The bigger picture of system verification

[Automated Reasoning, 2012/2013 1b — Lecture 8]

Doina Bucur

RuG

Jan 2013
Undecidability vs. model checking

**Decidability**: algorithmically determining whether, e.g., any predicate-logic formula (say, a SAT instance) is satisfiable. Many problems are undecidable: SAT, the halting problem.

You’d now think that properties such as termination of algorithms are always undecidable.

In practice however, by identifying useful restrictions of the problem (e.g. being finite-state), and by considering different search heuristics, representations of the state space, and properties, most instances can be decided.
<table>
<thead>
<tr>
<th>Tool name</th>
<th>Tool developer</th>
<th>Symbolic analysis</th>
<th>Abstraction</th>
<th>Counterexample</th>
<th>BMC</th>
<th>Concurrency</th>
<th>Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASTRÉE</td>
<td>École Normale Supérieure</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td>C (subset)</td>
</tr>
<tr>
<td>CODESONAR</td>
<td>Grammathec Inc.</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td>C, C++, ADA</td>
</tr>
<tr>
<td>PolySpace</td>
<td>PolySpace Technologies</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td>C, C++, ADA, UML</td>
</tr>
<tr>
<td>PREVENT</td>
<td>Coverity</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td>C, C++</td>
</tr>
<tr>
<td>BLAST</td>
<td>UC Berkeley/EPF Lausanne</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>F-SOFT (abs)</td>
<td>NEC</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>Java PathFind.</td>
<td>NASA</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td>Java</td>
</tr>
<tr>
<td>MAGIC</td>
<td>Carnegie Mellon University</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>SATABS</td>
<td>Oxford University</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td>C, C++, SpecC, SystemC</td>
</tr>
<tr>
<td>SLAM</td>
<td>Microsoft</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>SPIN</td>
<td>Bell Labs</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td>PROMELA, C</td>
</tr>
<tr>
<td>ZING</td>
<td>Microsoft Research</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td>ZING (object oriented)</td>
</tr>
<tr>
<td>CBMC</td>
<td>CMU/Oxford University</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td>C, C++, SpecC, SystemC</td>
</tr>
<tr>
<td>F-SOFT (bmc)</td>
<td>NEC</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>EXE</td>
<td>Stanford University</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>SATURN</td>
<td>Stanford University</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td>C</td>
</tr>
</tbody>
</table>

1) does not support shared memory concurrency
2) originally developed by Bell Labs, now freely available
3) C is supported by automatic translation to PROMELA

Timed model checking

Reactive systems, e.g., device drivers, communication protocols, ATMs have stringent timing constraints.

For a train crossing: on detecting the approach of a train, the gate is closed within a certain time bound in order to halt traffic before the train reaches the crossing.

A first choice to be made is the time domain: is it discrete or continuous?

A discrete time domain is conceptually simple, sufficient for synchronous systems (but leads to large models), and traditional model checking suffices.

General schemes exist for a continuous time domain.

“the light cannot be continuously switched on for more than 2 minutes”

\[ \forall \square (on \rightarrow \forall \Diamond > 2 \rightarrow \neg on) \]

“The traffic light will turn green within the next 30 seconds.”

\[ \square (red \Rightarrow \Diamond \leq 30 \text{ green}) \]

Timed CTL
for reals transitions emanate from the initial state example path of proceeds equally fast in both components. up to isomorphism. This is due to the fact that where as the location invariant

The transition system

Timed Automata

Labelings and actions are omitted. this automaton is to stay in location can progress without any restriction while residing in is reset to 0, the automaton traverses the self-loop at location

Figure 9.5(b) gives an example of an execution of this timed automaton, by depicting the

680

57

−−−−−→

−−−−−

−−

−−−−−

2+0

−−−−−

−−−−−−

switch

Switch

S

Doina Bucur (RuG)

System verification: The bigger picture

Jan 2013

5
Collision Avoidance Protocol: This protocol is implemented on top of an Ethernet-like medium such as the CSMA/CD protocol. It is to ensure an upper bound on the communication delay between the network nodes. In [6], the protocol is designed and proved correct using UPPAAL. The two main established properties show that the protocol is collision-free, and it does ensure an upper bound on the user-to-user communication delay (assuming a perfect medium).


UPPAAL is an integrated environment for modeling, validation and verification of real-time systems modeled as networks of timed automata with data types (bounded integers, arrays, etc.). Typical application areas are real-time controllers and communication protocols. The model-checker can check invariant and reachability properties.

http://www.uppaal.org/
Probabilistic model checking

Model-checking focuses on the absolute guarantee of correctness — “it is impossible that the system fails”.

