วิธีการตรวจสอบโปรแกรมโดยหลักการเชิงรูปนัย

นายฉัตรชัย เกิดสวัสดิ์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาการคณนา ภาควิชาคณิตศาสตร์
คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2544

ISBN 974-17-0279-5

A FORMAL APPROACH TO PROGRAM VERIFICATION

Mr.Chatchai Koetsawat

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Computational Science
Department of Mathematics
Faculty of Science
Chulalongkorn University
Academic Year 2001

ISBN 974-17-0279-5

Thesis Title	A FORMAL APPROACH TO PROGRAM VERIFICATION
By	Mr. Chatchai Koetsawat
Field of Study	Computational Science
Thesis Advisor	Assistant Professor Peraphon Sophatsathit, Ph.D.

Accepted by the Faculty of Science, Chulalongkorn University in Partial Fulfillment of the Requirements for the Master 's Degree

..... Deputy Dean for Administrative Affairs

(Associate Professor Pipat Karntiang, Ph.D.)Acting Dean, Faculty of Science

THESIS COMMITTEE

..... Chairman

(Professor Chidchanok Lursinsap, Ph.D.)

..... Thesis Advisor

(Assistant Professor Peraphon Sophatsathit, Ph.D.)

..... Member

(Assistant Professor Patanee Udomkavanich, Ph.D.)

ฉัตรชัย เกิดสวัสดิ์ : วิธีการตรวจสอบโปรแกรมโดยหลักการเชิงรูปนัย (A FORMAL APPROACH TO PROGRAM VERIFICATION) อ. ที่ปรึกษา : ผู้ช่วยศาสตราจารย์ ดร.พีระพนธ์ โสพัศสถิตย์, 19 หน้า. ISBN 974-17-0279-5.

ในกระบวนการผลิตซอฟแวร์ที่มีความน่าเชื่อถือนั้น การตรวจสอบความถูกต้องของโปรแกรมมีความสำคัญมาก ซึ่งวิธีการตรวจสอบที่เป็นรู้จักกันดีได้แก่ วิธีที่นำเสนอโดย Hoare และหลังจากนั้นได้มีนักคณิตศาสตร์และ โปรแกรมเมอร์จำนวนมากที่อาศัยพื้นฐานจากวิธีการของ Hoare ในการสร้างวิธีการตรวจสอบใหม่ๆ รวมถึงงานวิจัยชิ้นนี้ด้วย

ในงานวิจัยชิ้นนี้ได้นำเสนอวิธีการตรวจสอบความถูกต้องของโปรแกรม โดยใช้ Hoare triple และ Rule of Inference เป็นพื้นฐานในการสร้างวิธีการตรวจสอบความถูกต้องของโปรแกรม เนื่องจากมีการประยุกต์การตรวจสอบแบบ Black-Box และ White-Box อย่างมีระบบแน่นอน มาใช้ในกระบวนการนี้ด้วย

ภาควิชา **คณิตศาสตร์** ลายมือชื่อนิสิต.....

สาขาวิชา **วิทยาการคณนา** ลายมือชื่ออาจารย์ที่ปรึกษา.....

ปีการศึกษา 2544

4272242723 : MAJOR COMPUTATIONAL SCIENCE

KEY WORD: PROGRAM CORRECTNESS PROVE / HOARE TRIPLE / RULE OF INFERENCE / PROGRAM VERIFICATION

CHATCHAI KOETSAWAT : A FORMAL APPROACH TO PROGRAM VERIFICATION. THESIS ADVISOR : ASSIST. PROF. PERAPHON SOPHATSATHIT, Ph.D., 19 pp. ISBN 974-17-0279-5.

Program verification has played an important role in today's production of reliable software[3]. The most popular method of verification is given by Hoare. Other mathematicians and programmers also offered their method based on Hoare's principles. This thesis proposes a different approach to program verification using Hoare notation.

In this thesis, we introduced a new method based on Hoare triple and some inference rules as a tool for program correctness proof. The proposed approach adopted conventional black-box and white-box tests [6] to carried out a systematic and rigorous program verification.

Department Mathematics
Field of study Computational Science.
Academic year 2001

Student's signature.....
Advisor's signature.....

Acknowledgments

I am greatly indebted to Assistant Professor Dr.Peraphon Sophatsathit, my thesis advisor, for his untiring insight and helpful advice in preparing and writing this thesis.

Furthermore, I would like to thank all my instructors for their invaluable lecture and instructions.

I would like to express my sincere gratitude to my parents for their love, hearty encouragement and unselfish sacrifices during my study.

I would like to thank my uncle, Chaiwat, and his wife for their kindness and support, to Panat, my best friend, for his help, and to Jarunee for her love and care.

