

HOARE LOGIC

HOARE LOGIC

INTRODUCTION

- Since finding the exact wp or sp for while-loops is difficult, we will use an over-approximation in the form of an **inductive invariant** which preserves soundness.
- Much of the rest of the course (and majority of research in verification) deals with how to handle the verification problem for loops/loop-like constructs.
- Hoare Logic is a program logic/verification strategy which can be directly used to prove the validity of Hoare Triples.
 - Also provides a framework for specifying and verifying Inductive Loop Invariants.

DEFINITION

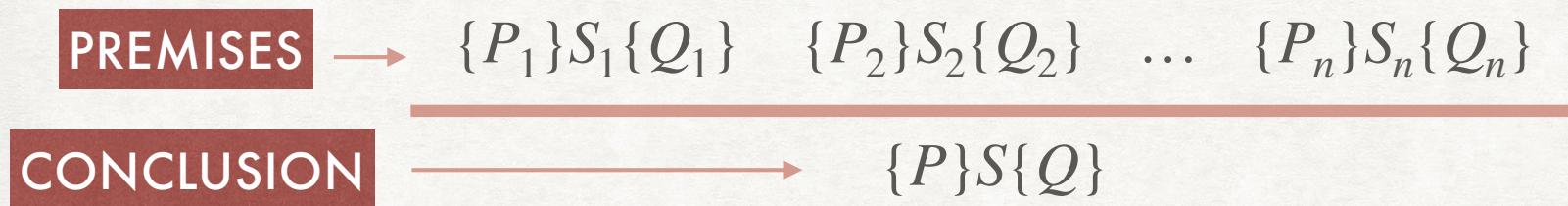
- Given sets of states P and Q , a program c satisfies the specification $\{P\}c\{Q\}$ if:
 - $\forall \sigma, \sigma'. \sigma \in P \wedge (\sigma, c) \hookrightarrow^* (\sigma', \text{skip}) \Rightarrow \sigma' \in Q$
- Using FOL formulae P and Q to express sets of states, we can now use the symbolic semantics $\rho(c)$:
 - $\forall V, V'. P \wedge \rho(c) \rightarrow Q[V'/V]$
- Hoare Logic is a program logic/proof system to directly prove the validity of Hoare Triples.
- We will study it in two forms:
 - A set of inference rules
 - A procedure to generate verification conditions (VCs) in FOL

RELATION WITH WP AND SP

- How are Hoare Triples, Weakest Pre-condition and Strongest Post-condition related with each other?
 - $\{wp(P, c)\} \subset \{P\}$
 - $\{P\} \subset \{sp(P, c)\}$
- **Homework:** Prove this formally using the definitions!

INFERENCE RULES

FORMAT



Key Idea: Use the validity of Hoare triples for smaller statements
to establish validity for compound statements

INFERENCE RULES

PRIMITIVE STATEMENTS

[R-ASSIGN]

$$\{P[e/x]\} \ x := e \ {P}$$

[R-HAVOC]

$$\{ \forall x . P \} \ x := \text{havoc} \ {P}$$

[R-ASSUME]

$$\{Q \rightarrow P\} \ \text{assume}(Q) \ {P}$$

[R-ASSERT]

$$\{Q \wedge P\} \ \text{assert}(Q) \ {P}$$

EXAMPLES

- Which of the following are true?
 - $\{y = 10\} \ x := 10 \ \{y = x\}$
 - $\{x = n - 1\} \ x := x + 1 \ \{x = n\}$
 - $\{y = x\} \ y := 2 \ \{y = x\}$
 - $\{z = 10\} \ y := 2 \ \{z = 10\}$
 - $\{y = 10\} \ y := x \ \{y = x\}$
- The last Hoare triple is valid, but we cannot prove it using [R-ASSIGN].
 - According to [R-ASSIGN], we have $\{y = x[x/y]\} \ y := x \ \{y = x\}$. Hence, $\{x = x\} \ y := x \ \{y = x\}$, which simplifies to $\{\top\} \ y := x \ \{y = x\}$.
 - Notice that $y = 10 \Rightarrow \top$.

PRE-CONDITION STRENGTHENING

$$\{P'\} \subset \{Q\} \quad P \Rightarrow P'$$

$$\{P\} \subset \{Q\}$$

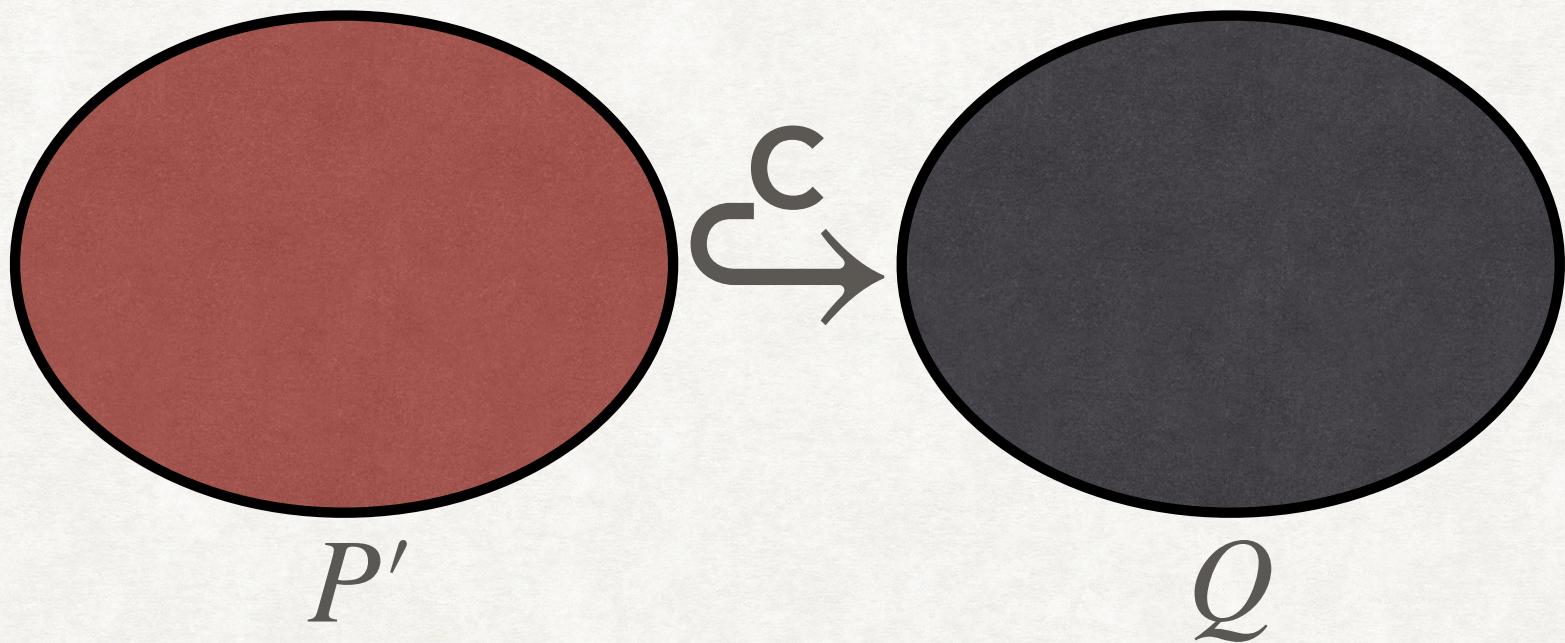
[R-STRENGTHEN-PRE]

