EPiC Series in Computing

Volume 57, 2018, Pages 471–487

LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning

Improving SAT-based Bounded Model Checking for Existential CTL through Path Reuse

Chuan Jiang and Gianfranco Ciardo

Department of Computer Science, Iowa State University
Ames, Iowa, USA
{cjiang, ciardo}@iastate.edu

Abstract

A complementary technique to decision-diagram-based model checking is SAT-based bounded model checking (BMC), which reduces the model checking problem to a propositional satisfiability problem so that the corresponding formula is satisfiable iff a counterexample or witness exists. Due to the branching time nature of computation tree logic (CTL), BMC for the universal fragment of CTL (ACTL) considers a counterexample in a bounded model as a set of bounded paths. Since the existential fragment of CTL (ECTL) is dual to ACTL, and ACTL formulas are often negated to obtain ECTL ones in practice, we focus on BMC for ECTL and propose an improved translation that generates a possibly smaller propositional formula by reducing the number of bounded paths to be considered in a witness. Experimental results show that the formulas generated by our approach are often easier for a SAT solver to answer. In addition, we propose a simple modification to the translation so that it is also defined for models with deadlock states.

1 Introduction

SAT-based bounded model checking (BMC) is a complementary technique to decision-diagram-based model checking [5]. By exploiting the observation that many real-life models have "shallow" counterexamples or witnesses, BMC only considers finite prefixes of infinite paths. It translates the semantics of temporal logic, bounded by some integer k, into a propositional formula, and leverages a SAT solver to check for satisfiability. If the formula is determined to be satisfiable, a counterexample or witness can be generated from the truth assignment produced by the SAT solver. Otherwise, k is increased progressively until either a counterexample or witness is found, or some preset upper bound is reached. BMC was first proposed for linear temporal logic (LTL) [1, 2] and later applied to the universal fragment of computation tree logic (ACTL) [15] and $\forall\mu$-calculus [14, 19]. Decision-diagram-based techniques inspired by the same idea were also proposed [18, 21].

In computation tree logic (CTL), the existential fragment (ECTL) is dual to ACTL, i.e., the negation of an ACTL formula is an ECTL formula. Therefore, these two fragments of CTL can often be considered together. In this paper, we focus on SAT-based BMC for ECTL, which is different from applying BMC to LTL, since the general form of ECTL witnesses is

tree-like [9]. In [6, 15, 22], a counterexample or witness in a bounded model is represented as a set of bounded paths. Different encoding schemes based on proof system were proposed in [14, 19], in the larger context of $\forall\mu$-calculus. In this paper, we improve the translation to propositional formulas from [6, 15, 22] by reducing the number of bounded paths that must be considered. Our approach generates a smaller formula, which is often easier for a SAT solver to answer, or the same one as in the classic approach, in the worst case. In addition, we propose a simple modification to the translation so that it is also defined for models with deadlock states.

The remainder of this paper is organized as follows. Section 2 defines ECTL and its bounded semantics, and presents the translation of bounded semantics into a propositional formula. Section 3 proposes an improved translation to produce a possibly smaller propositional formula. Section 4 compares the classic approach and ours with respect to the minimum bound to find a witness, which determines the earliest possibility for BMC to provide an answer, and with respect to the complexity of propositional formulas, in terms of the number of symbolic states considered to form a witness. Section 5 modifies the translation to cope with models containing deadlock states. Section 6 describes our experimental design and presents the results, while Section 7 concludes our discussion and outlines future work.

2 Background

We denote sets by calligraphic letters (e.g., $\mathcal{A}$), except for the natural numbers $\mathbb{N} = \{0, 1, ...\}$.

2.1 Kripke Structures and ECTL

A model is represented as a Kripke structure $M = (\mathcal{S}, \mathcal{S}_{init}, \mathcal{N}, \mathcal{A}, \mathcal{L})$, where $\mathcal{S}$ is the state space, $\mathcal{S}_{init} \subseteq \mathcal{S}$ is the set of initial states, $\mathcal{N} \subseteq \mathcal{S} \times \mathcal{S}$ is the total transition relation, $\mathcal{A}$ is a set of atomic propositions, and $\mathcal{L} : \mathcal{S} \to 2^{\mathcal{A}}$ is a labeling function that gives the atomic propositions holding in each state (subject to $true \in \mathcal{A}$ holding in every state). Let $\mathcal{P}$ be the set of paths in M, i.e., infinite sequences $(s_0, s_1, ...)$ of states, where $(s_i, s_{i+1}) \in \mathcal{N}$ for any $i \in \mathbb{N}$.

We consider ECTL, the existential fragment of the temporal logic CTL [8], where formulas have the following syntax (φ and ρ are ECTL formulas, a is an atomic proposition):

$$\varphi ::= a \mid \neg a \mid \varphi \wedge \rho \mid \varphi \vee \rho \mid \mathsf{EX}\varphi \mid \mathsf{E}(\varphi \mathsf{U} \rho) \mid \mathsf{EG}\varphi,$$

and the conditions for state s in model M to satisfy formula φ, written $s \models \varphi$ (M is omitted for brevity), are as follows:

$$\begin{array}{lcl}
s \models a & \Leftrightarrow & a \in \mathcal{L}(s) \\
s \models \neg a & \Leftrightarrow & a \notin \mathcal{L}(s) \\
s \models \varphi \wedge \rho & \Leftrightarrow & (s \models \varphi) \wedge (s \models \rho) \\
s \models \varphi \vee \rho & \Leftrightarrow & (s \models \varphi) \vee (s \models \rho) \\
s \models \mathsf{EX}\varphi & \Leftrightarrow & \exists (s_0, s_1, ...) \in \mathcal{P}, (s_0 \equiv s) \wedge (s_1 \models \varphi) \\
s \models \mathsf{E}(\varphi \mathsf{U} \rho) & \Leftrightarrow & \exists (s_0, s_1, ..., s_i, ...) \in \mathcal{P}, (s_0 \equiv s) \wedge (s_i \models \rho) \wedge (\forall j \in \{0, ..., i-1\}, s_j \models \varphi) \\
s \models \mathsf{EG}\varphi & \Leftrightarrow & \exists (s_0, s_1, ...) \in \mathcal{P}, (s_0 \equiv s) \wedge (\forall i \in \mathbb{N}, s_i \models \varphi)
\end{array}$$

(formulas $\mathsf{EF}\varphi$ and $\mathsf{E}(\varphi \mathsf{R} \rho)$ can be expressed as $\mathsf{E}(true \mathsf{U} \varphi)$ and $\mathsf{EG}\rho \vee \mathsf{E}(\rho \mathsf{U}(\varphi \wedge \rho))$, respectively, so we do not discuss them separately).

Definition 2.1. *An ECTL formula φ is (existentially) valid in a model $M = (\mathcal{S}, \mathcal{S}_{init}, \mathcal{N}, \mathcal{A}, \mathcal{L})$, written $M \models \varphi$, iff for some $s \in \mathcal{S}_{init}$, $s \models \varphi$.*

ACTL, the universal fragment of CTL, is the dual of ECTL. A counterexample to $\mathsf{AX}\varphi$, $\mathsf{AG}\varphi$, or $\mathsf{A}(\varphi\mathsf{U}\rho)$ is a witness for $\mathsf{EX}(\neg\varphi)$, $\mathsf{EF}(\neg\varphi)$, or $\mathsf{E}(\neg\rho\mathsf{U}(\neg\varphi\wedge\neg\rho))\vee\mathsf{EG}(\neg\rho)$, respectively (where the negation $\neg$ can be recursively "pushed down" to atomic propositions), i.e., the negation of an ACTL formula is an ECTL formula.

2.2 Bounded Semantics of ECTL

Given a model $M=(\mathcal{S},\mathcal{S}_{init},\mathcal{N},\mathcal{A},\mathcal{L})$ and $k\in\mathbb{N}$, a *k-path*, or a path of length k, is a finite sequence $\pi=(s_0,...,s_k)$ of $k+1$ states such that $(s_i,s_{i+1})\in\mathcal{N}$ for any $i\in\{0,...,k-1\}$. Let $\pi[i]$ denote s_i, the i-th state in π. A k-path, though finite, can still represent an infinite path if the last state is the same as any of the previous states. We define a function $loop(\pi)$ to determine if a k-path π is a loop:

$$loop(\pi)=\{i\in\{0,...,k-1\}\,|\,\pi[k]\equiv\pi[i]\}\,,$$

thus $loop(\pi)\neq\emptyset$ iff π is a loop.

