SD3049
Formal Methods in Software Engineering

FTMS Global

Established since 1986

Name : ______________________

Intake : ______________________

Contact Details : ______________________

First Edition 2011

ISBN

Publish by

FTMS Consultants (M) Sdn Bhd
Kuala Lumpur, Malaysia

www.ftmsglobal.com

Printed in Kuala Lumpur by
FTMS Consultants (M) Sdn Bhd

We are grateful to the academic members and module experts for permission to produce the syllabus, teaching guide and study materials of which contribute to FTMS.

Contents

Page

Overview

Overview

Normally UEL subject taught in FTMS KL do not have any guide books, thus the reason in 2011, Mr. Trevor wishes to have guide books for UEL students on UEL subjects.

As such this exercise will be carry out term by term and the lecturers teaching the given subjects will need to update their respective guide book for that given term.

And if all goes well by the end of the year all the books in FTMS will have been updated including UEL.

Thus I, Sola Lee was allocated this subject Network Programming as I have been teaching this subject for two (2) terms now.

SD3049 is a 3rd year subject and student have a prerequisite of SD2054 (Software Development) which in turn require SD1042 (Introduction to Software Development).

From my experience in teaching this subject, the student also needs to have a prerequisite of IM2042 (Information System Modeling and Design) which in turn require IM1045 (Information Systems). This is because Formal Methods main object is to scrutinize specification and design made before coding.

The topics will initially revise basic mathematical concept like proposition, predicates, sets, series or sequences and mathematical proofs.

Then the guide will use the above chapters and show how specification can be scrutinized using the above (especially during testing) to uncover potential omissions.

Then finally the guide book will end with Z notation which is a formal language.

Notice that for the basic mathematics concepts there will be a exercise classes to give the lecturers and the student time to try out and show their competency.
->

Chapter 01

Introduction to Formal Methods

1. SDLC
2. Formal Method
3. Advantage
4. Disadvantage
5. Critical Software
6. Integrity Level
7. Stages in Formal Methods

Revise Software Development

When creating a software there are few engineering stages that is normally be followed to ensure that they software is built within the time and budget. These stages collectively are called the software development life cycle (SDLC).

The SDLC can be divided into seven (7) stages;

1. Initial Study

   This is the first time the system development team meets the clients to collective information regarding the problem. Normally this stage delivers the proposal and quotation to the clients.

2. Analysis

   After the client has agreed to the proposal and price, the team will go in and study the current system with the intention to discover the source of the problem. The System analyst will use diagrams and data collection techniques (observation, inspections, interview, etc) to aid them. Normally this stage delivers a report stating the source of the problem and more then one alternative solutions.

3. Design

   After the client agrees with the analysis findings, the client will choose one (1) solution. From this one solution the system designer will create the specification. Take note that different IT section will require different specification. For the software section, the deliverables will take the form of a screen design, logic design, representation of the codes, etc.

4. Development

   Base on the given specification, the respective IT section will develop the solution. For the software section, the deliverables will be a full running software program created from the specification.

5. Testing

   The test documents (Test Plan and Test Case) are normally created by the System Analyst during the development stages. The tester (normally a 3rd party) will use the Test Plan and Test Case to complete the testing. The deliverables will be a letter from the tester stating the outcome of the test.

6. Implementation

   At this stage onward the software is no longer a concern, the main objective now will be to prepare the environment. The implementation plan will list the tasks necessary to prepare the environment to accept new software, such as installation, training, conversion of data, change over method, etc. There are many deliverables here depending on what is listed in the implantation plan. For example for user training a user manual is normally created.

7. Review

   This is the final stage where the software user and team will sit down to review the software performance and to decide negotiate on the maintenance contract. If all goes well normally but not necessary a sign off letter will be the last deliverables.

Formal Method

Formal method is a way to takes the specification (written in natural language) and converts it into its mathematical equivalent. Thus it is normally used in the SDLC Analysis and Design stages. The natural language usually contains ambiguous, incomplete and inconsistent statement.

Once a specification in English for example is translated to a mathematical form, it will remove all ambiguity and uncertainty in that statement.

Formal method will also bring to light all different probable perspective to any given variables and functions that could have been hidden behind the English language.

This can be done using a number of formal languages such as Z notation, VDM, Algebra, Functional Programming, etc.

Creating software need not use formal method, having said that, having formal method imbedded into the SDLC does give the software huge advantages and also a new set of disadvantages

Advantage of formal method

Formal Method forces the System Analyst and Designer to think carefully about the specification as it enforce proper engineering approach using discrete mathematics.

Formal Method forces the System Analyst and Designer to see all the different possible states for any given variables and functions thus will avoid many faults and therefore reduces the bugs and errors from the design stage onward.

Disadvantage of formal method

Formal Method requires the person to know how to apply discrete mathematics. It will obviously slow down the analysis and design stage resources and time therefore also the cost of the project.

There are too many different formal methods and most of them are not compatible with each other.

Formal methods do not guarantee that a specification is complete. For each variable and function, it just forces the System Analyst and Designer to view the specification from a different perspectives but it does not guarantee that variable and functions will not be left out.

Critical software

Having known the advantages and disadvantages, most clients will see the justification to use formal methods for critical systems, but this thinking is now slowly fading as most clients realize the important and cost saving and convenience of having a good specification initially in the SDLC.

There are basically three (3) different types of critical systems;

1. Business Critical System

   Business Critical System refers to a system where the honesty and integrity of the business is paramount. All data kept in the system must be accurate at all times. If a fault is found the entire process must be stop to allow correction. Most government, business and manufacturing company that requires payment are business critical.

2. Mission Critical System

   Mission Critical System refers to a system where the continuous running of the system is paramount. Accurate takes a lower priority compare to the running of the system. Auto Teller Machine, Car ticketing system, Alarm Systems are mission critical.

3. Safety Critical System

   Safety Critical System refers to a system where the safety of everyone directly or indirectly affected by the system is paramount. Functionality and Accurate takes a lower priority compare to the safety of the users. Most medical, construction and oil rig systems are safety critical system.

Many organizations today require a combination of the above as such you may have a business mission critical system, a business safety critical system, etc.

Integrity Level

Integrity level refers to how much cost is an organization is willing to spend and how much risk is an organization is willing to take when developing software.

Integrity Level	Cost	Risk	Example of System
1	Low	Low	Address Book System
2	Low	High	Global Tsunami Warning System
	High	Low	Waste Water System
3	High	High	Nuclear Reactor System

Stages in Formal Method

Formal Methods can be divided into five (5) main stages;

1. **Formal Specification**

   This is where normal system specification is use and translated using a formal language into a formal specification. There are basically two type of formal language; Model Oriented (VDM, Z, etc) and Properties Oriented (Algebraic Logic, Temporal Logic, etc). This is the cheapest way to handle formal method.

   The formal specification generally does the following process.

   1. Get user requirement usually from the specification written in the natural language.

   2. Clarify the requirement using mathematical approach. This is to remove all ambiguous, incomplete and inconsistent statement.

   3. After statements are clearly identified. Then find all assumptions (Things that must be in place before something can happen) that is state or not stated within the clarified requirement.

   4. Then expose every possible logic defect (fault) or omission in the clarified requirement.

   5. Identify what are the exceptions (bad things) that will arise if the defects are not corrected.

    6. Find a way to test for all the possible each exception. Only when you can test for an exception can you be able to stop that exception from happening.

2. **Formal Proof**

   This level studies the formal specification and retrieves the goals of the formal specific. Then fixed rules are created and with these rules step by step instructions are listed to achieve the specified goals. This is relatively cheaper but there are more task steps.

3. **Model Checking**

   This level studies the formal specification and formal proof deliverables to make sure that the system or software contains ALL possible properties to be able to handle all possible scenarios that could happen for a given specification. This stage is beginning to be more expensive.

