A BDD-based BMC Approach for the Verification of Multi-Agent Systems

Andrew V. Jones    Alessio Lomuscio
{andrewj, alessio}@doc.ic.ac.uk

Department of Computing
Imperial College London

Concurrency, Specification, and Programming
September, 2009
The majority of “symbolic” model checkers use binary decision diagrams.

Current bounded model checkers, either:
- Require a translation of the problem to SAT, or
- Aren’t very expressive

And – they concentrate on “bug hunting”, not verifying correctness.

SAT-based BMC and conventional model checking techniques are, at the moment, seen as complementary to each other.
Is it possible to convert an existing model checker to a bounded one?

And ... 

How can we distribute the verification process?
What is an agent?

“An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives.” – Weiss
A system is composed of a set of agents $A = \{1, \ldots, n\}$ and an environment $e$.

Each agent, $i$, is described by
- A set of local states – $L_i$
- A set of local actions – $\text{Act}_i$
- A local protocol function – $\mathcal{P}_i : L_i \rightarrow 2^{\text{Act}_i}$
- An evolution function – $\tau_i : L_i \times \text{Act} \rightarrow L_i$

$\text{Act}$ is the set of joint actions –
$\text{Act} \subseteq \text{Act}_1 \times \cdots \times \text{Act}_n \times \text{Act}_e$
G is the set of possible global states – $G \subseteq \mathcal{L}_1 \times \cdots \times \mathcal{L}_n \times \mathcal{L}_e$

- A global state $g = (l_1, \ldots, l_n, l_e) \in G$ represents a “snapshot” of the system
- $l_i : G \rightarrow \mathcal{L}_i$ is a projection of agent $i$’s local state from the given global state
- Each agent $i$ has an epistemic relation – $g \sim_i g'$ iff $l_i(g) = l_i(g')$

T is a transition relation for the system – $T \subseteq G \times \text{Act} \times G$

- $g \mathrel{T} g'$ iff there exists actions $a_1, \ldots, a_n$ such that for all $i$, $a_i \in \mathcal{P}_i(l_i(g))$ and $\tau_i(l_i(g), a_1, \ldots, a_n) = l_i(g')$
A model $M_{\mathcal{IS}}$ is a tuple $(G, \iota, T, \sim_1, \ldots, \sim_n, \mathcal{V})$

- $\iota \in G$ is an initial global state
- $G$ is the set of reachable states accessible from $\iota$ via $T$
- $\mathcal{V}$ is a mapping from global states to propositional variables $(\mathcal{PV})$ – $\mathcal{V} : G \rightarrow 2^{\mathcal{PV}}$

A path $\pi = (\iota, g_1, \ldots)$ is an infinite sequence of global states such that $\forall k \geq 0 \ (g_k, g_{k+1}) \in T$.

- $\pi(k)$ is the $k^{th}$ global state of the path $\pi$
- $\Pi(g)$ is the set of all paths starting at $g \in G$
A Temporal Epistemic Logic – CTLK

- Syntax

\[ \varphi ::= p \mid \neg \varphi \mid \varphi \land \varphi \mid EX \varphi \mid EG \varphi \mid E[\varphi U \psi] \mid Ki \varphi \]

- Derived Modalities

  - \( EF \varphi \overset{\text{def}}{=} E[\text{true}U \varphi] \)
  - \( AX \varphi \overset{\text{def}}{=} \neg EX \neg \varphi \)
  - \( AF \varphi \overset{\text{def}}{=} \neg EG \neg \varphi \)
  - \( AG \varphi \overset{\text{def}}{=} \neg EF \neg \varphi \)
  - \( \overline{Ki} \varphi \overset{\text{def}}{=} \neg Ki \neg \varphi \)

- \( A[\varphi U \psi] \) – Takes the expected meaning
$K_i \varphi$ – “agent $i$ knows that $\varphi$”

$\mathcal{M}_{IS}, g \models K_i \varphi$ iff $\forall g' \in G, g \sim_i g'$ implies $\mathcal{M}_{IS}, g' \models \varphi$

$\overline{K}_i \varphi$ – “agent $i$ considers it possible that $\varphi$”

$\mathcal{M}_{IS}, g \models \overline{K}_i \varphi$ iff $\exists g' \in G : g \sim_i g'$ and $\mathcal{M}_{IS}, g' \models \varphi$
Conventional “symbolic” model checkers use a canonical representation called ROBDDs. These can be used to efficiently represent a Boolean function.

