SAT–Based Bounded Strong Satisfiability Checking of Reactive System Specifications

Masaya Shimakawa, Shigeki Hagihara, Naoki Yonezaki

SAT–Based Bounded Strong Satisfiability Checking of Reactive System Specifications

Masaya Shimakawa, Shigeki Hagihara, and Naoki Yonezaki

Department of Computer Science,
Graduate School of Information Science and Engineering,
Tokyo Institute of Technology.
2-12-1-W8-67 Ookayama, Meguro-ku, Tokyo 152-8552, Japan

Abstract. Many fatal accidents involving safety-critical reactive systems have occurred in unexpected situations that were not considered during the design and test phases of the systems. To prevent these 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. For the purpose of detecting simple and typical defects in specifications, 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 present a checking method based on a satisfiability solver, and report experimental results.

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 and air traffic control systems, are also considered reactive systems. In designing a system of this kind, the requirements are analyzed and then described as specifications for the system. If a specification has a flaw, such as inappropriate case-splitting, a developed system may encounter unintended situations. Indeed, many fatal accidents involving safety-critical reactive systems have occurred in unexpected situations, which were not considered during the design and test phases of the systems. Therefore, it is important to ensure that a specification does not possess this kind of flaw[8].

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 is called realizability, and was introduced in [1, 12]. In [12], it was demonstrated that a reactive system can be synthesized from a realizable specification. However, realizability checking for reactive system specifications involves complex and

intricate analysis. Therefore, the sizes of specifications that can be checked in a practical application are strongly limited.

For the purpose of detecting simple and typical defects in specifications, we present a bounded checking method. In bounded checking, we verify the existence of a counterexample (or witness) of a given size k. Such methods are used successfully in other fields, for example, bounded model checking[3] and Alloy Analyzer[8]. The advantage of bounded checking is its ability to detect a small counterexample (or witness) efficiently.

We present a bounded property for strong satisfiability [10] (a necessary condition for realizability) known as bounded strong satisfiability, together with a method for checking this property. Strong satisfiability is the property that for any input sequences, there is a response that satisfies a given specification. Strong satisfiability can be checked with lower complexity than realizability[16]. Checking strong satisfiability is EXPSPACE-complete. On the other hand, checking realizability is 2EXPTIME-complete. Although this property is a necessary condition, many practical unrealizable specifications are also strongly unsatisfiable[11]. Bounded strong satisfiability restricts the input sequences of strong satisfiability to those represented by loop structures of size k. This means 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 loop structures of small size). Thus, we anticipate that many defects can be detected by checking this property.

In our method for checking bounded strong satisfiability, we use an 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. Then we check whether or not the NBA accepts all loop structures of size k, using an SAT solver. To accomplish this, checking the existence of a loop structure that is not accepted by NBA is reduced to an 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 at most d times for some d. This characterization is valid because only bounded loop structures are considered.

We implemented our method, and found that it can handle larger specifications than techniques based on other properties, and can detect defects efficiently. We also report experimental results in this paper.

2 Preliminaries

2.1 Reactive Systems

A reactive system is a system 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 $r : (2^X)^+ \mapsto 2^Y$ 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. A linear temporal logic (LTL) is a suitable language for describing the timing of events. In this paper, we use LTL to describe the specifications of reactive systems. We treat input events and output events as atomic propositions.
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 $f \vee g$, $f \rightarrow g$, $\top$, $f\mathbf{R}g$, $\mathbf{F}f$, and $\mathbf{G}f$ are abbreviations for $\neg(\neg f \wedge \neg g)$, $\neg(f \wedge \neg g)$, $\neg\bot$, $\neg(\neg f\mathbf{U}\neg g)$, and $\top\mathbf{U}f$, $\neg\mathbf{F}\neg f$ respectively, where $\bot$ is an atomic proposition representing 'falsity.'
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 $p \in \sigma[i]$
- $\sigma \models \neg f$ iff $\sigma \not\models f$
- $\sigma \models f\mathbf{U}g$ iff $\exists j \geq 0.((\sigma[j\ldots] \models g)$ and $\forall k(0 \leq k < j.\ \sigma[k\ldots] \models f))$
- $\sigma \models f \wedge g$ iff $\sigma \models f$ and $\sigma \models g$
- $\sigma \models \mathbf{X}f$ iff $\sigma[1\ldots] \models f$

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 $\tilde{a} = a_0 a_1 \ldots$, $behave_{RS}(\tilde{a}) = (a_0 \cup b_0)(a_1 \cup b_1)\ldots$, 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 [10].

