wp Calculation for Sequential Composition

Correctness of Programs: Sequential Composition

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

Rules of Weakest Precondition: Summary

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

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

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

Proof Rules using Weakest Precondition

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

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

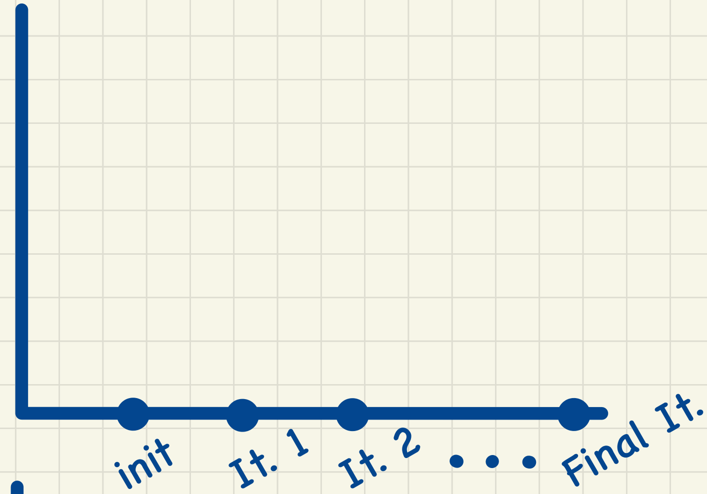
$$\{Q\}\ \mathtt{if}\ B\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2\ \mathtt{end}\ \{R\}$$

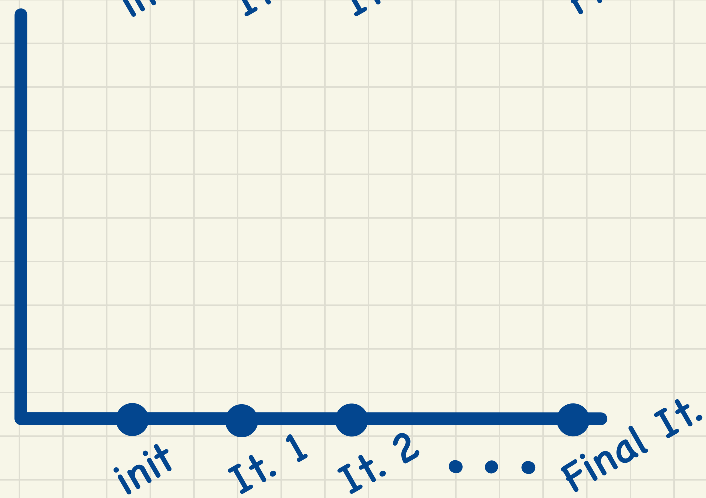
$$\iff \begin{pmatrix} \{\ Q \wedge B\ \}\ S_1\ \{\ R\ \} \\ \wedge \\ \{\ Q \wedge \neg B\ \}\ S_2\ \{\ R\ \} \end{pmatrix} \iff \begin{pmatrix} (Q \wedge B) \Rightarrow wp(S_1, R) \\ \wedge \\ (Q \wedge \neg B) \Rightarrow wp(S_2, R) \end{pmatrix}$$

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

Correctness of Loops

```
{ Q }
  S_init
  while ( B ) {
    S_body
  }
{ R }
```





