IEICE TRANS. INF. & SYST., VOL.E97–D, NO.7 JULY 2014

PAPER

Bounded Strong Satisfiability Checking of Reactive System Specifications*

Masaya SHIMAKAWA[†a)], **Shigeki HAGIHARA**[†], *Nonmembers*, *and* **Naoki YONEZAKI**[†], *Member*

SUMMARY Many fatal accidents involving safety-critical reactive systems have occurred in unexpected situations that were not considered during the design and test phases of development. To prevent such accidents, reactive systems should be designed to respond appropriately to any request from an environment at any time. Verifying this property during the specification phase reduces development reworking. This property of a specification is commonly known as realizability. Realizability checking for reactive system specifications involves complex and intricate analysis. The complexity of realizability problems is 2EXPTIME-complete. To detect typical simple deficiencies in specifications efficiently, we introduce the notion of bounded strong satisfiability (a necessary condition for realizability), and present a method for checking this property. Bounded strong satisfiability is the property that, for all input patterns represented by loop structures of a given size k, there is a response that satisfies a given specification. We report a checking method based on a satisfiability solver, and show that the complexity of the bounded strong satisfiability problem is co-NEXPTIME-complete. Moreover, we report experimental results showing that our method is more efficient than existing approaches.

key words: *reactive system, bounded analysis, SAT solver, realizability, LTL specification*

1. Introduction

A reactive system is one that responds to requests from an environment in a timely fashion. The systems used to control elevators or vending machines are typical examples of reactive systems. Many safety-critical systems, such as those that control nuclear power plants or air traffic control systems, are also considered reactive systems. In designing a system of this kind, the requirements should be analyzed and then described as specifications for the system. If a specification has a flaw, such as inappropriate case-splitting, the developed system may encounter unintended situations. Indeed, fatal accidents involving safety-critical reactive systems have occurred as a result of unexpected situations that were not considered during the design and test phases of development. Therefore, it is important to ensure that a specification does not contain this kind of flaw [1].

More precisely, a reactive system specification must have a model that can respond in a timely fashion to any request at any time. This property, called realizability, was introduced in [2], [3]. It has been demonstrated that a reactive system can be synthesized from a realizable specification [3]; however, realizability checking for reactive system specifications involves complex and intricate analysis. Therefore, the size of specifications that can be checked in a practical application is strongly limited.

Here we present a bounded checking method to detect typical simple deficiencies in specifications. In bounded checking, we verify the existence of a counterexample (or witness) of a given size k. Such methods have been applied successfully in other fields; for example, bounded model checking [4] and the software tool Alloy Analyzer [1]. The principle advantage of bounded checking is the ability to efficiently detect a small counterexample (or witness).

We propose the notion of a bounded property for strong satisfiability [5] (a necessary condition for realizability), termed bounded strong satisfiability, together with a method for checking this property. Strong satisfiability is the property that for any input sequence there is a response that satisfies a given specification. Strong satisfiability can be checked with less complexity than realizability [6]. Although this property is a necessary condition, many practical unrealizable specifications are also strongly unsatisfiable [7]. Bounded strong satisfiability restricts the input sequences of strong satisfiability to those represented by loop structures of size k. It follows that, for simple input patterns, there is a response that satisfies the specification. Our experience has shown that, in many instances, strongly unsatisfiable specifications have small counterexamples (which are input sequences that can be represented by small loop structures). Thus, we anticipate that many deficiencies can be detected by checking for this property.

In our method for checking bounded strong satisfiability, we use a SAT solver. Specifically, we first construct a non-deterministic Büchi automaton (NBA) that accepts input sequences for which there is a response that satisfies a specification. We then check whether the NBA accepts all loop structures of size k (k-universally acceptable), using a SAT solver. To accomplish this, checking the existence of a loop structure that is not accepted by NBA is reduced to a SAT problem. This reduction is based on the following characterization of non-accepted loop structures: a loop structure σ of bounded size is not accepted by NBA if and only if, for any run on σ, final states occur not more than d times for some d. This characterization is valid because only bounded loop structures are considered.

Moreover, we show that the complexity of bounded

Manuscript received December 25, 2013.

[†]The authors are with the Department of Computer Science, Tokyo Institute of Technology, Tokyo, 152–8552 Japan.

*This work is based on "SAT-Based Bounded Strong Satisfiability Checking of Reactive System Specifications", by M. Shimakawa, S. Hagihara, and N. Yonezaki which appeared in the proceeding of International Conference on Information and Communication Technology, ICT-EurAsia 2013.

a) E-mail: masaya@fmx.cs.titech.ac.jp

DOI: 10.1587/transinf.E97.D.1746

strong satisfiability for a specification written in linear temporal logic (LTL) is co-NEXPTIME-complete. Because the complexity of the realizability problem is 2EXPTIME-complete [8], and that of the strong satisfiability problem is EXPSPACE-complete [6], bounded strong satisfiability offers the advantage of less complex analysis compared with realizability and strong satisfiability.

We implemented our method, and found that it can handle larger specifications than existing techniques based on other properties, and can detect deficiencies efficiently.

The remainder of this paper is organized as follows. In Sect. 2, we introduce the concepts of reactive systems, LTL as a specification language, and strong satisfiability. In Sect. 3, we give the notion of bounded strong satisfiability. In Sect. 4, we describe a procedure for checking bounded strong satisfiability. In Sect. 5, we present a SAT-based method for checking the bounded universality of an NBA. In Sect. 6, we show that the complexity of the bounded strong satisfiability for a specification in LTL is co-NEXPTIME-complete. In Sect. 7, we describe experimental results. In Sect. 8, we compare our method with existing approaches. We present our conclusion in Sect. 9.

2. Preliminaries

2.1 Reactive Systems

A reactive system is one that responds to requests from an environment in a timely fashion.

Definition 1 (Reactive system): *A reactive system RS* is a triple $\langle X, Y, r\rangle$, where X is a set of events caused by an environment, Y is a set of events caused by the system, and $r : (2^X)^+ \to 2^Y$ is a reaction function.

We refer to events caused by the environment as 'input events,' and those caused by the system as 'output events.' The set $(2^X)^+$ is the set of all finite sequences of sets of input events. A reaction function r relates sequences of sets of previously occurring input events with a set of current output events.

2.2 A Language for Describing Reactive System Specifications

The timing of input and output events is an essential element of reactive systems, which can be described using linear temporal logic (LTL). We use LTL to describe the specifications of reactive systems here, and treat input and output events as atomic propositions.

2.2.1 Syntax

Formulae in LTL are inductively defined as follows:

- Atomic propositions (i.e., input events and output events) are formulae.
- $f \wedge g$, $\neg f$, $\mathbf{X}f$, $f\mathbf{U}g$ are formulae if f and g are formulae.

The notation $\mathbf{X}f$ means that 'f holds the next time,' while $f\mathbf{U}g$ means that 'f always holds until g holds.' The notations $f \vee g$, $f \to g$, $f \leftrightarrow g$, $f \oplus g$, $\top$, $\mathbf{F}f$, and $\mathbf{G}f$ are abbreviations for $\neg(\neg f \wedge \neg g)$, $\neg(f \wedge \neg g)$, $\neg(f \wedge \neg g) \wedge \neg(\neg f \wedge g)$, $\neg(f \leftrightarrow g)$, $\neg\bot$, $\top\mathbf{U}f$, and $\neg\mathbf{F}\neg f$ respectively, where $\bot$ is an atomic proposition representing 'falsity.'

