Scientific Issues
Jan Długosz University
in Częstochowa
Mathematics XXII (2017)
161–175
DOI http://dx.doi.org/10.16926/m.2017.22.12

COMPARING SAT- AND SMT- BASED BOUNDED MODEL CHECKING FOR ECTL PROPERTIES

AGNIESZKA M. ZBRZEZNY

Abstract

We compare two bounded model checking methods for properties expressed in the existential fragment of Computation Tree Logic (ECTL). To this end we use the generic pipeline paradigm (GPP) and the train controller system (TC), the classic concurrency problems, which we formalise by means of a finite transition system. We consider several properties of the problems that can be expressed in ECTL logic, and we present the performance evaluation of the mentioned bounded model checking methods by means of the running time and the memory used.

1. Introduction

Bounded model checking [2, 3, 9] (BMC) is one of the symbolic model checking techniques designed for finding witnesses for existential properties or counterexamples for universal properties. Its main idea is to consider a model reduced to a specific depth. The method works by mapping a bounded model checking problem to the satisfiability problem (SAT) or to satisfiability modulo theories problem (SMT). For the existential part of Computation Tree Logic (ECTL) [6] and a network of automata the BMC method can by described as follows: given a model $\mathcal{M}$ for a network of automata, an ECTL formula φ, and a bound k, a model checker creates a propositional formula or a quantifier-free first-order formula $[\mathcal{M}, \varphi]_k$ that is satisfiable if and only if the formula φ is true in the model $\mathcal{M}$.

In this paper, we make the following contributions. Firstly, we define and implement an SMT-based BMC method for ECTL and for a network of automata modelled using transition systems. Next, we report on the

• *Agnieszka M. Zbrzezny* — e-mail: agnieszka.zbrzezny@ajd.czest.pl
Jan Długosz University in Częstochowa.

Partly supported by National Science Centre under the grant No. 2014/15/N/ST6/05079.

initial experimental evaluation of our SMT-based BMC method. To this aim, we use two scalable benchmarks: the *generic pipeline paradigm* [8] and the *train controller system* [7].

The structure of the paper is as follows. In Section 2 we shortly recall definition of transition systems. Then, we present syntax and semantics of ECTL. In Section 3 we shortly present BMC technique for ECTL. and we define a new SMT-based BMC method for ECTL. In Section 4 we present experimental evaluation of the SAT-based BMC [12] and SMT-based BMC for ECTL and for a train controller system (TC) and generic pipeline paradigm (GPP). In Section 5 we conclude the paper.

2. Preliminaries

In this section we introduce the basic definitions used in the paper. In particular, we define the semantics of a transition system (TS) and syntax and semantics of ECTL.

2.1. **Transition System.** The transition system [1] (also called a *model*) is a tuple

$$\mathcal{M} = (S, Act, \longrightarrow, s^0, \mathcal{AP}, L) \text{ where:}$$

S is a set of states, Act is a set of actions, $\longrightarrow \subseteq S \times Act \times S$ is a transition relation, $s^0 \in S$ is the initial state, $\mathcal{AP}$ is a set of atomic propositions, and $L : S \to 2^{\mathcal{AP}}$ is a labelling function assigning to each state a set of atomic propositions that are assumed to be true at that state. The transition system is called finite if S, Act, and $\mathcal{AP}$ are finite. We write $s \xrightarrow{a} s'$ instead of $(s, a, s) \in \longrightarrow$. Moreover, we write $s \longrightarrow s'$ if $s \xrightarrow{a} s'$, for some $a \in Act$.

We assume that a considered model has no terminal states, i.e. for every $s \in S$ there exist $s' \in S$ such that $s \longrightarrow s'$. The set of all natural numbers is denoted by $\mathbb{N}$. A *path* in $\mathcal{M}$ is an infinite sequence $\pi = (s_0, s_1, \ldots)$ of states such that $s_i \longrightarrow s_{i+1}$ for each $i \in \mathbb{N}$. For a path $\pi = (s_0, s_1, \ldots)$ and $i \in \mathbb{N}$, the i-th state of π is defined as $\pi(i) = s_i$. By $\Pi(s)$ we denote the set of all the paths starting at $s \in S$.