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

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

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 speaking, 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.

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 being 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., $\neg y$). This specification is then represented by $\mathit{Spec}_1 : \mathbf{G}((x_1 \rightarrow \mathbf{F}y)\wedge(x_2 \rightarrow \neg y))$ in LTL. This is not strongly satisfiable, and is 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 $\tilde{a} = \{x_1,x_2\}\{x_2\}\{x_2\}\ldots$, $\exists\tilde{b}(\langle\tilde{a},\tilde{b}\rangle \models \mathit{Spec}_1)$ does not hold. Hence $\forall\tilde{a}\exists\tilde{b}(\langle\tilde{a},\tilde{b}\rangle \models \mathit{Spec}_1)$ does not hold.

3 Bounded strong satisfiability

In this section, we present 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.

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 $u = s_0s_1\ldots s_{l-1}$ and $v = s_ls_{l+1}\ldots s_k$ 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:*

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

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 the 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[1].

Theorem 2. *For all $k \in \mathbb{N}$, if a specification Spec is strongly satisfiable, then Spec is also k-strongly satisfiable. Moreover, if a specification Spec is not strongly satisfiable, then 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 an 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 or not a specification satisfies bounded strong satisfiability, we can know whether or not the specification has simple input patterns (represented by small loops) that cannot satisfy the specification. As the experiment in Section 6 shows, many practical defective specifications have such simple input patterns. Thus, we anticipate that checking bounded strong satisfiability will find many practical defects in reactive system specifications.

4 Procedure for checking bounded strong satisfiability

In this section, we present a procedure for checking bounded strong satisfiability using non-deterministic Büchi automata. This procedure is based on the procedure for (unbounded) strong satisfiability introduced in [6].

A *non-deterministic Büchi automaton*(NBA) is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F\rangle$, where Σ is an alphabet, Q is a finite set of states, q_0 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 $\varrho[0] = q_0$ 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 $\mathcal{A} = \langle 2^{X\cup Y}, Q, q_0, \delta, F\rangle$ s.t. $L(\mathcal{A}) = \{\sigma | \sigma \models Spec\}$ holds.
2. Let $\mathcal{A}' = \langle 2^X, Q, q_0, \delta,' F\rangle$ be the NBA obtained by restricting $\mathcal{A}$ to only input events, where $\delta' = \{(q, a, q') | \exists b\ (q, a \cup b, q') \in \delta\}$. Note that $L(\mathcal{A}') = \{\tilde{a} | \exists\tilde{b}\langle\tilde{a}, \tilde{b}\rangle \in L(\mathcal{A})\}$ holds due to the definition of δ'.
3. We check whether or not $\mathcal{A}'$ 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 based on the tableau methods of [2] and others. In step 3, bounded universality is checked using an SAT solver.

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

5 SAT-based bounded universality checking for NBA

We present an 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 an 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.

Definition 6 (run graph). *Let $\mathcal{A} = (\Sigma, Q, q_I, \delta, F)$ be 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). $E := \{((q,i), (q', suc(i)))|(q, s_i, q') \in \delta\}$, where $suc(i) = l$ if $i = k$, and $suc(i) = i + 1$ otherwise (the set of edges). $C := \{(q,i)|q \in F, 0 \leq i \leq k\}$(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 $Inf(\varrho) \cap F = \emptyset$ holds; i.e., for all runs, the number of occurrences of final states in the run is finite. A run on $\sigma[i \dots]$ from a state q corresponds to a path in the run graph from (q,i). Then the following holds:

Theorem 3. *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, the following result is derived [2]:

Lemma 1. *Let $d \geq |C|$. If for all paths $\tilde{v}$ in a run graph G from a state 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 $\tilde{v}$.*

