

# Hoare style program verification

#### Abhik Roychoudhury National University of Singapore

IISc Summer Course 2007 by Abhik Roychoudhury



#### Remarks

- SW Model Checking
  - Automated
  - Reason about transition systems.
  - Abstractions can make the reasoning imprecise albeit conservative.
- Theorem Proving
  - User-guided
  - Reason about programs.
  - Exact reasoning.

IISc Summer Course 2007 by Abhik Roychoudhury



#### Remarks

- SW Model Checking
  - Abstractions designed for a given program and/or property.
  - Path-sensitive.
    - False alarms, but because abstraction was coarse
  - Property required.
- Static Analysis
  - Abstract domain fixed for all programs in a PL, depending on the analysis performed.
  - Usually analysis results of diff paths are merged.
  - Hence false alarms
  - Abst. domain all imp.

IISc Summer Course 2007 by Abhik Roychoudhury



#### Remarks

- The approach of developing proof rules for reasoning about language constructs is radically different from model checking
  - Reason about programs (not transition systems)
  - Non-mechanized.
  - Notion of distinguished control locations ingrained
     Reason about pre- and post-conditions holding before and after execution of a block of code.
- · Consider sequential programs in this lecture
  - Can extend to develop proof rules for multithreaded programs.

IISc Summer Course 2007 by Abhik Roychoudhury



## Hoare triple

- {Pre} P {Post}
  - If P is run from a state where Pre holds and P terminates, then Post holds in the end-state [Partial correctness]
  - If P is run from a state where Pre holds, then P terminates and Post holds in the end-state [Total correctness]
  - A Hoare triple involving program P is a specification about P.

IISc Summer Course 2007 by Abhik Roychoudhury



## Trivial example

- Say P is while true do x = 0 endwhile
- P is partially correct w.r.t. any specification of the form {Pre} P {Post}
- P is not totally correct w.r.t. any specification of the form {Pre} P {Post}
- We will develop a proof system for reasoning about partial correctness
  - First step to reasoning about total correctness



#### **Notations**

- |-par {Pre} P {Post}
  - The Hoare triple can be shown to be partially correct in our proof system
- |= |= | Pre Post
  - The Hoare triple is partially correct.
- |-tot {Pre} P {Post}, |=tot {Pre} P {Post}
  - Similar
  - Standard notions of soundness/completeness

IISc Summer Course 2007 by Abhik Roychoudhury



## Factorial program

- $\{x \geq 0\}$
- $\{x \ge 0\}$
- /\* x is input \*/
- /\* x is input \*/
- y = 1; z = 0;
- y = 1;
- while (z != x) do
- while (x != 0) do
- z = z + 1;y = y \*z;
- y = y \*x;
- y y
- x = x 1;
- endwhile
- endwhile
- { y = x! }
- { ??? }

IISc Summer Course 2007 by Abhik Roychoudhury



## The problem

- x was destructively updated in Program2
  - In the end-state, we cannot say y = x!
  - To state correctness conditions, not enough to use program variables
  - Need to remember the original value of x
  - {x =x0/\ x≥0} Program2 {y = x0!}
  - x0 is a universally quantified logical variable.

IISc Summer Course 2007 by Abhik Roychoudhury



#### Logical variables

- {x =x0 ∧ x≥0} Program2 {y = x0!}
  - For all x0, if x = x0 and x ≥ 0 and we run Program2 such that it terminates, we will have y = x0! in the end state.
  - These variables appear only in the logical formulae of pre- and post-conditions.
  - Never appear in the program being verified.
- We now present the proof rules of our proof system.

IISc Summer Course 2007 by Abhik Roychoudhury



#### **Proof Rules**



Conclusion

Both premises and conclusion are Hoare triples.

If premises specify properties about programs C1, C2, ..., Cn

-- the conclusion specifies a property about a bigger program C typically containing C1,C2,...,Cn

IISc Summer Course 2007 by Abhik Roychoudhury



#### Rule for Assignment

 $\{\psi\,[\,x\to E]\,\}\ x=E\ \{\psi\,\}$ 

No premises in this rule.

To prove  $\psi$  after the assignment,  $\psi$  [  $x\to E]$  should hold before the assignment.