2.2. **ECTL.** The Existential Computation Tree Logic is a restriction of a propositional branching-time temporal logic CTL that was introduced by Emerson and Clarke in [6]. The syntax of ECTL formulae over the set $\mathcal{AP}$ of atomic propositions is defined by the following grammar: $\varphi := \mathbf{true} \mid \mathbf{false} \mid p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{EX}\varphi \mid \mathbf{E}(\varphi \mathbf{U} \varphi) \mid \mathbf{EG}\varphi$, where $p \in \mathcal{AP}$ and φ is a formula. The symbols $\mathbf{X}$, $\mathbf{U}$ and $\mathbf{G}$ are the modal operators for "neXt time", "Until" and "Globally", respectively. The symbol $\mathbf{E}$ is the existential path quantifier. The derived basic modalities are defined as follows:

$$\mathbf{EF}\alpha \stackrel{df}{=} \mathbf{E}(\mathbf{true}\,\mathbf{U}\,\alpha), \qquad \mathbf{E}(\alpha\,\mathbf{R}\,\beta) \stackrel{df}{=} \mathbf{E}(\beta\,\mathbf{U}\,(\alpha \wedge \beta)) \vee \mathbf{EG}\beta.$$

Let $\mathcal{M}$ be a model, and φ an ECTL formula. An ECTL formula φ is *true* in the model $\mathcal{M}$ (in symbols $\mathcal{M} \models \varphi$) iff $\mathcal{M}, s^0 \models \varphi$ (i.e., φ is true at the initial state of the model $\mathcal{M}$), where

$\mathcal{M}, s \models \mathbf{true}$,

$\mathcal{M}, s \not\models \mathbf{false}$,

$\mathcal{M}, s \models p$ iff $p \in L(s)$,

$\mathcal{M}, s \models \neg p$ iff $p \notin L(s)$,

$\mathcal{M}, s \models \alpha \wedge \beta$ iff $\mathcal{M}, s \models \alpha$ and $\mathcal{M}, s \models \beta$,

$\mathcal{M}, s \models \alpha \vee \beta$ iff $\mathcal{M}, s \models \alpha$ or $\mathcal{M}, s \models \beta$,

$\mathcal{M}, s \models \mathbf{EX}\alpha$ iff $(\exists \pi \in \Pi(s))(\mathcal{M}, \pi(1) \models \alpha)$,

$\mathcal{M}, s \models \mathbf{E}(\alpha \mathbf{U} \beta)$ iff $(\exists \pi \in \Pi(s))(\exists m \geq 0)(\mathcal{M}, \pi(m) \models \beta$ and $(\forall j < m)\mathcal{M}, \pi(j) \models \alpha)$,

$\mathcal{M}, s \models \mathbf{EG}\alpha$ iff $(\exists \pi \in \Pi(s))(\forall m \geq 0)(\mathcal{M}, \pi(m) \models \alpha)$.

3. Bounded Model Checking

The satisfiability modulo theories (SMT) problem is a decision problem for logical formulas with respect to combinations of background theories expressed in classical first-order logic with equality. The SMT-based Bounded Model Checking (BMC) is a popular model checking technique for the verification of concurrent systems. We have given a model $\mathcal{M}$, an existential modal formula φ, and a non-negative bound k, the SMT-based BMC consists in searching for a non-empty set of paths of length k that constitute a witness for the checked property φ. In particular, the SMT-based bounded model checking algorithms generate a quantifier-free first-order formula which is satisfiable if and only if the mentioned set of paths exist. The quantifier-free first-order formula is usually obtained as a combination of an SMT encoding of the unfolding of the transition relation of the given model, and an SMT encoding of the property in question. If the generated quantifier-free first-order formula is not satisfiable, then k is incremented until either the problem becomes intractable due to the complexity of solving the corresponding SAT instance, or k reaches the upper bound of the bounded model checking problem for the language under consideration.

We have implemented a translation to SMT strictly following the translation to SAT given in [12].

Since $\mathcal{M}$ is a parallel composition of a finite number n of finite transition systems, every state of $\mathcal{M}$ can be encoded as a natural number vector of the length n. Thus, each state of $\mathcal{M}$ can be represented by a valuation

of a vector (called a *symbolic state*) of different individual variables called *individual state variables*. Moreover, every action of $\mathcal{M}$ can be represented by a valuation of an individual variable, and the designated positions l of the k-paths used in the translation can be also be represented by valuations of individual variables. Furthermore, k-paths can be represented as vectors of symbolic states.

The SAT-based BMC method for ECTL was introduced in [9], and then it was improved in [12]. Unfortunately the encoding presented in [12] does not encode actions In our new SMT-based approach we encode actions as well.

3.1. **SMT-based bounded model checking.** Let $\mathcal{M}$ be a model, $k \geq 0$ a bound, φ an ECTL formula, and let $\mathcal{M}, s \models_k \varphi$ denotes that φ is k-true at the state s of $\mathcal{M}$. The formula φ is k-true in $\mathcal{M}$ (in symbols $\mathcal{M} \models_k \varphi$) iff $\mathcal{M}, s^0 \models_k \varphi$ (i.e., φ is k-true at the initial state of the model $\mathcal{M}$).