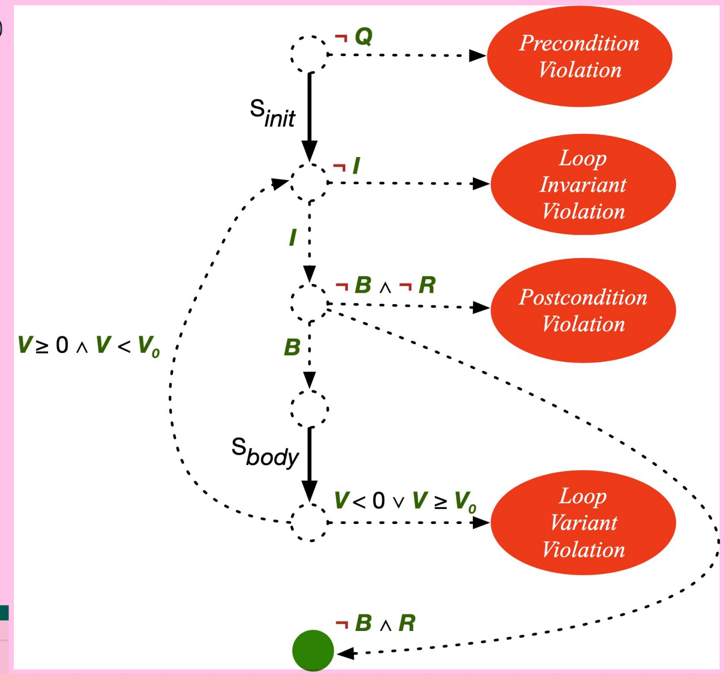
Contracts of Loops

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 *)
 }
}
```

Runtime Checks



Contracts of Loops: Example

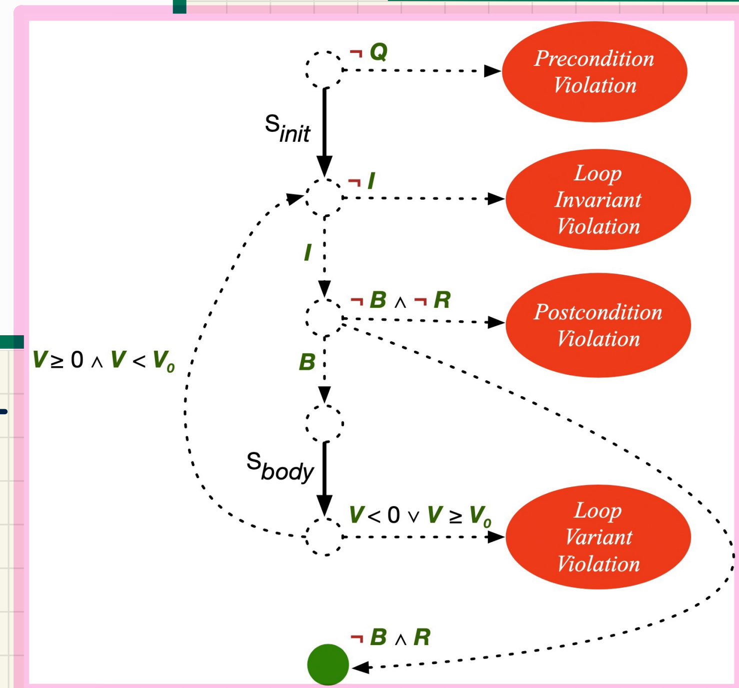
Assume: Q and R are true

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);
  } ;
 }
```

