Bounded Model Checking with Parametric Data Structures

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

www.avacs.org

August 15 2006
4th International Workshop on Bounded Model Checking
Automated analysis of complex systems is mandatory
⇒ Especially safety-critical ones
⇒ Real-world scenarios embed *discrete* control in *continuous* environments
⇒ Modeling relies on hybrid automata
⇒ Bounded Model Checking for correctness analysis
Symmetry is inherent to Bounded Model Checking!
Idea

Symmetry is inherent to Bounded Model Checking!

Exploiting symmetry algorithmically.
Strichmann (FMSD 2004):
- Constraints replication
- Constraints sharing
Idea

Symmetry is inherent to Bounded Model Checking!

Exploiting symmetry *algorithmically*. Strichmann (FMSD 2004):
- Constraints replication
- Constraints sharing

Exploiting symmetry by *data structure*. Our proposal:
- Parametric variables
- Parametric clauses
- **Parametric watch literals**
Symmetry is inherent to Bounded Model Checking!

Exploiting symmetry *algorithmically*.
Strichmann (FMSD 2004):
- Constraints replication
- Constraints sharing

Reduces CPU time.

Exploiting symmetry by *data structure*.
Our proposal:
- Parametric variables
- Parametric clauses
  **Parametric watch literals**
Symmetry is inherent to Bounded Model Checking!

Exploiting symmetry *algorithmically.*
Strichmann (FMSD 2004):
- Constraints replication
- Constraints sharing

Reduces CPU time.

Exploiting symmetry by *data structure.*
Our proposal:
- Parametric variables
- Parametric clauses
- **Parametric watch literals**

Reduces memory.
Outline

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

\[
x = x_{\text{max}} \\
\dot{x} \leq 0 \\
x \geq x_{\text{min}}
\]

\[
x = x_{\text{min}} \\
\dot{x} \geq 0 \\
x \leq x_{\text{max}}
\]
Hybrid automaton (Fischer’s mutual exclusion protocol)

used for asynchronous distributed systems
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. (TACAS 1999)]
SAT-LP-Solver: Lazy SMT approach

\[
\phi \\
\text{Boolean abstraction} \\
\text{SAT-solver} \\
\text{(In)equation set} \\
\text{Explanation} \\
\text{LP-solver} \\
\text{UNSAT} \\
\text{SAT}
\]

[Ábrahám et al. (VMCAI 2005)]
Two kinds of learning:

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

[Boolean constraints sharing: Strichmann (CAV 2000)]
[Boolean constraints replication: Strichmann (CHARME 2001)]
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 $a(1) \land b(1)$
- boolean conflict $a(k+1) \land b(k+1)$
Learning: $B$-Conflicts

Constraints harvesting

Combining *constraints sharing* and *constraints replication*: Constraint is added in *next* BMC iteration, shifted by 1 time frame only.

**Iteration k:**

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

**Iteration k+1:**

\[
I_0 \land T_{0,1} \land T_{1,2} \land \ldots \land T_{k-2,k-1} \land T_{k-1,k} \land T_{k,k+1} \land \neg S_{k+1}
\]
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 $x(2)>3 \land x(2)<0$
- ... real conflict $x(k-1)>3 \land x(k-1)<0$
- real conflict $x(k)>3 \land x(k)<0$

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 $x(2)>3 \land x(2)<0$
- ... real conflict $x(k-1)>3 \land x(k-1)<0$
- real conflict $x(k)>3 \land x(k)<0$
Memory requirements

Is memory really a limiting factor?
Memory requirements

Is memory really a limiting factor?

- Our experience: Yes!
- Learning decreases CPU time, but increases memory consumption.
Is memory really a limiting factor?

- Our experience: Yes!
- Learning decreases CPU time, but increases memory consumption.

Other approaches for reducing memory:
- Ganai et al. (CHARME 2003): Distributed SAT, proprietary benchmarks
- Dershowitz et al. (SAT 2005): QBF formulation of BMC, proprietary benchmarks
Basic idea

- Two-Watch-Literal scheme (Moskewicz et al., DAC 2001): Need to watch only two literals instead of whole clause
- Overhead of learning: Non-Watch-Literals are also stored
- Solution:
  - Parameterize variables and clauses
  - 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 time frame
- 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:

- \( T_1(1) \)
  - watches: \( \uparrow \uparrow \)

- \( T_1(2) \)
  - watches: \( \uparrow \uparrow \)

- \( T_1(k) \)
  - watches: \( \uparrow \uparrow \)

Parametric clauses:

- \( T_1 \)
  - watches: 1: \( \uparrow \uparrow \)
  - 2: \( \uparrow \uparrow \)
  - \( \ldots \)
  - k: \( \uparrow \uparrow \)

- \( T_2 \)
  - watches: \( \uparrow \uparrow \)
Experimental Results: Tcp (discrete system)
Experimental Results: Fischer’s protocol for 3 processes

Memory requirements

- **Parametric**
- **Non-parametric**

 ![Graph showing memory requirements over iterations for parametric and non-parametric Fischer's protocol for 3 processes.](image)
Experimental Results: Fischer’s protocol for 3 processes

CPU times

- parametric
- non-parametric

CPU time [secs]

iteration
Experimental Results: Fischer4, Railroad

Fischer’s protocol for 4 processes

- heap peak [MB] vs iteration
- parametric and non-parametric curves

Railroad Crossing

- heap peak [MB] vs iteration
- parametric and non-parametric curves
Conclusion

- 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!