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

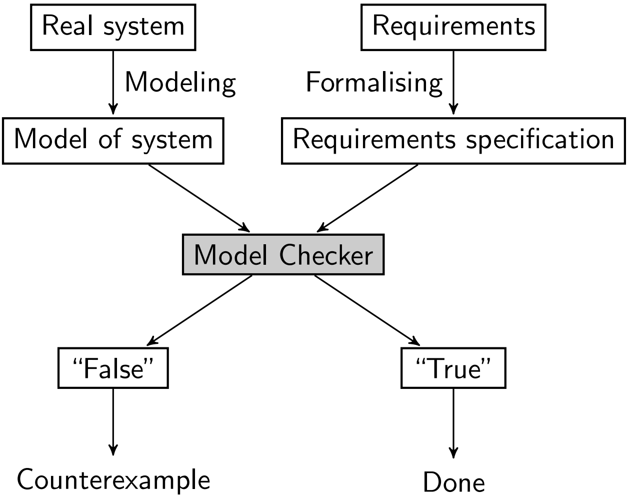
Andrew Jones
⟨avj05@doc.ic.ac.uk⟩

Supervisor: Dr. Alessio R. Lomuscio

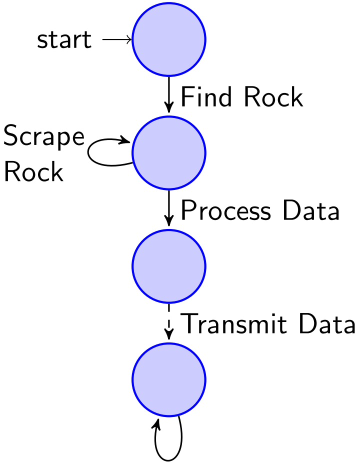
Department of Computing
Imperial College London

June, 2009

Teach Yourself Model Checking in 1 Minute



An Illustrative Scenario



The Mars Rover

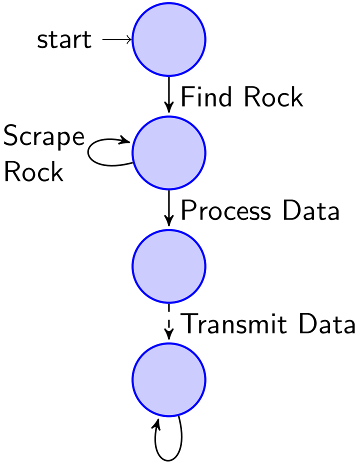
NASA want to ensure that their Mars Rovers *always* transmit data.

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

Specifying the Property

Assume that φ represents that the data has been transmitted

An Illustrative Scenario



The Mars Rover

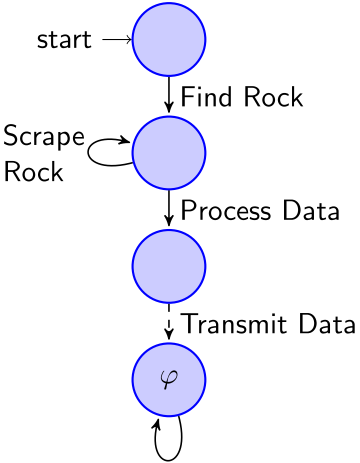
NASA want to ensure that their Mars Rovers *always* transmit data.

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

Specifying the Property

Assume that φ represents that the data has been transmitted

An Illustrative Scenario



The Mars Rover

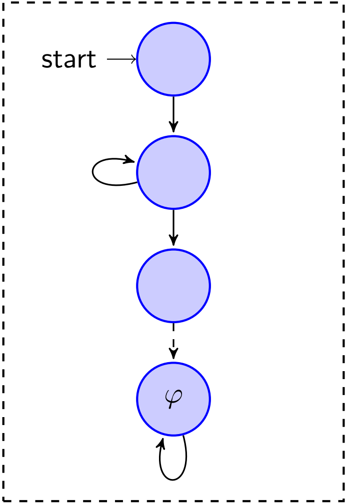
NASA want to ensure that their Mars Rovers *always* transmit data.

