Hoare Logic: Loops & Framing

COS 441 Slides 12

Agenda

- Last few lectures
 - Hoare Logic:
 - {P}C{Q}
 - If P is true in the initial state s. And C in state s evaluates to s'. Then Q must be true in s'.
 - Rules of Hoare logic:
 - rule of consequence
 - assignment rule, skip rule, sequence rule, if rule
- This time:
 - While Loops

HOARE LOGIC: WHILE LOOPS

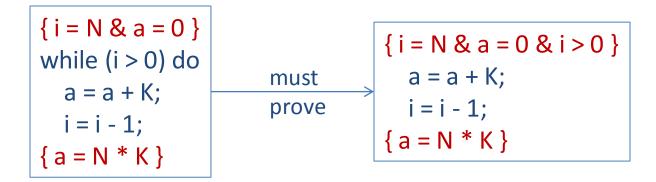
Rule for while statements

```
If ???
then { P } while (e > 0) do C { Q }
```

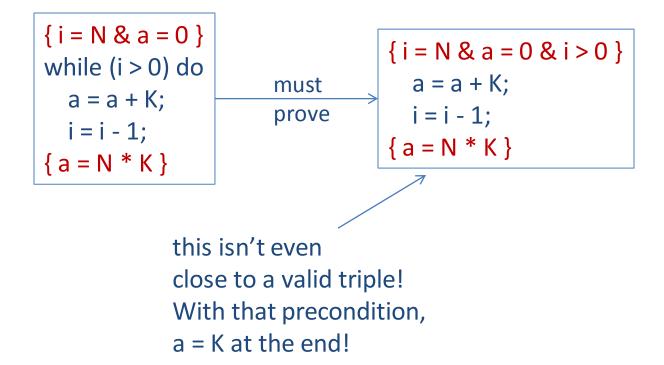
```
If { P & e > 0 } C { Q }
then { P } while (e > 0) do C { Q }
```

```
{ i = N & a = 0 }
while (i > 0) do
a = a + K;
i = i - 1;
{ a = N * K }
```

```
If { P & e > 0 } C { Q }
then { P } while (e > 0) do C { Q }
```



```
If { P & e > 0 } C { Q }
then { P } while (e > 0) do C { Q }
```



- Problem: We need to verify all iterations of a loop and we need to do it with a finite amount of work
- Solution: We will come up with an invariant that holds at the beginning and end of all iterations.
 - We prove that the loop body preserves the invariant every time around
- Unfortunate reality: Inferring invariants automatically is undecideable.
 - This puts significant limits on the degree to which we can automate verification.

• While rule:

```
If P \Rightarrow I and \{e > 0 \& I\} C \{I\} and I \& \sim (e > 0) \Rightarrow Q
then \{P\} while \{e > 0\} do \{Q\}
```

then $\{P\}$ while (e > 0) do $C\{Q\}$

While rule:

If $P \Rightarrow I$ and $\{e > 0 \& I\} C \{I\}$ and $I \& \sim (e > 0) \Rightarrow Q$

Inference rule notation:

$$P \Rightarrow I$$
 { $e > 0 & I$ } $C\{I\}$ | $l & \sim (e > 0) \Rightarrow Q$
{ $P \}$ while ($e > 0$) do $C\{Q\}$

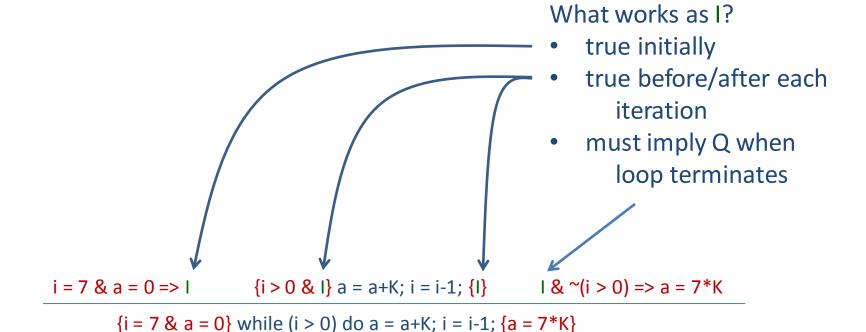
$$P \Rightarrow I$$
 { $e > 0 \& I$ } $C \{ I \}$ | $I \& \sim (e > 0) \Rightarrow Q$
 $\{ P \} \text{ while } (e > 0) \text{ do } C \{ Q \}$

```
\{i = 7 \& a = 0\} while (i > 0) do a = a+K; i = i-1; \{a = 7*K\}
```

