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

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.
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...
**Original Approach**

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

\[ M, \text{start} \models AF(\varphi) \]
A New Approach

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

\[ M, \text{start} \models EG(\neg \varphi) \]
Wait . . . Doesn’t that already exist?

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.
Is it possible to convert an existing model checker to a bounded one?
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 – $\mathcal{L}_i$
- A set of local actions – $\text{Act}_i$
- A local protocol function – $\mathcal{P}_i : \mathcal{L}_i \rightarrow 2^{\text{Act}_i}$
- An evolution function – $\tau_i : \mathcal{L}_i \times \text{Act} \rightarrow \mathcal{L}_i$

$\text{Act}$ 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.
- \( l_i : G \to \mathcal{L}_i \) 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 – \( T \subseteq G \times \text{Act} \times G \)

- \( g T g' \) 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 $M_{IS}$ 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

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

Derived Modalities

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

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

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

$\mathcal{M}_{IS}, g \models K_i \varphi$ 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

\[
\text{lfp}(Q) = (I(g) \lor \exists g' (T(g, a, g') \land Q(g')))
\]
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 formalism. ISPL syntax includes:

- Agent
- Evaluation
- Formulae
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...

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

**SAT-based-BMC**

**Idea**
- “Unroll” the transition relation $k$ times
- Translate the negated property and unrolled model to the boolean satisfiability problem

**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-based-BMC for Epistemic Logic
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 $\overline{K} i \varphi \overset{\text{def}}{=} \neg K i \neg \varphi$ . . .

**$\text{SAT}_K(\varphi : \text{FORMULA}, i : \text{AGENT}) : \text{set of STATE}$**

1: $X \leftarrow \text{SAT}_{\text{CTLK}}(\varphi)$
2: $Y \leftarrow \text{pre}_K(X, i)$
3: **return** $Y$
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 $k$
A Scalable Model
Faulty trains now have a service counter and a breaking depth...
The Faulty Train Gate Controller – Controller

![Diagram of a state machine with states Red and Green, transitions labeled with c1, c2, c3, c4, and c5, and actions for each label:]

- **c1**: IDLE
- **c2**: EXIT_TRAIN
- **c3**: IDLE
- **c4**: IDLE
- **c5**: ENTER_TRAIN_T, where T ∈ {E, W}

Andrew JONES ⟨avj05@doc.ic.ac.uk⟩  BDD-based-BMC for Epistemic Logic
Example Specifications

“Trains can’t stay in the tunnel forever”

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

Mutual Exclusion:

“Two trains never occupy the tunnel at the same time”

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

“When a train is in the tunnel it knows that another train is not”

\( AG (\text{TrainE\_in\_tunnel} \rightarrow K_{\text{TrainE}} (\neg \text{TrainW\_in\_tunnel})) \)
### Initial Results

<table>
<thead>
<tr>
<th>Model</th>
<th>Decrease</th>
<th>Memory</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>M</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>4</td>
<td>0.9750</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>6</td>
<td>0.9632</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>4</td>
<td>0.1561</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>2</td>
<td>0.2338</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>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, in a model containing two trains, with a maximum depth of 100 and various breaking depths – $\varphi_{TGC2}$
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>$\phi_{TGC1}$</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
Evaluating Distributed Bounded Model Checking

<table>
<thead>
<tr>
<th>Model</th>
<th>Decrease</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Memory</td>
<td>Time</td>
<td>States</td>
</tr>
<tr>
<td>Faulty</td>
<td>1.730</td>
<td>4.429</td>
<td>1.709</td>
</tr>
<tr>
<td>Working</td>
<td>0.907</td>
<td>0.005</td>
<td>0.003</td>
</tr>
</tbody>
</table>

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

<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>

Table: The length of time for seeded BMC compared to BMC, for a varying number of hosts when a counterexample cannot be found.
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
Are there any questions?
Backup Slides
SAT requires an encoding of a “back loop”

Semantics of ECTLK

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

Bounded Semantics of ECTLK

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

\[ \ldots \text{requires } \text{loop}(\pi) \neq \emptyset \]

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

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

1: Frontier\(_0\) ← \( I \)
2: for \((i = 0; i < k; i++)\) do
3:   if \((\text{Frontier}_i \cdot \text{Err} \neq \emptyset)\) then
4:     return (FAILURE)
5:   end if
6:   Frontier\(_{i+1}\) ← IMG(Trans, Frontier\(_i\))
7: end for
8: return (PASS)
“One-shot” BMC($\psi : $ ACTLK Formula, $I : $ Initial State, Trans : Transition Relation, OneShotBound : int) : STRING × BOOLEAN

1: $\varphi \leftarrow \neg \psi \{ \varphi : $ ECLTK Formula$\}$
2: Reach $\leftarrow I \{ Reach : $ BDD$\}$
3: for $k \leftarrow 0$ to OneShotBound do
4:    Reach $\leftarrow$ Reach $\lor$ (Reach $\land$ Trans)
5:    if Reach Unchanged then
6:        return FIXED POINT CASE: $[i \rightarrow \psi] = \text{Reach}$
7:    end if
8: end for
9: return ONE SHOT CASE: $[i \rightarrow \varphi] = \text{Reach}$
Variable Quantification using CUDD

CUDD ExistAbstract

- Quantification \( B_h = \exists x \ B_f \)
- Shannon’s expression \( h = (\neg x \land f|_{x \leftarrow 0}) \lor (x \land f|_{x \leftarrow 1}) \)
Inside MCMAS

```cpp
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
Parameterised Specifications

$\varphi_{TGC4}(N)$

$$AG \left( \left( \neg \text{Train}_i\text{\_IN\_TUNNEL} \rightarrow K_{\text{Train}_i} \right) \wedge \left( \bigwedge_{j=1}^{i-1} \neg \text{Train}_j\text{\_IN\_TUNNEL} \right) \right) \wedge \left( \bigwedge_{j=i+1}^{N} \neg \text{Train}_j\text{\_IN\_TUNNEL} \right)$$
Parameterised Specifications

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

\[ AG \left( \left( \text{TRAIN}_i \text{ IN TUNNEL} \rightarrow K_{\text{TRAIN}_i} \right) \land \left( \bigwedge_{j=1}^{i-1} AX\left( \neg \text{TRAIN}_j \text{ IN TUNNEL} \right) \right) \land \left( \bigwedge_{j=i+1}^{N} AX\left( \neg \text{TRAIN}_j \text{ IN TUNNEL} \right) \right) \right) \]
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>
<th>Decrease</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Memory</td>
<td>Time</td>
<td>Memory</td>
<td>Time</td>
</tr>
<tr>
<td>5</td>
<td>1.00</td>
<td>0.50</td>
<td>1.13</td>
<td>0.91</td>
</tr>
<tr>
<td>10</td>
<td>1.78</td>
<td>1.24</td>
<td>1.88</td>
<td>0.39</td>
</tr>
<tr>
<td>15</td>
<td>1.70</td>
<td>0.63</td>
<td>1.70</td>
<td>0.82</td>
</tr>
<tr>
<td>WORKING</td>
<td>1.00</td>
<td>0.05</td>
<td>WORKING</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Table: 2 Trains, Max Counter 20

Table: 3 Trains, Max Counter 7