# A Method to Generate Formulae for Temporal Logic Satisfiability Checkers


Toshifusa Sekizawa[1,2], Toshinori Takai[1],
Yoshinori Tanabe[1,3], and Koichi Takahashi[1]

[1] National Institute of
Advanced Industrial Science and Technology (AIST),
Research Center for Verification and Semantics (CVS).
[2] Graduate School of Information Science and Technology,
Osaka University.
[3] Graduate School of Information Science and Technology,
The University of Tokyo.

# A Method to Generate Formulae for Temporal Logic Satisfiability Checkers

Toshifusa Sekizawa [1,2], Toshinori Takai [1], Yoshinori Tanabe [1,3], and Koichi Takahashi [1]

[1] National Institute of Advanced Industrial Science and Technology (AIST), Research Center for Verification and Semantics (CVS)
[2] Graduate School of Information Science and Technology, Osaka University
[3] Graduate School of Information Science and Technology, The University of Tokyo

**Abstract**

In order to evaluate performances of temporal logic satisfiability checkers, benchmark with test formulae is required as well as analyzing complexity of algorithms. However, there seems no clear criteria of the formulae for benchmark. Thus, to evaluate performances of satisfiability checkers for some new logics, we have to prepare test formulae of the logics. In this paper, we propose a systematic method to generate formulae of two-way CTL, which aim to be benchmark formulae of satisfiability checkers. We also discuss the criteria of test formulae. Finally, we list formulae which are obtained by our method and mention the results of experiments with our two-way CTL satisfiability checker.

## 1 Introduction

Automatic verification methods are required in the field of system verification. Specifications of reactive systems such as embedded systems can be regarded as temporal behaviours of transition systems which are the models of the reactive systems. Temporal logics are used to describe such behaviours. Model checking, a major automatic verification technique of reactive systems, is a technique to decide effectively whether a given finite model satisfies a given temporal formula or not. In general, it is difficult to verify practical systems using straightforward model checking technique, because of the state explosion problem. For such complicated systems, one solution is abstraction, which reduces the size of models. The authors proposed an abstraction method using a temporal logic [10]. In the proposed abstraction method, not only model checking is required, but also satisfiability checking of temporal logics is heavily used. Furthermore, there are many other applications of satisfiability checking of temporal logics, for example synchronizing concurrent programs [5, 9], analysis of cellular automata [6], XML data transformation [14], etc. To implement such applications, fast satisfiability checking algorithms [12, 13, 14] are developed .

For evaluating implementations of satisfiability checking algorithms, two conventional methods have mainly used: (a) analyzing computational complexity, and (b) measuring running time for some formulae. In the case (a), it is sometimes difficult to obtain accurate complexities. Even if complexities are obtained, it is often not enough to compare algorithms in general. In the case (b), since criteria of selected formulae are rarely shown, evaluation is also difficult. Benchmarks using random generated formulae are not suitable for evaluation. Many researches have proposed benchmark methods for propositional logic satisfiability checkers, but there are a few benchmarks for temporal logics.

Balsiger et al. proposed a benchmark method for theorem provers of propositional modal logics [1] which uses *patterns*. A pattern is a formula with natural number parameters, which can generate an arbitrary long formula. For evaluation experiments, they used formulae generated from the patterns. In their method, by analyzing characteristics of patterns in advance, we can clarify characteristics of theorem provers. For example, a number of atomic propositions, depth of nested modal operators are characteristics of patterns. Balsiger et al. listed some patterns in the paper [1], but they did not describe how to give such patterns. Therefore, for other logical systems, it is not clear how to generate formulae to evaluate satisfiability checkers. In this paper, we propose systematic methods to give patterns which can be used for satisfiability checkers and theorem provers.

We first discuss conditions required for a set of temporal logic formulae. Balsiger et al. described conditions which should be satisfied by evaluation methods for theorem provers. Then, as a solution, they proposed formulae with natural number parameters [1]. We next consider desirable properties of such formulae with natural number parameters. After that, we propose some systematic methods to generate formulae of *two-way CTL*. Two-way CTL is used in the abstraction method of graph rewriting systems [10, 11], proposed by the authors.

When we try to deal with timed or spatial properties in modal logics, sometimes we need backward modalities. Two-way CTL is an extension of one-way CTL, i.e., backward modalities are allowed. Properties with respect to backward transitions can be described easily in two-way CTL. Complexities of satisfiability checking of two-way CTL and one-way CTL are both EXPTIME. On the other hand, two-way CTL does not have the finite model property in contrast to one-way CTL.

The outline of this paper is as follows. First, we show a theorem that provides the basis for our formulae generations, and then, we propose methods to generate formulae based on the theorem. Since the proposed methods need simple valid formulae as seeds, we next show how to obtain such formulae. Lastly, we list a set of formulae generated by our method and show some experiment results using satisfiability checker [12] developed by the authors.

## 2 Backgrounds

Balsiger et al. proposed seven postulates [1] for benchmark tests for theorem provers as cited in Figure 1 that benchmark formulae should concern. They proposed patterns of formulae with a natural number parameter as a solution to satisfy these postulates. In their methods, they prepared arbitrarily long formulae which were generated by applying rules to simple formulae. In this paper, we call such a generated formula a *natural number parameterized formula* or a *parameterized formula*. Evaluation of theorem provers consists of two

1. Provable as well as unprovable formulas.
2. Formulas of various structures.
3. Some of the benchmark formulas are hard enough for forth-coming provers.
4. For each formula the result is already known today.
5. Simple 'tricks' do not help to solve the problems.
6. Applying the benchmark test to a prover takes not too much time.
7. The results can be summarized.