PRE-CONDITION STRENGTHENING

$$\{P'\} \subset \{Q\} \quad P \Rightarrow P'$$

[R-STRENGTH-PRE]

$$\{P\} \subset \{Q\}$$

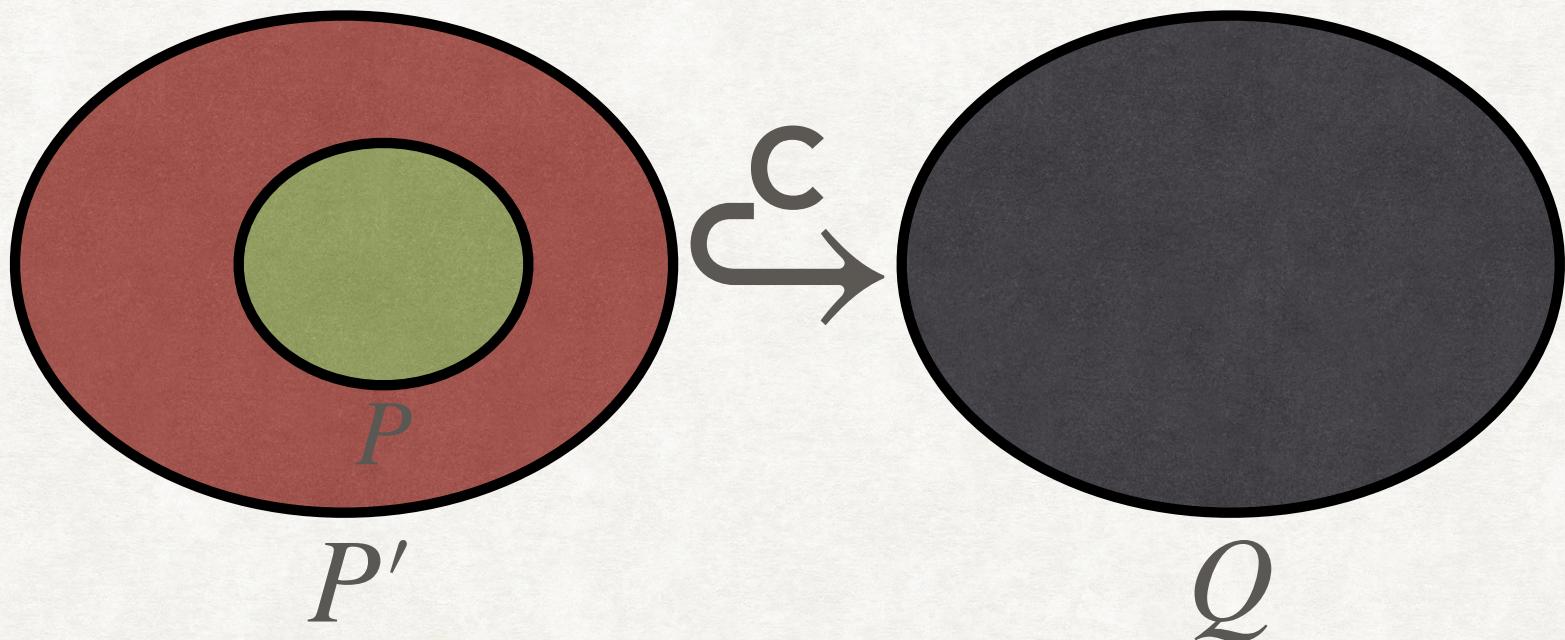


PRE-CONDITION STRENGTHENING

$$\{P'\} \subset \{Q\} \quad P \Rightarrow P'$$

[R-STRENGTH-PRE]

$$\{P\} \subset \{Q\}$$

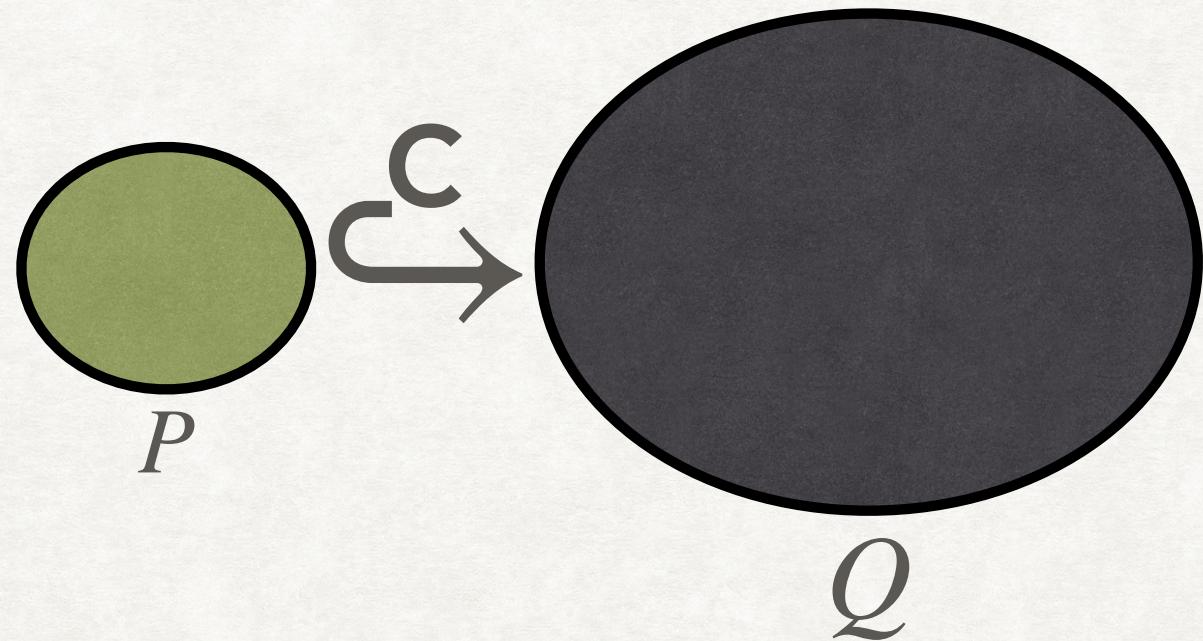


PRE-CONDITION STRENGTHENING

$$\{P'\} \subset \{Q\} \quad P \Rightarrow P'$$

[R-STRENGTH-PRE]

$$\{P\} \subset \{Q\}$$



PRE-CONDITION STRENGTHENING

$$\{P'\} \subset \{Q\} \quad P \Rightarrow P'$$

$$\{P\} \subset \{Q\}$$

[R-STRENGTHEN-PRE]