CONTENTS

CHAPTER 1

INTRODUCTION

A common task in program verification is to show that, for a given program code S, if a certain precondition $\{P\}$ is true before the execution of S then a certain postcondition $\{Q\}$ is true after execution, provided that S terminates. Hoare established the notation $\{P\}S\{Q\}$ for this logical proposition, called Hoare triple.

In this thesis we will present a formal method for program verification that uses Hoare propositions as the basic for our approach. Chapter 2 introduces definitions of important terms logically imply, rules of inference, predicates, program segments, and some basic mathematical theorems to be used in proving program correctness. Chapter 3 explains basic rules of inference to deal with each type of executable statements, namely, Assignment Rule, Concatenation Rule, Rule for Conditions, and While Rule. Then we developed an algorithm for proving program correctness based on all the above building blocks. Chapter 4 illustrates some examples on how the proposed approach works, and Chapter 5 concludes the research with a few final notes for future work.

CHAPTER 2

BASIC KNOWLEDGE

In this chapter, we introduce some basic mathematical theorems to be used in proving program correctness. The following definitions establish the fundamental of predicate calculus used in our theorems.

Definition 2.1 [5] For sets A and B, a **function** f from A to B is a unique assignment of an element of B for each element a of A. We write $f: A \to B$ to make explicit the role of the sets A and B. The set A is called the **domain** of f, and B the **codomain** of f. We write $b = f(a)$ to indicate that an element a of A is assigned by f to the element b of B. When $b = f(a)$, a is the input and b is the output for this input. The element b is also called the **image** of a. The set of outputs of f is called the **range** of f.

This definition is a "machine" or "formula" point of view. A function may be regarded as a kind of machine which takes the elements of the domain X, processes them, and for each element of the domain produces exactly one element of the range.

Definition 2.2 [5] We say that the statements $P_1, P_2, ..., P_n$ **logically imply** the statement P, provided that P is true whenever $P_1, P_2, ..., P_n$ are all true. In this case, we say that P is a valid conclusion from the hypotheses $P_1, P_2, ..., P_n$, denoted by $P_1, P_2, ..., P_n \Rightarrow P$.

Theorem 2.3 [5] Statements $P_1, P_2, ..., P_n$ logically imply P if and only if $P_1 \wedge P_2 \wedge ... \wedge P_n$ is a tautology.

Definition 2.4 [5] A **rule of inference** is a way of proceeding from several statements, $P_1, P_2, ..., P_k$, to another statement P. It is required that these statements P_i logically imply P. We write $\frac{P_1, ..., P_k}{P}$.

Definition 2.5 [5] Let $h_1, h_2, ..., h_n$ be statements (the hypotheses or premises), and let P be a statement (the conclusion), we say that P is a **consequence** of the h_i, or that P may be deduced from h_i, provided that there is a sequence of statements $A_1, A_2, ..., A_n$ with the following properties:

1. The final statement A_n is the conclusion of P.
2. Each statement A_i must either
   a. be one of the hypothesis h_i, or
   b. be a tautology, or
   c. follow from some previous statements by a valid rule of inference.

The sequence $A_1, A_2, ..., A_n$ is called a **deduction**.

Theorem 2.6 [5] Suppose it is possible to deduce P from the hypotheses $P_1, P_2, ..., P_n$, then $P_1, P_2, ..., P_n$ logically imply P.

Definition 2.7 [5] **Predicates** are expressions involving variables in the universe.

Predicates take the place of the variable statements involving p, q, r, s, ... of the propositional calculus. They, too, can take on the two values T and F, but these values depend on the values of the variables that are substituted in for the variables appearing in the predicate.

Definition 2.8 We call every statement or every group of statements that affects the state a **program segment**.

Definition 2.9 A program segment expresses a partial function in the sense that for each initial statement there is at most one final state.

We use these definitions as the fundamental building blocks to prove program correctness. We will give the definition of concatenation, composition function, define a notation, *A(x)*, that represents the image of *A* at *x*, and show how the theorem relates the correspondence between a computer program and these definitions.

Definition 2.10 [4] If there are two program segments A and B that are executed in sequence, one typically writes $A;B$, and the result is called the **concatenation** of A and B.

Definition 2.11 The final state of a code segment A corresponding to the initial state x is the image of A at x, denoted by $A(x)$.

Theorem 2.12 Concatenation is function composition.

Proof Let x be the initial state of $A;B$, that is, the state before A is executed.

By definition, the state after A is executed is $A(x)$, and this is the state before B is executed.

Since $B(y)$ is the final state of B, given an initial state y, and since $y = A(x)$, this means that the final state of $A;B$ is $B(A(x))$.

