CHAPTER 1
Overview of Formal Methods

Chapter Outline

- What are formal methods?
- Features of formal methods
- Benefits of formal methods
- Limitations of formal methods
- Role of formal methods in Software Engineering
- Formal specification

Formal Methods

What are formal methods?

- Two fundamental **views** of formal methods as a discipline:
  - As **a branch of pure mathematics**
    - An intellectuality challenging research field which may or may not have any application in the “real world”
  - As **a branch of software engineering**
    - Concerned with the design and application of a certain set of development techniques and tools to create better software systems

What are formal methods?

- In software engineering, formal methods:
  - Refer to a variety of **mathematical modeling techniques** that are applicable to computer system design.
  - Used for the **formal specification** and **development of software/program** where each step leads to a final solution, follow proper method to make sure that we do not take wrong steps.
  - Each part of the programs are expressed in a **formal language**, they have a formal semantics and can be treated as **mathematical theories** to **describe system properties**.
  - Help you create software so that you can understand it before you run it.

TARUMT TUNKU ABDUL RAHMAN UNIVERSITY OF MANAGEMENT AND TECHNOLOGY

What are formal methods?

- Formal methods can be **useful** in:
  - Articulating and representing requirements
  - Specifying software: developing a precise statement of what the software is to do
  - Software design – data refinement involves state machine specification, abstraction functions and simulation proofs.
  - Coding verification
  - Enhancing early error detection
  - Developing safe, reliable, secure software-intensive systems.

Features of formal methods

- Strong typing
  - Can be much richer than ordinary programming languages
  - Provides economy and clarity of expression
  - Type-checking provides consistency checks
- Built-in model for computation
  - To discharge simpler type-checking constraints
  - Enhance proof-checking
- Modularization
  - Ability to break specifications into independent modules
  - Parameterized modules allow for easier reusability

Features of formal methods

- Total functions
  - Most logics assume total functions
  - Subtypes help make total functions more flexible
- Axioms and definitions
  - Axioms should be used carefully to avoid introducing inconsistencies
  - Definitional principle ensures well-formed definitions
  - In some languages type checking assertions will be generated to ensure valid definitions

Visions of formal methods

- Complement other analysis and design methods
- Help find bugs in code and specification
- Reduce development, and testing, cost
- Ensure certain properties of the formal system model
- Should be highly automated

Benefits of formal methods

- Increase confidence
  - Formal method is precise and there is no risk for misinterpretations
  - This can provide insights and understanding of the software requirements and software design
    - Clarify customers’ requirements
    - Reveal and remove ambiguity, inconsistency and incompleteness
    - Facilitate communication of requirement or design
    - Provide a basis for an elegant software design
    - Traceability - System-level requirements should be traceable to subsystems or component

Benefits of formal methods

- Early defect detection
  - Formal methods can be applied to the earliest design artifacts, thereby leading to earlier detection and elimination of design defects.
  - By using the mathematical modeling and formal analysis, can reduce the occurrence of defects in software products especially really important for critical projects (safety and security critical software)
  - Finding errors earlier reduces development costs.

Benefits of formal methods

- Contribute to the overall quality of the final product
  - Formal methods allow for a complete verification of the entire state space of the system and that the properties that can be proved to hold in the system will hold for all possible inputs.
    - Verification is the procedure of confirming that software meets its requirement. In other words it means checking the software with admiration to the specification
  - Formal methods have the property of completeness, i.e. it covers all aspects of the system.
    - Discovers ambiguity, incompleteness, and inconsistency in the software

Benefits of formal methods

- Correctness of software
  - By using rigorous mathematical techniques, it may be possible to make correct software
  - Formal analysis tools such as model checkers consider all possible execution paths through the system.
  - If there is any possibility of a fault/error, a model checker will find it.
  - In a multi-threaded system where concurrency is an issue, formal analysis can explore all possible interleaving and event orderings. This level of coverage is impossible to achieve through testing.

Benefits of formal methods

- Abstraction
  - If the working of software or hardware product is simple, then one can write the code straight away, but in the majority of systems, the code is far too big, which generally needed the detailed description of the system
  - A formal specification, on the other hand, is a description that is abstract, precise and in some senses complete.
  - It allows us to understand the big picture of the software product easily.

Benefits of formal methods

- Trustworthy
  - Formal methods provide the kind of evidence that is needed in heavily regulated industries such as aviation.
  - They demonstrate and provide concrete reasons for the trust in the product.
- Effective test cases
  - From formal specification, we can systematically derive effective test cases directly from the specification. It is a cost effective way to generate test cases.

