Given ***LI*** *I*, and not yet to exit, S_{body} maintains ***LV*** *V* as non-negative.
  - Given ***LI*** *I*, and not yet to exit, S_{body} decrements ***LV*** *V*.

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.

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

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:

Before Termination:

Upon Termination:

Non-Negative Variant:

Decreasing Variant: