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

Model Checking vs. Bounded Model Checking

- 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

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* –
  $\mathrm{Act} \subseteq \mathrm{Act}_1 \times \cdots \times \mathrm{Act}_n \times \mathrm{Act}_e$

- 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
  - $l_i : \mathrm{G} \to \mathcal{L}_i$ 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 – $\mathrm{T} \subseteq \mathrm{G} \times \mathrm{Act} \times \mathrm{G}$
  - $g \, \mathrm{T} \, g'$ 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* $\mathcal{M}_{\mathcal{IS}}$ 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 ($\mathcal{PV}$) – $\mathcal{V} : \mathrm{G} \rightarrow 2^{\mathcal{PV}}$

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

- $\pi(k)$ 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
  - $EF\varphi \stackrel{\text{def}}{=} E\left[trueU\varphi\right]$
  - $AX\varphi \stackrel{\text{def}}{=} \neg EX\neg\varphi$
  - $AF\varphi \stackrel{\text{def}}{=} \neg EG\neg\varphi$
  - $AG\varphi \stackrel{\text{def}}{=} \neg EF\neg\varphi$
  - $\overline{K}_i\varphi \stackrel{\text{def}}{=} \neg K_i\neg\varphi$
- $A\left[\varphi U \psi\right]$ – 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

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

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 $\mathcal{PV}$
- Contains $\overline{K}_i$ *not* K_i

ACTLK

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

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

Current Approaches to Bounded Model Checking

SAT-based BMC

Idea

- "Unroll" the transition relation k times
- Translate the **negated** property and unrolled model 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 of the error state with the "fringe"

Problem

How does one specify properties as a *single* error state? Okay for *invariant*/*safety* 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 $\textsc{Sat}_{\textsc{ctlk}}$

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

$\textsc{Sat}_{\overline{\mathrm{K}}}(\varphi : \textsc{Formula}, i : \textsc{Agent})$: *set of* $\textsc{State}$

1: $\mathrm{X} \leftarrow \textsc{Sat}_{\textsc{ctl}\overline{\mathrm{K}}}(\varphi)$
2: $\mathrm{Y} \leftarrow \mathrm{pre}_{\mathrm{K}}(\mathrm{X}, i)$
3: **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

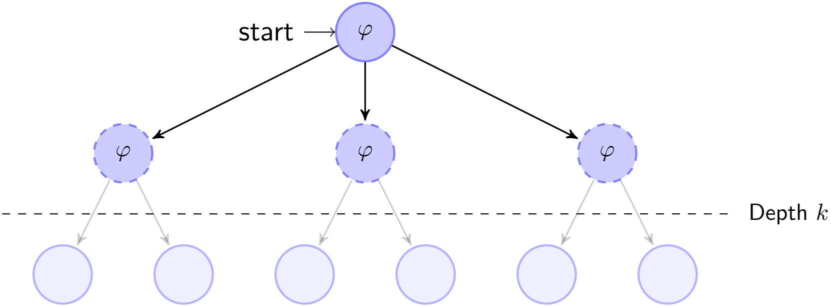
Distributed Verification of Invariant Properties

We restrict verification to invariant properties, i.e. "AG" is the top most connective.