Figure 1: Postulates concerning benchmark tests for automated theorem provers. (cited from [1])

phases, (1) to prepare some parameterized formulae, and (2) to measure the maximum value of the parameters such that formula corresponds to the value can be judged within a fixed time, for example 100 seconds. The correspondence between parameterized formulae and the postulates in Figure 1 is as follows:

1. Prepare both valid and non-valid parameterized formulae.
2. Prepare parameterized formulae which contain various characteristics.
3. Arbitrarily long formulae can be obtained by increasing a natural number parameter.
4. Prepare parameterized formulae which preserve validity as well as invalidity.
5. Parameterized formulae are chosen so that simple tricks do not help to solve, although the effect is unclear.
6. Measurement completes in a fixed time since the criterion of measurement is the value of parameter.
7. The results can be summarized with a table of natural number parameters.

These postulates were proposed for theorem provers. They are also applicable to satisfiability checkers by replacing 'provable' with 'unsatisfiable', and 'unprovable' with 'satisfiable' because the negation of a provable formula is unsatisfiable, and the negation of an unprovable formula is satisfiable.

In order to evaluate theorem provers, Balsiger et al. prepared a set of parameterized formulae which contains enough variety in consideration of combinations of the following viewpoints: (1) the number of different atomic propositions in a formula: some increase the number according to the parameter, and some does not, and (2) the depth of nested temporal operators: some increase the number according to the parameter, and some does not. But they did not mention the length of formulae against the parameter except the length can be arbitrarily long. Since the lower limit of the complexity of satisfiability checking of two-way CTL formulae is `EXPTIME` [4], it is expected that the checking time increases exponentially even if the length of formula increasing linearly with the parameter.

From that point of view, parameterized formulae in which the length increases linearly would be appropriate for evaluating satisfiability checkers. On the other hand, we also need formulae in which the length increases exponentially to satisfy the postulate 3 in Figure 1. Therefore, we propose a new characteristic of formulae: (3) the length of formula increases linearly as well as exponentially. In addition to the characteristics (1) and (3), we propose a set of formulae satisfy the following (2'), (4) and (5). (2') the depth of nested temporal operators is constant and increasing, (4) the number of modalities is constant and increasing (5) the number of occurrences of backward modalities is zero and non-zero.

# 3 Preliminaries

## 3.1 Syntax of two-way CTL

Let $\mathtt{AP}$ be a set of atomic propositions, and $\mathtt{Mod}$ be a set of modalities. For each modality $a \in \mathtt{Mod}$, we assume that there exists a modality $\overline{a} \in \mathtt{Mod}$ such that $\overline{\overline{a}} = a$. The syntax of *two-way CTL* is defined as:

$$\varphi ::= p \,|\, \neg\varphi \,|\, \varphi \vee \varphi \,|\, \mathtt{E}_A\mathtt{X}\varphi \,|\, \mathtt{E}_A[\varphi \;\mathtt{U}\; \varphi] \,|\, \mathtt{E}_A[\varphi \;\mathtt{R}\; \varphi]$$

where $p$ is an atomic proposition, and $A$ is a finite nonempty set of modalities. Intuitively, each modality corresponds to a transition labels in transition systems. In the following, we only deal with two-way CTL formulae. For formulae $\varphi,\ \psi_1, \ldots, \psi_n$ and atomic propositions $p_1, \ldots, p_n$, we write $\varphi[\psi_1/p_1, \ldots, \psi_n/p_n]$ for the formula $\varphi$ in which $p_1, \ldots, p_n$ are replaced by $\psi_1, \ldots, \psi_n$, respectively. We write $\varphi[\psi_1/p_1, \ldots, \psi_n/p_n]^2$ for the formula obtained by the same replacement to $\varphi[\psi_1/p_1, \ldots, \psi_n/p_n]$. For the formula obtained by the replacements $m$ times, we write $\varphi[\psi_1/p_1, \ldots, \psi_n/p_n]^m$ .

## 3.2 Semantics of two-way CTL

A tuple $\mathcal{M} = (M, \{R_a \mid a \in \mathtt{Mod}\}, \lambda)$ is a *Kripke structure* if $\mathcal{M}$ satisfies 1) $R_a \subseteq M \times M$, 2) $R_{\overline{a}} = \{(s,t) \mid (t,s) \in R_a\}$, and 3) $\lambda\colon \mathtt{AP} \to 2^M$. We call an element of $M$ a *state*, and $\lambda$ a *labelling function*. For labelling function $\lambda\colon \mathtt{AP} \to 2^M$, atomic proposition $x \in \mathtt{AP}$ , and $X \subseteq M$, we write $\lambda\{x \mapsto X\}$ for the labelling function $\lambda$ in which the value of $x$ is changed to $X$. For a Kripke structure $\mathcal{M} = (M, R, \lambda)$, we write $\mathcal{M}\{x \mapsto X\}$ for $(M, R, \lambda\{x \mapsto X\})$. In this paper, we do not require a Kripke structure to be total. For set of modalities $A \subseteq \mathtt{Mod}$ , an *A-path* $\sigma$ satisfies the following conditions (1), (2), and (3). (1) $\sigma$ is a finite or infinite sequence of $M$. The length of $\sigma$ is denoted $\mathrm{len}(\sigma)$, and the $i$-th element ($i \geq 0$) of $\sigma$ is denoted $\sigma_i$. When the length of $\sigma$ is infinite, we regard $i < \mathrm{len}(\sigma)$ holds for any natural number $i$. (2) For any natural number $i$ such that $i + 1 < \mathrm{len}(\sigma)$, there exist $a \in A$ and $(\sigma_i, \sigma_{i+1}) \in R_a$ holds. (3) If $\sigma$ is finite, there is no pair $(s,\ a) \in M \times A$ such that $(\sigma_{\mathrm{len}(\sigma)-1}, s) \in R_a$.

