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

Context

⇒ 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

Idea

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**

Idea

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**

Idea

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:

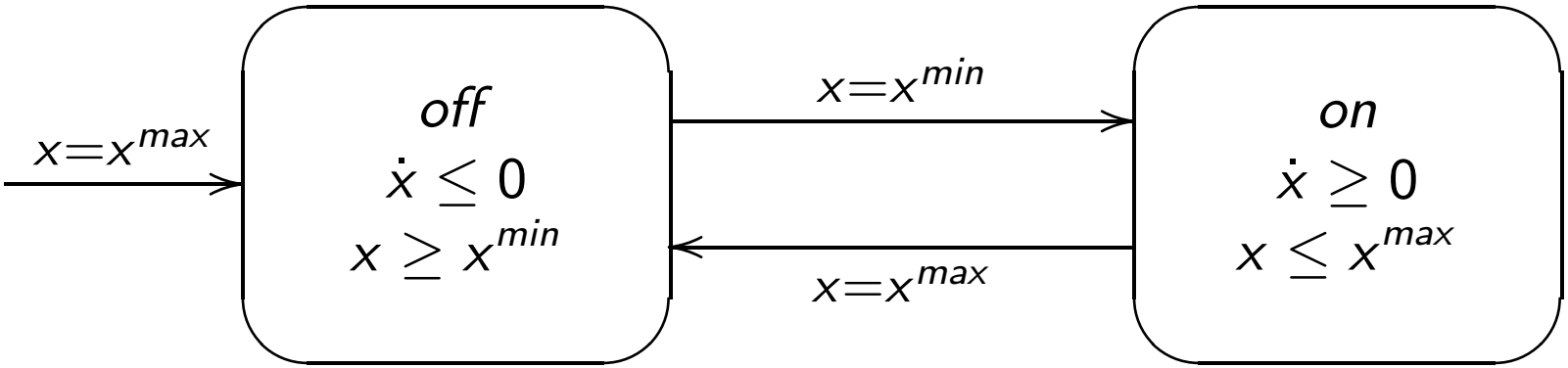
- Parametric variables
- Parametric clauses
- **Parametric watch literals**

Reduces memory.

Outline

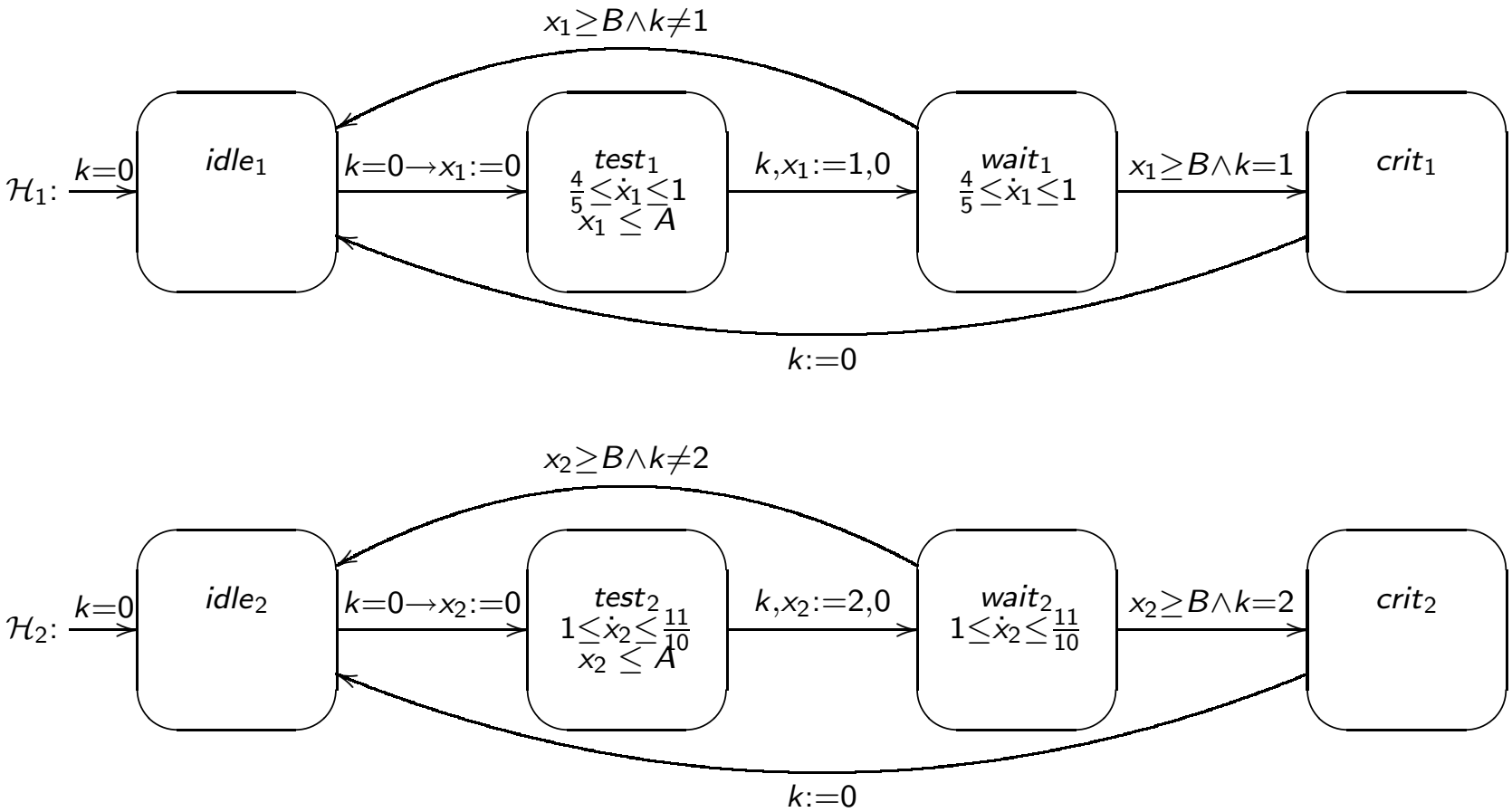
- Bounded Model Checking for Linear Hybrid Automata
- Symmetry-based Learning
  $\Rightarrow$ Leads to high memory requirements
- Memory-aware storage
  $\Rightarrow$ Parametric data structures
- Experimental results
- Conclusions