Consequently, $A;B = B \circ A$. ❑

Remark The final state of $A;B$ can also be written as $(A;B)(x)$.

Theorem 2.13 A computer program is the composition function.

Proof Let $A_1, A_2, ..., A_n$ be n segments in a program.

First, we concatenate A_1 and A_2, forming $A_1;A_2$.

Since $A_1;A_2$ is also a program segment, we concatenate $(A_1;A_2)$ and A_3, forming $(A_1;A_2);A_3$ and repeatedly concatenate until A_n. Thus we have $(((A_1;A_2);A_3);...);A_n$.

Since $(((A_1;A_2);A_3);...);A_n = A_n \circ A_{n-1} \circ ... \circ A_1$

This shows that a computer program is the composition function. ❑

Definition 2.14 [6] **Black-box test** is used to demonstrate that program code is operational, that input is properly accepted and output is correctly produced and that the integrity of external information is maintained.

Definition 2.15 [6] **White-box test** is predicated on close examination of procedural detail.

Black-box test considers only input and output of the program code, whereas white-box test, sometime called glass-box test, considers every statement in the program code.

CHAPTER 3

PROGRAM CORRECNESS PROOFS

The fundamental principles to correctness proof are preconditions and post-conditions. In Chapter 2, we established definition 2.8 for the states that constitute a program segment. To show how different program segments work correctly, one must associate preconditions and postconditions with each program segment. In fact, when the program is executed, the postconditions of each program segment imply the preconditions of the next segment. Consequently, one needs a rule of inference to deal with each type of executable statement in program correctness proof.

The tools required for this work are Assignment Rule, Concatenation Rule, Rule for Conditions, and While Rule. We will explain how these rules make up a formal method for program verification.

We will first present some basic program assertions needed for correctness proof.

Definition 3.1 [4] If x is a state and $g(x)$ is either true or false, then g is called an **assertion**. An assertion concerning the initial state of a program segment is called a **precondition**, and an assertion regarding the final state of a program segment is called a **postcondition**. We denoted $\{g\}$ as an assertion g.

Definition 3.2 [1] A program segment is said to be **partially correct** with respect to the precondition $\{P\}$ and the postcondition $\{Q\}$ if the final state of the computer program always satisfies $\{Q\}$, provided that the computer program starts in $\{P\}$ and that it terminates.

Definition 3.3 [1] A computer program is said to be **totally correct** if for each state satisfying $\{P\}$ the computer program terminates with a final state satisfying $\{Q\}$.

Definition 3.4 [4] If S is a program segment, $\{P\}$ is the precondition of S and $\{Q\}$ is the postcondition of S, we write $\{P\}S\{Q\}$. The triple $\{P\}S\{Q\}$ is called a **Hoare triple**.

We introduce the empty assertion {}, which can be read as "true for all possible states."

Definition 3.5 $\{P\}S\{Q\}$ is said to be partially correct is the final state of S satisfies $\{Q\}$, provided that the initial state satisfies $\{P\}$. If $\{P\}S\{Q\}$ is partially correct and S terminates, then $\{P\}S\{Q\}$ is said to be totally correct.

Consider now the correctness of a single statement, in many cases, the postcondition for a given precondition follows immediately from the definition of the statement in question. For instance, it is clear that $\{\ \}x := 3\{x = 3\}$ is correct, the statement $x := 3$ is exactly the statement that demands the postcondition $x = 3$ to be satisfied.

We will show the rules of inference which appeared in [1, 2, 4] and prove them. We use these rules to set up a formal method for program verification. First, we deal with precondition strengthening and postcondition weakening. We will discuss the rules that can strengthen or weaken preconditions and postconditions simultaneously. The following definitions explain what is meant by strengthening and weakening assertions.

Definition 3.6 [4] If R and S are two assertions, then R is said to be **stronger** than S if $R \Rightarrow S$. If R is stronger than S, then S is said to be **weaker** than R.

If a program segment is correct under precondition $\{P\}$, then it remains correct if $\{P\}$ is strengthened. For instance, if a code is correct for precondition $i > 0$, it remains correct for precondition $i > 1$, which is stronger. This leads to the following rules:

Theorem 3.7 [4] **(Precondition Strengthening)** Suppose that $\{P\}S\{Q\}$ is correct and $P_1 \Rightarrow P$ has been proved, one is allowed to conclude that $\{P_1\}S\{Q\}$ is correct. This leads to the following rule of inference:

$$\begin{array}{l} P_1 \Rightarrow P \\ \{P\}S\{Q\} \\ \text{-----} \\ \{P_1\}S\{Q\} \end{array}$$

Proof If $P_1 \Rightarrow P$ is true, then every state for which P_1 is true also satisfies P.