For a Kripke structure $\mathcal{M} = (M, \{R_a \mid a \in \mathtt{Mod}\}, \lambda)$, a state $s \in M$, and a formula $\varphi$, we define $\mathcal{M}, s \models \varphi$ inductively as follows:

- If $\varphi$ is of the form $p \in \mathtt{AP}$ and $s \in \lambda(p)$, then $\mathcal{M}, s \models p$.
- If $\varphi$ is of the form $\neg\varphi'$ and $\mathcal{M}, s \not\models \varphi'$, then $\mathcal{M}, s \models \neg\varphi'$.

- If $\varphi$ is of the form $\varphi_1 \vee \varphi_2$, $\mathcal{M}, s \models \varphi_1$ or $\mathcal{M}, s \models \varphi_2$, then $\mathcal{M}, s \models \varphi_1 \vee \varphi_2$.

- When $\varphi$ is of the form $\mathtt{E}_A\mathtt{X}\varphi'$, and if there exist $s' \in M$ and $a \in A$, such that $(s, s') \in R_a$ and $\mathcal{M}, s' \models \varphi'$, then $\mathcal{M}, s \models \mathtt{E}_A\mathtt{X}\varphi'$.

- When $\varphi$ is of the form $\mathtt{E}_A[\varphi_1 \ \mathtt{U} \ \varphi_2]$, if there exist $A$-path $\sigma$ and an integer $i$ such that $\sigma_0 = s$ and $i < \mathrm{len}(\sigma)$ and they satisfy both conditions (1) and (2) below, then $\mathcal{M}, s \models \mathtt{E}_A[\varphi_1 \ \mathtt{U} \ \varphi_2]$. (1) $\sigma_i \models \varphi_2$ and (2) for any integer j, $0 \leq j < i$ implies $\sigma_j \models \varphi_1$ hold.

- When $\varphi$ is of the form $\mathtt{E}_A[\varphi_1 \ \mathtt{R} \ \varphi_2]$, if there exist an $A$-path $\sigma$ such that $\sigma_0 = s$ and they satisfy at least one of conditions (1) and (2) below, then $\mathcal{M}, s \models \mathtt{E}_A[\varphi_1 \ \mathtt{R} \ \varphi_2]$. (1) For all integers $i < \mathrm{len}(\sigma)$, $\sigma_i \models \varphi_2$ or (2) there exists an integer $i < \mathrm{len}(\sigma)$, $\sigma_i \models \varphi_1$ holds and for all integers $j \leq i$, $\sigma_j \models \varphi_2$ hold.

In the following, we use symbols $\wedge, \rightarrow, \leftrightarrow, \top$ and $\perp$ with their usual meanings. Additionally, we use the following abbreviations:

- $\mathtt{A}_A\mathtt{X}\varphi \stackrel{\mathrm{def}}{=} \neg\mathtt{E}_A\mathtt{X}\neg\varphi$

- $\mathtt{A}_A[\varphi_1 \ \mathtt{U} \ \varphi_2] \stackrel{\mathrm{def}}{=} \neg\mathtt{E}_A[\neg\varphi_1 \ \mathtt{R} \ \neg\varphi_2]$

- $\mathtt{A}_A[\varphi_1 \ \mathtt{R} \ \varphi_2] \stackrel{\mathrm{def}}{=} \neg\mathtt{E}_A[\neg\varphi_1 \ \mathtt{U} \ \neg\varphi_2]$

- $\mathtt{A}_A\mathtt{G}\varphi \stackrel{\mathrm{def}}{=} \mathtt{A}_A[\perp \ \mathtt{R} \ \varphi]$

- $\mathtt{A}_A\mathtt{F}\varphi \stackrel{\mathrm{def}}{=} \mathtt{A}_A[\top \ \mathtt{U} \ \varphi]$

- $\mathtt{E}_A\mathtt{G}\varphi \stackrel{\mathrm{def}}{=} \mathtt{E}_A[\perp \ \mathtt{R} \ \varphi]$

- $\mathtt{E}_A\mathtt{F}\varphi \stackrel{\mathrm{def}}{=} \mathtt{E}_A[\top \ \mathtt{U} \ \varphi]$

We assume that `Mod` includes a modality $a$. We omit the suffix $\{a\}$ in formulae, for example we write $\mathtt{AX}\varphi$ for $\mathtt{A}_{\{a\}}\mathtt{X}\varphi$. For an integer $n \geq 0$, we write $\mathtt{EX}^n\varphi$ for the formula that $\varphi$ is preceded by $n$ EXs, i.e., $\mathtt{EX}^0\varphi \stackrel{\mathrm{def}}{=} \varphi$ and $\mathtt{EX}^{n+1}\varphi \stackrel{\mathrm{def}}{=} \mathtt{EX}(\mathtt{EX}^n\varphi)$.