Two main stages:
1. Calculate the entire reachable state space
2. Recursively evaluate the property using fix point methods

Interpreted Systems Reachable State Space

$$\text{lfp}(Q) = (I(g) \lor \exists g' (T(g, a, g') \land Q(g'))$$
Model Checking Multi-Agent Systems

MCMAS:

- Symbolic model checker for verifying *certain* aspects of multi-agent systems
- Based on BDDs from the CUDD library
- Based on the “Interpreted Systems Programming Language”

Interpreted Systems Programming Language – ISPL – allows for the definition of multi-agent systems following the interpreted systems formalism. ISPL syntax includes:

- Agent
- Evaluation
- Formulae
ECTLK

CTLK as before:
- Restricts negation to only appear in front of elements of $\mathcal{P}V$
- Contains $\overline{K}_i$ not $K_i$

ACTLK

A formula is in ACTLK if the negation is ECTLK... 

$$ACTLK \subseteq \{\varphi | \neg \varphi \in ECTLK\}$$
### Current Approaches to Bounded Model Checking

---

**SAT-based BMC**

<table>
<thead>
<tr>
<th>Idea</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Unroll” the transition relation $k$ times</td>
</tr>
<tr>
<td>Translate the <strong>negated</strong> property and unrolled model to the Boolean satisfiability problem</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requires a SAT solver</td>
</tr>
<tr>
<td>Not straightforward to convert a BDD model checker to a SAT one</td>
</tr>
</tbody>
</table>

---

**BDD-based BMC**

<table>
<thead>
<tr>
<th>Idea</th>
</tr>
</thead>
<tbody>
<tr>
<td>Represent the “error” as a bad state</td>
</tr>
<tr>
<td>Find all of the reachable states at a depth $k$</td>
</tr>
<tr>
<td>Check the intersection of the error state with the “fringe”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>How does one specify properties as a <em>single</em> error state? Okay for invariant/safety properties – but apart from that?</td>
</tr>
</tbody>
</table>

---

Andrew V. Jones, Alessio Lomuscio

BDD-based BMC for Multi-Agent Systems
BDD-BMC(\(\psi : \text{ACTLK Formula}, \mathcal{I} : \text{Initial State}, \text{Trans} : \text{Transition Relation} \)) : \text{Boolean}

1: \(\varphi \leftarrow \neg \psi \) \{\(\varphi : \text{ECLTK Formula} \}\}
2: \text{Reach} \leftarrow \mathcal{I} \{\text{Reach} : \text{BDD} \}\)
3: \textbf{while} \ TRUE \ \textbf{do}
4: \ \textbf{if} \ 1 \rightarrow D  = \text{Reach} \ \textbf{then}
5: \ \ \ \textbf{return} \ \text{FALSE} \ \{\text{Counterexample found} \}\)
6: \ \textbf{end if}
7: \ \text{Reach} \leftarrow \text{Reach} \lor (\text{Reach} \land \text{Trans})
8: \ \textbf{if} \ \text{Reach} \ \text{Unchanged} \ \textbf{then}
9: \ \ \ \textbf{break} \ \{\text{Fixed point reached} \}\)
10: \ \textbf{end if}
11: \ \textbf{end while}
12: \ \textbf{return} \ 1 \rightarrow D  = \text{Reach}
Extending $\text{SAT}_{\text{CTLK}}$

Current fix point methods work even when using non-serial transition relations, but we need a symbolic method for $\overline{K}_i$.

We could just use $\overline{K}_i \varphi \overset{\text{def}}{=} \neg K_i \neg \varphi$.