In our definition, a loop is thus a k-path containing at least two states [12]. This is slightly different from the notation in [2, 15], which explicitly requires a back loop from the last state to some state on the path: $\{i\in\{0,...,k\}\,|\,(\pi[k],\pi[i])\in\mathcal{N}\}$. Figure 1 illustrates the two ways of thinking about, and defining, the same loop. We choose this notation because encoding state equivalence is often simpler and more compact than encoding a transition relation step. Our notation requires $k\geq 1$, which we then assume in the rest of the paper.

(a) The loop shape in [2, 15] (b) The loop shape in this paper

Figure 1: The two kinds of loop shapes

The *k-model* of M is a tuple $M_k=(\mathcal{S},\mathcal{S}_{init},\mathcal{P}_k,\mathcal{A},\mathcal{L})$, where $\mathcal{P}_k$ is the set of all the k-paths in M. The bounded semantics is defined over a k-model M_k.

Definition 2.2 (Bounded semantics of ECTL)**.** *Let M_k be the k-model of a model M, s a state of M, a an atomic proposition, and φ, ρ ECTL formulas. The conditions for s in M_k to satisfy φ, written $s\models_k\varphi$ (M_k is omitted for brevity), are defined as follows:*

$$\begin{array}{lcl}
s\models_k a & \Leftrightarrow & a\in\mathcal{L}(s)\\
s\models_k \neg a & \Leftrightarrow & a\notin\mathcal{L}(s)\\
s\models_k \varphi\wedge\rho & \Leftrightarrow & (s\models_k\varphi)\wedge(s\models_k\rho)\\
s\models_k \varphi\vee\rho & \Leftrightarrow & (s\models_k\varphi)\vee(s\models_k\rho)\\
s\models_k \mathsf{EX}\varphi & \Leftrightarrow & \exists\pi\in\mathcal{P}_k,(\pi[0]\equiv s)\wedge(\pi[1]\models_k\varphi)\\
s\models_k \mathsf{E}(\varphi\mathsf{U}\rho) & \Leftrightarrow & \exists\pi\in\mathcal{P}_k,(\pi[0]\equiv s)\wedge\\
 & & (\exists i\in\{0,...,k\},(\pi[i]\models_k\rho)\wedge(\forall j\in\{0,...,i-1\},\pi[j]\models_k\varphi))\\
s\models_k \mathsf{EG}\varphi & \Leftrightarrow & \exists\pi\in\mathcal{P}_k,(\pi[0]\equiv s)\wedge(loop(\pi)\neq\emptyset)\wedge(\forall i\in\{0,...,k\},\pi[i]\models_k\varphi).
\end{array}$$

Theorem 2.1. *Let M be a model, $\mathcal{R}$ the reachable states of M, s a state of M, and φ an ECTL formula. Then, $s\models\varphi$ iff $s\models_k\varphi$ for some $k\in\{1,...,|\mathcal{R}|\}$.*

Definition 2.3. *An ECTL formula φ is (existentially) valid in a k-model $M_k=(\mathcal{S},\mathcal{S}_{init},\mathcal{P}_k,\mathcal{A},\mathcal{L})$, written $M\models_k\varphi$, iff $s\models_k\varphi$ for some $s\in\mathcal{S}_{init}$.*

Theorem 2.2. *Given an ECTL formula φ, $M \models \varphi$ iff $M \models_k \varphi$ for some $k \in \{1, ..., |\mathcal{R}|\}$.*

Based on Theorem 2.2, an ECTL model checking problem is reduced to an ECTL BMC problem: the unbounded and bounded semantics are equivalent for a sufficiently large bound.

2.3 Translation into a Propositional Formula

SAT-based BMC for ACTL formulas was proposed in [15], and later improved with a more efficient translation to propositional formulas [6, 22]. Since it begins with negating an ACTL formula, this approach actually checks an ECTL formula over a bounded model. Compared to the original BMC for LTL [2], the main difference is that this approach represents a witness or counterexample as a set of symbolic k-paths, due to the branching time nature of CTL. For example, a witness for $\mathsf{EG}(\mathsf{EF}a)$ contains a lasso-shaped loop of length k where $\mathsf{EF}a$ holds in every state, and, for each state on that loop, a path of length k showing that we can reach a state satisfying a from it. The bound k does not describe the size of a potential witness or counterexample, but the length of each individual path demonstrating satisfaction or violation of a subformula in a state.

The number of k-paths needed to check an ECTL formula is given by a function f_k:

Definition 2.4. *Function $f_k : ECTL \to \mathbb{N}$ is defined as follows:*

$$\begin{array}{lcl} f_k(a) & = & 0, \text{ where } a \in \mathcal{A} \\ f_k(\neg a) & = & 0, \text{ where } a \in \mathcal{A} \\ f_k(\varphi \wedge \rho) & = & f_k(\varphi) + f_k(\rho) \\ f_k(\varphi \vee \rho) & = & \max(f_k(\varphi), f_k(\rho)) \\ f_k(\mathsf{EX}\varphi) & = & f_k(\varphi) + 1 \\ f_k(\mathsf{E}(\varphi \mathsf{U} \rho)) & = & k \cdot f_k(\varphi) + f_k(\rho) + 1 \\ f_k(\mathsf{EG}\varphi) & = & k \cdot f_k(\varphi) + 1, \end{array}$$

where $f_k(\mathsf{EG}\varphi)$ is slightly different from that in [6, 15, 22] because of our loop notation.

For example, given a bound k, the maximum witness to show $\mathsf{E}(\varphi \mathsf{U} \rho)$ consists of a k-path $(s_0, s_1, ..., s_k)$, $f_k(\rho)$ k-paths to show $s_k \models \rho$, and, for each $i \in \{0, ..., k-1\}$, $f_k(\varphi)$ k-paths to show $s_i \models \varphi$.

Symbolically, a state is represented as a vector of boolean variables. Let π_i denote the i-th symbolic k-path, which is a sequence of $k+1$ symbolic states. Checking an ECTL formula φ over a bounded model M_k is then reduced to checking the satisfiability of a propositional formula $[M, \varphi]_k = [M_k]_\varphi \wedge [\varphi]_k$, where:

- $[M_k]_\varphi$ is a propositional formula that enforces the $f_k(\varphi)$ state sequences to be valid k-paths and $\pi_0[0]$ to be an initial state:

$$[M_k]_\varphi = I(\pi_0[0]) \wedge \bigwedge_{i=0}^{f_k(\varphi)-1} \bigwedge_{j=0}^{k-1} \mathcal{N}(\pi_i[j], \pi_i[j+1]),$$

  where $I(s)$ iff $s \in \mathcal{S}_{init}$ and $\mathcal{N}(s, s')$ iff $(s, s') \in \mathcal{N}$.

- $[\varphi]_k = [\varphi, \pi_0[0]]_k^0$ is a propositional formula that assembles k-paths and enforces φ or subformulas of φ on each corresponding k-path; specifically, $[\varphi, s]_k^i$ enforces φ on the i-th

k-path, where the first state of that k-path must be equivalent to s (unless φ is a pure propositional formula):

$$
\begin{array}{lcl}
[a, s]_k^i & = & a(s), \text{ where } a \in \mathcal{A} \\
[\neg a, s]_k^i & = & \neg a(s), \text{ where } a \in \mathcal{A} \\
[\varphi \wedge \rho, s]_k^i & = & [\varphi, s]_k^i \wedge [\rho, s]_k^{i+f_k(\varphi)} \\
[\varphi \vee \rho, s]_k^i & = & [\varphi, s]_k^i \vee [\rho, s]_k^i \\
[\mathsf{EX}\varphi, s]_k^i & = & (s \equiv \pi_i[0]) \wedge [\varphi, \pi_i[1]]_k^{i+1} \\
[\mathsf{E}(\varphi \mathsf{U} \rho), s]_k^i & = & (s \equiv \pi_i[0]) \wedge \bigvee_{j=0}^{k} \left([\rho, \pi_i[j]]_k^{i+1} \wedge \bigwedge_{t=0}^{j-1} [\varphi, \pi_i[t]]_k^{i+1+f_k(\rho)+t \cdot f_k(\varphi)} \right) \\
[\mathsf{EG}\varphi, s]_k^i & = & (s \equiv \pi_i[0]) \wedge \bigvee_{j=0}^{k-1} (\pi_i[k] \equiv \pi_i[j]) \wedge \bigwedge_{j=0}^{k-1} [\varphi, \pi_i[j]]_k^{i+1+j \cdot f_k(\varphi)},
\end{array}
$$

where, again, $[\mathsf{EG}\varphi, s]_k^i$ is slightly different from that in [6, 15, 22].