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

Specifying the Property

Assume that φ represents that the data has been transmitted

An Illustrative Scenario – "Global" Model Checking

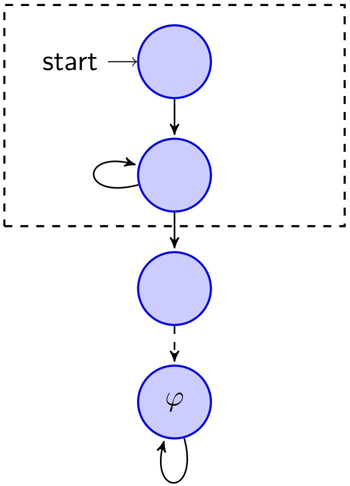


Original Approach

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

$$\mathcal{M}, \mathrm{start} \models AF(\varphi)$$

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?

$$\mathcal{M}, \text{start} \models EG(\neg\varphi)$$

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.

The, ahem, “million” dollar question

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

Multi-Agent Systems

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

Temporal Epistemic Interpreted Systems

- A system is composed of a set of agents $\mathrm{A} = \{1, \ldots, n\}$ and an environment e.
- Each agent, i, is described by
  - A set of local *states* – $\mathcal{L}_i$
  - A set of local *actions* – Act_i
  - A local *protocol* function – $\mathcal{P}_i : \mathcal{L}_i \to 2^{\mathrm{Act}_i}$
  - An *evolution* function – $\tau_i : \mathcal{L}_i \times \mathrm{Act} \to \mathcal{L}_i$
- Act is the set of *joint actions* –
  $\mathrm{Act} \subseteq \mathrm{Act}_1 \times \cdots \times \mathrm{Act}_n \times \mathrm{Act}_e$

Temporal Epistemic Interpreted Systems

- G is the set of *possible global states* – $\mathrm{G} \subseteq \mathcal{L}_1 \times \cdots \times \mathcal{L}_n \times \mathcal{L}_e$
    - A *global state* $g = (l_1, \ldots, l_n, l_e) \in \mathrm{G}$ represents a "snapshot" of the system
    - $l_i : \mathrm{G} \rightarrow \mathcal{L}_i$ 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 – $\mathrm{T} \subseteq \mathrm{G} \times \mathrm{Act} \times \mathrm{G}$
    - $g \, \mathrm{T} \, g'$ if there exists actions $a_1, \ldots, a_n$ such that for all i, $a_i \in \mathcal{P}_i(l_i(g))$ and $\tau_i(l_i(g), a_1, \ldots, a_n) = l_i(g')$

Temporal Epistemic Models

A *model* $\mathcal{M}_{\mathcal{IS}}$ is a tuple $(\mathrm{G}, \iota, \mathrm{T}, \sim_1, \ldots, \sim_n, \mathcal{V})$

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

A *path* $\pi = (\iota, g_1, \ldots)$ is a sequence of global states such that $(g_k, g_{k+1}) \in \mathrm{T}$.

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

A Temporal *Epistemic* Logic – CTLK

- Syntax

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid EX\varphi \mid EG\varphi \mid E\left[\varphi U \psi\right] \mid K_i\varphi$$

- Derived Modalities
  - $EF\varphi \stackrel{\text{def}}{=} E\left[trueU\varphi\right]$
  - $AX\varphi \stackrel{\text{def}}{=} \neg EX\neg\varphi$
  - $AF\varphi \stackrel{\text{def}}{=} \neg EG\neg\varphi$
  - $AG\varphi \stackrel{\text{def}}{=} \neg EF\neg\varphi$
  - $K_i\varphi \stackrel{\text{def}}{=} \neg\overline{K}_i\neg\varphi$
- $A\left[\varphi U \psi\right]$ – Takes the expected meaning

Epistemic Modalities – K and $\overline{K}$

$K_i\varphi$ – "agent i knows that φ"

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

$\overline{K}_i\varphi$ – "agent i considers it possible that φ"

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

Symbolic Model Checking

Conventional "symbolic" model checkers use a canonical representation called ROBDDs. These can be used to efficiently represent a boolean function.