## Why not forwards?

- {φ} x = E {ψ}
  - $\blacksquare$  How to define  $\psi$  in terms of  $\phi$  ?
  - Cannot be achieved mechanically in general
  - The backwards formulation of the rule allows deducing Hoare triple by mechanically substituting
  - Instead define  $\varphi$  in terms of  $\psi$

IISc Summer Course 2007 by Abhik Roychoudhury



# **Sequential Composition**

{φ}C1 {ψ1} {ψ1} C2 {ψ}

 $\{\phi\}$  C1; C2  $\{\psi\}$ 

Need assertion for end-state of C1 and begin-state of C2.

IISc Summer Course 2007 by Abhik Roychoudhury



#### **If-statement**

{φ∧b}C1{ψ}

 $\{\phi \land \neg b\} \ C2 \, \{\, \psi \, \}$ 

{φ} if b then C1 else C2 {ψ}

Involves a case-split.

Pre-condition typically does not say anything about b Needs to augmented with truth/falsehood of b.

IISc Summer Course 2007 by Abhik Roychoudhury



#### While statement

 $\{\,\psi \wedge b\}\,C\,\,\{\psi\,\}$ 

 $\{\psi\}$  while b do c  $\{\psi \land \neg b\}$ 

 $\psi$  is the loop invariant.

Rule for partial correctness (number of times the loop executes/ termination is not known/ not guaranteed).

IISc Summer Course 2007 by Abhik Roychoudhury



# **Implications**

 $\phi' \Rightarrow \phi \quad \{\phi\} C \{\psi\}$ 

 $\psi \Rightarrow \psi'$ 

 $\{\phi'\} \ C \ \{\psi'\}$ 

- 1. Strengthening the pre-condition
- 2. Weakening the post-condition

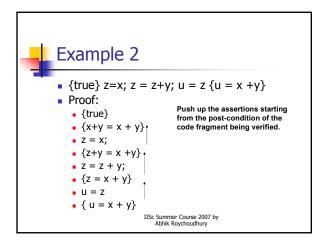
Why do we need this rule?

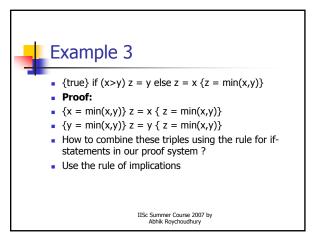
IISc Summer Course 2007 by Abhik Roychoudhury

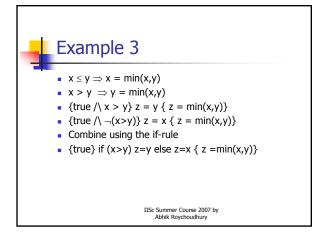


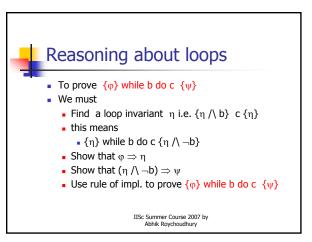
## Example 1

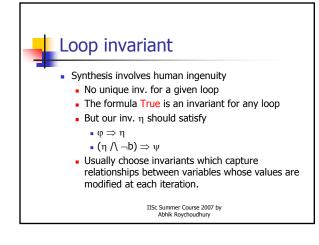
- $\{y < 2\}$  y = y + 1  $\{y < 5\}$
- Proof:
- {y < 2}
- {y +1 < 5} *implication rule*
- y = y+1
- {y < 5} assignment rule

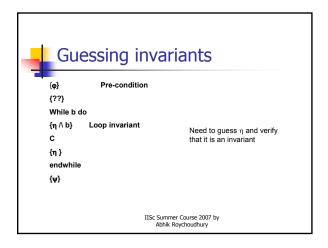


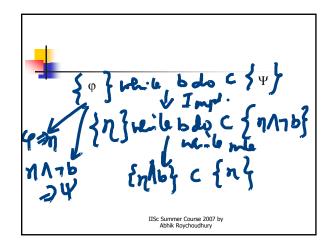












```
Factorial program

• {true}
• y = 1; z = 0; Guess the loop invariant
• {y = z!} y = z!
• while (z!= x) do
• z = z + 1; y = y *z
• endwhile
• {y = x!}

IISc Summer Course 2007 by
Abhik Roychoudhury
```