If each initial state that satisfies P leads to a final state that satisfies Q, then each state in which P_1 holds leads to a final state in which Q is true. This means that $\{P_1\}S\{Q\}$ holds. ❑

Example 3.8 Using the Hoare triple $\{i < 4\}i := i + 1\{i < 5\}$ to prove that

$$\{i < 3\}i := i + 1\{i < 5\}$$

Solution Clearly, $(i < 3) \Rightarrow (i < 4)$ is true.

Then we use the pattern given in above theorem

$$\begin{array}{l} (i < 3) \Rightarrow (i < 4) \\ \{i < 4\}i := i + 1\{i < 5\} \\ \text{----------} \\ \{i < 3\}i := i + 1\{i < 5\} \end{array}$$

Note: $i < 3$, $i < 4$, and $i := i + 1$ match P_1, P, and S, respectively. ❑

Theorem 3.9 [4] **(Postcondition Weakening)** The principle of postcondition weakening allows one to conclude that $\{P\}S\{Q_1\}$ once $\{P\}S\{Q\}$ and $Q \Rightarrow Q_1$ are established. Formally, this can be expressed as follows:

$$\frac{\{P\}S\{Q\} \qquad Q \Rightarrow Q_1}{\{P\}S\{Q_1\}}$$

Proof According to the definition, $\{P\}S\{Q\}$ means that the final state of S satisfies Q, provided that the initial state satisfies P.

If, in addition to this, $Q \Rightarrow Q_1$, then the final state must also satisfy Q_1, which translates into $\{P\}S\{Q_1\}$. ❑

Example 3.10 Using Hoare triple $\{\ \}\max := b\{\max = b\}$, prove that

$$\{\ \}\max := b\{\max \geq b\}$$

Solution Clearly, $\max = b \Rightarrow \max \geq b$ is true.
According to the principle of postcondition weakening, one finds

$$\frac{\{\ \}\max := b\{\max = b\} \qquad (\max = b) \Rightarrow (\max \geq b)}{\{\ \}\max := b\{\max \geq b\}}$$

❑

Next, we will discuss 4 rules (Assignment Rule, Concatenation Rule, Rule for Conditions with Else Clause, and While Rule) that will be used to prove program correctness.

We will start with Assignment Rule. Given the definition of an assignment statement, we will set up the assignment rule as follows:

Definition 3.11 [4] **Assignment statements** are statements of the form $V := E$, where V is a variable and E is an expression. Here ":=" is the assignment operator.

We call V the **left hand side** and E the **right hand side** of an assignment. The effect of the assignment operator is that the right hand side is evaluated using the initial values of the variables. The result is assigned to the left hand side to obtain the final state. If E_α is the value of the right hand side evaluated based on the initial state, then, by definition, the postcondition of the assignment statement $V := E$ becomes

$$\{\ \}V := E\{V = E_\alpha\} \qquad (*)$$

This assumes that E can be evaluated for all possible states.

Theorem 3.12 [4] **(Assignment Rule)** Let E be an expression and V be an unsubscripted variable. If C is a statement of the form $V := E$ with postcondition $\{Q\}$, then the precondition of C can be found by replacing all instances of V in Q by E. If Q_E^V is the expression thus obtained, one has the following:

$$\{Q_E^V\} V := E\{Q\}$$

Proof Suppose that the postcondition of $V := E$ is Q.

Because of (*), one can add the term $V = E_\alpha$ to the postcondition, which yields the new postcondition $\{Q \wedge (V = E_\alpha)\}$. Since V and E_α are equal, one can substitute for the other yielding

$$Q \wedge (V = E_\alpha) \equiv Q_{E_\alpha}^V \wedge (V = E_\alpha)$$

$Q_{E_\alpha}^V$ does not contain V because all instances of V have been substituted by E_α. The assignment does not change any variable except V. Consequently, all variables of $Q_{E_\alpha}^V$ refer to the initial state, so $Q_{E_\alpha}^V$ can be converted to precondition. Once this is done, the subscript α may be dropped. This completes the proof. ❑

Example 3.13 Consider the statement $j := i+1$. Suppose that the postcondition of this statement is $j > 0$, find the precondition.

Solution According to the assignment rule, we will replace j in the postcondition $\{j > 0\}$ by $i+1$, which yields P as $i+1 > 0$.

We have $\{i+1>0\}\ j := i+1\ \{j>0\}$.

So the precondition is $i+1 > 0$. ❑

Some arithmetic operations are undefined for some states. For example, if E is an expression, $\frac{1}{E}$ is undefined if E evaluates to zero. In such case, the condition that makes the evaluation of E possible must be added to the precondition.