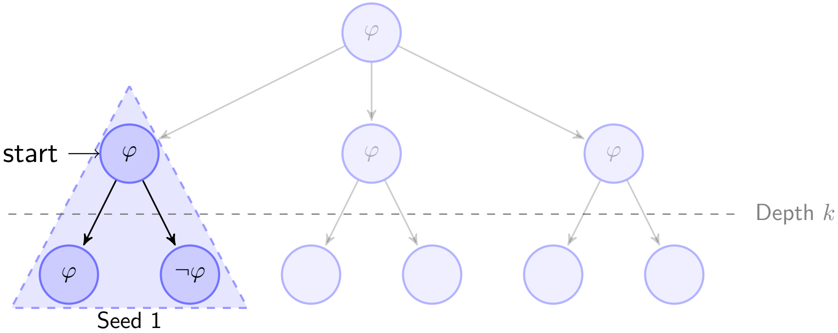
Distributed Approach

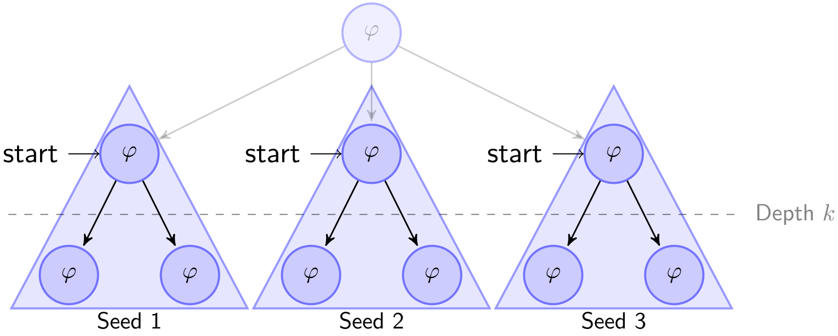
1. *Fixed-Depth BDD-based BMC*.
2. *Seed State Generation*.
3. *Distributed Parallel BDD-based BMC*.

Distributed Verification of Invariant Properties

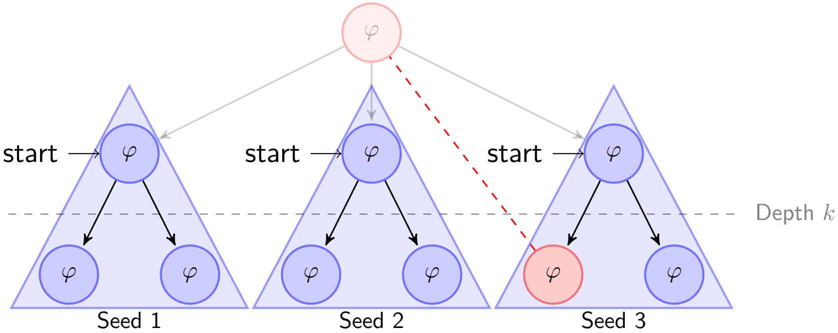


Distributed Verification of Invariant Properties



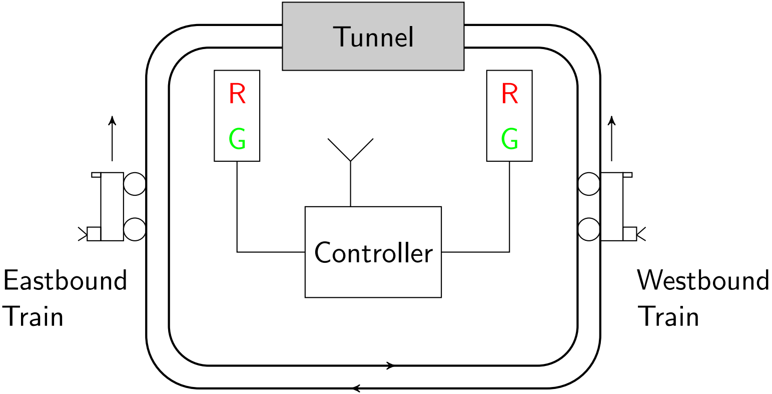

φ
start
φ
start
φ
start
φ
Depth k
φ
φ
φ
φ
φ
φ
Seed 1
Seed 2
Seed 3

Distributed Verification of Invariant Properties

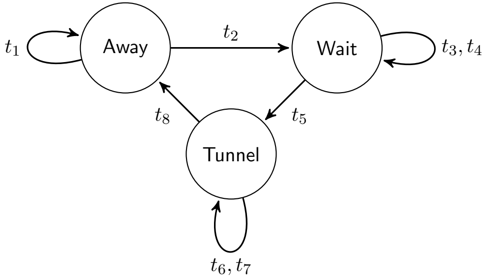


A Scalable Model

The Train Gate Controller Model

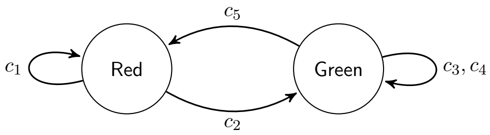


Faulty trains now have a **service counter** and a **breaking depth**...