A formula $\varphi$ is *valid* if $\mathcal{M}, s \models \varphi$ holds for an arbitrary Kripke structure $\mathcal{M}$ and for an arbitrary state $s$ of $\mathcal{M}$. A formula $\varphi$ is *satisfiable* if there exist a Kripke structure $\mathcal{M}$ and a state $s$ such that $\mathcal{M}, s \models \varphi$ holds. Note that the negation of an unsatisfiable formula is valid, and the negation of an non-valid formula is satisfiable. We write $[\![\varphi]\!]_{\mathcal{M}}$ for the set of states in Kripke structure $\mathcal{M}$ satisfying formula $\varphi$, i.e., $[\![\varphi]\!]_{\mathcal{M}} = \{s \in M \mid \mathcal{M}, s \models \varphi\}$ where $\mathcal{M} = (M, R, \lambda)$.

# 4 Automatic formulae generation methods

In this section, we give construction methods which generate formulae with a natural number parameter, i.e., parameterized formulae. The outline of the method is (1) to give a *seed* formula $\chi(0)$, and then (2) to give a construction method of formulae $\chi(n)$ for arbitrary positive integer parameter $n$ from $\chi(n-1)$. We call the procedure (2) a *complication.* We say a *complication preserves validity* $\chi(n)$ is valid for any $n$, provided that $\chi(0)$ is valid. We say a *complication preserves satisfiability* $\chi(n)$ is satisfiable for any $n$, provided that $\chi(0)$ is satisfiable. Both satisfiable and unsatisfiable formulae are required to evaluate satisfiability checkers. In the following, we only discuss complications preserving validity, because we can obtain unsatisfiable formulae by negating valid formulae.

## 4.1 Complication preserving validity to generate formulae

A simple method to obtain valid parameterized formulae is to use a substitution which replaces atomic propositions in a valid formula with complicated formulae. That is, let $\alpha_0$ and $\varphi$ be formulae and $x$ be an atomic proposition appearing in $\alpha_0$ and $\varphi$, then, we can obtain parameterized formula $\chi_0$ as:

$$\chi_0(n) = \alpha_0[\varphi/x]^n$$

if $\alpha_0$ is valid.

In this complication, if a satisfiability checker detects the base valid formula $\alpha_0$, then any complicated formulae above can be checked in constant time. This is an example that a 'simple trick' can be used efficiently for solving parameterized formulae. This means that such complication is not appropriate for evaluation of satisfiability checkers. In this paper, we give 'non-trivial' complication methods. First, we show a theorem which will be the basis of our complications preserving validity.

Here are some preliminaries. We say a formula is in *positive form* if it contains no occurrence of the negation operator $\neg$ except immediately before atomic propositions. Every formula can be transformed to a formula in positive form. An atomic proposition $x$ occurs *positively* in formula $\varphi$ if the negation operator precedes no occurrences of $x$ in $\varphi$. Let $\varphi$ be a formula, $x$ be an atomic proposition, $\mathcal{M} = (M, R, \lambda)$ be a Kripke structure, and $X, X' \subseteq M$ be sets of states. If $x$ occurs positively in $\varphi$ and $X \subseteq X'$, then we have the following relation:

$$[\![\varphi]\!]_{\mathcal{M}\{x\mapsto X\}} \subseteq [\![\varphi]\!]_{\mathcal{M}\{x\mapsto X'\}} \tag{1}$$

The relation (1) can be proved by an induction on construction of $\varphi$.

The following theorem gives the basis for our complications. Some corollaries are derived from the theorem. Our complications are generated based on these corollaries.

**Theorem 1.** *Let $x$ be an atomic proposition, $\alpha_1, \beta_1$ be formulae in positive form in which $x$ occurs positively, and $\alpha_0, \beta_0$ be formulae. If the two formulae $\alpha_1 \to \beta_1$ and $\alpha_0 \to \beta_0$ are valid, then so is formula $\alpha_1[\alpha_0/x] \to \beta_1[\beta_0/x]$.*

**Proof** Let $\mathcal{M}$ be an arbitrary Kripke structure. It is sufficient to show that $[\![\alpha_1[\alpha_0/x]]\!]_{\mathcal{M}} \subseteq [\![\beta_1[\beta_0/x]]\!]_{\mathcal{M}}$ holds. The theorem can be shown as follows:

$$\begin{aligned}
[\![\alpha_1[\alpha_0/x]]\!]_{\mathcal{M}} &= [\![\alpha_1]\!]_{\mathcal{M}\{x\mapsto X_1\}} && (2)\\
&\subseteq [\![\beta_1]\!]_{\mathcal{M}\{x\mapsto X_1\}} && (3)\\
&\subseteq [\![\beta_1]\!]_{\mathcal{M}\{x\mapsto X_2\}} && (4)\\
&= [\![\beta_1[\beta_0/x]]\!]_{\mathcal{M}}
\end{aligned}$$

where $X_1 = [\![\alpha_0]\!]_{\mathcal{M}}$ and $X_2 = [\![\beta_0]\!]_{\mathcal{M}}$, (3) is proved by the fact formula $\alpha_1 \to \beta_1$ is valid and (1), (4) is proved because $\alpha_0 \to \beta_0$ is valid. □

Note that the assignment used in Theorem 1 is not simple like $\chi_0$. We have following corollaries by repeatedly applying Theorem 1.

