A Method of Bounded Model Checking for a Temporal Epistemic Logic Based on Reduced Ordered Binary Decision Diagrams

Andrew JONES
⟨avj05@doc.ic.ac.uk⟩

Supervisor: Dr. Alessio R. LOMUSCIO

Department of Computing
Imperial College London

June, 2009
Teach Yourself Model Checking in 1 Minute

Real system

Modeling

Model of system

Requirements

Formalising

Requirements specification

Model Checker

"False"

Counterexample

"True"

Done
An Illustrative Scenario

The Mars Rover

NASA want to ensure that their Mars Rovers *always* transmit data.

So, someone suggests that they should use model checking to verify this...

Specifying the Property

Assume that $\varphi$ represents that the data has been transmitted.
An Illustrative Scenario

The Mars Rover
NASA want to ensure that their Mars Rovers always transmit data.

So, someone suggests that they should use model checking to verify this...

Specifying the Property
Assume that $\varphi$ represents that the data has been transmitted

Andrew Jones (avj05@doc.ic.ac.uk)  BDD-based-BMC for Epistemic Logic
An Illustrative Scenario

The Mars Rover

NASA want to ensure that their Mars Rovers always transmit data.

So, someone suggests that they should use model checking to verify this...

Specifying the Property

Assume that $\varphi$ represents that the data has been transmitted.

Andrew JONES ⟨avj05@doc.ic.ac.uk⟩

BDD-based-BMC for Epistemic Logic
An Illustrative Scenario – “Global” Model Checking

Original Approach

On all possible paths through the model, does the rover always transmit the data?

\[ \mathcal{M}, \text{start} \models AF(\varphi) \]
An Illustrative Scenario – “Bounded” Model Checking

A New Approach

*Does there exist a path through the model in which the rover never transmits the data?*

\[ \mathcal{M}, \text{start} \models EG(\neg \varphi) \]
Yes, but …

- 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.
The, ahem, “million” dollar question

Is it possible to convert an existing model checker to a bounded one?

Andrew JONES ⟨avj05@doc.ic.ac.uk⟩

BDD-based-BMC for Epistemic Logic
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 – $P_i : L_i \rightarrow 2^{\text{Act}_i}$
- 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 function for the system – $T \subseteq G \times \text{Act} \times G$

  • $g \xRightarrow{T} g'$ if 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 $\mathcal{M}_{IS}$ 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 a sequence of global states such that $(g_k, g_{k+1}) \in T$.

- $\pi(k)$ is the $k^{th}$ global state of the path $\pi$
- $\Pi(g)$ is the set of all paths starting at $g \in G$
Syntax

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

Derived Modalities

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

- \( A[\varphi U \psi] \) – Takes the expected meaning
Epistemic Modalities – $K$ and $\overline{K}$

$K_i \varphi$ – “agent $i$ knows that $\varphi$”

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

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

$$\mathcal{M}_{IS}, g \models \overline{K}_i \varphi \text{ iff } \exists g' \in G : g \sim_i g' \text{ and } \mathcal{M}_{IS}, g' \models \varphi$$
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

\[ \text{lfp}(Q) = (I(g) \lor \exists g' (T(g, a, g') \land Q(g'))) \]
ROBDD Variable Ordering

$x_1 < x_2 < x'_1 < x'_2$

$x_1 < x'_1 < x_2 < x'_2$
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” formalisation.

CUDD ExistAbstract

- Quantification – $B_h = \exists x \ B_f$
- Shannon’s expression – $h = (\neg x \land f|_{x←0}) \lor (x \land f|_{x←1})$
The Existential and Universal Fragments

**ECTLK**

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

**ACTLK**

A formula is in ACTLK if the negation is ECTLK…

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

**SAT-based-BMC**

**Idea**
- “Unroll” the transition relation \( k \) times
- Translate the **negated** property and unrolled model
- Use a SAT solver

**Problem**
- Requires a SAT solver
- Not straightforward to convert a BDD model checker to a SAT one

**BDD-based-BMC**

**Idea**
- Represent the “error” as a bad state
- Find all of the reachable states at a depth \( k \)
- Check the intersection

**Problem**
- How does one specify properties as a *single* error state? Okay for *invariant* properties – but apart from that?

