Combining Partial Order Reduction with Bounded Model Checking

CPA 2009
José Vander Meulen and Charles Pecheur

UC Louvain
A Concurrent System

- Set of asynchronous and interacting processes

Can we verify this system with Symbolic Model Checking?

Up to what $q$?
Model Checking

- **Exhaustive** exploration of the state space of a system
Symbolic Model Checking

- Principle:
  - Compute **sets of states** (BDDs), or
  - Resolve a **SAT** problem (BMC)

- Brilliant results in the hardware domain
  [Biere + 03, Mc Millan 93]

- Conventional wisdom: Symbolic Model Checking methods are not well suited for asynchronous systems.

- How can we use symbolic Model Checking with asynchronous system?
Outline

- Background
  - Bounded Model Checking
  - Partial Order Reduction
- Combining Partial Order Reduction with Bounded Model Checking
- Experimental results
- Conclusion
- Perspectives
Bounded Model Checking [Biere + 99]

- Search for a counterexample in executions whose length $= k$
- e.g. paths of length 3
Bounded Model Checking [Biere + 99]

- Reduce model checking problem to a SAT problem
- Unfold the transition relation $k$ times to obtain a boolean formula $[M]_k$
  \[
  I(\vec{x}_0) \land T(\vec{x}_0, \vec{x}_1) \land T(\vec{x}_1, \vec{x}_2) \land \cdots \land T(\vec{x}_{k-1}, \vec{x}_k)
  \]
- Translate the negation of a LTL property $f$ to a Boolean formula $[\neg f]_k$
- If $[M]_k \land [\neg f]_k$ is satisfiable, an error is found
Partial Order Reduction

- **Partial order reduction** methods are best suited for asynchronous systems
  - Can we use these methods with BMC and LTL?
- **Verification** = only check some interleavings of a transition system
- Based on **independence** between transitions and **invisibility** of a transition
Partial Order Reduction

- Partial order reduction methods are best suited for asynchronous systems
  - Can we use these methods with BMC and LTL?
- Verification = only check some interleavings of a transition system

- Based on independence between transitions and invisibility of a transition
Partial Order Reduction

- **Algorithm**: modified depth-first search (DFS)
  - At each step $s$, a subset of the successors is selected: $ample(s)$
  - $ample(s)$ has to respect a set of conditions
- **c1**: Along every path in the full state graph that starts at $s$: a transition that is dependent on a transition in $ample(s)$ cannot be executed without a transition in $ample(s)$ occurring first.
Partial Order Reduction

- **c2** at least one state $s$ per cycle is fully expanded
- **c3** If $ample(s) \neq enable(s)$, all transitions in $ample(s)$ are invisible.
- **c4** if $ample(s) \neq enable(s)$, then $ample(s)$ is a singleton
  - **C1 – C3** preserve deadlocks, $LTL_X$ properties
  - **C1 – C4** preserve $CTL_X$ properties
**Two-phase algorithm** [Nalumasu + 97]

- A modified DFS: performs alternatively 2 phases
  - Phase-1: explore for each process as many safe transitions (C1, C4) as possible
  - Phase-2: fully expand the current state

Two-phase algorithm can check $CTL_X$ properties
SBTP

- Algorithm combining POR with BMC:
  - SBTP: Phase-1 performs a fixed number $n$ of partial expansions for each process
  - A process might not be able to produce $n$ safe transitions (idle transitions)
SBTP

- From a transition system to a computation tree

$M$ and $CT(M)$ are equivalent
- A modified computation tree \((\approx CT(M))\)
- Given \(p\) processes, a fixed number \(n\) of partial expansions, construct a reduced computation tree.
  - e.g number of processes \(p = 2\), and \(n = 3\)
Given $p$ processes, a fixed number $n$ of partial expansions, and $k = m(p \times n + 1)$, apply $m$ times the two phases to obtain $[[M]]^{SBTP}_{k,n}$

- e.g number of processes $p = 2$, and $n = 3$

![Diagram of states and transitions]

- Translate the negation of a $LTL_X$ property $f$ to a boolean formula $[\neg f]_k$

- If $[[M]]^{SBTP}_{k,n} \land [\neg f]_k$ is satisfiable, an error is found
There exists $k \geq 0$ such that $[M, \neg f]_{SBTP}^{k, n}$ if and only if $M \nvDash f$

