Lecture 22 - April 2

Program Verification

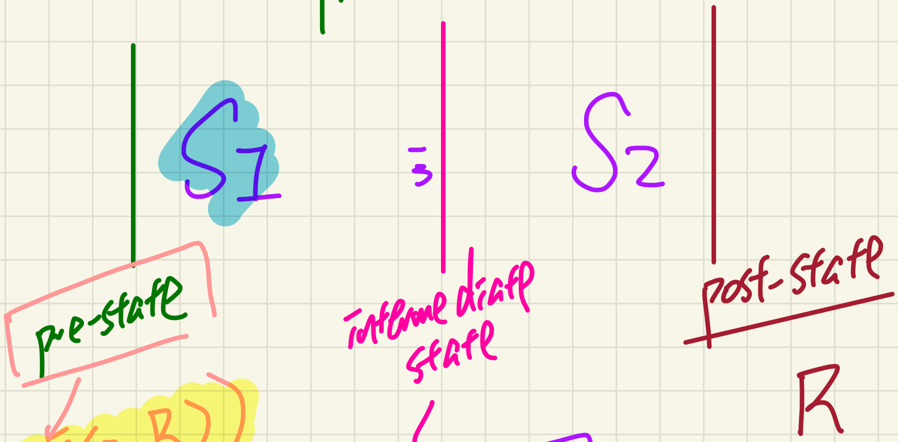
wp rule: Sequential Composition
Loop Invariant vs. Loop Variant
Correctness Conditions of Loops

Announcements/Reminders

- Exam guide released
- Some example questions to be released by April 7
- WrittenTest2 result released
- Lab4 released
- Bonus opportunity: Final Course Evaluation
- Office Hour this week: 3pm on Wed, Thu
- TA contact information (on-demand for labs) on eClass

wp Calculation for Sequential Composition

$wp(\underbrace{S_1}_{\text{phase 1}} ; \underbrace{S_2}_{\text{remaining phases}}, \underline{R})$ postcondition.



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

$\boxed{wp(S_2, R)}$ think of this as the postcond. for S_1

Correctness of Programs: Sequential Composition

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

✓ $\quad Q \quad S$ ✓ $\quad R$

$$\{Q\}\ S\ \{R\} \iff Q \overset{??}{\Rightarrow} \boxed{wp(S, R)}$$ 1.

(Step 1) Calculate $wp(\boxed{tmp := x} ; \boxed{x := y ; y := tmp}, x > y)$ ✓

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

$wp(tmp := x, wp(\boxed{x := y} ; \boxed{y := tmp, x > y}))$

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

$wp(tmp := x, wp(x := y, \boxed{wp(y := tmp, x > y)}))$ ✓

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

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

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

$\boxed{wp(tmp := x, y > tmp)} = \{wp \text{ rule of } :=\} = y > x$

(Step 2)