For example, the interpretation of $[\mathsf{E}(\varphi \mathsf{U} \rho), s]_k^i$ is as follows: First, the first state of π_i is equivalent to the given state s. Then, for each $j \in \{0, ..., k\}$, we start from π_{i+1} to search for a witness for ρ in $\pi_i[j]$, consisting of $f_k(\rho)$ k-paths, and from $\pi_{i+1+f_k(\rho)+t \cdot f_k(\varphi)}$ to search for a witness for φ in $\pi_i[t]$, consisting of $f_k(\varphi)$ k-paths, for each $t \in \{0, ..., j-1\}$.

We refer to the translation above as the Classic approach.

Theorem 2.3. *Let M be a model, and φ an ECTL formula. $M \models_k \varphi$ iff $[M, \varphi]_k$ is satisfiable.*

Theorem 2.4. *$M \models \varphi$ iff for some $k \in \{1, ..., |\mathcal{R}|\}$, $[M, \varphi]_k$ is satisfiable.*

In practice, it is often the case that, if $M \models \varphi$, there exists a small k such that $M \models_k \varphi$, i.e., a "shallow" witness exists. This is the reason why BMC is often very efficient in error detection. The deeper the witness is, the less advantage BMC has.

3 Improved Translation of Bounded Semantics

We now give an example to explain how we improve on Classic. Given an ECTL formula, the goal of BMC is to find a witness demonstrating satisfaction. Figure 2(a) can be viewed as a witness for $\mathsf{EG}(\mathsf{EF}a)$ when $k = 3$, consisting of two k-paths, $(1, 2, 3, 2)$ and $(3, 4, 5, *)$, where $*$ can be any valid state, because we can extract Figure 2(b) from it. By *reusing* paths, we can represent a witness in such a compact form. For example, the witness for $\mathsf{EF}a$ in state 2 is built by concatenating state 2 to the witness for $\mathsf{EF}a$ in state 3. BMC can benefit from this compact form, since the minimum k to find a witness is determined by the longest subpath (not counting $*$) in it. We can find the witness in Figure 2(a) when $k = 3$, since the longest subpath is $(1, 2, 3, 2)$. Without reusing, we must have $k = 4$, since the longest subpath is now $(1, 2, 3, 4, 5)$, the witness for $\mathsf{EF}a$ in state 1, and use five k-paths to find that witness (in a different form) as shown in Figure 2(c): (1, 2, 3, 2, 3), (1, 2, 3, 4, 5), (2, 3, 4, 5, $*$), (3, 4, 5, $*$, $*$), and (2, 3, 4, 5, $*$).

Given two states s and s' such that $(s, s') \in \mathcal{N}$, if $\mathsf{E}(\varphi \mathsf{U} \rho)$ (or $\mathsf{EG}\varphi$) holds in s', and φ holds in s, then $\mathsf{E}(\varphi \mathsf{U} \rho)$ (or $\mathsf{EG}\varphi$) also holds in s. This is due to the inductive definitions of CTL temporal operators, and can be exploited to improve the translation of bounded semantics. To enforce $\mathsf{E}(\varphi \mathsf{U} \rho)$ to hold in every state along a finite path, we only need to enforce $\mathsf{E}(\varphi \mathsf{U} \rho)$ to hold in the last state, and $\varphi \vee \rho$ in any previous state. Similarly, in the case of $\mathsf{EG}\varphi$, we only need to enforce $\mathsf{EG}\varphi$ to hold in the last state, while φ is enough in any previous state. This potentially reduces the number of necessary k-paths, thus the size of the resulting propositional formula, to represent a witness in the bounded model.

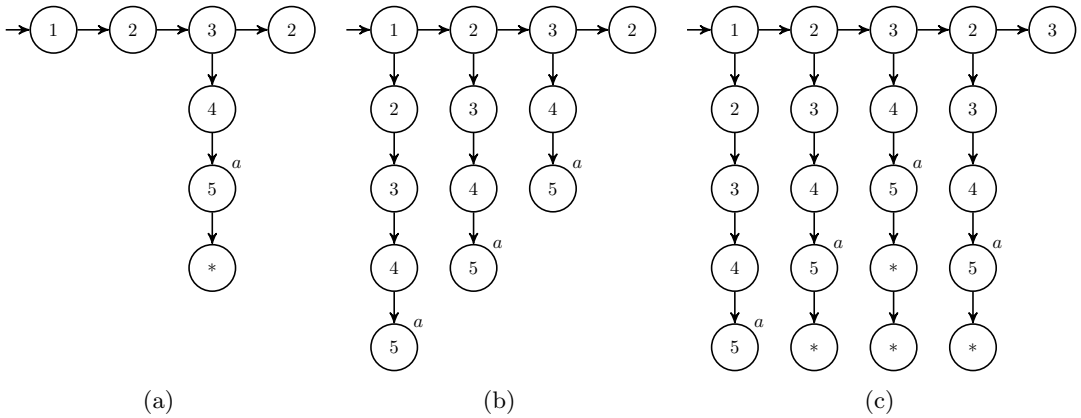


Figure 2: Different forms of witnesses for $\mathsf{EG}(\mathsf{EF}a)$.

We then define an auxiliary function $\mu : ECTL \to ECTL$ providing the *sufficient predecessor formula* to be enforced in "any previous state":

Definition 3.1. *Function $\mu : ECTL \to ECTL$ is defined as follows:*

$$\begin{array}{lcl} \mu(a) & = & a, \text{ where } a \in \mathcal{A} \\ \mu(\neg a) & = & \neg a, \text{ where } a \in \mathcal{A} \\ \mu(\varphi \wedge \rho) & = & \mu(\varphi) \wedge \mu(\rho) \\ \mu(\varphi \vee \rho) & = & \varphi \vee \rho \\ \mu(\mathsf{EX}\varphi) & = & \mathsf{EX}\varphi \\ \mu(\mathsf{E}(\varphi \mathsf{U} \rho)) & = & \varphi \vee \rho \\ \mu(\mathsf{EG}\varphi) & = & \mu(\varphi). \end{array}$$

Theorem 3.1. *Let $M = (\mathcal{S}, \mathcal{S}_{init}, \mathcal{N}, \mathcal{A}, \mathcal{L})$ be a model, s, s' states of M such that $(s, s') \in \mathcal{N}$, and φ an ECTL formula. If $s' \models \varphi$ and $s \models \mu(\varphi)$, then $s \models \varphi$.*

Proof. We only need to prove correctness in the cases of conjunction, EU, and EG.

- If $s' \models \mathsf{E}(\varphi \mathsf{U} \rho)$ and $s \models \varphi \vee \rho$, then $s \models \mathsf{E}(\varphi \mathsf{U} \rho)$:
  If $s \models \rho$, it is trivial to see that $s \models \mathsf{E}(\varphi \mathsf{U} \rho)$. If $s \models \varphi$, since s has a successor $s' \models \mathsf{E}(\varphi \mathsf{U} \rho)$, then $s \models \mathsf{E}(\varphi \mathsf{U} \rho)$ by the definition of EU.

The proof for conjunction and EG is based on induction on the structure of the ECTL formula. The basis is that $\mu(a) = a$, whose correctness is trivial.

- If $s' \models \varphi \wedge \rho$ and $s \models \mu(\varphi) \wedge \mu(\rho)$, then $s \models \varphi \wedge \rho$:
  Assume that, if $s' \models \varphi$ and $s \models \mu(\varphi)$ then $s \models \varphi$, and that, if $s' \models \rho$ and $s \models \mu(\rho)$ then $s \models \rho$. By these assumptions, we have $s \models \varphi$ and $s \models \rho$, thus $s \models \varphi \wedge \rho$.

- If $s' \models \mathsf{EG}\varphi$ and $s \models \mu(\varphi)$, then $s \models \mathsf{EG}\varphi$:
  Assume that, if $s' \models \varphi$ and $s \models \mu(\varphi)$ then $s \models \varphi$. Since $s' \models \mathsf{EG}\varphi$, we have $s' \models \varphi$, thus $s \models \varphi$ by assumption; since s has a successor $s' \models \mathsf{EG}\varphi$, $s \models \mathsf{EG}\varphi$ by the definition of EG.