Example 3.14 Find the precondition of the statement $x := \frac{1}{x}$, given that the postcondition is $x \geq 0$.

Solution By the assignment rule, we have $\frac{1}{x} \geq 0$.

And because of division by zero is not define, we have $x \neq 0$.

Together, these two conditions imply that $x > 0$, that is,

$$\{x > 0\} x := \frac{1}{x} \{x \geq 0\}$$ ❑

We will construct a Concatenation Rule based on definition 2.10 as follows:

Theorem 3.15 [2] **(Concatenation Rule)** Let C_1 and C_2 be two program segments and $C_1;C_2$ be their concatenation. If $\{P\}C_1\{R\}$ and $\{R\}C_2\{Q\}$ are correct, we conclude that $\{P\}C_1;C_2\{Q\}$. Hence,

$$\frac{\{P\}C_1\{R\} \quad \{R\}C_2\{Q\}}{\{P\}C_1;C_2\{Q\}}$$

Proof From definition 2.10, concatenation means that the program segments are executed in sequence such that the final state of C_1 becomes the initial state of C_2, and assertion R that holds for the final state of C_1 must be true for the initial state of C_2.

And because of $\{P\}C_1\{R\}$ and $\{R\}C_2\{Q\}$ are both correct, one is allowed to conclude that $\{P\}C_1;C_2\{Q\}$. This completes the proof of the concatenation rule. ❑

Example 3.16 Prove that the following code is correct.

$$\{\ \}c := a+b; c := \frac{c}{2}\left\{c = \frac{a+b}{2}\right\}$$

Solution We start with assignment rule, the postcondition of the second statement is given as $\left\{c = \frac{a+b}{2}\right\}$, so we can derive its precondition to $\left\{\frac{c}{2} = \frac{a+b}{2}\right\}$ which will be the postcondition of the first statement.

To find the precondition of the first statement, we use the assignment rule to obtain $\left\{\frac{a+b}{2} = \frac{a+b}{2}\right\}$.

This reduces to an empty assertion. So, we can conclude that

$$\frac{\{\ \}c := a+b\left\{\frac{c}{2} = \frac{a+b}{2}\right\} \quad \left\{\frac{c}{2} = \frac{a+b}{2}\right\}c := \frac{c}{2}\left\{c = \frac{a+b}{2}\right\}}{\{\ \}c := a+b; c := \frac{c}{2}\left\{c = \frac{a+b}{2}\right\}}$$

❑

The Concatenation Rule can be generalized as follows:

Theorem 3.17 [2] **(Modified Concatenation Rule)** Let C_1 and C_2 be two program segments and $C_1;C_2$ be their concatenation. If $\{P\}C_1\{R\}$ and $\{S\}C_2\{Q\}$ are correct, and if $R \Rightarrow S$, we conclude that $\{P\}C_1;C_2\{Q\}$. Hence,

$$\{P\}C_1\{R\}$$
$$\{S\}C_2\{Q\}$$
$$R \Rightarrow S$$
$$\text{-------}$$
$$\{P\}C_1;C_2\{Q\}$$

Proof Refer to theorem 3.7, $\{R\}C_2\{Q\}$ can be obtained from $\{S\}C_2\{Q\}$.
And from theorem 3.15, we can conclude that $\{P\}C_1;C_2\{Q\}$.
This completes the proof of modified concatenation rule. ❑

We can use Concatenation Rule and Modified Concatenation Rule for more than two program segments. This is shown in the example below.

Example 3.18 Given three program segments:

$s := 1;$
$s := s + r;$
$s := s + r \times r$

Find their precondition, if the postcondition is $\{s = 1 + r + r^2\}$.

Solution We start with the last statement, its postconditon is $\{s = 1 + r + r^2\}$. By assignment rule, we can find the precondition is $\{s + r^2 = 1 + r + r^2\}$, one can simplify this expression to $\{s = 1 + r\}$ and this will be the postcondition of the previous statement.

According to the assignment rule, we can find the precondition of second statement is $\{s + r = 1 + r\}$ or $\{s = 1\}$ and this will be the postcondition of the first statement.

And by assignment statement, the precondition of first statement is $\{1 = 1\}$ or $\{\ \}$ and this is the precondition of the compound statement. Heuce,

$$\{\ \}s := 1; s := s + r; s := s + r \times r\{s = 1 + r + r^2\}$$ ❑

Any assertion that is at the same time a precondition and a postcondition of a program segment is called an invariant. This will be further discussed when we analyze loops.