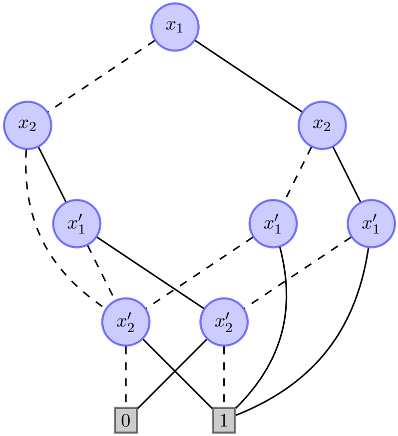
Two Main stages:

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

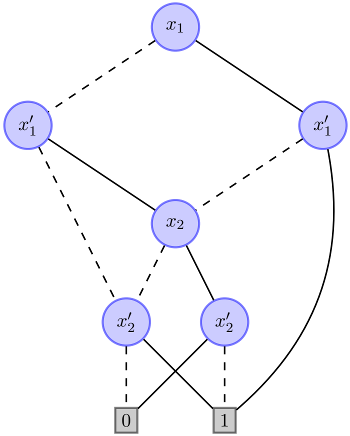
Interpreted Systems Reachable State Space

$$\mathrm{lfp}(Q) = (I(g) \vee \exists g' \, (T(g, a, g') \wedge Q(g')))$$

ROBDD Variable Ordering



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



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

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" formalisation.

CUDD `ExistAbstract`

- Quantification – $\mathcal{B}_h = \exists x \, \mathcal{B}_f$
- Shannon's expression – $h = (\neg x \wedge f|_{x \leftarrow 0}) \vee (x \wedge f|_{x \leftarrow 1})$

The Existential and Universal Fragments

ECTLK

CTLK as before:

- Restricts negation to *only* appear in front of elements of $\mathcal{PV}$.
- Contains $\overline{\mathrm{K}}_i$ *not* K_i

ACTLK

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

$$\{\varphi | \neg\varphi \in \mathrm{ECTLK}\}$$

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?

BDD-BMC(ψ : ACTLK Formula, $\mathcal{I}$: Initial State, Trans : Transition Relation) : Boolean

$\varphi \leftarrow \neg\psi$ $\{\varphi$: ECLTK Formula$\}$
Reach $\leftarrow \mathcal{I}$ $\{$Reach : BDD$\}$
while True **do**
    if $[\![\iota \rightarrow \varphi]\!]$ = Reach **then**
        return False $\{$*Counterexample found*$\}$
    end if
    Reach $\leftarrow$ Reach $\vee$ (Reach $\wedge$ Trans)
    if Reach *Unchanged* **then**
        break $\{$*Fixed point reached*$\}$
    end if
end while
return $[\![\iota \rightarrow \psi]\!]$ = Reach

Extending SAT_{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 $\overline{K}_i\varphi \stackrel{\text{def}}{=} \neg K_i \neg\varphi$ ...

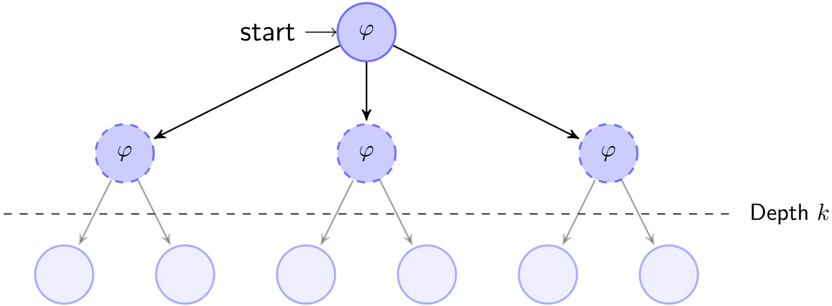
$\text{SAT}_{\overline{\text{K}}}(\varphi : \text{FORMULA}, i : \text{AGENT})$: *set of* STATE

