



**BITS Pilani**  
Pilani Campus



# CS/IS F214 Logic in Computer Science

## MODULE: TEMPORAL LOGICS

### Formal (Program / System) Verification: Introduction Approaches

# What is Formal Verification?

- Formal Verification is often done to verify (and therefore ensure) correctness of a program (or a computing system):
  - the goal is to provide a guarantee that the system is correct as stated formally.
- The word **Formal** here refers to systematically constructed (i.e. formed) verification
  - e.g. a proof (using axioms and rules) is constructed to verify a statement (specified in a logic)



# Formal Verification vs. Testing

- Contrast this with testing:
  - Testing cannot be exhaustive always!
    - Why?
    - Example(s)?
  - So it is an empirical approach to providing a guarantee.
    - In practice, it is often quantified as a probabilistic guarantee.
      - i.e.
        - Testing a program with a certain number and kinds of test cases

is translated to

  - a probable correctness measure



# Formal Verification of Systems

- **Formal Verification** relates to proving properties of systems such as
  - *hardware, communication protocols, software, or a combination of these.*
- Formal Verification frameworks typically include:
  1. a modeling language  
for describing a system  
examples?
  2. a specification language  
for describing properties of a system  
examples?
  3. a verification method  
for establishing whether the system (as defined by a model) satisfies the specification





**BITS Pilani**  
Pilani Campus



# CS/IS F214 Logic in Computer Science

## MODULE: **PROGRAM VERIFICATION**

### **Program Verification – Floyd-Hoare Logic**

# Program Correctness

- Correctness Arguments (for a given sequential program **A**):
  - a) Does **A** compute what it is expected (or required) to?*  
Assumption: ?
  - b) Does **A** terminate?*



# Program Correctness

- Correctness Arguments (for a given program A):
  - a) Does A compute what it is expected (or required) to?*  
Assumption: ?
  - b) Does A terminate?*
- Partial Correctness vs. Total Correctness
  - **Partial Correctness** refers to correctness argument stated in (a) above, assuming that the program terminates.
  - **Total Correctness** refers to correctness arguments stated in (a) and (b) above.



# Partial Correctness

- Statement of (Partial) Correctness is a contract:  
/\* **Pre-condition:** Input **x** satisfies some properties \*/  
**A(x)**  
/\* **Post-condition:** Running **A** on **x** results in ... \*/  
  
i.e. *if the state before executing A satisfies the **pre-condition** (on input x)  
then the state after executing A would satisfy the **post-condition** (on the results)*





# Partial Correctness - Examples

- Contracts: Examples

**/\* Pre-condition:  $x \geq 0$  \*/**

**F(x) { ... }**

**/\* Post-condition:  $F(x) = x!$  \*/**

**/\* Pre-condition:  $N > 0$  \*/**

**F(Ls, N) { ... }**

**/\* Post-condition:  $\forall X \ (0 \leq X) \wedge (X < N-1) \rightarrow Ls[X] \leq Ls[X+1]$  \*/**



# Floyd-Hoare Logic

- A logic system for proving correctness of “**imperative**” programs:
  - **Imperative programming** is a style of programming that is based on ***state transformations***: i.e.
    - a program is specified as a sequence of statements
    - each statement is a transformation on the state (i.e. data stored in memory)
- Examples of imperative programming:
  - Programming at assembly/machine level (e.g. using x86 Instruction set)
  - Programming using C (or similar languages)



# Imperative Programming

- Imperative programs are usually constructed out of the following types of statements:

## 1. Assignment statement

Change the state *by changing the contents of one variable*

## 2. Sequencing

*Control the order of execution of statements*

## 3. Conditional Statement

Choose one of two *sequences of statements to execute*

## 4. Iterative Statement

Repeat *a sequence of statements*



# Partial Correctness - Examples (Revisited)

- Contracts: Examples

**/\* Pre-condition:  $x \geq 0$  \*/**

**S1**

**...**

**Sn**

**/\* Post-condition:  $F(x) = x!$  \*/**

**/\* Pre-condition:  $N > 0$  \*/**

**S1**

**...**

**Sn**

**/\* Post-condition:  $\forall X (0 \leq X) \wedge (X < N-1) \rightarrow Ls[X] \leq Ls[X+1]$  \*/**



# Floyd-Hoare Logic (a.k.a. Hoare Logic)

- Hoare Logic reduces the correctness argument for a program

/\* Pre-condition: Input  $x$  satisfies some properties \*/

**S1**

...

