1/13/2023

PROJECT DOCUMENTATION

Formal methods in software engineering

- Jibran bilal khan (20-SE-046)
- Arslan dilshad (20-SE-046)
- Bilal ahmed chattha (20-SE-60)

Analyzing correctness in formal methods

- **Introduction:**

Analyzing the correctness of a software system through formal methods involves using mathematical techniques to formally prove that the system meets its specifications. This is done by creating a formal model of the system and using mathematical logic to prove that the system satisfies certain properties.Formal methods include techniques such as mathematical logic, formal verification, model checking, and theorem proving. These techniques can be used to analyze the correctness of a software system at different levels, including the requirements, design, and implementation of the system.

Formal methods can be used to prove that a system is free from certain types of errors, such as deadlocks and race conditions. They can also be used to prove that a system meets certain safety and security requirements

Formal methods are particularly useful for systems that have a high degree of criticality, such as aviation and medical systems, where a single error can have severe consequences. They are also useful for systems that have a high degree of complexity, such as distributed systems and embedded systems.

STATIC ANALYSIS

Before starting some discussion on formal methods and testing, it is necessary to introduce some terminology. Unfortunately, there is no consensus on these issues among the various research communities working in the area of software. There is not even an agreement on the meanings of the words “validation” and “verification” .Similarly, the word “testing” is often used with different meanings. Looking in a dictionary, one gets definitions such as: “subjecting somebody or something to challenging difficulties” In the case of software and formal methods, the “somebody or something” and the “challenging difficulties” are sometimes understood in different ways. In most cases, the entity under test is a system, and the “challenging difficulties” are inputs, or sequences of inputs, aiming at revealing some dysfunctions

- **Debugging or Testing Formal Specifications**

  In some other cases, testing is understood as debugging of formal descriptions or models. The formal description is the subject of the test. The challenges are either properties to be satisfied or refuted or inputs for some simulation of the future system, based on the formal description .As the main characteristic of formal specifications is the ability of reasoning, theorem proving is used either to prove that a required property is a consequence of the specification, or to refute a property that corresponds to a forbidden situation. The choice of such challenges is far from being simple. It requires a very good expertise in the application domain. As the specification may be wrong, is probably a good idea to make this choice as independent of it as possible , even if some positive experiments have been performed on mutation of formal specifications.

- **Testing Implementations**

  When testing implementations against a formal specification, the situation is different. As said in the introduction, the subject of the test is an executable system, whose internal state is often unknown. The system under test is not a formal entity.
  The only way to observe it is to interact via some specific (and often limited) interface, submitting inputs and collecting outputs.

- **Specifications, Implementations, and Testing**

  Given a specification SP and a system under test SUT, any testing activity
  must be based on a relation of satisfaction (sometimes called conformance relation) that we note SUT sat SP. This relation is usually defined on a semantic domain common to implementations and specifications

- **Test Experiments, Exhaustiveness, and Testability**

  The satisfaction relation SUT sat SP is generally a large conjunction of elementaryproperties (for instance it may begin by "for all traces in the specification…"). These elementary properties are the basis for the definition of what is a test experiment, a test data, and the verdict of a test experiment, i.e. the decision whether SUT passes a test
  However, an implementation's passing all the tests in the exhaustive test set does not necessarily mean that it satisfies the specification. This is true for a class of reasonable implementations. But a totally erratic system, or a diabolic one, may pass the exhaustive test set and then fail. More formally, the implementation under test must fulfil some basic requirements coming from the semantic domain considered for the implementations. As an example, in the case of finite state machines the

implementation must behave without memory of its history. Or when faced to nondeterministic SUT, some reasonable assumptions on the way of controlling it, or on the way of covering all the possible behaviors, are needed. We call such properties of the implementation the testability hypothesis, or the minimal hypothesis.
When restricting the class of implementations under test, using for instance some knowledge on the way it was developed, it is possible to lessen Exhaust(SP).

SIMULATION

Simulation models in formal methods are mathematical representations of a system that are used to analyze its behavior and verify its properties. These models are typically created using a formal language, such as first-order logic or temporal logic, and may include a combination of discrete and continuous variables. They can be used to model a wide range of systems, including software, hardware, and systems with both physical and logical components

There are several types of simulation models that can be used in formal methods, including:

- Finite State Machines (FSM): a mathematical model that describes the behavior of a system as a set of states and transitions between those states.
- Petri nets: a mathematical model that describes the behavior of a system as a set of places (representing states) and transitions (representing events or actions)
- Timed automata: a mathematical model that describes the behavior of a system as a set of states and transitions, with the added constraint of timing constraints
- Hybrid automata: a mathematical model that describes the behavior of systems that involve both continuous and discrete dynamics.

These models are usually analyzed using formal verification techniques such as model checking, theorem proving, and testing.

Simulation models in formal methods are useful for checking the correctness, reliability, and performance of systems, and for identifying potential errors before the system is implemented.

MODEL CHECKING

Model checking is a formal verification technique used in formal methods to automatically check the properties of a system against a formal specification. It is used to verify that a system, represented as a mathematical model, meets a set of desired properties or requirements.

The process of model checking involves:

1. Creating a mathematical model of the system using a formal language, such as first-order logic or temporal logic.
2. Specifying a set of properties or requirements that the system must satisfy using temporal logic or another formal language.
3. Using an automated model checking algorithm to check if the system's model satisfies the specified properties by exploring the state space of the model.
4. Generating a counterexample if the system does not satisfy a property.

Model checking can be applied to a variety of systems, including software, hardware, and systems with both physical and logical components. It is particularly useful for checking the correctness and reliability of complex systems, and for identifying potential errors before a system is implemented.

Model checking is an efficient method for checking the correctness of systems, as it can automatically explore the state space of the system's model and can handle large and complex models.

THE END