$True \Rightarrow y > x$

$\equiv \{\text{rid of } \Rightarrow\}$

$\boxed{y > x}$ ✗

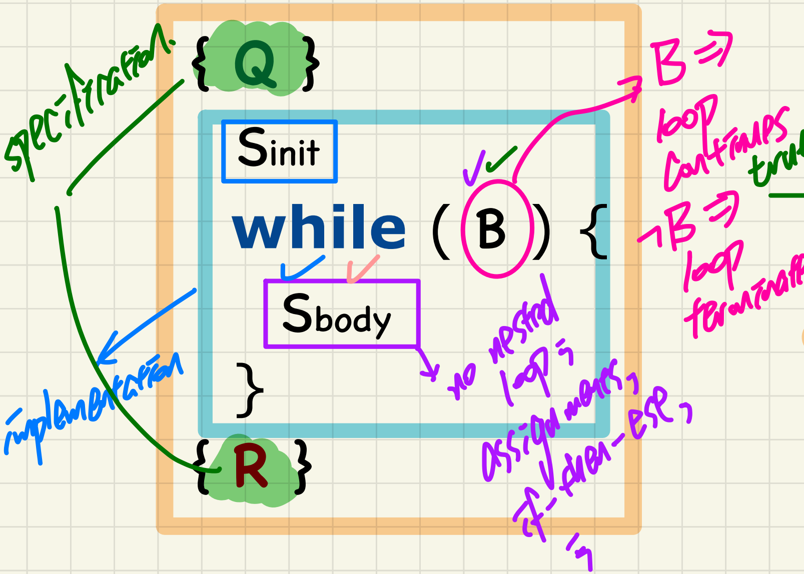
↳ counterex

$y \le x$

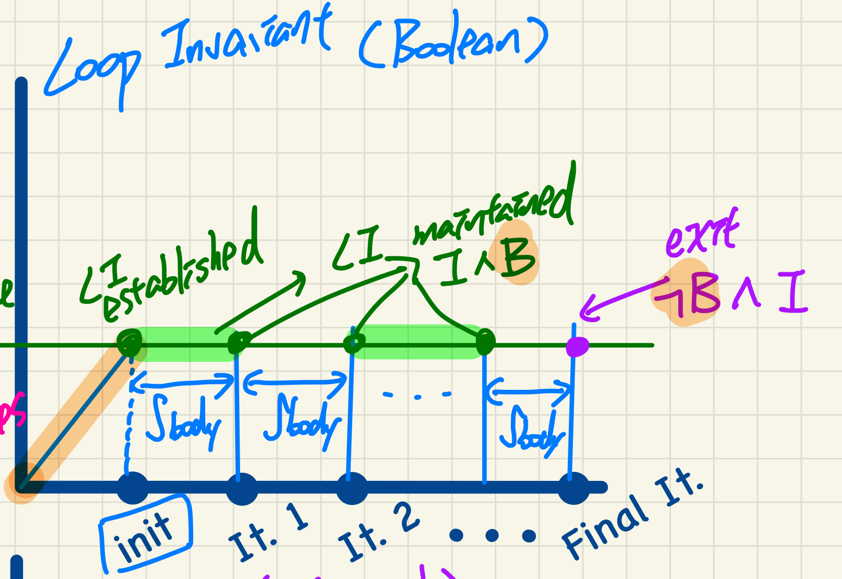
$x = 3$

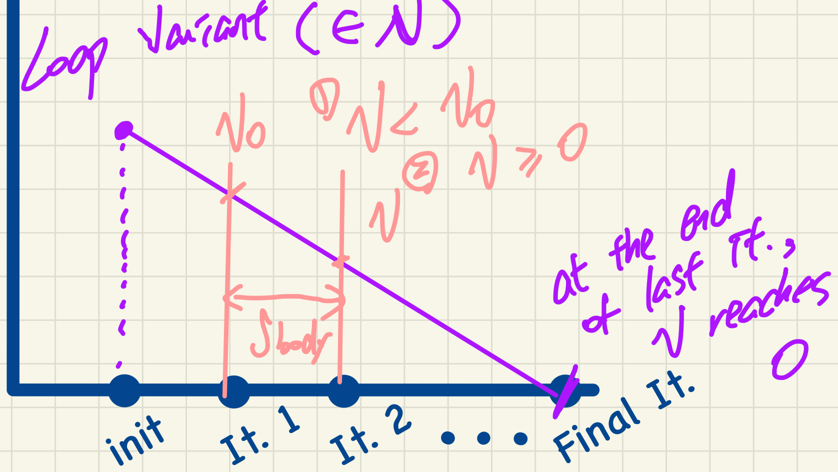
$y = 2$

Correctness of Loops



```
{ Q }
Sinit
while ( B ) {
    Sbody
}
{ R }
```





Contracts of Loops

Syntax

```
CONSTANT ... (* input list *)
I(var_list) == ...
V(var_list) == ...
--algorithm MyAlgorithm {
  variables ..., variant_pre = 0, variant_post = 0
  {
    assert Q; (* Precondition *)
    S_init
    assert I(...); (* Is LI established? *)
    while( B ) {
      variant_pre := V(...);
      S_body
      variant_post := V(...);

      assert variant_post >= 0;
      assert variant_post < variant_pre;
      assert I(...); (* Is LI preserved? *)
    }
    assert R; (* Postcondition *)
  }
}
```

