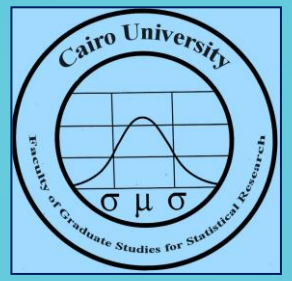


FGSSR Fall 2023

Lecture One

CONSTRUCTION OF CORRECT PROGRAM

WHY CORRECT PROGRAM?

- Disaster is an understatement for any organization that has incurred losses due to an overtly minuscule but catastrophic software glitch.
- While technology and innovative applications have empowered brands, enterprises have recorded numerous disabling instances.
- Software disasters have the potential to cause widespread disruptions and highlight the importance of rigorous testing and quality control measures in preventing catastrophic software failures.
- Any application's unexpected crashes and data inconsistencies were a direct result of the failure designing Correct software, revealing significant gaps in the quality assurance process.
- https://raygun.com/blog/costly-software-errors-history/
- https://www.cigniti.com/blog/37-software-failures-inadequate-software-testing/

APPLICATIONS: WHICH?

- Communication Protocols
- Processors (CPUs)
- Kernel of secured distributed O/S
- Compilers
- Safety critical: medical systems, nuclear control
- Railway, aerospace controls
- Telephone and internet switching systems
- Any critical systems (that cause loses in human lives, properties, Money,…

COURSE OBJECTIVE

- *The course tackle the verification problem by constructing a correct program step at a time, applying theory of Programming*
- *we are concerned in:*
  1. **Proving each step as we develop it.**
  2. **Modification of programs (with proof that the modification is correct.)**

WHO WANTS THEORY OF PROGRAMMING,… WHY?

- One answer is that a mathematical theory gives a much greater degree of precision by providing a method of calculation.
- The software industry has an overwhelming experience of buggy programs to support that statement.
- Another answer is that a theory provides a kind of understanding.
- Programming changes from an art to a science when we learn to understand programs in the same way we understand mathematical theorems.

- With a scientific outlook, we change our view of the world.. It is a valuable part of education for anyone.
- Professional Software engineers, to be worthy of the name, must know and apply a theory of programming.
- The subject of this course sometimes goes by the names "programming methodology", "science of programming", "logic of programming", "theory of programming", "formal methods of program development", "programming from specifications", or "verification".
- It concerns those aspects of programming that are amenable to mathematical proof.
- A good theory helps us to write precise specifications, and to design programs whose executions provably satisfy the specifications.