4. **Abstraction**

   This level uses mathematical and physical models to create a prototype of the entire system for simulation. This prototype is use to focus on the properties and characteristic of the system. This is the most expensive formal method.

Integrity Level and Formal Method Stages

The integrity level decided by the organization will determine how deep to go into the Formal Method stage.

Remember that the deeper into the formal method means more time and resources thus more cost will be incurred.

Integrity Level	Cost	Risk	Formal Method Stages
1	Low	Low	**Formal Specification**
2	Low	High	**Formal Proof**
	High	Low	**Model Checking**
3	High	High	**Abstraction**

->

Chapter 02

Proposition

1. Introduction to Proposition
2. Proposition Operators
3. Introduction to Truth Table
4. Truth Table and Proposition
5. Result terminology

Introduction to Proposition

Proposition is a declarative statement that can result in either true or false. The statement must be a constant thus the value cannot change.

In formal methods, the natural language is scan for propositions. Each proposition (either or false) will be translated into an expression usually joined using operators, and all the possible value for each proposition will be listed in a truth table to cover all possible value.

For example:

Statement	Result
FTMS College KL is a college.	True
FTMS College KL is not a college	False “liar paradox”

If two statements have the same meaning, that statement or proposition is considered to the equal even if they are spoken in different format or language.

The following are operations that can be done unto a proposition

Precedence	Symbol	Meaning	Example
1	( )		
2	⌐ ~	NOT	Fail means NOT Pass
3	^	AND Conjunction	Hard work AND good attitude
4	v	OR Disjunction	Code in Java OR Code in C++
5	→	Conditional Implies	If you pass then you get reward
6	↔	Equals Bi-directional Bi-implication	Pass if and only if marks above 40
	⊗	Different Exclusive	Success is different from Failure

Introduction to Truth Table

Truth tables is used to tell whether a propositional is true or false not only for one (1) instance but for all possible instance of the variable.

Since proposition is either true or false thus we can use a truth table to list down all the position state of that proposition.

Proposition: A = Ali is a boy.
Possible value: true or false therefore in a truth table.

(A) Ali is a boy.
T
F

Proposition: A = Ali is NOT a boy or NOT (Ali is a boy)
Possible value: true or false therefore in a truth table.

With an NOT operator true becomes false and false become true.

A	**Result** (⌐A)
T	F
F	T

For every increase in proposition, there is a double increase in possible value.

Proposition: Ali is a boy and Mary is a girl.
A = Ali is a boy
M = Mary is a girl

Notice that this is actually two proposition join with the AND operator.
We see it as A ^ M

Possible value: true or false for A and possible value: true or false for M

With an AND operator both statement must be true then the join statement will be true.

A	M	**Result** (A ^ M)
T	T	T
T	F	F
F	T	F
F	F	F

Once we understand the above explanation we can use the same principal for the remaining proposition operators.

OR

A	M	**Result** (A v M)
T	T	T
T	F	T
F	T	T
F	F	F

Implies

A	M	**Result** (A→M)
T	T	T
T	F	F
F	T	T
F	F	T

Equals

A	M	**Result** (A ↔ M)
T	T	T
T	F	F
F	T	F
F	F	T

Different

A	M	**Result** (A ⊗ M)
T	T	F
T	F	T
F	T	T
F	F	F

A more complex proposition

(P → Q) v (Q → P)

P	Q	P → Q	Q → P	**Result** (P → Q) v (Q → P)
T	T	T	T	T
T	F	F	T	T
F	T	T	F	T
F	F	T	T	T

Result terminology (how to describe the result)

Tautology a.k.a. Valid (the above example) happen when all the result in a true table is TRUE.

Un-satisfiable a.k.a. Invalid (Q ^ ⌐Q) happen when all the result in a true table is FALSE.

Satisfiable happen when at least one (1) truth value brings the TRUE result.

Contingent happen when at least one (1) truth value brings the TRUE result and at least one (1) false value also brings the TRUE result
->

Chapter 03

Proposition Exercise

Do the following proposition exercise and show the results to your lecturer.

1. ⌐ ⌐ P
2. (P → Q) v (Q → P)
3. P ^ (P → ⌐Q) ^ Q
4. (P → Q) ^ (P → ⌐Q)
5. (P v Q) v R
6. P v (Q v R)
7. (P v Q)
8. (Q v P)
9. P v (Q ^ R)
10. (P v Q) ^ (P V R)
11. P ^ (Q v R)
12. (P ^ Q) v (P ^ R)
13. (P → Q)
14. (⌐Q → ⌐P)
15. ⌐ (P v Q)
16. ⌐ P ^ ⌐Q
17. (P→ Q)
18. (⌐P v Q)
19. ⌐ (P ^ ⌐Q)
20. (P ↔ Q)
21. (P → Q) ^ (Q → P)
22. (P ^ Q) → R
23. P → (Q → R)

->

Chapter 04

Predicates

1. Introduction
2. Existential
3. Universal

Introduction to Predicate

Proposition is a constant declarative statement that can result in either true or false.

In formal methods, the natural language is scan for predicates. Each functions and variables (bounded or free) will be translated into an expression also usually joined using operators. Then all possible qualifiers will be listed. Sometime a truth table is used to cover all possible value.

But this is not practical as most statement actually contains variables and changes in the variables will change the validity of the statement to true or false, because some statement refers to a set of different elements.

Therefore we use predicate to handle such statement. For this subject we will use First-Order Logic only.

To use predicate there must at least two (2) elements;

1. A variable or a constant
2. A function that performs the variable

For example:

An ostrich has wing can fly, a eagle has wing can fly
The constant object here is “Wing”
The variable object here is either “Ostrich” or “Eagle”
The function here is “Fly”

Constant / Variable	Function	Result
Wing / Ostrich	Fly(Wing, Ostrich)	False
Wing / Eagle	Fly(Wing, Eagle)	True

Predicate quantifiers

The following are quantifiers that can be use with a predicate;

Symbol	Meaning	Example
∃	Existential	There exists some or For some the elements including itself
∀	universal	For every element For all the elements obviously including itself

Qualifier is normally place with an object (variable or constant)

Assuming a function call fly with only one (1) set of object Airplane, therefore using the above qualifier we can have two (2) different statements.

∀Airplane	Fly (Airplane)	= All airplane can fly
∃Airplane	Fly (Airplane)	= Some airplane can fly

If we have two sets of Airplane (Plane A and Plane B) using the above “Fly Faster” function we can have four (4) different statements. Plane A is denoted as A and Plane B is denoted as B.

∀A ∀B	Fly Faster (A, B)	= All Plane A fly faster then All Plane B
∀A ∃B	Fly Faster (A, B)	= All Plane A fly faster then Some Plane B
∃A ∀B	Fly Faster (A, B)	= Some Plane A fly faster then All Plane B
∃A ∃B	Fly Faster (A, B)	= Some Plane A fly faster then Some Plane B

If we have two sets of different object Boys and Girls using the above “Run Faster” function we also have four (4) different statements. Boys is denoted as B and Girls is denoted as G.

∀B ∀G	Run Faster (B, G)	= All Boys run faster then All Girls
∀B ∃G	Run Faster (B, G)	= All Boys run faster then Some Girls
∃B ∀G	Run Faster (B, G)	= Some Boys run faster then All Girls
∃B ∃G	Run Faster (B, G)	= Some Boys run faster then Some Girls

Predicates and Operators

All the operators used with proposition can be use to join different predicates.

