Memory-Aware Bounded Model Checking of Linear Hybrid Systems

Marc Herbstritt
(joint work with Erika Ábrahám, Bernd Becker, Martin Steffen)

www.avacs.org

21. Februar 2006
9. ITG/GI/GMM Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen
Motivation

⇒ Analysis of safety-critical systems is mandatory
⇒ Real-world scenarios embed discrete control in continuous environments
⇒ Analysis requires modeling using hybrid systems
Overview

- Bounded Model Checking for Hybrid Systems
- Symmetry-based Learning
  - Leads to high memory requirements
- Memory-aware storage
  - Parametric data structures
- Experimental results
- Conclusions
Hybrid system (Thermostat controller)

\[
x = x^{\text{max}}
\]

**off**

\[
\dot{x} \leq 0 \\
x \geq x^{\text{min}}
\]

\[
x = x^{\text{max}}
\]

**on**

\[
\dot{x} \geq 0 \\
x \leq x^{\text{max}}
\]

\[
x = x^{\text{min}}
\]
Hybrid system (Fischer’s mutual exclusion protocol) used for asynchronous distributed systems with multiple clocks
Counterexamples of length $k$ described by first-order logic formulas over $(\mathbb{R}, +, <, 0, 1)$:

$$\varphi_k(s_0, \ldots, s_k) : \text{Init}(s_0) \land \text{Trans}(s_0, s_1) \land \ldots \land \text{Trans}(s_{k-1}, s_k) \land \text{Bad}(s_k)$$

- $\varphi_k$ is satisfiable $\iff$ exists run of length $k$ leading to an unsafe state

$\Rightarrow$ Check $\varphi_k$ incrementally for $k = 0, 1, \ldots$ using a suitable solver

[BMC for discrete systems: Biere et al. 1999]
SAT-LP-Solver

\( \phi \)

Boolean abstraction

SAT-solver

\( \text{unsat} \rightarrow \text{UNSAT} \)

\( \text{sat} \)

(In)equation set

LP-solver

\( \text{unsat} \)

\( \text{sat} \rightarrow \text{SAT} \)
Memory requirements

Fischer’s protocol for 4 processes

heap peak [MB]

iteration

non-parametric

Herbstritt, Ábrahám, Becker, Steffen
Two kinds of learning:

- $B$-conflicts
  in boolean domain
- $R$-conflicts
  in real-valued domain

[Boolean conflict learning: Shtrichman 2001]
Learning: $B$-conflicts

Iteration $k$:

\[ I_0 \land T_{0,1} \land T_{1,2} \land \ldots \land T_{k-2,k-1} \land T_{k-1,k} \land \neg S_k \]

- boolean conflict $a(0) \land b(0)$
- boolean conflict $a(2) \land b(2)$
- boolean conflict $a(k) \land b(k)$

Iteration $k+1$:

\[ I_0 \land T_{0,1} \land T_{1,2} \land \ldots \land T_{k-2,k-1} \land T_{k-1,k} \land T_{k,k+1} \land \neg S_{k+1} \]

- boolean conflict $a(0) \land b(0)$
- boolean conflict $a(1) \land b(1)$
- boolean conflict $a(k) \land b(k)$
- boolean conflict $a(k+1) \land b(k+1)$
Learning: \( \mathcal{B} \)-conflicts

**Iteration k:**

\[
I_0 \land T_{0,1} \land T_{1,2} \land \ldots \land T_{k-2,k-1} \land T_{k-1,k} \land \neg S_k
\]

- boolean conflict \( a(0) \land b(0) \)
- boolean conflict \( a(2) \land b(2) \)
- boolean conflict \( a(k) \land b(k) \)

**Iteration k+1:**

\[
I_0 \land T_{0,1} \land T_{1,2} \land \ldots \land T_{k-2,k-1} \land T_{k-1,k} \land T_{k,k+1} \land \neg S_{k+1}
\]

- boolean conflict \( a(0) \land b(0) \)
- boolean conflict \( a(2) \land b(2) \)
- boolean conflict \( a(k+1) \land b(k+1) \)

\[
\text{boolean conflict } a(3) \land b(3)
\]
Learning: $R$-conflicts

Iteration $k$:

\[
I_0 \land T_{0,1} \land T_{1,2} \land \ldots \land T_{k-2,k-1} \land T_{k-1,k} \land \neg S_k
\]

- Real conflict: $x(1) > 3 \land x(1) < 0$
- Real conflict: $x(2) > 3 \land x(2) < 0$
- \ldots
- Real conflict: $x(k-1) > 3 \land x(k-1) < 0$

Iteration $k+1$:

\[
I_0 \land T_{0,1} \land T_{1,2} \land \ldots \land T_{k-2,k-1} \land T_{k-1,k} \land T_{k,k+1} \land \neg S_{k+1}
\]

- Real conflict: $x(1) > 3 \land x(1) < 0$
- Real conflict: $x(2) > 3 \land x(2) < 0$
- \ldots
- Real conflict: $x(k-1) > 3 \land x(k-1) < 0$
- Real conflict: $x(k) > 3 \land x(k) < 0$
Memory requirements

Fischer’s protocol for 4 processes

heap peak [MB]

iteration

non-parametric
Basic idea

- Two-Watch-Literal scheme (Chaff): need to watch/store only two literals instead of whole clause
- Overhead of learning: Non-Watch-Literals are also stored
- Solution: Parameterize variables and clauses and store only parameterized watch literals for (learned) clauses
Parametric solver structure: Variables

- Variables are represented by pairs \((a, i)\)
  - abstract id \(a\) identifies variable’s name
  - instance id \(i\) identifies variable’s state

  Example:
  - if variable \(x\) has abstract id 2, then
  - \(x_0\) is identified by \((2, 0)\), and
  - \(x_5\) is identified by \((2, 5)\).

- Clauses are also represented by pairs \((a, i)\)
  - index \(i\) is used as offset for instance id’s of the variables

  Example:
  - If the 7th (abstract) clause has literals \\{(5, 0), (8, 1)\}, then
  - \((7, 0)\) identifies the clause \\{(5, 0), (8, 1)\}, and
  - \((7, 2)\) identifies the clause \\{(5, 2), (8, 3)\}.

- Advantage: Easy conflict shifting
- Advantage: Reduced memory requirements
SAT-solver exploits Two-Watch-Literal scheme (Chaff)
Memory reduction by compact clause representation

Non-parametric clauses:

T1(1)

watches:

T1(2)

watches:

T1(k)

watches:

Parametric clauses:

T1

watches: 1:

T2

watches: 2:

... k:
Experimental Results: Fischer’s protocol for 3 processes

Memory requirements

heap peak [MB]

iteration

parametric
non-parametric

Memory-aware BMC of Linear Hybrid Systems
Experimental Results: Fischer’s protocol for 3 processes

- CPU times

- Iteration vs. cpu time [secs]
  - Parametric
  - Non-parametric

- Memory-aware BMC of Linear Hybrid Systems
Experimental Results: Fischer4, Railroad

Fischer’s protocol for 4 processes

- parametric
- non-parametric

Railroad Crossing

- parametric
- non-parametric
Conclusions

- Symmetry-based learning sets high memory requirements
- Parametric data structures
  - parameterize Two-Watch-Literal scheme
  - strongly reduce memory consumption
  - without slowing down the computation
Thank you for your attention!

Questions? Answers!