Andrew JONES (avj05@doc.ic.ac.uk)
BDD-BMC(ψ : ACTLK Formula, I : Initial State, Trans : Transition Relation) : Boolean

1: ϕ ← ¬ψ {ϕ : ECLTK Formula}
2: Reach ← I {Reach : BDD}
3: while True do
4:    if \([ι \rightarrow ϕ] = \text{Reach}\) then
5:      return FALSE \{Counterexample found\}
6:    end if
7:    Reach ← Reach \lor (\text{Reach} \land \text{Trans})
8:    if Reach Unchanged then
9:      break \{Fixed point reached\}
10:  end if
11: end while
12: return \([ι \rightarrow ψ] = \text{Reach}\)
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 $\overline{K}_i \varphi \overset{\text{def}}{=} \neg K_i \neg \varphi$ ...

```plaintext
$\text{SAT}_\overline{K}(\varphi : \text{Formula}, i : \text{Agent}) : \text{set of State}$

1: $X \leftarrow \text{SAT}_{\text{CTLK}}(\varphi)$
2: $Y \leftarrow \text{pre}_K(X, i)$
3: return $Y$
```
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
Distributed Verification of Invariant Properties

\( \phi \)

Depth \( k \)

Andrew Jones (avj05@doc.ic.ac.uk)

BDD-based-BMC for Epistemic Logic
Distributed Verification of Invariant Properties

Andrew Jones (avj05@doc.ic.ac.uk)  BDD-based-BMC for Epistemic Logic
Distributed Verification of Invariant Properties

Seed 1

Seed 2

Seed 3

Depth $k$

Andrew Jones (avj05@doc.ic.ac.uk) BDD-based-BMC for Epistemic Logic
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?</td>
</tr>
</tbody>
</table>

Andrew Jones ⟨avj05@doc.ic.ac.uk⟩

BDD-based-BMC for Epistemic Logic
Initial Results

<table>
<thead>
<tr>
<th>Model</th>
<th>Decrease</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Memory</td>
<td>States</td>
</tr>
<tr>
<td>T</td>
<td>M</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>2</td>
</tr>
</tbody>
</table>

Table: *vanilla* CUDD ($\varphi_{TGC5}$)

<table>
<thead>
<tr>
<th>Model</th>
<th># Reorderings</th>
<th># Garbage Collections</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original</td>
<td>BDD-BMC</td>
</tr>
<tr>
<td>T</td>
<td>M</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>2</td>
</tr>
</tbody>
</table>

Table: Reorderings and garbage collections performed by CUDD
Figure: Resource use of BMC against regular model checking, with a model containing two type 1 trains, with a maximum depth of 100 and various breaking depths - $\varphi_{TGC2}$
Figure: Memory usage for two type 2 trains, with a full service depth of 20 – when checking various formulae
Figure: Time required for two type 2 trains, with a full service depth of 20 – when checking various formulae
Counterexamples

- Biere et al (1999): bounded model checking “finds counterexamples of minimal length”.
- Groce et al (2005): “bounded model checkers often produce counterexamples that are difficult to understand due to the values chosen by a SAT solver”.

<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
Effectiveness of One-Shot BMC

- Zero initial CUDD cache
- Garbage collection and asynchronous reordering enabled

<table>
<thead>
<tr>
<th>B</th>
<th>Decrease</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Memory</td>
<td>Time</td>
</tr>
<tr>
<td>5</td>
<td>1.00</td>
<td>0.50</td>
</tr>
<tr>
<td>10</td>
<td>1.78</td>
<td>1.24</td>
</tr>
<tr>
<td>15</td>
<td>1.70</td>
<td>0.63</td>
</tr>
<tr>
<td>WORKING</td>
<td>1.00</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Table: 2 Trains, Max Counter 20

<table>
<thead>
<tr>
<th>B</th>
<th>Decrease</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Memory</td>
<td>Time</td>
</tr>
<tr>
<td>2</td>
<td>1.13</td>
<td>0.91</td>
</tr>
<tr>
<td>4</td>
<td>1.88</td>
<td>0.39</td>
</tr>
<tr>
<td>6</td>
<td>1.70</td>
<td>0.82</td>
</tr>
<tr>
<td>WORKING</td>
<td>1.00</td>
<td>0.14</td>
</tr>
</tbody>
</table>