Predicate Example	Symbol	Meaning
⌐ Fly (Airplane)	⌐	Airplane cannot fly
Fly (Airplane) ^ Fly (Birds)	^	Airplane fly AND Bird fly
Fly (Airplane) v Fly (Birds)	v	Airplane fly OR Bird fly
Repair (Airplane) → Fly (Airplane)	→	Repair the airplane then airplane will fly.
Repair (Car) ↔ Repair(Bus)	↔	Repair the Car is the same as Repair Bus
Repair (Airplane) ⊗ Repair(Bus)	⊗	Repair the airplane is different from Repair Bus

Predicates and Truth Table

Because the result of a predict function can be true or false therefore Truth tables can also be use with predicates. For example:

For a one function predict A = ⌐ Fly (Airplane)

A	**Result** (⌐ A)
T	F
F	T

For a two (2) function predict Fly (Airplane) ^ Fly (Birds)

A = ∀Airplane Fly (Airplane)
B = ∀Birds Fly (Birds)

A	B	**Result** (A ^ B)
T	T	T
T	F	F
F	T	F
F	F	F

Bound and Free (Bound) variables

This term is use in mathematics, in formal languages (mathematical logic and computer science).

A free variable is a notation that specifies places in an expression where substitution may take place.

A bound variable is a notation that specifies places in an expression no changes can take place.

The idea is related to a symbol that will later be replaced with Strings or values. It can also be represented by a wildcard character that stands for an unspecified symbol.

Base on the example, below:

1. ∀x, function (x, y)
2. ∃x, function (x, y)

If symbol x in the function represents a bound variable because it is stated in the qualifier. The symbol y in the function represents a free variable because it is not stated in the qualifier. The symbol w (or any other value) is a neither bound nor free as it was never use in the function.

∀x on the left refers to an instance of x
function (x, y) on the right should also refers to an instance of x

But technically the left x and right x could mean something else but this will cause a lot of confusion, thus the symbol on the left is usually kept in consistent with the symbol on the right.
->

Chapter 05

Predicates Exercise

Do the following predicate exercise and show the results to your lecturer.

1. Assuming the following:
   a is an animal, b is a bear
   function IsA (b,a) means that bear is an animal.

   Show all possible qualifiers for the above and state if that predicate statement is true or false and explain why.

2. Assuming the following:
   b is an boy, g is a girl
   function likes (b,g) means that b likes g.

   Show all possible qualifiers for the above and state if that predicate statement is true or false and explain why.

Scenario

Assuming the following:
h is a human and devil is not human
p is peace and war is not peace
function loves (h,p) means that h love p.

3. Translate into plain English the following expression:

   1. ∀h ∀p loves (h,p)
   2. ∀h ∃p loves (h,p)
   3. ∃h ∀p loves (h,p)
   4. ∃h ∃p loves (h,p)
   5. ⌐ (∀h) ∀p loves (h,p)
   6. ∀h ⌐ (∀p) loves (h,p)
   7. ⌐ (∀h ∀p) loves (h,p)
   8. ∀h ∀h loves (h,p) → Peace on earth
   9. ∀h ∀p loves (h,p) ^ ∃h ∀h loves (h,p) → Peace on earth
   10. ∀h ⌐ (∀h) loves (h,p) → ⌐ (peace on earth)

4. Base on the above scenario, show the true table for the following predicates

   1. ∀h
   2. ∀h ∀p loves (h,p)
   3. ∀h ∀h loves (h,p) → Peace on earth

->

Chapter 06

Sets

1. Universe
2. Elements
3. Cardinality
4. Sets Relationship
5. Sets Operation

Introduction to Set Theory

Set is very basic mathematical concept use to group objects. It is basically used to show the relationship between each type of objects. The Venn diagram is always used to picture the set theory graphically. A set is a group that may contain none or one (1) or more elements.

In formal methods, there are many ways to use set theory. One example will be to categories many types of objects available in a system mostly in the form of data. Base on the purpose of the particular system or software the objects are properly grouped and then related to one another.

Universe (U)

The Universe represents the scope of the system. All elements that is within the universe is considered necessary and elements not mentioned in the universe is considered none existence. Thus it is very important that we define the universe accurately.

Elements

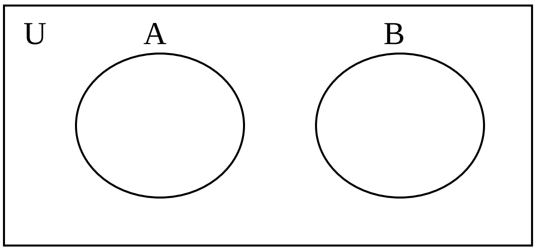
All elements in a set must be unique. In the Venn Diagrams the individual small letter element name are prefix with a dot. Capital Letter names is use to group many duplicates elements (sets) do not have a dot. The sequence or arrangement of the elements is not important.

There are two (2) ways to describe elements in a set.

1. Using a rule or semantic description:

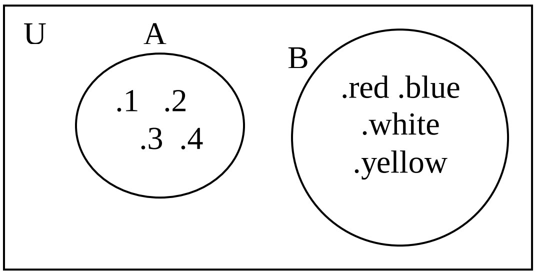
   A is the set whose members are the first four positive integers.

   B is the set of colors of the Malaysian flag.



2. Listing each member of the set or extensional definition. Elements in a list are enclosed inside curly brackets separated by commas.

   A = {4, 2, 1, 3}
   B = {blue, white, red, yellow}



Finite Elements

Some elements may be finite (with a starting and ending value) thus it can be representation as:

To show a value from 1 to 100 is represented as {1, 2, 3, ..., 100}

To show a value between 1 to 100 is represented as {2, 3, ..., 99}

F contains a number power by 2 minus 4 **such that (: or |)** all the numbers are integer starting from 0 to 19 is represented as

$F = \{n^2 - 4 \mid n \text{ is an integer; and } 0 \le n \le 19\}$

Or

$F = \{n^2 - 4 : n \text{ is an integer; and } 0 \le n \le 19\}$

Infinite Elements

Some elements may be infinite (no ending value) thus it can be representation as:

To show a integer value above 1 is represented as {1, 2, 3, ...}

F contains all the teachers in FTMS Global KL is represented as

F = { F | F all the teachers in FTMS Global KL }

Or

F = { F : F all the teachers in FTMS Global KL }

Cardinality

Cardinality means the number of elements in a set. Cardinality is denoted by vertical bars around the set.

For example
$| \{1, 3, 9, 15 \} | = 4$
$| \{1, 2, 3, . . .\} | = \infty$

Reserve Letter used by Mathematician

P Set of all primes: P = {2, 3, 5, 7, 11, 13, 17, ...}
N Set of all natural numbers: N = {1, 2, 3, . . .}
Z Set of all integers (positive/ negative / zero): Z = {..., −2, −1, 0, 1, 2, ...}
Q Set of all rational numbers (that is, the set of all proper and improper fractions):
R Set of all real numbers (rational numbers, irrational numbers)
C Set of all complex numbers: $C = \{a + bi : a, b \in R\}$.
H Set of all Quaternions: For example, $1 + i + 2j - k \in H$.