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

The Faulty Train Gate Controller – `Controller`



Label	Action
c_1	IDLE
c_2	EXIT_TRAIN
c_3	IDLE
c_4	IDLE
c_5	ENTER_TRAIN_T T $\in$ {E,W}

Example Specifications

φ_{TGC1} : "Trains can't stay in the tunnel forever"

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

Mutual Exclusion:

φ_{TGC2} : "Two trains never occupy the tunnel at the same time"

$AG\,(\neg\textsc{TrainE\_in\_tunnel} \vee \neg\textsc{TrainW\_in\_tunnel})$

φ_{TGC3} :
"When a train is in the tunnel it knows that another train is not"

$AG\,(\textsc{TrainE\_in\_tunnel} \rightarrow K_{\textsc{TrainE}}\,(\neg\textsc{TrainW\_in\_tunnel}))$

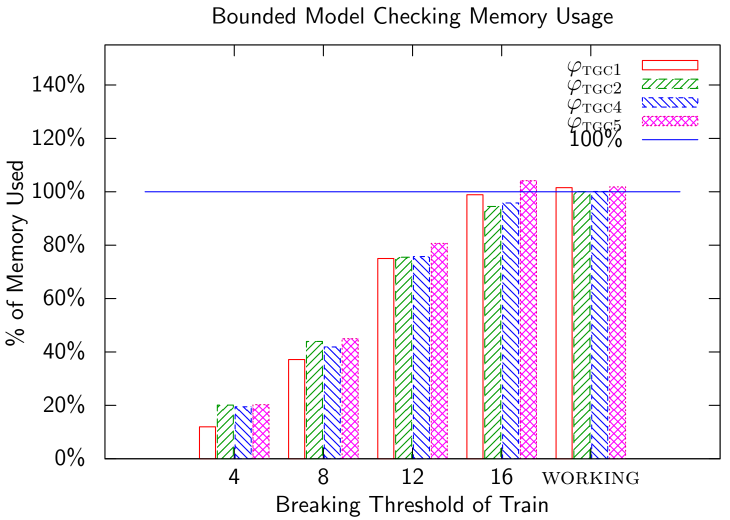


Figure: Memory usage for two trains, with a full service depth of 20 – when checking various formulae

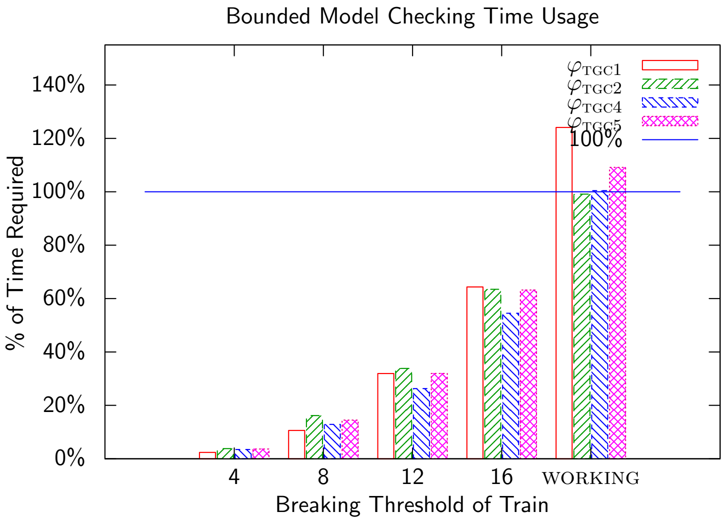
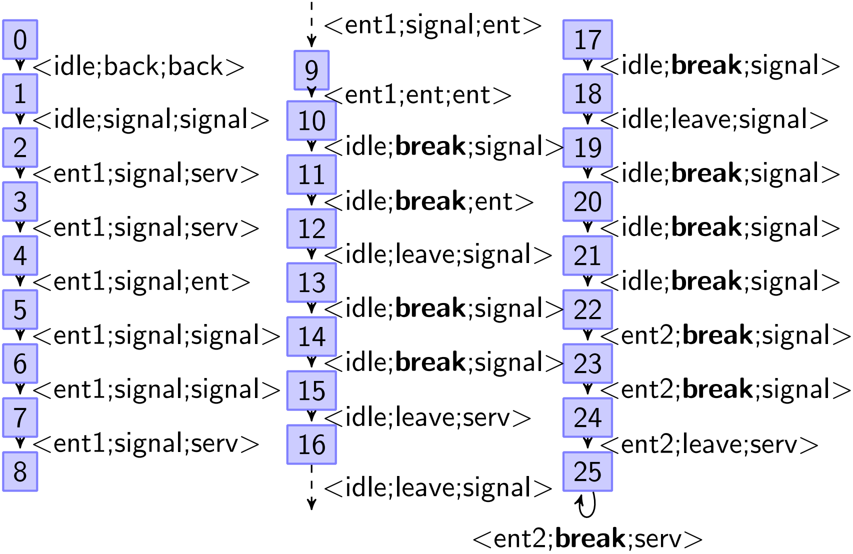


