# IA169 System Verification and Assurance

## Deductive Verification

Jiří Barnat

# Verification of Algorithms

**Validation and Verification**

- A general goal of V&V is to prove correct behaviour of algorithms.

**Reminder**

- Testing is incomplete.
- Testing can detect errors but cannot prove correctness.
- Model checking is limited.
- Requires finite state space description, suffers from state space explosion, verifies specific program properties only.

**Conclusion**

- Need for yet another way of verification.

# Formal Verification

**Goal of formal verification**

- The goal is to show that system behaves correctly with the same level of confidence as it is given with a mathematical proof.

**Requirements**

- Formally precise semantics of system behaviour.
- Formally precise definition of system properties to be shown.

**Methods of formal verification**

- Model checking
- Deductive verification
- Abstract interpretation

# Deductive verification

# Notion of Correctness

**Program is correct if ...**

- ... it **terminates** for a valid input and returns **correct** output.
- There is a need to show two parts – partial correctness and termination.

**Partial correctness** (Correctness, Soundness)

- If the computation terminates for valid input values (i.e. values for which the program is defined) the resulting values are correct.

**Termination** (Completness, Convergence)

- If executed on valid input values, the computation always terminates.

# Verification of Serial Programs

**Serial programs** (sequential)

- Input-output-closed programs.
  - All input values are known prior program execution.
  - All output values are stored in output variables.
- Examples: Quick sort, Greatest Common Divider, ...

**General Principle**

- Program instructions are viewed as state transformers.
- The goal is to show that the mutual relation of input and output values is as expected or given by the specification.
- To verify the correctness of procedure of transformation of input values to output values.

# Expressing Program Properties

**State of Computation**

- State of computation of a program is given by the value of program counter and values of all variables.

**Atomic predicates**

- Basic statements about individual states of the computation.
- The validity is deduced purely from the values of variables given by the state of computation.
- Examples of atomic propositions: $(x == 0)$, $(x1 >= y3)$.
- Beware of the scope of variables.

**Set of States**

- Can be described with a Boolean combination of atomic predicates.
- Example: $(x == m) \wedge (y > 0)$

## Assertion

- For a given program location defines a Boolean expression that should be satisfied with the current values of program variables in the given location during program execution.
- Invariant of a program location.

## Assertions – Proving Correctness

- Assigning properties to individual locations of Control Flow Graph.
- Robert Floyd: Assigning Meanings to Programs (1967)

# Error Detection — Assertion Violation

**Testing**

- Assertion violation serves as a test oracle.

**Run-Time Checking**

- Checking location invariants during run-time.
- Improved error localisation as the assertion violated relates to a particular program line.

**Undetected Errors**

- If an error does not manifest itself for the given input data.
- If the program behaves non-deterministically (parallelism).

# Hoare Proof System

# Hoare Proof System

**Principle**

- Programs = State Transformers.
- Specification = Relation between input and output state of computation.

**Hoare logic**

- Designed for showing partial correctness of programs.
- Let $P$ and $Q$ be predicates and $S$ be a program, then

$$\{\mathrm{P}\}\ \mathrm{S}\ \{\mathrm{Q}\}$$

  is the so called *Hoare triple*.

**Intended meaning of $\{P\}\ S\ \{Q\}$**

- $S$ is a program that transforms any state satisfying *pre-condition* $P$ to a state satisfying *post-condition* $Q$.

# Pre- and Post- Conditions

**Example**

- $\{z = 5\}\ x = z * 2\ \{x > 0\}$
- Valid triple, though post-condition could be more precise (stronger).
- Example of a stronger post-condition: $\{x > 5 \wedge x < 20\}$.
- Obviously, $\{x > 5 \wedge x < 20\} \implies \{x > 0\}$.