□

It is worthwhile to emphasize that Theorem 3.1 does not hold if we define $\mu(\varphi \vee \rho)$ as $\mu(\varphi) \vee \mu(\rho)$. Consider the simple example where $\varphi = \mathsf{EG}a$ and $\rho = \mathsf{EG}b$, in Figure 3. $(\mathsf{EG}a) \vee (\mathsf{EG}b)$ does not hold in s, because (s, s', s'), obtained through path reuse, is not a witness for $(\mathsf{EG}a) \vee (\mathsf{EG}b)$ in s, even though $a \vee b$ holds in s and (s', s') witnesses $(\mathsf{EG}a) \vee (\mathsf{EG}b)$ in s'.

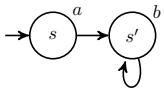


Figure 3: An example showing that $\mu(\varphi \vee \rho) \neq \mu(\varphi) \vee \mu(\rho)$.

According to Theorem 3.1, given a finite path $(s_0, ..., s_n)$, to enforce an ECTL formula φ in s_i for any $i \in \{0, ..., n\}$, we enforce $s_n \models \varphi$ but just $s_i \models \mu(\varphi)$, for $i \in \{0, ..., n-1\}$, which usually simplifies the formula to be enforced in s_i.

We define function g_k, as an improvement of f_k, giving the potentially smaller number of k-paths needed to check an ECTL formula in a bounded model. g_k differs from f_k only in the case of EU and EG:

Definition 3.2. *Function $g_k : ECTL \to \mathbb{N}$ is defined as follows:*

$$\begin{array}{lcl}
g_k(a) & = & 0, \text{ where } a \in \mathcal{A} \\
g_k(\neg a) & = & 0, \text{ where } a \in \mathcal{A} \\
g_k(\varphi \wedge \rho) & = & g_k(\varphi) + g_k(\rho) \\
g_k(\varphi \vee \rho) & = & \max(g_k(\varphi), g_k(\rho)) \\
g_k(\mathsf{EX}\varphi) & = & g_k(\varphi) + 1 \\
g_k(\mathsf{E}(\varphi \mathsf{U} \rho)) & = & (k-1) \cdot g_k(\mu(\varphi)) + g_k(\varphi) + g_k(\rho) + 1 \\
g_k(\mathsf{EG}\varphi) & = & (k-1) \cdot g_k(\mu(\varphi)) + g_k(\varphi) + 1.
\end{array}$$

For example, given a bound k, the maximum witness to show $\mathsf{E}(\varphi \mathsf{U} \rho)$, according to our approach, consists of a k-path $(s_0, ..., s_k)$, $g_k(\rho)$ k-paths to show $s_k \models \rho$, $g_k(\varphi)$ paths to show $s_{k-1} \models \varphi$, and, for each $i \in \{0, ..., k-2\}$, $g_k(\mu(\varphi))$ k-paths to show $s_i \models \mu(\varphi)$, since $s_i \models \varphi$ can be inferred.

Theorem 3.2. *Given an ECTL formula φ, $g_k(\mu(\varphi)) \leq g_k(\varphi)$.*

Theorem 3.3. *Given an ECTL formula φ, $g_k(\varphi) \leq f_k(\varphi)$.*

Then, we update the translation of conjunctive, EU, and EG formulas to propositional formulas (while we just replace f_k with g_k for the remaining formulas):

$$\begin{array}{lcl}
[\varphi \wedge \rho, s]_k^i & = & [\mu(\varphi), s]_k^i \wedge [\mu(\rho), s]_k^{i+g_k(\mu(\varphi))} \\
[\mathsf{E}(\varphi \mathsf{U} \rho), s]_k^i & = & (s \equiv \pi_i[0]) \wedge \Big([\rho, \pi_i[0]]_k^{i+1} \vee \bigvee_{j=1}^{k} \big([\rho, \pi_i[j]]_k^{i+1} \wedge [\varphi, \pi_i[j-1]]_k^{i+1+g_k(\rho)} \\
 & & \quad \wedge \bigwedge_{t=0}^{j-2} [\mu(\varphi), \pi_i[t]]_k^{i+1+g_k(\rho)+g_k(\varphi)+t \cdot g_k(\mu(\varphi))} \big) \Big) \\
[\mathsf{EG}\varphi, s]_k^i & = & (s \equiv \pi_i[0]) \wedge \bigvee_{j=0}^{k-1} (\pi_i[k] \equiv \pi_i[j]) \\
 & & \quad \wedge [\varphi, \pi_i[k-1]]_k^{i+1} \wedge \bigwedge_{j=0}^{k-2} [\mu(\varphi), \pi_i[j]]_k^{i+1+g_k(\varphi)+j \cdot g_k(\mu(\varphi))}.
\end{array}$$

We refer to this new translation as the Reuse approach.

The templates of ACTL and ECTL formulas guaranteeing that their instances have linear counterexamples and witnesses (if they exist) have been studied in [4, 20]. For formulas instantiated from linear ECTL templates, $g_k(\varphi) = f_k(\varphi)$, i.e., Reuse generates exactly the same

propositional formula as Classic. For some ECTL formulas whose general form of witnesses is not linear (e.g., $\mathsf{EG}(\mathsf{EX}a)$, $(\mathsf{EG}a) \vee (\mathsf{EG}b)$), the two approaches may also generate the same formulas. A complete template of ECTL formulas for which $g_k(\varphi) < f_k(\varphi)$ is still unclear.

4 Comparison of the Two Translation Approaches

In this section, we compare Classic and Reuse with respect to the minimum bound needed to find a witness (Section 4.1) and the complexity of propositional formulas (Section 4.2).

4.1 Minimum Bound to Find a Witness

We already saw an example where Reuse finds a witness using a smaller k than Classic, at the beginning of Section 3. Let k_{min}^{Classic} and k_{min}^{Reuse} denote the minimum bounds for which Classic and Reuse find a witness for a particular formula, respectively.

Theorem 4.1. *Given an ECTL formula φ holding in a model M, $k_{min}^{\text{Reuse}} \leq k_{min}^{\text{Classic}}$.*

Proof. If Classic finds a witness, Reuse can always find a witness in the compact form, as a substructure of the witness found by Classic, thus using the same k. For example, suppose Classic finds the witness for $\mathsf{E}((\mathsf{EF}a)\mathsf{U}b)$ shown in Figure 4(a), using $k = 3$. Its substructure (shown in the dashed box) is a witness Reuse can also find. This implies that k_{min}^{Reuse} is never greater than k_{min}^{Classic}. □

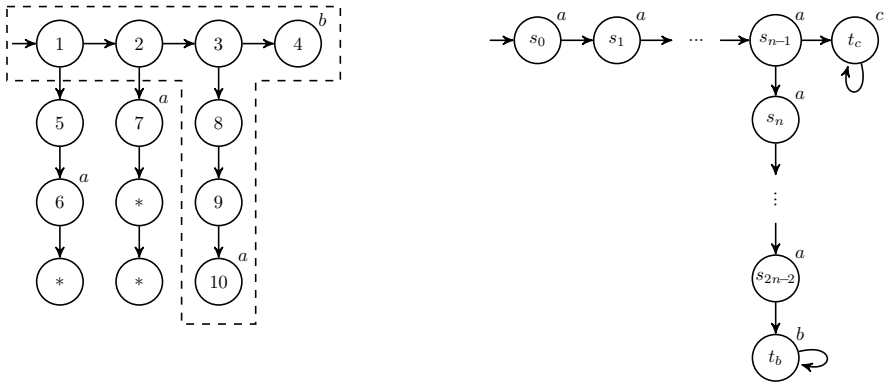
Theorem 4.2. *Given an ECTL formula φ holding in a model M, k_{min}^{Classic} can be as large as $2k_{min}^{\text{Reuse}} - 1$.*

Proof. To prove this theorem, it suffices to consider the ECTL formula $\varphi = \mathsf{E}(\mathsf{E}(a\mathsf{U}b)\mathsf{U}c)$ and the model shown in Figure 4(b). Reuse seeks two k-paths, one path π where some $\pi[j]$ satisfies c, $\pi[j-1]$ satisfies $\mathsf{E}(a\mathsf{U}b)$, and $\pi[0], ..., \pi[j-2]$ satisfy $\mu(\mathsf{E}(a\mathsf{U}b)) = a \vee b$, and another path σ where $\sigma[0]$ coincides with $\pi[j-1]$, some $\sigma[l]$ satisfies b, and $\sigma[0], ..., \sigma[l-1]$ satisfy a. For the model in Figure 4(b), these two paths correspond to $(s_0, ..., s_{n-1}, t_c)$ and $(s_{n-1}, ..., s_{2n-2}, t_b)$, respectively, and Reuse finds them using a bound as low as $k_{min}^{\text{Reuse}} = n$.