Figure: Time required for two trains, with a full service depth of 20 – when checking various formulae

- Biere *et al.* (1999): bounded model checking "finds counterexamples of minimal length".

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>

A Temporal Formula

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).

Model	Ratio		
	Memory	Time	States
FAULTY	1.8255	3.8130	1.7297
WORKING	0.9500	0.0013	0.0008

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

# Hosts	Ratio
2	0.0033
4	0.0066
6	0.0098
8	0.0131

Conclusion

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

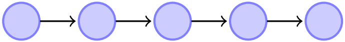
SAT requires an encoding of a “back loop”

Semantics of ECTLK

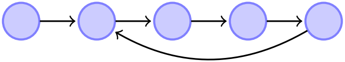
$\mathcal{M}_{\mathcal{IS}}, g \vDash EG\varphi$ iff $\exists \pi \in \Pi(g)\ \forall_{m \geq 0}\ \mathcal{M}_{\mathcal{IS}}, \pi(m) \vDash \varphi$

Bounded **Semantics of ECLTK**

$\mathcal{M}_k, g \vDash EG\varphi$ iff $\exists \pi \in P_k(\pi(0) = g$ and $\forall_{0 \leq j \leq k}\ \mathcal{M}_k, \pi(j) \vDash \varphi)$
... *requires* $loop(\pi) \neq \emptyset$



$loop(\pi) = \emptyset$



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

BDD approaches require an explicit "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 (FAILURE)
    end if
    $\text{Frontier}_{i+1} \leftarrow \text{Img}(\text{Trans}, \text{Frontier}_i)$
end for
return (PASS)

CUDD `ExistAbstract`

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

Parameterised Specifications

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

$$AG\left(\textsc{Train}_i\textsc{\_in\_tunnel} \rightarrow \text{K}_{\textsc{Train}_i}\left(\bigwedge_{j=1}^{i-1}\left(\neg\textsc{Train}_j\textsc{\_in\_tunnel}\right) \wedge \bigwedge_{j=i+1}^{N}\left(\neg\textsc{Train}_j\textsc{\_in\_tunnel}\right)\right)\right)$$

Parameterised Specifications

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

$$
\begin{array}{rl}
AG & \Big(\textsc{Train}_i\text{\_IN\_TUNNEL} \rightarrow \mathrm{K}_{\textsc{Train}_i} \\
\Big(\bigwedge_{j=1}^{i-1} & AX\Big(\neg\textsc{Train}_j\text{\_IN\_TUNNEL}\Big) \wedge \\
\bigwedge_{j=i+1}^{N} & AX\Big(\neg\textsc{Train}_j\text{\_IN\_TUNNEL}\Big)\Big)\Big)
\end{array}
$$