**The Weakest Pre-Condition**

- $P$ is the weakest pre-condition, if and only if
- $\{P\}S\{Q\}$ is a valid triple and
- $\forall P'$ such that $\{P'\}S\{Q\}$ is valid, $P' \implies P$.
- Edsger W. Dijkstra (1975)

# Proving in Hoare System

**How to prove {P} S {Q}**

- Pick suitable conditions P' a Q'
- Decomposition into three sub-problems:

$$\{P'\}\ S\ \{Q'\} \qquad P \implies P' \qquad Q' \implies Q$$

- Use axioms and rules of Hoare system to prove {P'} S {Q'}.
- $P \implies P'$ and $Q' \implies Q$ are called proof obligations.
- Proof obligations are **proven in the standard way.**

# Hoare System – Axiom for Assignment

**Axiom**

- Assignment axiom: $\{\phi[\text{x replaced with k}]\}\ x := k\ \{\phi\}$

**Meaning**

- Triple $\{P\}x := y\{Q\}$ is an axiom in Hoare system, if it holds that $P$ is equal to $Q$ in which all occurrences of $x$ has been replaced with $y$.

**Examples**

- {y+7>42} x:=y+7 {x>42} — is an axiom
- {r=2} r:=r+1 {r=3} — is not an axiom
- {r+1=3} r:=r+1 {r=3} — is an axiom

# Hoare Logic – Example 1

### Example

- Prove that the following program returns value greater than zero if executed for value of 5.
- Program: $out := in * 2$

### Proof

1) We built a Hoare triple:
   $\{in = 5\}\ out := in * 2\ \{out > 0\}$

2) We deduce/guess a suitable pre-condition:
   $\{in * 2 > 0\}$

3) We prove Hoare triple:
   $\{in * 2 > 0\}\ out := in * 2\ \{out > 0\}$ (axiom)

4) We prove auxiliary statement:
   $(in = 5) \implies (in * 2 > 0)$

# Hoare System – Example of a Rule

### Rule

- Sequential composition: $\frac{\{\phi\}S_1\{\chi\}\wedge\{\chi\}S_2\{\psi\}}{\{\phi\}S_1;S_2\{\psi\}}$

### Meaning

- If $S_1$ transforms a state satisfying $\phi$ to a state satisfying $\chi$ and $S_2$ transforms a state satisfying $\chi$ to a state satisfying $\psi$ then the sequence $S_1; S_2$ transforms a state satisfying $\phi$ to a state satisfying $\psi$.

### In the proof

- Should $\{\phi\}S_1; S_2\{\psi\}$ be used in the proof, an intermediate condition $\chi$ has to be found, and $\{\phi\}S_1\{\chi\}$ and $\{\chi\}S_2\{\psi\}$ have to be proven.

# Hoare System – Partial Correctness

Axiom for **skip**: $\{\phi\}$ **skip** $\{\phi\}$

Axiom for **:=**: $\{\phi[x := k]\}x\mathbf{:=}k\{\phi\}$

Composition rule: $\dfrac{\{\phi\}S_1\{\chi\}\wedge\{\chi\}S_2\{\psi\}}{\{\phi\}S_1;S_2\{\psi\}}$

Conditional rule: $\dfrac{\{\phi\wedge B\}S_1\{\psi\}\wedge\{\phi\wedge\neg B\}S_2\{\psi\}}{\{\phi\}\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}\{\psi\}}$

While rule: $\dfrac{\{\phi\wedge B\}S\{\phi\}}{\{\phi\}\textbf{while } B \textbf{ do } S \textbf{ od } \{\phi\wedge\neg B\}}$

Consequence rule: $\dfrac{\phi \Longrightarrow \phi',\{\phi'\}S\{\psi'\},\psi' \Longrightarrow \psi}{\{\phi\}S\{\psi\}}$

# Hoare Logic – Example 2

**Prove that for $n \geq 0$ the following code computes $n!$.**

- 
```
r = 1;

while (n≠ 0) {
   r = r * n;

   n = n - 1;
}
```

**Notes:**

## Hoare Logic – Example 2

**Prove that for $n \geq 0$ the following code computes $n!$.**

- $\{ \text{n} \geq 0 \wedge \text{t=n} \}$ $\{P\}$

```
r = 1;

while (n≠ 0) {
   r = r * n;

   n = n - 1;
}
```

$\{ \text{r=t!} \}$ $\{Q\}$

**Notes:**

- Reformulation in terms of Hoare logic.
- Note the use of auxiliary variable $t$.