Terminology used to describe sets Relationship

Membership ($\in$)

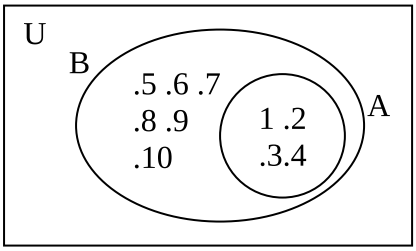
Membership happens when one element or a set is found inside another set. This symbol is normally used in describing a set for example

$A = \{A \in \text{Color of the rainbow}\}$

If A is a member of B, this is denoted $A \in B$.
If A is not a member of B then $A \notin B$

$A = \{1, 2, 3, 4\}$ therefore $4 \in A$ but $9 \notin A$
$B = \{\text{blue, white, red}\}$ therefore "blue" $\in B$ but "pink" $\notin B$

Subsets ($\subseteq$)/ Supersets ($\supseteq$)

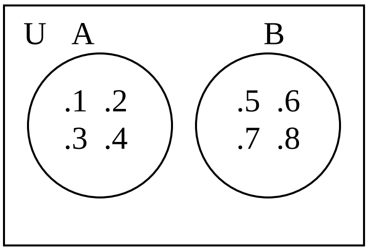
$A = \{4, 2, 1, 3\}$ $B = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ $A \subseteq B$ and $A \subset B$	

If every member of set A is found inside set B, then A is a subset of B ($A \subseteq B$).
If set B has every member of A and more then B is a super set of A ($B \supseteq A$)
This kind of relationship is also known as inclusion or containment.

If B is not a subset of A then we use the not a subset $\subsetneq$
If A is not a superset of B then we use the not a subset $\supsetneq$

We call "A" a proper subset of "B" if $A \subseteq B$ and $A \neq B$
Let $A = \{1, 2, 3\}$, $B = \{1, 2, 3, 4\}$, then $A \subseteq B$ and also $A \subset B$
Let $A = \{1, 2, 3\}$, $B = \{3, 2, 1\}$, then $A \subseteq B$ because $A = B$

Disjoint Sets

$A = \{4, 2, 1, 3\}$ $B = \{5, 6, 7, 8\}$	

If every member of set A has no relation with set B and vice versa then we say that A disjoint B. There is no special symbol to show this relationship.

NULL Set ($\varnothing$)

Every universe or set or subset contains a NULL set. A null set is an empty set ({ }) that carries no elements. We can say that the NULL set is a subset for every set.

Family Sets

There are times when a set does not contain individual elements but it contains many subsets. Conveniently this is call a family set and is it describes using the curly bracket within a curly bracket.

A = { {1, 2, 3, 4, 5} , {6, 7, 8, 9, 10} , {11, 12, 13, 14, 15} }

Power Sets (P(setName))

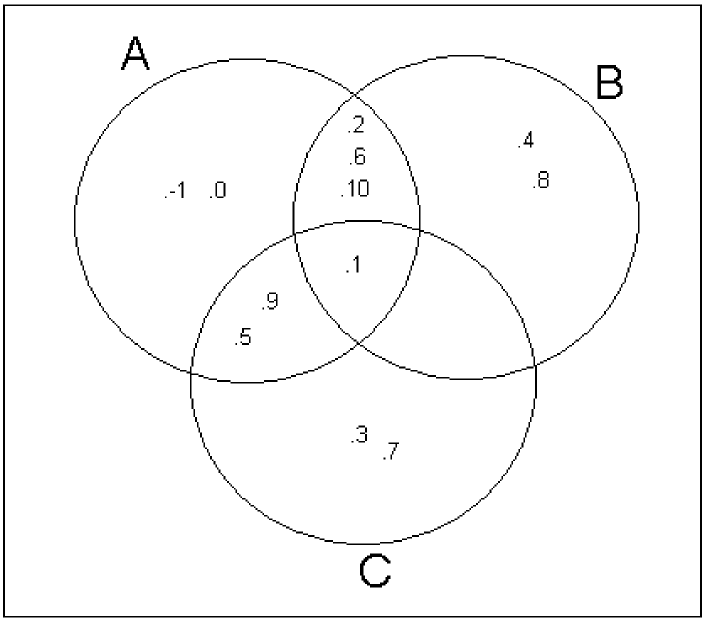
Remember that a set is a group that may contain none or one (1) or more elements. A power set means to show how many possible different ways to group all the elements in a set. In other words power set is the set of all subsets of a given set.

A = {1, 2, 3} , A has 3 elements, there is 8 possible ways to arrange this $2^3 = 8$.

P(A) = { $\varnothing$, {1}, {2}, {3}, {1,2}, {1,3}, {2,3}, {1, 2, 3} }

Terminology used to describe sets Operation

Given the following sets:
U = {-1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
A = {-1, 0, 1, 2, 5, 6, 9, 10}
B = {1, 2, 4, 6, 8, 10}
C = {1, 3, 5, 7, 9}



Union (∪): Add in all elements that are found in both sets.

A ∪ B = {-1, 0, 1, 2, 4, 5, 6, 8, 9, 10}
A ∪ C = {-1, 0, 1, 2, 3, 5, 6, 7, 9, 10}
B ∪ C = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
A ∪ B ∪ C = U

Intersect (∩): Show only elements that is found only in both sets

A ∩ B = {1, 2, 6, 10}   A ∩ C = {1, 5, 9}   B ∩ C = {1}

Difference (-): Also known as subtract, this show only elements that is found in this set but NOT found in another sets

A - B = {-1, 0, 5, 9}   A - C = {-1, 0, 2, 6, 10}   B ∩ C = {2, 4, 6, 8, 10}

B - A = {4, 8}   C - A = {3, 7}

Complement (‘): Show only elements that is found NOT found this sets

A’ = {3, 4, 7, 8}   B’ = {-1, 0, 3, 5, 7, 9}   C’ = {-1, 0, 2, 4, 6, 8, 10}

(A ∪ B)’ = {3, 7}   (A ∩ B)’ = {-1, 0, 3, 4, 5, 7, 8, 9}

Difference can be seen as the same as complement.

Equality: Both sets must have exactly the same number of elements with exactly the same value. Take note that sequence and duplication does not affect the set.

A = {3, 4, 7, 8}   Z = {4, 3, 7, 8}   therefore A = Z

B = {3, 4, 7, 8, 4}   Y = {3, 4, 7, 8, 7}   therefore B = Y

A = B = Y = Z

Compatible: Two sets are compatible if all element in one of the set can fit nicely inside another set.

A = {x, b}   Z = {x, b, c}   therefore A is compatible to Z
Because elements in A (x and b) can fit inside element in Z also have (x, b)

A = {x, b}   Y = {b, c}   therefore A is not compatible to Y
Because elements in Y cannot contain (x) and A cannot contain (c).

->

Chapter 07

Sets Exercise

Do the following predicate exercise and show the results to your lecturer.

1. Assuming the following:
   U = {all the letters in the English alphabet}
   $A = \{a, e, i, o\}$
   $B = \{q, w, e, r, t, y, u, l, o, p\}$
   $C = \{a, s, d, f, g, h, j, k, l\}$

   Show the set in Venn diagram for the following set expression.

    1. $A \cup A$
    2. $A \cap A$
    3. A'
    4. $(A')'$
    5. $A \cap \varnothing$
    6. $A \cup \varnothing$
    7. $A \cup B$
    8. $A \cap B$
    9. $(A \cup B) \cup C$
    10. $A \cup (B \cup C)$
    11. $(A \cap B) \cap C$
    12. $A \cap (B \cap C)$
    13. $(A \cup B) \cap C$
    14. $(A \cap C) \cup (B \cap C)$
    15. $(A \cap B) \cup C$
    16. $(A \cup C) \cap (B \cup C)$
    17. $(A \cap B)'$
    18. $A' \cap B'$
    19. $A' \cup B'$
    20. $A \cup (A \cap B)$
    21. $A \cap (A \cup B)$

2. Proof $(A - B) \cap C = (A \cap C) \cap B'$
   Draw the following using a Venn diagram
    1. $A - B$
    2. $(A - B) \cap C$
    3. $A \cap C$
    4. B'
    5. $(A \cap C) \cap B'$

->

Chapter 08

Series or Sequence

1. Sigmoid
2. Finite Sequence
3. Infinite Sequence
4. Arithmetic Sequence
5. Geometric Sequence
6. Find Sequence for a given term
7. Find Sum for a given sequence and term

Sequence

A sequence is simply a list, such as 2, 4, 6, ... where the numbers 2, 4, etc. are the terms of the sequence.