$\text{SAT}_{\overline{K}}(\varphi : \text{FORMULA}, i : \text{AGENT}) : \text{set of STATE}$

1. $X \leftarrow \text{SAT}_{\text{CTLK}}(\varphi)$
2. $Y \leftarrow \text{pre}_K(X, i)$
3. $\text{return } Y$
Inside MCMAS

BDD basic_agent::project_local_state(BDD *state, BDDvector* v)
{
    BDD tmp = bddmgr->bddOne();

    // For all of the state variables before the agent ...
    for (int j = 0; j < get_var_index_start(); j++)
    {
        // 'and' them on
        tmp = tmp * (*v)[j];
    }

    // and after the agent ...
    for (int j = get_var_index_end() + 1; j < v->count(); j++)
    {
        // 'and' them on
        tmp = tmp * (*v)[j];
    }

    return state->ExistAbstract(tmp);
}

Figure: The simplified project_local_state method
We restrict verification to invariant properties, i.e. “AG” is the top most connective.

### Distributed Approach

1. **Fixed-Depth BDD-based BMC.**
2. **Seed State Generation.**
3. **Distributed Parallel BDD-based BMC.**
Distributed Verification of Invariant Properties

Andrew V. Jones, Alessio Lomuscio

BDD-based BMC for Multi-Agent Systems
Distributed Verification of Invariant Properties

Seed 1

Depth $k$

Andrew V. Jones, Alessio Lomuscio

BDD-based BMC for Multi-Agent Systems
Distributed Verification of Invariant Properties

Seed 1
Seed 2
Seed 3

Depth $k$

Andrew V. Jones, Alessio Lomuscio

BDD-based BMC for Multi-Agent Systems
Distributed Verification of Invariant Properties

Seed 1

Seed 2

Seed 3

Depth $k$

Andrew V. Jones, Alessio Lomuscio

BDD-based BMC for Multi-Agent Systems
A Scalable Model
The Train Gate Controller Model

Controller

Tunnel

Eastbound Train

Westbound Train

Andrew V. Jones, Alessio Lomuscio

BDD-based BMC for Multi-Agent Systems
Faulty trains now have a service counter and a breaking depth...
The Faulty Train Gate Controller – Controller

<table>
<thead>
<tr>
<th>Label</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>IDLE</td>
</tr>
<tr>
<td>$c_2$</td>
<td>EXIT_TRAIN</td>
</tr>
<tr>
<td>$c_3$</td>
<td>IDLE</td>
</tr>
<tr>
<td>$c_4$</td>
<td>IDLE</td>
</tr>
<tr>
<td>$c_5$</td>
<td>ENTER_TRAIN_T</td>
</tr>
</tbody>
</table>

$T \in \{E,W\}$
Example Specifications

\( \varphi_{TGC1} : \text{“Trains can’t stay in the tunnel forever”} \)

\[ AG(\mathbf{AF}(\neg \text{TrainE\_in\_tunnel})) \]

**Mutual Exclusion:**

\( \varphi_{TGC2} : \text{“Two trains never occupy the tunnel at the same time”} \)

\[ AG(\neg \text{TrainE\_in\_tunnel} \lor \neg \text{TrainW\_in\_tunnel}) \]

\( \varphi_{TGC3} : \text{“When a train is in the tunnel it knows that another train is not”} \)

\[ AG(\text{TrainE\_in\_tunnel} \rightarrow K_{\text{TrainE}}(\neg \text{TrainW\_in\_tunnel})) \]
Figure: Memory usage for two trains, with a full service depth of 20 – when checking various formulae
Figure: Time required for two trains, with a full service depth of 20 – when checking various formulae
Biere et al. (1999): bounded model checking “finds counterexamples of minimal length”.

<table>
<thead>
<tr>
<th>Method</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \varphi_{TGC1} )</td>
</tr>
<tr>
<td>Regular</td>
<td>25</td>
</tr>
<tr>
<td>BMC</td>
<td>13</td>
</tr>
</tbody>
</table>

Table: Length of counterexamples generated between BMC and full verification
Given we don’t have completeness for epistemic sub-formulae, we need a different formula to verify . . .

**Temporal-Epistemic**

\[ AG(\text{TrainE\_in\_tunnel} \rightarrow K_{\text{TrainE}} (\neg \text{TrainW\_in\_tunnel})) \]

**Temporal-Only**

\[ AG(\text{TrainE\_in\_tunnel} \rightarrow AX (\neg \text{TrainW\_in\_tunnel})) \]
Table: A comparison of seeded BMC vs. BMC for a single master and 3 slaves (seed depth of 4).