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 $v \in V \setminus C$ (w.r.t. $v \in C$), $AtMost(v,j)$ holds if and only if for all successors $v' \in vE$, $AtMost(v',j)$ holds(w.r.t. $AtMost(v', j-1)$ holds). Additionally, for all $v \in C$, $AtMost(v,0)$ does not hold. Based on this idea, the following result can be proved:

Theorem 4. *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 sets of nodes $V_0, V_1, \dots, V_d$ such that the following are true:*

[2] If the number of occurrences of final nodes in a path is more than $|C|$, then there exists a final state q_c that occurs at least twice, which implies the existence of a path on which the final state q_c occurs infinitely often.

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 $0 \leq j < d$, 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 $F(V_d)$).*

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

Theorem 5. *Let $\mathcal{A} = (\Sigma, Q, q_I, \delta, F)$ be an NBA and $k \in \mathbb{N}$. For all $d \in \mathbb{N}$, (2) implies (1), and for $d \geq (k+1) \cdot |F|$, (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 sets $V_0, V_1, \ldots, V_d$ of nodes of the run graph G for $\mathcal{A}$ and σ, $I(V_0) \wedge \bigwedge_{0 \leq j < d} (V_j, V_{j+1}) \wedge F(V_d)$ holds.

5.2 Reduction to SAT

We present a reduction to SAT based on Theorem 5. In other words, for an NBA $\mathcal{A}$ and k, we construct a propositional formula $|[notAcc(\mathcal{A}, k, d)]|$ such that condition (2) of Theorem 5 holds if and only if $|[notAcc(\mathcal{A}, k, d)]|$ is satisfiable.
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 $p \in P$, $0 \leq i \leq k$, which indicate whether or not the i-th element s_i of a k-loop satisfies $p \in s_i$; (b) l_i for $0 \leq i \leq k$, which indicate whether or not the i-th element follows the k-th element; (c) $v^j_{(q,i)}$ for $q \in Q, 0 \leq i \leq k, 0 \leq j \leq d$, which indicate whether or not $(q, i) \in V_j$ holds.
Constraint To represent the concept "be a k-loop correctly", we define the formula $|[loop(k)]| := \bigvee_{0 \leq i \leq k} l_i \wedge \bigwedge_{0 \leq i \leq k} (l_i \rightarrow \bigwedge_{0 \leq i' \leq k, i' \neq i} \neg l_{i'})$.

To represent $I_\sigma \wedge \bigwedge_{0 \leq j < d} T(V_j, V_{j+1}) \wedge F(V_d)$, we define the formulae $|[I(\mathcal{A}, k)]|_0$ $|[T(\mathcal{A}, k)]|_{j,j+1}$ and $|[F(\mathcal{A}, k)]|_d$, which respectively indicate that $I(V_0)$, $T(V_i, V_{i+1})$ and $F(V_d)$ hold. Let $\mathcal{A} = (2^P, Q, q_I, \delta, F)$ be an NBA and $k \in \mathbb{N}$. These formulae are defined in Table 1, where $|[a]|_i$ is the formula indicating that the i-th element of σ is a, and $|[suc]|^j_{(q,i)}$ is the formula indicating that $(q, suc(i)) \in V_j$ holds.

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

Theorem 6. *Let $\mathcal{A}$ be an NBA and $k \in \mathbb{N}$. For all $d \in \mathbb{N}$, (2) implies (1), and for $d \geq (k+1) \cdot |F|$, (1) implies (2), where (1) and (2) are as follows:*