The *bounded model checking problem* asks whether there exists $k \in \mathbb{N}$ such that $\mathcal{M} \models_k \varphi$. The following theorem states that for a given model and an ECTL formula there exists a bound k such that the model checking problem ($\mathcal{M} \models \varphi$) can be reduced to the BMC problem ($\mathcal{M} \models_k \varphi$). The theorem can be proven by induction on the length of the formula φ.

Theorem 1 ([12])**.** *Let $\mathcal{M}$ be a model and φ an* ECTL *formula. Then, the following equivalence holds: $\mathcal{M} \models \varphi$ iff there exists $k \geq 0$ such that $\mathcal{M} \models_k \varphi$.*

Translation to SMT. The translation to SMT is based on the bounded semantics. Let $\mathcal{M}$ be a model, φ an ECTL formula, and $k \geq 0$ a bound. The presented SMT encoding of the BMC problem for ECTL is based on the SAT encoding of the same problem [12], and it relies on defining the quantifier-free first-order formula $[\mathcal{M}, \varphi]_k := [\mathcal{M}^{\varphi, s^0}]_k \wedge [\varphi]_{\mathcal{M},k}$ that is satisfiable if and only if $\mathcal{M} \models_k \varphi$ holds.

The definition of the formula $[\mathcal{M}^{\varphi,s^0}]_k$ assumes that states of the model $\mathcal{M}$ are encoded in a symbolic way. Such a symbolic encoding is possible, since the set of states of $\mathcal{M}$ is finite. In particular, each state s can be represented by a vector $\overline{\mathbf{w}}$ (called a *symbolic state*) of different individual variables ranging over the natural numbers (called *individual state variables*) and each action can be represented by a valuation of a symbolic action $\overline{\mathbf{a}}$ that is a vector of the individual variables ranging over the natural numbers. The formula $[\mathcal{M}^{\varphi,s^0}]_k$ encodes a rooted tree of $k-$paths of the model $\mathcal{M}$. The number of branches of the tree depends on the value of the auxiliary function $f_k : \text{ECTL} \to \mathbb{N}$ defined in [10].

Given the above, the j-th symbolic k-path $\boldsymbol{\pi}_j$ is defined as the following sequence $(\overline{\mathbf{w}}_{0,j}, \ldots, \overline{\mathbf{w}}_{k,j})$, where $\overline{\mathbf{w}}_{i,j}$ are symbolic states for $0 \leq i \leq k$ and $0 \leq j < f_k(\varphi)$.

Let $\overline{\mathbf{w}}$ and $\overline{\mathbf{w}}'$ be two different symbolic states. We assume definitions of the following auxiliary quantifier-free first-order formulae: $I_{s^0}(\overline{\mathbf{w}})$ - encodes the initial state of the model $\mathcal{M}$, $\mathcal{T}(\overline{\mathbf{w}}, \overline{\mathbf{a}}, \overline{\mathbf{w}}')$ - encodes the transition relation of $\mathcal{M}$, and $p(\overline{\mathbf{w}})$ - encodes the set of states of $\mathcal{M}$ in which $p \in \mathcal{AP}$ holds.

The formula $[\mathcal{M}^{\varphi,s^0}]_k$ encoding the unfolding of the transition relation of the model $\mathcal{M}$ $f_k(\varphi)$-times to the depth k is defined as follows:

$$[\mathcal{M}^{\varphi,s^0}]_k := I_{s^0}(\overline{\mathbf{w}}_{0,0}) \wedge \bigwedge_{j=0}^{f_k(\varphi)-1} \bigwedge_{i=0}^{k-1} \mathcal{T}(\overline{\mathbf{w}}_{i,j}, \overline{\mathbf{a}}_{i,j} \overline{\mathbf{w}}_{i+1,j}) \tag{1}$$

For every ECTL formula φ the function f_k determines how many symbolic k-paths are needed for translating the formula φ. Given a formula φ and a set A of k-paths such that $|A| = f_k(\varphi)$, we divide the set A into subsets needed for translating the subformulae of φ. To accomplish this goal we need the auxiliary functions $h_n^{\mathbf{U}}(A, e)$ and $h_n^{\mathbf{R}}(A, e)$ that were defined in [12].

Below we show the translation for the temporal operators **EX**, **EU** and **EG** only.