Now we will consider if – statement. If C_1 and C_2 are two program segments and if B is some condition, then the statement

$$if\ B\ then\ C_1\ else\ C_2$$

is executed as follows: if B is true, C_1 is executed and if B is false, C_2 is executed. We will derive a rule to prove that an if – statement with precondition $\{P\}$ and postcondition $\{Q\}$ is correct.

Theorem 3.19 [2] **(Rule for Conditions)** If C_1 and C_2 are two program segment and if B is a logical expression, then one has

$$\{P \wedge B\}C_1\{Q\}$$
$$\{P \wedge \neg B\}C_2\{Q\}$$
$$\text{---------------}$$
$$\{P\}\ if\ B\ then\ C_1\ else\ C_2\ \{Q\}$$

Proof If the initial state satisfies B in addition to $\{P\}$, then C_1 is executed and the proof amounts to a demonstration that $\{P \wedge B\}C_1\{Q\}$ is correct.

Similarly, if the initial state satisfies $\neg B$, then C_2 is executed and this implies that $\{P \wedge \neg B\}C_2\{Q\}$ is correct.

We conclude that $\{P\}\ if\ B\ then\ C_1\ else\ C_2\ \{Q\}$ is correct. ❑

Example 3.20 Prove that $\{\ \}\ if\ a > b\ then\ m := a\ else\ m := b\ \{(m \geq a) \wedge (m \geq b)\}$ is correct.

Solution According to the Rule for Conditions, we must prove that

$$\{a > b\}m := a\{(m \geq a) \wedge (m \geq b)$$

and

$$\{\neg(a > b)\}m := b\{(m \geq a) \wedge (m \geq b)\}$$

We will prove the first statement. By the Assignment Rule, we will replace m in the postcondition by a, so we have the precondition $\{(a \geq a) \wedge (a \geq b)\}$ or $\{a \geq b\}$.

Since $a > b$ implies $a \geq b$, by precondition strengthening

$$\{a > b\}m := a\{(m \geq a) \wedge (m \geq b)$$

is correct.

Similarly, we can prove that $\{\neg(a > b)\}m := b\{(m \geq a) \wedge (m \geq b)\}$ is correct.

Hence,

$$\{a > b\}m := a\{(m \geq a) \wedge (m \geq b)\}$$
$$\{\neg(a > b)\}m := b\{(m \geq a) \wedge (m \geq b)\}$$
$$\text{---------------------------------}$$
$$\{\ \}\ \textit{if}\ a > b\ \textit{then}\ m := a\ \textit{else}\ m := b\ \{(m \geq a) \wedge (m \geq b)\}$$

❑

In [1], While Rule is stated as follows:

$$P \Rightarrow I$$
$$\{I \wedge B\}S\{I\}$$
$$(I \wedge \neg B) \Rightarrow Q \qquad (**)$$
$$\text{----------}$$
$$\{P\}\ \textit{while}\ B\ \textit{do}\ S\ \{Q\}$$

Here, I denotes the invariant assertion. The condition $P \Rightarrow I$ states that the invariant I is true when we enter the loop and the condition $\{I \wedge B\}S\{I\}$ conveys that if I is true before executing the loop body S, and if the execution of S terminates, I will be true afterwards. Then the condition $(I \wedge \neg B) \Rightarrow Q$ ensures that if control ever exits from the loop, then Q will be true.

We did not use this rule in our method because this rule adheres to conventional restrictive while construct. Our approach to the While Rule is to examine a while loop as a concatenation of sequential statements executing a pre-defined number of repetitions. As a consequence, loop correctness proof can be carried out in similar manner as the previous proofs.

Theorem 3.21 For any positive integer n, an n times while loop is equivalent to a sequential program that is set up by concatenating statements sequentially n times.

Proof Consider Hoare triple:

$$\{P\}\ \textit{while}\ B\ \textit{do}\ S\ \{Q\}$$

From (**), we have the condition $P \Rightarrow I$, $\{I \wedge B\}S\{I\}$, and $(I \wedge \neg B) \Rightarrow Q$.

Let S_i be a statement inside the loop executing i times.

The first time the loop is executed, we have $\{I \wedge B\}S_1\{I\}$.

Since B is true, we have $\{I \wedge B\}S_i\{I\}$ until B become false, at that point, we have $(I \wedge \neg B) \Rightarrow Q$

From Concatenation Rule we have

$$
\begin{array}{l}
P \Rightarrow I \\
\{I \wedge B\} S_1 \{I\} \\
\{I \wedge B\} S_2 \{I\} \\
\quad \vdots \\
\{I \wedge B\} S_n \{I\} \\
(I \wedge \neg B) \Rightarrow Q \\
\text{----------} \\
\{P\} S_1; S_2; \ldots; S_n \{Q\}
\end{array}
$$

This completes the proof. ❑

From this theorem it is straightforward to prove a while loop as shown with the help of the Concatenation Rule. We can construct a program correctness proof algorithm based on Hoare triple, $\{P\}S\{Q\}$, meaning that if the precondition $\{P\}$ is true on the initial state and the program segment S terminates then the postcondition $\{Q\}$ will be true at the final state.