**Prove that for $n \geq 0$ the following code computes $n!$.**

- $\{\ n \geq 0 \land t{=}n\ \}$ $\{P\}$

```
r = 1;
```

$\{\ n \geq 0 \land t{=}n \land r = 1\ \}$ $\{I_1\}$

```
while (n≠ 0) {
   r = r * n;

   n = n - 1;
}
```

$\{\ r{=}t!\ \}$ $\{Q\}$

**Notes:**

- $\{n \geq 0 \land t{=}n \land 1{=}1\}$ r=1 $\{\ n \geq 0 \land t{=}n \land r{=}1\ \}$
- $(n \geq 0 \land t{=}n) \implies (n \geq 0 \land t{=}n \land 1{=}1)$

# Hoare Logic – Example 2

**Prove that for $n \geq 0$ the following code computes $n!$.**

- $\{\ n \geq 0 \land t{=}n\ \}$ $\{P\}$

  ```
  r = 1;
  ```

  $\{\ n \geq 0 \land t{=}n \land r = 1\ \}$ $\{I_1\}$

  ```
  while (n≠ 0)
  ```

  $\{\ r{=}t!/n! \land t \geq n \geq 0\ \}$ `{` $\{I_2\}$

  ```
     r = r * n;

     n = n - 1;
  }
  ```

  $\{\ r{=}t!\ \}$ $\{Q\}$

**Notes:**

- Invariant of a cycle: $\{I_2\} \equiv \{\ r{=}t!/n! \land t \geq n \geq 0\ \}$
- $I_1 \implies I_2 \qquad (\ I_2 \land \neg(n{\neq}0)\ ) \implies Q$

## Hoare Logic – Example 2

**Prove that for $n \geq 0$ the following code computes $n!$.**

- $\{ \text{ n} \geq 0 \wedge \text{t=n } \}$ $\{P\}$

```
r = 1;
```

$\{ \text{ n} \geq 0 \wedge \text{t=n} \wedge r = 1 \}$ $\{I_1\}$

while (n$\neq$ 0) $\{ \text{ r=t!/n!} \wedge \text{t} \geq \text{n} \geq 0 \}$ { $\{I_2\}$

```
   r = r * n;
```

$\{ \text{ r=t!/(n-1)!} \wedge \text{t} \geq \text{n} > 0 \}$ $\{I_3\}$

```
   n = n - 1;
}
```

$\{ \text{ r=t! } \}$ $\{Q\}$

**Notes:**

- $\{ \text{ r*n} = \text{t!/(n-1)!} \wedge \text{t} \geq \text{n} > 0 \}$ r=r*n $\{I_3\}$
- $I_2 \wedge (\text{n}\neq 0) \implies ( \text{ r*n} = \text{t!/(n-1)!} \wedge \text{t} \geq \text{n} > 0 )$

# Hoare Logic – Example 2

**Prove that for $n \geq 0$ the following code computes $n!$.**

- $\{\ n \geq 0 \wedge t{=}n\ \}$ $\{P\}$

  r = 1;

  $\{\ n \geq 0 \wedge t{=}n \wedge r = 1\ \}$ $\{I_1\}$

  while (n$\neq$ 0) $\{\ r{=}t!/n! \wedge t \geq n \geq 0\ \}$ { $\{I_2\}$

  r = r * n;

  $\{\ r{=}t!/(n{-}1)! \wedge t \geq n > 0\ \}$ $\{I_3\}$

  n = n - 1;

  }

  $\{\ r{=}t!\ \}$ $\{Q\}$

**Notes:**

- $\{\ r = t!/(n{-}1)! \wedge t \geq (n{-}1) \geq 0\ \}$ n=n-1 $\{I_2\}$
- $I_3 \implies (\ r = t!/(n{-}1)! \wedge t \geq (n{-}1) \geq 0\ )$

# Hoare Logic and Completness

**Observation**

- Hoare logic allowed us to reduce the problem of proving program correctness to a problem of proving a set of mathematical statements with arithmetic operations.

**Notice about correctness's and (in)completeness**

- Hoare logic is correct, i.e. if it is possible to deduce $\{P\}S\{Q\}$ then executing program $S$ from a state satisfying $P$ may terminate only in a state satisfying $Q$.
- If a proof system is strong enough to express integral arithmetics, it is necessarily incomplete, i.e. there exists claims that cannot be proven or dis-proven using the system.
- Hoare system for proving correctness of programs is incomplete due to the proof obligations generated with the consequence rule.