end of iteration	i	I	V	B

Runtime Checks



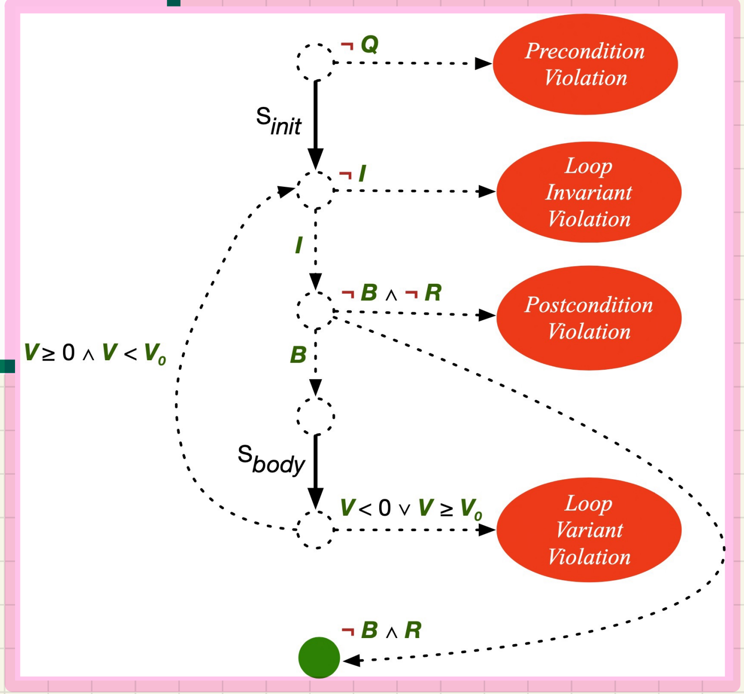
Contracts of Loops: Violations

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);
   } ;
 }
```

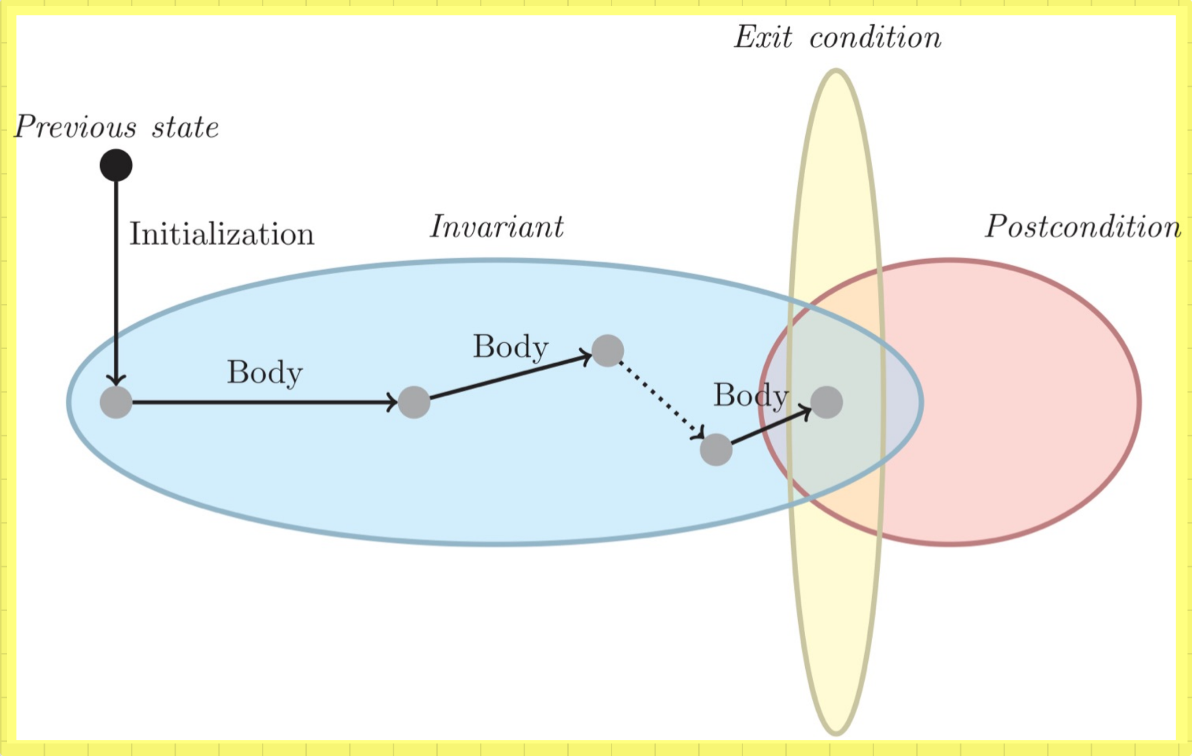
Runtime Checks



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.
  - 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.

```
{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*.
  - 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*.