$$\{true\} \ y := x \ \{y = x\} \quad y = 10 \Rightarrow true$$

$$\{y = 10\} \ y := x \ \{y = x\}$$

POST-CONDITION WEAKENING

$$\{P\} \subset \{Q'\} \quad Q' \Rightarrow Q$$

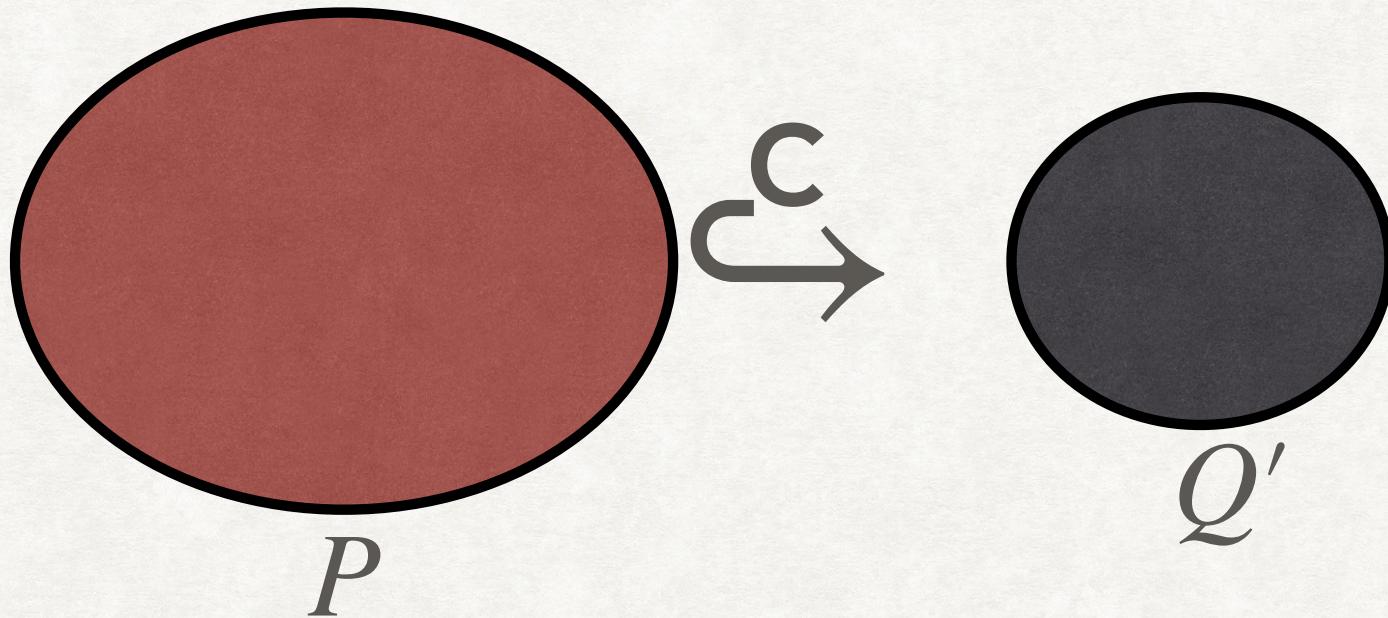
$$\{P\} \subset \{Q\}$$

[R-WEAKEN-POST]

POST-CONDITION WEAKENING

$$\frac{\{P\} \subset \{Q'\} \quad Q' \Rightarrow Q}{\{P\} \subset \{Q\}}$$

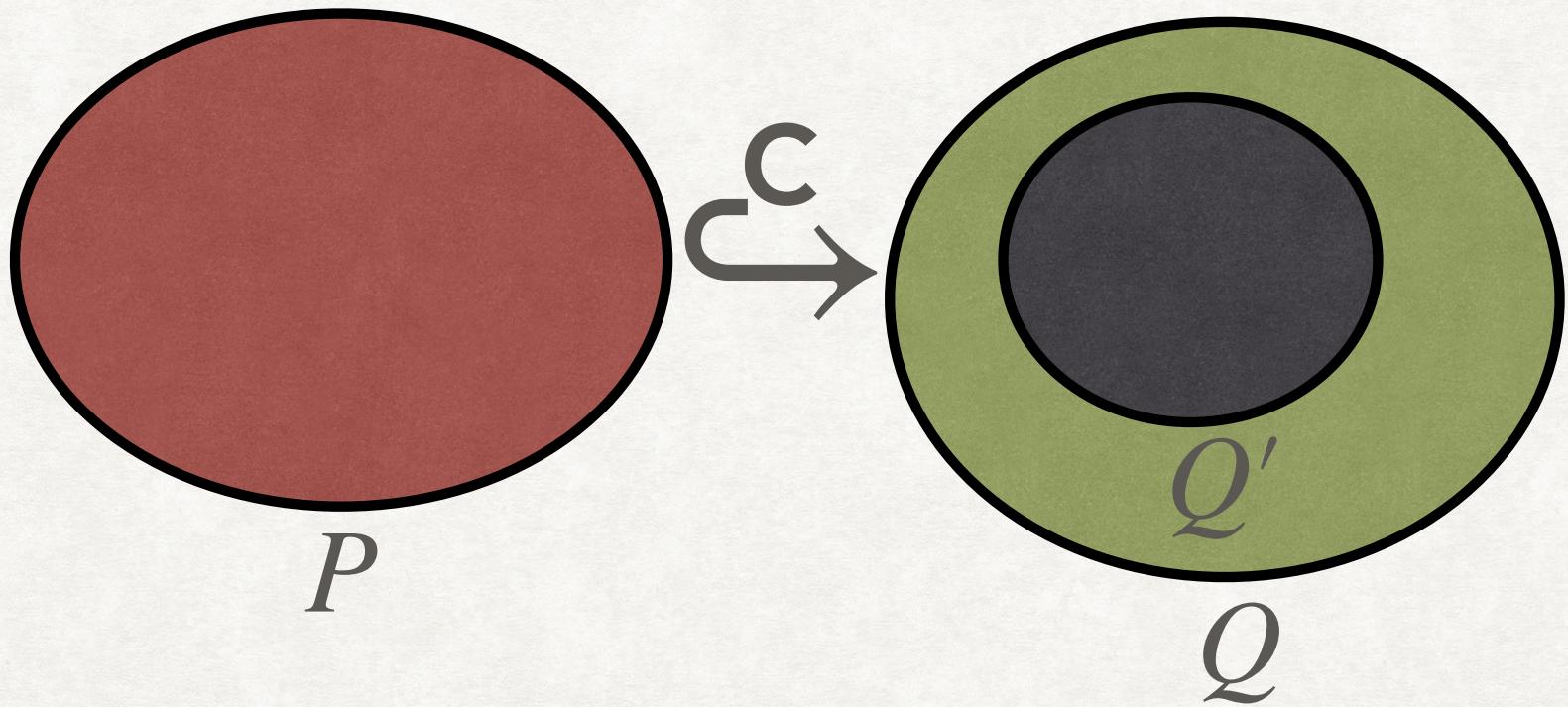
[R-WEAKEN-POST]