Hybrid automaton (Thermostat controller)



Hybrid automaton (Fischer's mutual exclusion protocol)

used for asynchronous distributed systems



BMC of Linear Hybrid Automata

- Counterexamples of length k described by first-order logic formulas over $(\mathbb{R}, +, <, 0, 1)$:

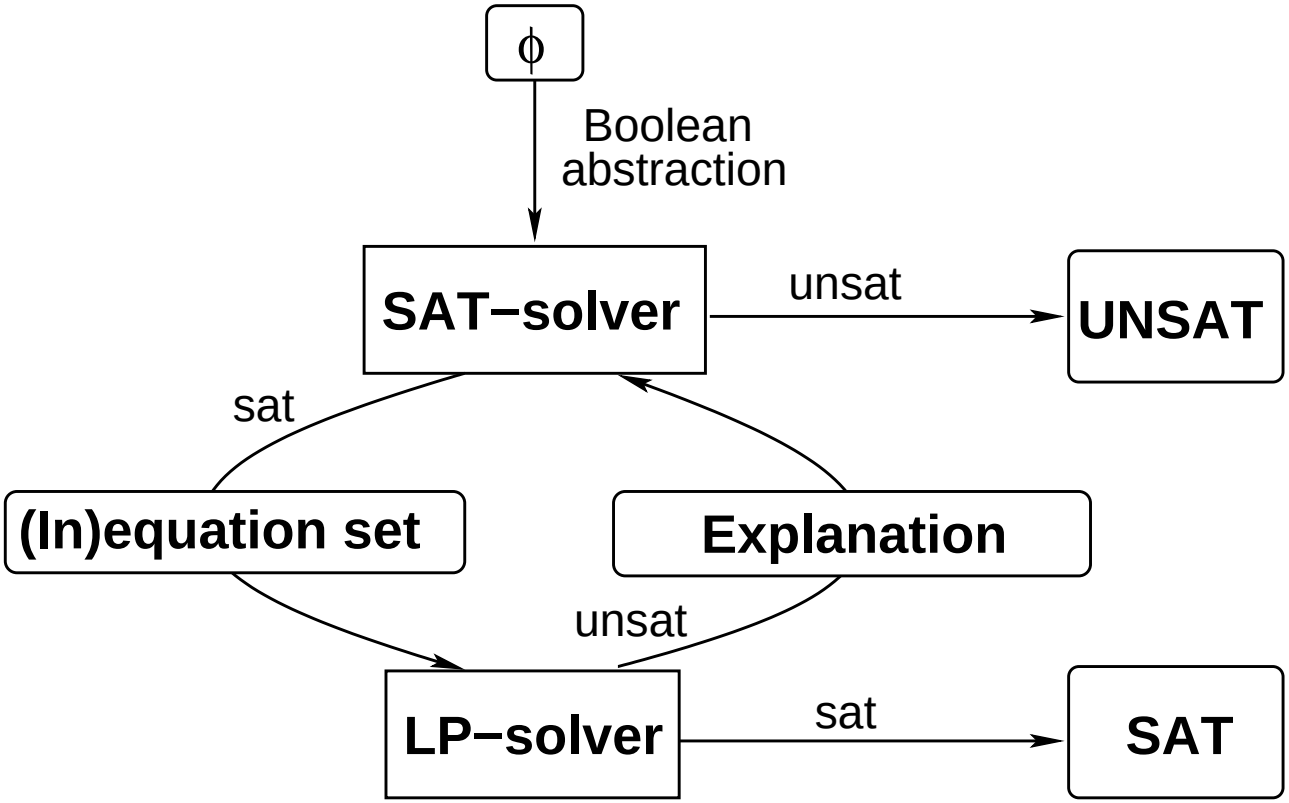
$$\varphi_k(s_0, \ldots, s_k) :$$
$$\mathsf{Init}(s_0) \wedge \mathsf{Trans}(s_0, s_1) \wedge \ldots \wedge \mathsf{Trans}(s_{k-1}, s_k) \wedge \mathsf{Bad}(s_k)$$

