Basics of advanced software systems

Lecture 6 – Introduction to formal methods
Module structure and objectives

Real-time Computing (15UE)
- Understand network and processor scheduling issues
- Get to know RT software & hardware technologies
- Design methods of large and complex critical embedded systems
- Illustration on Automotive Embedded Systems

Principles of formal methods (21UE)
- Understand when formal methods are appropriate and what to expect from them
- Provide first practical experience in formal verification: modeling, specification of properties, verification and result analysis
- Give an insight (limited) into the theory underlying formal methods
- Get to know industrial applications of formal methods and existing tools

MDE with executable UML (3UE)
- Introduction to MDE: PIM, PSM, etc
- What is xUML wrt UML? xUML vs fUML
- Domain modeling and action language
- UML modeling tools capabilities wrt xUML
Outline for today

1. **Introduction**: motivations for formal methods, use in industry

2. **Developing models**: automaton, specification of correctness properties

3. **Verification Techniques**: simulation, model checking, schedulability analysis, theorem proving, etc

4. **Focus**: Promela Spin for the verification of concurrent processes

5. **Lab work**: modeling & verification of an electronic purse
Building systems which are correct with respect to requirements

Two ways:

1. Checking that a system or a model of system meets the requirements

2. Correctness by construction, by using algorithms, protocols, architectures with guaranteed properties (e.g. time-triggered architecture)

Personal view: correctness by construction is certainly helpful (e.g. timing behavior) but is not possible for all facets of a system yet
Approaches for checking correctness
(adapted from [3])

- PROTOTYPES
- SIMULATION MODELS
- CHECKING CORRECTNESS
  - FORMAL MODELS
    (well defined notion of STATE and TRANSITION)
    - Formal methods
      - FORMAL VERIFICATION
        - Algorithmic verification: e.g., model checking
        - Deductive Verification: theorem proving
        - Analytic techniques: e.g. schedulability analysis
    - Code generation
Formal methods are mainly used in critical systems (not in general purpose software with strong time-to-market pressure)

1. **Safety-critical**: failure may endanger human life, ex: automotive, aerospace, railway signalling

2. **Security-critical**: failure means non-authorized access to sensitive information, ex: medical records, electronic transactions, security databases

3. **Business-critical (quality critical)**: cost of failure is very high, ex: semiconductor, financial applications, telecommunication

Safety standards and certification are a strong incentive for formal methods. For instance, standards for transportation systems and medical devices requires formal methods.
Formal methods are used for Validation & Verification (V&V)

- **Validation**: “system/component satisfies specified requirements” → proof of properties on the specification

- **Verification**: “products of a given development phase satisfy conditions imposed at the start” (code correct wrt specification – does not mean spec. is correct) → proof of correctness of implementation

Cost of change (time + money)
Testing versus formal verification

- Tests = check execution according to some coverage scheme. Tests informs us when there are errors but do not prove that there are no errors (except test can be exhaustive which is rare).

- Formal verification applies on models, not on what happens at run-time → models usually have to make simplifications.

In practice, formal verification can help to reduce the number of tests (costly) and help to generate the sequences of tests to be performed.
Formal method: theorem proving (deductive verification)

- System $\rightarrow$ set of axioms
- Properties on the specification $\rightarrow$ theorems to prove

- Data types and mathematical operations on those data types are formally defined
- Popular proofs assistants: PVS, COQ, Isabelle

✓ Complex, initial investment is high, can hardly be automatized

✓ Powerful and allow parametric verification. In practice, initial investment tends to pay off.
Formal method: model checking

- System + properties $\rightarrow$ finite state automaton

- Popular model checkers include SPIN and UPPAAL

- State explosion (but model checkers’ progresses and Moore’s law alleviates this issue)

- Not need for the user to care about the verification technique, is fully automatized ("push-button")
Formal method: analytic techniques

- Ad-hoc techniques on a formal model, e.g.:
  schedulability analysis for response time computations, queuing theory, et

- Limited applicability: e.g. performance evaluation, reliability

- Most often well suited for the problem at hand, there are dedicated tools
Example of a successful project [2]: Mondex Smart card electronic cash system (1999)
- Verification: under budget, within time, protocol flaw identified, 200-page manual proof, ex: no value created, all value accounted for
- No bugs discovered in operation
Modeling with automaton
- Automata: examples and definition
- from automaton to code
An automaton is made of states and transitions and evolves upon events

Example: a modulo 3 counter
Exercise: model a code entry system (digicode) as an automaton (from [4])

- 3 possible characters: A, B, C
- Door opens when the last 3 letters entered are A, B, A

A run of the automaton is \textit{accepted} if it terminates in a final state, here for instance: (1,A,2), (2,B,3), (3,A,4)
Modeling with automaton

1. Model the system under study
2. Associate to each state elementary basic properties (atomic propositions) that we know for sure if they are met or not
3. Use the atomic propositions to express and prove the more complex properties we are interested in

Atomic properties for the Digicode
- \( Pa \): A has just been typed in
- \( Pb \): B has just been typed in
- \( Pc \): C has just been typed in
- \( pred2 \): preceding state is state 2
- \( pred3 \): preceding state is state 3

Using the atomic properties, one can deduct the property: “if the doors opens, the last 3 letters typed in are A,B,A” and thus prove (part of) the correctness of the automaton
Defining automata

A finite state automaton is a tuple \((Q,E,T,q0,F,l)\) where

- \(Q\) is the set of states
- \(E\) is the set of labels of the transitions
- \(T\) is the set of transitions with \(T \subseteq (S \times L \times S)\)
- \(q0\) is the initial state, \(q0 \in Q\)
- \(F\) is the set of final states with \(F \subseteq Q\)
- \(l : Q \rightarrow 2^{\text{Prop}}\) is the set of atomic propositions verified for a given state, where Prop is set of atomic propositions
Defining automata: digicode example

\[
\begin{align*}
\text{Prop} &= \{P_A, P_B, P_C, \text{pred}_2, \text{pred}_3\} \\
Q &= \{1, 2, 3, 4\} \\
E &= \{A, B, C\} \\
T &= \{(1, A, 2), (1, B, 1), (1, C, 1), (2, A, 2), (2, B, 3), (2, C, 1), \\
&\quad (3, A, 4), (3, B, 1), (3, C, 1)\} \\
q_0 &= 1 \\
l &= \begin{cases} 
1 &\mapsto \emptyset \\
2 &\mapsto \{P_A\} \\
3 &\mapsto \{P_B, \text{pred}_2\} \\
4 &\mapsto \{P_A, \text{pred}_3\} 
\end{cases}
\end{align*}
\]
Programming an automata

• **Question:** write and test a program (language of you choice) that implements the automaton of the Digicode.

Write the program with the objective in mind that changing the automaton does not change the code (just the data).

Change the automaton so that the code to unlock the door becomes A, B, A, C and update your program accordingly.
References