$\lvert[I(\mathcal{A},k)]\rvert_0$	$\bigwedge_{\substack{q\in Q\setminus F,\\ 0\le i\le k}} \left(v^0_{(q,i)} \leftrightarrow \bigwedge_{(q,a,q')\in\delta} (\lvert[a]\rvert_i \rightarrow \lvert[suc]\rvert^0_{(q',i)}) \right) \wedge \bigwedge_{\substack{q\in F,\\ 0\le i\le k}} \left(\neg v^0_{(q,i)}\right)$
$\lvert[T(\mathcal{A},k)]\rvert_{j,j+1}$	$\bigwedge_{\substack{q\in Q\setminus F,\\ 0\le i\le k}} \left(v^{j+1}_{(q,i)} \leftrightarrow \bigwedge_{(q,a,q')\in\delta} (\lvert[a]\rvert_i \rightarrow \lvert[suc]\rvert^{j+1}_{(q',i)}) \right) \wedge \bigwedge_{\substack{q\in F,\\ 0\le i\le k}} \left(v^{j+1}_{(q,i)} \leftrightarrow \bigwedge_{(q,a,q')\in\delta} (\lvert[a]\rvert_i \rightarrow \lvert[suc]\rvert^{j}_{(q',i)}) \right)$
$\lvert[F(\mathcal{A},k)]\rvert_d$	$v^d_{(q_I,0)}$

Table 1. The definitions of $\lvert[I(\mathcal{A},k)]\rvert_0$, $\lvert[T(\mathcal{A},k)]\rvert_{j,j+1}$ and $\lvert[T(\mathcal{A},k)]\rvert_{j,j+1}$

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

Theorem 7. *Let $\mathcal{A} = (\Sigma, Q, q_I, \delta, F)$ and $k, d \in \mathbb{N}$. The size of $\lvert[notAcc(\mathcal{A},k,d)]\rvert$ is $\mathcal{O}(k^2 + k\cdot d\cdot|\delta|)$. Checking that $\mathcal{A}$ is not k-universally acceptable can be reduced to the SAT problem for a formula of size $\mathcal{O}(k^2\cdot|F|\cdot|\delta|)$.*

5.3 Improvement

Incremental checking. The condition of Theorem 5 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. Consequently, the incremental technique of [14] can be applied to our method. Using this technique, k-universality checking can be accomplished for small d.

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 $\mathcal{O}(k\cdot|Q|\cdot|\delta|)$ 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 or not final nodes occurred before the given node was reached in each turn. We also define $C' := \{(q,k,\top)|q\in Q\}$. The modified run graph has the same features as the normal run graph. The size $|V'|$ is also $\mathcal{O}(k\cdot|Q|)$, but the size $|C'|$ is $|Q|$, while the size $|C|$ is $(k+1)\cdot|F|$. Thus $\mathcal{O}(k\cdot|Q|\cdot|\delta|)$-encoding is possible.

6 Experiments

We implemented our method and compared its execution time to that of realizability and (unbounded) strong satisfiability[3].

Our implementation (denoted by BSS) is as follows. Steps 1 and 2 in the procedure of Section 4 are based on [2]. The k-universality checking of Step 3 is accomplished incrementally for $d = 0, 1, \ldots$, based on the modified run

[3] All experiments were performed on a Pentium(R) D 3.0 GHz with 2.0 GB RAM.

$Spec_n$	BSS $k=1$	SS	R Lily
2	0.05	219.32	955.23
3	0.13	>3600	>3600
4	1.17		
5	12.95		
6	181.62		
7	1962.62		

$Spec'_n$	BSS $k=1$	$k=5$	$k=10$	SS	R lily
2	0.06	0.12	0.21	>3600	>3600
3	0.17	1.12	2.00		
4	1.60	44.08	82.95		
5	24.76	>3600	>3600		
6	297.59				
7	2942.89				

Table 2. The checking times (sec) for each property

graph (described in Section 5.3). We use MiniSat[5] as the SAT solver. To check (unbounded) strong satisfiability (denoted by SS), we check (unbounded) universality. Universality is checked with the symbolic model checker NuSMV[4]. To check realizability (denoted by R), we use Lily[9].

Table 2 lists the checking times for the specification $Spec_n$ in [2] (a specification for an elevator control system involving n floors) and $Spec'_n$, which includes a fairness assumption. The specification $Spec_n$ and $Spec'_n$ have $3n+6$ atomic propostions ($|X| = n+2$, $|Y| = 2n+4$). The numbers of the occurrence of temporal operators in $Spec_n$ and $Spec'_n$ are $6n-1$ and $7n$, respectively. In all tests of $Spec_n$, the judgment was "NO." The results indicate that our method can handle larger specifications more efficiently, and can obtain the simple input pattern of a defect in $Spec_n$. Because bounded strong satisfiability is a necessary condition for realizability, our method also successfully showed that $Spec_n$ is not unrealizable for $n > 2$, which the realizability checker failed to indicate. In all tests of $Spec'_n$, the judgment was "Yes.' Hence, our method showed that for any simple input pattern, there is a response that satisfies $Spec'_n$ in real time.