Let p stand for the proposition "the precondition $\{P\}$ is true at the initial state", q for the proposition "the program segment S terminates", and r for the proposition "the postcondition $\{Q\}$ is true at the final state". The Hoare triple takes the form $(p \wedge q) \rightarrow r$ which is equivalent to $\neg r \rightarrow (\neg p \vee \neg q)$. This means that if the postcondition $\{Q\}$ is false at the final state then the precondition $\{P\}$ is false at the initial state or the program segment S does not terminate. This principle is implemented as the algorithm shown below.

Algorithm for program correctness proof

Step 1 Check each program segment if it terminates (to ensure totally correct). If the program code does not have a loop, it will terminate, otherwise check the termination conditions of the loop (see algorithm supplement).

Step 2 Find the preconditions and the postconditions of the program code. (to ensure partially correct).

Step 3 (Black-box test) Compare the preconditions with the input specification and the postconditions with the output specification. If it is true the program code is correct and the program will end, otherwise go to step 6.

Step 4 Divide the program code into program segments according to basic program constructs, namely, sequence, selection, and repetition. If the program code contains library or procedure calls, we will consider them as separate program code.

Step 5 Use the rule of inference to prove every program segment as follows:

Sequence : Theorem 3.12, 3.15, 3.17
Selection : Theorem 3.19
Repetition : Theorem 3.21

Step 6 Locate the incorrect statements.

Algorithm Supplement Additional precautionary steps for loop construct are furnished as follows:

1. Take into account the restrictions on global variables for the preconditions and the postconditions of the loop. The side-effect of global variables may disturb the intrigrity of loop invariant.
2. Loop termination check rests upon the exit condition of the loop (and each decomposed sequential equivalent code segment) $i \blacklozenge n$, where i and n denote loop index and loop repetitious control, respectively, and $\blacklozenge$ is a compare operator. The variable i produces a sequence for checking loop termination. If this sequence is finite, the loop will terminate. If it is infinite, the loop will never terminate. The procedures proceed as follows:

   Case1: If the values of both i and n remain unchange inside the loop, the loop is never entered if $i \blacklozenge n$ is false, or it never terminates if $i \blacklozenge n$ is true.

   Case 2: If the value of i changes but n remains unchange inside the loop and the value of i converge to n, the loop will terminate.

   Case 3: If the value of i remains unchange but n changes inside the loop and the value of n converge to i, the loop will terminate.

   Case 4: If both i and n change inside the loop, the result may be undefined, depending on the following behavior:

   4.1 if the rate of change of i is langer than that of n by a factor of m, $m \geq 1$, and the value of i converge to n or the value of n converge to i, the loop will terminate; or

   4.2 if the rate of change of i is not the case as in 4.1, the loop may not terminate and the behavior is undefined.
3. We do not consider parallel computing.

We will demonstrate how this algorithm works in the next Chapter.

CHAPTER 4

APPLICATION OF THE ALGORITHM

In this Chapter, we will apply our proposed algorithm to demonstrate how it works with some sample code. We selected a good representative yet straightforward gcd (greatest common divisor) problem. Gcd problem is a basic program that most beginners must go through.

The greatest common divisor gcd of two nonnegative integers x and y is the largest integer that can divide both x and y. We write gcd(x,y) for gcd of x and y. For example, gcd(6,3) = 3, gcd(12,8) = 4. If x and y are both zero, the gcd is undefined. The following examples demonstrate program correctness proof using the proposed algorithm.

Example 4.1 Prove that the sample code is correct.

```
int gcd(x, y)
int x, y, z;
     {
        int r;
        while (y != 0) {
                    r = x % y;
                    x = y;
                    y = r;
                    }
        z = x;
        return z;
       }
```

The input specification x and y are both integers, and the output specification is $z = gcd(x,y)$.

Proof Applying our algorithm to this problem as follows:

Checke termination.

Due to while loop in this code, this code may or may not terminate.

Consider the exit condition of while loop, y *!= 0*, y changes inside the loop and may not be equal to zero. Thus, we may have an infinite sequence. This means that the program code may not terminate, so the exit condition is the error of this program code. ❑

Example 4.2 This is a favorite problem statement for finding the gcd of x and y.

