Formal Methods
Rabéa Ameur-Boulifa

Lecture 1: Introducing Formal Methods

Formal methods are mathematical approaches to software and system development which support the rigorous specification, design, and verification of computer systems. The exploit the power of mathematical notation and proofs.

Why? To avoid design error that can cause systems to fail in catastrophic ways leading to death or enormous financial loss.

Formal Method does not replace Testing, which can be used to show the presence of bugs but never to show their absence.

Formal Method = Specification Language + Formal Reasoning

- Formal Specification: precise and complete statement of the system requirements.
- Formal Proof: complete argument for a property validity (e.g. $(a+b)^2 = a^2 + b^2 + 2ab$).

Benefits of Formal Methods: better system understanding, earlier defects detection, increase product reliability
Limits of Formal Methods: require mathematical knowledge, have limited scalability (real systems are huge)
→ They can successfully complement simulation and testing analysis.

Tool support for Formal Method

- Specification/modeling: visual/textual editors
- Analysis: syntactic/type/model/proof checking
- Synthesis: refinement, code & test case generation

Lecture 2: Specification

A specification is an exact (precise, unambiguous) statement to be satisfied.
Specification languages: informal (free form, natural language), formal (syntax rigorously defined)
Formal Specification: syntax, semantic, proof theory.
→ unambiguous, consistent, may be incomplete

Approach #1: Logical Systems

A logical system consists of language (e.g. alphabet), semantics (rules to determine truth) and inference system (rules to write proofs).
An atomic proposition is a statement or assertion that must be true or false, but not both.
A proposition uses variables ($a, b, c, ...$), connectives ($\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$) and parenthesis (e.g. $a \wedge (b \vee c) \Rightarrow b$).
Interpretation: $\mathcal{I}: V \mapsto \{0,1\}$ provide the meaning of propositional formula (e.g. $\mathcal{I}(\neg A) = 1 - \mathcal{I}(A)$).

Terminology:
A formula can be:

- Satisfiable: there exists an interpretation such that it evaluates to true.
- Unsatisfiable: there is no interpretation such that it evaluates to true.
- Falsifiable: there exists an interpretation such that it evaluates to false
- Valid: under all the interpretations it evaluates to true.

Two formulas A and B are equivalent ($A \equiv B$) if every model of A is a model of B, i.e. $A \equiv B$ if for all $\mathcal{I}$, $\mathcal{I} \vDash A$ iff $\mathcal{I} \vDash B$, e.g. $A \Rightarrow B \equiv \neg A \vee B$.

First-Order Logic extends propositional logic (which only deals with facts but not individuals e.g. books and naturals) by adding quantified variables, constants, functions, and relations, e.g. $\forall x P(x) \Rightarrow \exists x P(x)$.

Approach #1: Algebraic Specification
This provides a mathematical framework for describing abstract data types.
Signature: names of objects and the operations on the objects.
Equations: use algebraic axioms to describe their characteristic properties.
Examples: Stack, Binary Tree, Natural Numbers.

Approach #2: B-Method
Based on the notion of abstract machine. It provides verification and concrete representations (source code). It is defined as states and operations to modify the state.

- STATE: variable set and STATE INVARIANT to be permanently verified.
- OPERATIONS to modify the STATE (pre-conditions and post-conditions on the STATE).
- Proof Obligations: each operation must preserve the STATE INVARIANT.
- Foundation: set theory and predicate calculus.

Clauses of B-Model:

- MACHINE: name
- SETS: abstract types
- CONSTANTS: logical variables with constant values
- PROPERTIES: constraints on the constants
- VARIABLE: state variables whose values can change
- INVARIANTS: constraints on variables
- INITIALIZATION: initial values for the abstract variables
- OPERATION: the various ways in which the abstract variable can be modified

Approach #2: Process-Algebra
Family of languages conceived for describing interactions between processes (systems with specific behavior). A system is described as a set of communicating processes, each process executes a sequence of actions, i.e. inputs/outputs or internal computation steps.
Processes are represented by vertices of edge-labelled and oriented graphs ($s \xrightarrow{a} s'$).

Key Points

- Formal system specification complements informal specification techniques.
- Formal specifications are precise and unambiguous. They remove areas of doubt in a specification.
- Formal specification forces an analysis of the system requirements at an early stage. Correcting errors at this stage is cheaper than modifying a delivered system.
- Formal specification techniques are most applicable in the development of critical systems and standards.

Lecture 3: Verification

Verification is the act of proving or disproving the correctness of a system with respect to formal specifications.

Verification: are we building the system right?

Validation: are we building the right system?

Verification Techniques

Approach #1: Test-Based Technique

Test the system at chosen inputs and observe behavior to make faults visible.

Problem with testing: What tests? How to generate? When to stop? How to evaluate results?

Black-box Testing (only check output), White-box Testing (check all internal paths), Automated Testing (choose input samples for each possible behavior).

Approach #2: Logical Inference

Theorem proving = logical deduction performed by machine.

Proof System:

- Axioms: set of well-formed formulas (wff)
  - $Ax_1: A \to (B \to A)$
  - $Ax_2: \big(A \to (B \to C)\big) \to \big((A \to B) \to (A \to C)\big)$
  - $Ax_3: (\neg A \to \neg B) \to (B \to A)$
- Deduction: given a set of hypotheses $\mathcal{H}$, a deduction is a sequence of wff $A_1, \dots, A_n$ where each A_i is:
  - An axiom ($A_i \in Ax$); or
  - A hypothesis ($A_i \in \mathcal{H}$); or
  - Follows from earlier steps by the rule of inference ($A, A \to B \vdash B$).

Formal approach #3: Model-Checking

Model checking is a technique for automatically verifying correctness properties of finite-state systems.

An exhaustive search of the state space of the system to determine if some specification is true or not.

1. Modeling: what the system does.
2. Specification: what the system ought to do.
3. Verification: check that the model respects the specification.

Kripke Structure: Model is a finite state machine (FSM), a directed graph consisting of nodes (or vertices) and edges.

The verification is done by exhaustively exploring the state space, determine whether some specification is true or not.