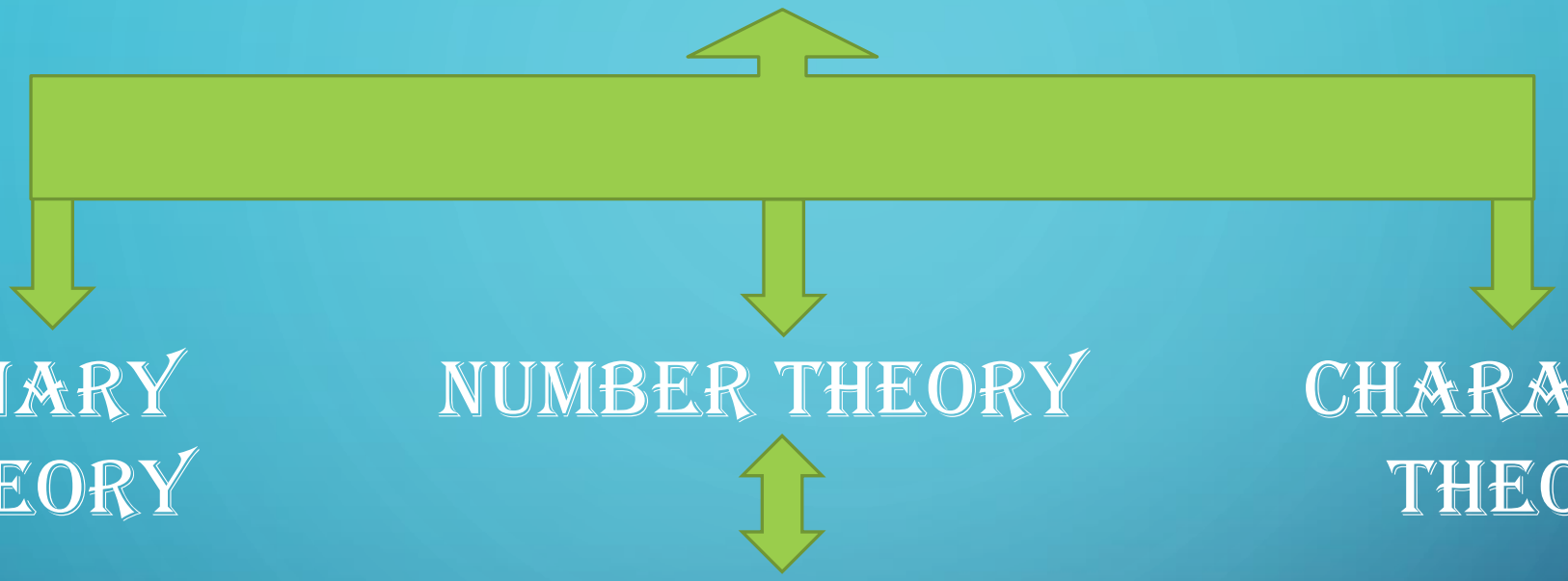
Our Theory

- In this theory, a specification is just a binary expression.
- Refinement is just implication.
- it is, also, more comprehensive, because it is applicable to:
  - both terminating and nonterminating computation,
  - both sequential and concurrent computation,
  - both stand-alone and interactive computation.

The Relevant Mathematics

- Formal Methods of Software Engineering means the use of mathematics as an aid to writing programs.
- Let us first introduce the relevant mathematics (only what is needed).
- Most of you may have already learned most of this math in other courses.
- Just to be sure we're all using the same notations.
- If you look at the handout, you'll see all the notations used in the formal verification methodology.

REQUIRED MATH



BINARY THEORY

NUMBER THEORY

CHARACTER THEORY

Domain

Function effects over the domain of the local variables

COURSE CONTENTS

- Basic theories: binary theory, number theory, character theory
- Basic data structures: bunch theory, set theory, string theory, list theory
- Function theory: functions, function notations, scope and substitution, quantifiers
- Program theory: specifications, specification notations, specification laws refinement, programs, program development, refinement laws
- Time: real time, recursive time, termination, soundness and completeness

Binary Theory

- Binary Theory, is also called Boolean algebra, basic logic, or propositional calculus.
- Some binary expressions are called theorems, and others are antitheorems.
- Binary expressions represent anything that comes in two kinds.
- The theorems represent one kind, and the antitheorems represent the other.
- One may think that statements about the world come in two kinds, (true statements and false statements),
- One, can use binary expressions to represent statements, (theorems and antitheorems)

Binary Theorems

- Here are the two simplest binary expressions:
  - $\top, \bot$ **(True/False, power/ground, innocent/guilty, top/bottom)**

Logical Notations

- One operand: ¬ Not or negation

Logical Notations

Two operands:

- $x \wedge y$ (conjunction with two conjuncts)
- $x \vee y$ (disjunction, and its operands are called disjuncts.)
- $x \Rightarrow y$ (implication, the antecedent $\Rightarrow$ the consequent)
- $x \Leftarrow y$ pronounced x is implied by y.

  the consequent $\Leftarrow$ the antecedent.)

Logical Notations

Two operands:

- $x \wedge y$ minimum
- $x \vee y$ maximum
- $x \Rightarrow y$ Implication is the ordering operator. It means:
- "lower than or equal to". OR “x is stronger than or equal to y.”
- $x \Leftarrow y$ a reverse implication is an ordering operator. It means “higher than or equal to". OR “x is weaker than or equal to y.”

Logical Notations

- $x = y$ (It's an expression whose result is true or false.
- $x \neq y$ (x is unequal to y.)

Logical Notations

- Three operands:
- [It's called conditional composition, or if-then-else-fi, and the operands are called the if-part, the then-part, and the else part]
- If x then a else b fi
- A precedence table for order of evaluation is in the handout.
- And if you want some other order of valuation, use parentheses, (round brackets).
- On that same table it define operators' associativity.

