Forwarding, Splitting, and Block Ordering to Optimize BDD-based Bisimulation Computation

Ralf Wimmer

(joint work with Marc Herbstritt and Bernd Becker)

Institute of Computer Science
University of Freiburg
Germany

March 7th, 2007
Outline

1 Motivation

2 Foundations
   • Signature-based Computation
   • Symbolic Implementation

3 Optimizations
   • Block Forwarding
   • Split-driven Refinement
   • Block Ordering

4 Experimental Results

5 Conclusion
Motivation
Model Checking

Real System

Requirements

Model Checker

System Model
Possible behaviour

Requirement specification
Allowed Behaviour

No
Counterexample

Yes

Modelling

Formalizing

Next Requirement

Done!
The Models: **STATEMATE**

Industrial state-of-the-practice tool:

Hierarchical, state-transition oriented specifications of reactive systems. Underlying: an LTS $M = (S, A, T)$ with internal behaviour ($\tau$-steps).
The Specification: Timed Reachability

Example

What is the probability to reach a set of goal states within a certain time bound?

⇒ Timed reachability for uniform continuous-time Markov decision processes.
Tool Flow

- Statemate description
- Safety requirements
- Failure-modes

Failure injection → Cone-of-influence reduction → Symbolic LTS → Symbolic Branching Minimization → Explicit Quotient LTS

Stochastic model checking → Continous-Time Markov Decision Process → Composition → Interactive Markov Chain

Discrete Domain

Stochastic Domain
Foundations
An equivalence relation \( P \subseteq S \times S \) on the state space is a **branching bisimulation** iff \( s \xrightarrow{a} s' \) and \( a \neq \tau \lor (s, s') \notin P \) implies for all \( t \) with \( (s, t) \in P \):
Idea

Characterize the states by the ability to execute visible actions.

\[(a, B) \in \text{sig}(P, s) \subseteq \mathcal{A} \times P \text{ iff }\]

\[a \neq \tau \lor B \neq B'.\]
Refinement Operator

Group states according to their signature:

\[
\text{sigref}(P) = \bigcup_{B \in P} \{ \{ t \in B \mid \text{sig}(P, s) = \text{sig}(P, t) \} \mid s \in B \}
\]

Applying \text{sigref} until a fixpoint is reached yields the coarsest branching bisimulation [Blom/Orzan, 2003].
Data Representation

- Use the **characteristic function** of
  - state space
  - transition relation

  ⇒ BDDs $S(s), T(s, a, t)$.

- **Partition representation:**
  - Assign a unique number to each block, i.e., $P = \{B_1, \ldots, B_n\}$.
  - Binary encoding of the block numbers.
  - BDD $P(s, k) = 1 \iff s \in B_{\langle k \rangle}$.

- **Signature representation:**
  $\sigma(s, a, k) = 1 \iff (a, B_{\langle k \rangle}) \in \text{sig}(P, s)$
Signature computation

Operations

Current BDD packages (e.g. CuDD) provide all necessary operations:

- Reflexive transitive closure of a relation
- Concatenation of relations
- Substitution of a state by its block number
- ...
Partition Refinement

New operation needed:

- **Signature of all states that lead to node** $v$

- **BDD-representation of the new block number** $k_0$
Optimizations
Avoiding expensive expressions

Observation

The computation of

\[ \exists k : (P(s, k) \land P(t, k)) \]  

(needed for the identification of inert transitions) is very expensive.
Avoiding expensive expressions

Observation
The computation of
\[ \exists k : (P(s, k) \land P(t, k)) \]  (1)
(needed for the identification of inert transitions) is very expensive.