POST-CONDITION WEAKENING

$$\frac{\{P\} \subset \{Q'\} \quad Q' \Rightarrow Q}{\{P\} \subset \{Q\}}$$

[R-WEAKEN-POST]



INFERENCE RULES

COMPOUND STATEMENTS

$$\{P\} \ c_1 \ \{R\} \quad \{R\} \ c_2 \ \{Q\}$$

$$\{P\} \ c_1; c_2 \ \{Q\}$$

[R-SEQ]

INFERENCE RULES

COMPOUND STATEMENTS

$$\{P\} \ c_1 \ \{R\} \quad \{R\} \ c_2 \ \{Q\}$$

[R-SEQ]

$$\{P\} \ c_1; c_2 \ \{Q\}$$
$$\{P \wedge F\} \ c_1 \ \{Q\} \quad \{P \wedge \neg F\} \ c_2 \ \{Q\}$$

[R-IF-THEN-ELSE]

$$\{P\} \text{ if } (F) \text{ then } c_1 \text{ else } c_2 \ \{Q\}$$

INFERENCE RULES

COMPOUND STATEMENTS

$$\frac{\{P\} \ c_1 \ \{R\} \quad \{R\} \ c_2 \ \{Q\}}{\{P\} \ c_1; c_2 \ \{Q\}}$$

[R-SEQ]

$$\frac{\{P \wedge F\} \ c_1 \ \{Q\} \quad \{P \wedge \neg F\} \ c_2 \ \{Q\}}{\{P\} \text{ if } (F) \text{ then } c_1 \text{ else } c_2 \ \{Q\}}$$

[R-IF-THEN-ELSE]

Prove This!

SEQUENCING

EXAMPLE

$$\{P\} \ c_1 \ \{R\} \quad \{R\} \ c_2 \ \{Q\}$$

[R-SEQ]

$$\{P\} \ c_1; c_2 \ \{Q\}$$

$$\{true\} \ x := 2; y := x \ \{y = 2 \wedge x = 2\}$$

SEQUENCING

EXAMPLE

$$\{P\} \ c_1 \ \{R\}$$
$$\{R\} \ c_2 \ \{Q\}$$

[R-SEQ]
$$\{P\} \ c_1; c_2 \ \{Q\}$$

$$\{true\} \ x := 2 \ \{x = 2\}$$

$$\{x = 2\} \ y := x \ \{y = 2 \wedge x = 2\}$$

$$\{true\} \ x := 2; y := x \ \{y = 2 \wedge x = 2\}$$

IF-THEN-ELSE

EXAMPLE

$$\{P \wedge F\} \ c_1 \ \{Q\}$$
$$\{P \wedge \neg F\} \ c_2 \ \{Q\}$$

[R-IF-THEN-ELSE]

$$\{P\} \text{ if } (F) \text{ then } c_1 \text{ else } c_2 \ \{Q\}$$

$$\{true\} \text{ if } (x > 0) \text{ then } y := x \text{ else } y := -x \{y \geq 0\}$$

IF-THEN-ELSE

EXAMPLE

$$\{P \wedge F\} \ c_1 \ \{Q\}$$
$$\{P \wedge \neg F\} \ c_2 \ \{Q\}$$

[R-IF-THEN-ELSE]

$$\{P\} \text{ if } (F) \text{ then } c_1 \text{ else } c_2 \ \{Q\}$$

$$\{x \geq 0\} \ y := x \ \{y \geq 0\} \quad x > 0 \Rightarrow x \geq 0$$

$$\{x > 0\} \ y := x \ \{y \geq 0\}$$

$$\{x \leq 0\} \ y := -x \ \{y \geq 0\}$$

$$\{true\} \text{ if } (x > 0) \text{ then } y := x \text{ else } y := -x \{y \geq 0\}$$

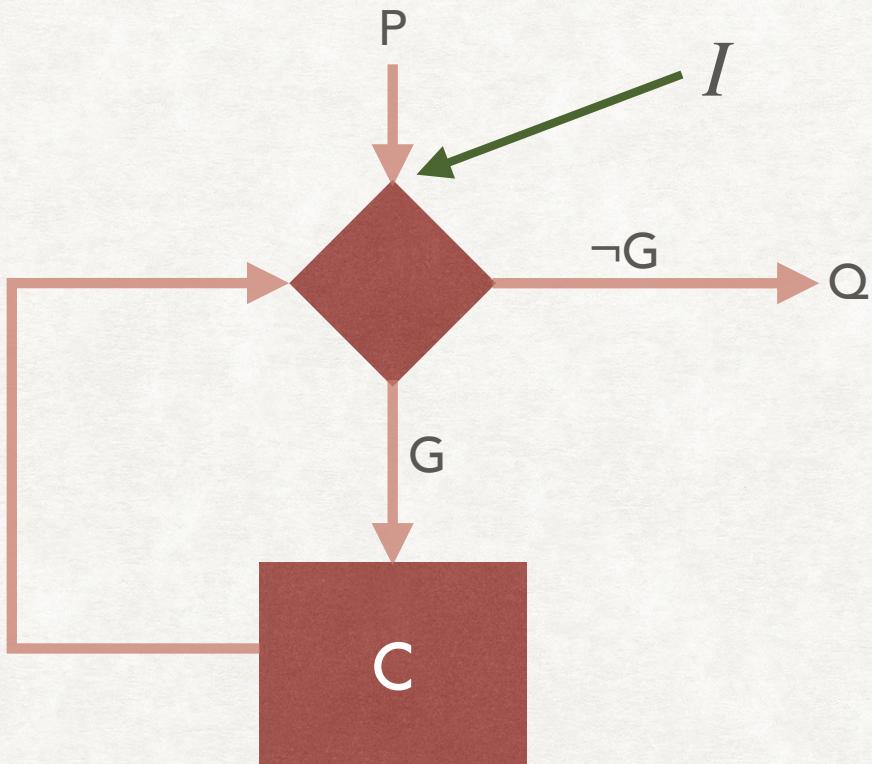
WHILE LOOPS

LOOP INVARIANTS

- Our goal is to prove the validity of $\{P\} \text{ while}(G) \text{ do } c \{Q\}$
 - Both sp and wp lead to non-terminating procedures.
- We will instead assume a loop invariant I which is an over-approximation of the states possible during execution of the loop, but is sufficient enough to prove the Hoare Triple.
 - I is assumed to be provided by the programmer.
 - Hoare Logic provides inference rules to prove that I is indeed a loop invariant.

WHILE LOOPS

LOOP INVARIANTS



- I needs to satisfy three properties:
 - I must hold initially at the start of the loop.
 - I must hold at the end of every iteration of the loop.
 - After exiting from the loop, I must imply the post-condition.