Logical Notations

- Conjunction and disjunctions are associative, for example:

$a \wedge b \wedge c \equiv a \wedge b \wedge c$; $a \vee b \vee c \equiv a \vee b \vee c$

$a \vee b \vee c \vee d \vee e \vee f$

- Same for Conjunction.

Logical Notations

- Implication, equal and unequal connectors are both associative and continuing operators.

$$A \Rightarrow b \Rightarrow c \equiv (a \Rightarrow b) \wedge (b \Rightarrow c)$$

$$a=b=c \equiv a=b \wedge b=c \; ; \quad a \neq b \neq c \equiv a \neq b \wedge b \neq c$$

- As you notice, no need of parentheses to write a conjunction of equations,
- Yet, conjunction of implications needed parantheses. Why?

EVALUATION

$\neg$	T	$\bot$
	$\bot$	T

	TT	T$\bot$	$\bot$T	$\bot\bot$
$\wedge$	T	$\bot$	$\bot$	$\bot$
$\vee$	T	T	T	$\bot$
$\Rightarrow$	T	$\bot$	T	T
$\Leftarrow$	T	T	$\bot$	T
$=$	T	$\bot$	$\bot$	T
$\neq$	$\bot$	T	T	$\bot$

if x then a else b fi	TTT	TT$\bot$	T$\bot$T	T$\bot\bot$	$\bot$TT	$\bot$T$\bot$	$\bot\bot$T	$\bot\bot\bot$
	T	T	$\bot$	$\bot$	T	$\bot$	T	$\bot$

$$x \Rightarrow a \wedge \neg x \Rightarrow b \equiv x \wedge a \vee \neg x \wedge b$$

Substitution (Instantiation)

- Given a conjunction $x \wedge y$, it stand for all expressions obtained by replacing the variables x and y with arbitrary binary expressions.
- Example, we might:
  - replace x by $(\bot \Rightarrow \neg(\bot \vee \top))$ and replace y by $(\bot \vee \top)$

Get the conjunction $(\bot \Rightarrow \neg(\bot \vee \top)) \wedge (\bot \vee \top)$.

- Replacing a variable with an expression is called substitution or instantiation.

Applying Binary Rules To An Application:

the grass is green, Theorem

the sky is green, Antitheorem

there's life elsewhere in the universe, Not a Theorem nor Antitheorem

intelligent messages are coming from space Not a Theorem nor Antitheorem

In number theory: 1+1=2 Theorem 0/0 =5 Not a Theorem nor Antitheorem

- The application, also, must classify (which is theorem, and which is antitheorem) those new expressions.
- When classifying binary expressions as theorems and antitheorems:
  - Be consistent (don't classify some binary expression as both a theorem and an antitheorem.)
  - It is a choice to be complete, but you MUST be consistent.

Classification of Theorem

- the binary expression $1+1 = 2$, is classified as a theorem.
- The binary expression $1 + 1>2$, is classified as antitheorem.
- But it is perfectly legitimate to leave a binary expression unclassified.
- For example, $1/0 = 5$ will be neither a theorem nor an antitheorem.
- An unclassified binary expression may correspond to a statement whose truth or falsity we do not know or do not care about.

Consistency Theorem

- A theory is called consistent if NO binary expression (exp) is both a theorem and an antitheorem.

$(exp=T \wedge \neg exp=\perp \vee exp=\perp \wedge \neg exp= T)$

$\equiv (exp=T \wedge exp\neq\perp \vee exp=\perp \wedge exp\neq T)$

Consistency is an ESSENTIAL PROPERTY for any binary expression.

Ex:{ $exp \wedge \neg exp$ is always antitheorem, while $exp \vee \neg exp$ is always a theorem} Consistent

- A theory is inconsistent if some binary expression is both a theorem and an antitheorem.

- A theory is inconsistent if some binary expression is both a theorem and an antitheorem.