Solution
Avoid it by refining not all blocks in one step but only one block \( B(s) \) at a time. Replace (1) by
\[ B(s) \land B(t). \]
Block Forwarding

Idea

Update the partition after each refinement step:

\[ P \leftarrow (P \setminus B) \cup \text{sigref}(P, B) \]

⇒ Faster convergence.
Split-driven Refinement (1)

**Idea**

Refine only those blocks which are possibly unstable.

\[ bw_{\text{sig}}(P, B) = \{ B' \in P \mid \exists s \in B' \exists a \in A : (a, B) \in \text{sig}(P, s) \} \]
Split-driven Refinement (2)

Problem
Because we are walking backwards, we again need the expensive expression
\[ \exists k : (\mathcal{P}(s, k) \land \mathcal{P}(t, k)) \].
(We have to ignore inert \( \tau \)-steps).

\[ \tau^* \]
\[ a \]
\[ \tau^* \]
\[ b \]

Block B was split
Potentially unstable blocks
Split-driven Refinement (3)

Solution

Compute an over-approximation of the potentially unstable blocks. This does not impact the correctness.

\[
bw_{\text{sig}}^{oa}(P, B) = \{ B' \in P \mid \exists s' \in B', s \in B, a \in A : s' \xrightarrow{\tau^*a} s \}
\]

Most of \(bw_{\text{sig}}^{oa}\) can be computed in a preprocessing step. So, it’s very efficient.
Block Ordering

Observation
The order in which the blocks are refined influences the runtimes.

Heuristics for the determination of a block order:

- **SortByBlockSize**
  Refine blocks with many states first.

- **SortByBWSig**
  Refine blocks first which influence many other blocks.

- **SortByBDDSize**
  Refine blocks represented by a small BDD first.
Experimental Results
Benchmarks

- Milner’s Scheduler ($ml-n$)
- Kanban Production System ($kb-n$)
- STATEMATE models of a train control system ($etcs-n$), a braking controller of an airplain ($bs-p$), and an industrial benchmark ($ctrl$).
Number of Refined Blocks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Number of Refined Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>kb-4</td>
<td></td>
</tr>
<tr>
<td>kb-5</td>
<td></td>
</tr>
<tr>
<td>kb-6</td>
<td></td>
</tr>
<tr>
<td>kb-7</td>
<td></td>
</tr>
<tr>
<td>kb-8</td>
<td></td>
</tr>
<tr>
<td>kb-9</td>
<td></td>
</tr>
<tr>
<td>ml-4</td>
<td></td>
</tr>
<tr>
<td>ml-5</td>
<td></td>
</tr>
<tr>
<td>ml-6</td>
<td></td>
</tr>
<tr>
<td>ml-7</td>
<td></td>
</tr>
<tr>
<td>ml-8</td>
<td></td>
</tr>
<tr>
<td>etcs1</td>
<td></td>
</tr>
<tr>
<td>etcs2</td>
<td></td>
</tr>
<tr>
<td>etcs3</td>
<td></td>
</tr>
<tr>
<td>bs-p</td>
<td></td>
</tr>
<tr>
<td>ctrl</td>
<td></td>
</tr>
</tbody>
</table>

- original
- with split-driven refinement
- sdr + order w.r.t. block sizes
- sdr + order w.r.t. size of bw_sig

Number of Refined Blocks: 0, 50000, 100000, 150000, 200000, 250000, 300000, 350000, 400000.
Runtimes

Runtime of Sigref Benchmark
original
with split-driven refinement
sdr + order w.r.t. block sizes
sdr + order w.r.t. size of bw_sig

Benchmark
kb-4    kb-5    kb-6    kb-7    kb-8    kb-9    ml-4    ml-5    ml-6    ml-7    ml-8    etcs1    etcs-2    etcs-3    bs-p    ctrl
Conclusion
Summary + Future Work

We have seen:

- symbolic computation of branching bisimulations
- signature-based approach
- optimizations that speed-up the computation
- experimental results showing the effectiveness of the optimizations.

To be done:

- Symbolic computation of stochastic bisimulations
- Handling different notions of divergence
Thank you for your attention!
Do you have questions?