Limitations of formal methods

- No guarantee on **completeness** of specifications
  - Generally, actual user requirements might be different from what the user states, and will usually vary with time.
  - While using formal methods, there is **no way to guarantee completeness** of a specification with respect to the user's informal requirements.
  - And, one can never be sure to have gathered all user requirements correctly.
  - Useful for consistency checks, but formal methods cannot guarantee the completeness of a specifications

Limitations of formal methods

- No guarantee on **correctness** of specifications
  - It is very difficult to identify whether or not a given program satisfies the given specifications.
  - It is impossible to prove the correctness of an existing program that has **NOT** been written with the correctness proof in mind.
  - Correctness proofs are only feasible if programming and proof go simultaneously.

Limitations of formal methods

- Dealing with complex language features
  - Formal definitions of semantics of most of the important language constructs and software system components are either not available or too complex to be useful.
  - Different aspects of a design may be represented by different formal specification methods.

Limitations of formal methods

- Requires a sound mathematical knowledge of the developer.
  - Difficult to use as a communication mechanism for non technical personnel.
  - Extensive training is required since only few developers have the essential knowledge to implement.
  - Also other important parts such as customer support, maintenance or installation must be dealt with separately.

Limitations of formal methods

- Time consuming and expensive
  - For the majority of systems, formal method does not offer significant cost or quality advantages over others.
  - To make analysis economically feasible, the cost of specification must be dramatically reduced, and the analysis itself must be automated. The cost of specification alone is often beyond a project's budget.

Role of formal methods in Software Engineering

Formal Methods = Formal Specification

+

Formal Verification

↑

Set theory, logics, algebra, etc.

Role of formal methods in Software Engineering

- Formal methods may be employed at a **number of levels**:
  - **Formal specification only (program developed informally)**
  - Formal specification, refinement and verification (some proofs)
  - Formal specification, refinement and verification (with extensive theorem proving)

Formal Specification

Formal Specification

- The specifications used in formal methods are well-formed statements which describes **“what”** the software system should do, not (necessarily) **“how”** the system should do it.
- It is a process of describing a system behavior and its desired properties by using mathematics to specify the desired properties of the system.
- Two activities of formal specification:
  - Modelling
  - Design

Modelling

- Mathematical models (such as Z) enable us to describe and predict program behavior accurately and comprehensively, and often much shorter and clearer than the code.
- Modelling makes the behavior of the program predictable – a good property for any program to have, an essential property for a safety-critical system.

Design

- Design means organising the internal structure of a program.
- Two dimensions:
  - Partition
    - Dividing the whole system into parts or modules that can be developed independently
  - Refinement
    - Adding detail, going from an abstract model that clearly satisfies the original requirements to a concrete design that is closer to code.
    - A.k.a reification

Formal Specification Approach

- Formal specification comprises of TWO approaches:
  - Model-Oriented Specification/ Model-Based Specification
  - Property-Oriented Specification/ Property-Based Specification

Model-based Specification

- An approach to formal specification where the system specification is expressed as a system state model. This state model is constructed using well-understood mathematical entities such as sets and functions.
- System operations are specified by defining how they affect the state of the system model.
- The state of the system is not hidden and state changes are straightforward to define.

Model-based Specification

- Can specify the operations that may be performed on your model
  - Indicate how the operations can transition the system from state to state.
  - **Transition** are described in terms of pre- and post- conditions
- Provide a direct way of describing the system behaviour in terms of a model using mathematical objects/ well-defined types (sets, sequences, relations, functions, tuples etc.) and defines operations by showing effects on model.

Model-based Specification

- Provides a twofold description of system behaviour:
  - Data structures (such as strings, numbers, sets, tuples, relations, sequences, etc.) that constitute the system state are described.
  - Operations that manipulate the state are defined using assertions in a notation similar to first-order predicate calculus.
- Utilise the tools of set theory, function theory, and logic to develop an abstract model of the system.

Model-based Specification

- Consider a simple instant messaging application for your cell phone. States the system may be in might include the very similar-sounding states:
  - Starting up
  - Sending message
  - Receiving message
  - Displaying message, and
  - Shutting down
- As for transitions, they might include:
  - Clicking the application icon to enter starting up
  - Or, pressing the send button to leave the sending message state

Model-based Specification

Sequential	Concurrent
Z (Spivey, 1992) VDM (Jones, 1980) B (Wordsworth, 1996)	CSP (Hoare, 1985) Petri Nets (Peterson, 1981)

- VDM and Z are the most widely used model-based specification languages.
- Concurrent - Defines operations in terms of simultaneously occurring events

Property-based Specification

- Describes the operations performed on the system, and the relationships that exists among these operations.
- State desired properties in a purely declarative way.
- Define system's behavior indirectly by stating a set of properties using axioms, rules etc.
- Consists of two parts: signatures, which determine the syntax of operations and an equation, which defines the semantics of the operations through a set of equations known as axioms