WHILE LOOPS

LOOP INVARIANTS

- Consider the code
 - `i:=0; j:=0; while(i<n) do i:=i+1; j:=i+j;`
- Which of the following are loop invariants?
 - $i \leq n$
 - $i < n$
 - $i \geq 0$

WHILE LOOP

INFERENCE RULE

$$\{I \wedge F\} \subset \{I\}$$

$$\{I\} \text{ while}(F) \text{ do } c; \{I \wedge \neg F\}$$

[R-WHILE-1]

WHILE LOOP

INFERENCE RULE

$$\{I \wedge F\} \; c \; \{I\}$$

[R-WHILE-1]

$$\{I\} \text{ while}(F) \text{ do } c; \{I \wedge \neg F\}$$

$$P \Rightarrow I \quad \{I \wedge F\} \; c \; \{I\} \quad I \wedge \neg F \Rightarrow Q$$

[R-WHILE-2]

$$\{P\} \text{ while}(F) \text{ do } c; \{Q\}$$

WHILE LOOP

INFERENCE RULE - EXAMPLE

Prove $\{i = 0 \wedge n > 0\}$ while($i < n$) do $i := i + 1$; $\{i = n\}$

Loop Invariants: $i \geq 0, i \leq n, n > 0\dots$

Which loop invariant is useful for proving the Hoare Triple?

$$I \triangleq i \leq n$$

WHILE LOOP

INFERENCE RULE - EXAMPLE

Prove $\{i = 0 \wedge n > 0\}$ while($i < n$) do $i := i + 1$; $\{i = n\}$

Loop Invariants: $i \geq 0, i \leq n, n > 0\dots$

Which loop invariant is useful for proving the Hoare Triple?

$$I \triangleq i \leq n$$

$\{i = 0 \wedge n > 0\}$ while($i < n$) do $i := i + 1$; $\{i = n\}$

WHILE LOOP

INFERENCE RULE - EXAMPLE

Prove $\{i = 0 \wedge n > 0\} \text{ while}(i < n) \text{ do } i := i + 1; \{i = n\}$

Loop Invariants: $i \geq 0, i \leq n, n > 0\dots$

Which loop invariant is useful for proving the Hoare Triple?

$$I \triangleq i \leq n$$

$$\{i \leq n \wedge i < n\} i := i + 1; \{i \leq n\}$$

$$\{i = 0 \wedge n > 0\} \Rightarrow i \leq n \quad \{i \leq n \wedge i < n\} i := i + 1; \{i \leq n\} \quad i \leq n \wedge i \geq n \Rightarrow i = n$$

$$\{i = 0 \wedge n > 0\} \text{ while}(i < n) \text{ do } i := i + 1; \{i = n\}$$

LOOP INVARIANT VS INDUCTIVE LOOP INVARIANT

- Consider again the code
 - `i:=0; j:=0; while(i<n) do i:=i+1; j:=i+j;`
- Is $j \geq 0$ a loop invariant?
 - Yes, it does hold at the beginning and at the end of every iteration.
- Does $\{j \geq 0 \wedge i < n\} i:=i+1; j:=i+j; \{j \geq 0\}$ hold?
 - NO! $j \geq 0$ is not an inductive loop invariant.
 - The inference rule admits only inductive loop invariants.
- How to strengthen the invariant to make it inductive?
 - $j \geq 0 \wedge i \geq 0$ is an inductive loop invariant.

COMPLETE EXAMPLE

```
{n > 0}
i := 0;
j := 0;
while(i < n) do
    i := i + 1;
    j := i + j;
{2j = n(n+1)}
```

COMPLETE EXAMPLE

$\{n > 0\}$

$\{P_1\}$

$i := 0;$

$\{P_2\}$

$j := 0;$

$\{P_3\}$

`while($i < n$) do`

$\{P_4\}$

$i := i + 1;$

$\{P_5\}$

$j := i + j;$

$\{P_6\}$

$\{P_7\}$

$\{2j = n(n+1)\}$

Loop Invariant:
???

COMPLETE EXAMPLE

$\{n > 0\}$

$\{P_1\}$

$i := 0;$

$\{P_2\}$

$j := 0;$

$\{P_3\}$

`while($i < n$) do`

$\{P_4\}$

$i := i + 1;$

$\{P_5\}$

$j := i + j;$

$\{P_6\}$

$\{P_7\}$

$\{2j = n(n+1)\}$

Loop Invariant:

$2j = i(i+1) \wedge i \leq n$

COMPLETE EXAMPLE

$\{n > 0\}$

$\{P_1\}$

$i := 0;$

$\{P_2\}$

$j := 0;$

$\{2j = i(i+1) \wedge i \leq n\}$

$\text{while}(i < n) \text{ do}$

$\{2j = i(i+1) \wedge i \leq n \wedge i < n\}$

$i := i + 1;$

$\{P_5\}$

$j := i + j;$

$\{2j = i(i+1) \wedge i \leq n\}$

$\{2j = i(i+1) \wedge i \leq n \wedge \neg(i < n)\}$

$\{2j = n(n+1)\}$

$\{I \wedge F\} \subset \{I\}$

$\{I\} \text{ while}(F) \text{ do } c; \{I \wedge \neg F\}$

COMPLETE EXAMPLE

$\{n > 0\}$

$\{P_1\}$

$i := 0;$

$\{P_2\}$

$j := 0;$

$\{2j = i(i+1) \wedge i \leq n\}$

while($i < n$) do

$\{2j = i(i+1) \wedge i \leq n \wedge i < n\}$

$i := i + 1;$

$\{P_5\}$

$j := i + j;$

$\{2j = i(i+1) \wedge i \leq n\}$

$\{2j = i(i+1) \wedge i \leq n \wedge \neg(i < n)\}$

$\{2j = n(n+1)\}$

$\{P\} \subset \{Q'\} \quad Q' \Rightarrow Q$

$\{P\} \subset \{Q\}$

[R-WEAKEN-POST]

$2j = i(i+1) \wedge i \leq n \wedge \neg(i < n)$

$\Rightarrow 2j = n(n+1)$

COMPLETE EXAMPLE

$\{n > 0\}$

$\{P_1\}$

$i := 0;$

$\{P_2\}$

$j := 0;$

$\{2j = i(i+1) \wedge i \leq n\}$

while($i < n$) do

$\{2j = i(i+1) \wedge i \leq n \wedge i < n\}$

$i := i + 1;$

$\{P_5\}$

$j := i + j;$

$\{2j = i(i+1) \wedge i \leq n\}$

$\{2j = n(n+1)\}$

$\{P\} \subset \{Q'\} \quad Q' \Rightarrow Q$

$\{P\} \subset \{Q\}$

[R-WEAKEN-POST]

$2j = i(i+1) \wedge i \leq n \wedge \neg(i < n)$

$\Rightarrow 2j = n(n+1)$

COMPLETE EXAMPLE

$\{n > 0\}$

$\{P_1\}$

$i := 0;$

$\{P_2\}$

$j := 0;$

$\{2j = i(i+1) \wedge i \leq n\}$

`while($i < n$) do`

$\{2j = i(i+1) \wedge i \leq n \wedge i < n\}$

$i := i + 1;$

$\{2i + 2j = i(i+1) \wedge i \leq n\}$

$j := i + j;$

$\{2j = i(i+1) \wedge i \leq n\}$

$\{2j = n(n+1)\}$

$\{P[e/x]\} \quad x := e \quad \{P\}$

[R-ASSIGN]