LI

LV

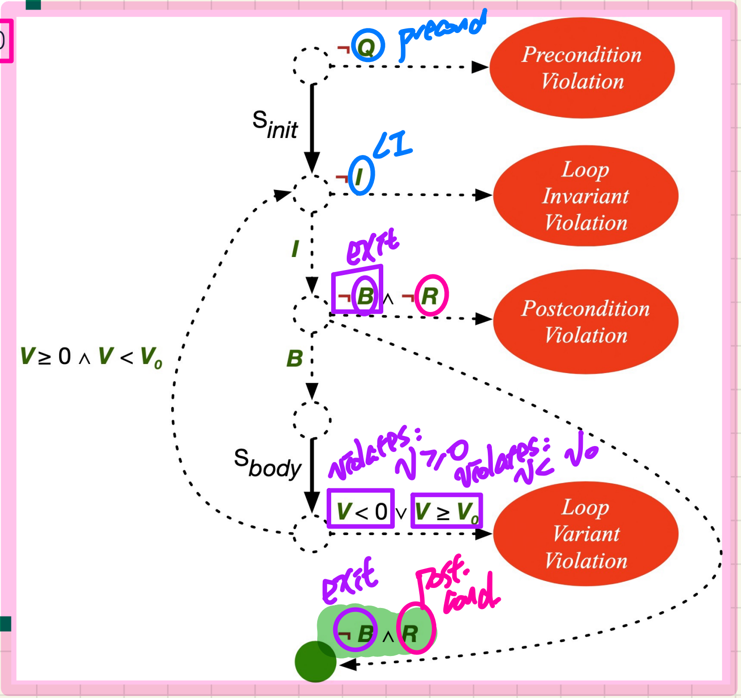
meta-variables to capture V in pre- and post-state

before entering loop

1. $V_{post} < V_{pre}$

2. $V_{post} \in \mathbb{N}$

Runtime Checks



Contracts of Loops: Example

Assume: Q and R are true

Specification

LI

LV

```
I(i) == (1 <= i) /\ (i <= 6)
V(i) == 6 - i
--algorithm loop_invariant_test
  variables i = 1, variant_pre = 0, variant_post = 0;
  {
   assert I(i);
    while (i <= 5) {
     variant_pre := V(i);
     i := i + 1;
     variant_post := V(i);
     assert variant_post >= 0;
     assert variant_post < variant_pre;
     assert I(i);
    } ;
  }
```

✓

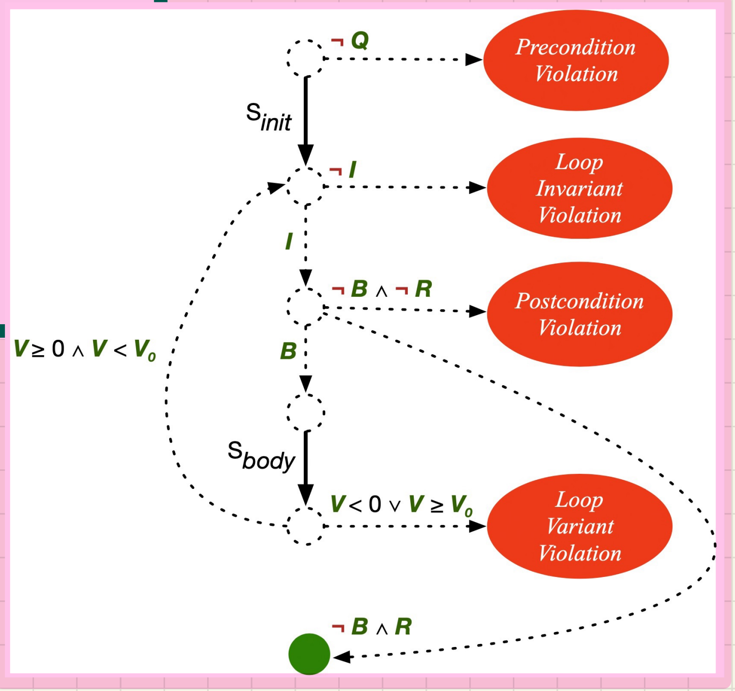
Sinit

not part of Sinit

Sbody

end of iteration	i	I	Vpre	Vpost	B
	1	T	—	—	T
1	2	T	6-1=5	6-2=4	T
2	3		4	3	
3	4		3	2	
4	5		2	1	
5	6		1	0	F