Property-based Specification

- Consists of:
  - Algebraic approach
    - The system is specified in terms of its operations and their relationships. Data type viewed as an algebra, a set of equations (axioms) state properties of data type's operations.
    - E.g. IOTA, OBJ, Larch, Anna
  - Axiomatic approach
    - Define the syntax of operations (what parameters they take and return). Describe desired behavior by providing model of system.
    - E.g. LOTOS

Property-based Specification

- Consider a simple instant messaging application for your cell phone. Then some operations might be:
  - Start up
  - Send message
  - Receive message
  - Display message, and
  - Shut down
- The relationships between these operations might include:
  - Startup must come before any other operation.
  - Shut down must be the last operation performed.
  - Display message comes during each send message and after each receive message

Formal Verification

Formal Verification

- Formal verification means showing that our code will do what we intend.
- It can also be defined to be the act of proving or disproving the correctness of some algorithm in a system with respect to a certain formal specification or property.
- Deals with the final product of our development: code in some executable programming language.

Formal Verification

- Where mostly applied:
  - Generally safety-critical systems
    - A system whose failure can cause death, injury or big financial loses (e.g. aircraft, nuclear station)
  - Particularly embedded system
    - Often safety critical
    - reasonably small and thus amenable to formal verification

Formal Verification

- There are two important forms: Model Checking and Theorem proving (formal proof).
- Techniques:
  - Manual - Human tries to produce a proof of correctness
  - Semi-automatic - Theorem proving
  - Automatic - Algorithm takes a model (program) and a property; decides whether the model satisfies the property

Theorem Proving (Formal proof)

- Proof is a convincing demonstration - based only on the specification and the program text, NOT on executing the program (the code).
- Proofs are constructed as a series of small steps, each of which is justified using a small set of rules.
- Eliminates ambiguity and subjectivity inherent when drawing informal conclusions.

Theorem Proving (Formal proof)

- Proofs can be done manually, but usually constructed with some automated assistance.
- Proof can provide greater confidence than testing because it considers all cases, while testing just samples some of them.
  - with special purpose capabilities
- Complete and convincing argument for validity of some property of the system description
  - Both the system and its desired properties are expressed in some mathematical logic.

Model Checking

- A technique relies on building a finite model of a system and checking that a desired property holds in that model.
- Can be used to check if an initial design satisfies certain properties.
- Operational rather than analytic
- Model checking is completely automatic.
- Two general approaches:
  - Temporal model checking
  - Automaton model checking

Model Checking

- Use model checkers:
  - SMV (Symbolic Model Verifier)
  - NuSMV
  - Cadence
- Model checker determines if the given finite state machine model satisfies requirements expressed as formulas in a given logic.
- Basic method is to explore all reachable paths in a computational tree derived from the state machine model.

Formal Specification Language

Formal Specification Language

- Formal specification is expressed in a formal specification language whose syntax and semantics are formally defined and at some level of abstraction of a collection of properties that some system should satisfy.
- Formal specification languages describe system properties that might include functional behavior, timing behavior, performance characteristics and internal structure.

Formal Specification Language

- The formal specification language comprises:
  - Syntax that defines specific notation used for specification representation and can be mechanically processed and checked.
  - Semantic, which uses objects to describe the system and is defined unambiguously by mathematical means.
  - A set of relations, which uses rules to indicate the objects for satisfying the specification.
  - Abstraction, above the level of source code and several levels possible

Formal Specification Language

- Generally non-executable - designed to specify **what** is to be computed, not **how** the computation is to accomplished
- Most are based on axiomatic set theory or higher-order logic.

Formal Specification Language

- Features of formal specification languages:
  - Explicit semantics - Language must have a mathematically secure basis.
  - Expressiveness – flexibility, convenience, economy of expression
  - Programming language data types - arrays, structs, strings, sets, lists, etc.
  - Diagrammatic notation
  - Convenient syntax

Formal Specification Language

- Some available formal specification languages:
  - Abstract State Machines (ASMs)
  - Alloy
  - B-Method
  - Java Modeling Language (JML)
  - LOTOS
  - RAISE
  - Petri Nets
  - VDM (Vienna Development Method)
  - **Z**

Why Formal Methods are NOT widely used?

- Software quality has improved
- Time-to-market is more important
- User interfaces are a greater part of systems
- Formal methods have limited scalability

- Lack of automated support
- Lack of user friendly tools
- Requires perfection and mathematical sophistication
- High learning curve
- Techniques not scalable
- Etc.

Questions to be Asked and Answered

- Do we really need formal methods?
- Was one formal method superior to another?
- What needs to be done to make “formal methods” industrial strength?
- Did formal methods quantitatively affect code quality?

* Please do some research on these questions

Summary

- Formal methods are a mathematically based techniques and tools for the specification, design and verification of software systems.
- Formal specifications are well-formed statements which describes what some software should do.
- Formal verifications are the act of proving and disproving the correctness of some algorithm is a system.
- Formal methods cannot guarantee the completeness of a specification

THANK YOU!!