SLR 204: Basics of verification of distributed systems

Vadim Malvone
vadim.malvone@telecom-paris.fr
Introduction to Formal Verification
System Correctness

- Hardware and software systems are growing in their abilities and applications.

- From health-care and transportation to smartphones, systems are becoming a fundamental tool in everyday life!

- System failure can effect safety and induces a loss of money, as well as time and market reputation.
Critical systems

- Systems in which the failure is not an option.

- The main critical systems are divided into:
  - safety, in which failure can cost lives;
  - mission, in which failure can cost in terms of objectives;
  - business, in which failure can cost money.
Example: Therac-25

- The Therac-25 was a computer-controlled radiation therapy machine produced in 1982 after the Therac-6 and Therac-20 units.
- Problem: radiation overdose.
- Where: Canada and the USA.
- Consequence: three people dead and three seriously injured (of which one paralyzed).
- Why: errors in the software system and software/hardware interface (erroneous integration of pre-existing software components in the Therac-20).
Example: Denver Airport

- Designed to be an avant-garde airport, equipped with a complex baggage handling system computerized and 5,300 miles of cabling in optical fibers.
- Problem: errors in the management system baggage caused the crushing of suitcases and drove automatic carts against the walls.
- Where: Denver, the USA.
- Consequence: the airport opened 16 months late with a loss of 3.2 billion dollars and with an essentially manual baggage system.
Example: Pentium

- Problem: Prof. Thomas Nicely of Lynchburg College in Virginia found out that the Pentium chip was giving incorrect answers for some complex equations.
- Consequence: Intel was forced to withdraw the chip and replace it. All this cost Intel 450 million dollars.
- Why: the error was due to a design flaw of the floating point division algorithm.
- Result: since 1994 Intel has been using verification techniques!
Example: AT&T

- It is the world's largest telecommunications company.
- Problem: errors in the computer software that handled phone calls caused a 9-hour disconnect for all of the company's long distance networks.
- When: 1990
- Consequence: it was the worst blackout, among the many suffered in the history of the system. It involved hundreds of services.
- Why: a single wrong line of code.
Example: Ariane 5

- Problem: the test flight of the Ariane 5 failed after only 40 seconds from the start of the flight sequence. At an altitude of 3.700m, the rocket deviated from its course, separated and exploded.
- When: 1996
- Consequence: loss of 500 million dollars.
- Why: exception of the software occurred when converting data from 64-bit floating point to 16-bit signed integer.
- The navigation package was inherited from the Ariane 4 without executing an accurate compatibility check.
How to handle system correctness?

- Verifying the correctness of a software system before its use is a fundamental aspect.

- The main software verification methods are:
  - testing;
  - simulation;
  - formal verification.
Testing

- Software testing is an operation performed during the development phase.
- The product is checked for any malfunctions.
- In case of malfunctions, they are fixed before the release.
- **Drawbacks:**
  - not all input configurations can be checked;
  - input sets must be automatically generated;
  - no guarantee that “bad” inputs will ever be checked;
  - very difficult especially for multi-component systems.
Simulation

- Simulation is similar to testing.
- The verification process is performed on an abstract model of the system, by providing:
  - input data;
  - external events.
- Drawbacks:
  - there is no guarantee that all possible executions will be simulated;
  - it is often several times slower than the real system;
  - it can be very expensive.
Testing and simulation can reveal the presence of errors but can never establish the absence of errors.
Formal Verification (I)

- The ability to describe systems and express specifications in a precise and unambiguous manner.

- The ability to clearly define when an implementation meets specifications.

- The properties are checked for all possible system behaviors.

- The analysis is exhaustive even if performed on a formal model of the system, not on the real system.
We can check whether a system is correct with respect to a desired behavior (specification), by checking whether a formal representation of the system meets a formal specification.
Example: Scheduler (I)

- A scheduler should be designed so that jobs of two users are not printed simultaneously, and whenever a user sends a job, the job is eventually printed.
Example: Scheduler (II)

- Using formal methods we can check reliability for such a scheduler by:
  - providing an appropriate model for the scheduler $M$
  - a specification for the desired behavior $\phi$
  - a formal technique that allows to check that $M$ meets $\phi$
Formal Verification (III)

- Verification is:
  - formal, that is the model and the property are described in an unambiguous way;
  - definitive, that is verification proves or refutes the property.