2.2.2 Semantics

A behavior is an infinite sequence of sets of events. Let i be an index such that $i \geq 0$. The i-th set of a behavior σ is denoted by $\sigma[i]$. The i-th suffix of a behavior σ is denoted by $\sigma[i\ldots]$. When a behavior σ satisfies a formula f, we write $\sigma \models f$, and inductively define this relation as follows:

- $\sigma \models p$ iff $p \in \sigma[0]$
- $\sigma \not\models \bot$
- $\sigma \models f \wedge g$ iff $\sigma \models f$ and $\sigma \models g$
- $\sigma \models \neg f$ iff $\sigma \not\models f$
- $\sigma \models \mathbf{X}f$ iff $\sigma[1\ldots] \models f$
- $\sigma \models f\mathbf{U}g$ iff $\exists j \geq 0.((\sigma[j\ldots] \models g)$ and $\forall k(0 \leq k < j.\ \sigma[k\ldots] \models f))$

We say that f is satisfiable if there exists a σ that satisfies f.

2.3 Properties of Reactive System Specifications

It is important for reactive system specifications to satisfy realizability. Realizability requires the existence of a reactive system such that, for any input events with any timing, the system produces output events such that the specification holds.

Definition 2 (Realizability): A specification *Spec* is *realizable* if the following holds:

$$\exists RS \forall \tilde{a}(behave_{RS}(\tilde{a}) \models Spec),$$

where $\tilde{a}$ is an infinite sequence of sets of input events, i.e., $\tilde{a} \in (2^X)^\omega$. $behave_{RS}(\tilde{a})$ is the infinite behavior of $\tilde{a}$ caused by *RS*, defined as follows. If $\tilde{a} = a_0a_1\ldots$, $behave_{RS}(\tilde{a}) = (a_0 \cup b_0)(a_1 \cup b_1)\ldots$, where b_i is a set of output events caused by *RS*, i.e., $b_i = r(a_0 \ldots a_i)$.

The following property was shown to be a necessary condition for realizability in [5].

Definition 3 (Strong satisfiability): A specification *Spec* is *strongly satisfiable* if the following holds:

$$\forall \tilde{a} \exists \tilde{b}(\langle \tilde{a}, \tilde{b}\rangle \models Spec),$$

where $\tilde{b}$ is an infinite sequence of sets of output events, i.e., $\tilde{b} \in (2^Y)^\omega$. If $\tilde{a} = a_0a_1\ldots$ and $\tilde{b} = b_0b_1\ldots$, then $\langle \tilde{a}, \tilde{b}\rangle$ is defined by $\langle \tilde{a}, \tilde{b}\rangle = (a_0 \cup b_0)(a_1 \cup b_1)\ldots$.

Intuitively, strong satisfiability is the property that, if a reactive system is given an infinite sequence of sets of future input events, the system can determine an infinite sequence of sets of future output events. Strong satisfiability is a necessary condition for realizability, i.e., all realizable specifications are strongly satisfiable. Conversely, many practical

strongly satisfiable specifications are also realizable.

Example 1: The following is a specification of a control system for a door. Let us consider a simple example of a door control system. The initial specification is as follows.

1. The door has two buttons: an open button and a close button.
2. If the open button is pushed, the door eventually opens.
3. While the close button is pushed, the door remains shut.

The events 'the open button is pushed' and 'the close button is pushed' are both input events. We denote these events by x_1 and x_2, respectively. The event 'the door is open (closed)' is an output event. We denote this event by y (resp., $\neg y$). The initial specification is then represented by $Spec_1 : \mathbf{G}((x_1 \rightarrow \mathbf{F}y) \wedge (x_2 \rightarrow \neg y))$ in LTL. This specification is not strongly satisfiable, and consequently unrealizable, due to the fact that there is no response that satisfies $Spec_1$ for the environmental behavior in which the close button is still being pushed after the open button has been pushed. Formally, for $\tilde{a} = \{x_1, x_2\}\{x_2\}\{x_2\}\ldots$, $\exists\tilde{b}(\langle\tilde{a}, \tilde{b}\rangle \models Spec_1)$ does not hold. Hence $\forall\tilde{a}\exists\tilde{b}(\langle\tilde{a}, \tilde{b}\rangle \models Spec_1)$ does not hold.

However, suppose that constraint 3 in the initial specification can be weakened to 3':

3'. If the close button is pushed, the door eventually closes.

Then the modified specification is represented by $\mathbf{G}((x_1 \rightarrow \mathbf{F}y) \wedge (x_2 \rightarrow \mathbf{F}\neg y))$, and this is both strongly satisfiable and realizable.

3. Bounded Strong Satisfiability

In this section, we describe the notion of bounded strong satisfiability. This property is a restricted version of strong satisfiability, in which only input sequences represented by loop structures of size k are considered, as in the case of bounded model checking (e.g., [4]).

Definition 4 (k-loop): Let $k, l \in \mathbb{N}$ and $l \leq k$. An infinite sequence σ is a (k, l)-*loop* if there exists $u = s_0 s_1 \ldots s_{l-1}$ and $v = s_l s_{l+1} \ldots s_k$ such that $\sigma = u \cdot v^\omega$. An infinite sequence σ is a k-*loop* if there exists an l such that σ is a (k, l)-loop.

Definition 5 (Bounded strong satisfiability): Let $k \in \mathbb{N}$. A specification *Spec* is *k-strongly satisfiable* if the following holds:

$$\forall\tilde{a}(\tilde{a} \text{ is } k\text{-loop} \implies \exists\tilde{b}(\langle\tilde{a}, \tilde{b}\rangle \models Spec)).$$

If an infinite sequence is a k-loop and $k < k'$, then the sequence is a k'-loop. Therefore, the following holds:

Theorem 1: Let $k < k'$. If a specification *Spec* is k'-strongly satisfiable, then *Spec* is also k-strongly satisfiable.

It is clear from this definition that, if a specification *Spec* is strongly satisfiable, then *Spec* is also k-strongly satisfiable. Moreover, if *Spec* is described in LTL and the bound k is sufficiently large, then the converse is also true[†].

[†]This is derived from the fact that *Spec* can be represented by a finite state Büchi automaton.

Theorem 2: For all $k \in \mathbb{N}$, if a specification *Spec* is strongly satisfiable, then *Spec* is also k-strongly satisfiable.

Theorem 3: If a specification *Spec* described in LTL is not strongly satisfiable, then *Spec* is not k-strongly satisfiable for some k.

Proof. There exists a non-deterministic Büchi automaton (NBA) $\mathcal{A}'$ such that $L(\mathcal{A}') = \{\tilde{a} \mid \exists\tilde{b}(\langle\tilde{a}, \tilde{b}\rangle \models Spec)\}$ [6], [9][††]. Moreover, we can obtain an NBA $\overline{\mathcal{A}'}$ such that $L(\overline{\mathcal{A}'}) = \{\tilde{a} \mid \neg\exists\tilde{b}(\langle\tilde{a}, \tilde{b}\rangle \models Spec)\}$, because the class of NBA-recognizable languages is closed under complementation [10]. Assume that *Spec* is not strongly satisfiable. Then $\overline{\mathcal{A}'}$ is non-empty. If an NBA is non-empty, then there exists a sequence $\sigma = u \cdot v^\omega$ (where u, v are finite sequences) such that σ is accepted by the NBA [10]. Hence, there exists a k-loop $\tilde{a}$ for some k such that $\tilde{a} \in L(\overline{\mathcal{A}'})$. That is, *Spec* is not k-strongly satisfiable for some k. □

Example 2: The specification $Spec_1$ in Example 1 is not 1-strongly satisfiable, and consequently not k-strongly satisfiable for all $k > 1$ because $\tilde{a} = \{x_1, x_2\}\{x_2\}\{x_2\}\ldots$ is a 1-loop$(\{x_1, x_2\}\{x_2\}^\omega)$, which does not satisfy $\exists\tilde{b}(\langle\tilde{a}, \tilde{b}\rangle \models Spec_1)$.

