

Deductive Verification II

COMP6210 - Automated Software Verification

ECS, University of Southampton

9th November 2020

Logic

We can **prove** correctness of our algorithms (and programs) if we

- understand what correctness means (i.e. the exact specification),
- can give a *completely convincing argument* that our code conforms to the correctness specification.

Logic

We can **prove** correctness of our algorithms (and programs) if we

- understand what correctness means (i.e. the exact specification),
- can give a *completely convincing argument* that our code conforms to the correctness specification.

Definition, according to *The Chambers Dictionary* (TCD):

logic /loj'ik/ (*noun*)

- 1 The science and art of reasoning correctly
- 2 The science of the necessary laws of thought

Completely Convincing Arguments

"If each man had a definite set of rules of conduct by which he regulated his life he would be no better than a machine. But there are no such rules, so men cannot be machines."

A. M. Turing, *Computing Machinery and Intelligence*, 1950.

Completely Convincing Arguments

"If each man had a definite set of rules of conduct by which he regulated his life he would be no better than a machine. But there are no such rules, so men cannot be machines."

A. M. Turing, *Computing Machinery and Intelligence*, 1950.

More formally stated as a *rule of inference*:

(1.) *There are definite rules of conduct* \rightarrow *Men are machines*

(2.) *There are no definite rules of conduct*

Men are not machines

Completely Convincing Arguments

"If each man had a definite set of rules of conduct by which he regulated his life he would be no better than a machine. But there are no such rules, so men cannot be machines."

A. M. Turing, *Computing Machinery and Intelligence*, 1950.

Following the same argument:

(1.) *You work for Google* \rightarrow *You are employed*

(2.) *You do not work for Google*

You are unemployed

Completely Convincing Arguments

"If each man had a definite set of rules of conduct by which he regulated his life he would be no better than a machine. But there are no such rules, so men cannot be machines."

A. M. Turing, *Computing Machinery and Intelligence*, 1950.

Following the same argument:

(1.) *You work for Google* \rightarrow *You are employed*

(2.) *You do not work for Google*

You are unemployed

This is an example of a logical fallacy – an incorrect form of argument where the premises (stated above the bar) do not entail the conclusion (stated below the bar).

Completely Convincing Arguments: Sound Inferences

A rule of inference is *sound* if the conclusion is a logical consequence of the premises.

In a sound rule of inference it is impossible for all the premises to be true and the conclusion to be false.

For example (*modus ponens*):

(1.) *You work for Google* \rightarrow *You are employed*

(2.) *You work for Google*

You are employed

Completely Convincing Arguments: Sound Inferences

A rule of inference is *sound* if the conclusion is a logical consequence of the premises.

In a sound rule of inference it is impossible for all the premises to be true and the conclusion to be false.

For example (*modus ponens*):

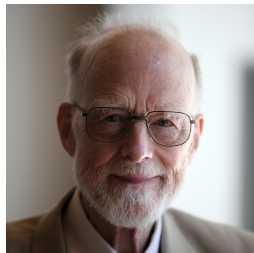
$$(MP) \frac{\begin{array}{c} A \rightarrow B \\ A \end{array}}{B}$$

Sound arguments can be constructed this way to prove *theorems* from *axioms* (self-evident statements that require no proof).

A formal proof can be stored in a file *and checked by a computer*.

Hoare Logic

- Introduced by **Sir Tony Hoare** in 1969.
- Followed in the footsteps of **R. W. Floyd** (who devised a similar system for flowcharts in 1967).
- Allows us to formally prove that a program satisfies its specification (contract).



Sir Tony Hoare

Side note: Hoare logic provides a way of formally defining the *semantics* of a programming language (so-called *axiomatic semantics*).

Hoare Logic

Hoare logic is a logic for reasoning about programs. It involves

- a few basic axioms,
- some sound rules of inference.

健全的规则

To state and prove properties of programs using Hoare logic we need to understand the following terms:

- preconditions,
- postconditions,
- **Hoare triples**,
- invariants,
- variants.

