Space Effective Model Checking for Component-Interaction Automata

Nikola Beneš  Milan Křivánek  Filip Štefaňák

ParaDiSe Laboratory
Faculty of Informatics
Masaryk University

MEMICS ’09
Motivation

Component-based system development

- autonomous third-party components
- component interaction – critical correctness issue
Motivation

Component-based system development
- autonomous third-party components
- component interaction – critical correctness issue

Component-Interaction automata
- automata-based formalism
- intended for automated verification (model checking)
Motivation

Component-based system development
- autonomous third-party components
- component interaction – critical correctness issue

Component-Interaction automata
- automata-based formalism
- intended for automated verification (model checking)

Problem with automated verification
- exponentially large state space w.r.t. number of components
- solution: reduction methods

Our aim: Partial order reduction for Component-Interaction automata
three types of actions (input, output, internal) via structured labels
- capture important interaction information
- flexible composition preserving information about internal labels
- hierarchical structure
 Specification Logic

State/Event LTL

- extension of the classical Linear Temporal Logic
- properties of infinite runs
- allows propositions about both states and actions
- in our setting: enabledness properties of states
Specification Logic

State/Event LTL

- extension of the classical Linear Temporal Logic
- properties of infinite runs
- allows propositions about both states and actions
- in our setting: enabledness properties of states

Formulae

- possible interaction $E(label)$
- actual interaction $P(label)$
- LTL operators (next $X$, until $U$, propositional and derived operators)
Specification Logic

State/Event LTL

- extension of the classical Linear Temporal Logic
- properties of infinite runs
- allows propositions about both states and actions
- in our setting: enabledness properties of states

Formulae

- possible interaction $E(label)$
- actual interaction $P(label)$
- LTL operators (next $X$, until $U$, propositional and derived operators)

Example: Whenever $(1, a, 2)$ is enabled, it is eventually also performed.

$$G(E(1, a, 2) \Rightarrow F P(1, a, 2))$$
Partial Order Reduction

- Exploits commutativity of concurrent transitions.
- Stuttering equivalence
- At least one run of each equivalence class is preserved.
- LTL–X properties are the same.
Partial Order Reduction

- Exploits commutativity of concurrent transitions.
- Stuttering equivalence
- At least one run of each equivalence class is preserved.
- LTL–X properties are the same.

Example:
- Two concurrent processes
  
  \((\text{inc } x; \text{inc } x) \parallel (\text{inc } y; \text{inc } y)\)
- Interested only in value of \(x\)
- \text{inc } x, \text{inc } y \text{ are independent}
- \text{inc } y \text{ is invisible}
Partial Order Reduction

- Exploits commutativity of concurrent transitions.
- Stuttering equivalence
- At least one run of each equivalence class is preserved.
- LTL–X properties are the same.

Example:
- Two concurrent processes $(\text{inc } x; \text{inc } x) \parallel (\text{inc } y; \text{inc } y)$
- Interested only in value of $x$
- $\text{inc } x, \text{inc } y$ are independent
- $\text{inc } y$ is invisible
Partial Order Reduction

- Exploits commutativity of concurrent transitions.
- Stuttering equivalence
- At least one run of each equivalence class is preserved.
- LTL–X properties are the same.

Example:
- Two concurrent processes 
  \((\text{inc } x; \text{inc } x) \parallel (\text{inc } y; \text{inc } y)\)
- Interested only in value of \(x\)
- \(\text{inc } x, \text{inc } y\) are independent
- \(\text{inc } y\) is invisible
Partial Order Reduction

- Exploits commutativity of concurrent transitions.
- Stuttering equivalence
- At least one run of each equivalence class is preserved.
- LTL–X properties are the same.

**State/Event LTL**
- Different stuttering equivalence
- Different logic of preserved properties (weak SE-LTL)
- The method remains the same

Partial Order Reduction – Ample Set Method

The ample set
- a subset of enabled actions from a given state
The ample set

- a subset of enabled actions from a given state
- the reduction is correct if the following conditions hold:

The ample set conditions

**C0** (nonemptiness) Some actions must be enabled.

**C1** (dependency) There is no dependent action in the original system that could be performed before an action from the ample set.