Instead, Classic seeks $k+1$ k-paths, the first one, π, where some $\pi[j]$ satisfies c, and $\pi[0], ..., \pi[j-1]$ satisfy $\mathsf{E}(a\mathsf{U}b)$, as shown by the j k-paths $\sigma_0, ..., \sigma_{j-1}$ having initial state $\pi[0], ..., \pi[j-1]$, respectively (the remaining $k-j$ paths can be any valid k-paths). For the model in Figure 4(b), path π corresponds to $(s_0, ..., s_{n-1}, t_c)$, the same as for Reuse; however, path σ_0 cannot be built unless $k \geq 2n-1$, thus $k_{min}^{\text{Classic}} = 2n-1$. □

4.2 Complexity of Propositional Formulas

We now compare the complexity of propositional formulas generated by Classic and Reuse in terms of the number of symbolic states needed to form a witness, $N_{\text{Classic}}(\varphi) = (k+1) \cdot f_k(\varphi)$ and $N_{\text{Reuse}}(\varphi) = (k+1) \cdot g_k(\varphi)$. Modern SAT solvers accept formulas in *conjunctive normal form* (CNF) as input. A common practice to transform a propositional formula into CNF is Tseitin transformation [16, 17], which introduces additional boolean variables to avoid an exponential growth in the size of the CNF formula. Therefore, N_{Classic} and N_{Reuse} do not quantitatively depict the size of the search space a SAT solver explores or the hardness of the formula, but affect the efficiency of CNF transformation, and it seems plausible to compare and use them as a way to assess the hardness of the formula.



(a) A witness for $\mathsf{E}((\mathsf{EF}a)\mathsf{U}b)$ (b) A model where $\mathsf{E}(\mathsf{E}(a\mathsf{U}b)\mathsf{U}c)$ holds

Figure 4: Examples used in the proofs of Theorem 4.1 and 4.2.

We first consider the simple formula $\mathsf{EG}(\mathsf{EF}a)$. Table 1 lists the values of $N_{\text{CLASSIC}}(\mathsf{EG}(\mathsf{EF}a))$ and $N_{\text{REUSE}}(\mathsf{EG}(\mathsf{EF}a))$ w.r.t. k. It can be seen that $N_{\text{CLASSIC}}(\mathsf{EG}(\mathsf{EF}a))$ grows quadratically, while $N_{\text{REUSE}}(\mathsf{EG}(\mathsf{EF}a))$ grows linearly, as k increases. The larger k is, the more significant the difference becomes.

k	$f_k(\mathsf{EG}(\mathsf{EF}a))$	$N_{\text{CLASSIC}}(\mathsf{EG}(\mathsf{EF}a))$	$g_k(\mathsf{EG}(\mathsf{EF}a))$	$N_{\text{REUSE}}(\mathsf{EG}(\mathsf{EF}a))$
1	2	4	2	4
2	3	9	2	6
3	4	16	2	8
4	5	25	2	10
5	6	36	2	12

Table 1: Comparing the two translation approaches on $\mathsf{EG}(\mathsf{EF}a)$:
$f_k(\mathsf{EG}(\mathsf{EF}a)) = k \cdot f_k(\mathsf{EF}a) + 1 = k \cdot (f_k(a) + 1) + 1 = k + 1\,,$
$g_k(\mathsf{EG}(\mathsf{EF}a)) = (k-1) \cdot g_k(\mu(\mathsf{EF}a)) + g_k(\mathsf{EF}a) + 1 = (k-1) \cdot g_k(a) + (g_k(a) + 1) + 1 = 2\,.$

Then, let us investigate a family of more complex formulas. Let a_i be an atomic proposition for $i \in \mathbb{N}$ and consider the family of ECTL formulas $\{\varphi_1, \varphi_2, ...\}$, where $\varphi_1 = \mathsf{E}(a_0\mathsf{U}a_1)$ and $\varphi_i = \mathsf{E}(\varphi_{i-1}\mathsf{U}a_i)$ for $i \geq 2$. According to Def. 2.4, $f_k(\varphi_1) = 1$ and, for $i \in \{2, ..., n\}$,

$$f_k(\varphi_i) = k \cdot f_k(\varphi_{i-1}) + 1\,,$$

which can be rewritten as

$$\frac{f_k(\varphi_i) + \frac{1}{k-1}}{f_k(\varphi_{i-1}) + \frac{1}{k-1}} = k\,.$$

This implies that $f_k(\varphi_1) + \frac{1}{k-1}, f_k(\varphi_2) + \frac{1}{k-1}, ..., f_k(\varphi_i) + \frac{1}{k-1}$ is a geometric series with ratio k. Since its first element $f_k(\varphi_1) + \frac{1}{k-1}$ equals $1 + \frac{1}{k-1} = \frac{k}{k-1}$, its i-th element $f_k(\varphi_i) + \frac{1}{k-1}$ equals $\frac{k^i}{k-1}$, from which we can conclude that

$$f_k(\varphi_i) = \frac{k^i - 1}{k - 1}\,.$$

Finally,

$$N_{\text{CLASSIC}}(\varphi_i) = (k+1) \cdot f_k(\varphi_i) = \frac{k+1}{k-1}(k^i - 1)\,. \tag{1}$$

Now we compute $N_{\text{REUSE}}(\varphi_i)$. According to Def. 3.2, $g_k(\varphi_1) = 1$, $g_k(\varphi_2) = 2$, and, for $i \in \{3, ..., n\}$,

$$\begin{aligned} g_k(\varphi_i) &= (k-1) \cdot g_k(\mu(\varphi_{i-1})) + g_k(\varphi_{i-1}) + g_k(a_i) + 1 \\ &= (k-1) \cdot g_k(\varphi_{i-2} \vee a_{i-1}) + g_k(\varphi_{i-1}) + 1 \\ &= (k-1) \cdot g_k(\varphi_{i-2}) + g_k(\varphi_{i-1}) + 1\,. \end{aligned}$$

We build two geometric series by rewriting the equation above as

$$\frac{g_k(\varphi_i) - \frac{1-\sqrt{4k-3}}{2} g_k(\varphi_{i-1}) - \frac{2}{1-\sqrt{4k-3}}}{g_k(\varphi_{i-1}) - \frac{1-\sqrt{4k-3}}{2} g_k(\varphi_{i-2}) - \frac{2}{1-\sqrt{4k-3}}} = \frac{1+\sqrt{4k-3}}{2}$$

and

$$\frac{g_k(\varphi_i) - \frac{1+\sqrt{4k-3}}{2} g_k(\varphi_{i-1}) - \frac{2}{1+\sqrt{4k-3}}}{g_k(\varphi_{i-1}) - \frac{1+\sqrt{4k-3}}{2} g_k(\varphi_{i-2}) - \frac{2}{1+\sqrt{4k-3}}} = \frac{1-\sqrt{4k-3}}{2}\,.$$

Therefore, the two geometric series have ratio $\frac{1+\sqrt{4k-3}}{2}$ and $\frac{1-\sqrt{4k-3}}{2}$, respectively. Following similar steps to those used to compute $f_k(\varphi_i)$, we obtain

$$\begin{aligned} g_k(\varphi_i) - \frac{1-\sqrt{4k-3}}{2} g_k(\varphi_{i-1}) - \frac{2}{1-\sqrt{4k-3}} &= -\frac{\left(1+\sqrt{4k-3}\right)^2}{2\left(1-\sqrt{4k-3}\right)} \cdot \left(\frac{1+\sqrt{4k-3}}{2}\right)^{i-2} \\ &= -\frac{2}{1-\sqrt{4k-3}} \cdot \left(\frac{1+\sqrt{4k-3}}{2}\right)^{i}\,, \end{aligned}$$

$$\begin{aligned} g_k(\varphi_i) - \frac{1+\sqrt{4k-3}}{2} g_k(\varphi_{i-1}) - \frac{2}{1+\sqrt{4k-3}} &= -\frac{\left(1-\sqrt{4k-3}\right)^2}{2\left(1+\sqrt{4k-3}\right)} \cdot \left(\frac{1-\sqrt{4k-3}}{2}\right)^{i-2} \\ &= -\frac{2}{1+\sqrt{4k-3}} \cdot \left(\frac{1-\sqrt{4k-3}}{2}\right)^{i}\,. \end{aligned}$$

Combining the two equations above, we have:

$$g_k(\varphi_i) = \frac{\left(1+\sqrt{4k-3}\right)^{i+2} - \left(1-\sqrt{4k-3}\right)^{i+2}}{(k-1) \cdot 2^{i+2} \cdot \sqrt{4k-3}} - \frac{1}{k-1}\,.$$

Finally,

$$N_{\text{REUSE}}(\varphi_i) = (k+1) \cdot g_k(\varphi_i) = \frac{k+1}{k-1}\left(\frac{\left(1+\sqrt{4k-3}\right)^{i+2} - \left(1-\sqrt{4k-3}\right)^{i+2}}{2^{i+2} \cdot \sqrt{4k-3}} - 1\right). \tag{2}$$

Note that i is a constant for a given formula in the family we are considering. Therefore, according to Equations 1 and 2, $N_{\text{CLASSIC}}(\varphi_i) \sim O(k^i)$ and $N_{\text{REUSE}}(\varphi_i) \sim O(k^{\frac{i+1}{2}})$. To visualize the difference between the two, we plot $N_{\text{CLASSIC}}(\varphi_3)$ and $N_{\text{REUSE}}(\varphi_3)$ in Figure 5. It can be clearly observed that, as k increases, $N_{\text{REUSE}}(\varphi_3)$ grows significantly slower than $N_{\text{CLASSIC}}(\varphi_3)$.