By checking whether a specification satisfies bounded strong satisfiability, we can know whether the specification has simple input patterns (represented by small loops) that cannot satisfy the specification. As the experiment in Sect. 7 shows, many practical defective specifications have such simple input patterns. Thus, we anticipate that checking bounded strong satisfiability will find many practical deficiencies in reactive system specifications.

4. Procedure for Checking Bounded Strong Satisfiability

In this section, we describe a procedure for checking bounded strong satisfiability using non-deterministic Büchi automata. This procedure is based on the procedure for (unbounded) strong satisfiability described in [9].

A *non-deterministic Büchi automaton* (NBA) is a tuple $\mathcal{A} = \langle\Sigma, Q, q_I, \delta, F\rangle$, where Σ is an alphabet, Q is a finite set of states, q_I is an initial state, $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation, and $F \subseteq Q$ is a set of final states. A run of $\mathcal{A}$ on an ω-word $\sigma = \sigma[0]\sigma[1]\ldots$ is an infinite sequence $\varrho = \varrho[0]\varrho[1]\ldots$ of states, where $\varrho[0] = q_I$ and $(\varrho[i], \sigma[i], \varrho[i+1]) \in \delta$ for all $i \geq 0$. We say that $\mathcal{A}$ accepts σ, if there is a run ϱ on σ such that $Inf(\varrho) \cap F \neq \emptyset$ holds, where $Inf(\varrho)$ is the set of states that occurs infinitely often in ϱ. The set of ω-words accepted by $\mathcal{A}$ is called the language accepted by $\mathcal{A}$, and is denoted by $L(\mathcal{A})$.

Let *Spec* be a specification written in LTL. We can check the bounded strong satisfiability of *Spec* via the following procedure.

1. We obtain an NBA $\mathcal{A} = \langle 2^{X \cup Y}, Q, q_I, \delta, F\rangle$ s.t. $L(\mathcal{A}) =$

[††]In next section, we give the construction method.

$\{\sigma \mid \sigma \models Spec\}$ holds.
2. Let $\mathcal{A}' = \langle 2^X, Q, q_I, \delta', F\rangle$ be the NBA obtained by restricting $\mathcal{A}$ to only input events, where $\delta' = \{(q, a, q') \mid \exists b\ (q, a \cup b, q') \in \delta\}$. Note that $L(\mathcal{A}') = \{\tilde{a} \mid \exists \tilde{b} \langle \tilde{a}, \tilde{b}\rangle \in L(\mathcal{A})\}$ holds due to the definition of δ'.
3. We check whether $\mathcal{A}'$ accepts all k-loops (*i.e., is k-universally acceptable*). If it is k-universally acceptable, we conclude that *Spec* is k-strongly satisfiable. Otherwise, we conclude that *Spec* is not k-strongly satisfiable.

The construction of the NBA in step 1 can be found in [11]. For bounded universality checking in step 3, we take a SAT-based approach.

5. SAT-Based Bounded Universality Checking for NBA

Here we describe a SAT-based method for checking the bounded universality of an NBA. In this method, the complement of the bounded universality checking problem is reduced to a SAT problem.

5.1 Characterization of Non-Accepted k-Loops

As a preliminary to the reduction, we characterize the k-loops that are not accepted by NBA, based on the notion of a run graph. This characterization is similar to that of the sequence that is not accepted by NBA [12] (which may not be a k-loop), and to that of the transition system accepted by universal co-Büchi automaton [13].

Definition 6 (run graph): Let $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F\rangle$ be an NBA and $\sigma = s_0 \dots s_{l-1}\ (s_l \dots s_k)^\omega$ be a (k, l)-loop. The *run graph* for $\mathcal{A}$ and σ is $G = (V, v_I, E, C)$: $V := Q \times \{0, 1, \dots, k\}$ (the set of nodes). $v_I := (q_I, 0)$ (the initial node). $E := \{((q, i), (q', suc(i))) \mid (q, s_i, q') \in \delta\}$, where $suc(i) = l$ if $i = k$, and $suc(i) = i + 1$ otherwise (the set of edges). $C := \{(q, i) \mid q \in F, 0 \le i \le k\}$ (the set of final nodes).

An NBA does not accept a (k, l)-loop σ if and only if there does not exist a run ϱ on σ such that $Inf(\varrho) \cap F \neq \emptyset$ holds; i.e., for all runs, the number of occurrences of final states in the run is finite. For $i \le k$, a run on $\sigma[i \dots]$ from a state q corresponds to a path in the run graph from (q, i). Therefore the following holds:

Theorem 4: An NBA $\mathcal{A}$ does not accept a (k, l)-loop σ if and only if, for all paths from the initial node in the run graph for $\mathcal{A}$ and σ, the number of occurrences of final nodes in the path is finite.

The number of final nodes in a run graph is bounded. From this, we obtain the following result:

Lemma 1: Let $d \ge |C|$. If for all paths $\tilde{v}$ in a run graph G from a node v, the number of occurrences of final nodes in $\tilde{v}$ is finite, then for all paths $\tilde{v}$ from v in G, final nodes occur at most d times in $\tilde{v}$.

Proof. We prove the contraposition. Assume that the number of occurrences of final nodes in a path is more than d. Because $d \ge |C|$, there exists a final state q_c that occurs at least twice in the path. This means that there exists a path from q_c to q_c, from which it follows that there is a path on which the final state q_c occurs infinitely often. □

The property that "for all paths $\tilde{v}$ from v, the number of occurrences of final nodes is at most j" (denoted by $AtMost(v, j)$) is characterized as follows: For $v \in V \setminus C$ (for $v \in C$), $AtMost(v, j)$ holds if and only if for all successors $v' \in vE$, $AtMost(v', j)$ holds ($AtMost(v', j-1)$ holds). In addition, for all $v \in C$, $AtMost(v, 0)$ does not hold. Based on this idea, the following result can be proved:

Theorem 5: Let $G = (V, v_I, E, C)$ be a run graph and $d \in \mathbb{N}$. For all paths $\tilde{v}$ from v_I in G, final nodes occur at most d times if and only if there exist a sequence $V_0, V_1, \dots, V_d$ of sets of nodes such that the following are true:

1. The following condition (denoted by $I(V_0)$) holds:
$$v \in V_0 \iff \begin{cases} \forall v' \in vE.\ v' \in V_0 & \text{if } v \in V \setminus C \\ \bot & \text{if } v \in C \end{cases}$$
2. For all $0 \le j < d$, the following condition (denoted by $T(V_j, V_{j+1})$) holds:
$$v \in V_{j+1} \iff \begin{cases} \forall v' \in vE.\ v' \in V_{j+1} & \text{if } v \in V \setminus C \\ \forall v' \in vE.\ v' \in V_j & \text{if } v \in C \end{cases}$$
3. $v_I \in V_d$ holds (denoted by $F(V_d)$).