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

$\Rightarrow$ Check φ_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

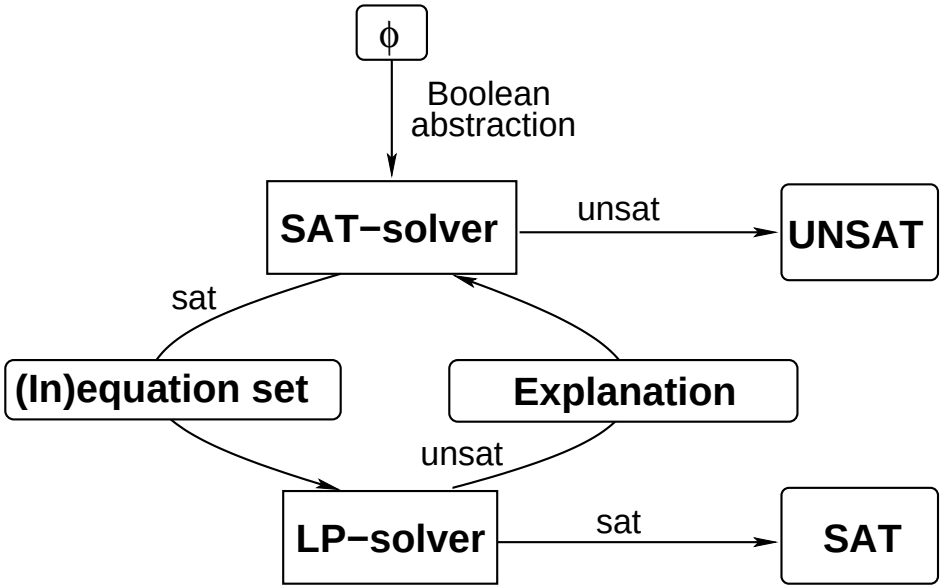


[Ábrahám et al. (VMCAI 2005)]

Learning

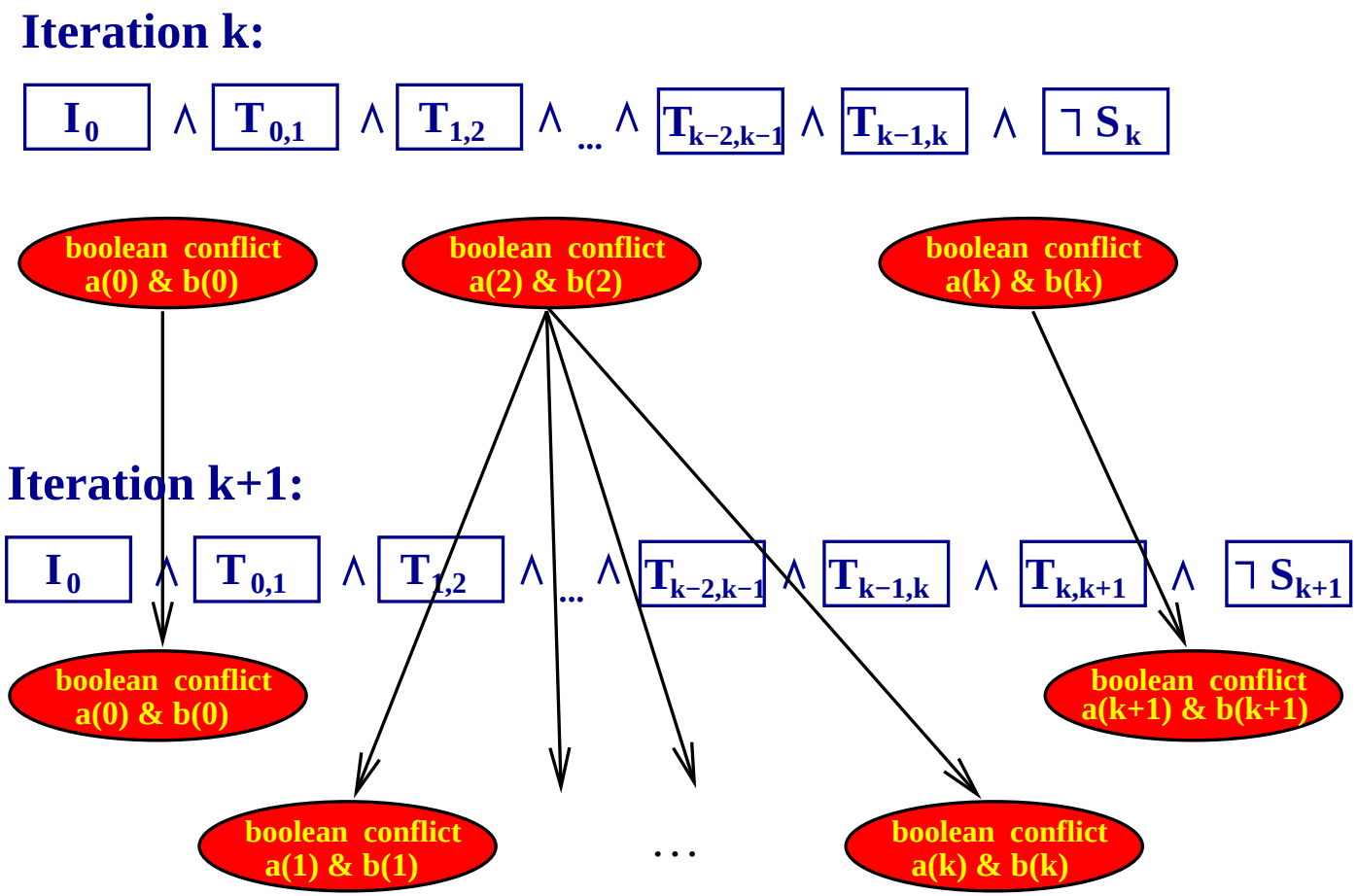
Two kinds of learning:

- $\mathcal{B}$-conflicts
  in boolean domain
- $\mathcal{R}$-conflicts
  in real-valued domain



[Boolean constraints sharing: Strichmann (CAV 2000)]
[Boolean constraints replication: Strichmann (CHARME 2001)]

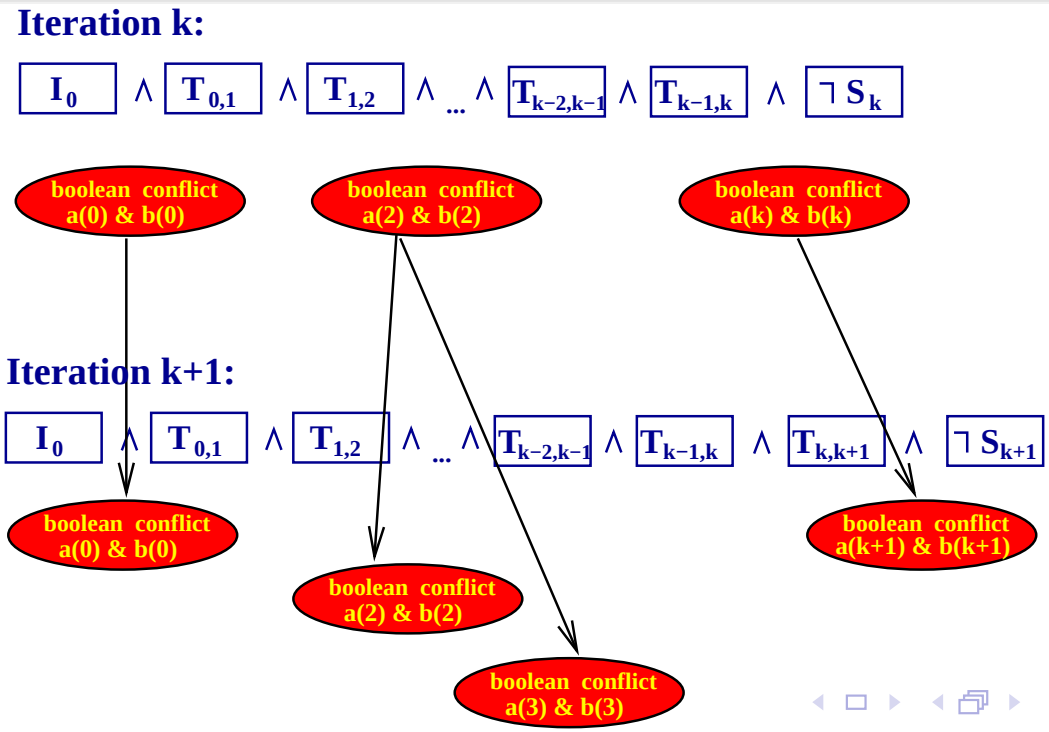
Learning: $\mathcal{B}$-Conflicts



Learning: $\mathcal{B}$-Conflicts

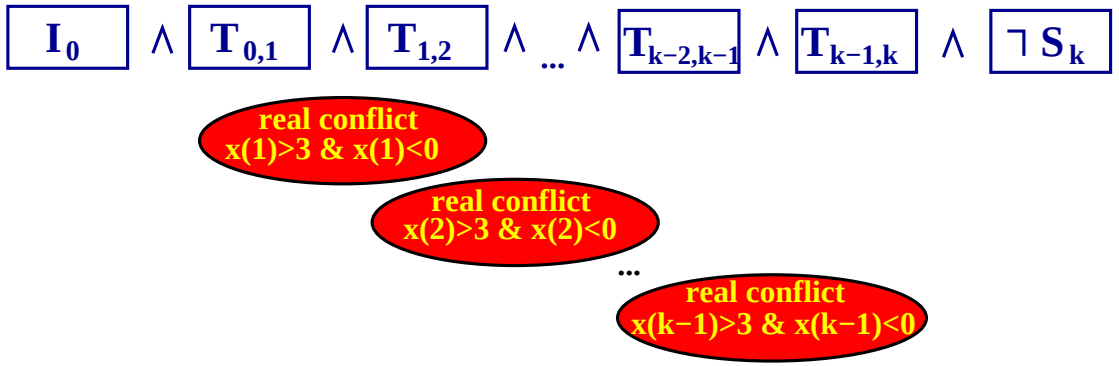
Constraints harvesting

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

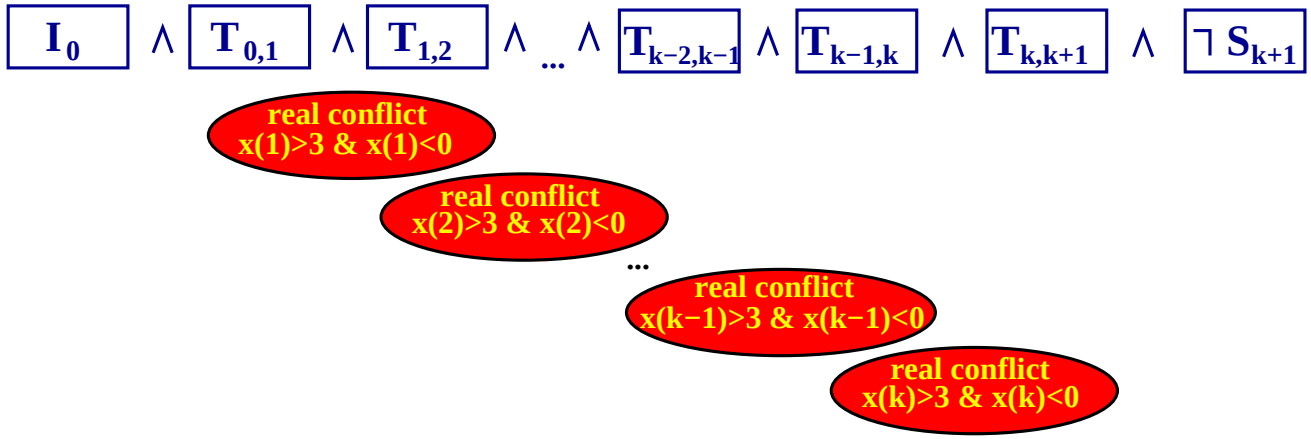


Learning: $\mathcal{R}$-Conflicts

Iteration k:



Iteration k+1:



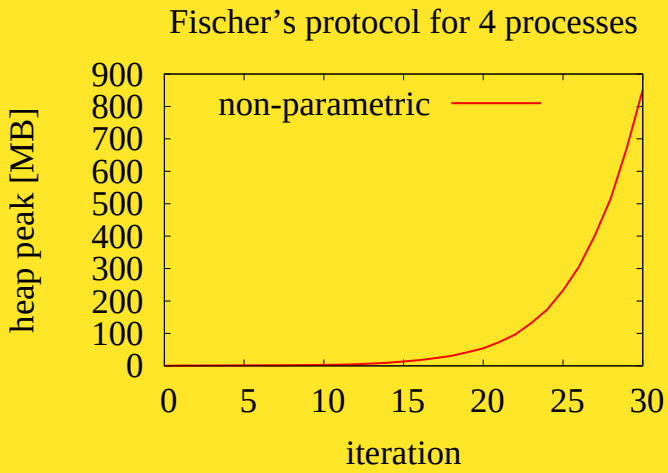
Memory requirements

Is memory really a limiting factor?

