

# An Introduction to Formal Methods for Specification and Analysis in Software Engineering

Siddharth Bhardwaj

March 28, 2020

# 1 Software Engineering and Formal Methods?

1. SE Methodology is based on a recommended development process:
  - (a) Analysis
  - (b) Specification
  - (c) Design
  - (d) Coding
  - (e) Unit Testing
  - (f) Integration and System Testing
  - (g) Maintenance
2. Formal Methods can be:
  - Be a foundation for describing complex systems.
  - Be a foundation for reasoning about systems.
  - Provide support for program development.
3. In essence, formal methods have a complementary approach to SE Methodology,

## 2 Testing: Static vs Dynamic Analysis

### 2.1 Static Analysis of code

- Does not require execution of code.
- Lexical analysis of program syntax and investigates and checks the structure and usage of individual statements; often automated.

### 2.2 Dynamic Analysis of code

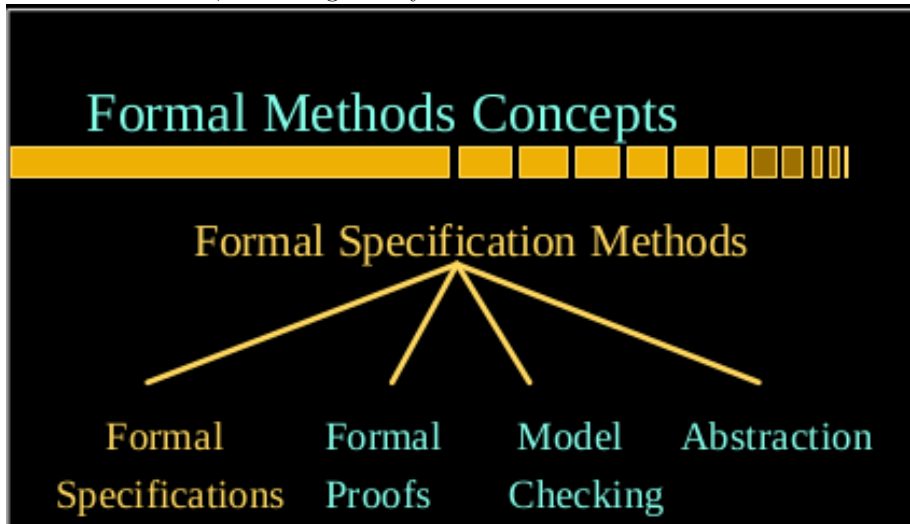
- Program run formally under controlled condition with specific results expected.
- Path and Branch Testing.

## 3 What are Formal Methods Actually?

**Techniques and tools based on Mathematics and formal logic** Allow us to assume various forms and levels of rigor.

## 4 Why Consider a Formal Method?

Most of today's systems are complex. Thus, we use formal methods to systems. Though keep in mind, the more complex a system, the harder is it to model. But as we will see, Modeling is only **one of 4 formal methods**.



## 5 Formal Methods Concepts

- Formal Specifications
- Formal Proofs
- Model Checking
- Abstraction

## 6 Formal Specifications

- **Definition:** Translation of prose (diagrams, tables, english text) into formal specification language.
- A formal spec provides a concise definition of high-level behavior and properties of a system.
- Well-defined language semantics support formal deduction about specification.
- **Informal Spec**
  - Free form, natural language

- Ambiguity and lack of organization can lead to incompleteness, inconsistency and misunderstandings.

- **Formatted Spec**

- Standardized syntax
- Basic consistency and completeness checks
- Impercise sematincs implies other sources of error may be still present not complete or sound.

- **Formal Spec**

- Syntax and Semantics rigorously defined.
- Percise form, perhaps mathematical.
- Eliminate impercision and ambiguity.
- Provide basis for mathematically verifying equivalence between specification and implementation.
- May be hard to read without training.

## 7 Formal Specs

- Goal: Describe external behavior without describing or constraining implemetation.
- This formal method has 2 parts:
  - Logic Theory:
    - \* Means by which one reasons about Specifications, properties and programs
    - \* First Order Predicate Calculus (quantification over calculus)
    - \* Second Order Predicate Calculus (quantification over relations)
    - \* Temporal Logic
  - Strcturing Theory:
    - \* Defines elements being reasoned about.

## 8 Types of Formal Specifications

### 8.1 Property Oriented

- State desired properties in a purely declarative way.
- Involves 2 Key Concepts:
  - Algebraic: Data type viewed as an Algebra, Axioms state properties of data type's operations.

- Axiomatic: Uses first order predicate logic, pre and post conditons
- Operation Specification: Describe desired behaviour by providing model of system.