**C2** (invisibility) All actions from the ample set are invisible.

**C3** (cycle) No action can be put off forever.
The ample set

- a subset of enabled actions from a given state
- the reduction is correct if the following conditions hold:

The ample set conditions

- **C0** (*nonemptiness*) Some actions must be enabled.
- **C1** (*dependency*) There is no dependent action in the original system that could be performed before an action from the ample set.
- **C2** (*invisibility*) All actions from the ample set are invisible.
- **C3** (*cycle*) No action can be put off forever.

- In reality, these are very difficult to check.
- Overapproximations of these conditions are used.
Exploiting the hierarchical tree structure.
Exploiting the hierarchical tree structure.
To get less candidates, actions of whole subtrees are considered.
Actions within a single composition are more likely to be dependent.
Selecting Candidates for Ample Sets

- Exploiting the hierarchical tree structure.
- To get less candidates, actions of whole subtrees are considered.
- Actions within a single composition are more likely to be dependent.
- Simple automata are checked first, as they make up smaller sets.
We can’t choose C for ample set if it synchronizes with some other automaton. Solution: We choose D, if C synchronizes only with B, which does all communication with other automata, but has no enabled actions. All simple automata synchronize in good systems. Consequence: Not natural, we require weaker condition.
We can’t choose C for ample set if it synchronizes with some other automaton.
We can’t choose C for ample set if it synchronizes with some other automaton.

Solution: We choose D, if C synchronizes only with B, which does all communication with other automata, but has no enabled actions.
We can’t choose C for ample set if it synchronizes with some other automaton.

Solution: We choose D, if C synchronizes only with B, which does all communication with other automata, but has no enabled actions.

All simple automata synchronize in good systems.

Consequence: Not natural, we require weaker condition.
We consider only synchronizations, that simple automata could do from their current state.

Synchronizations that do not affect the situation are not considered and we can choose directly the candidate with good behaviour.

Proven stronger than the dependency condition.
Invisibility and Cycle Conditions

**invisibility**

- All labels that are used in the formula are visible.
- Straightforward for action labels.
- Synchronization affects state enabledness labels.
- Closure of interesting labels: We add partial transitions of sync transitions.
Invisibility and Cycle Conditions

invisibility

- All labels that are used in the formula are visible.
- Straightforward for action labels.
- Synchronization affects state enabledness labels.
- Closure of interesting labels: We add partial transitions of sync transitions.

cycle

- A transition in ample set shouldn’t complete a cycle.
- Cycles are detected by states already in DFS stack.
- safe overapproximation, standard method
nonemptiness

- Trivial: there must be an enabled transition.
Implementation of Ample Set Conditions

nonemptiness

- Trivial: there must be an enabled transition.

invisibility and cycle

- Invalidated by some enabled transition of the set, which is also in all ancestors.
- Checked for each enabled transition of the system instead of sets.
Implementation of Ample Set Conditions

nonemptiness
- Trivial: there must be an enabled transition.

invisibility and cycle
- Invalidated by some enabled transition of the set, which is also in all ancestors.
- Checked for each enabled transition of the system instead of sets.

dependency
- Checking is more difficult, is attempted only if other conditions hold
- Looking for input / output pairs with the same name, one from current state of an automaton in the candidate set and the second one from an automaton not in the set.
- Matching couple is a counterexample for the set.
### Case Study

