

Chapter 1

Formal methods

Course “Specification and Verification of Parallel Systems”

Juliane Päßler, Tobias John

Spring 2024



Section

Specification and Verification?

Spring 2024



Specification and **Verification** of parallel Systems.

- Specification is tailored towards verification method
- What is *verification*?

Correctness

- A system is **correct** if it meets its "requirements" (or specification)

Examples:

- **System:** A telephone system
- **Requirement:** If user A wants to call user B (and has credit), then *eventually* A will manage to establish a connection
- **System:** An operating system
  Requirement: A deadlock will never happen





Section

Recent Example

Spring 2024

Defining Requirements



Financial Times, 18.08.2023

What are requirements for such systems?

Defining Requirements

Cruise's self-driving taxis are causing chaos in San Francisco

These cars (and the company running them) have had a rough week.

BY HARRY GUINNESS | PUBLISHED AUG 17, 2023 4:00 PM EDT

Popular Sciece, 17.08.2023

The Big Read Driverless vehicles

+ Add to myFT

Here come the driverless taxis

San Francisco finally permits a 24/7 service despite local resistance. Can autonomous carmakers now win over the world?



Financial Times, 18.08.2023

What are requirements for such systems?



Section

Validation using Formal Methods

Spring 2024

How to guarantee correctness?

- not enough to show that it **can** meet its requirements
- show that a system **cannot fail** to meet its requirements

Dijkstra's dictum

"Program testing can be used to show the presence of bugs, but never to show their absence"

- motivates the usage of *Formal Methods*



Validation & Verification



- In general, **validation** is the process of checking if something satisfies a certain criterion
- Do not confuse validation with *verification*

Validation	**Verification:**
"Are we building the right product?", i.e., does the product do what the user requires	"Are we building the product right?", i.e., does the product conform to the specification

Approaches for validation

Testing
- check the actual system rather than a model
- focused on sampling executions according to some coverage criteria
- not exhaustive ("coverage")
- often informal, formal approaches exist (MBT)

Simulation
- A model of the system is written in a programming language, which is run with different inputs
- not exhaustive

Verification "[T]he process of applying a manual or automatic technique for establishing whether a given system satisfies a given property or behaves in accordance to some abstract description (specification) of the system"



What are formal methods?

FM

"Formal methods are a collection of notations and techniques for describing and analyzing systems" [**?**]

- **Formal**: based on "math" (logic, automata, graphs, type theory, set theory ...)
- formal **specification** techniques: to unambiguously describe the system itself and/or its properties
- formal **analysis/verification**: techniques serve to verify that a system satisfies its specification (or to help finding out why it is not the case)



Terminology: Verification



Formal verification

Formal verification is the process of applying a manual or automatic *formal* technique for establishing whether a given system satisfies a given property or behaves in accordance to some abstract description (*formal* specification) of the system

Terminology: Verification

Formal verification

Formal verification is the process of applying a manual or automatic *formal* technique for establishing whether a given system satisfies a given property or behaves in accordance to some abstract description (*formal* specification) of the system

Limitations?



Limitations

- Software verification methods do not guarantee, in general, the correctness of the code itself but rather of an abstract **model** of it
- It cannot identify fabrication faults (e.g. in digital circuits)
- If the specification is incomplete or wrong, the verification result will also be wrong
- The implementation of verification tools may be faulty
- The bigger the system (number of possible states) more difficult is to analyze it (*state space explosion problem*)



Any advantage?

be modest

Formal methods are not intended to guarantee absolute reliability but to *increase* the confidence on system reliability. They help minimizing the number of errors and in many cases allow to find errors impossible to find manually.



Using formal methods

1. We describe the system giving a *formal specification*
2. We can then *prove some properties* about the specification
3. We can proceed by:
   - Deriving a program from its specification (formal synthesis)
   - *Verifying* the specification wrt. implementation



Formal specification

- A specification formalism must be unambiguous: it should have a *precise syntax and semantics*
  - Natural languages are not suitable
- A trade-off must be found between **expressiveness** and **analysis feasibility**
  - More expressive the specification formalism more difficult its analysis

Do not confuse the specification of the system itself with the specification of some of its properties

- Both kinds of specifications may use the same formalism but not necessarily. For example:
  - the system specification can be given as a program or as a state machine
  - system properties can be formalized using some logic



Verifying specifications wrt. implementations

Mainly two approaches:

- **Deductive** approach ((automated) theorem proving)
  - Describe the specification φ_{spec} in a formal model (logic)
  - Describe the system's model φ_{imp} in the same formal model
  - Prove that $\varphi_{imp} \implies \varphi_{spec}$
- **Algorithmic** approach
  - Describe the specification φ_{spec} as a formula of a logic
  - Describe the system as an interpretation M_{imp} of the given logic (e.g. as a finite automaton)
  - Prove that M_{imp} is a "model" (in the logical sense) of φ_{spec}





Section

Classes of Systems and Properties

Spring 2024

Classification of systems

- Not all methodologies/techniques may be applied to all kind of systems
- Systems may be classified depending on
  - *architecture*
  - *type of interaction*



Classification of systems: architecture

- Analog vs. digital hardware
- Mono- vs. multi-processor systems
- Imperative vs. functional vs. logical vs. object-oriented software
- Concurrent vs. sequential software
- Embedded vs. local vs. distributed systems



Classification of systems: type of interaction

- **Transformational** systems: Read inputs and produce outputs – These systems should always terminate
- **Interactive** systems: Same as previous, but they are not assumed to terminate (unless explicitly required) – Environment has to wait till the system is ready
- **Reactive** systems: Non-terminating systems. The environment decides when to interact with the system – These systems must be fast enough to react to an environment action (real-time systems)



Classification of properties

Functional correctness The program for computing the square root really computes it

Temporal behavior The answer arrives in less than 40 seconds

Safety properties (*"something bad never happens"*): Traffic lights of crossing streets are never green simultaneously

Liveness properties (*"something good eventually happens"*): process A will eventually be executed

Persistence properties (stabilization): For all computations there is a point where process A is always enabled

Fairness properties (some property will hold infinitely often): No process is ignored infinitely often by an OS/scheduler





Section

Summary

Spring 2024

Summary

- Formal methods are useful and needed
- which FM to use depends on the problem, the underlying system and the property we want to prove
- for real complex systems, only part of the system may be formally proved and no single FM can do the task
- our course will concentrate on
  - temporal logics and transition systems as a specification formalism
  - safety, liveness and (maybe) fairness properties
  - LTL and CTL model checking
  - few other techniques from student presentation (e.g., abstract interpretation, timed automata, bounded model checking, ...)



References I



Bibliography