**Corollary 1.** *Let $x$ be an atomic proposition, $\alpha_i, \beta_i$ $(1 \leq i \leq n)$ be formulae in positive form in which $x$ occurs positively, and $\alpha_0, \beta_0$ be formulae. If $\alpha_i \to \beta_i$ $(1 \leq i \leq n)$ and $\alpha_0 \to \beta_0$ are valid, then so are $\varphi_i \to \psi_i$ $(1 \leq i \leq n)$, where $\varphi_0 = \alpha_0$, $\psi_0 = \beta_0$, $\varphi_i = \alpha_i[\varphi_{i-1}/x]$, and $\psi_i = \beta_i[\psi_{i-1}/x]$ $(1 \leq i \leq n)$.* □

**Corollary 2.** *Let $x_i$ $(1 \leq i \leq m)$ be atomic propositions, $\alpha$, $\beta$ be formulae in positive form in which $x_1, \ldots, x_m$ occur positively, and $\alpha_i, \beta_i$ $(0 \leq i \leq m)$ be formulae. If $\alpha \to \beta$ and $\alpha_i \to \beta_i$ $(0 \leq i \leq m)$ are valid, then so is $\alpha[\alpha_1/x_1, \ldots, \alpha_m/x_m] \to \beta[\beta_1/x_1, \ldots, \beta_m/x_m]$.* □

These theorem and corollaries can be applied to any logical systems of which models are Kripke structures, e.g., CTL, LTL, and CTL*. Therefore, our formula generation methods based on them mentioned below can be applied to other logical systems.

We show how to construct parameterized formulae based on the corollaries. To apply the corollaries, we need a number of formulae of the form $\alpha \to \beta$. We show some simple ways to give such formulae. To obtain parameterized formulae, we need seeds of the form $\alpha_0 \to \beta_0$, which we will discuss in Section 4.2.

**Example 1.** First we consider the case in which for any $i$, $j$ $(1 \leq i, j \leq n)$, $\alpha_i = \alpha_j$ and $\beta_i = \beta_j$ in Corollary 1. For example, let $\alpha_i = \alpha$, and $\beta_i = \beta$ $(1 \leq i \leq n)$ where $\alpha = \mathtt{EX}x$, and $\beta = \mathtt{EX}x$. By applying Corollary 1 to a valid formula $\mathtt{EX}x \to \mathtt{EX}x$, we obtain the following parameterized formula $\chi_1$:

$$\chi_1(n) = \mathtt{EX}^n \alpha_0 \to \mathtt{EX}^n \beta_0$$

This parameterized formula corresponds to the inference rule in a deduction system for CTL [4]:

$$\frac{\gamma \to \delta}{\mathtt{EX}\gamma \to \mathtt{EX}\delta}$$

A satisfiability checker which knows the deduction rule can easily solve formulae generated by $\chi_1$. We therefore consider a derived parameterized formula as follows:

$$\chi_2(n) = \bigvee_{1 \leq i \leq n} \left(\mathtt{EX}^i(\alpha_0) \to \mathtt{EX}^n(\beta_0)\right)$$

This complication corresponds to the complication *k_d4_p* introduced in [1]. In our methods, any valid formula of the form $\alpha \to \beta$ can be used to construct parameterized formulae. For example, $\alpha_i = \alpha_a$, $\beta_i = \beta_a$ ($i$: even), $\alpha_i = \alpha_b$, $\beta_i = \beta_b$ ($i$: odd), $\alpha_a = x \wedge \mathtt{E}_a\mathtt{X}x$, $\beta_a = \mathtt{E}_a\mathtt{X}\mathtt{E}_{\overline{a}}\mathtt{X}(x)$, $\alpha_b = x \wedge \mathtt{E}_b\mathtt{X}x$, and $\beta_b = \mathtt{E}_b\mathtt{X}\mathtt{E}_{\overline{b}}\mathtt{X}(x)$. Note that both formulae $x \wedge \mathtt{E}_a\mathtt{X}x \to \mathtt{E}_a\mathtt{X}\mathtt{E}_{\overline{a}}\mathtt{X}(x)$ and $x \wedge \mathtt{E}_b\mathtt{X}x \to \mathtt{E}_b\mathtt{X}\mathtt{E}_{\overline{b}}\mathtt{X}(x)$ are valid. Consequently, the obtained parameterized formula can be written as follows:

$$\chi_3(n) = \bigvee_{1 \leq i \leq n} (\varphi_i \to \psi_n),$$

where $\varphi_0 = \alpha_0$, $\psi_0 = \beta_0$, $\varphi_i = \varphi_{i-1} \wedge \mathtt{E}_a\mathtt{X}\varphi_{i-1}$ ($i$ is odd), $\varphi_i = \varphi_{i-1} \wedge \mathtt{E}_b\mathtt{X}\varphi_{i-1}$ ($i$ is even), $\psi_i = \mathtt{E}_a\mathtt{X}\mathtt{E}_{\overline{a}}\mathtt{X}(\psi_{i-1})$ ($i$ is odd), $\psi_i = \mathtt{E}_b\mathtt{X}\mathtt{E}_{\overline{b}}\mathtt{X}(\psi_{i-1})$ ($i$ is even), and $\alpha_0$ and $\beta_0$ are any formula such that $\alpha_0 \to \beta_0$ is valid. □