$(2j = i(i+1) \wedge i \leq n)[i + j/j]$

$\equiv 2(i + j) = i(i+1) \wedge i \leq n$

$\equiv 2i + 2j = i(i+1) \wedge i \leq n$

COMPLETE EXAMPLE

$\{n > 0\}$

$\{P_1\}$

$i := 0;$

$\{P_2\}$

$j := 0;$

$\{2j = i(i+1) \wedge i \leq n\}$

`while($i < n$) do`

$\{2j = i(i+1) \wedge i \leq n \wedge i < n\}$

$\{2j = i(i+1) \wedge i + 1 \leq n\}$

$i := i + 1;$

$\{2i + 2j = i(i+1) \wedge i \leq n\}$

$j := i + j;$

$\{2j = i(i+1) \wedge i \leq n\}$

$\{2j = n(n+1)\}$

$\{P[e/x]\} \quad x := e \quad \{P\}$

[R-ASSIGN]

$$\begin{aligned} & (2i + 2j = i(i+1) \wedge i \leq n)[(i+1)/i] \\ & \equiv 2(i+1) + 2j = (i+1)(i+2) \wedge i+1 \leq n \\ & \equiv 2j = i(i+1) \wedge i+1 \leq n \end{aligned}$$

COMPLETE EXAMPLE

$\{n > 0\}$

$\{P_1\}$

$i := 0;$

$\{P_2\}$

$j := 0;$

$\{2j = i(i+1) \wedge i \leq n\}$

while($i < n$) do

$\{2j = i(i+1) \wedge i \leq n \wedge i < n\}$

$2j = i(i+1) \wedge i \leq n \wedge i < n$

$\{2j = i(i+1) \wedge i + 1 \leq n\}$

$\Rightarrow 2j = i(i+1) \wedge i + 1 \leq n$

$i := i + 1;$

$\{2i + 2j = i(i+1) \wedge i \leq n\}$

$j := i + j;$

$\{2j = i(i+1) \wedge i \leq n\}$

$\{2j = n(n+1)\}$

COMPLETE EXAMPLE

$\{n > 0\}$

$\{P_1\}$

$i := 0;$

$\{P_2\}$

$j := 0;$

$\{2j = i(i+1) \wedge i \leq n\}$

while($i < n$) do

$\{2j = i(i+1) \wedge i \leq n \wedge i < n\}$

$i := i + 1;$

$\{2i + 2j = i(i+1) \wedge i \leq n\}$

$j := i + j;$

$\{2j = i(i+1) \wedge i \leq n\}$

$\{2j = n(n+1)\}$

$2j = i(i+1) \wedge i \leq n \wedge i < n$

$\Rightarrow 2j = i(i+1) \wedge i + 1 \leq n$

COMPLETE EXAMPLE

$\{n > 0\}$

$\{P_1\}$

$i := 0;$

$\{i(i+1) = 0 \wedge i \leq n\}$

$j := 0;$

$\{2j = i(i+1) \wedge i \leq n\}$

`while($i < n$) do`

$\{2j = i(i+1) \wedge i \leq n \wedge i < n\}$

$i := i + 1;$

$\{2i + 2j = i(i+1) \wedge i \leq n\}$

$j := i + j;$

$\{2j = i(i+1) \wedge i \leq n\}$

$\{2j = n(n+1)\}$

$\{P[e/x]\} \quad x := e \quad \{P\}$

[R-ASSIGN]

COMPLETE EXAMPLE

{ $n > 0$ }

{ $n \geq 0$ }

i := 0;

{ $i(i+1) = 0 \wedge i \leq n$ }

j := 0;

{ $2j = i(i+1) \wedge i \leq n$ }

while(i < n) do

{ $2j = i(i+1) \wedge i \leq n \wedge i < n$ }

i := i + 1;

{ $2i + 2j = i(i+1) \wedge i \leq n$ }

j := i + j;

{ $2j = i(i+1) \wedge i \leq n$ }

{ $2j = n(n+1)$ }

{ $P[e/x]$ } $x := e \ {P}$

[R-ASSIGN]

COMPLETE EXAMPLE

$\{n > 0\}$

$i := 0;$

$\{i(i+1) = 0 \wedge i \leq n\}$

$j := 0;$

$\{2j = i(i+1) \wedge i \leq n\}$

while($i < n$) do

$\{2j = i(i+1) \wedge i \leq n \wedge i < n\}$

$i := i + 1;$

$\{2i + 2j = i(i+1) \wedge i \leq n\}$

$j := i + j;$

$\{2j = i(i+1) \wedge i \leq n\}$

$\{2j = n(n+1)\}$

$\{P'\} \subset \{Q\} \quad P \Rightarrow P'$

$\{P\} \subset \{Q\}$

[R-STRENGTHEN-PRE]

SOUNDNESS AND COMPLETENESS

- All the inference rules together provide a procedure for establishing a Hoare triple $\{P\}c\{Q\}$.
- **Soundness:** If we can establish $\{P\}c\{Q\}$ using the inference rules, then is $\{P\}c\{Q\}$ a valid Hoare Triple?
 - Yes.
- **Completeness:** If $\{P\}c\{Q\}$ is a valid Hoare Triple, then can we always use the inference rules to establish it?
 - **Relatively Complete.**
 - If the underlying FOL theory is complete, then Hoare Logic is complete.

HOARE LOGIC

VERIFICATION CONDITION GENERATION

- We have already seen that the weakest pre-condition operator can be used to prove Hoare Triples:
 - $\{P\}c\{Q\}$ iff $P \Rightarrow wp(Q, c)$
 - Finding exact wp for loops is hard. We will instead use the loop invariant as an approximate wp .
 - $awp(Q, \text{while}(F)@\text{I do } c) = \text{I}$
 - Does this always hold?
 - Also need to show that following side-conditions hold:
 - $\{\text{I} \wedge F\}c\{\text{I}\}$
 - $\text{I} \wedge \neg F \Rightarrow Q$

RELATION BETWEEN AWP AND WP

- Let us formally define awp :
 - $\forall \sigma \in awp(Q, c) . \forall \sigma' . (\sigma, c) \hookrightarrow^* (\sigma', \text{skip}) \rightarrow \sigma' \in Q$
 - Homework: Prove that this holds for $awp(Q, \text{while}(F)@I \text{ do } c) = I$, when the side-conditions hold.
- We defined $wp(Q, c) \triangleq \{\sigma \mid \forall \sigma' . (\sigma, c) \hookrightarrow^* (\sigma', \text{skip}) \rightarrow \sigma' \in Q\}$
 - $awp(Q, c) \subseteq wp(Q, c)$
- We can then use awp for verifying the validity of Hoare Triples:
 - If $P \Rightarrow awp(Q, c)$ then $\{P\}c\{Q\}$.