Runtime Checks



Contracts of Loops: Violations

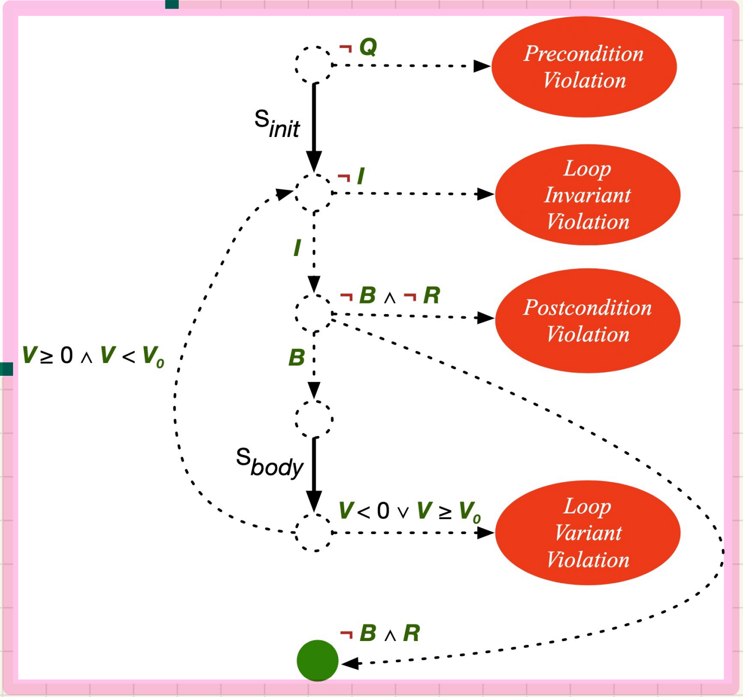
Exercise

Assume: Q and R are true

Specification

```
I(i) == (1 <= i) /\ (i <= 6)
V(i) == 6 - i
--algorithm loop_invariant_test
 variables i = 1, variant_pre = 0, variant_post = 0;
 {
   assert I(i);
   while (i <= 5) {
     variant_pre := V(i);
     i := i + 1;
     variant_post := V(i);
     assert variant_post >= 0;
     assert variant_post < variant_pre;
     assert I(i);
   } ;
 }
```