Proof. ($\Rightarrow$) Assume that final nodes occur at most d times for all paths $\tilde{v}$ from v_I in G. Let V_j be $\{v \mid$ final nodes occur at most j times for all paths from $v\}$ for $0 \le j \le d$. Then, $I(V_0) \wedge \bigwedge_{0 \le j < d} T(V_j, V_{j+1}) \wedge F(V_d)$ holds.
($\Leftarrow$) Assume that there exists a sequence $V_0, V_1, \dots, V_d$ of sets of nodes such that $I(V_0) \wedge \bigwedge_{0 \le j < d} T(V_j, V_{j+1}) \wedge F(V_d)$ holds, and assume that there exists a path $\tilde{v} = v_0 v_1 \dots$ from v_I in G such that final nodes occur more than d times for $\tilde{v}$. Let v_{i_x} be the x-th node that is in C for $1 \le x \le d+1$. Because $F(V_d)$ and $\bigwedge_{0 \le j < d} T(V_j, V_{j+1})$ hold, $v_i \in V_d$ for $0 \le i \le i_1$, and if $1 \le x \le d$, then $v_i \in V_{d-x}$ for all $i_x < i \le i_{x+1}$ holds. Hence $v_{i_{d+1}} \in V_0$. Because $I(V_0)$ holds, $v_{i_{d+1}} \notin C$ should be satisfied. This contradicts $v_{i_{d+1}} \in C$. □

We can summarize the characterization of non-accepted k-loops by the following:

Theorem 6: Let $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F\rangle$ be an NBA and $k \in \mathbb{N}$. For all $d \in \mathbb{N}$, (2) implies (1), and for $d \ge (k+1) \cdot |F|$, (1) implies (2), where (1) and (2) are as follows:

(1) There exists a k-loop that is not accepted by $\mathcal{A}$.
(2) There exists a k-loop σ such that for some sequence $V_0, V_1, \dots, V_d$ of sets of nodes of the run graph G for $\mathcal{A}$ and σ, $I(V_0) \wedge \bigwedge_{0 \le j < d} T(V_j, V_{j+1}) \wedge F(V_d)$ holds.

Example 3: Consider an NBA $\mathcal{A}_{ex}$ illustrated in Fig. 1, and a $(1, 0)$-loop $\sigma_{ex} = (ab)^\omega$. Note that $\mathcal{A}_{ex}$ does not accept σ_{ex}. The run graph for $\mathcal{A}_{ex}$ and σ_{ex} is illustrated in Fig. 2.

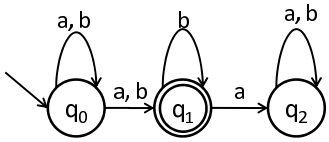

Fig. 1 NBA $\mathcal{A}_{ex}$.

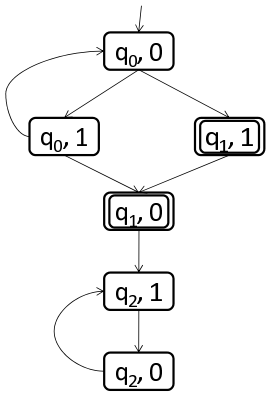

Fig. 2 Run graph for $\mathcal{A}_{ex}$ and σ_{ex}.

For $d = 2$ ($AtMost(v_I, d)$ holds), the sequence V_0, V_1, V_2, which consists of the following sets, satisfies the condition $I(V_0) \wedge \bigwedge_{0 \le j < d} T(V_j, V_{j+1}) \wedge F(V_d)$.

$$V_0 = \{(q_2, 0), (q_2, 1)\}$$
$$V_1 = \{(q_1, 0), (q_2, 0), (q_2, 1)\}$$
$$V_2 = \{(q_0, 0), (q_0, 1), (q_1, 1), (q_1, 0), (q_2, 0), (q_2, 1)\}$$

For $d = 1$ ($AtMost(v_I, d)$ does not hold), there is no sequence V_0, V_1 such that the condition holds.

5.2 Reduction to SAT

We describe a reduction to SAT based on Theorem 6. That is, for an NBA $\mathcal{A}$ and k, we construct a propositional formula $|[notAcc(\mathcal{A}, k, d)]|$ such that condition (2) of Theorem 6 holds if and only if $|[notAcc(\mathcal{A}, k, d)]|$ is satisfiable.

5.2.1 Variables

To represent a (k, l)-loop and $V_0, V_1, \ldots, V_d$, we introduce the following variables (assuming that $\Sigma = 2^P$): (a) p_i for each $p \in P$, $0 \le i \le k$, which indicate whether the i-th element s_i of a k-loop satisfies $p \in s_i$; (b) l_i for $0 \le i \le k$, which indicate whether the i-th element follows the k-th element; and (c) $v^j_{(q,i)}$ for $q \in Q, 0 \le i \le k, 0 \le j \le d$, which indicate whether $(q, i) \in V_j$ holds.

5.2.2 Constraint

To represent the concept "be a k-loop correctly", we define the formula $|[loop(k)]| := \bigvee_{0 \le i \le k} l_i \wedge \bigwedge_{0 \le i \le k} (l_i \rightarrow \bigwedge_{0 \le i' \le k, i' \ne i} \neg l_{i'})$.

To represent $I(V_0) \wedge \bigwedge_{0 \le j < d} T(V_j, V_{j+1}) \wedge F(V_d)$, we define the formulae $|[I(\mathcal{A}, k)]|_0$, $|[T(\mathcal{A}, k)]|_{j,j+1}$ and $|[F(\mathcal{A}, k)]|_d$, which indicate that $I(V_0)$, $T(V_i, V_{i+1})$ and $F(V_d)$ hold, respectively. Let $\mathcal{A} = \langle 2^P, Q, q_I, \delta, F \rangle$ be an NBA and $k \in \mathbb{N}$. These formulae are defined in Table 1, where $|[a]|_i$ is the formula $\bigwedge_{p \in a} p_i \wedge \bigwedge_{p \notin a} \neg p_i$, indicating that the i-th element of σ is a, and $|[suc]|^j_{(q,i)}$ is the formula $\bigwedge_{0 \le i' \le k} l_{i'} \rightarrow v^j_{(q,i')}$ if $i = k$, $v^j_{(q,i+1)}$ otherwise.

The formula $|[notAcc(\mathcal{A}, k, d)]|$ is defined by $|[notAcc(\mathcal{A}, k, d)]| := |[loop(k)]| \wedge |[I(\mathcal{A}, k)]|_0 \wedge \bigwedge_{0 \le j < d} |[T(\mathcal{A}, k)]|_{j,j+1} \wedge |[F(\mathcal{A}, k)]|_d$.

Theorem 7: Let $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ and $k \in \mathbb{N}$. For all $d \in \mathbb{N}$, (2) implies (1), and for $d \ge (k + 1) \cdot |F|$, (1) implies (2), where (1) and (2) are as follows:

(1) There exists a k-loop which is not accepted by $\mathcal{A}$.
(2) $|[notAcc(\mathcal{A}, k, d)]|$ is satisfiable.

Theorem 8: Let $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ and $k, d \in \mathbb{N}$. The size of $|[notAcc(\mathcal{A}, k, d)]|$ is $O(k^2 + k \cdot d \cdot |\delta|)$. Checking that $\mathcal{A}$ is not k-universally acceptable can be reduced to the SAT problem for a formula of size $O(k^2 \cdot |F| \cdot |\delta|)$.

5.3 Improvement

5.3.1 Incremental Checking

Even for small d, if $|[notAcc(\mathcal{A}, k, d)]|$ is satisfiable, then there exists a k-loop that is not accepted by $\mathcal{A}$ (by Theorems 6 and 7). The smaller the value of d, the smaller the size of $|[notAcc(\mathcal{A}, k, d)]|$ and the lower the checking cost. Therefore, it is effective to check whether $|[notAcc(\mathcal{A}, k, d)]|$ is satisfiable incrementally for $d = 0, 1, \ldots, (k + 1) \cdot |F|$. An incremental approach reduces the cost of finding a k-loop that is not accepted by $\mathcal{A}$.