Let φ be an ECTL formula, $\mathcal{M}$ a model, and $k \in \mathbb{N}$ a bound. The quantifier-free first-order formula $[\varphi]_{\mathcal{M},k} := [\varphi]_k^{[0,0,F_k(\varphi)]}$, where $F_k(\varphi) = \{j \in \mathbb{N} \mid 0 \leqslant j < f_k(\varphi)\}$, encodes the bounded semantics for ECTL, and it is defined inductively as shown below. Namely, let $0 \leqslant n < f_k(\varphi)$, $m \leqslant k$, $n' = min(A)$, $h_{\mathbf{X}} = h_{\mathbf{X}}(A)$, $h_k^{\mathbf{U}} = h_k^{\mathbf{U}}(A, f_k(\beta))$, and $h_k^{\mathbf{G}} = h_k^{\mathbf{G}}(A, f_k(\alpha))$, then:

$$[\mathbf{EX}\alpha]_k^{[m,n,A]} := \overline{\mathbf{w}}_{m,n} = \overline{\mathbf{w}}_{0,n'} \wedge [\alpha]_k^{[1,n',h_{\mathbf{X}}]},$$

$$[\mathbf{EG}\alpha]_k^{[m,n,A]} := \overline{\mathbf{w}}_{m,n} = \overline{\mathbf{w}}_{0,n'} \wedge L_k(n') \wedge \bigwedge_{j=0}^{k} [\alpha]_k^{[j,n',h_{\mathbf{G}}(j)]},$$

$$[\mathbf{E}(\alpha\,\mathbf{U}\,\beta)]_k^{[m,n,A]} := \overline{\mathbf{w}}_{m,n} = \overline{\mathbf{w}}_{0,n'} \wedge \bigvee_{i=0}^{k} \left([\beta]_k^{[i,n',h_{\mathbf{U}}(k)]} \wedge \bigwedge_{j=0}^{i-1} [\alpha]_k^{[j,n',h_{\mathbf{U}}(j)]} \right).$$

Theorem 2. *Let $\mathcal{M}$ be a model, and φ an* ECTL *formula. Then for every $k \in \mathbb{N}$, $M \models_k \varphi$ if, and only if, the formula $[\mathcal{M}, \varphi]_k$ is satisfiable.*

3.2. **Example.** Now we show how to apply our SMT-based BMC method to verify the *generic pipeline paradigm* (GPP) TS model. The model of GPP involves $n + 2$ automata: Producer producing data or being inactive, Consumer receiving data or being inactive, and a chain of n intermediate Nodes which can be ready for receiving data, processing data or sending data.

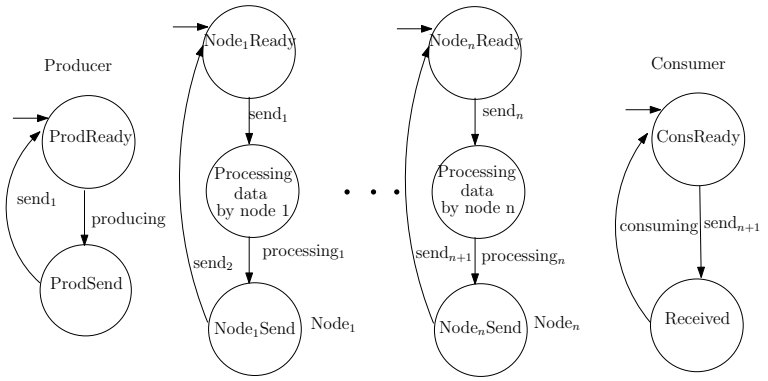


FIGURE 1. The GPP system [11]

As an example we consider the system with one node. We will show the important parts of the unfolding (to the depth 2) of the transition relation and also the translation of the temporal formula $\varphi_5 = \mathbf{EF}(ProdSend \wedge Received)$ on the k-path of the length 2. To this aim we need some auxiliary notations.

Let w denotes the symbolic global state and a denotes the symbolic action. We encode the location $ProdReady$ by 0 and the location $ProdSend$ by 1; the locations $Node_1Ready$ is encoded by 0, the location $Node_1Proc$ by 1, and the location $Node_1Send$ by 2; eventually the location $ConsReady$ is encoded by 0, and the location $ConsReceived$ by 1.

Moreover, we encode the action $Produce$ by 1, the action $Send_1$ by 2, the action $Proc_1$ by 4, the action $Send_2$ by 3, and the action $Consume$ by 5.

Now we are ready to present some of the formulae involved in the unfolding of the transition relation and in the translation of the formula φ.

The only initial state is represented by the following formula: $a_0 = 0 \wedge w_0 = 0 \wedge w_1 = 0 \wedge w_2 = 0$.