1: $\text{X} \leftarrow \text{SAT}_{\text{CTLK}}(\varphi)$
2: $\text{Y} \leftarrow \text{pre}_{\text{K}}(\text{X}, i)$
3: **return** Y

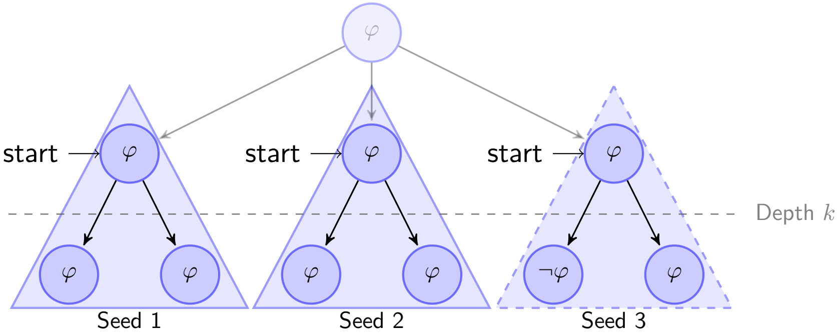
```
BDD basic_agent::project_local_state(BDD *state, BDDvector* v)
{
    BDD tmp = bddmgr->bddOne();

    // 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

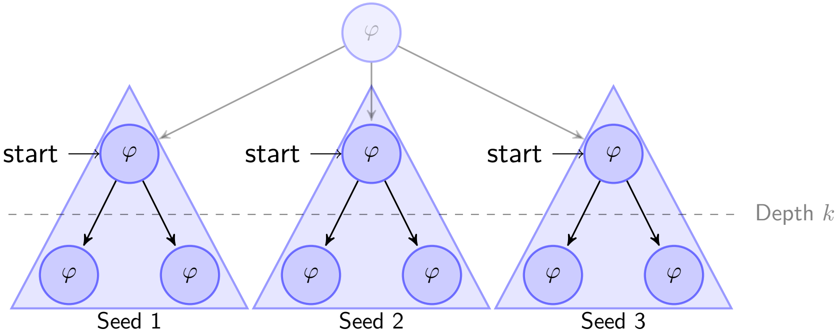
Distributed Verification of Invariant Properties



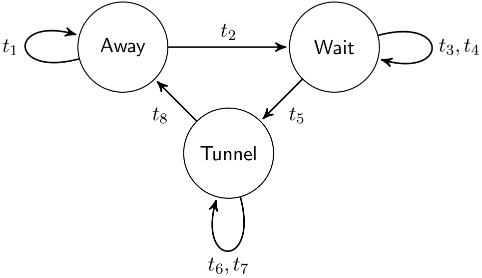
Distributed Verification of Invariant Properties



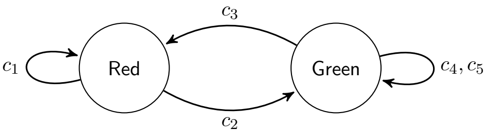
Distributed Verification of Invariant Properties



Faulty trains now have a **service counter** and a **breaking depth**. . .



Label	Action
t_1, t_3	SERVICE
t_2	BACK
t_4	SIGNAL
t_5	ENTER
t_6, t_7	BREAK
t_8	LEAVE



Label	Action
c_1	IDLE
c_2	EXIT_TRAIN
c_3	IDLE
c_4	IDLE
c_5	ENTER_TRAIN?

Initial Results

Model			Decrease	
T	M	B	Memory	States
2	10	4	0.9750	3.6318
2	10	6	0.9632	3.2970
3	10	4	0.1561	1.4815
4	10	2	0.2338	1.2672

Table: *vanilla* CUDD (φ_{TGC5})

Model			# Reorderings		# Garbage Collections	
T	M	B	Original	BDD-BMC	Original	BDD-BMC
2	10	4	6	11	10	16
2	10	6	12	9	17	11
3	10	4	13	55	27	69
4	10	2	17	26	30	61

Table: Reorderings and garbage collections performed by CUDD

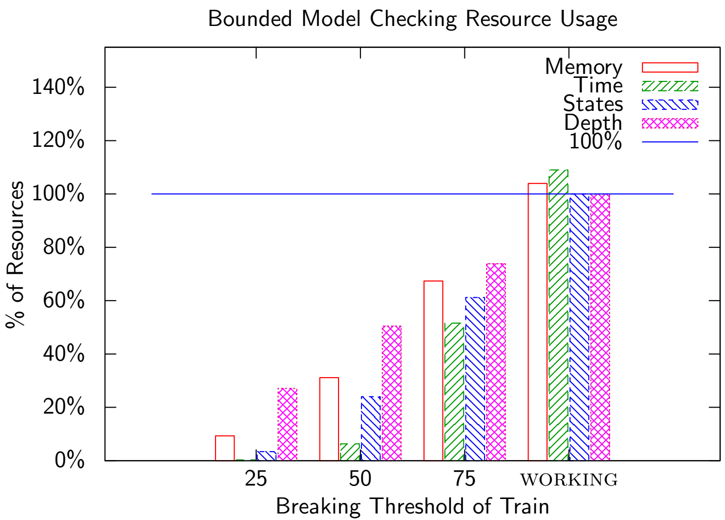


Figure: Resource use of BMC against regular model checking, with a model containing two type 1 trains, with a maximum depth of 100 and various breaking depths - φ_{TGC2}

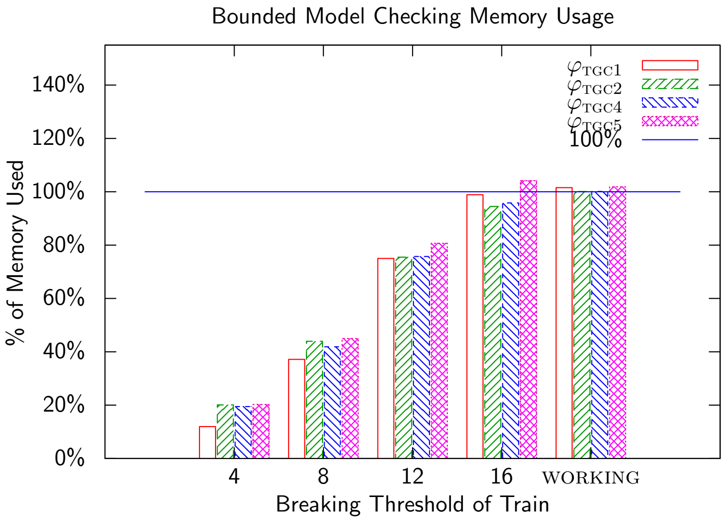


Figure: Memory usage for two *type 2* trains, with a full service depth of 20 – when checking various formulae