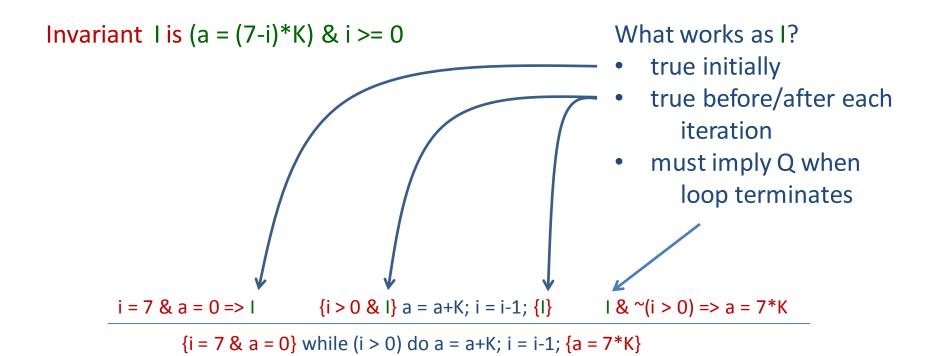
$$P \Rightarrow I$$
 { $e > 0 \& I$ } $C \{ I \}$ | $I \& \sim (e > 0) \Rightarrow Q$
 $\{ P \} \text{ while } (e > 0) \text{ do } C \{ Q \}$

$$i = 7 \& a = 0 \Rightarrow I$$
 { $i > 0 \& I$ } $a = a + K$; $i = i - 1$; { I } $I \& \sim (i > 0) \Rightarrow a = 7 * K$
{ $i = 7 \& a = 0$ } while ($i > 0$) do $a = a + K$; $i = i - 1$; { $a = 7 * K$ }

$$P \Rightarrow I$$
 { $e > 0 \& I$ } $C \{ I \}$ | $I \& \sim (e > 0) \Rightarrow Q$
 $\{ P \} \text{ while } (e > 0) \text{ do } C \{ Q \}$



$$P \Rightarrow I$$
 { $e > 0 \& I$ } $C \{ I \}$ | $I \& \sim (e > 0) \Rightarrow Q$
 $\{ P \} \text{ while } (e > 0) \text{ do } C \{ Q \}$



$$P \Rightarrow I$$
 { $e > 0 \& I$ } $C \{ I \}$ | $I \& \sim (e > 0) \Rightarrow Q$
 $\{ P \} \text{ while } (e > 0) \text{ do } C \{ Q \}$

Invariant I is
$$(a = (7-i)*K) \& i >= 0$$

Checking I:

• $i = 7 \& a = 0 \Rightarrow (a = (7-i)*K) \& i >= 0$

To for i

O for a

 $0 = (7-7)*K$

$$i = 7 \& a = 0 \Rightarrow I$$
 { $i > 0 \& I$ } $a = a + K$; $i = i - 1$; { I } $I \& \sim (i > 0) \Rightarrow a = 7 * K$
{ $i = 7 \& a = 0$ } while ($i > 0$) do $a = a + K$; $i = i - 1$; { $a = 7 * K$ }

$$P \Rightarrow I$$
 { $e > 0 & I$ } $C\{I\}$ | $I & \sim (e > 0) \Rightarrow Q$
 $\{P\} \text{ while } (e > 0) \text{ do } C\{Q\}$

Invariant I is (a = (7-i)*K) & i >= 0Checking I:

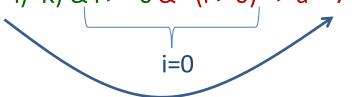
- $i = 7 \& a = 0 \Rightarrow (a = (7-i)*K) \& i >= 0$
- (a = (7-i)*K) & i >= 0 & ``(i > 0) => a = 7*K

$$i = 7 \& a = 0 \Rightarrow I$$
 { $i > 0 \& I$ } $a = a + K$; $i = i - 1$; { I } $I \& \sim (i > 0) \Rightarrow a = 7 * K$ } { $i = 7 \& a = 0$ } while ($i > 0$) do $a = a + K$; $i = i - 1$; { $a = 7 * K$ }

$$P \Rightarrow I$$
 { $e > 0 \& I$ } $C \{ I \}$ | $I \& \sim (e > 0) \Rightarrow Q$
 $\{ P \} \text{ while } (e > 0) \text{ do } C \{ Q \}$