Please take time to understand this terminology:

Terms (usually represented with a subscript italic letter “*n*”) refers to the index for a given sequence starting from 0 to infinite. The sequence (usually represented with any small letter “a”) refers to the value for a specific term. For example:

Terms $_n$	0	1	2	3
Sequence (a)	0	2	4	6
a $_n$	a_0	a_1	a_2	a_3

The term 0 has the number 0
The term 1 has the number 2
The term 2 has the number 4
The term 3 has the number 6

The formula for this sequence will be 2n (2 multiple by Terms)

The symbol ∑ (sigmoid) is normally used to represent a sequence.

$$\sum_{n=1}^{10} 2n$$

The number starts with the term 1 and ends with the term 10. The formula for this sequence is 2n

$$2(1), 2(2), 2(3), 2(4), 2(5), 2(6), 2(7), 2(8), 2(9) , 2(10)$$
$$2 , \quad 4, \quad 6, \quad 8, \quad 10, \quad 12, \quad 14, \quad 16, \quad 18, \quad 20$$

The Sequence Summation is 2 +4+6+8+10+12+14+16+18+20 = 110

Type of sequence

There are two type of sequence;

Finite sequence
A finite sequence has both a starting value and an ending value.
E.g. 1, 2, 3, 4, 5 and 6

Infinite sequence
An infinite sequence has both a starting value but no ending value
E.g. 1, 2, 3, 4, 5, 6 …

Sequences can be applied in two areas;

Arithmetic sequence

Arithmetic sequence a.k.a. Arithmetic progression or is a sequence of numbers that goes from one term to the next by always **adding** (or **subtracting**) the same value.

Geometric sequence

Geometric sequence a.k.a. geometric progression is a sequence of numbers that goes from one term to the next by always **multiplying** (or **dividing**) by the same value. Value multiple by the same value, Value divided by the same value.

How to find a SEQUENCE for a given term

Arithmetic sequence

Finite or Infinite Sequence
a = the number for the first term in the sequence
d = the common difference (first term - second term)
n = the number of terms in the sequence needed

$$a_n = d(n-1) + a$$

Example:

Term	1	2	3	4	5	6	7	8	9	10
	1	3	5	7	9	11	13	15	17	**19**

Given X = 1, 3, 57, 9, 11, what is the 10 terms?

X_{10} = +2(10 – 1) + 1 = +2 (9) + 1 = 18 + 1 = **19**

Geometric sequence

Finite or Infinite Sequence
a = the number for the first term in the sequence
r = ratio for the sequence
n = the number of terms in the sequence needed

$$a_n = r^n$$

Example:

Term	1	2	3	4	5	6	7	8	9	10
	4	16	64	256	1024	4096	16384	65536	262144	**1048576**

Given X = 4, 16, 64, 256, 1024, what is the 10 terms?

$X_{10} = 4^{10}$ = **1048576**

How to find a SUM for a given term

Arithmetic sequence

Finite or Infinite Sequence

a = the first term in the sequence
a_n = the last term in the sequence
d = the common difference (first term - second term)
n = the number of terms in the sequence needed

$$S_n = \left(\frac{a + a_n}{2}\right)(n)$$

Remember:
Average the first and last then multiply by the number of terms

Example:

Term	1	2	3	4	5	6	7	8	9	10
	1	3	5	7	9	11	13	15	17	19

SUM = 1+ 3+5+7+ 9+11+13+15+17+19 = 100

Given X = 1, 3, 57, 9, 11, 13, 15, 17, 19 calculate the sum of X_{10}?

X_{10} = ((1+ 19)/2)10 = (20/2)10 = (10)10 = 100

Geometric Sequence

a = the number for the first term in the sequence
m = start terms for the given sequence
n = stop terms for the given sequence
r = ratio for the sequence

Finite sequence (**the first term value must above 0**)

$$S_n = \frac{a(1-r^n)}{1-r}$$

Example:

Term	1	2	3	4	5	6	7	8	9	10
	4	16	64	256	1024	4096	16384	65536	262144	1048576

SUM = 4 + 16 + 64 + 256 + 1024 + 4096 + 16384 + 65536 + 262144 + 1048576 = 1398100

Given X = = 4, 16, 64, 256, 1024, 4096, 16384, 65536, 262144, 1048576, Calculate the sum of X?

Sum X_{10} = 4 (1 - 4^{10}) / 1 – 4
= 4 (1-1048576) / -3 = 4 (-1048575) / -3
= -4194300 / -3 = 1398100

Infinite sequence that start (**the first term value must above 0**)

$$S\infty = \frac{a}{1-r}$$

Example:

Term	0	1	2	3	4	5	6	…
	4	16	64	256	1024	4096	16384	…

SUM = 4 + 16 + 64 + 256 + 1024 + 4096 + 16384 + …

Given X = = 4, 16, 64, 256, 1024, 4096, 16384 … Calculate the sum of X?

Sum X_{∞} = 4 / 1 – 4 = 4 / -3 = -1.3333333

->

Chapter 09

Series or Sequence Exercise

Do the following predicate exercise and show the results to your lecturer.

1. Study the following series and create the Sigmoid notation

   1) 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

   2) 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, …

   3) 2, 4, 6, 7, 8, 10, 12, 14, 16, 18, 20

   4) , 4, 6, 7, 8, 10, …

   5) 1, 3, 6, 10, 15, 21, 28, 36, 45, 55

2. Given Sigmoid notation create the sequences

   1) $\sum_{n=1}^{10} 2n$ 2) $\sum_{n=1}^{10} 2+1$ 3) $\sum_{n=1}^{10} 2^2$ 4) $\sum_{n=1}^{10} 2(n+1)$ 5) $\sum_{n=1}^{10} n^2$

3. Given Sigmoid notation calculate the sum

   1) $\sum_{n=1}^{10} 2n$ 2) $\sum_{n=1}^{10} 2+1$ 3) $\sum_{n=1}^{10} 2^2$ 4) $\sum_{n=1}^{10} 2(n+1)$ 5) $\sum_{n=1}^{10} n^2$

->

Chapter 10

Mathematical Proof

1. Direct Proof
2. Contradiction Proof
3. Contra positive Proof
4. Induction Proof

What is proof?

Proof simply means to be able to show that a statement is correct or true. No matter how the statement is twisted and turned or set against many different scenario, that statement comes up with the constant answer.

Before a statement can be proof it can have two (2); the conditions followed by the result. For example, if it rain then I will be wet. This can then express in a using proposition symbols as;

Rain → I am wet (if it rain then I will be wet)

Terminology

1) **Conjecture/ Hypothesis**

   This is a statement that is believe to be true but has yet to be proven.

2) **Axiom/ Postulate**

   If the statement is taken for granted to be true even though it was never tested, but base on logic it is assume to be true.

3) **Paradox/ Antinomy**

   This is a statement which appears to contradict itself or contrary to expectations

4) **Theorem**

   This is a statement that has been proven to be true.

5) **Un-decidable**

   This is a statement that cannot be proven right or wrong.

6) **Lemma**

   A proven theorem that is used to prove other statements

7) **Converse**

   Theorem that is reversed or turned upside down or inward out thus a converse of a theorem need not be always true.

Four (4) common proofing methods

1. **Direct Proof**

   In direct proof, the conclusion is established by logically combining the axioms, definitions, and earlier theorems. From the expression one can directly see the answer.