**Sn**

/\* Post-condition: Running A on  $x$  results in ... \*/

- to each statement in the program:

/\*  $p_0$  \*/                      such that

**S1**                       $p_N$  is the required post-condition (for the entire program)

/\*  $p_1$  \*/

**S2**                       $p_0$  is the given pre-condition (for the program)

/\*  $p_2$  \*/

...                       $p_i$  is the post-condition and  $p_{i-1}$  is the pre-condition for each  $S_i$ , for  $i$  from 1 to  $N$

/\*  $p_N$  \*/



# Floyd-Hoare Logic (a.k.a. Hoare Logic)

- We refer to  
     $\langle \underline{\text{PRE}}, S, \underline{\text{POST}} \rangle$   
as a *Hoare-triple*
  - where  $S$  is a statement
  - $\underline{\text{PRE}}$  is the pre-condition and
  - $\underline{\text{POST}}$  is the post-condition
- What does this mean?
  - If  $\underline{\text{PRE}}$  is *satisfied by a state*,
  - then *executing statement  $S$  on that state*
    - would *result in a state that satisfies  $\underline{\text{POST}}$*





**BITS Pilani**  
Pilani Campus



# CS/IS F214 Logic in Computer Science

## MODULE: **PROGRAM VERIFICATION**

### **Floyd-Hoare Logic: Correctness of Assignment Statements**

# Assignment Statements

- Assignment statements change the **state** of a program:
  - Typically, an assignment statement changes the value of one variable.
    - e.g.  $v = v + 5$
    - e.g.  $z = x * y$
    - e.g.  $y = \text{sqrt}(x)$





# Correctness of Assignment Statements

- How do we argue the correctness of an assignment statement?
  - e.g.  $v = v + 5$
  - e.g.  $z = x * y$
  - e.g.  $a = \text{gcd}(x, y)$
- We try to relate *a desired post-condition* with *a required pre-condition*.



# Assignment Statements: Pre-conditions and Post-conditions

- Example:
  - What would be the pre-condition for the following assignment statement for (each of) the given post-condition(s)?
    - $v = v + 5$ 
      - i. /\* Post-condition:  $v > 0$  \*/
      - ii. /\* Post-condition:  $v < 10$  \*/
      - iii. /\* Post-condition:  $v * v < 100$  \*/



# Floyd-Hoare Logic: Assignment Statements

- Rule for Assignment Statements:

*/\* Pre:  $\phi$  [e / v] \*/*

**v = e**    */\* e is any expression \*/*

*/\* Post:  $\phi$  with v as free variable \*/*

$\langle \phi [e/v], v=e, \phi \rangle$

Assignment

*Note on Notation:*

We specify rules in Hoare logic in two forms:

(i) in programming style syntax, with *pre-condition* and *post-condition* expressed *inside comments*.

(ii) in proof-rules style syntax using a Hoare-triple in angle brackets.

*End of Note.*



# Floyd-Hoare Logic: Examples

- Exercise 1:

`/* ? */ mid = (x + y)/2; /*mid is the average of x and y*/`

- Exercise 2:

`/* ? */ mid = x + (y-x)/2; /*mid is the average of x and y*/`





**BITS Pilani**  
Pilani Campus



# CS/IS F214 Logic in Computer Science

## MODULE: **PROGRAM VERIFICATION**

### **Floyd-Hoare Logic: Correctness of Sequencing**

# Floyd-Hoare Logic: Sequencing (or Composition)

- Rule for Sequencing

$/* \varphi_1 */$

**S1**

$/* \varphi_2 */$

**S2**

$/* \varphi_3 */$

Post-condition for **S1**  
and  
Pre-condition for **S2**

## Sequence

$\langle \varphi_1, S1, \varphi_2 \rangle \langle \varphi_2, S2, \varphi_3 \rangle$

---

$\langle \varphi_1, S1; S2, \varphi_3 \rangle$



# Floyd-Hoare Logic: Sequencing (or Composition)

- Rule for Sequencing

*/\*  $\varphi_1$  \*/*

**S1**

*/\*  $\varphi_2$  \*/*

**S2**

*/\*  $\varphi_3$  \*/*

Post-condition for S1  
and  
Pre-condition for S2

## Sequence

$\langle \varphi_1, S1, \varphi_2 \rangle \langle \varphi_2, S2, \varphi_3 \rangle$

---

$\langle \varphi_1, S1; S2, \varphi_3 \rangle$

- What does this mean?

- If statement **S1** transforms a state satisfying  $\varphi_1$  to a state satisfying  $\varphi_2$  and
- if statement **S2** transforms a state satisfying  $\varphi_2$  to a state satisfying  $\varphi_3$  then
- then the sequence **S1; S2** transforms a state satisfying  $\varphi_1$  to a state satisfying  $\varphi_3$

# Floyd-Hoare Logic: Examples

- Exercise S1:

- $/* \ ? \ */ \quad t=x; x=y; y=t \quad /* \ x = A \wedge y = B \ */$

- Exercise S2:

- $/* \ ? \ */ \quad x = x+y; y=x-y; x=x-y \quad /* \ x = A \wedge y = B \ */$