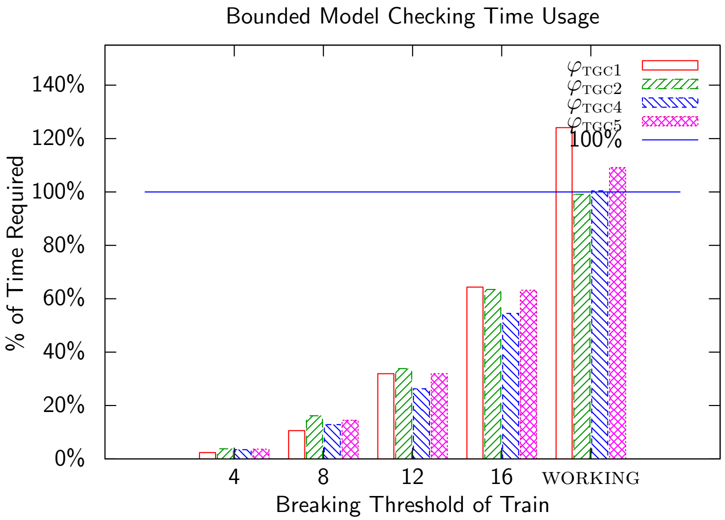


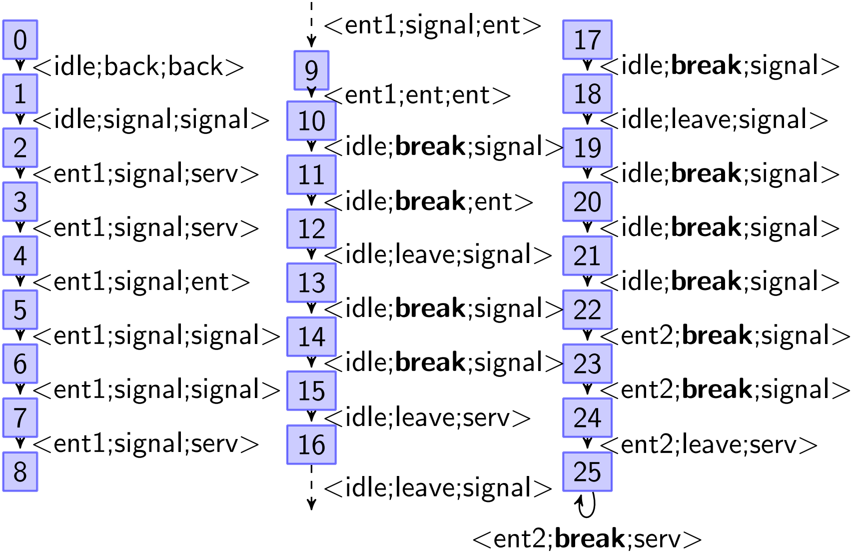
Figure: Time required for two *type 2* trains, with a full service depth of 20 – when checking various formulae

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".

Method	Formula				
	φ_{TGC1}	φ_{TGC2}	φ_{TGC3}	φ_{TGC4}	φ_{TGC5}
Regular	25	17	4	4	12
BMC	13	16	4	4	FAIL

Table: Length of counterexamples generated between BMC and full verification


0
<idle;back;back>
1
<idle;signal;signal>
2
<ent1;signal;serv>
3
<ent1;signal;serv>
4
<ent1;signal;ent>
5
<ent1;signal;signal>
6
<ent1;signal;signal>
7
<ent1;signal;serv>
8
<ent1;signal;ent>
9
<ent1;ent;ent>
10
<idle;**break**;signal>
11
<idle;**break**;ent>
12
<idle;leave;signal>
13
<idle;**break**;signal>
14
<idle;**break**;signal>
15
<idle;leave;serv>
16
<idle;leave;signal>
17
<idle;**break**;signal>
18
<idle;leave;signal>
19
<idle;**break**;signal>
20
<idle;**break**;signal>
21
<idle;**break**;signal>
22
<ent2;**break**;signal>
23
<ent2;**break**;signal>
24
<ent2;leave;serv>
25
<ent2;**break**;serv>

0
↓ <idle;back;back>
1
↓ <idle;signal;signal>
2
↓ <enter1;signal;service>
3
↓ <enter1;signal;service>
4
↓ <enter1;signal;service>
5
↓ <enter1;signal;signal>
6
⇣ <enter1;signal;signal>

7
↓ <enter1;signal;service>
8
↓ <enter1;signal;enter>
9
↓ <enter1;enter;enter>
10
↓ <idle;**break**;enter>
11
↓ <idle;**break**;service>
12
↓ <idle;leave;signal>
13

<idle;leave;enter>

Effectiveness of One-Shot BMC

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

B	Decrease	
	Memory	Time
5	1.00	0.50
10	1.78	1.24
15	1.70	0.63
WORKING	1.00	0.05