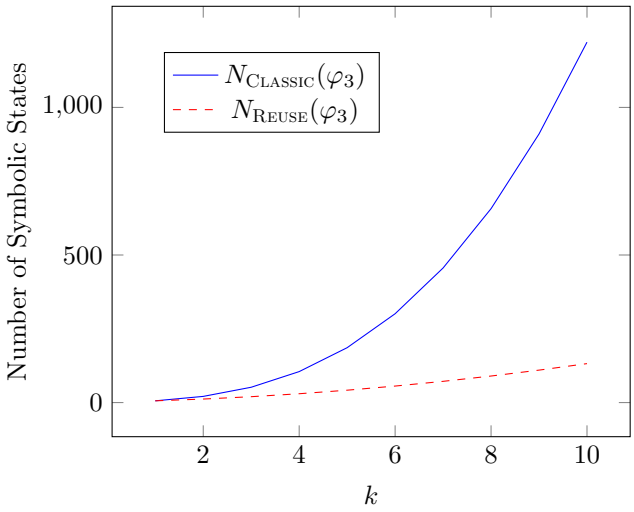


Figure 5: Comparing the growth of $N_{\text{CLASSIC}}(\varphi_3)$ and $N_{\text{REUSE}}(\varphi_3)$.

5 Coping with Models Containing Deadlock States

Generally, CTL assumes that the model does not contain any deadlock state. Unfortunately, this assumption is not true for many real-life models. For models containing deadlock states, using either CLASSIC or REUSE may fail to find a witness if deadlock states are part of every witness. Figure 6 shows a simple model containing a deadlock state 4. Suppose we are searching a witness for $\mathsf{EF}(a \wedge \mathsf{EF}b)$, which consists of two k-paths. There is no such witness when $k = 1$, because we must take two steps from the initial state to reach a state where a holds. When $k = 2$, the corresponding propositional formula is also unsatisfiable, because we are not able to build a path of length 2 from state 3.

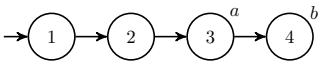


Figure 6: A model containing a deadlock state 4.

A common practice to cope with models containing deadlock states is to add a self-loop to every deadlock state. When using a SAT solver, we propose another approach that adds additional variables to the propositional formula, so that the formula allows k-paths whose actual length is smaller than k. Each symbolic state $\pi_i[j]$ is associated with a boolean variable $\tau_{i,j}$, called *transition flag*, which is *true* if and only if $\pi_i[j]$ is a true successor of $\pi_i[j-1]$, i.e., $\mathcal{N}(\pi_i[j-1], \pi_i[j])$. We modify the encoding of $[M_k]$ as follows:

$$[M_k]_\varphi = I(\pi_0[0]) \wedge \bigwedge_{i=0}^{f_k(\varphi)-1} \bigwedge_{j=0}^{k-1} (\mathcal{N}(\pi_i[j], \pi_i[j+1]) \vee \overline{\tau_{i,j+1}}) \wedge \bigwedge_{i=0}^{f_k(\varphi)-1} \bigwedge_{j=0}^{k-1} (\tau_{i,j+1} \Rightarrow \tau_{i,j}).$$

The constraint $\mathcal{N}(\pi_i[j], \pi_i[j+1]) \vee \overline{\tau_{i,j+1}}$ relaxes the assumption that $\pi_i[j]$ must have successors, by simply setting $\tau_{i,l}$ to *false* for $l \in \{j+1, ..., k\}$ if $\pi_i[j]$ is a deadlock state. The constraint $\bigwedge_{i=0}^{f_k(\varphi)-1} \bigwedge_{j=0}^{k-1} (\tau_{i,j+1} \Rightarrow \tau_{i,j})$ assures that, when $\tau_{i,j+1}$ is *true*, the corresponding

symbolic state $\pi_i[j+1]$ is reachable from $\pi_i[0]$, i.e., $\bigwedge_{l=0}^{j}\mathcal{N}(\pi_i[l],\pi_i[l+1])$, otherwise $\pi_i[j+1]$ can take an arbitrary state.

The translation of EX, EF, and EU formulas is also updated (we demonstrate the modification to Reuse; similar modification can also be applied to Classic):

$$
\begin{aligned}
[\mathsf{EX}\varphi, s]_k^i &= (s \equiv \pi_i[0]) \wedge [\varphi, \pi_i[1]]_k^{i+1} \wedge \tau_{i,1} \\
[\mathsf{E}(\varphi\mathsf{U}\rho), s]_k^i &= (s \equiv \pi_i[0]) \wedge \Big(\big([\rho, \pi_i[0]]_k^{i+1} \wedge \tau_{i,0} \big) \vee \textstyle\bigvee_{j=1}^{k} \big([\rho, \pi_i[j]]_k^{i+1} \wedge [\varphi, \pi_i[j-1]]_k^{i+1+g_k(\rho)} \\
&\qquad \wedge \textstyle\bigwedge_{t=0}^{j-2} [\mu(\varphi), \pi_i[t]]_k^{i+1+g_k(\rho)+g_k(\varphi)+t\cdot g_k(\mu(\varphi))} \wedge \tau_{i,j} \big) \Big) \\
[\mathsf{EG}\varphi, s]_k^i &= (s \equiv \pi_i[0]) \wedge \textstyle\bigvee_{j=0}^{k-1} (\pi_i[k] \equiv \pi_i[j]) \\
&\qquad \wedge [\varphi, \pi_i[k-1]]_k^{i+1} \wedge \textstyle\bigwedge_{j=0}^{k-2} [\mu(\varphi), \pi_i[j]]_k^{i+1+g_k(\varphi)+j\cdot g_k(\mu(\varphi))} \wedge \tau_{i,k}.
\end{aligned}
$$

Of course, if the model is known to be deadlock-free (using a priori knowledge, or some deadlock detection technique), the proposed modification should not be applied, as it increases the size of the resulting formulas.

6 Experiments

We describe our experimental design in Section 6.1 and present the results in Section 6.2.

6.1 Experimental Design

We implemented both Classic and Reuse in the model checker SMART [7], making use of the SAT solver Nigma [10, 11]. Our benchmark suite is a subset of models and CTL formulas from the CTL cardinality examination of the Model Checking Contest (MCC) 2018 (https://mcc.lip6.fr/). Models are described as Petri nets, most of which have one or more scaling parameters, affecting size and complexity. An experimental instance is a pair of a model instance and an ECTL formula. To select a set of instances eligible for our experiment, we apply the following filtering process:

1. Run SMART up to 1 hour to determine if the model instance is a safe Petri net (the maximum number of tokens at each place is 1). Discard unsafe Petri nets, so that the places in the remaining model instances can be represented as binary variables (while the places in bounded unsafe Petri nets can be represented using one-hot or binary encoding, we restrict ourselves to safe Petri nets for convenience). In practice, we used the results published in MCC 2018. This leaves 2,256 instances.

2. The formulas associated with the remaining model instances are first simplified if some subformulas can be determined to be *true* or *false*, using knowledge that the model instance is a safe Petri net. We remove constant *true* or *false* formulas and pure propositional formulas (leaving 1,288 instances). Then, the formulas are transformed into *negation normal form* (NNF) and ACTL formulas are negated to obtain ECTL ones. We remove non-ECTL formulas (leaving 845 instances), non-nested formulas (leaving 278 instances), and formulas that are instances of linear ECTL templates (leaving 110 instances). We also remove formulas containing summations, e.g., the total number of tokens in a set of places (leaving 89 instances). Therefore, the remaining experimental instances have non-trivial ECTL formulas that may have non-linear witnesses and do not contain summations.