The SMT file for the GPP system and the formula φ_5 generated by our implementation is following:

```
(set-logic QF_LIA)
```

```
(declare-fun w0 () Int)
(declare-fun w1 () Int)
(declare-fun w2 () Int)
(declare-fun a0 () Int)
(declare-fun w3 () Int)
(declare-fun w4 () Int)
(declare-fun w5 () Int)
(declare-fun a3 () Int)
(declare-fun w6 () Int)
(declare-fun w7 () Int)
(declare-fun w8 () Int)
(declare-fun a6 () Int)

; Path nr 0: first state
(assert true)

(assert (and (or (and (= w0 0) (= w3 1) (= a3 1)) (and (= w0 1)
(= w3 0) (= a3 2)) (and (not (= a3 1)) (not (= a3 2))
(= w0 w3))) (or (and (= w1 0) (= w4 1) (= a3 2)) (and (= w1 2)
(= w4 0) (= a3 3)) (and (= w1 1) (= w4 2) (= a3 4))
(and (not (= a3 2)) (not (= a3 3)) (not (= a3 4)) (= w1 w4)))
(or (and (= w2 0) (= w5 1) (= a3 3)) (and (= w2 1) (= w5 0)
(= a3 5)) (and (not (= a3 3)) (not (= a3 5)) (= w2 w5)))
(or (= a3 1) (= a3 2) (= a3 3) (= a3 4) (= a3 5))))

(assert (and (or (and (= w3 0) (= w6 1) (= a6 1)) (and (= w3 1)
(= w6 0) (= a6 2)) (and (not (= a6 1)) (not (= a6 2)) (= w3 w6)))
(or (and (= w4 0) (= w7 1) (= a6 2)) (and (= w4 2) (= w7 0)
(= a6 3)) (and (= w4 1) (= w7 2) (= a6 4)) (and (not (= a6 2))
(not (= a6 3)) (not (= a6 4)) (= w4 w7))) (or (and (= w5 0)
(= w8 1) (= a6 3)) (and (= w5 1) (= w8 0) (= a6 5))
(and (not (= a6 3)) (not (= a6 5)) (= w5 w8)))
(or (= a6 1) (= a6 2) (= a6 3) (= a6 4) (= a6 5))))

; Translated formula
(assert (and (or (and (= w0 1) (= w2 1)) (and (= w3 1) (= w5 1))
(and (= w6 1)(= w8 1))) (and (= w0 w0) (= w1 w1) (= w2 w2))))

; Initial path: 0
(assert (and (= a0 0) (= w0 0) (= w1 0) (= w2 0)))

(check-sat)
(get-model)
```

The corresponding DIMACS file is following:

```
p cnf 83 222
-80 81 82 0
-77 82 83 0
-74 -76 77 0
-70 -71 72 0
-69 72 73 0
-66 -67 68 0
-64 68 69 0
-61 -62 63 0
-59 -60 61 0
-57 63 64 0
-52 -54 55 0
-49 -62 67 0
-49 -55 56 0
-48 -62 71 0
-48 -49 50 0
-46 -47 48 0
-45 -50 51 0
-41 -42 43 0
-40 43 44 0
-37 -38 39 0
-35 -58 59 0
-35 -36 37 0
-34 39 40 0
-31 -32 33 0
-29 -30 31 0
-27 -53 54 0
-27 -28 29 0
-26 33 34 0
...
```

4. Experimental Results

In this section we experimentally evaluate the performance of our SMT-based BMC encoding for Ectl over the TS semantics. We compare our experimental results with the experimental results generated using SAT-based [12] We have conducted the experiments using two benchmarks: the generic pipeline paradigm (GPP) TS model [8] and the train controller system (TCS) TS model [7]. We would like to point out that both benchmarks are very useful and scalable examples.

An evaluation of both BMC algorithms, which have been implemented in C++ is given by means of the running time, the memory used, and the number of generated variables and clauses.

We would like to point out that both benchmarks are very useful and scalable examples.

4.1. **A Train Controller System.** To evaluate the BMC techniques for ECTL, we analyse a scalable concurrent system, which is a train controller system (TCS). The system consists of a controller, and n trains (for $n \geq 2$), and it is assumed that each train uses its own circular track for travelling in one direction. All trains have to pass through a tunnel, but because there is only one track in the tunnel, arriving trains cannot use it simultaneously. There are signal lights on both sides of the tunnel, which can be either red or green. All trains notify the controller when they request entry to the tunnel or when they leave the tunnel. The controller controls the colour of the signal lights

An automata model of the TCS system is shown on Figure 2.