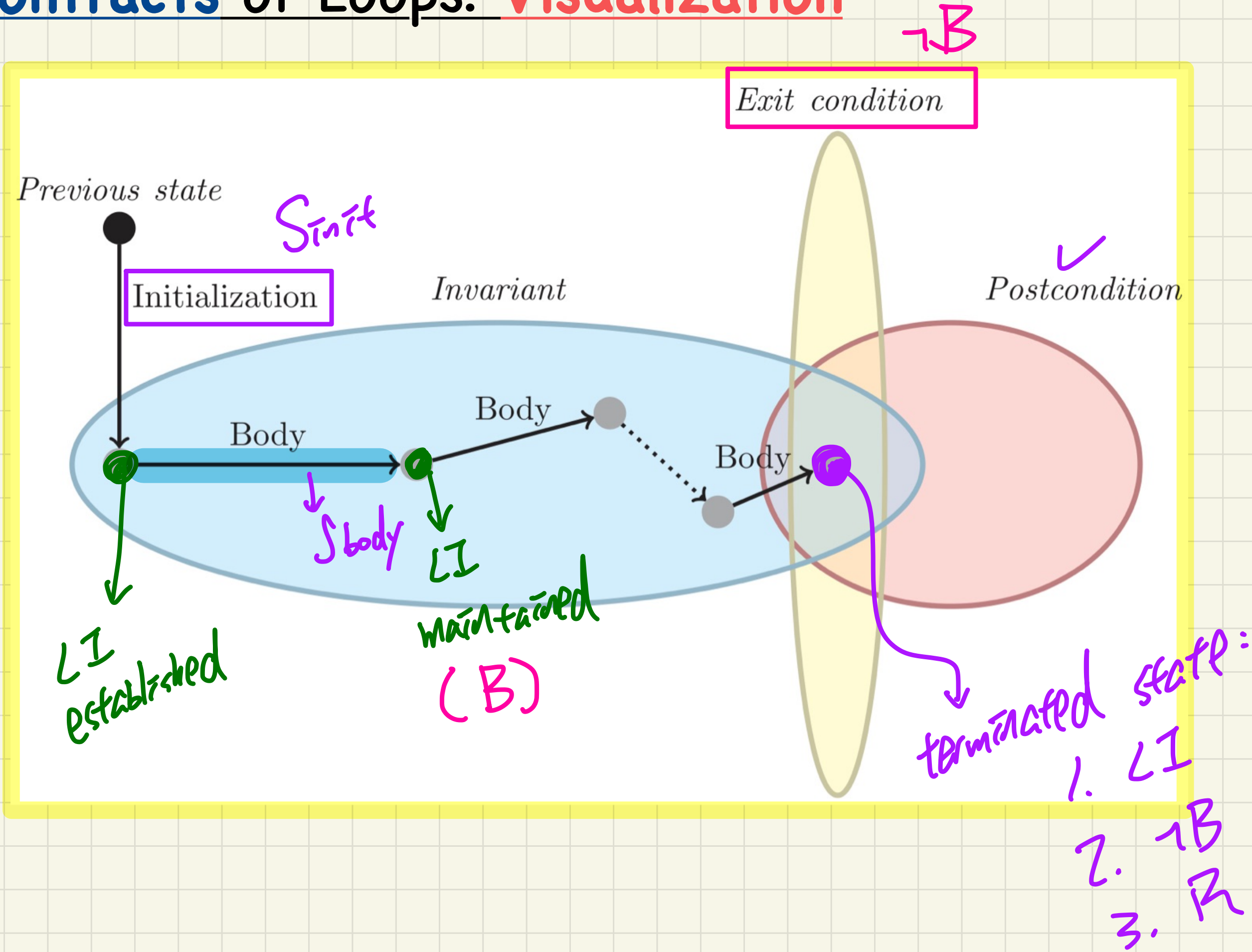
Runtime Checks



invariant: 1 <= i <= 5

variant: 5 - i

Contracts of Loops: Visualization



Correct Loops: Proof Obligations

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

    $\{Q\}\ S_{init}\ \{LI\}$

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

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

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

    $\neg B \wedge LI \Rightarrow R$

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

    $\{LI \wedge B\}\ S_{body}\ \{V \geq 0\}$

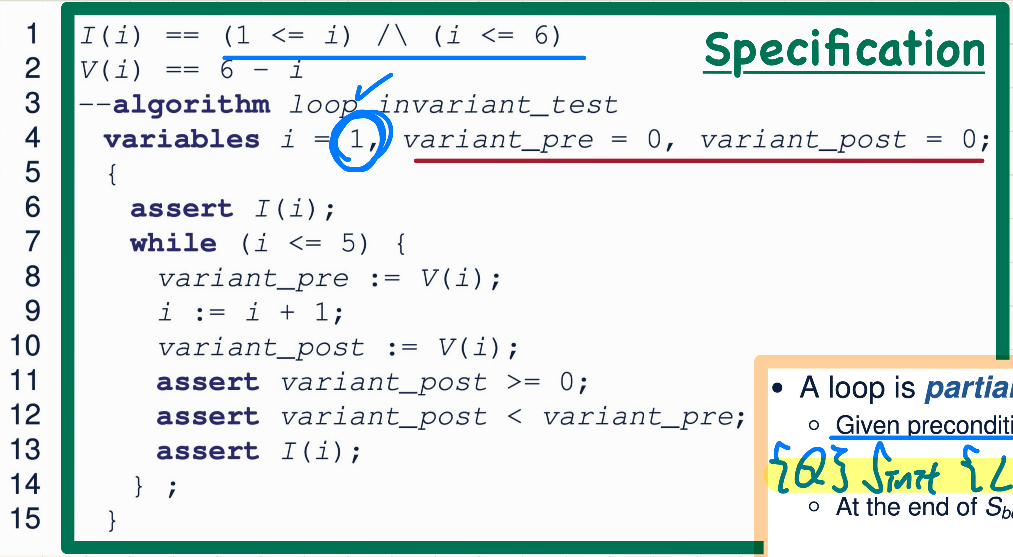
  - Given ***LI*** *I*, and not yet to exit, S_{body} decrements ***LV*** *V*.

    $\{LI \wedge B\}\ S_{body}\ \{V < V_0\}$

```
{Q}
$S_{init}$
assert I(...);
while( B ) {
  variant_pre := V(...);
  $S_{body}$
  variant_post := V(...);
  assert variant_post >= 0;
  assert variant_post < variant_pre;
  assert I(...);
}
{R}
```

Correct Loops: Proof Obligations

Example



Specification

```
I(i) == (1 <= i) /\ (i <= 6)
V(i) == 6 - i
--algorithm loop_invariant_test
 variables i = 1, variant_pre = 0, variant_post = 0;
 {
  assert I(i);
  while (i <= 5) {
   variant_pre := V(i);
   i := i + 1;
   variant_post := V(i);
   assert variant_post >= 0;
   assert variant_post < variant_pre;
   assert I(i);
  } ;
 }
```

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

    $\{Q\}\ S_{init}\ \{LI\}$ $\{True\}\ i := 1\ \{1 \le i \land i \le 6\}$
  - 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*.

- No multiple choice questions
- definitions
- short answers
  - ↳ justification
  - ↳ assertions (Lab2) e.g.,; algorithm (not as long as in ProgTest).
  - ↳ proofs. (math review, LTL).

- qs booklet
- answer booklet