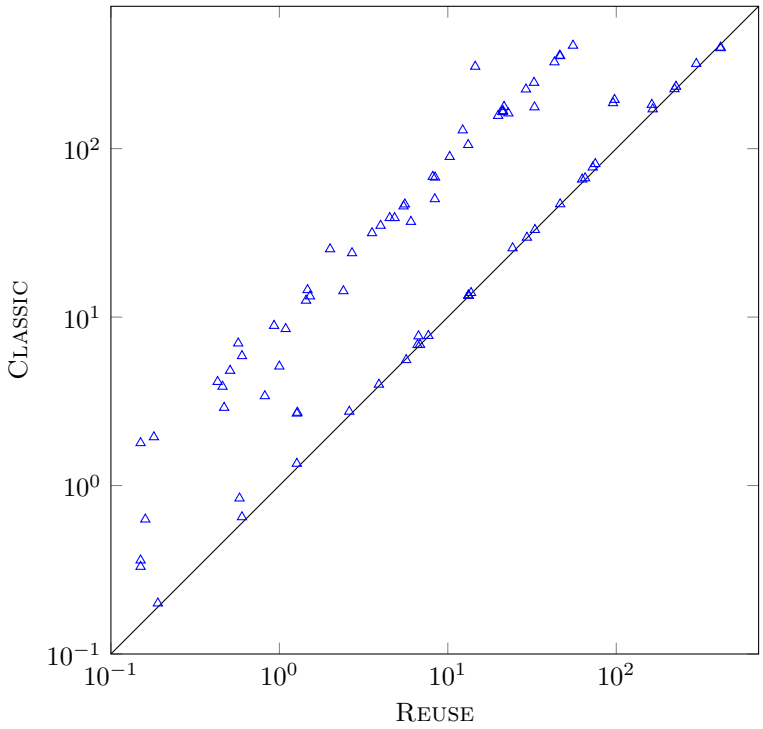


Figure 7: Comparing the total time (in seconds) spent on CNF transformation.

Finally, we have 89 instances, taken from 60 model instances from 22 different models. Among the 89 instances, the ECTL formulas hold in 50 instances, do not hold in 32 instances, while it is not known whether they hold in the remaining 7 instances, according to SMART.

To compare the performance of the two translation approaches, we run BMC up to $k = 20$. For each k, the SAT solver is given 10 minutes to work on the generated CNF formula. BMC terminates either if a witness is found for some k, or if, for every k up to 20, the SAT solver reports UNSAT or runs out of time. In the latter case, we cannot conclude satisfaction or violation of the corresponding ECTL formula, but we can tell that there is no witness up to the largest k for which the SAT solver reports UNSAT (no "simple" witness exists).

The model instances from MCC 2018 may contain deadlock states, thus we always apply the modification proposed in Section 5.

6.2 Experimental Results

First, we compare the time spent on transforming propositional formulas into CNF. Since Reuse never generates a larger propositional formula than Classic, it is expected to outperform Classic on this metric, and the results confirm our expectation. Figure 7 presents a logscale scatter plot comparing the total time (in seconds) spent on CNF transformation for each experimental instance. A data point above the diagonal means that CNF transformation completes faster using Reuse. From the plot, we can see that Reuse is often substantially better than Classic, and equal to it in the remaining cases, thus we conclude that CNF transformation always benefits from Reuse.

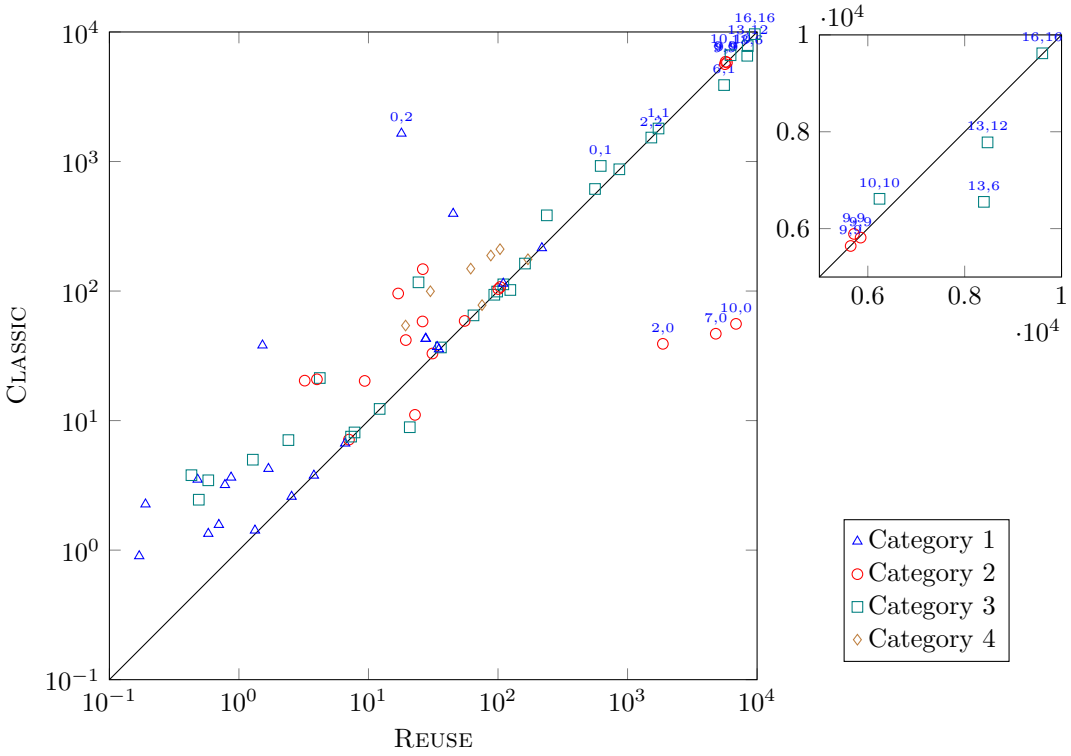


Figure 8: Comparing the total time (in seconds) spent on satisfiability checking.

Then, we compare the time spent on satisfiability checking. The results on each experimental instance are classified according to the following categories:

Category 1 (△) The ECTL formula holds, and at least one approach found a witness.

Category 2 (◯) The ECTL formula holds, but neither approach found a witness.

Category 3 (□) The ECTL formula does not hold.

Category 4 (♢) We do not know whether the ECTL formula holds or not.

Figure 8 presents a logscale scatter plot comparing the total time (in seconds) spent on satisfiability checking. It also has a zoom-in view of the top-right corner in linear scale for clarity. A data point above the diagonal means that the SAT solver terminates (reporting SAT or UNSAT, or running out of time) in a shorter total time, working on the propositional formulas generated by Reuse. The pair of numbers i, j above a data point (omitted if $0, 0$) report the number of timeouts for Reuse and Classic, respectively. For most instances, we can observe a better performance using Reuse.

Instances in Category 1 are those where we can take full advantage of BMC. For them, Reuse always found a witness faster. There is even one instance (the topmost △ in Figure 8) where Reuse found a witness but Classic did not. Beside the hardness of formulas, another reason why Reuse is more efficient is that it may find a witness using a smaller k, thus terminate earlier than Classic.

Most instances in Category 2 have relatively "deep" witnesses. BMC may find a witness running with a larger k, at the risk of working on a huge and complex propositional formula. The rightmost three ◯'s below the diagonal are instances where the SAT solver struggled with the formulas generated by Reuse. We noticed more timeouts using Reuse in these instances, which provides more evidence of the well-known fact that there is no strict connection between the size and the hardness of formulas for satisfiability checking [13].

In practice, BMC is not able to answer the instances in Category 3 and Category 4, since the upper bound of the maximal length of symbolic paths is the number of reachable states (see Theorem 2.2), often a huge number. In these cases, BMC can only tell us that no "simple" witness exists (up to the largest k for which the SAT solver reports UNSAT). We can see that most of the time, Reuse draws this conclusion faster than Classic.