## 8.2 Model Oriented

- Provide direct way of describing system behaviour(sets, squences, tuples, maps)
- Involves 2 Key Concepts:
  - Abstract Model (in terms of previously defined mathematical objects, eg: sets, sequences, functions and mappings)
  - State Machines.

## 9 Property Oriented

### 9.1 Uses

- Input-Output Assertions
- Sets of operations
- Axioms specifying behaviour of operations.

### 9.2 Two Parts of Specification

- Syntax
- Axioms.

## 10 Model Oriented: Abstract Model Specifications

- Build an abstract model of required software behaviour using mathematically defined types (sets, relations)
- Define operations by showing effects of that operation on the model.
- Specification includes:
  - Model Types
  - Invariant properties of Model
  - For each operation Name, Parameters and return values.
  - Pre and post conditons

## 11 Formal Proofs

- Complete and convincing argument for validity of some property of some the system description.
- Constructed as a series of steps, each of which is justified from a small set of rules.
- Eliminates ambiguity and subjectivity inherent when drawing informal conclusions.
- May be manual but usually constructed with automated assistance.

## 12 Model Checking

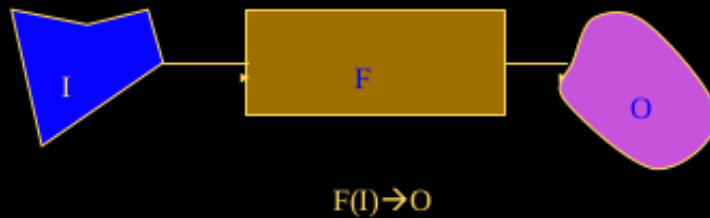
- Operational rather than analytic.
- State machine model of a system is expressed in a suitable language.
- Model checker determines if the given finite state machine model satisfies requirements expressed as formulas in a given logic.
- Basic method is to explore all reachable paths in a computational tree derived from the state machine model.

## 13 Abstraction

- Simplify and ignore irrelevant details
- Focus on and generalize important central properties and characteristics.
- Avoid premature commitment to design and implementation choices.

## Program as Mathematical Object

- $\text{Program} \Leftrightarrow \text{Mathematical Object}$
- $\text{Programming Language} \Leftrightarrow \text{Mathematical Language}$
- Can prove properties about the program



## 14 Logical Errors in Formal Specification

### 14.1 Logical Inconsistency

Easiest logical errors to detect.

### 14.2 Accuracy

does this specification mean what it intended? System invariants can help in detection.

### 14.3 Completeness

does the specification identify all contingencies and specify appropriate behavior for all cases? (Peer review can aid detection)

## 15 Techniques for Detection of Errors in Formal Specifications

Listed in increasing order of rigor and cost of application:

- Inspection of formal specification (manual)
- Parsing for syntactic correctness (automated)
- Type-checking for semantic consistency (automated)
- Simulation/animation based on the specification (automated).
- Theorem proving, proof checking, model checking for logic anomalies.

## 16 Formal Specification as a System Description

- Clarify requirements and high-level design.
- Articulate implicit assumptions.
- Identify undocumented or unexpected assumptions.
- Expose flaws.
- Identify exceptions.
- Evaluate test coverage.

## 17 Benefits of Formal Specifications

- Higher level of rigor enable a better understanding of the problem.
- Defects are uncovered that would likely go unnoticed with traditional specification methods.
- Identify defects earlier in life cycle.
- Can guarantee the absence of certain defects.
- Formal specification language semantics allow checks for self-consistency of a problem specification.
- Formal specification enable formal proofs which can establish fundamental system properties and invariants.
- Repeatable analysis means reasoning and conclusions can be checked by colleagues.
- Encourages an abstract view of system— focusing on what a proposed system should accomplish as opposed to how to accomplish it.



- Abstract formal view helps separate specification from design.
- Enhances existing review processes by adding a degree of rigor.

## 18 Limitation to Formal Methods

- Used as an adjunct to, not a replacement for, standard quality assurance methods.
- Formal methods are not a panacea (solution), but can increase confidence in a product's reliability if applied with care and skill.
- Very useful for consistency checks, but can not assure completeness of a specification.

## 19 Conclusion

- FM are no panacea(solution).
- FM can detect defects earlier in life cycle.
- FM can be applied at various levels of resource investment.
- FM can be integrated within existing project process models.
- FM can improve quality assurance when applied judiciously(with good judgement or sense) to appropriate projects.

## 20 Reference text

<https://web.mit.edu/16.35/www/lecturenotes/FormalMethods.pdf>