

Solnt

Modeling and Verification

Didier Buchs

Centre Universitaire d'Informatique, Université de Genève

September 23, 2021

Modeling and Verification

Lectures: Thursday Battelle 316 - 14:15-16.00,

Exercises: Thursday Battelle 316 - 16:15-18.00,

Credits: Lectures (6 credits): oral exam (2/3) + TPS (1/3)
(more than 3.0 to pass the exam and the second session)

Assistants: Damien Morard

Teaching material

- ADT (algebraic abstract data types), and DD (Decision diagrams) (model checking) with contribution from **Alexis Marechal**, **Steve Hostettler**, **Dimitri Racordon** and **Damien Morard** from SMV group, Geneva.
- The outline of the Petri net course and temporal logic will follow a course by **Stefan Schwoon** and **Keijo Heljanko**, whose slides form the basis of this course.

The objective of the course

- The course aims to give the necessary skills for **modelling** , functional, parallel and distributed systems, **specifying** requirements for them, and for **verifying** these requirements hold on these systems.
 - **Modelling**: Petri nets, Algebraic Specification, Algebraic Petri nets, transition systems and process algebra
 - **Specifying**: formulae in temporal logic, equational logic.
 - **Verifying**: reachability analysis, model checking with decision diagrams, rewrite systems.
- Coding in Swift will be used to support concrete understanding of the theoretical concepts.



Software Modelling

Software Modelling:

quoi

What system to realize?

↳ la tâche à résoudre

as opposed to

différent

Software Implementation:

comment

How are we producing the product?

↳ language
↳ algorithm

Software Modelling needs expressive language to describe the expected functionalities of a system

Software Validation:

Are we producing the **right product?**
le bon produit

as opposed to

Software Verification:

Are we producing the **product right?** - Boehm
on le fait correctement

Software verification deals with checking if a software system performs the specified functionalities correctly

Analysis versus Modelling Systems

- **Analysis**

represent — problem domains from multiple perspectives

Discover characteristics of the system

→ prototype
↳ UML

- **Modelling**

Describe entirely and non-ambiguously the system

Need a very expressive language adapted to the problem domain

Focus on functional or non-functional aspects

↳ trier des valeurs

↳ ex temp
raisonnable
< 10 ms

Parallel and distributed systems: what are they?

- **parallel:** ^{→ partage de mémoire} multiple processes act simultaneously (concurrently),

- **distributed:** processes may co-operate to achieve a goal

A distributed system is one on which I cannot get any work done, because a machine I have never heard of has crashed.

L. Lamport

- **communication:** the processes exchange messages (synchronously or asynchronously)

- **reactive**: the system operates continuously, reacting on external events (rather than computing one result and then terminating)

- **nondeterminism:** there are alternative execution orders (stemming from concurrency, or from incomplete system description) *l'ordre n'est pas tjrs le même*

Parallel and distributed systems: problems

Systems with inherent parallelism often are so complex that they simply cannot be built by trial-and-error. Some problems caused by the specific nature of these systems are:

- **nondeterminism:** *→ user input*

- All **possible execution sequences** need to be **considered to** prove **correctness of a system.**

- **communication:**

- synchronous: **parties may deadlock** while waiting for another party.
- asynchronous: message transmission may be unreliable, **messages may arrive at any time** (including at inconvenient times) *↳ non déterminisme*

Parallel and distributed systems: problems

- **reactivity:**
 - need to consider potentially infinite executions.
- **distribution:** operating in an unknown environment
 - number of processes (e.g. participants in a protocol) may be unknown
 - behaviour of other components may be unknown

Specifications: properties of systems

- **Safety:** "The system never reaches a bad state"; some property holds throughout the execution.
 - Examples: deadlock freedom, mutual exclusion, ... *↳ bloquage de ressource*
- **Liveness:** "There is progress in the system"; some actions occur infinitely often. *↳ on avance*
- **Inevitability:** "Eventually, something will happen."
- **Response:** "Whenever X occurs, Y will eventually occur."
 - Examples: sent messages are eventually received, each request is served
- **Fairness assumptions:**
 - "X holds, assuming that no process is starved of CPU time."

Specifying systems

- A system specification captures the assumptions and requirements of the operations.
- Specifications should be unambiguous but not necessarily complete.
- Specifications describe the allowed computations (or executions). *↳ type user input*
- A specification is a “contract” between the customer and the software supplier. *↳ comportement*
- **In our setting**, specifications are **formal** descriptions of systems.

Advantages of formal description techniques

- **unambiguity**: specifications with precise mathematical meaning instead of colloquial language
- **correctness**: automated verification that the system fulfils its requirements
- **completeness**: *→ lié à spécification* the specification forms a checking list
- **consistency**: inconsistent or unreasonable requirements can be detected from specifications in an early phase
- Many software description techniques, such as UML, often aren't formal enough.

Why formal methods?

- Computers invade more and more aspects of our lives (home PCs, mobile phones, car electronics, ...)
- Software becomes increasingly more complex.
- Therefore:
 - Producing bug-free software becomes more difficult.
 - Cost of bugs can be enormous (economic, reputation, even human lives).
- The **later an error** is detected, the **more expensive** it is to correct.

Some (in)famous bugs

- Floating point error in Pentium processor (1994)
- Toll collection on German motorways not working
- Errors in space missions: Ariane 5, Mars polar lander
- Bug in Needham-Schröder protocol (key exchange)
- Other examples:

<http://www5.in.tum.de/~huckle/bugs.html>

Are formal methods a silver bullet?

- No - we can't hope to automatically analyse arbitrary properties of arbitrary programs, because
 - they are too large;
 - the problems may actually be undecidable in principle.
- Thus, formal efforts are concentrated on:
 - critical parts of systems: *→ systèmes essentiels*
 - decidable subclasses of systems or properties.

Applications for the methods

- **Communication**

- verifying and testing communication protocols
- evaluating the performance (queueing times, throughput, ...)

- **Safety-critical systems**

- railroad interlocking
- aircraft and air traffic control systems

- **Hardware design**

- processors, peripheral interfaces, memory caches and buses
- (verification of Pentium 4 FPU logic)

Plan of the lectures

- Modeling Data: ADT (Algebraic Abstract Data Types)
 - Syntax
 - Model Semantics
 - Deductive Semantics
 - Operational Semantics by rewriting
- Modeling Concurrency: Petri Nets (Reminder of 2nd year Bachelor)
 - Syntax
 - Semantics with transition systems
 - Operational Semantics with covering graphs
- Specifying Properties
 - Syntax of Temporal Logic
 - Semantics of Temporal Logic
 - Model Checking
 - Implementing Model Checking with Decision Diagram (SFDD)