   For example: Rain → I am wet

2. **Contradiction Proof**

   In proof by contradiction, if that statement is true and we logically contradict it then it will not be true anymore.

   For example: Rain → I am wet
   No Rain → I am Dry

3. **Contra-positive/ Transposition Proof**

   Proof by transposition or contra-positive turns the statement inside out and upside down. This method swaps the result into the condition and negates both the result and condition.

   For example: Rain → I am wet
   I am Not wet → No Rain

4. **Induction Proof**

   This proof method insists that if the statement is true for one instance, it should be true for every instance.

   For example: Rain → I am wet

On Monday	(Rain → I am wet)
On Tuesday	(Rain → I am wet)
On Wednesday	(Rain → I am wet)
On Thursday	(Rain → I am wet)
On Friday	(Rain → I am wet)

->

Chapter 11

Testing

1. SDLC
2. Test Plan and Test case
3. Test Flow
4. Test Size
5. Test Depth
6. Other Testing

Revise SDLC

The SDLC can be divided into seven (7) stages;

1. Initial Study = Team collects information regarding the problem.
2. Analysis = Team discover the source of the problem.
3. Design = Team creates the specification for the solution.
4. Development = Team built the solution base on the given specification.
5. Testing = Test the software to make sure it solves the problem.
6. Implementation = Team prepare the environment to accept software.
7. Review = Team and client review the software.

Revise Formal Method Process

The formal specification generally does the following process.

1. Get user requirement usually from the specification written in the natural language.

2. Clarify the requirement using mathematical approach. This is to remove all ambiguous, incomplete and inconsistent statement.

3. After statements are clearly identified. Then find all assumptions (Things that must be in place before something can happen) that is state or not stated within the clarified requirement.

4. Then expose every possible logic defect (fault) or omission in the clarified requirement.

5. Identify what are the exceptions (bad things) that will arise if the defects are not corrected.

7. Find a way to test for all the possible each exception. Only when you can test for an exception can you be able to stop that exception from happening.

Testing stage

A test stage has only one (1) important purpose, that is to ensure that they software solution built solves the problem as specified in the analyst report and the specification.

Validation focuses on are we building the right solution and **Verification** focuses on are we building the product correctly are two terms used a lot during testing.

No software is 100% bugs free and testing cannot guarantee that there are no bugs, it can only ensure to a certain reasonable level that the system is able to perform the task it was created to do. It does not mean there are no more bugs in the system.

Once the testing is done, the tester will write a letter to give their opinion on the testing and the test result.

This letter will be given to the SA who then decides the following action, which may take the form of returning to:

1) The development stage where the programmer will debug the problem.

2) The design stage to redesign the specification then later continue to the development stage.

3) Worst case scenario, to return to the Analyst stage to redo the analyst for that given module which will then continue into the design and development stage again.

The same test document is reuse when the software returns to the testing stage. The SA may add in new test item in the test plan and new test case but the previous test document must remain intact. The tester will then repeat the testing for the failed modules and the new modules.

After a successful test, if there are future changes, the same test document is also reuse, thus the test plan can be use to audit the changes to ensure changes do not introduce new problem.

The test document consist of a test plan that list down all the test item, each test item will then be reflected in one (1) or more test case.

For example; Test plan will have many test items. In one (1) of the test item there is a test for the customer name. There may be three (3) test cases for that test item;

1) To test if the customer name can be save.

2) To test if the customer name can be numeric.

3) To test if the customer name can be blank.

Test plan

A test plan is a document that state down clearly every step (test item) that will be taken during testing, basically a systematic approach to testing a system.

The Test Plan is created first by the System Analyst (SA) using the Analyst and Design deliverables. This is done during the development stage when the work load has been transferred to the software programmers.

In order to create the Test Plan, the SA must understand the testing concept, because the strategy applied by the SA can easily be seen in the test plan.

The test plan will normally contain:

1. A sequence number call the test plan no.

2. The general description of the test item.

3. The date for the completion of each test plan no. One (1) test item can hold many test cases and each test case has a different purpose.

   The tested date is inserted only after ALL the test cases for one (1) test item is successfully tested. If after the testing is done and the test plan date remains blank means that the software fails the testing.

This is sample of a test plan

No	Description	Tested Date
1	System Installation into a Windows XP Professional Operating System	
2	Login Program.	
3	Main Menu	
4	Customer Module	
5	Customer Particular	
6	Customer Name	
7	Customer Search	
. . .	. . .	. . .

Test Case

After completing the test plan the SA will then created a test case. There will be at least one (1) test case for each of the test in the test plan. Each test case can contain only one (1) set of instruction and one (1) outcome.

There will always be a BLANK table inserted in the test case to be use by the tested to fill in the result of that particular testing.

The test case will normally contain:

1. The test plan no to tally back to the test plan and a test case number for that given test case.

2. The test instruction explains to the tester exactly how to run that particular test.

3. The expected result for that given test usually with a simple screen design or simple sentences to describe the result.

Please remember, SA creates the test cases for the tester.

This is sample of a test case

Test No	6	Description	Customer Name
Test Case No	1	Description	Save Customer Name
Instructions	Go to a new customer In the customer particular screen, Enter the Customer Name as John Click on the save button		
Expected Result	The customer record will be save and a pop up saying “New Customer Record Created”.		

Test Run :

No	Date	Comment	Good	Bad
1				
2				
3				
4				
5				

** Please state the successful date in the test plan when all test cases is done.

Testing Concept

Test Flow

1. Top Down

   This is a test flow that starts from a general level down to the specific detail level. Example for an inventory system will be to start from a main menu and slowly make the way down to the product module.

2. Bottom Up

   This is a test flow that starts from a detail specific level up to the general level. Example for an inventory system will be to start from the products and slowly make the way up to the main menu.

Test Size

1. Unit Testing

   This is a test that focuses on an individual specific independent module. Example for an inventory system will be to start test the products module alone and then the customer module alone.

2. Integration Testing

   This is a test that starts to join individual module. Example for an inventory system will be after testing out the product and customer module to test out the sales invoice module.

3. System Testing

   This is a test that starts to studies the system environment surrounding the software. Example for an inventory system will be to test if the bar code reader at the POS can read the barcode label on the product.

4. User Acceptance Testing

   This is the final a test where the end user will physically tested out the system themselves using real life data but still in a control testing environment. Example for an inventory system will be to ask the POS staff to test out the POS system and enter 100 products and produce the correct balance on the receipt.

Test Depth

1. Black box testing

   This is commonly known as a validation testing. This is an effectiveness test that is result base, input is given and output is produce and compared. Example for an inventory system will be to scan a barcode label on the product and see it appear on the POS interface with the correct total.

2. White box testing

   This is commonly known as a verification testing. This is an efficiency test, to test out how much resources and time is required to complete a process. Example for an inventory system will be to see how much time and processing resources to list and print a sales report for 1000 products.

3. Grey box testing

   This is partial effective and an efficiency test. Basic processing information is needed to discover how a process works. This test is normally use to create the test case.

Other Test

1. Boundary Testing

   This is a test done usually with a black box that can be done at the unit testing or integration testing stage. The main objective of this test is to make sure that the software can make the correct decision. All the possible result and alternative value is determined for the condition. Then extreme test data are generated to see if the software can produce a correct result.

2. Stress Testing

   This is a test done usually with a black box, unit testing approach. The main objective will be to break the system. Thus this test will take along time as it will continue until the system breaks down. From the break down, a safe level can be reach for contingency planning.

->

Chapter 12

Application to Formal Specification

1. Analyze Stage
2. Design Stage

Formal method

Formal methods are a way to apply mathematically-based techniques to the specification, development and verification of software for computer science or software engineering. When Formal method is applied it is **likely** to produce a more reliability and robustness specification design.