Our method finds a true assignment satisfying $\neg f$

$\iff$

Classical BMC on $SBTP(M, n)$ finds a true assignment satisfying $\neg f$

$\iff$

$SBTP(M, n)$ does not satisfy $f$

$\iff$

$M$ does not satisfy $f$
Tool

- Implemented in Scala:
  - Smoothly integrates features of object-oriented and functional languages.
  - Fully interoperable with Java.
- SAT part uses the Yices SMT solver.
- Main Features:
  - Modelling language based on processes and synchronization by rendezvous
  - BMC of LTL properties
  - SBTP of LTL$_X$ properties
Case Study: Producer-Consumer

- A variant of the Producer-Consumer problem:
  - with $q$ producers, $q$ consumers, and $n = 8$
  - $P_2$: in all cases the buffer will eventually contain more than one piece

<table>
<thead>
<tr>
<th>$q$</th>
<th>states</th>
<th>$k$</th>
<th>sec</th>
<th>$k$</th>
<th>cycles</th>
<th>sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1,059</td>
<td>26</td>
<td>73</td>
<td>153</td>
<td>9</td>
<td>122</td>
</tr>
<tr>
<td>2</td>
<td>51,859</td>
<td>44</td>
<td>29,898</td>
<td>297</td>
<td>9</td>
<td>211</td>
</tr>
<tr>
<td>3</td>
<td>3,807,747</td>
<td>—</td>
<td>—</td>
<td>441</td>
<td>9</td>
<td>401</td>
</tr>
<tr>
<td>4</td>
<td>$\approx 10^8$</td>
<td>—</td>
<td>—</td>
<td>585</td>
<td>9</td>
<td>1,238</td>
</tr>
<tr>
<td>5</td>
<td>$\approx 10^{10}$</td>
<td>—</td>
<td>—</td>
<td>729</td>
<td>9</td>
<td>1,338</td>
</tr>
<tr>
<td>6</td>
<td>$\approx 10^{12}$</td>
<td>—</td>
<td>—</td>
<td>873</td>
<td>9</td>
<td>1,926</td>
</tr>
<tr>
<td>7</td>
<td>$\approx 10^{14}$</td>
<td>—</td>
<td>—</td>
<td>1,017</td>
<td>9</td>
<td>4,135</td>
</tr>
</tbody>
</table>
Case Study: Producer-Consumer

- Influence of the parameter $n$ when the number of producers (resp. consumers) = 2

<table>
<thead>
<tr>
<th>$n$</th>
<th>$k$</th>
<th># cycles</th>
<th>TIME (sec)</th>
<th>MEM (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>44</td>
<td>44</td>
<td>29,898</td>
<td>131</td>
</tr>
<tr>
<td>1</td>
<td>95</td>
<td>19</td>
<td>855</td>
<td>159</td>
</tr>
<tr>
<td>2</td>
<td>135</td>
<td>15</td>
<td>235</td>
<td>167</td>
</tr>
<tr>
<td>3</td>
<td>169</td>
<td>13</td>
<td>305</td>
<td>194</td>
</tr>
<tr>
<td>4</td>
<td>187</td>
<td>11</td>
<td>217</td>
<td>192</td>
</tr>
<tr>
<td>5</td>
<td>231</td>
<td>11</td>
<td>375</td>
<td>308</td>
</tr>
<tr>
<td>6</td>
<td>275</td>
<td>11</td>
<td>381</td>
<td>240</td>
</tr>
<tr>
<td>7</td>
<td>319</td>
<td>11</td>
<td>583</td>
<td>318</td>
</tr>
<tr>
<td>8</td>
<td>297</td>
<td>9</td>
<td>211</td>
<td>224</td>
</tr>
<tr>
<td>9</td>
<td>333</td>
<td>9</td>
<td>240</td>
<td>295</td>
</tr>
</tbody>
</table>
Conclusion

• Combining Partial Order Reduction with Bounded Model Checking
  • From 2 Producers/Consumers ($51,859$ states) to $7$ Producers/Consumers ($\approx 10^{14}$ states)
  • How to choose the number $n$ of partial expansions during Phase-1?
  • Need to apply SBTP to other case studies (more complex, more realistic)
• Appropriate algorithm to check asynchronous systems with symbolic model-checking
Perspectives

- Extend SBTP to handle models featuring variables on infinite domains (SMT solvers)
- Automatically determine the number $n$ of partial expansions during Phase-1
- Consolidate our prototype:
  - Perform state-of-the-art BMC translations
  - Improve input language