**Troubles with Proof Construction**

- Often pre- and post- condition must be suitable reformulated for the purpose of the proof.
- It is very difficult to identify loop invariants.

**Partial Correctness in Practice**

- Often reduced to formulation of all the loop invariants, and demonstration that they actually are the loop invariants.
- The proof of being an invariant is often achieved with math induction.

**Well-Founded Domain**

- Partially ordered set that does not contain infinitely decreasing sequence of members.
- Examples: $(\mathbf{N},<)$, $(\mathit{PowerSet}(\mathbf{N}),\subseteq)$

**Proving Termination**

- For every loop in the program a suitable well-founded domain and an expression over the domain is chosen.
- It is shown that the value associated with a location cannot grow along any instruction that is part of the loop.
- It is shown that there exists at least one instruction in the loop that decreases the value of the expression.

# Automating Deductive Verification

# Principles of Automation of Deductive Verification

**Pre-processing**

- Transformation of program to a suitable intermediate language.
- Examples of IL: Boogie (Microsoft Research), Why3 (INRIA)

**Structural Analysis and Construction of the Proof Skeleton**

- Identification of Hoare triples, loop invariants and suitable pre- and post-conditions (some of that might be given with the program to be verified).
- Generation of auxiliary proof obligations.

**Solving proof obligations**

- Using tools for automated proving.
- May be human-assisted.

# Solving Proof Obligations

**Tools for Automated Proving**

- User guides a tool to construct a proof.
- HOL, ACL2, Isabelle, PVS, Coq, ...

**Reduced to the satisfiability problem**

- Employ SAT and SMT solvers.
- Z3, ...

**Proof**

- A finite sequence of steps that using axioms and rules of a given proof system that transforms assumptions $\psi$ into the conclusion $\varphi$.

**Observation**

- For systems with finitely many axioms and rules, proofs may be systematically generated. Hence, for all provable claims the proof can be found in finite time.
- All reasonable proving systems has infinitely many axioms. Consider, e.g. an axiom $x = x$. This is virtually a shortcut (template) for axioms $1 = 1$, $2 = 2$, $3 = 3$, etc.
- Semi-decidable with dove-tailing approach.

**Searching for a Proof of Valid Statement**

- The number of possible finite sequences of steps of rules and axiom applications is too many (infinitely many).
- In general there is no algorithm to find a proof in a given proof system even for a valid statement.
- Without some clever strategy, it cannot be expected that a tool for automated proof generation will succeed in a reasonable short time.
- The strategy is typically given by an experienced user of the automated proving tool. The user typically has to have appropriate mathematical feeling and education.
- At the end, the tool is used as a mechanical checker for a human constructed proof.

**Theorem Provers**

- The goal is find the proof within a given proof system.
- the proof is searched for in two modes:
  - Algorithmic mode – Application of rules and axioms
    - Guided by the user of the tool.
    - Application of the general proving techniques, such as deduction, resolution, unification, . . . .
  - Search mode – Looking for new valid statements
    - Employs brute-force approach and various heuristics.

**Existing Tools**

- The description of system (axioms, rules) as well as the claim to be proven is given in the language of the tool.

# Results of Proof Searching

**Possible Outputs**

a) Proof has been found and checked.

b) Proof has not been found.

- The statement is valid, can be proven, but the proof has not yet been found.
- The statement is valid, but it cannot be proven in the system.
- The statement is invalid.

**Observation**

- In the case that no proof has been found, there is no indication of why it is so.

# Dafny

`http://rise4fun.com/dafny`

## Homework

- Prove correctness of the following program using Dafny

```
method Count(N: nat, M: int, P: int) returns (R: int) {
  var a := M;
  var b := P;
  var i := 1;
  while (i <= N) {
    a := a+3;
    b := 2*a+b+1;
    i := i+1;
  }
  R := b;
}
```

- Read and repeat:

  **Jaco van de Pol**: *Automated verification of Nested DFS*

  http://dx.doi.org/10.1007/978-3-319-19458-5_12