Moreover, by using the induction technique for incremental SAT-based unbounded reachability checking in [14], [15], at the stage whereby d is less than $(k + 1) \cdot |F|$, we judge that $\mathcal{A}$ is k-universally acceptable. The condition of Theorem 6 can be regarded as the reachability problem of a transition system for which the initial condition is I, the transition relation is T, and the final condition is F. It is relatively straightforward to apply the induction technique described in [14], [15] to our method.

5.3.2 Reduction Based on a Modified Run Graph

Checking that an NBA is not k-universally acceptable can be also reduced to the SAT problem for a formula of size $O(k^2 + k \cdot |Q| \cdot |\delta|)$ by modifying the construction of the run graph as follows.

We add a check bit to each node: $V' := Q \times \{0, 1, \ldots, k\} \times \{\top, \bot\}$. The check bit indicates whether final nodes occurred before the node was reached in each iteration. Then, the set of edges is as follows: $E' := \{((q, i, b), (q', suc(i), sucb(i, b, q'))) \mid (q, s_i, q') \in \delta\}$, where $sucb(i, b, q') = b \vee (q' \in F)$ if $i < k$, and $sucb(i, b, q') =$

Table 1 The definitions of $|[I(\mathcal{A},k)]|_0$, $|[T(\mathcal{A},k)]|_{j,j+1}$ and $|[T(\mathcal{A},k)]|_{j,j+1}$.

$\|[I(\mathcal{A},k)]\|_0$	$\bigwedge_{\substack{q\in Q\setminus F,\\ 0\le i\le k}} \left(v^0_{(q,i)} \leftrightarrow \bigwedge_{(q,a,q')\in\delta} (\|[a]\|_i \rightarrow \|[suc]\|^0_{(q',i)}) \right) \wedge \bigwedge_{\substack{q\in F,\\ 0\le i\le k}} \left(\neg v^0_{(q,i)}\right)$
$\|[T(\mathcal{A},k)]\|_{j,j+1}$	$\bigwedge_{\substack{q\in Q\setminus F,\\ 0\le i\le k}} \left(v^{j+1}_{(q,i)} \leftrightarrow \bigwedge_{(q,a,q')\in\delta} (\|[a]\|_i \rightarrow \|[suc]\|^{j+1}_{(q',i)}) \right) \wedge$ $\bigwedge_{\substack{q\in F,\\ 0\le i\le k}} \left(v^{j+1}_{(q,i)} \leftrightarrow \bigwedge_{(q,a,q')\in\delta} (\|[a]\|_i \rightarrow \|[suc]\|^{j}_{(q',i)}) \right)$
$\|[F(\mathcal{A},k)]\|_d$	$v^d_{(q_1,0)}$

$(q' \in F)$ otherwise. We set $C' := \{(q,k,\top) \mid q \in Q\}$.

The modified run graph has the same features as the normal run graph. The size $|V'|$ is also $O(k \cdot |Q|)$. But the size $|C'|$ is $|Q|$, while the size $|C|$ is $(k+1)\cdot|F|$. Thus $O(k^2 + k \cdot |Q| \cdot |\delta|)$-encoding is possible.

6. Complexity

In this section, we show that the bounded strong satisfiability problem for a specification written in LTL and a non-negative integer k encoded in binary (i.e., we consider the size of k to be $\lfloor \log k \rfloor + 1$) is co-NEXPTIME-complete. In other words, (1) the complement problem of the bounded strong satisfiability problem is in the class NEXPTIME (the class of problems solvable in $O(2^{p(n)})$ time by a non-deterministic Turing machine, where $p(n)$ is a polynomial function of the input size n), and (2) all problems in NEXPTIME are reducible to the complement of the bounded strong satisfiability problem.

6.1 Upper Bound

Theorem 9: The bounded strong satisfiability problem for a specification written in LTL and a non-negative integer k encoded in binary is in the complexity class co-NEXPTIME.

Proof. We prove that the complement version of the procedure in Sect. 4 is accomplished in $O(2^{p(n)})$ time by a non-deterministic Turing machine. $\mathcal{A}$ can be constructed within $O(2^{|Spec|})$ time, even by a deterministic Turing machine, and the size of $\mathcal{A}$ is also $O(2^{|Spec|})$ [11], where $|Spec|$ is the length of *Spec*. Since $\mathcal{A}'$ is obtained by projection, $\mathcal{A}'$ can be constructed within $O(|\mathcal{A}|)$ time, even by a deterministic Turing machine, and the size of $\mathcal{A}'$ is $O(|\mathcal{A}|)$, where $|\mathcal{A}|$ is the size of $\mathcal{A}$. From Theorem 8, the complement problem of the bounded universality problem for a Büchi automaton and a non-negative integer k encoded in unary is in NP. The complement version of Step 3 is then accomplished within $O(p(|\mathcal{A}'| + k))$ time by a non-deterministic Turing machine. Therefore, we can solve the complement problem of the bounded strong satisfiability problem in $O(2^{p(|Spec|+(\lfloor \log k \rfloor+1))})$ time by a non-deterministic Turing machine, and we can conclude that the bounded strong satisfiability problem is in the class co-NEXPTIME. □

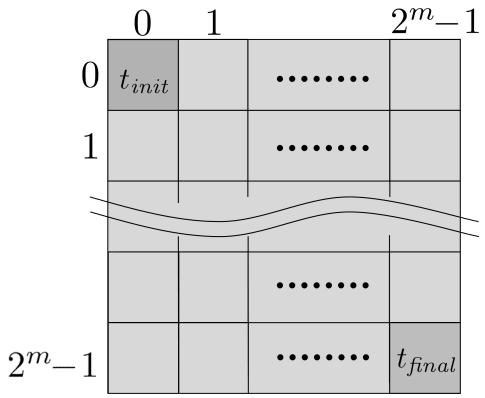

Fig. 3 The EXP-square tiling problem.

6.2 Lower Bound

In this subsection, we show that the bounded strong satisfiability problem is co-NEXPTIME-hard, by providing polynomial time reduction from the EXP-square tiling problem (see, e.g., [16], [17]) to the complement of the bounded strong satisfiability problem. The tiling problem is NEXPTIME-complete.

Definition 7 (EXP-square tiling problem): The EXP-square tiling problem is as follows: For a given $(T, H, V, t_{init}, t_{final}, m)$, where T is a finite set of tile types, $H, V \subseteq T \times T$ are horizontal and vertical adjacency constraints, $t_{init} \in T$ is the initial tile type, $t_{final} \in T$ is the final tile type, and m is a natural number encoded in unary, determine whether there exists an assignment function $f : [0,(2^m-1)] \times [0,(2^m-1)] \rightarrow T$ such that the following conditions are satisfied:

1. $f(0,0) = t_{init}$
2. $f((2^m-1),(2^m-1)) = t_{final}$
3. for any $0 \le j \le 2^m-1$, $0 \le i < 2^m-1$, $(f(i,j), f(i+1,j)) \in H$ holds.
4. for any $0 \le i \le 2^m-1$, $0 \le j < 2^m-1$, $(f(i,j), f(i,j+1)) \in V$ holds.

As shown in Fig. 3, the tiling grid has $2^m \times 2^m$ points. Intuitively, this leads to the following question: "For a given tiling grid, can a tile be assigned to each point (i,j) for which $0 \le i < 2^m$ and $0 \le j < 2^m$, satisfying conditions 1–4?"