Example:

$p = x \Rightarrow (x \wedge \neg x)$??

assume $x = T$ (Theorem) then $p = T \Rightarrow \bot = \bot$ (Antitheorem)

assume $x = \bot$, (Antitheorem) then $p = \bot \Rightarrow \bot$ (theorem)

EXAMPLE

- Two or more logical expression are logically ***consistent*** if it is possible for them to all be true at the same time. If there is no way for them all to be true at once, they are ***inconsistent***. Inconsistent logical expressions are said to ***contradict*** one another.
- These three expressions $a \Rightarrow b$, $a \vee b$, $\neg a$ are consistence
- While those two $(A \wedge B)$, $(\neg A \vee \neg B)$ are inconsistence

A	B	(A ⇒B)	(A ∨ B)	¬A
0	0	1	0	1
0	1	1	1	1
1	0	0	1	0
1	1	1	1	0

Consistence

A	B	(A ∧ B)	(¬A ∨ ¬B)
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Inconsistence

Complete Theorem

- A theory is called complete if every **fully instantiated** binary expression is either a theorem or an antitheorem, (T $\vee\perp$)
- A + b = c instantiated to 1+2 = 3 $\equiv$ T

  instantiated to 1+2 = 5 $\equiv \perp$
- A theory is incomplete if some fully instantiated binary expression has no classification (underdetermined, unprovable to be either a Theorem or an Antitheorem).
- x= 0/0; 5=1/0 (both 0/0 and 1/0 are undefined)

Example: Given the four statements.

(i) This statement is true.

(ii) This statement is false.

(iii) This statement is either true or false.

(iv) This statement is neither true nor false.

Which of these statements are

(a) true? (b) false? (c) neither true nor false (underdetermined, incomplete)? (d) both true and false (overdetermined, inconsistent)?

Answer:

(i) This statement is true. True

(ii) This statement is false. False , Inconsistence

(iii) This statement is either true or false. Inconsistence

(iv) This statement is neither true nor false. Incomplete

Theorem Proving: Proof Rules

- Finding out if a binary expression is a theorem or antitheorem is called PROVING.

Five rules for proving

1. Axiom Rule: If a binary expression is an axiom, then it is a theorem. If a binary expression is an Antiaxiom, then it is an antitheorem.
2. Evaluation Rule: If all the binary subexpressions of a binary expression are classified, then it is classified according to the truth tables.

Theorem Proving: Proof Rules

3. Completion Rule: If a binary expression contains unclassified binary subexpressions, and all ways of classifying them place it in the same class, then it (expression) is in that class.
4. Consistency Rule: If a classified binary expression contains binary subexpressions, and only one way of classifying them is consistent, then they are classified that way.
5. Instance Rule: If a binary expression is classified, then all its instances have that same classification.

First Proof Rule: Axiom

- An axiom is a binary expression that is stated to be a theorem.
- An antiaxiom is similarly a binary expression stated to be an antitheorem.

- the grass is green, Axiom
- the sky is green, an Antiaxiom
- there's life elsewhere in the universe, unclassified
- intelligent messages are coming from space, unclassified

intelligent messages are coming from space $\Rightarrow$ there is life elsewhere Axiom

First Proof Rule: Axiom

- Axiom: $\top$ Theorem
- Antiaxiom: $\bot$ Antitheorem
- Axiom: grass is green Theorem
- Antiaxiom: sky is green Antitheorem
- Axiom: (intelligent messages are coming from space)

  $\Rightarrow$ (there is a life elsewhere in the universe) Theorem

Second Proof Rule: The Evaluation

If all the subexpressions are classified, then use the truth tables.

- If you know what the operands are, then the truth tables tell you what the whole expression is.

Third Proof Rule: The Completion

- If an expression contains unclassified subexpressions, and all ways of classifying them place it (expression) in the same class, then it is in that class.
- If you don't know what the operands are, you might still be able to tell what the whole expression is.
- Example: Theorem: there is life elsewhere in the universe $\vee$ T.

if there's life elsewhere, then that's $T \vee T \equiv T$.

If there isn't life elsewhere, then that's $\perp \vee T \equiv T$.

It's a theorem by completion rule

Third Proof Rule: The Completion

Examples:

Theorem: $(a) \vee \neg (a)$ because $\mathbf{T} \vee \bot = \bot \vee \mathbf{T} = \mathbf{T}$

- Antitheorem: $p = (a) \wedge \neg (a)$

Assume a= T then $p = \mathbf{T} \wedge \bot = \bot$

Assume a= $\bot$ then $p = \bot \wedge \mathbf{T} = \bot$

So it's false (Antitheorem).