Memory requirements

Is memory really a limiting factor?

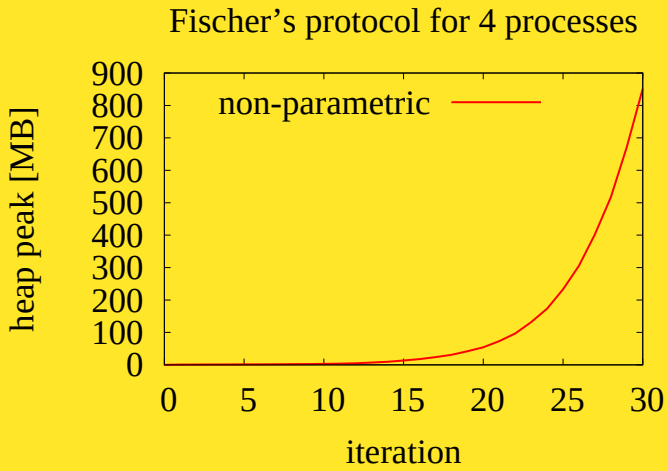
- Our experience: Yes!
- Learning *decreases* CPU time, but *increases* memory consumption.



Memory requirements

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*

Parametric solver structure

Basic idea

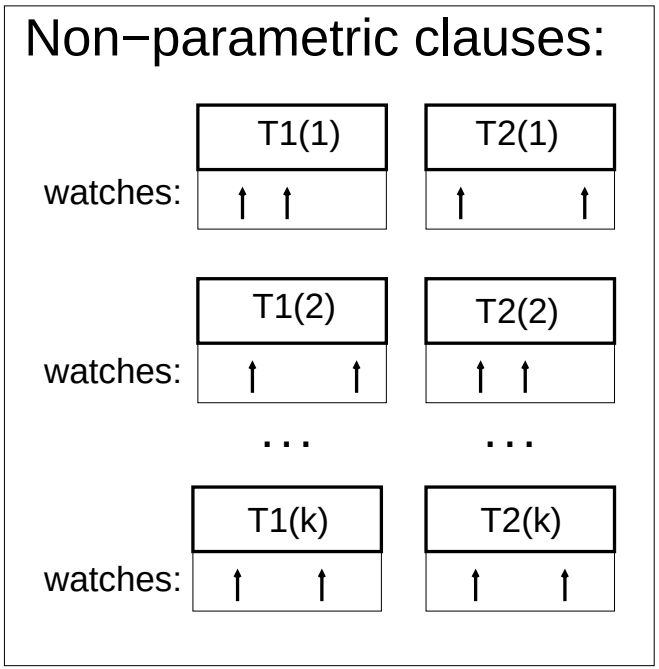
- Two-Watch-Literal scheme (Moskewicz et al., DAC 2001): Need to watch only two literals instead of whole clause
- Overhead of learning: Non-Watch-Literals are also stored
- Solution:
  - Parameterize variables and clauses
  - Store only parameterized watch literals for (learned) clauses