7 Related work

Encoding methods for bounded model checking. For bounded model checking, SAT encoding methods of problems whether or not models satisfy specifications represented by LTL, very weak alternating Büchi automata (VWABA) and weak alternating Büchi automata (WABA) are presented in [3, 15, 7]. 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 [7]. However, our method is more efficient than the WABA approach[4] .

Bounded realizability. In [13], the notion of bounded realizability and a checking method using an SMT solver are presented. 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. On the other hand, in our method, k-loops (which are simpler) are searched. Because of this, our method can detect simple and typical defects in larger specifications.

[4] The total size of the propositional formulae of the WABA approach is $\mathcal{O}(k \cdot |Q|^2 \cdot |\delta|)$, whereas we have provided $\mathcal{O}(k \cdot |Q| \cdot |\delta|)$-encoding.

8 Conclusion

We introduced the notion of bounded strong satisfiability and a checking method for this property, for detecting simple defects in reactive system specifications. In our method, we construct an NBA that accepts input sequences for which there is no response that satisfies a specification, then check whether or not the NBA is k-universally acceptable, using an SAT solver. We implemented our method and demonstrated that it can handle larger specifications than the checking techniques for other properties, and can also detect simple defects efficiently.

References

1. Abadi, M., Lamport, L., Wolper, P.: Realizable and unrealizable specifications of reactive systems. In: Proc. ICALP. pp. 1–17 (1989)
2. Aoshima, T., Sakuma, K., Yonezaki, N.: An efficient verification procedure supporting evolution of reactive system specifications. In: Proc. International Workshop on Principles of Software Evolution. pp. 182–185 (2001)
3. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without BDDs. In: Proc. TACAS. pp. 193–207 (1999)
4. Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: Nusmv 2: An opensource tool for symbolic model checking. In: Proc. CAV. pp. 359–364 (2002)
5. Eén, N., Sörensson, N.: An extensible sat-solver. In: Proc. SAT. pp. 502–518 (2003)
6. Hagihara, S., Yonezaki, N.: Completeness of verification methods for approaching to realizable reactive specifications. In: Proc. Asian Working Conference on Verified Software. pp. 242–257 (2006)
7. Heljanko, K., Junttila, T.A., Keinänen, M., Lange, M., Latvala, T.: Bounded model checking for weak alternating Büchi automata. In: Proc. CAV. pp. 95–108 (2006)
8. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. The MIT Press (2006)
9. Jobstmann, B., Bloem, R.: Optimizations for LTL synthesis. In: Proc. FMCAD. pp. 117–124 (2006)
10. Mori, R., Yonezaki, N.: Several realizability concepts in reactive objects. In: Proc. Information Modeling and Knowledge Bases IV: Concepts, Methods and Systems. pp. 407–424. IOS Press (1993)
11. Mori, R., Yonezaki, N.: Derivation of the input conditional formula from a reactive system specification in temporal logic. In: Proc. Formal Techniques in Real-Time and Fault-Tolerant Systems. LNCS, vol. 863, pp. 567–582 (1994)
12. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proc. POPL. pp. 179–190 (1989)
13. Schewe, S., Finkbeiner, B.: Bounded synthesis. In: Proc. ATVA. pp. 474–488 (2007)
14. Sheeran, M., Singh, S., Stålmarck, G.: Checking safety properties using induction and a SAT-solver. In: Proc. FMCAD. pp. 108–125 (2000)
15. Sheridan, D.: Bounded model checking with SNF, alternating automata, and Büchi automata. Electron. Notes Theor. Comput. Sci. 119(2), 83–101 (2005)
16. Shimakawa, M., Hagihara, S., Yonezaki, N.: Complexity of checking strong satisfiability of reactive system specifications. In: Proc. International Conference on Advances in Information Technology and Communication. pp. 42–51 (2012)