In practice such rigidity is hard to guarantee. Systems are subject to various phenomena of a stochastic nature, such as message loss or garbling and the like, and correctness — “with 99% chance the system will not fail” — is less absolute.

\[
P=1(\diamond \text{delivered}) \land P=1\left(\Box (\text{try\_to\_send} \rightarrow P\geq 0.99(\diamond \leq 3 \text{delivered}))\right)
\]

“Surely some message will be delivered (first conjunct), and surely for any attempt to send a message, with probability at least 0.99, the message will be delivered within three time steps.”

Probabilistic CTL
Discrete-time Markov chains

Markov chains are depicted by their underlying digraph where edges are equipped with the transition probabilities in $]0,1]$. If a state $s$ has a unique successor $s'$, i.e., $P(s, s') = 1$, the transition probability may be omitted.

**Example 10.2. A Simple Communication Protocol**

Consider a simple communication protocol operating with a channel. It is error-prone in the sense that messages may be lost, see the Markov chain depicted in Figure 10.1.

Here, $ι_{\text{init}}(\text{start}) = 1$ and $ι_{\text{init}}(s) = 0$ for $s \neq \text{start}$, i.e., start is the unique initial state.

In the state start, a message is generated that is sent off along the channel in its unique successor state try. The message is lost with probability $\frac{1}{10}$, in which case the message will be sent off again, until it is eventually delivered. As soon as the message has been delivered correctly, the system returns to its initial state.

Using the enumeration $\text{start}$, $\text{try}$, $\text{lost}$, $\text{delivered}$ for the states, the transition probability function $P$ viewed as a $4 \times 4$ matrix and the initial distribution $ι_{\text{init}}$ viewed as a column vector are

$$P = \begin{pmatrix}
0 & 1 & 0 & 0 \\
0 & 0 & 1 & \frac{9}{10} \\
1 & 0 & 0 & 0 \\
\frac{1}{10} & 1 & 1 & 0
\end{pmatrix},
ι_{\text{init}} = \begin{pmatrix}
1 \\
0 \\
0 \\
0
\end{pmatrix}.$$

An example of a path is $π = (\text{start try lost try lost try delivered})$. Along this path each message has to be retransmitted two times before delivery. It follows that $\text{inf}(π) = S$. For $T = \{\text{lost}, \text{delivered}\}$, we have $P(\text{try}, T) = 1$.

**Markov chain for a simple communication protocol**

**Discrete-time Markov chains**
**Probabilistic model checking**

- **Overview of the probabilistic model checking process**
  - Two distinct phases: model construction, model checking

**Model construction**
- High-level model
- PRISM language description

**Model checking**
- Property
- PCTL/CSL/LTL/… formula
- DTMC, MDP or CTMC

**Result**

**PRISM** is a probabilistic model checker, a tool for formal modelling and analysis of systems that exhibit random or probabilistic behaviour. It has been used to analyse systems including communication and multimedia protocols, randomised distributed algorithms, security protocols, and biological systems.

http://www.prismmodelchecker.org/
SMT-based model checking

The **Satisfiability Modulo Theories (SMT)** problem is a decision problem for **logical formulas** in particular theories, expressed in classical first-order logic with equality.

Theories used in computer science are: the theory of real numbers, the theory of integers, and the theories of data structures such as lists, arrays, bit vectors.

SMT is a generalization of **Boolean SAT**. The first-order theory of the **natural numbers** with addition (but not multiplication), called **Presburger arithmetic**, is **decidable**. Most SMT solvers only support such decidable problems.
\[ C = \{ i_1 = (\text{TRUE} \ ? \ \text{select}(a_0, 0) : i_0), \]
\[ x_1 = ((x_0 > 0 \land x_0 < 10) \ ? \ x_0 + 1 : x_0), \]
\[ x_2 = ((x_0 > 0 \land \neg(x_0 < 10)) \ ? x_1 - 1 : x_1), \]
\[ a_1 = (\text{TRUE} ? \ \text{store}(a_0, y_0, i_1) : a_0) \}\]

\[ \mathcal{P} = \{ \text{TRUE} \supset (y_0 > 0 \land y_0 < 5) \}\]
State of the art in... avionics

[ Compiler verification and beyond: verified tools for high-assurance software
Xavier Leroy, INRIA Paris-Rocquencourt, HCSS, 2011 ]

Some success stories in verification of avionics code (e.g. fly-by-wire systems)

Simulink, Scade

Rockwell-Collins toolchain
(model-checking + PVS proof)

Simulink, Scade

Caveat
(program proof)

C code

Astrée
(absence of run-time errors, incl. floating-point)

AiT WCET
(precise time bounds)

Executable
State of the art in... code security

**Exploit**: software, data, or commands which take advantage of a bug, glitch or vulnerability in order to cause unintended or unanticipated behavior to occur on computer software or hardware.

"[My] hobbies include watching the security industry reinvent 30-year-old academic research... badly."

*S. Heelan,*
Security Researcher,
founder, Persistence Labs

http://en.wikipedia.org/wiki/Valgrind