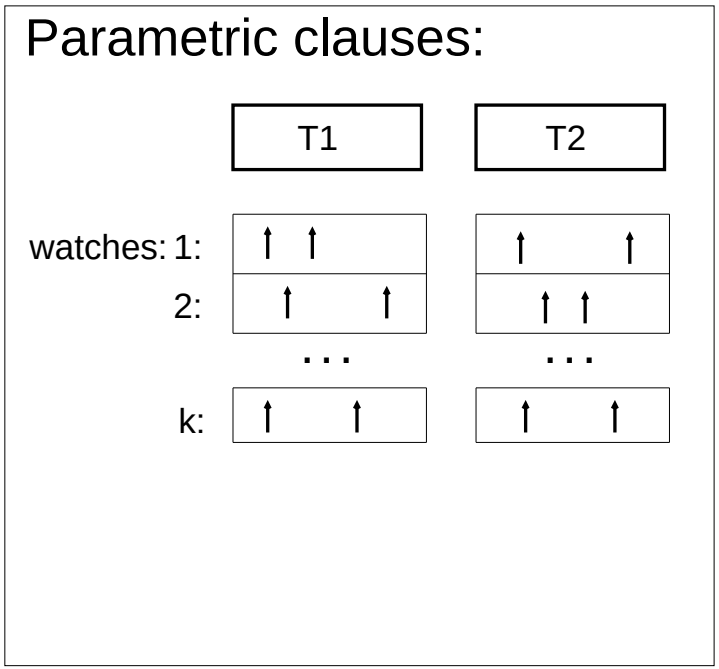
Parametric solver structure: Variables

- 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

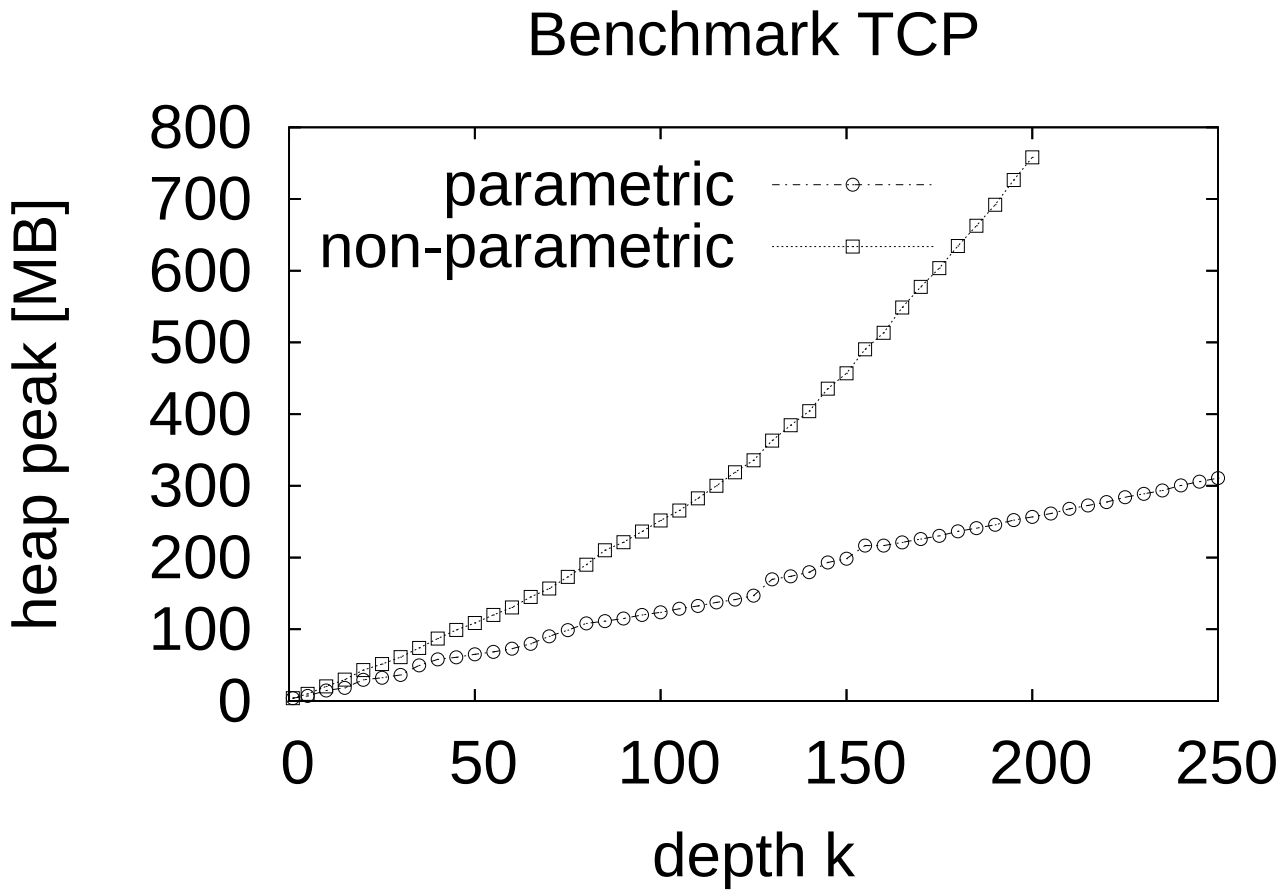
Parametric solver structure: Two-Watch-Literal Scheme

- SAT-solver exploits Two-Watch-Literal scheme (Chaff)
- Memory reduction by compact clause representation