Condition 1 is the condition for the initial tile, and states that the tile of type t_{init} is assigned to the leftmost, uppermost point. Condition 2 is the condition for the final tile, and states that a tile of type t_{final} is assigned to the rightmost, lowermost point. Condition 3 is the condition for horizontal lines, and states that horizontally neighboring tiles satisfy the horizontal adjacency constraint H. Condition 4 is the analogous condition for vertical lines.

We provide a polynomial time reduction from the EXP-square tiling problem to the complement of the bounded strong satisfiability problem. That is, for the EXP-square tiling problem $(T, H, V, t_{init}, t_{final}, m)$, we construct a formula φ_{tiling} and a non-negative integer k_{tiling} such that $\exists\tilde{a}(\tilde{a}$ is k_{tiling}-loop and $\forall\tilde{b}(\langle\tilde{a}, \tilde{b}\rangle \models \neg\varphi_{tiling}))$ holds if and only if the answer to the tiling problem $(T, H, V, t_{final}, t_{final}, m)$ is affirmative.

In this reduction, we relate "there exists a tiling assignment" in the tiling problem to "there exists a sequence of sets of input events that is k_{tiling}-loop." Furthermore, "the tiling assignment satisfies the conditions" is translated to "the corresponding sequence of sets of input events does not satisfy φ_{tiling} for any infinite sequence of sets of output events". That is, a tiling assignment is represented by a sequence $\tilde{a}$ of sets of input events, and the conditions in the tiling problem are represented by $\forall\tilde{b}(\langle\tilde{a}, \tilde{b}\rangle \models \neg\varphi_{tiling})$, universally quantified on infinite sequences of sets of output events.

In the representation of conditions 1–3, we do not require meta-level universal quantification; i.e., conditions 1–3 are represented by pure LTL formulae over input events only. To represent condition 4, however, we do require meta-level universal quantification. The representation of the conditions is based on that of the reduction from another tiling problem (which is EXPSPACE-complete) to the (unbounded) strong satisfiability problem in [6].

(a) Input events

To relate a sequence of sets of input events to a tiling assignment, we introduce the following input events.

- x_t for each $t \in T$: "the tile of type t is placed on the point (i, j)" is translated to "the input events x_t occur at time $i + (2^m) \cdot j$."
- *end*: "tiling is finished at the point (i, j)" is related to "*end* occurs at time $i + (2^m) \cdot j$."
- $c_0, \ldots, c_{2m-1}$: We let these events function as a $2m$ bit counter, which keeps track of the amount of time that has passed. The lower (higher) m bits represent a column (row) of the tiling grid.

(b) Output events

We introduce the following output events.

- $y_0, \ldots, y_{m-1}$: These are used to identify a column.

(c) The formula φ_{tiling}

The formula φ_{tiling} is the negation of the conjunction of the formulae (1)–(6) listed below. Here we use the following abbreviations:

$$\bar{c} = 2^{2m} - 1 \equiv \bigwedge_{0 \le i < 2m} c_i$$

$$\bar{c}_{low} = 2^m - 1 \equiv \bigwedge_{0 \le i < m} c_i$$

$$\bar{c}_{high} = 2^m - 1 \equiv \bigwedge_{m \le i < 2m} c_i$$

$$\bar{c}_{low} = \bar{y} \equiv \bigwedge_{0 \le i < m} (c_i \leftrightarrow y_i)$$

- The constraint for $2m$ bit counters $c_0, \ldots, c_{2m-1}$.

$$\left(\bigwedge_{0 \le i < 2m} \neg c_i\right) \wedge \mathbf{G}\left(\neg end \rightarrow \bigwedge_{0 \le i < 2m} \left(\left(c_i \oplus \bigwedge_{0 \le j < i} c_j\right) \leftrightarrow \mathbf{X} c_i\right)\right) \quad (1)$$

  This represents the statement "the value of $\bar{c}$ is initially 0, and is incremented at each time step if tiling is not finished", which means that the value of $\bar{c}_{low}$ ($\bar{c}_{high}$) represents the current column (row) number.
- Single assignment of tile types.

$$\bigwedge_{t \in T} \mathbf{G}\left(x_t \rightarrow \bigwedge_{t' \neq t} \neg x_{t'}\right) \wedge \mathbf{G}\left(\neg end \rightarrow \bigvee_{t \in T} x_t\right) \quad (2)$$

  This represents the statement "at most one tile type is assigned to each grid point, and if tiling is not finished, some tile type must be assigned."
- The constraint for condition 1.

$$x_{t_{init}} \quad (3)$$

  This represents the statement "the initial tile of type t_{init} is placed on the point $(0, 0)$."
- The constraint for condition 2.

$$\neg end \mathbf{U}(\neg end \wedge \bar{c} = 2^{2m} - 1 \wedge x_{t_{final}} \wedge \mathbf{XG} end) \quad (4)$$

  This represents the statement "the final tile of type t_{final} is placed on the point $(2^m - 1, 2^m - 1)$ (which corresponds to the $2^{2m} - 1$-th time point), and tiling is finished".
- The constraint for condition 3.

$$\mathbf{G}\left(\bar{c}_{low} \neq 2^m - 1 \wedge \neg end \rightarrow \bigvee_{(t,t') \in H} (x_t \wedge \mathbf{X} x_{t'})\right) \quad (5)$$

  This represents the statement "if tiling is not finished and the current point is not in the $(2^m - 1)$-th column (i.e., a point exists to the right of it), then the tile at the current point and the tile at the point to the right of it (which corresponds to the next time point) satisfy condition H".

- The constraint for condition 4.

$$\left(\bigwedge_{0\le i<m} \mathbf{G}(y_i \leftrightarrow \mathbf{X}y_i)\right) \rightarrow$$
$$\mathbf{G}\left((\bar{c}_{low} = \bar{y} \wedge \bar{c}_{high} \neq 2^m - 1 \wedge \neg end) \rightarrow \right.$$
$$\left.\bigvee_{(t,t')\in V} (x_t \wedge \mathbf{X}((\bar{c}_{low} \neq \bar{y})\mathbf{U}(\bar{c}_{low} = \bar{y} \wedge x_{t'})))\right) \quad (6)$$

  This represents the statement "if the value of $\bar{y}$ behaves as a constant, for any current point in the column indicated by $\bar{y}$, if the current point is not in the 2^m-1-th row and tiling is not finished, the tile at the current point and the tile at the point beneath it (which corresponds to the time point after 2^m time units) satisfy condition V". The point beneath the current one is characterized by the phrase "the first time point after the current point at which the value of $\bar{c}_{low}$ (the column number) again becomes $\bar{y}$". Because $y_0, y_1, \ldots, y_{m-1}$ are output events, $\forall\tilde{b}(\langle\tilde{a}, \tilde{b}\rangle \models (6))$ represents the statement "for any column, tiles in the column satisfy condition V", which means "any tiles satisfy condition V".

(d) k_{tiling}

We define $k_{tiling} = 2^{2m}$ (encoded as $2m + 1$ bits). From the formulae (1) and (4), only input sequences that are $(2^{2m}, 2^{2m})$-loops are allowed. The elements from the 0-th to the $2^{2m} - 1$-th correspond to the points of the tiling grid, and only the 2^{2m}-th element contains the event *end*.

Theorem 10: The bounded strong satisfiability problem for a specification written in LTL and a non-negative integer k encoded in binary is co-NEXPTIME-hard.