**Example 2.** Next, we consider a complication obtained by applying Corollary 1 to formulae $\alpha_i$ and $\beta_i$ ($1 \leq i \leq n$) where all $\alpha_i$s are identical except atomic propositions and so are $\beta_i$s. For example, let $\alpha_i = \mathtt{E}[x \;\mathtt{U}\; q_i]$, and $\beta_i = \mathtt{E}[x \;\mathtt{U}\; q_i]$ ($1 \leq i \leq n$), where $q_i$ is an atomic proposition and $\alpha_i \to \beta_i$ ($1 \leq i \leq n$) is valid. Then we obtain a parameterized formula $\chi_4$ by applying Corollary 1:

$$\chi_4(n) = \varphi_n \to \psi_n$$

In the formula above, $\varphi_0 = \alpha_0$, $\psi_0 = \beta_0$, $\varphi_n = \mathtt{E}[\varphi_{n-1} \;\mathtt{U}\; q_n]$, and $\psi_n = \mathtt{E}[\psi_{n-1} \;\mathtt{U}\; q_n]$. The number of atomic propositions in parameterized formula $\chi_4$ increases according to the parameter $n$. While it is constant in the parameterized formula in Example 1. For evaluating satisfiability checkers, it is preferable to prepare both of these two types of parameterized formulae. As we will describe in Section 4.2, formula $\mathtt{A}[c_0 \;\mathtt{U}\; p_0] \wedge \mathtt{AG}(p_0 \to \mathtt{AG}(\neg c_0 \wedge x)) \to \mathtt{AFAG}(\neg c_0 \wedge x)$ is valid. If we use this formula as $\alpha_i \to \beta_i$, we obtain a parameterized formula $\chi_5$: $\chi_5(n) = \varphi_n \to \psi_n$, where $\varphi_n = \mathtt{A}[c_n \;\mathtt{U}\; p_n] \wedge \mathtt{AG}(p_n \to \mathtt{AG}(\neg c_n \wedge x))$, and $\psi_n = \mathtt{AFAG}(\neg c_n \wedge x)$. □

**Example 3.** We give a parameterized formula in which the depth of temporal operators is constant. We use formulae $x \wedge \mathtt{A}[p_i \;\mathtt{U}\; q]$ and $x \wedge (q \vee (p_i \wedge \mathtt{AXA}[p_i \;\mathtt{U}\; q]))$ ($1 \leq i \leq n$) for $\alpha_i$ and $\beta_i$ respectively in Corollary 1. Note that $\mathtt{A}[p \;\mathtt{U}\; q] \to q \vee (p \wedge \mathtt{AXA}[p \;\mathtt{U}\; q])$ is valid. Then we obtain a parameterized formula:

$$\chi_6(n) = \bigwedge_{1 \leq i \leq n} \mathtt{A}[p_i \;\mathtt{U}\; q] \to \bigwedge_{1 \leq i \leq n} (q \vee (p_i \wedge \mathtt{AXA}[p_i \;\mathtt{U}\; q]))$$

□

**Example 4.** We consider a complication which replaces formulae $\alpha_i$, and $\beta_i$ ($1 \leq i \leq n$) in Corollary 1 with the same formulae except modalities. For example, let $\alpha_i = \mathtt{A}_{a_i,a_{i+1}}\mathtt{G}x$, and $\beta_i = \mathtt{A}_{a_i}\mathtt{G}x$ ($1 \leq i \leq n$). Since $\alpha_i \to \beta_i$ is valid, we can obtain a parameterized formula by the same way as in Example 2. Then we have a parameterized formula $\chi_7$: $\chi_7(n) = \varphi_n \to \psi_n$, where $\varphi_i = \mathtt{A}_{a_i,a_{i+1}}\mathtt{G}x$, and $\psi_i = \mathtt{A}_{a_i}\mathtt{G}x$ ($0 \leq i \leq n$). In this parameterized formula, the number of modalities increases as well according to the parameter. □

**Example 5.** We show a complication derived from Corollary 2 in which formulae $\alpha_i$ and $\beta_i$ are replaced with the same formulae. For example, we take $\varphi_n$ and $\psi_n$ introduced in Example 2 for $\alpha$ and $\beta$ respectively. Let $\alpha_i = \alpha'$, and $\beta_i = \beta'$ ($1 \leq i \leq n$), where $\alpha' = x \wedge \mathtt{E}_a\mathtt{X}\top$, and $\beta' = \mathtt{E}_a\mathtt{XE}_{\overline{a}}\mathtt{X}x$. Since formula $x \wedge \mathtt{E}_a\mathtt{X}\top \to \mathtt{E}_a\mathtt{XE}_{\overline{a}}\mathtt{X}x$ is valid, we can apply Corollary 2, and obtain a parameterized formula $\chi_8$:

$$\chi_8(n) = \varphi'_n \to \psi'_n$$

where $\varphi'_0 = \alpha_0$, $\psi'_0 = \beta_0$, $\varphi'_n = \mathtt{E}[\varphi'_{n-1} \;\mathtt{U}\; (p \wedge \mathtt{E}_a\mathtt{X}\top)]$, and $\psi'_n = \mathtt{E}[\psi'_{n-1} \;\mathtt{U}\; \mathtt{E}_a\mathtt{XE}_{\overline{a}}\mathtt{X}p]$. □

Now, we have some complications preserving validity which generate valid formulae from seeds. In the next subsection, we describe several ways to obtain appropriate seeds.