Preconditions

A **precondition** is a property that has to be true just before a program is executed (i.e. just before an operation is performed).

As a contract specification, a precondition can be treated as an *assumption* by the programmer.

An (informal) example:

“The initial values of x are in the range $[-10, 60]$.”

凭直觉地

Intuitively, if a program receives inputs (i.e. starts executing in a state) which *violates* the precondition, then it has no **obligation** to do anything sensible.

义务, 职责

Postcondition

A **postcondition** is a property that has to be true once the program is executed **and** terminates.

As a contract specification, a postcondition can be treated as an *requirement* (i.e. a contractual obligation to be fulfilled).

An (informal) example:

“The array will be stored in non-decreasing order when the program finishes.”

Intuitively, a program *promises* to achieve the postcondition (*provided that the precondition is met*). [It is also usually desirable for programs to terminate.]

Hoare Triples

A Hoare triple has the form:

$$\{PRE\} Prog \{POST\}$$

where

- PRE is the precondition,
- $POST$ is the postcondition,
- $Prog$ is the program.

A Hoare triple states that $Prog$ will establish $POST$ on completion, provided PRE holds before execution of $Prog$.

Hoare Logic: Some Examples of Hoare Triples

$$\{\mathbf{true}\} \mathbf{x:=10} \{x > 0\}$$

Hoare Logic: Some Examples of Hoare Triples

$$\{\mathbf{true}\} \ x:=10 \ \{x > 0\}$$
$$\{\mathbf{true}\} \ x:=10 \ \{x = 10\}$$

Hoare Logic: Some Examples of Hoare Triples

$$\{\mathbf{true}\} \mathbf{x}:=10 \{x > 0\}$$

$$\{\mathbf{true}\} \mathbf{x}:=10 \{x = 10\}$$

$$\{x > 2\} \mathbf{x}:=\mathbf{x}+1 \{x \leq 3\}$$

Hoare Logic: Some Examples of Hoare Triples

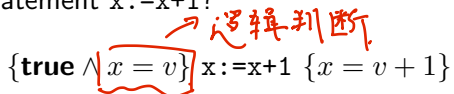
$$\{\mathbf{true}\} x:=10 \{x > 0\}$$

$$\{\mathbf{true}\} x:=10 \{x = 10\}$$

$$\{x > 2\} x:=x+1 \{x \leq 3\}$$

$$\{\mathbf{true}\} x:=x+1 \{???\}$$

How can we express that the value of x has increased by executing the program statement $x:=x+1$?


$$\{\mathbf{true} \wedge x = v\} x:=x+1 \{x = v + 1\}$$

where v is a *fresh variable*.

Hoare Logic

Axioms:

$$\overline{\{P\} \text{ skip } \{P\}}$$

$$\overline{\{P_0\} x := E \{P\}}$$

P_0 : 将 P 中的 x 替换为 E

In the assignment axiom (schema), P_0 denotes the statement P where every (free occurrence) of x is syntactically replaced by the expression E .

[In practice, to check a Hoare triple $\{PRE\} x := E \{POST\}$ we check $PRE \rightarrow POST_0$.]

Hoare Logic: Assignment Examples

In practice, to check a Hoare triple $\{PRE\} x := E \{POST\}$ we check $PRE \rightarrow POST_0$. Examples

$$\{x + 1 = 3\} x := x + 1 \{x = 3\}$$

$$\{x + 1 = 3 \wedge y = 0\} x := x + 1 \{x = 3\}$$

$$\{y + z = 5\} x := y + z \{x \geq 5\}$$

Hoare Logic: Rules of Inference

Rules (consequence):

$$\frac{\{P\} \text{ } \textit{prog} \text{ } \{R\} \qquad R \rightarrow S}{\{P\} \text{ } \textit{prog} \text{ } \{S\}}$$