Proof. As described above, we can construct a formula φ_{tiling} and an integer k_{tiling} such that the answer to the EXP-square tiling problem is affirmative if and only if the corresponding φ_{tiling} is not k_{tiling}-strongly satisfiable. The size of k_{tiling} encoded in binary is linear and the size of φ_{tiling} is polynomial in the size of the problem $(T, H, V, t_{init}, t_{final}, m)$. Then, k_{tiling} and φ_{tiling} can be constructed in polynomial time. Therefore, the EXP-square tiling problem is reducible to the complement of the bounded strong satisfiability problem. Because the EXP-square tiling problem is NEXPTIME-complete, the bounded strong satisfiability problem is co-NEXPTIME-hard. □

6.3 Discussion

We discuss the complexity of the bounded strong satisfiability problem in relation to that of the satisfiability problem, the (unbounded) strong satisfiability problem, and the realizability problem. The complexity of the satisfiability problem for specifications written in LTL is PSPACE-complete [18], the complexity of the (unbounded) strong satisfiability problem for such specifications is EXPSPACE-complete [8], and the complexity of the realizability problem for such specifications is 2EXPTIME-complete [8]. PSPACE (EXPSPACE) is the complexity class of problems solvable in $O(p(n))$ space ($O(2^{p(n)})$ space) by a deterministic Turing machine, and 2EXPTIME is the complexity class of problems solvable in $O(2^{2^{p(n)}})$ time by a deterministic Turing machine. The relationship between these classes is as follows:

$$\text{PSPACE} \subseteq \text{co-NEXPTIME}$$
$$\subseteq \text{EXPSPACE} \subseteq \text{2EXPTIME}$$

Therefore, the bounded strong satisfiability problem is more difficult than the satisfiability problem, and is easier than, or of equal difficulty to, the strong satisfiability problem and the realizability problem.

7. Experiments

We implemented our method and compared the execution time with that of (unbounded) strong satisfiability[†].

Our implementation (denoted by BSS) is as follows. Steps 1 and 2 in the procedure described in Sect. 4 are based on [19]. The k-universality checking of Step 3 is accomplished incrementally for $d = 0, 1 \ldots$, as described in Sect. 5.3.1. We use MiniSat 2.2 [20] as the SAT solver. To check for (unbounded) strong satisfiability, denoted by SS, we check for (unbounded) universality. Universality is checked using the antichain-based technique reported in [21].

Table 2 lists the total checking times and the checking times of bounded universality or universality for the n-floors elevator specification Ele_n in [22], and Ele_n^a includes the fairness assumption[††].

In all tests of Ele_n^a, the result was "yes". This indicates that our method can handle larger specifications, and show that for any simple input pattern (represented by a k-loop) there is a response that satisfies the specification in a reasonable period of time.

In all tests of Ele_n, the result was "no". This indicates that our method can handle larger specifications more efficiently, and can obtain the simple input pattern of a deficiency in Ele_n. Because bounded strong satisfiability is a necessary condition for strong satisfiability, our method also shows that Ele_n is not strongly satisfiable for $n > 5$, which the strong satisfiability checker failed to indicate. Moreover, bounded strong satisfiability is a necessary condition for realizability, and a counterexample of bounded strong satisfiability (i.e., an input pattern that cannot satisfy the specification) can be considered a counterexample of realizability. Hence, our method also successfully shows that Ele_n ($n \le 7$)

[†]All experiments were performed on a machine with an Intel(R) Core(TM) i7-3820 3.60 GHz processor and 32 GB of RAM.

[††]The specifications Ele_n and Ele_n^a have $3n + 6$ atomic propositions ($|X| = n + 2$, $|Y| = 2n + 4$). The number of temporal operators in Ele_n and Ele_n^a are $6n - 1$ and $7n$, respectively.

Table 2 The checking times (in seconds) for Ele_n and Ele_n^a. The total checking times are given in the column denoted "total", and the checking times for bounded universality or universality are given in the column denoted by "(b).u". The notation "T/O" corresponds to a calculation that required more than 1000 seconds, and "T/O/A" corresponds to time out while constructing NBA.

				2		3		4		5		6		7		8	
		k	judgment	(b).u	total	(b).u	total	(b).u	total	(b).u	total	(b).u	total	(b).u	total	(b).u	total
Ele_n^a	BSS	0	"Yes"	0.00	0.02	0.01	0.03	0.11	0.15	1.37	1.66	23.01	27.98	453.20	533.35	T/O/A	
	BSS	2	"Yes"	0.02	0.04	0.24	0.26	5.20	5.27	125.23	125.54		T/O				
	BSS	4	"Yes"	0.16	0.18	4.26	4.27	245.24	245.30		T/O						
	SS	-	"Yes"	0.90	0.91	201.65	201.67		T/O								
Ele_n	BSS	0	"No"	0.00	0.01	0.01	0.02	0.06	0.09	0.52	0.83	7.62	13.08	251.20	341.19	T/O/A	
	SS	-	"No"	0.00	0.01	0.10	0.11	2.02	2.06	50.99	51.29		T/O				

is not unrealizable. The realizability checkers Acacia+ [23] and Unbeast [24] cannot handle even Ele_4. (The checking times of Acacia+ and Unbeast for Ele_3 were 540.97 seconds and over 1000 seconds, respectively. For Ele_4, Acacia+ also was not able to check realizability within 1000 seconds.) Our method can therefore detect deficiencies in practical specifications that existing realizability checkers cannot deal with.

These successful results result from the restriction of input patterns to k-loops, the usage of an efficient SAT solver and, importantly, our encoding method. In our encoding method, we bound the number of occurrences of final nodes by d. Then, k-universality checking can be accomplished incrementally for $d = 0, 1, \ldots$ (as described in Sect. 5.3.1). In the case of Ele_n, a counterexample is found when $d = 0$.

8. Discussion

Encoding methods for bounded model checking. For bounded model checking, SAT encoding methods of problems that satisfy specifications represented by LTL, very weak alternating Büchi automata (VWABA) and weak alternating Büchi automata (WABA) can be found in [4], [25], [26]. LTL and VWABA are less expressive than NBA, which was used in this work. WABA and NBA are of equal expressiveness. Indeed, bounded universality checking for NBA can also be accomplished via WABA, using the WABA encoding method of [26]. However, our method is more efficient than the WABA approach. The size of the propositional formulae of the WABA approach is $O(k^2+k\cdot|Q|^2\cdot|\delta|)$, whereas we have provided $O(k^2+k\cdot|Q|\cdot|\delta|)$-encoding.

Bounded realizability. In the methods described in [27], [28], to simplify the procedures for realizability checking, the (non-)acceptance condition of automata is bounded, as with our method. That is, the condition that the number of occurrences of final states is "at most d", instead of "finite" is used as a (non-)acceptance condition of the automata. An incremental approach is also taken. However, the size of witnesses or counterexamples is not bounded in the approach described in [27], [28].

In [13], the notion of bounded realizability and a checking method using a SMT solver were reported. Bounded realizability is the property that there exists a reactive system that acts as a transition system of k states, such that all behaviors satisfy the specification. In bounded realizability checking, transition systems of k states are searched. In contrast, with our method, k-loops (which are simpler) are searched. Because of this, our method can detect typical simple deficiencies in larger specifications.

9. Conclusion

We introduced the notion of bounded strong satisfiability and a checking method for this property, to detect simple deficiencies in reactive system specifications. In our method, we construct an NBA that accepts input sequences for which there is a response that satisfies a specification, then check whether the NBA is k-universally acceptable using a SAT solver. We show that the bounded strong satisfiability problem for a specification in LTL is co-NEXPTIME-complete. This implies that the bounded strong satisfiability problem is easier than or of equal difficulty compared with the realizability problem and the strong satisfiability problem. We implemented our method and demonstrated that it can handle larger specifications than exisiting checking techniques for other properties, and that it can also detect simple deficiencies efficiently. We believe that our methods are of practical use in the requirement analysis phase in developing safety critical systems.