Thus it is important to stress that Formal Method in itself cannot guarantee perfect software. It depends very much on how the formal methods are interpreted and applied into the specification.

Formal Method stages consist of; **Formal Specification**, Formal Proof, Model Checking and Abstraction.

Formal Specification

Formal Specification is the initial part of formal method that describes what the system must do without saying how it is to be done. It is totally language independent and focuses only on the abstract rather then detail logic.

A formal specification can serve as a single, reliable reference point for those who investigate the customer's needs, those who implement programs to satisfy and those needs, those who test the results, and those who write instruction manuals for the system.

Formal Specification in the SDLC

Two (2) things are very important during requirement gathering; the data and the process done on the data. Formal specification will be applied directly on these things.

If formal Specification is use during the Analysis and Design stage, there is no need to use them again. If formal specification is not used by the SA and SD then the only window of opportunity to apply it will be during the pre-development stage.

If no formal specification was ever applied, and it is applied in the testing stage, this will expose a lot of possible bugs not catered for and the problem will loop back to the design and possibly the analysis stage.

Analyst Stage

> The main purpose of Analyze is to find the source of the problem. During this stage the System Analyst (SA) would collect all the data and processes (observation, document inspection, interview, etc…). The SA would then express the requirements into some form of diagrams such as DFD or Rich Picture, Use Cases and Case Diagram, etc…
>
> Then the SA will write a report (in English – common natural language) to indicate the source of the problem and some alternative solution.

When studying the current system the SA could apply proposition and predicate to every client's statement thus translating them into mathematical equivalent. This will remove ambiguity and expose all possible hidden state of the process and data. Proof is then use to be sure if the statement given by the client is correct. The SA can also apply set theories and series to categories and view data from different perspective. This is very helpful when SA needs to understand how reports are generated.

By doing this the SA can be to a certain degree confident to cover all possible alternative to a given statement.

Design Stage

The main purpose of Design is to find create a specification for the selected solution. It should be stressed that, the specification will be use to built the solution, thus a good specification will create a good software and a bad specification will create a bad software.

During this stage the System Designer (SD) uses the analyze report to create the specification. The SD also expresses their specification into some form of diagrams sometimes similar to those used by the SA.

Before creating the specification the SD could translate all the natural language found in the analyze report into mathematical equivalent (proposition and predicates) that will remove all ambiguity and uncertainty. Proof can be use here to ensure that every statement given is logically correct. Set theories and series are use to categories and view data from different perspective and to create relevant reports.

Then studying the mathematical form, the SD will be able to create the new system environment and also the solution that can cater for all possible scenario and state for each data and processes.

Development Stage

The main purpose of Development is to find built the solution base on the specification. During this stage the Senior Programmer (SP) will study the specification, create the relevant data structure, study the modules and delegate the programming team to develop the solution.

To apply formal specification at this stage will be a bit late but a small window of opportunity still exists.

During this stage the Senior Programmer (SP) will study the specification. The SP could translate all the natural language found in the specification into mathematical equivalent (proposition and predicates) that will remove all ambiguity and uncertainty. Proof can be use here to ensure that every statement given is logically correct. Set theories and series are use to categories and view data from different perspective and to create relevant reports.

The SP can then verify that the specification is complete before starting out the development.

If there is any problem with the design, the SP will stop the development and return the specification to the SD for correction. Worst case scenario, the entire specification is drop and the system is reanalyzed.

Testing Stage

The main purpose of Testing is to make sure that the solution solves the problem found in the analysis and created base on the specification. Before this stage the (SA) will have already created the test plan and test case. In this stage the tester will then use the test plan and test case to execute the testing.

To apply formal specification at this stage is really very late.

During this stage the SA will revise the specification built during by the SD. The SA could then translate all the natural language found in the specification into mathematical equivalent (proposition and predicates) that will remove all ambiguity and uncertainty. Proof can be use here to ensure that every statement given is logically correct. Set theories and series are use to categories and view data from different perspective and to create relevant reports.

After studying the specification, then the SA can create a Test Plan that will cover all aspect of the system. Using what is learnt from the Formal specification, the SA will be able to create a test case for each test item to test out all the different exception.

If there is any problem with the testing, the SA will stop the testing and return the specification to the SD for correction. Worst case scenario, the entire specification is drop and the system is reanalyzed.

Notice that this will not return to the development, because development only follows the specification created during the deign stage.

Formal Specification should not be use during the Implementation stage
->

Chapter 13

Z notation

Z notation

Before you can write create a formal specification in Z notation you must understand some basic concept and terminology.

Terminology and Concept

Data Refers to necessary data use in the system

Schema Refers to a structure that has two (2) sections the signature and predicates.

The signature (top section) contents all the set proposition thus the data, and variables.

The predicate (bottom section) shows the relationship between the variables. The relationship must be true in every state of the system and is maintained by every operation on it:

Functions Functions are a special case of relations in which each element in one set (domain - dom) has one more value associated with another set (range - ran). They are represented by arrows.

Injections
Surjections

Here is a list of the other standard types of function available in Z:

1	Partial function ⇸	X is a set and Y is a set X partial functions Y Each member X related to ONE (1) Y.
2	Total function →	X total functions Y These are partial functions whose domain is the whole of X; they relate each member of X to exactly one member of Y.

7	Partial Injection ⤔	This is a one-to-one mapping between elements in the domain and the range. The value in the domain may differ from the values in the range.
6	Total Injection $\rightarrowtail$	This is a one to many mapping where the domain completely populates the set. X $\rightarrowtail$ Y, then dom f =X. Then every X point to one Y
3	Partial surjections ⤀	This is a one to many mapping where the range completely populates the set Y. X ⤀ Y, then dom f =X. Then every X point to one Y
4	Total surjections $\twoheadrightarrow$	This is where both the domain and the range completely populate X and Y respectively. X $\twoheadrightarrow$ Y, then dom f =X. Then every X point to one Y and every Y point to X
5	Bijections ⤖	one-to-one total injective and surjective functions.

Variable Variable name here refers a pre-condition or current condition.
Meaning the variable is in a state before any changes is made to it

Variable’ Variable’ name here refers a post-condition.
Meaning the variable is in a state after changes is made to it

Variable? The value for this variable comes from an input device. (input)
Variable! The value for this variable will be shown or listed out (output)

The following steps are base on an example of a simple Birthday Book.

Steps

2. First we read the specification and list down all the important data. In the Birthday book there are two (2) data, NAME and DATE. Every data is seen as a Mathematics SET.

2. Then we create an object schema and identify the state space.

Birthday Book

known : NAME
birthday : NAME → DATE

known = dom birthday

Signature

We have a variable call ***known*** represents set NAME(s).
We have a variable call **birthday** represents set DATE.

This birthday (representing set DATE) has a function (partial function) that says every instance of known (representing set NAME) is related to ONLY ONE (1) instance inside set DATE.

Meaning everyone has only 1 birthday.

Predicate

The known variable (set NAME) exists within a domain call Birthday.
We can also show the above using the elements in the set.

Birthday Book

known = {John, Mike, Susan}
birthday = {(John,25/3),(Mike,22/12),(Susan,22/12)}

known = dom birthday

OR

Birthday Book

known = {John, Mike, Susan}
birthday = {John↦ 25/3, Mike↦ 22/12, Susan↦ 22/12}

known = dom birthday

Notice that the ↦ replaces the bracket, making it easier to read.

2. Initially nothing exist expect the birthday book (something like the constructor in a class). We need an operation to set all default value to the variables. Take note that **known** is a default variable found in all schema.

Init. Birthday Book

birthday

known = ϕ

2. Then we create an operation, let us do “add new birthday”, this will also uses a schema, we call it (operation schema).