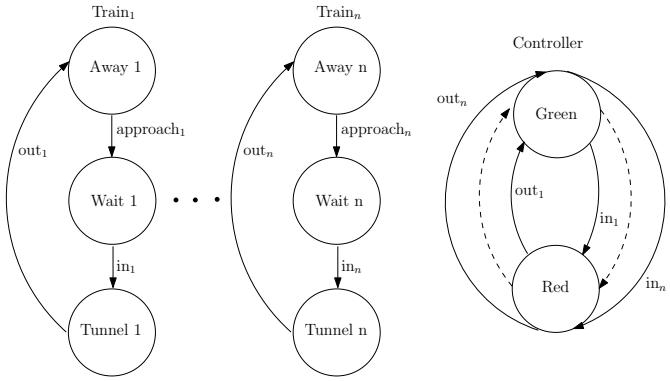


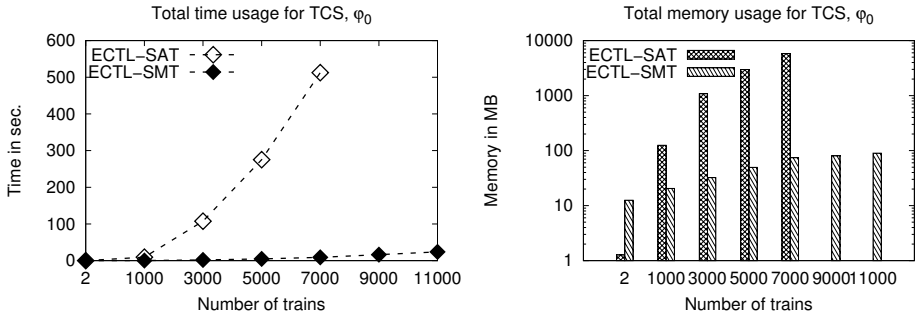
FIGURE 2. A network of automata for a train controller system ([11])

The specifications for it are given in the existential form, i.e., they are expressed in the ECTL language.

Formula φ_0 states that there exists the case that all trains are outside the tunnel and Train n is in the tunnel, where n is the number of trains.

$$\varphi_0 = \mathbf{EF}\big((\bigwedge_{i=1}^{n-1} \neg InTunnel_i) \wedge InTunnel_n\big).$$

In Figures 3(a) and 3(b) we present a comparison of total time usage and total memory usage for the formulae φ_0. As we can see, in this case SMT-BMC is much better than SAT approach. The reason is that we need only one symbolic to verify the formula.



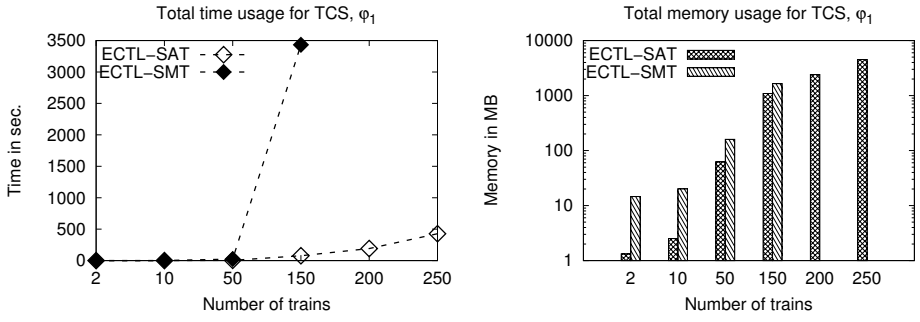
(a) (b)

FIGURE 3. A comparison of total time usage and total memory usage for the formulae φ_0

Formula φ_1 states that there exists the case that Train 1 is in the tunnel and either it and other train will not be in the tunnel during the next $n+1$ steps, where n is the number of trains.

$$\varphi_1 = \mathbf{EF}(InTunnel_1 \wedge \mathbf{EX}(\bigwedge_{i=1}^{n} \neg InTunnel_i \wedge \underbrace{\mathbf{EX}(\bigwedge_{i=1}^{n} \neg InTunnel_i \wedge \mathbf{EX}(\bigwedge_{i=1}^{n} \neg InTunnel_i \ldots))}_{n})).$$

In Figures 4(a) and 4(b) we present a comparison of total time usage and total memory usage for the formulae φ_1.



(a) (b)

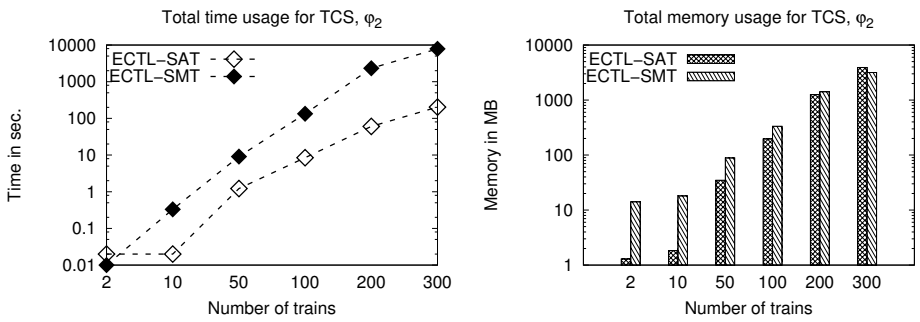
FIGURE 4. A comparison of total time usage and total memory usage for the formulae φ_1

An analysis of experimental results for formula φ_1 leads to the conclusion that SAT-based BMC for ECTL uses less time and memory comparing to SMT-based BMC for ECTL. The reason is that although BMC needs a lot of paths for verification, but these paths are short. The SAT-based algorithm was not able to verify the system with 300 on lack of memory. In the case of SAT-based BMC, generation of DIMACS file consume most of the memory and time.