References

[1] D. Jackson, Software Abstractions: Logic, Language, and Analysis, The MIT Press, 2006.
[2] M. Abadi, L. Lamport, and P. Wolper, "Realizable and unrealizable specifications of reactive systems," Proc. 16th International Colloquium on Automata, Languages, and Programming, LNCS, vol.372, pp.1–17, Springer, 1989.
[3] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," Proc. 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp.179–190, ACM, 1989.
[4] A. Biere, A. Cimatti, E.M. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," Proc. 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems, LNCS, vol.1579, pp.193–207, Springer, 1999.
[5] R. Mori and N. Yonezaki, "Several realizability concepts in reactive objects," Proc. Information Modeling and Knowledge Bases IV: Concepts, Methods and Systems, pp.407–424, IOS Press, 1993.
[6] M. Shimakawa, S. Hagihara, and N. Yonezaki, "Complexity of strong satisfiability problems for reactive system specifications," IEICE Trans. Inf. & Syst., vol.E96-D, no.10, pp.2187–2193, Oct. 2013.

[7] R. Mori and N. Yonezaki, "Derivation of the input conditional formula from a reactive system specification in temporal logic," Proc. Formal Techniques in Real-Time and Fault-Tolerant Systems, LNCS, vol.863, pp.567–582, Springer, 1994.
[8] R. Rosner, Modular Synthesis of Reactive Systmes, Ph.D. thesis, Weizmann Institute of Science, 1992.
[9] S. Hagihara and N. Yonezaki, "Completeness of verification methods for approaching to realizable reactive specifications," Proc. 1st Asian Working Conference on Verified Software, AWCVS '06, UNU-IIST, vol.348, pp.242–257, 2006.
[10] W. Thomas, "Automata on infinite objects," in Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B), pp.133–192, Elsevier and MIT Press, 1990.
[11] H. Tauriainen, "On translating linear temporal logic into alternating and nondeterministic automata," Research Report A83, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, 2003.
[12] F. Klaedtke, "Complementation of Büchi automata using alternation," Automata, Logics, and Infinite Games, LNCS, vol.2500, pp.61–78, Springer, 2001.
[13] S. Schewe and B. Finkbeiner, "Bounded synthesis," Proc. 5th International Symposium on Automated Technology for Verification and Analysis, LNCS, vol.4762, pp.474–488, Springer, 2007.
[14] M. Sheeran, S. Singh, and G. Stålmarck, "Checking safety properties using induction and a SAT-solver," Proc. Third International Conference on Formal Methods in Computer-Aided Design, LNCS, vol.1954, pp.108–125, Springer, 2000.
[15] R. Armoni, L. Fix, R. Fraer, S. Huddleston, N. Piterman, and M.Y. Vardi, "SAT-based induction for temporal safety properties," Electr. Notes Theor. Comput. Sci., vol.119, no.2, pp.3–16, 2005.
[16] B.S. Chlebus, "From domino tilings to a new model of computation," Symposium on Computation Theory, pp.24–33, 1984.
[17] P.V.E. Boas, "The convenience of tilings," Complexity, Logic, and Recursion Theory, Lecture Notes in Pure and Applied Mathematics, vol.187, pp.331–363, Marcel Dekker Inc, 1997.
[18] A.P. Sistla and E.M. Clarke, "The complexity of propositional linear temporal logics," J. ACM, vol.32, no.3, pp.733–749, 1985.
[19] T. Aoshima, K. Sakuma, and N. Yonezaki, "An efficient verification procedure supporting evolution of reactive system specifications," Proc. International Workshop on Principles of Software Evolution, IWPSE 2001, pp.182–185, 2001.
[20] N. Eén and N. Sörensson, "An extensible SAT-solver," Proc. 6th International Conference on Theory and Applications of Satisfiability Testing, LNCS, vol.2919, pp.502–518, Springer, 2003.
[21] M.D. Wulf, L. Doyen, N. Maquet, and J.F. Raskin, "Antichains: Alternative algorithms for LTL satisfiability and model-checking," Proc. 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, LNCS, vol.4963, pp.63–77, Springer, 2008.
[22] T. Aoshima and N. Yonezaki, "Verification of reactive system specification with outer event conditional formula," Proc. International Symposium on Principles of Software Evolution, ISPSE 2000, pp.195–199, 2000.
[23] A. Bohy, V. Bruyère, E. Filiot, N. Jin, and J.F. Raskin, "Acacia+, a tool for LTL synthesis," Proc. 24th International Conference on Computer Aided Verification, LNCS, vol.7358, pp.652–657, Springer, 2012.
[24] R. Ehlers, "Unbeast: Symbolic bounded synthesis," Proc. 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, LNCS, vol.6605, pp.272–275, Springer, 2011.
[25] D. Sheridan, "Bounded model checking with SNF, alternating automata, and Büchi automata," Electron. Notes Theor. Comput. Sci., vol.119, no.2, pp.83–101, 2005.
[26] K. Heljanko, T.A. Junttila, M. Keinänen, M. Lange, and T. Latvala, "Bounded model checking for weak alternating Büchi automata," Proc. 18th International Conference on Computer Aided Verification, LNCS, vol.4144, pp.95–108, Springer, 2006.
[27] E. Filiot, N. Jin, and J.F. Raskin, "An antichain algorithm for LTL realizability," Proc. 21st International Conference on Computer Aided Verification, LNCS, vol.5643, pp.263–277, Springer, 2009.
[28] R. Ehlers, "Symbolic bounded synthesis," Proc. 22nd International Conference on Computer Aided Verification, LNCS, vol.6174, pp.365–379, Springer, 2010.

Masaya Shimakawa received the B.E. and M.E. degrees in computer science from Tokyo Institute of Technology, Tokyo, Japan, in 2004 and 2006, respectively. His research interests include temporal logic, automata theory and formal verification.

Shigeki Hagihara received the B.E., M.E. and D.E. degrees in computer science from Tokyo Institute of Technology in 1993, 1995 and 2000, respectively. From 2000 to 2001, he was a research associate in Research for the Future Program, Japan Society for the Promotion of Science. Since 2001, he has been an assistant professor in Department of Computer Science, Tokyo Institute of Technology. His research interests include proof methods of non-classical logics such as temporal logics, epistemic logics, etc., and software verification, especially, specification analysis and model-checking for reactive systems, and security protocol analysis. He is a member of Japan Society for Software Science and Technology.

Naoki Yonezaki received B.E., M.E. and D.E. degrees from Tokyo Institute of Technology in 1972, 1974 and 1977, respectively. He has been a Professor of Tokyo Institute of Technology since 1991. He was also a Professor of Japan Advanced Institute of Science and Technology from 1991 to 1995. His research interests are formal approach to construction and analysis of complex systems, including algorithms, verification of software, verification of security and formal analysis of biological systems. Currently he is a dean of the School of Information Science and Engineering, and the Director General of Inter-departmental Organization for Informatics in Tokyo Institute of Technology. He is also the head of the Information Technology Specialist educational program (IT Bauhaus), and was the head of the joint education program for Translational Biomedical Informatics with Tokyo Medical and Dental University. He has been the editor in chief of Computer Software, the main Journal of Japan Society of Software Science and Technology (JSSST) from 2000 to 2004. He also served as a program committee chair of IWTS99, IWPSE01, etc. He was awarded a fellowship by JSSST in 2008.