Add Birthday

Δ Birthday Book
name? : NAME
date? : DATE

name? $\notin$ known

birthday’ = birthday U {name? $\mapsto$ date?}

Signature

Remember previously we declared known is a domain of birthday.

Δ means this is an operation schema for Birthday Book where the value in birthday domain will change

We have a variable call ***name?*** represents set NAME(s). This value will be input by the user.

We have a variable call **date?** represents set DATE. This value will be input by the user.

Predicate

The name? must not be found in the known (domain of birthday)

2. Then we create an operation, let us do “edit birthday”, this will also uses a schema, we call it (operation schema).

```
┌─ Edit Birthday ──────────────────────
│
│ Δ Birthday Book
│ name? : NAME
│ date? : DATE
│
├──────────────────────────────────────
│ name? ∈ known
│ date!  = birthday (name?)
│ date’  = date?
└──────────────────────────────────────
```

Signature

Remember previously we declared known is a domain of birthday.

Δ means this is an operation schema for Birthday Book where the value in birthday domain will change

We have a variable call ***name?*** represents set NAME(s). This value will be input by the user.

We have a variable call **date?** represents set DATE. This value will be input by the user.

Predicate

The name? must be found in the known (domain of birthday)

The date! will be shown base from the birthday domain where the name? is use as a parameter.

Meaning, base on the given name show the corresponding birthday

The date! will be updated with the new date? entered by the user.

2. Then we create an operation, let us do “delete birthday”, this will also uses a schema, we call it (operation schema).

Delete Birthday

Δ Birthday Book
name? : NAME
date? : DATE

name? $\in$ known
birthday’ = birthday - {name? $\mapsto$ date?}

Signature

Remember previously we declared known is a domain of birthday.

Δ means this is an operation schema for Birthday Book where the value in birthday domain will change

We have a variable call ***name?*** represents set NAME(s). This value will be input by the user.

We have a variable call **date?** represents set DATE. This value will be input by the user.

Predicate

The name? must be found in the known (domain of birthday)

The record with the name will be remove from the birthday domain.

2. Let us try to do “search birthday”, again we create an operation schema.

Search Birthday Base on Name

Ξ Birthday Book
name? : NAME
date! : DATE

name? ∈ known
date! = birthday(name?)

Signature

Ξ means this is an operation schema for Birthday Book where no value will change

We have a variable call ***name?*** represents set NAME(s). The value will be entered by the user.

We have a variable call **date!** represents set DATE. The value will be shown or listed by the operation.

Predicate

Remember previously we declared known is a domain of birthday.

The name? must be found in the known (domain of birthday)

The date will be shown base from the birthday domain where the name? is use as a parameter.

Meaning, base on the given name show the corresponding birthday.

Others ways to represent the same predicate logic:

known’ = dom birthday’ OR
known’ = dom (birthday { name? ↦ date?) OR
known’ = dom birthday { dom name? ↦ date? } OR
known’ = dom birthday { name? } OR
known’ = known { name?}:

2. Let us try to do “reminder birthday”, again we create an operation schema.

Remind Birthday

Ξ Birthday Book
today? : DATE
cards! : ***P*** NAME (Power-set of NAME)

cards! = {n : known | birthday(n) = today?}

Signature

Ξ means this is an operation schema for Birthday Book where no value will change

We have a variable call ***today?*** represents set DATE(s). The value will be entered by the user.

We have a variable call **cards!** represents a set function. The value will be shown or listed by the operation.

Predicate

Remember previously we declared known is a domain of birthday.

Cards will contain all n, where n is the domain birthday and the date that is birthday is part of the n must be equal to today.

Meaning, show all birthday that is today from the set of birthday dates.

Other ways to show the predicate will be:

cards! ∈ { n : known | birthday(n) = today?} ↔ cards! ∈ known ^ birthday(m) = today?

->

Chapter 14

Revision

Answer all the question below in your own handwriting then submit your propose answer to your lecturer for assessment.

1 Describe the following terms used in Formal Methods.

Proposition
Predicate
Truth Table
VDM-SL
Bound variable
Unbound variable
Sets Logic
Finite Series or Sequence Logic
Infinite Series or Sequence Logic
Programming language

2 Explain the following symbols used in Formal Methods.

“^” used in Proposition
“v” used in Proposition
“┐” used in Proposition
“⇒”used in Proposition
“⇔” used in Proposition
“⊕”used in Proposition
“∀”used in Predicate
“∃”used in Predicate
“Ø” used in Sets
“∞” used in series

3 Formal Methods consists of three (3) levels; Formal Specification, Formal Verification and Theorem Proves. Discuss the above three (3) levels and show the systems that requires the given level. Explain what will most likely happen if the wrong level is chosen for the wrong system.

4 Show the formula for the given series:

A = {1,2,3,4,5,6,7,8,9,10}
B = {2,4,6,8,10,12,14,16,18,20}
C = {1,3,5,7,9,11,13,15,17,19}
D = {2, 4, 8, 16, 32, 64, 128,256}
E = {1,3, 6,10,15,21, 28,36,45}

5 Using 1 for true and 0 for false, show the truth table for the following proposition:

1. ┐a
2. a ^ ┐a
3. a v b
4. a ^ b
5. (a => b) v (b => a)

6 Formal Methods are used when designing critical systems such as; Business Critical System, Mission Critical System and Safety Critical System. Using a clear example explain the above three (3) Systems. Explain what will most likely happen if formal method is not used when creating a critical system.

7 List the value and total sum for the given formula.

1	2	3	4	5
4 ∑ k2 k = 1	4 ∑ k+1 k = 1	4 ∑ 2k k =	4 ∑ 2k+ k = 1	4 ∑ 0k k = 1

8 Assume that the universe consists of only bicycles and fx(x,y) predicate indicates that x is faster then y, give the full statement for the following formula.

a. $\forall x \ \forall y \ f(x, y)$

b. $\forall x \ \exists y \ f(x, y)$

c. $\exists x \ \forall y \ f(x, y)$

d. $\exists x \ \exists y \ f(x, y)$

->

Reference:

To you, the student who is using this guide book.

Take note that I did not plagiaries any of the following online resources. I read the article, understood the contents and wrote it out in my own words. All the above knowledge and exercise are my own unless otherwise specified below.

References:

1. Formal methods - Wikipedia, the free encyclopedia [online]
   Available at http://en.wikipedia.org/wiki/Formal_methods
   [Accessed: 07 April 2011]

2. Proposition - Wikipedia, the free encyclopedia [online]
   Available at http://en.wikipedia.org/wiki/Proposition
   [Accessed: 07 April 2011]

3. Predicate - Wikipedia, the free encyclopedia [online]
   Available at http://en.wikipedia.org/wiki/Predicate_logic
   [Accessed: 08 April 2011]

4. Free variables and bound variables - Wikipedia, the free encyclopedia [online]
   Available at http://en.wikipedia.org/wiki/Free_variables_and_bound_variables
   [Accessed: 08 April 2011]

5. Set (mathematics) - Wikipedia, the free encyclopedia [online]
   Available at http://en.wikipedia.org/wiki/Set_(mathematics)
   [Accessed: 08 April 2011]

6. Arithmetic and Geometric Sequences [online]
   Available at http://www.purplemath.com/modules/series3.htm
   [Accessed: 12 April 2011]

7. Mathematical proof- Wikipedia, the free encyclopedia [online]
   Available at http://en.wikipedia.org/wiki/Mathematical_proof
   [Accessed: 14 April 2011]

8. Test plan - Wikipedia, the free encyclopedia [online]
   Available at http://en.wikipedia.org/wiki/Test_plan
   [Accessed: 14 April 2011]

9. J. M. Spivey, 1998. The Z Notation: A Reference Manual 2nd Edition