Table: 2 Trains, Max Counter 20

B	Decrease	
	Memory	Time
2	1.13	0.91
4	1.88	0.39
6	1.70	0.82
WORKING	1.00	0.14

Table: 3 Trains, Max Counter 7

Model	Decrease		
	Memory	Time	States
Type 1	1.033	0.772	1.002
Type 2	1.730	4.429	1.709
Type 3	0.907	0.005	0.003

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

# Hosts	Time	Decrease
2	118.88	0.016
4	58.78	0.032
6	39.93	0.048
8	30.37	0.063

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`

Questions?

Backup Slides

SAT requires an encoding of a "back loop"

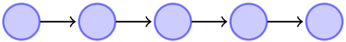
Semantics of ECTLK

$\mathcal{M}_{\mathcal{IS}}, g \models EG\varphi$ iff $\exists \pi \in \Pi(g)\ \forall_{m \geq 0}\ \mathcal{M}_{\mathcal{IS}}, \pi(m) \models \varphi$

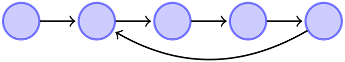
***Bounded* Semantics of ECLTK**

$\mathcal{M}_k, g \models EG\varphi$ iff $\exists \pi \in P_k(\pi(0) = g$ and $\forall_{0 \leq j \leq k}\ \mathcal{M}_k, \pi(j) \models \varphi)$

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



$loop(\pi) = \emptyset$



$loop(\pi) = \{2\}$

BDD approaches require an explict "error state"

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

$\text{Frontier}_0 \leftarrow \mathcal{I}$
for $(i = 0;\ i < k;\ i{+}{+})$ **do**
    if $(\text{Frontier}_i \cdot \text{Err} \neq \emptyset)$ **then**
        return $(\textsc{failure})$
    end if
    $\text{Frontier}_{i+1} \leftarrow \textsc{Img}(\text{Trans}, \text{Frontier}_i)$
end for
return $(\textsc{pass})$

One-Shot BMC

“ONE-SHOT” BMC(ψ : ACTLK FORMULA, $\mathcal{I}$: INITIAL STATE, Trans : TRANSITION RELATION, OneShotBound : INT) : STRING $\times$ BOOLEAN