Invariant I is (a = (7-i)*K) & i >= 0Checking I:

- $i = 7 \& a = 0 \Rightarrow (a = (7-i)*K) \& i >= 0$
- (a = (7-i)*K) & i >= 0 & ``(i > 0) => a = 7*K



$$i = 7 \& a = 0 \Rightarrow I$$
 {i > 0 & I} a = a+K; i = i-1; {I} I & ~(i > 0) => a = 7*K
{i = 7 & a = 0} while (i > 0) do a = a+K; i = i-1; {a = 7*K}

$$P \Rightarrow I$$
 { $e > 0 & I$ } $C\{I\}$ | $I & \sim (e > 0) \Rightarrow Q$
 $\{P\} \text{ while } (e > 0) \text{ do } C\{Q\}$

Invariant I is (a = (7-i)*K) & i >= 0Checking I:

- $i = 7 \& a = 0 \Rightarrow (a = (7-i)*K) \& i >= 0$
- (a = (7-i)*K) & i >= 0 & ``(i > 0) => a = 7*K
- validate the triple: {i > 0 & I} a = a+K; i = i-1; {I}

$$i = 7 \& a = 0 \Rightarrow I$$
 {i > 0 & I} a = a+K; i = i-1; {I} I & ~(i > 0) => a = 7*K
{i = 7 & a = 0} while (i > 0) do a = a+K; i = i-1; {a = 7*K}

$$P \Rightarrow I$$
 { $e > 0 \& I$ } $C \{ I \}$ | $I \& \sim (e > 0) \Rightarrow Q$
 $\{ P \} \text{ while } (e > 0) \text{ do } C \{ Q \}$

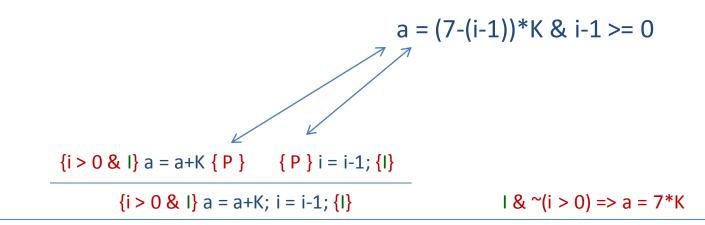
Invariant I is (a = (7-i)*K) & i >= 0

 $\{i = 7 \& a = 0\}$ while (i > 0) do a = a+K; i = i-1; $\{a = 7*K\}$

$$P \Rightarrow I$$
 { $e > 0 \& I$ } $C \{ I \}$ | $I \& \sim (e > 0) \Rightarrow Q$
 $\{ P \} \text{ while } (e > 0) \text{ do } C \{ Q \}$

Invariant I is (a = (7-i)*K) & i >= 0

 $i = 7 \& a = 0 \Rightarrow I$



 $\{i = 7 \& a = 0\}$ while (i > 0) do a = a+K; i = i-1; $\{a = 7*K\}$

$$P \Rightarrow I$$
 { $e > 0 \& I$ } $C \{ I \}$ | $I \& \sim (e > 0) \Rightarrow Q$
 $\{ P \} \text{ while } (e > 0) \text{ do } C \{ Q \}$

Invariant I is (a = (7-i)*K) & i >= 0

$$a + K = (7-(i-1))*K \& i-1 >= 0$$

$$a = (7-(i-1))*K \& i-1 >= 0$$

$$a = (7-(i-1))*K \& i-1 >= 0$$

$$\{0\} = a+K \{P\}$$

$$\{i = 7 \& a = 0\}$$
 while $(i > 0)$ do $a = a+K$; $i = i-1$; $\{a = 7*K\}$

$$P \Rightarrow I$$
 { $e > 0 \& I$ } $C \{I\}$ | $I \& \sim (e > 0) \Rightarrow Q$
 $\{P\} \text{ while } (e > 0) \text{ do } C \{Q\}$

Invariant I is (a = (7-i)*K) & i >= 0

$$i > 0 \& a = (7-i)*K \& i >= 0 \Rightarrow a + K = (7-(i-1))*K \& i-1 >= 0$$

$$i > 0 \& I \Rightarrow Q \qquad \{Q\} \ a = a + K \{P\} \}$$

$$\{i > 0 \& I\} \ a = a + K \{P\} \qquad \{P\} \ i = i-1; \{I\} \}$$

$$i = 7 \& a = 0 \Rightarrow I \qquad \{i > 0 \& I\} \ a = a + K; \ i = i-1; \{I\} \qquad I \& ``(i > 0) \Rightarrow a = 7*K$$

$$\{i = 7 \& a = 0\}$$
 while $(i > 0)$ do $a = a+K$; $i = i-1$; $\{a = 7*K\}$

$$P \Rightarrow I$$
 { $e > 0 \& I$ } $C \{ I \}$ | $I \& \sim (e > 0) \Rightarrow Q$
 $\{ P \} \text{ while } (e > 0) \text{ do } C \{ Q \}$