Hoare Logic: Rules of Inference

Rules (consequence):

$$\frac{\{P\} \text{PROG} \{R\} \quad R \rightarrow S}{\{P\} \text{PROG} \{S\}}$$
$$\frac{\{P\} \text{PROG} \{R\} \quad S \rightarrow P}{\{S\} \text{PROG} \{R\}}$$

条件 P 在程序执行完满足条件 R
条件 S 推导出 P

Hoare Logic: Rules of Inference

Rules (composition):

$$\frac{\{P\} \text{ } PROG_1 \text{ } \{R_1\} \qquad \{R_1\} \text{ } PROG_2 \text{ } \{R\}}{\{P\} \text{ } PROG_1; PROG_2 \text{ } \{R\}}$$

Hoare Logic: Rules of Inference

Rules (composition):

$$\frac{\{P\} \text{ } PROG_1 \{R_1\} \qquad \{R_1\} \text{ } PROG_2 \{R\}}{\{P\} \text{ } PROG_1; \text{ } PROG_2 \{R\}}$$

Suppose we want to prove:

$$\{y = z\} \text{ } x:=z+1; \text{ } z:=y+1 \{z = x\}$$

Hoare Logic: Rules of Inference

Rules (composition):

$$\frac{\{P\} \text{ } \text{PROG}_1 \{R_1\} \qquad \{R_1\} \text{ } \text{PROG}_2 \{R\}}{\{P\} \text{ } \text{PROG}_1; \text{ } \text{PROG}_2 \{R\}}$$

Suppose we want to prove:

$$\{y = z\} \text{ } \mathbf{x := z + 1; \quad z := y + 1} \{z = x\}$$

Let $R_1 = y + 1 = x$, so we need to prove (left sub-goal):

$$\{y = z\} \text{ } \mathbf{x := z + 1} \{y + 1 = x\} \qquad (\text{check } y = z \rightarrow y + 1 = z + 1)$$

Hoare Logic: Rules of Inference

Rules (composition):

$$\frac{\{P\} \text{PROG}_1 \{R_1\} \quad \{R_1\} \text{PROG}_2 \{R\}}{\{P\} \text{PROG}_1; \text{PROG}_2 \{R\}}$$

Suppose we want to prove:

$$\{y = z\} \text{ x:=z+1; z:=y+1 } \{z = x\}$$

?

Let $R_1 = y + 1 = x$, so we need to prove (left sub-goal):

$$\{y = z\} \text{ x:=z+1 } \{y + 1 = x\} \quad (\text{check } y = z \rightarrow y + 1 = z + 1)$$

And the right sub-goal: $\{y + 1 = x\} \text{ z:=y+1 } \{z = x\}$

(check $y + 1 = x \rightarrow y + 1 = x$).

Hoare Logic: Rules of Inference

Rules (conditionals):

$$\frac{?}{\{P\} \text{ \textbf{if}}(cond) \text{ \textit{PROG}}_1 \text{ \textbf{else}} \text{ \textit{PROG}}_2 \{S\}}$$

Hoare Logic: Rules of Inference

Rules (conditionals):

$$\frac{\{P \wedge cond\} PROG_1 \{S\}}{\{P\} \textbf{if}(cond) PROG_1 \textbf{else} PROG_2 \{S\}}$$

Hoare Logic: Rules of Inference

Rules (conditionals):

$$\frac{\{P \wedge cond\} PROG_1 \{S\} \qquad \{P \wedge \neg cond\} PROG_2 \{S\}}{\{P\} \mathbf{if}(cond) PROG_1 \mathbf{else} PROG_2 \{S\}}$$

More on correctness and loop invariants:

- 1 **C.A.R. Hoare**, "*An axiomatic basis for computer programming.*" Communications of the ACM 12.10 (1969): 576-580.
- 2 **Edsger Dijkstra**'s 1990 lecture "*Reasoning about programs*"
<https://www.youtube.com/watch?v=GX3URhx6i2E>