## 4.2 Methods to obtain simple valid formulae

In Example 1, we use formula $\mathtt{EX}x \rightarrow \mathtt{EX}x$. As mentioned in the example, such a trivial formula can be used for constructing parameterized formulae. In general, a valid formula of the form $\alpha \rightarrow \beta$, where $\alpha$ and $\beta$ are different from each other, would produce better results to evaluate satisfiability checkers. In the rest of this subsection, we show some easy ways to obtain simple valid formulae.

**Trivial Theorems** Some CTL theorems are described in standard textbooks on system verification using temporal logics [2, 7]. For example, $\mathtt{E}[\varphi\ \mathtt{U}\ \psi] \leftrightarrow \psi \vee (\varphi \wedge \mathtt{EXE}[\varphi\ \mathtt{U}\ \psi])$, $\mathtt{A}[p\ \mathtt{U}\ q] \leftrightarrow \neg\mathtt{E}[\neg q\ \mathtt{U}\ (\neg p \vee \neg q) \wedge \neg\mathtt{EG}(\neg q)]$, etc. Axioms of a formal system for CTL are also useful references [4, 8]. For example, $\mathtt{EX}(\varphi \vee \psi) \leftrightarrow \mathtt{EX}\varphi \vee \mathtt{EX}\psi$, $\mathtt{AX}\varphi \wedge \mathtt{AX}\psi \rightarrow \mathtt{AX}(\varphi \wedge \psi)$, etc. In the case of two-way CTL, we can easily see that $\mathtt{A}_{a,b}\mathtt{G}x \rightarrow \mathtt{A}_a\mathtt{G}x,\ \ x \wedge \mathtt{E}_a\mathtt{X}\top \rightarrow \mathtt{E}_a\mathtt{XE}_{\overline{a}}\mathtt{X}x$ are valid.

**Combination of Patterns** Dwyer et al. analyzed frequently observed properties for system verification and gave their patterns [3]. They call the patterns *specification patterns* and express them in temporal logics. For example, a property "an event $S$ does not occur before an event $P$ occurs" can be expressed by specification patterns. The corresponding CTL formula is $\mathtt{A}[\neg S\ \mathtt{U}\ (P \vee \mathtt{AG}\neg P)]$. We can use specification patterns to construct simple valid formulae. We name the formula $\mathtt{A}[\neg S\ \mathtt{U}\ (P \vee \mathtt{AG}\neg P)]$ $\mathtt{absence}(P, S)$. Then we have a non-trivial valid formula $\mathtt{AF}P \wedge \mathtt{AG}\neg S \rightarrow \mathtt{absence}(P, S)$. It is valid since if $P$ occurs eventually and $S$ never occurs, then $S$ cannot occur before $P$.

Another way is to consider models which satisfy some formulae. For example, we consider a property "event $c_1$ will eventually occurs at some states, and from these states, event $c_1$ permanently holds and event $c_0$ never holds". This property can be expressed as $\mathtt{AFAG}(\neg c_0 \wedge c_1)$ in CTL. On the other hand, the models satisfying $\mathtt{A}[c_0\ \mathtt{U}\ p_0] \wedge \mathtt{AG}(p_0 \rightarrow \mathtt{AG}(\neg c_0 \wedge c_1))$ also satisfy the property above. That is, $c_0$ holds until certain state where $p_0$ holds, but $c_0$ does not hold and $c_1$ holds permanently from the state. From the discussion above, we have a valid formula $\mathtt{A}[c_0\ \mathtt{U}\ p_0] \wedge \mathtt{AG}(p_0 \rightarrow \mathtt{AG}(\neg c_0 \wedge c_1)) \rightarrow \mathtt{AFAG}(\neg c_0 \wedge c_1)$.

## 4.3 Satisfiable parameterized formula

In order to generate satisfiable formulae, we consider a property $P(n)$ on a Kripke structure where $n$ is a natural number parameter. A statement "within $n$ steps, there is a state where an atomic proposition $p$ holds " is an example of a property $P(n)$. We adopt such $P(n)$ that the size of Kripke structures which satisfies $P(n)$ increases according to parameter $n$. Parameterized formulae can be obtained by expressing such properties in two-way CTL. The example mentioned above can be expressed as $\chi_9(n) = \mathtt{EX}^n(p)$. We show some examples.