<table>
<thead>
<tr>
<th>Model</th>
<th>Ratio</th>
<th>Memory</th>
<th>Time</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faulty</td>
<td>1.8255</td>
<td>3.8130</td>
<td>1.7297</td>
<td></td>
</tr>
<tr>
<td>Working</td>
<td>0.9500</td>
<td>0.0013</td>
<td>0.0008</td>
<td></td>
</tr>
</tbody>
</table>

Table: Ratios comparing time for seeded BMC vs. BMC, for a varying number of slaves (seed depth of 3).

<table>
<thead>
<tr>
<th># Hosts</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.0033</td>
</tr>
<tr>
<td>4</td>
<td>0.0066</td>
</tr>
<tr>
<td>6</td>
<td>0.0098</td>
</tr>
<tr>
<td>8</td>
<td>0.0131</td>
</tr>
</tbody>
</table>
BDD-based BMC is shown to be effective with variable reordering disabled; for satisfiable formulae the overhead imposed is negligible.

Distributed BDD-based BMC:
- Out-performs the sequential approach for falsifiable formulae
- Increasing the number of hosts increases verification efficiency

Further work
- Completeness of temporal-epistemic formulae
- Intersection-based BMC using MCMAS’s RedStates
- More Models
- Comparison to VERICS
- Smarter use of CUDD using Cudd RecursiveDeref
Are there any questions?
Backup Slides
SAT requires an encoding of a “back loop”

**Semantics of ECTLK**

\[ M_{IS}, g \models EG\varphi \iff \exists \pi \in \Pi(g) \ \forall m \geq 0 \ M_{IS}, \pi(m) \models \varphi \]

**Bounded Semantics of ECLTK**

\[ M_k, g \models EG\varphi \iff \exists \pi \in P_k(\pi(0) = g \ \wedge \ \forall 0 \leq j \leq k \ M_k, \pi(j) \models \varphi) \]

... requires \( \text{loop}(\pi) \neq \emptyset \)

\[
\text{loop}(\pi) = \emptyset \quad \quad \quad \text{loop}(\pi) = \{2\}
\]
BDD approaches require an explicit “error state”

**BoundedTraversal** (Trans : Transition Relation, $\mathcal{I}$ : Initial States, Err : Error State, $k$ : Depth)

1: Frontier$_0$ ← $\mathcal{I}$
2: for ($i = 0; i < k; i++$) do
3: if (Frontier$_i$ · Err $\neq \emptyset$) then
4: return (FAILURE)
5: end if
6: Frontier$_{i+1}$ ← IMG(Trans, Frontier$_i$)
7: end for
8: return (PASS)
CUDD ExistAbstract

- Quantification – $B_h = \exists x \ B_f$
- Shannon’s expression – $h = (\neg x \land f|_{x\leftarrow 0}) \lor (x \land f|_{x\leftarrow 1})$
\[ \varphi_{\text{TGC4}}(N) \]

\[
AG \left( \left( \text{TRAIN}_i \_\text{IN\_TUNNEL} \rightarrow K_{\text{TRAIN}_i} \right) \wedge \left( \bigwedge_{j=1}^{i-1} \neg \text{TRAIN}_j \_\text{IN\_TUNNEL} \right) \wedge \left( \bigwedge_{j=i+1}^{N} \neg \text{TRAIN}_j \_\text{IN\_TUNNEL} \right) \right) \]

Andrew V. Jones, Alessio Lomuscio
BDD-based BMC for Multi-Agent Systems
Parameterised Specifications

$$\varphi_{TGC5}(N)$$

$$\begin{align*}
AG & \left( \text{TRAIN}_i \_ \text{IN} \_ \text{TUNNEL} \rightarrow K_{\text{TRAIN}_i} \right) \\
& \left( \bigwedge_{j=1}^{i-1} AX \left( \neg \text{TRAIN}_j \_ \text{IN} \_ \text{TUNNEL} \right) \right) \land \\
& \left( \bigwedge_{j=i+1}^{N} AX \left( \neg \text{TRAIN}_j \_ \text{IN} \_ \text{TUNNEL} \right) \right)
\end{align*}$$