RELATION BETWEEN AWP AND WP

EXAMPLES

- $awp(i \geq 0, \text{while}(i < n)@(i \geq 0) \text{ do } i := i + 1;) = ???$

RELATION BETWEEN AWP AND WP

EXAMPLES

- $\text{awp}(i \geq 0, \text{while}(i < n) @ (i \geq 0) \text{ do } i := i + 1;) = i \geq 0$

RELATION BETWEEN AWP AND WP

EXAMPLES

- $awp(i \geq 0, \text{while}(i < n)@(i \geq 0) \text{ do } i := i + 1;) = i \geq 0$
- $wp(i \geq 0, \text{while}(i < n)@(i \geq 0) \text{ do } i := i + 1;) = ???$

RELATION BETWEEN AWP AND WP

EXAMPLES

- $\text{awp}(i \geq 0, \text{while}(i < n) @ (i \geq 0) \text{ do } i := i + 1;) = i \geq 0$
- $\text{wp}(i \geq 0, \text{while}(i < n) @ (i \geq 0) \text{ do } i := i + 1;) = n \geq 0 \vee i \geq 0$

VC GENERATION - I

- We define $VC(Q, c)$ to collect the side-conditions needed for verifying that Q holds after execution of c .
- For **while(F)@ I do c** , there are two side-conditions:
 - $\{I \wedge F\}c\{I\}$
 - $I \wedge \neg F \Rightarrow Q$
- $\{I \wedge F\}c\{I\}$ is valid if $I \wedge F \Rightarrow awp(I, c)$.
 - c may contain loops, so we also need to consider $VC(I, c)$.
- Hence,
$$VC(Q, \text{while}(F)@I \text{ do } c) \triangleq (I \wedge \neg F \Rightarrow Q) \wedge (I \wedge F \Rightarrow awp(I, c)) \wedge VC(I, c)$$

VC GENERATION - II

- $VC(Q, x:=e) \triangleq T$
 - Also defined as T for all primitive program commands (assert, assume, havoc).
- $VC(Q, c_1; c_2) \triangleq ???$

VC GENERATION - II

- $VC(Q, x:=e) \triangleq true$
 - Also defined as *true* for all simple program commands (*assert*, *assume*, *havoc*).
- $VC(Q, c_1; c_2) \triangleq VC(Q, c_2) \wedge VC(awp(Q, c_2), c_1)$

VC GENERATION - II

- $VC(Q, x := e) \triangleq true$
 - Also defined as *true* for all simple program commands (*assert*, *assume*, *havoc*).
- $VC(Q, c_1; c_2) \triangleq VC(Q, c_2) \wedge VC(awp(Q, c_2), c_1)$
- $VC(Q, \text{if}(F) \text{ then } c_1 \text{ else } c_2) \triangleq ???$

VC GENERATION - II

- $VC(Q, x := e) \triangleq true$
 - Also defined as *true* for all simple program commands (*assert*, *assume*, *havoc*).
- $VC(Q, c_1; c_2) \triangleq VC(Q, c_2) \wedge VC(awp(Q, c_2), c_1)$
- $VC(Q, \text{if}(F) \text{ then } c_1 \text{ else } c_2) \triangleq VC(Q, c_1) \wedge VC(Q, c_2)$

VC GENERATION - III

- $awp(Q, c) \triangleq wp(Q, c)$ except for while loops, for which $awp(Q, \text{while}(F)@l \text{ do } c) = l$.
- Putting it all together, $\{P\}c\{Q\}$ is valid if the following FOL formula is valid:
 - $(P \rightarrow awp(Q, c)) \wedge VC(Q, c)$

RELATION BETWEEN AWP AND HOARE TRIPLES

- What is the relation between $awp(Q, c)$ and validity of the Hoare Triple $\{P\}c\{Q\}$?
 - Is it possible that $P \rightarrow awp(Q, c)$ is valid and $\{P\}c\{Q\}$ is not valid?
 - Is it possible that $\{P\}c\{Q\}$ is valid and $\neg(P \rightarrow awp(Q, c))$ is satisfiable?
 - How about $\neg(P \rightarrow wp(Q, c))$?

RELATION BETWEEN AWP AND WP

EXAMPLES

- $\text{awp}(i \geq 0, \text{while}(i < n) @ (i \geq 0) \text{ do } i := i + 1;) = i \geq 0$
- $\text{wp}(i \geq 0, \text{while}(i < n) @ (i \geq 0) \text{ do } i := i + 1;) = n \geq 0 \vee i \geq 0$

VC GENERATION

SOUNDNESS AND COMPLETENESS

- Is the VC generation procedure sound?
 - Yes. Prove this!
- Is the VC generation procedure complete?
 - No. It is not even relatively complete.
 - The annotated loop invariant may not be strong enough.
- Can the VC generation procedure be fully automated?
 - Yes. Whole point of the exercise!

EXAMPLE

{ T }

i := 1;

sum := 0;

while(i <= n) do

 j := 1;

 while(j <= i) do

 sum := sum + j; j := j + 1;

 i := i + 1;

{sum ≥ 0}

EXAMPLE

{ T }

i := 1;

sum := 0;

while(i <= n)@(sum ≥ 0) do

j := 1;

while(j <= i)@(sum ≥ 0 ∧ j ≥ 0) do

 sum := sum + j; j := j + 1;

 i := i + 1;

{sum ≥ 0}

- $VC(sum \geq 0, \text{outer loop}) :$
 - $sum \geq 0 \wedge i > n \rightarrow sum \geq 0$
 - $sum \geq 0 \wedge i \leq n \rightarrow sum \geq 0 \wedge 1 \geq 0$
 - $VC(sum \geq 0, \text{inner loop})$

EXAMPLE

{ T }

i := 1;

sum := 0;

while(i <= n)@(sum ≥ 0) do

j := 1;

while(j <= i)@(sum ≥ 0 ∧ j ≥ 0) do

 sum := sum + j; j := j + 1;

 i := i + 1;

{sum ≥ 0}

- $VC(sum \geq 0, \text{inner loop})$:

- $sum \geq 0 \wedge j \geq 0 \wedge j > i \rightarrow sum \geq 0$

- $sum \geq 0 \wedge j \geq 0 \wedge j \leq i \rightarrow sum + j \geq 0 \wedge j + 1 \geq 0$

EXAMPLE

{ T }

i := 1;

sum := 0;

while(i <= n)@(sum ≥ 0) do

j := 1;

while(j <= i)@(sum ≥ 0 ∧ j ≥ 0) do

 sum := sum + j; j := j + 1;

 i := i + 1;

{sum ≥ 0}

- Final Formula:
 - $T \rightarrow 0 \geq 0 \wedge VC(sum \geq 0, \text{outer loop})$

ANNOUNCEMENTS

- Tutorial session on Dafny tomorrow, conducted by Sheera Shamsu.
 - Tool Assignment-2 will be based on Dafny.
- Theory Assignment-2 will be released next week.
- We will start project meetings soon.