Invariant I is (a = (7-i)*K) & i >= 0

$$i > 0 \& a = (7-i)*K \& i > = 0 \Rightarrow a + K = (7-(i-1))*K \& i-1 > = 0$$

$$= a + K = (7-i)*K + K \& i-1 > = 0$$

$$= a = (7-i)*K \& i-1 > = 0$$

$$= a = (7-i)*K \& i-1 > = 0$$

$$(0) a = a+K \{P\}$$

$$(i > 0 \& I) a = a+K \{P\}$$

$$\{i > 0 \& I\} a = a+K \{P\}$$

$$\{i > 0 \& I\} a = a+K; i = i-1; \{I\}$$

$$1 \& ``(i > 0) = > a = 7*K$$

 $\{i = 7 \& a = 0\}$ while (i > 0) do a = a+K; i = i-1; $\{a = 7*K\}$

```
{x = 0} while (x < 15) { x = x + 2 } { even(x) }
```

```
{x = 0} while (x < 15) { x = x + 2 } { even(x) }
Invariant: even(x)
• x = 0 => even(x)
• even(x) & x >= 15 => even(x)
• { even(x) & x < 15 } x = x + 2 { even (x) }</pre>
```

```
{x = 0 \& y = N \& N > 0} while {y > 0} {y = y - 1; x = x + 1} {x = N}
```

```
{x = 0 \& y = N \& N > 0} while {y > 0} {y = y - 1; x = x + 1} {x = N}
Invariant: x + y = N
```

```
\{x = 0 \& y = N \& N > 0\} while (y > 0) \{y = y - 1; x = x + 1\} \{x = N\}
Invariant: x + y = N
• x = 0 \& y = N \& N > 0 => x + y = N
```

```
\{x = 0 \& y = N \& N > 0\} while (y > 0) \{y = y - 1; x = x + 1\} \{x = N\}

Invariant: x + y = N

• x = 0 \& y = N \& N > 0 \Rightarrow x + y = N

• y <= 0 \& x + y = N \Rightarrow x = N

nope!
```

$$\{x = 0 \& y = N \& N > 0\} \text{ while } (y > 0) \{y = y - 1; x = x + 1\} \{x = N\}$$
 Invariant: $x + y = N \& y >= 0$

$$\bullet \quad x = 0 \& y = N \& N > 0 \qquad => \quad x + y = N \& y >= 0$$

$$\bullet \quad y <= 0 \& x + y = N \& y >= 0 \qquad => \quad x = N$$
 add constraint to invariant invariant invariant initially easier to establish postcondition

$$\{x = 0 \& y = N \& N > 0\} \text{ while } (y > 0) \{y = y - 1; x = x + 1\} \{x = N\}$$
 Invariant: $x + y = N \& y >= 0$

$$\bullet \quad x = 0 \& y = N \& N > 0 \qquad => \quad x + y = N \& y >= 0$$

$$\bullet \quad y <= 0 \& x + y = N \& y >= 0 \qquad => \quad x = N$$
 add constraint to invariant invariant invariant initially easier to establish postcondition

While Statements: Summary

Given a Hoare triple for a while loop:

```
— { P } while (e > 0) do C { Q }
```

- We prove it correct by:
 - guessing an invariant | (this is the hard part)
 - proving I holds initially: P => I
 - showing the loop body preserves I:

```
• {e > 0 & I} C{I}
```

– showing the postcondition holds on loop termination:

```
• 1 \& (e > 0) => Q
```

As a rule:

```
P \Rightarrow I { e > 0 \& I } C \{ I \} | I \& (e > 0) \Rightarrow Q 
 { P \} while (e > 0) do C \{ Q \}
```

- Note: one often adds I as an annotation on the loop:
 - while [I] (e > 0) do C

FRAMING & MODULARITY

Another Issue: Framing

Another valid triple:

$$\{x = 9 \& y = 7 \& z = 23\}x = x + 1 \{x = 10 \& y = 7 \& z = 23\}$$

Proving it using the rules:

Another Issue: Framing

Another valid triple:

$$\{x = 9 \& y = 7 \& z = 23\}x = x + 1 \{x = 10 \& y = 7 \& z = 23\}$$

Proving it using the rules:

```
(1) \{x + 1 = 10 \& y = 7 \& z = 23 \} x = x + 1 \{x = 10 \& y = 7 \& z = 23 \} (assignment rule)
```

(2)
$$x = 9 \& y = 7 \& z = 23 \implies x + 1 = 10 \& y = 7 \& z = 23$$
 (implication)

(3)
$$\{x = 9 \& y = 7 \& z = 23\}x = x + 1 \{x = 10 \& y = 7 \& z = 23\}$$
 (by (1), (2), consequence)

Another Issue: Framing

Another valid triple:

$$\{x = 9 \& y = 7 \& z = 23\}x = x + 1 \{x = 10 \& y = 7 \& z = 23\}$$

• Proving it using the rules:

```
(1) \{x + 1 = 10 \& y = 7 \& z = 23\} x = x + 1 \{x = 10 \& y = 7 \& z = 23\} (assignment rule)
```

(2)
$$x = 9 \& y = 7 \& z = 23 \Rightarrow x + 1 = 10 \& y = 7 \& z = 23$$
 (implication)

(3)
$$\{x = 9 \& y = 7 \& z = 23\}x = x + 1 \{x = 10 \& y = 7 \& z = 23\}$$
 (by (1), (2), consequence)

- Note: Formulae not involving x are just propagated
- More generally, conjuncts not involving variables that are not modified are just propagated
- Can we factor those expressions out of most of the proof?

The Simple Frame Rule

The Simple Frame Rule (also called the rule of constancy)

```
{P}C{Q} C does not modify the (free) variables of R {P&R}C{Q&R}
```

- What counts as "modifying"?
 - In our simple language, the only way a variable may be modified is if it appears on the left in an assignment statement
 - In languages with functions or methods, calling one of them may have a modification effect
 - In C, you might be able to intentionally modify variables on the stack
 - In C, you might also have a buffer overflow ... yikes!
- The frame rule is a way of *simplifying* proofs
- Why are Haskell proofs so easy? Nothing is modified!

The Simple Frame Rule

The Simple Frame Rule (also called the rule of constancy)

```
{P}C{Q} C does not modify the (free) variables of R {P&R}C{Q&R}
```

Example:

```
\{x = 6 \& y = 7 \& z = 23\}x = x + 1; x = x * 2; x = x - 4; \{x = 10 \& y = 7 \& z = 23\}
```

The Simple Frame Rule

The Simple Frame Rule (also called the rule of constancy)

```
{P}C{Q} C does not modify the (free) variables of R {P&R}C{Q&R}
```

Example:

```
x = x + 1; x = x * 2; x = x - 4;

x = x + 1; x = x * 2; x = x - 4;

x = x + 1; x = x * 2; x = x - 4;

x = x + 1; x = x * 2; x = x - 4;

x = x + 1; x = x * 2; x = x - 4;

x = x + 1; x = x * 2; x = x - 4;

x = x + 1; x = x * 2; x = x - 4;

x = x + 1; x = x * 2; x = x - 4;

x = x + 1; x = x * 2; x = x - 4;

x = x + 1; x = x * 2; x = x - 4;

x = x + 1; x = x * 2; x = x - 4;

x = x + 1; x = x * 2; x = x - 4;

x = x + 1; x = x * 2; x = x - 4;

x = x + 1; x = x * 2; x = x - 4;

x = x + 1; x = x * 2; x = x - 4;

x = x + 1; x = x * 2; x = x - 4;

x = x + 1; x = x * 2; x = x - 4;

x = x + 1; x = x * 2; x = x - 4;

x = x + 1; x = x * 2; x = x - 4;
```

SUMMARY!

Summary

- States map variables to values
- Formulae describe states:
 - semantics in Haskell: fsem :: State -> Form -> Maybe Bool
 - semantics in Math: [[f]]s
 - formulae and states we deal with are well-formed
 - well-formedness is a very simple syntactic analysis
 - P => Q means P describes a subset of the states that Q does
- Hoare Triples characterize program properties
 - { P } C { Q } know when it is valid
 - know the statement rules you can use to conclude { P } C { Q }
 - understand the structural rules:
 - rule of consequence
 - frame rule
 - know how to build formal proofs using inference rules