Fourth Proof Rule: The Consistency

- If a classified binary expression contains binary subexpressions, and only one way of classifying them is consistent, then they are classified that way.
- This time, we know what the whole expression is, and we're wondering what the parts might be.

Example 1: x , $x \Rightarrow y$ are theorems, what is y?

Assume y is an antitheorem,

$x \Rightarrow \bot \equiv \neg T \vee \bot \equiv \bot \vee \bot \equiv \bot$

i.e., $x \Rightarrow y$ is antitheorem contradicting the premises

therefore, y is not antitheorem, i.e., y is a theorem

Example 2:

Theorem: ¬x , What's x?

- Assume x = T theorem
- then by the evaluation rule, ¬x = ⊥ is an antitheorem, contradicting that (¬x) is a theorem. So *x* has to be an antitheorem.
- This example means no need to talk about Antiaxioms or antitheorems.
- If we want to say something is an antitheorem, we just say its negation is a theorem. (We also don't need to have both true and false)

Fifth Proof Rule: The Instance

If a binary expression is classified as a theorem or an antitheorem, then all its instances have that same classification.

axiom: $x = x$. Theorem: $x = x$

Theorem: $(T= \bot \vee \bot) \equiv (T= \bot \vee \bot)$

Classification of Logical Systems

- Classical logic: all five rules, (which we are using in this course)
- Constructive logic: don't allow the completion. In a logic course, it's interesting to see what you can do if we DO NOT allow completion rule.
- Evaluation logic: don't allow the consistency rule and the completion rule.

EXPRESSION and PROOF FORMATS

- $x \wedge y \vee z$ leave more space around operators with less precedence

$x \wedge y \vee z$ is misleading, and causes errors.

- When an expression is too long to fit on one line, it should be broken at a sensible place, which usually means at the main connective.

(First part)
$\wedge$ (Second part)

- Put left brackets on the left, and right brackets on the right.
- Here's a continuing equation, nicely formatted.

Expression 0
$=$ *Expression 1*
$=$ *Expression 2*
$=$ *Expression 3*

means

Expression 0 $=$ *Expression 1*
$\wedge$ *Expression 1* $=$ *Expression 2*
$\wedge$ *Expression 2* $=$ *Expression 3*

Expression And Proof Format

- Do $x \wedge y \vee z$ Do Not $x \wedge y \vee z$

  (First part)

  $\wedge$ (Second part)

	Expression 0	*Hint0 (tells why expression 0 equal expression 1*
=	*Expression 1*	*Hint1*
=	*Expression 2*	*Hint2*
=	*Expression 3*	

Expression and Proof Format

- Example: Prove: $a \wedge b \Rightarrow c \equiv a \Rightarrow (b \Rightarrow c)$

§	$a \wedge b \Rightarrow c$	**Material implication**
≡	$\neg(a \wedge b) \vee c$	**Duality**
≡	$\neg a \vee \neg b \vee c$	**Material implication**
≡	$a \Rightarrow \neg b \vee c$	**Material implication**
≡	$a \Rightarrow (b \Rightarrow c)$	

Material implication: $x \Rightarrow y = \neg x \vee y$

$a \wedge b \Rightarrow c = \neg (a \wedge b) \vee c$

Expression and Proof Format

- Simplify to true : $a \wedge b \Rightarrow c \equiv a \Rightarrow (b \Rightarrow c)$

§ $a \wedge b \Rightarrow c \equiv a \Rightarrow (b \Rightarrow c)$ **Material implication 3 times**

$= (\neg (a \wedge b) \vee c \equiv \neg a \vee (\neg b \vee c))$ **Duality**

$= (\neg a \vee \neg b \vee c \equiv \neg a \vee \neg b \vee c)$ **Reflixivity of ≡**

$= T$

Read The Handout Carefully, To Realize The Precedence and The Rules.
You Should Know The Logic and Numerals Laws.
Memorize As Much, or at Least Know Where To Find Them

SOLVING THE ASSIGNMENTS ARE ESSENTIAL FOR TRAINING

UNDERSTANDING AND PASSING THE COURSE