$\varphi \leftarrow \neg\psi$ $\{\varphi$: ECLTK FORMULA$\}$
Reach $\leftarrow \mathcal{I}$ $\{$Reach : BDD$\}$
for $k \leftarrow 0$ to OneShotBound **do**
    Reach $\leftarrow$ Reach $\vee$ (Reach $\wedge$ Trans)
    if Reach *Unchanged* **then**
        return FIXED POINT CASE : $[\![\iota \rightarrow \psi]\!] =$ Reach
    end if
end for
return ONE SHOT CASE : $[\![\iota \rightarrow \varphi]\!] =$ Reach

Regular Specifications

φ_{TGC1}

$$AG\,(AF\,(\neg\textsc{Train}_1\textsc{\_in\_tunnel}))$$

φ_{TGC2} – "Mutual Exclusion"

$$AG\,(\neg\textsc{Train}_1\textsc{\_in\_tunnel} \vee \neg\textsc{Train}_2\textsc{\_in\_tunnel})$$

φ_{TGC3}

$$AG\,(\mathrm{K}_{\textsc{Train}_1}\,(\neg\textsc{Train}_1\textsc{\_in\_tunnel} \vee \neg\textsc{Train}_2\textsc{\_in\_tunnel}))$$

Parameterised Specifications

$\varphi_{\text{TGC4}}(\mathrm{N})$

$$AG \quad \left(\textsc{Train}_i\textsc{\_in\_tunnel} \rightarrow \mathrm{K}_{\textsc{Train}_i}\right.$$

$$\left(\quad \bigwedge_{j=1}^{i-1} \quad \left(\neg\textsc{Train}_j\textsc{\_in\_tunnel}\right) \wedge\right.$$

$$\left.\left.\left.\bigwedge_{j=i+1}^{N} \quad \left(\neg\textsc{Train}_j\textsc{\_in\_tunnel}\right)\right)\right)\right)$$

Parameterised Specifications

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

$$AG \quad \Big(\textsc{Train}_i\_\textsc{in\_tunnel} \rightarrow \mathrm{K}_{\textsc{Train}_i}$$

$$\Big(\quad \bigwedge_{j=1}^{i-1} \quad AX\Big(\neg\textsc{Train}_j\_\textsc{in\_tunnel} \Big) \wedge$$

$$\bigwedge_{j=i+1}^{N} \quad AX\Big(\neg\textsc{Train}_j\_\textsc{in\_tunnel} \Big)\Big)\Big)$$