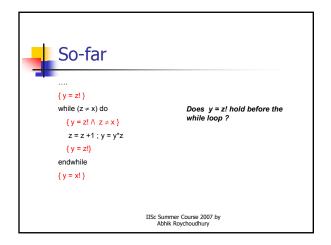
```
Checking the post-loop states

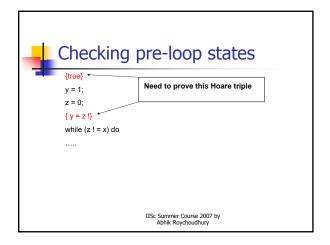
\{\text{true}\} \qquad y = z! \land \neg (z \neq x)
y = 1; z = 0; \qquad = y = z! \land z = x
\{y = 0!\} \qquad = y = x!
while (z \mid = x) do
z = z + 1; y = y^*z;
endwhile
\{y = z! \land \neg (z \neq x)\} \qquad \textit{Implication}
\{y = x!\}
IISc Summer Course 2007 by
Abbik Roychoudhury
```

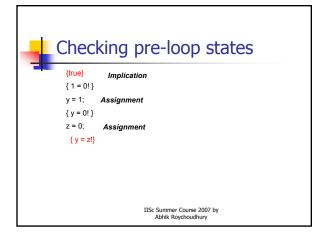
```
Verifying the invariant

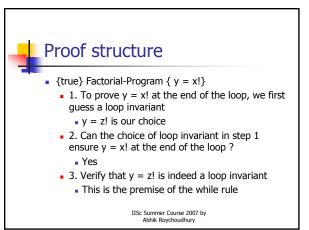
{rue}
y=1; z=0;
\{y=0!\}
while \{z!=x\} do
\{y=z! \land z=x+1;
y=y^2z;
\{y=z!\}
endwhile

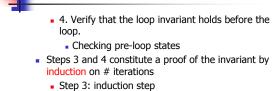
IISc Summer Course 2007 by
Abhik Roychoudhury
```











**Proof structure** 

Step 4: base case of the proofThe loop invariant itself is the ind. Hypothesis, no

IISc Summer Course 2007 by Abhik Roychoudhury

strengthening involved in this proof.



- The loop invariant must be strong enough to be "proved" an invariant.
  - The while rule is essentially accomplishing induction on # of loop iterations.
  - Often guided by the choice of the post-condition after the loop
    - Our post-condition was y = x!
    - Since z = x at loop exit and z is modified at every loop iteration, choose y = z! as invariant.



#### Proving total correctness

- Our proof system only shows partial correctness of triples  $\{\phi\}$  P  $\{\psi\}$
- To prove total correctness
  - Need to prove termination
  - Only the proof rule for while statement needs to change.
  - To prove termination
    - Find a non-negative integer quantity which decreases in every iteration ( call it variant )

IISc Summer Course 2007 by Abhik Roychoudhury



#### Finding variant

- a = x; y = 1;
- while (a > 0) do
- y = y\*a; a = a-1;
- endwhile
- Trivial to find the variant
  - a in this case

IISc Summer Course 2007 by Abhik Roychoudhury



## Finding variant

- y= 1; z = 0;
- while (z != x) do
- z = z + 1; y = y\*z
- endwhile
- Variant is x z (lifted from loop guard here)
- In general, finding variant cannot be automated even if the loop is guaranteed to terminate.

IISc Summer Course 2007 by Abhik Roychoudhury



#### **New Proof Rule**

 $\{\eta \wedge b \wedge (E = E0 \ge 0)\} \quad C \quad \{\eta \wedge (E0 > E \ge 0)\}$ 

 $\{ \eta \land E \ge 0 \}$  while b do c  $\{ \eta \land \neg b \}$ 

E is the variant.

If it is E0 before the loop, it strictly decreases but remains non-

Of course E should be non-negative before the loop starts. IISc Summer Course 2007 by Abhik Roychoudhury



## Factorial program

- $\{ x \ge 0 \}$
- y = 1; z = 0;

Use the variant x - z to prove termination

while (z != x) do

Use the loop invariant y =z! as before z = z + 1;for proving partial correctness

y = y\*z;

- endwhile
- $\{ y = x! \}$

IISc Summer Course 2007 by Abhik Roychoudhury



 $\{y = x!\}$ 

#### Reasoning about the loop

 $\{y = z! \land x - z \ge 0\}$ while (x! = z) do z = z + 1; y = y \* z; endwhile;  $\{ y = z! \land x = z \} \leftarrow$ 

From the conclusion of the while rule (total correctness)

-- How to show the premise ?