[1] "Until x is zero, repeat the follow process: if y is greater than or equal to x, replace it by their difference otherwise interchange the two numbers and continue. When x becomes zero, the answer is y."

Here is the solution created for this problem statement:

```
int gcd(x, y)
int r, x, y, z;
     {
      while (x!= 0)
           {
            if y ≥ x then y:= y-x else
                                          {
                                           r := x;
                                           x := y;
                                           y := r;
                                          }
           }
      z = y;
      return z;
     }
```

Prove this code is correct.

Proof Applying our algorithm to this problem as follows:

Step 1 Check termination.

Due to while loop in this code, this code may or may not terminate.

Consider the exit condition of while loop, x *!= 0*, x changes inside the loop and creates a sequence that converges to zero. Thus, this sequence is finite and the loop will terminate.

Step 2 Find the precondition and the postcondition of the program code.

We set the postcondition according to the program output specification. Thus, the postcondition becomes $\{z = gcd(x,y)\}$ or$\{z = max[u : u/x \wedge u/y]\}$ or $\{z = max[u : x = u \times m \wedge y = u \times n]\}$. The precondition can be found using the rule of inference as follows:

The code is a concatenation of an assignment statement and a while loop statement that encompasses an if – statement.

First, find the precondition of the assignment statement $z = y$. According to the Assignment Rule, we have $\{z = max[u : x = u \times m \wedge y = u \times n]\}$ the precondition whicle becomes the postcondition of the previous statement (the while statement).

Next, find the precondition of the while statement.

We apply theorem 3.21 to this while loop and transform the while loop to a series of concatenation of if – statements as follows:

if $y \geq x$ *then* $y := y - x$ *else* $r := x$; $x := y$; $y := r$;
if $y \geq x$ *then* $y := y - x$ *else* $r := x$; $x := y$; $y := r$;
⋮
if $y \geq x$ *then* $y := y - x$ *else* $r := x$; $x := y$; $y := r$;

According to the Rule for Conditions, we have $\{x \geq 0 \wedge y \geq 0 \wedge (x \neq 0 \vee y \neq 0)\}$ the precondition which is the precondition of the program.

Step 3 Compare the precondition and the postcondition with input specification and output specification of the program. The precondition satisfies the input specification, where x and y are both integer not being zero simultaneously. The postcondition satisfies the output specification $\{z = max[u : x = u \times m \wedge y = u \times n]\}$ while means z is the gcd of x and y.

We conclude that this code is correct, provided that the input is correct. ❑

CHAPTER 5

CONCLUSION AND FUTURE WORK

The proposed approach presented in this thesis introduces the following ideas for program correctness proof:

First we use black-box test as a preliminary check that consumes very little time to prove program correctness.

Second we consider loop statement as concatenation of statements that simplifies the proof.

Third we check termination by simply checking if program contains loops. If it does not have loop, the program will terminate. Otherwise, we consider only the exit condition. This is considerably less time consuming than conventional approaches.

And last, we use white-box test for any program constructs that could result in normal or error statement to warrant program termination.

The shortcomings of the proposed approach are that users must be well-prepared and equipped with mathematics, hence unpoppular for average users and programmers. In addition, preconditions and postconditions sometimes are hard to derive which is an obstacle for program correctness proof.

We envision, as pointed out by [1], that correctness proof depends primilarily on formal and rigorous program specifications and fundamental of theoretical programming as follows:

1. We can never be sure that the specifications are correct,
2. No verification system can verify every correct program, and
3. We can never be certain that a verification system is correct.

The proposed approach will be more convenient to use with support from efficient precondition and postcondition instrumentation.

REFERENCES

[1] Manna, Z. and Waldinger, R. *The logic of computer programming.* IEEE Transactions on software engineering Vol. SE-4, No. 3, May 1978 : 199-229.

[2] Manna, Z. Mathematical Theory of Computation. New York : McGraw-Hill Inc., 1974.

[3] Misra, J. *Some aspects of the verification of loop computations.* IEEE Transactions on software engineering Vol. SE-4, No. 6, November 1978 : 478-486.

[4] Grassmann, W.K. and Tremblay, J.P. Logic and Discrete Mathematics : A computer science perspective. New Jersey : Prentice- Hall, Inc., 1996.

[5] Hausner, M. Discrete Mathematics. Florida : Saunders College Publishing, 1992.

[6] Pressman, R.S. Software Engineer : Practioner's approach. New York : McGraw-Hill Inc., 1992.

BIOGRAPHY

Name	Chatchai Koetsawat
Date of birth	September 9, 1977
Place of birth	Samutsongkhram
Education	Bachelor Degree of Science (Mathematics)
	Chiangmai University , 1999.