Table: 3 Trains, Max Counter 7
# Evaluating Distributed Bounded Model Checking

## Table: A comparison between BMC and seeded BMC, at a seed state generation depth of 4.

<table>
<thead>
<tr>
<th>Model</th>
<th>Decrease</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Memory</td>
<td>Time</td>
<td>States</td>
<td></td>
</tr>
<tr>
<td>Type 1</td>
<td>1.033</td>
<td>0.772</td>
<td>1.002</td>
<td></td>
</tr>
<tr>
<td>Type 2</td>
<td>1.730</td>
<td>4.429</td>
<td>1.709</td>
<td></td>
</tr>
<tr>
<td>Type 3</td>
<td>0.907</td>
<td>0.005</td>
<td>0.003</td>
<td></td>
</tr>
</tbody>
</table>

## Table: The length of time for seeded BMC compared to BMC, for a varying number of hosts when a counterexample cannot be found.

<table>
<thead>
<tr>
<th># Hosts</th>
<th>Time</th>
<th>Decrease</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>118.88</td>
<td>0.016</td>
</tr>
<tr>
<td>4</td>
<td>58.78</td>
<td>0.032</td>
</tr>
<tr>
<td>6</td>
<td>39.93</td>
<td>0.048</td>
</tr>
<tr>
<td>8</td>
<td>30.37</td>
<td>0.063</td>
</tr>
</tbody>
</table>
Conclusion

BDD-based-BMC is shown to be effective with variable reordering disabled; for satisfiable formulae the overhead imposed is negligible.

Further work

- Intersection-based-BMC using MCMAS’s RedStates
- More Models
- Comparison to VERICS
- Smarter use of CUDD using Cudd_RecursiveDeref
Questions?
Backup Slides
SAT requires an encoding of a “back loop”

### Semantics of ECTLK

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

### Bounded Semantics of ECTLK

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

... requires loop(\pi) \neq \emptyset

---

\[ \text{loop}(\pi) = \emptyset \]

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

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

1: Frontier\(_0\) \(\leftarrow\) \(\mathcal{I}\)
2: for \(i = 0; i < k; i++\) do
3: if (Frontier\(_i\) \(\cdot\) Err \(\neq\) \(\emptyset\)) then
4: return (FAILURE)
5: end if
6: Frontier\(_{i+1}\) \(\leftarrow\) IMG(Trans, Frontier\(_i\))
7: end for
8: return (PASS)
"One-shot" BMC(ψ : ACTLK Formula, I : Initial State, Trans : Transition Relation, OneShotBound : int) : String × Boolean

1: ϕ ← ¬ψ {ϕ : ECLTK Formula}
2: Reach ← I {Reach : BDD}
3: for k ← 0 to OneShotBound do
4:   Reach ← Reach ∨ (Reach ∧ Trans)
5:   if Reach Unchanged then
6:     return Fixed point case: [ι → ψ] = Reach
7:   end if
8: end for
9: return One shot case: [ι → ϕ] = Reach
Regular Specifications

\[ \varphi_{TGC1} \]
\[ AG( AF( \neg Train_1\_in\_tunnel)) \]

\[ \varphi_{TGC2} - \text{"Mutual Exclusion"} \]
\[ AG( \neg Train_1\_in\_tunnel \lor \neg Train_2\_in\_tunnel) \]

\[ \varphi_{TGC3} \]
\[ AG( K_{Train_1}( \neg Train_1\_in\_tunnel \lor \neg Train_2\_in\_tunnel)) \]
Parameterised Specifications

\[ \varphi_{TGC4}(N) \]

\[
AG \left( \left( \neg TRAIN_{i \text{-} IN\text{-}TUNNEL} \rightarrow K_{TRAIN_i} \right) \land \left( \neg TRAIN_{j \text{-} IN\text{-}TUNNEL} \right) \land \left( \neg TRAIN_{j \text{-} IN\text{-}TUNNEL} \right) \right) \]

Andrew Jones (avj05@doc.ic.ac.uk)  BDD-based-BMC for Epistemic Logic
Parameterised Specifications

\[ \varphi_{TGC5}(N) \]

\[ AG \left( \left( \bigwedge_{j=1}^{i-1} AX \left( \neg \text{Train}_j \text{ in tunnel} \right) \right) \wedge \left( \bigwedge_{j=i+1}^{N} AX \left( \neg \text{Train}_j \text{ in tunnel} \right) \right) \rightarrow K_{\text{Train}_i} \right) \]