<table>
<thead>
<tr>
<th>Model Property</th>
<th>without POR</th>
<th>with POR</th>
<th>reduction ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># states</td>
<td>RAM (MB)</td>
<td>time (s)</td>
</tr>
<tr>
<td>C $\varphi_1$</td>
<td>749 340</td>
<td>139</td>
<td>498</td>
</tr>
<tr>
<td>C $\varphi_2$</td>
<td>1 498 679</td>
<td>274</td>
<td>1 010</td>
</tr>
<tr>
<td>SC $\varphi_1$</td>
<td>29 341</td>
<td>9</td>
<td>19</td>
</tr>
<tr>
<td>SC $\varphi_2$</td>
<td>58 681</td>
<td>15</td>
<td>39</td>
</tr>
<tr>
<td>SCM $\varphi_1$</td>
<td>22 745 391</td>
<td>4 045</td>
<td>21 656</td>
</tr>
<tr>
<td>SCM $\varphi_2$</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SCR $\varphi_1$</td>
<td>2 994 016</td>
<td>570</td>
<td>2 119</td>
</tr>
<tr>
<td>SCR $\varphi_2$</td>
<td>5 988 032</td>
<td>1 128</td>
<td>4 631</td>
</tr>
<tr>
<td>SCSM $\varphi_1$</td>
<td>6 369 598</td>
<td>1 135</td>
<td>4 692</td>
</tr>
<tr>
<td>SCSM $\varphi_2$</td>
<td>12 739 195</td>
<td>2 263</td>
<td>10 434</td>
</tr>
<tr>
<td>TSC $\varphi_1$</td>
<td>1 356 277</td>
<td>245</td>
<td>934</td>
</tr>
<tr>
<td>TSC $\varphi_2$</td>
<td>2 712 553</td>
<td>484</td>
<td>1 936</td>
</tr>
</tbody>
</table>

### Verification framework for CI automata

- **Coln tool**
### Case Study

**Model of a large trading system.**

140 simple automata, hierarchically composed (up to 6 levels of depth, with lot of synchronization.

**Effective implementation: saves space and time.**

<table>
<thead>
<tr>
<th>Model Property</th>
<th>without POR</th>
<th>with POR</th>
<th>reduction ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># states</td>
<td>RAM (MB)</td>
<td>time (s)</td>
</tr>
<tr>
<td>C $\varphi_1$</td>
<td>749,340</td>
<td>139</td>
<td>498</td>
</tr>
<tr>
<td>C $\varphi_2$</td>
<td>1,498,679</td>
<td>274</td>
<td>1,010</td>
</tr>
<tr>
<td>SC $\varphi_1$</td>
<td>29,341</td>
<td>9</td>
<td>19</td>
</tr>
<tr>
<td>SC $\varphi_2$</td>
<td>58,681</td>
<td>15</td>
<td>39</td>
</tr>
<tr>
<td>SCM $\varphi_1$</td>
<td>22,745,391</td>
<td>4,045</td>
<td>21,656</td>
</tr>
<tr>
<td>SCM $\varphi_2$</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SCR $\varphi_1$</td>
<td>2,994,016</td>
<td>570</td>
<td>2,119</td>
</tr>
<tr>
<td>SCR $\varphi_2$</td>
<td>5,988,032</td>
<td>1,128</td>
<td>4,631</td>
</tr>
<tr>
<td>SCSM $\varphi_1$</td>
<td>6,369,598</td>
<td>1,135</td>
<td>4,692</td>
</tr>
<tr>
<td>SCSM $\varphi_2$</td>
<td>12,739,195</td>
<td>2,263</td>
<td>10,434</td>
</tr>
<tr>
<td>TSC $\varphi_1$</td>
<td>1,356,277</td>
<td>245</td>
<td>934</td>
</tr>
<tr>
<td>TSC $\varphi_2$</td>
<td>2,712,553</td>
<td>484</td>
<td>1,936</td>
</tr>
</tbody>
</table>
Conclusion

General method – partial order reduction

Specific setting – Component-Interaction automata
  - state/event-based formalism (enabledness)
  - prevalence of synchronization actions
  - hierarchical structure

Ample set heuristics
  - different heuristic for dependency condition (C1)
  - overapproximation for visibility condition (C2)
  - (C0) and (C3) remain the same

Implementation in CoIn tool and experimental evaluation