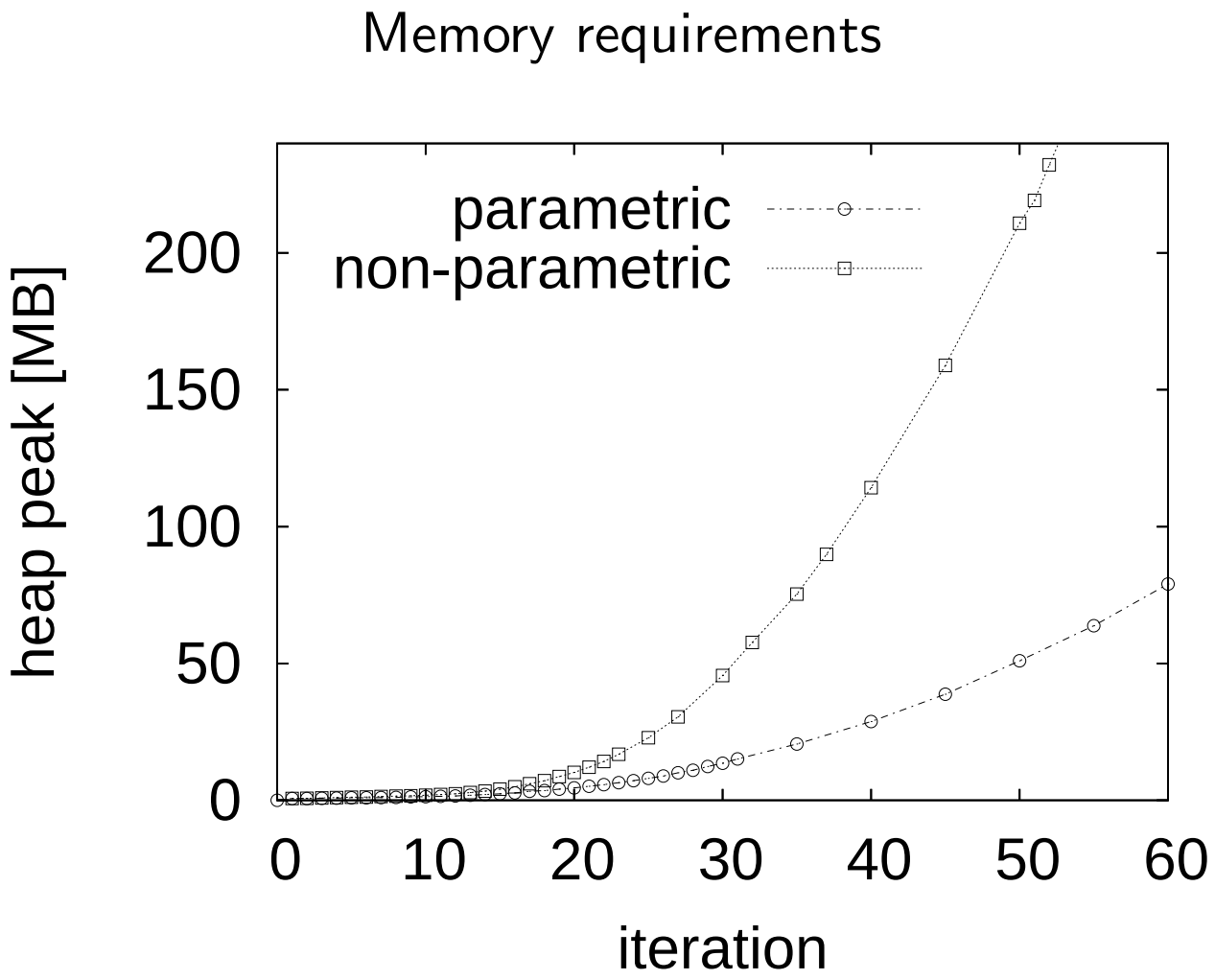




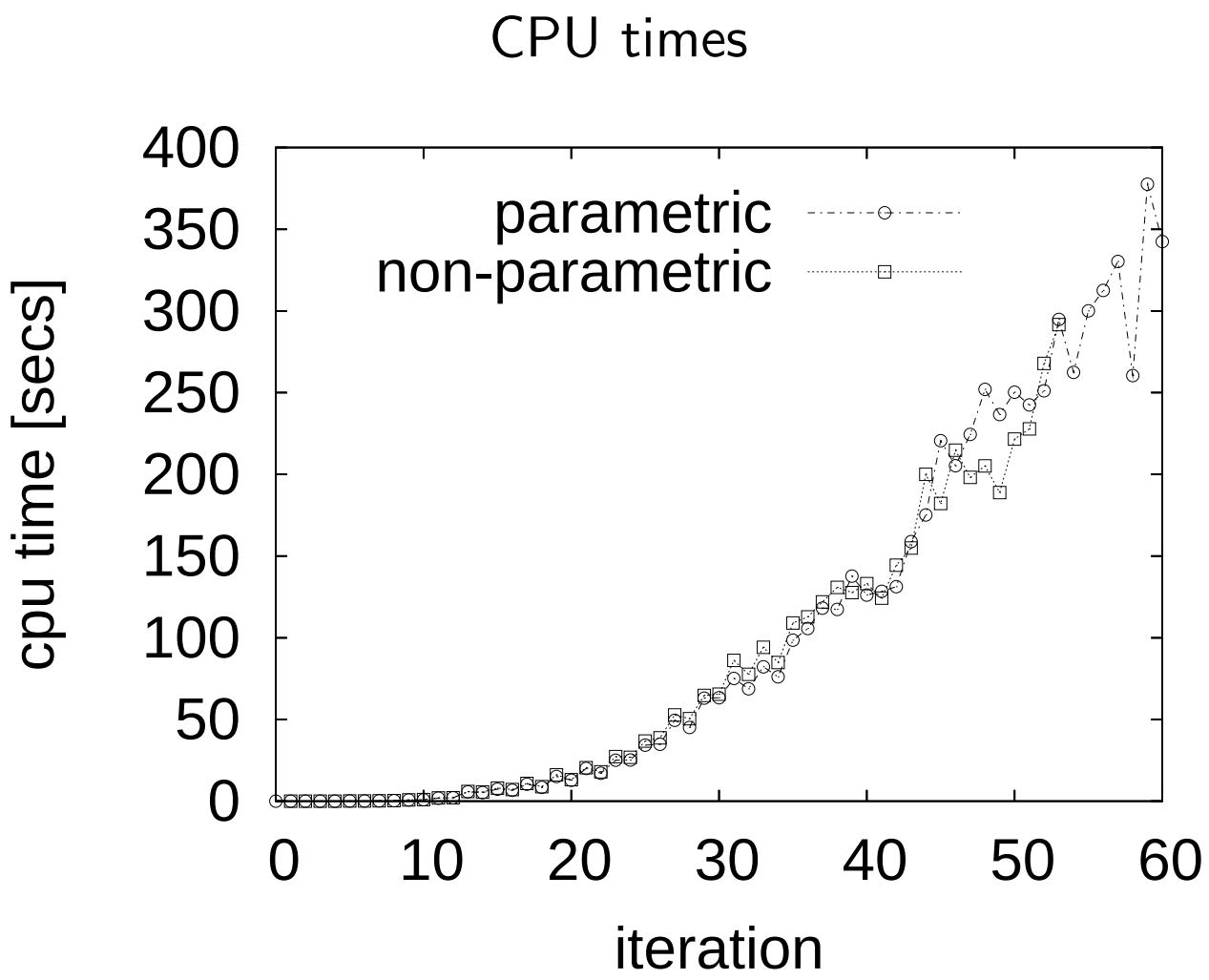
Experimental Results: Tcp (discrete system)



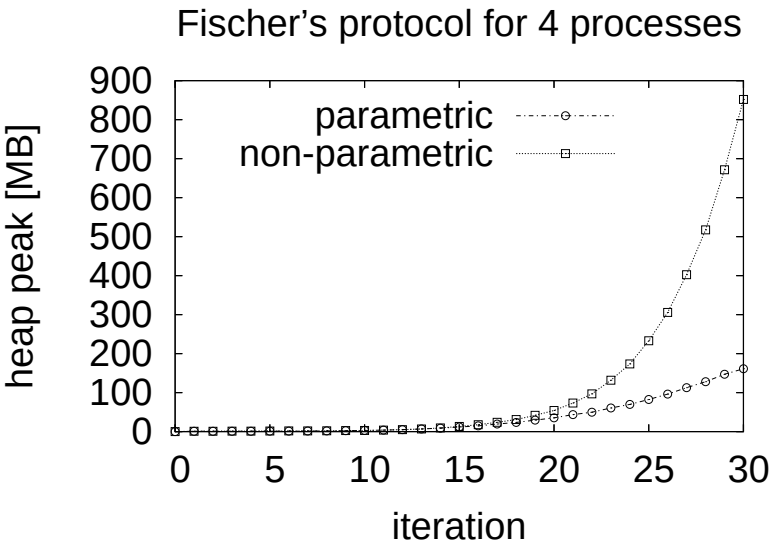
Experimental Results: Fischer's protocol for 3 processes

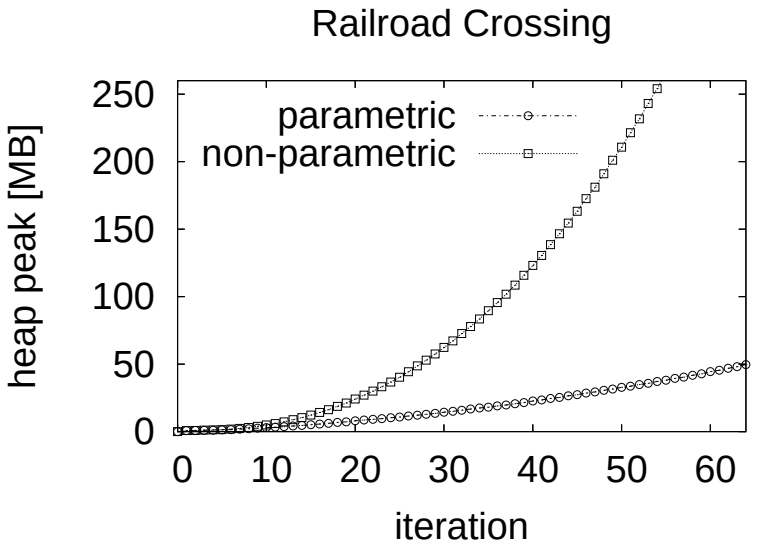


Experimental Results: Fischer's protocol for 3 processes



Experimental Results: Fischer4, Railroad





Conclusion

- Symmetry-based learning sets high memory requirements
- Parametric data structures

  $\Rightarrow$ parameterize Two-Watch-Literal scheme

  $\Rightarrow$ strongly reduce memory consumption

  $\Rightarrow$ without slowing down the computation

Questions and Answers

Thank you for your attention!

Questions? Answers!