Finally, we select an experiment instance from Category 1 for a detailed comparison for each value of k in Table 2. The model instance is AutoFlight-PT-05a and the formula is the negation of $\mathsf{A}((p33 \leq p79)\mathsf{U}\,\mathsf{AG}(p89 \leq p88))$. BMC finds a witness for $k = 17$ using Classic and for $k = 13$ using Reuse. **Vars**, **Clauses** and **Literals** are the numbers of variables, clauses, and literals in the CNF formulas. We can see that the CNF formulas generated by Reuse grow slower and are significantly smaller than the ones generated by Classic. **CNF** and **SAT** are the time (in seconds) spent in CNF transformation and satisfiability checking, respectively. CNF transformation always benefits from Reuse, but satisfiability checking may not. With Reuse, the SAT solver spends more time reporting UNSAT for $k = 8$, 9, 10, 11, and 12, though this disadvantage is finally offset by reporting SAT and terminating for a smaller k. An explanation could be that for some model checking problems, a small CNF formula may also have a small unsatisfiable core, which can be deep and hard for a SAT solver to identify. For example, suppose that we are searching a k-path π where $\pi[i] \models_k \mathsf{EF}a$ for any $i \in \{0, ..., k-1\}$. For the formula generated by Classic, the SAT solver reports UNSAT as long as it finds an i such that $\pi[i] \not\models_k \mathsf{EF}a$. However, for the formula generated by Reuse, it reports UNSAT only when it proves that $\pi[k-1] \not\models_k \mathsf{EF}a$.

In addition, the SAT solver reports SAT faster on a large formula ($k = 17$ for Classic; $k = 13$ for Reuse) than it reports UNSAT on a small formula ($k = 16$ for Classic; $k = 12$ for Reuse), which confirms that the hardness of satisfiable and unsatisfiable formulas should be evaluated using different criteria.

7 Conclusions and Future Work

We have presented an improved translation to propositional formulas for ECTL BMC, which generates smaller, or at worst the same formulas as the ones generated by Classic. Experimental results show that CNF transformation always benefits from our approach, and that satisfiability checking is more efficient most of the time. In addition, we proposed a simple modification to the translation so that it is also defined for models containing deadlock states.

BMC for ACTL formulas having linear counterexamples has been investigated in [20]. It seems promising to combine their work and ours, because our approach has advantages for formulas that are not instantiated from linear templates, thus the two approaches work on disjoint sets of ECTL and ACTL formulas and have no conflict in application.

In Def. 3.1, the sufficient predecessor formula of a disjunctive formula does not introduce any improvement: $\mu(\varphi \vee \rho) = \varphi \vee \rho$. However, improvement is possible if we "push down" disjunction. For example, $\mathsf{E}(a\mathsf{U}b) \vee \mathsf{E}(a\mathsf{U}c)$ can be rewritten into $\mathsf{E}(a\mathsf{U}(b \vee c))$, then $\mu(\mathsf{E}(a\mathsf{U}(b \vee c))) = a \vee b \vee c$. Therefore, simplifying and rewriting CTL formulas [3] may help path reuse achieve a broader applicability.

k	Classic					Reuse				
	Vars	**Clauses**	**Literals**	**CNF**	**SAT**	**Vars**	**Clauses**	**Literals**	**CNF**	**SAT**
1	2,357	27,108	56,460	0.04	0.00	2,357	27,108	56,460	0.04	0.00
2	5,384	70,968	147,217	0.12	0.02	4,172	53,661	111,547	0.08	0.01
3	9,353	131,733	272,576	0.23	0.06	5,985	80,213	166,632	0.13	0.02
4	14,266	209,405	432,541	0.41	0.06	7,798	106,766	221,719	0.18	0.03
5	20,123	303,984	627,112	0.59	0.09	9,611	133,320	276,808	0.25	0.05
6	26,924	415,470	856,289	0.82	0.15	11,424	159,875	331,899	0.28	0.06
7	34,669	543,863	1,120,072	1.18	0.22	13,237	186,431	386,992	0.35	0.22
8	43,358	689,163	1,418,461	1.53	0.27	15,050	212,988	442,087	0.4	0.61
9	52,991	851,370	1,751,456	1.68	0.32	16,863	239,546	497,184	0.45	1.67
10	63,568	1,030,484	2,119,057	2.20	0.39	18,676	266,105	552,283	0.52	2.76
11	75,089	1,226,505	2,521,264	2.91	1.64	20,489	292,665	607,384	0.59	5.04
12	87,554	1,439,433	2,958,077	3.03	2.61	22,302	319,226	662,487	0.61	28.66
13	100,963	1,669,268	3,429,496	3.67	2.30	24,115	345,788	717,592	0.64	5.95
14	115,316	1,916,010	3,935,521	4.00	28.05					
15	130,613	2,179,659	4,476,152	5.10	27.50					
16	146,854	2,460,215	5,051,389	5.28	220.14					
17	164,039	2,757,678	5,661,232	6.02	112.17					
Total				38.81	395.99				4.52	45.08

Table 2: Searching for a counterexample to $\mathsf{A}\,((p33 \leq p79)\mathsf{U}\,\mathsf{AG}(p89 \leq p88))$ in AutoFlight-PT-05a.

Acknowledgments

This work was supported in part by the National Science Foundation under grant ACI-1642397.

References

[1] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Advances in Computers*, 58(11):117–148, 2003.

[2] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *Proc. TACAS*, pages 193–207. Springer, 1999.

[3] Frederik Bønneland, Jakob Dyhr, Peter G Jensen, Mads Johannsen, and Jiří Srba. Simplification of CTL formulae for efficient model checking of Petri nets. In *International Conference on Applications and Theory of Petri Nets and Concurrency*, pages 143–163. Springer, 2018.

[4] Francesco Buccafurri, Thomas Eiter, Georg Gottlob, and Nicola Leone. On ACTL formulas having linear counterexamples. *Journal of Computer and System Sciences*, 62(3):463–515, 2001.

[5] Jerry R Burch, Edmund M Clarke, Kenneth L McMillan, David L Dill, and Lain-Jinn Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[6] Wei Chen and Wenhui Zhang. Bounded model checking of ACTL formulae. In *International Symposium on Theoretical Aspects of Software Engineering.*, pages 90–99. IEEE, 2009.

[7] Gianfranco Ciardo, Robert L. Jones, Andrew S. Miner, and Radu Siminiceanu. Logical and stochastic modeling with SMART. *Perf. Eval.*, 63:578–608, 2006.

[8] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. IBM Workshop on Logics of Programs*, LNCS 131, pages 52–71, 1981.

[9] Edmund M. Clarke, Somesh Jha, Yuan Lu, and Helmut Veith. Tree-like counterexamples in model checking. In *Proc. LICS*, pages 19–29. IEEE Comp. Soc. Press, 2002.

[10] Chuan Jiang and Gianfranco Ciardo. Nigma 1.2. In *SAT Race*, 2015.

[11] Chuan Jiang and Ting Zhang. Partial backtracking in CDCL solvers. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 490–502. Springer, 2013.

[12] Timo Latvala, Armin Biere, Keijo Heljanko, and Tommi Junttila. Simple bounded LTL model checking. In *International Conference on Formal Methods in Computer-Aided Design*, pages 186–200. Springer, 2004.

[13] Zoltán Ádám Mann. Typical-case complexity and the SAT competitions. In Daniel Le Berre, editor, *POS-14. Fifth Pragmatics of SAT workshop*, volume 27 of *EPiC Series in Computing*, pages 72–87. EasyChair, 2014.

[14] Rotem Oshman and Orna Grumberg. A new approach to bounded model checking for branching time logics. In *International Symposium on Automated Technology for Verification and Analysis*, pages 410–424. Springer, 2007.

[15] Wojciech Penczek, Bożena Woźna, and Andrzej Zbrzezny. Bounded model checking for the universal fragment of CTL. *Fundamenta Informaticae*, 51(1-2):135–156, 2002.

[16] David A Plaisted and Steven Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293–304, 1986.

[17] Grigori S Tseitin. On the complexity of derivation in propositional calculus. In *Automation of reasoning*, pages 466–483. Springer, 1983.

[18] András Vörös, Dániel Darvas, and Tamás Bartha. Bounded saturation-based CTL model checking. *Proceedings of the Estonian Academy of Sciences*, 62(1):59–70, 2013.

[19] Bow-Yaw Wang. Proving $\forall\mu$-calculus properties with SAT-based model checking. In *International Conference on Formal Techniques for Networked and Distributed Systems*, pages 113–127. Springer, 2005.

[20] Zhaowei Xu and Wenhui Zhang. Linear templates of ACTL formulas with an application to SAT-based verification. *Information Processing Letters*, 127:6–16, 2017.

[21] Andy Jinqing Yu, Gianfranco Ciardo, and Gerald Lüttgen. Bounded reachability checking of asynchronous systems using decision diagrams. In *Proc. TACAS*, LNCS 4424, pages 648–663, 2007.

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