ADDING FUNCTIONS TO IMP

$p = l^*$

$l = \text{function } f(x_1, \dots, x_n)\{c\}$

$c = x := \text{exp} \mid x := \text{havoc}$

$= \mid \text{assume}(F) \mid \text{assert}(F)$

$= \mid \text{skip} \mid c; c \mid \text{if}(F) \text{ then } c \text{ else } c \mid \text{while}(F) \text{ do } c$

$= \mid x := f(\text{exp}_1, \dots, \text{exp}_n) \mid \text{return exp}$

- We will assume that all variables are local, and local variables across different function instances are disjoint.
- We will also assume that formal parameters are not modified by a function.

MODULAR VERIFICATION

- Each function is annotated with a pre-condition and a post-condition.
- Pre-condition specifies what is expected of the function's arguments
 - Formula in FOL whose free variables are the formal parameters of the function.
- Post-condition describes the function's return value
 - Formula in FOL whose free variables are the formal parameters and a special variable called *ret*.
- Together, pre-condition and post-condition specify a **contract**.
 - If the function is called with values which obey the pre-condition, then the output of the function will obey the post-condition.

VERIFYING FUNCTION CONTRACT

```
function f(x1,...,xn)
    requires(Pre)
    ensures(Post)
    {Body;}
```

- The function contract can be verified by proving the validity of the Hoare Triple $\{Pre\} \text{ Body } \{Post\}$

VERIFYING FUNCTION CALLS

- The function body may have calls to other functions (or even itself)
 - $\{P\}x := f(e_1, \dots, e_n)\{Q\}$
- If we can guarantee that the function's pre-condition holds before the call, then we can assume that the function's post-condition will hold after the call.
- We model the function call as follows:

VERIFYING FUNCTION CALLS

- The function body may have calls to other functions (or even itself)
 - $\{P\}x := f(e_1, \dots, e_n)\{Q\}$
- If we can guarantee that the function's pre-condition holds before the call, then we can assume that the function's post-condition will hold after the call.
- We model the function call as follows:

```
assert(Pre[e1/x1, ..., en/xn]);  
assume(Post[tmp/ret, e1/x1, ..., en/xn]);  
x := tmp;
```

VERIFYING FUNCTION CALLS

- The function body may have calls to other functions (or even itself)
 - $\{P\}x := f(e_1, \dots, e_n)\{Q\}$
- If we can guarantee that the function's pre-condition holds before the call, then we can assume that the function's post-condition will hold after the call.
- We model the function call as follows:

```
assert(Pre[e1/x1, ..., en/xn]);  
assume(Post[tmp/ret, e1/x1, ..., en/xn]);  
x := tmp;
```

- Why do we have to use *tmp*?
- What is the generated VC?

VERIFYING FUNCTION CALLS

- The function body may have calls to other functions (or even itself)
 - $\{P\}x := f(e_1, \dots, e_n)\{Q\}$
- If we can guarantee that the function's pre-condition holds before the call, then we can assume that the function's post-condition will hold after the call.
- We model the function call as follows:

```
assert(Pre[e1/x1, ..., en/xn]);  
assume(Post[tmp/ret, e1/x1, ..., en/xn]);  
x := tmp;
```

- Why do we have to use *tmp*?
- What is the generated VC? $P \rightarrow (Pre \wedge (Post \rightarrow Q[tmp/y]))$

EXAMPLE

```
FindMax(a,l,u)
    requires(l >= 0 && l <= u && u < |a|)
ensures(∀i. l<=i<=u → ret >= a[i])
{
    if (l == u)
        return a[l];
    else
        m := FindMax(a, l+1, u);
        if (a[l] > m)
            return a[l];
        else
            return m;
}
```

EXAMPLE

```
FindMax(a,l,u)
    requires(l >= 0 && l <= u && u < |a|)
ensures(∀i. l<=i<=u → ret >= a[i])
{
    if (l == u)
        return a[l];
    else
        assert(Pre[l+1/l]);
        assume(Post[tmp/ret, l+1/l]);
        m := tmp;
        if (a[l] > m)
            return a[l];
        else
            return m;
}
```

EXAMPLE

```
{l ≥ 0 ∧ l ≤ u ∧ u < |a| }  
if (l == u)  
    ret:=a[l];  
else  
    assert(Pre[l+1/l]);  
    assume(Post[tmp/ret, l+1/l]);  
    m := tmp;  
    if (a[l] > m)  
        ret:=a[l];  
    else  
        ret:=m;  
{ ∀i . l ≤ i ≤ u → ret ≥ a[i] }
```

$$\begin{aligned} Pre \rightarrow (l = u \rightarrow Post[a[l]/ret]) \wedge \\ l \neq u \rightarrow Pre[(l + 1)/l] \\ \wedge Post[tmp/ret, (l + 1)/l] \rightarrow \\ (a[l] > tmp \rightarrow Post[a[l]/ret]) \wedge (a[l] \leq tmp \rightarrow Post[tmp/ret]) \end{aligned}$$

EXAMPLE - BINARY SEARCH

```
BinarySearch(a,l,u,e)
  requires(l >= 0 && u < |a|)
  ensures(ret  $\leftrightarrow$   $\exists i. l \leq i \leq u \ \& \ a[i] == e$ )
{
  if (l > u) then
    return false;
  else
  {
    m := (l+u)/2;
    if (a[m]==e) then
      return true;
    else
    {
      if (a[m] < e)
        return BinarySearch(a,m+1,u,e);
      else
        return BinarySearch(a,l,m-1,e);
    }
  }
}
```

EXAMPLE - BINARY SEARCH

```
BinarySearch(a,l,u,e)
  requires(l >= 0 && u < |a| && sorted(a,l,u) )
  ensures(ret  $\leftrightarrow$   $\exists i. l \leq i \leq u \wedge a[i] == e$ )
{
  if (l > u) then
    return false;
  else
  {
    m := (l+u)/2;
    if (a[m]==e) then
      return true;
    else
    {
      if (a[m] < e)
        return BinarySearch(a,m+1,u,e);
      else
        return BinarySearch(a,l,m-1,e);
    }
  }
}
```

sorted(a, l, u) $\Leftrightarrow \forall i, j. l \leq i \leq j \leq u \rightarrow a[i] \leq a[j]$

EXAMPLE - BINARY SEARCH

```
BinarySearch(a,l,u,e)
  requires(l >= 0 && u < |a| && sorted(a,l,u) )
  ensures(ret  $\leftrightarrow$   $\exists i. l \leq i \leq u \wedge a[i] == e$ )
{
  if (l > u) then
    return false;
  else
  {
    m := (l+u)/2;
    if (a[m]==e) then
      return true;
    else
    {
      if (a[m] < e)
        return BinarySearch(a,m+1,u,e);
      else
        return BinarySearch(a,l,m-1,e);
    }
  }
}
```

sorted(a, l, u) $\Leftrightarrow \forall i, j. l \leq i \leq j \leq u \rightarrow a[i] \leq a[j]$

IN THE BOOK...

- More Examples (Chapters 5,6)
 - Linear Search
 - Bubble Sort
 - Quick Sort
- A slightly different VC generation procedure
- Heuristics for crafting loop invariants