- Opposite to:
  - testing of the real system on selected inputs;
  - simulation of a model on selected inputs;
  - manual inspection of the code or of the model.

- Two main approaches:
  - Deductive verification;
  - Model checking.
Deductive Verification

- It uses axioms to model the formal system and rules of inference to carry out the verification of requirements in proof form.
- The property to be verified is expressed by a formal system theorem.
- The theorem is proved with the help of an automatic theorem-prover.

Drawbacks:
- very difficult to automate (often impossible even in theory);
- requires user intervention;
- producing the formal system can be quite cumbersome;
- rarely effective.
1. Model: formal description (of part) of the behavior of the system
2. Specification: formal description of the expected properties of the system via logics
3. Verification: Does the model meet the specification?

\[ M \models \varphi \]
Advantages of Model Checking

- Applied to system models.
- Used at a very early stage of a project.
- Based on robust mathematical theories.
- Fast, compared to other rigorous methods such as deductive verification.
- Exhaustive, as it can check all possible computations.
- Diagnostic counterexamples.
- No problem with partial specifications.
- The specification can easily express several critical situations.
System Verification Scenarios

- System and specification models depend on the particular system/behavior we are dealing with.

- The algorithm analysis also depends on the specific setting.
System

- Closed systems:
  - Behavior is fully characterized by system states (one source of non-determinism).

- Open systems (system vs. environment):
  - Interaction with an unpredictable environment (two sources of non-determinism)

- Multi-agent systems:
  - The system is composed of several entities acting adversarial or in a cooperative way.
Specification (I)

- Example: ticket machine.
- "After any introduction of a new card, if tickets are released, the card will be ejected"
- Formalization in first order logic:
  - Intro_card(t): the card is introduced at time t
  - Tickets(t): tickets are distributed at time t
  - Eject_card(t): the card is ejected at instant t

\[ \forall t_0 \ [\text{Intro}_\text{card}(t_0) \land (\exists t_1>t_0 \ (\text{Tickets}(t_1) \land (\forall t_2,t_0 < t_2 \leq t_1 \Rightarrow \neg \text{Intro}_\text{card}(t_2))))] \]
\[ \Rightarrow \exists t_3 > t_0 \ [\text{Eject}_\text{card}(t_3) \land (\forall t_4,t_0 < t_4 \leq t_3 \Rightarrow \neg \text{Intro}_\text{card}(t_4))] \]

- Use of a variable modeling time.
- Very heavy!
Specification (II)

- **Solution: Temporal logics**

  - Temporal logics allows to describe the dynamic behavioral properties of systems, i.e., the evolution of systems computations along the time.

  - We intrinsically assume that systems computations are infinite.

  - Temporal logics extend classical propositional logic with temporal operators.

  - Primitive operators allow the expression of notions such as "always", "as long as", ...
Depending on the underlying nature of the time, we distinguish between:

- **Linear-time temporal-logics**
  - Every moment has a unique successor
  - Infinite sequences (words)

- **Branching-time temporal-logics**
  - Every moment has several successors
  - Infinite trees
History of Temporal Logics and Formal Verification

- Temporal logics were born as a philosophical study: ethics, free will, etc. Thomas Prior in the 1950’s is the first to use a concept of time-delay in computer circuits. With his Tense Logic, Prior has inspired many researchers.
- In 1977, Amir Pnueli is the first to use a future linear temporal logic (LTL) for the specification of non-terminating and concurrent programs: a temporal logic with “next” and “until”.
- Edmund Clarke and Ernest Allen Emerson in the early 1980’s developed a framework to temporal logic reasoning about programs (CTL and Model Checking).
- Independently, Jean-Pierre Queille and Joseph Sifakis essentially proposed the same method at this time.
- Pnueli won the 1996 Turing award for his contribution to temporal logic specifications.
- Clarke, Emerson, and Sifakis won the 2007 Turing award for their contribution to Model Checking.
Plan of the first part of the course

- 23/04, 1TH - Introduction to formal verification of systems: the model checking problem
- 23/04, 1TH - Step 1: Formal modelling
- 07/05, 1TH - Step 2.1: Formal specification of properties in Linear Temporal Logic (LTL)
- 07/05, 1TH - Step 2.2: Formal specification of properties in Computation Tree Logic (CTL)
- 21/05, 2TH - Step 3.1: Formal verification of LTL properties
- 28/05, 2TH - Step 3.2: Formal verification of CTL properties