Formula φ_2 expresses that there exists the case that Train 1 is in the tunnel or either it or other train will not be in the tunnel during the next $n+1$ steps, where n is the number of trains.

$$\varphi_2 = \mathbf{EF}(InTunnel_1 \vee \mathbf{EX}(\bigwedge_{i=1}^{n} \neg InTunnel_i \vee \underbrace{\mathbf{EX}(\bigwedge_{i=1}^{n} \neg InTunnel_i \vee \mathbf{EX}(\bigwedge_{i=1}^{n} \neg InTunnel_i \ldots))}_{n}).$$

In Figures 5(a) and 5(b) we present a comparison of total time usage and total memory usage for the formulae φ_2.



(a) (b)

FIGURE 5. A comparison of total time usage and total memory usage for the formulae φ_2.

An observation of experimental results for formula φ_2 leads to the conclusion that SAT-BMC for ECTL uses less time and memory comparing to SMT-based BMC for ECTLand SAT-based BMC is significantly faster than SMT-based approach. In this case almost all memory was used by SMT- and SAT-solvers. The reason is that BMC needs a lot of short paths for verification. In this case SAT-based BMC is usually better than SMT-based BMC.

4.2. **Generic Pipeline Paradigm.** The benchmark we consider is the generic pipeline paradigm (GPP) [8], which consists of three parts: Producer producing data, Consumer receiving data, and a chain of n intermediate Nodes that transmit data produced by Producer to Consumer. The local states for each component (Producer, Consumer, and intermediate Nodes), and their protocols are shown on Fig. 1.

Formula φ_3 states that there exists a path that always Producer is ready to produce data or either Consumer will receive data in maximum $2n+1$ steps.

$$\varphi_3 = \mathbf{EG}(\neg ProdSend \vee \underbrace{\mathbf{EX}(}_{2n+1} Received))$$

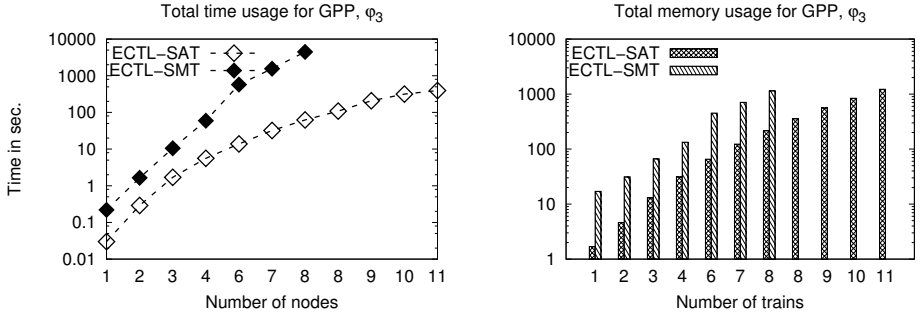
Formula φ_4 states that there exists a path that always in maximum $n^2 + 2n$ steps Consumer will receive the data.

$$\varphi_4 = \underbrace{\mathbf{EX}(Received \wedge \mathbf{EX}(Received \ldots)))}_{n^2+2n}$$

Formula φ_5 states that there exists a path that Producer will produce data and Consumer will receive the data.

$$\varphi_5 = \mathbf{EF}(ProdSend \wedge Received)$$

In Figures 6(a) and 6(b) we present a comparison of total time usage and total memory usage for the formulae φ_3.

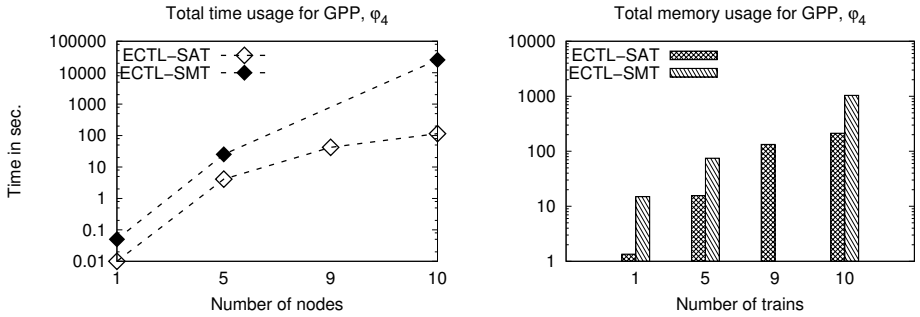


(a) (b)

FIGURE 6. A comparison of total time usage and total memory usage for the formulae φ_3.

An observation of experimental results for formula φ_3 leads to the conclusion that SAT-BMC uses less time and memory comparing to SMT-BMC for ECTL.

In Figures 7(a) and 7(b) we present a comparison of total time usage and total memory usage for the formulae φ_4.

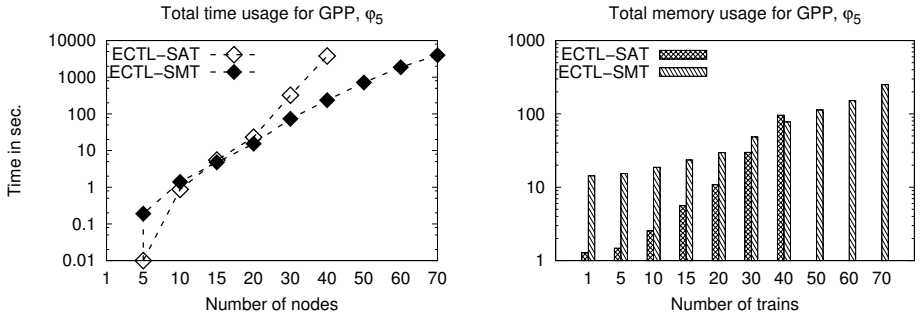


(a) (b)

FIGURE 7. A comparison of total time usage and total memory usage for the formulae φ_4.

An analysis of experimental results for formula φ_4 leads to the conclusion that also in this case SAT-BMC for ECTL uses less time and memory comparing to SMT-BMC. SMT-BMC is worse in this case because we need many short paths to verify the formula.

In Figures 8(a) and 8(b) we present a comparison of total time usage and total memory usage for the formulae φ_5.



(a) (b)

FIGURE 8. A comparison of total time usage and total memory usage for the formulae φ_5.

The SAT-BMC is able to verify the formula φ_5 for GPP with 40 nodes and the SMT-BMC is able to verify the formula for GPP with 70 nodes memory usage for the SAT-BMC is lower than for SMT-BMC.

For the tests we have used a computer equipped with Intel Core i7-5500U 2.4 GHz x 4 processor and 12 GB of RAM, running Ubuntu Linux. We have used the state of the art SAT-solver MiniSat5 [4] and Z3 SMT-solver [5].

5. Conclusions

In this paper we presented an SMT encoding for ECTL. We implemented the new method and we showed a comparison between the SAT- and SMT-based BMC methods for ECTL. The experimental results showed that, in general, SAT-based approach is better for tested systems and properties than SMT-based approach. In general, results show that the approaches are complementary, and that the SMT-based BMC approach appears to be superior for the short paths. This is a novel and interesting result, which shows that the choice of the BMC method should depend on the considered system and formula.

References

[1] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, 2008.

[2] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proceedings of the 5th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *LNCS*, pages 193–207. Springer-Verlag, 1999.

[3] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Advances in Computers*, 58:117–148, 2003.

[4] N. Eén and N. Sörensson. MiniSat. http://minisat.se/MiniSat.html.

[5] N. Eén and N. Sörensson. MiniSat - A SAT Solver with Conflict-Clause Minimization. In *Proceedings of 8th International Conference on Theory and Applications of Satisfiability Testing(SAT'05)*, LNCS. Springer-Verlag, 2005.

[6] E. A. Emerson and E. Clarke. Using branching-time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.

[7] W. van der Hoek and M. Wooldridge. Model checking knowledge and time. In *Proceedings of the 9th Int. SPIN Workshop (SPIN'02)*, volume 2318 of *LNCS*, pages 95–111. Springer-Verlag, 2002.

[8] D. Peled. All from one, one for all: On model checking using representatives. In *Proceedings of the 5th Int. Conf. on Computer Aided Verification (CAV'93)*, volume 697 of *LNCS*, pages 409–423. Springer-Verlag, 1993.

[9] W. Penczek, B. Woźna, and A. Zbrzezny. Bounded model checking for the universal fragment of CTL. *Fundamenta Informaticae*, 51(1-2):135–156, 2002.

[10] B. Woźna. ACTL* properties and bounded model checking. *Fundamenta Informaticae*, 63(1):65–87, 2004.

[11] B. Woźna-Szcześniak, A. M. Zbrzezny, and A. Zbrzezny. The BMC method for the existential part of RTCTLK and interleaved interpreted systems. In *Proceedings of*

the 15th Portuguese Conference on Artificial Intelligence (EPIA'2011), volume 7026 of *LNAI*, pages 551–565. Springer-Verlag, 2011.

[12] A. Zbrzezny. Improving the translation from ECTL to SAT. *Fundamenta Informaticae*, 85(1-4):513–531, 2008.

Received: June 2017

Agnieszka M. Zbrzezny
Jan Długosz University in Częstochowa
Institute of Mathematics and Computer Science
al. Armii Krajowej 13/15, 42-200 Częstochowa, Poland
E-mail address: agnieszka.zbrzezny@ajd.czest.pl