**Example 6.** We show some variations of $\chi_9$. Let $\chi_{10}(n) = \bigwedge_{0 \le i \le n} \mathtt{EX}^i(\alpha_0)$ where $\alpha_0$ is an arbitrary formula which does not contain reverse modalities. Another variation is: $\chi_{11}(n) = \bigwedge_{1 \le i \le n} \left(\mathtt{EX}^{i-1}(\neg\alpha_0) \wedge \mathtt{EX}^i(\alpha_0)\right)$. But in this case, we have to choose an appropriate formula $\alpha_0$. □

**Example 7.** Two-way CTL does not have the finite model property. For example, if a Kripke structure satisfies the formula $z \wedge \mathtt{A}_a\mathtt{X}\mathtt{A}_a\mathtt{G}(\neg z) \wedge (\mathtt{A}_a\mathtt{G}(\mathtt{E}_a\mathtt{X}p \wedge \mathtt{A}_{\overline{a}}\mathtt{F}z))$, then the set of its states is infinite. We can prove it as follows. We assume that the formula holds on a state $s$ in a finite Kripke structure. Then there exists an infinite $A$-path $\sigma$ starting with $s$, because $s \models \mathtt{A}_a\mathtt{G}\mathtt{E}_a\mathtt{X}p$. Since the state space is finite, there exist $i$ and $j$ with $\sigma_i = \sigma_j$ $(i < j)$. For the loop from $\sigma_i$ to $\sigma_j$ through backward modality $\overline{a}$, we have $\sigma_i \models \mathtt{E}_{\overline{a}}\mathtt{G}\neg z$, which contradicts $s \models \mathtt{A}_a\mathtt{G}(\mathtt{E}_a\mathtt{X}p \wedge \mathtt{A}_{\overline{a}}\mathtt{F}z)$. Based on the formula, we can devise parameterized formulae. For example, a number line, which continues in the right and left infinitely, can be expressed as follows:

$$\begin{aligned}\chi_{12}(0) \quad = \quad & z_0 \wedge \mathtt{A}_{a_0}\mathtt{X}\mathtt{A}_{a_0}\mathtt{G}(\neg z_0) \wedge \mathtt{A}_{\overline{a_0}}\mathtt{X}\mathtt{A}_{\overline{a_0}}\mathtt{G}(\neg z_0) \wedge \\ & \mathtt{A}_{a_0}\mathtt{G}(\mathtt{E}_{a_0}\mathtt{X}p_0 \wedge \mathtt{A}_{\overline{a_0}}\mathtt{F}z_0) \wedge \\ & \mathtt{A}_{\overline{a_0}}\mathtt{G}(\mathtt{E}_{\overline{a_0}}\mathtt{X}n_0 \wedge \mathtt{A}_{a_0}\mathtt{F}z_0)\end{aligned}$$

This formula can be parameterized as $\chi_{12}(n) = \alpha_n \wedge \chi_{12}(n-1) \wedge \beta_n[\chi_{12}(n-1)/x] \wedge \gamma_n[\chi_{12}(n-1)/x]$ for $n \geq 1$ where

$$\begin{aligned}\alpha_i \quad &= \quad z_i \wedge \mathtt{A}_{a_i}\mathtt{X}\mathtt{A}_{a_i}\mathtt{G}(\neg z_j) \wedge \mathtt{A}_{\overline{a_i}}\mathtt{X}\mathtt{A}_{\overline{a_i}}\mathtt{G}(\neg z_j), \\ \beta_i \quad &= \quad \mathtt{E}_{a_i}\mathtt{X}(p_i) \wedge \mathtt{A}_{a_i}\mathtt{G}(\mathtt{E}_{a_i}\mathtt{X}(p_i) \wedge \mathtt{A}_{\overline{a_i}}\mathtt{F}(z_i) \wedge x), \text{and} \\ \gamma_i \quad &= \quad \mathtt{E}_{\overline{a_i}}\mathtt{X}(p_i) \wedge \mathtt{A}_{\overline{a_i}}\mathtt{G}(\mathtt{E}_{\overline{a_i}}\mathtt{X}(p_i) \wedge \mathtt{A}_{a_i}\mathtt{F}(z_i) \wedge x).\end{aligned}$$

The length of the parameterized formulae increases exponentially. □

**Example 8.** We consider a complication in which the number of modalities increases. Formula $\chi_{13}(0) = \mathtt{A}_{a_1}\mathtt{G}(\neg\alpha_0) \wedge \mathtt{A}_{a_2}\mathtt{G}(\neg\alpha_0) \wedge \mathtt{E}_{a_1,a_2}\mathtt{F}(\alpha_0)$ expresses that $\alpha_0$ is not reachable through the modality $a_1$ only or $a_2$ only, but it is reachable if both modalities $a_1$ and $a_2$ are allowed. A parameterized formula is given as follows:

$$\begin{aligned}\chi_{13}(n) \quad = \quad & \mathtt{A}_{a_1,\ldots,a_{n-1}}\mathtt{G}(\neg\alpha_0) \wedge \\ & \mathtt{A}_{a_1,\ldots,a_{n-2},a_n}\mathtt{G}(\neg\alpha_0) \wedge \cdots \wedge \\ & \mathtt{A}_{a_2,\ldots,a_n}\mathtt{G}(\neg\alpha_0) \wedge \mathtt{E}_{a_1,\ldots,a_n}\mathtt{F}(\alpha_0)\end{aligned}$$

This complication is rather simple, but it can be combined with other complications in order to obtain parameterized formulae with increasing modalities. □

**Example 9.** We consider a complication in which the depth of nested temporal operators is constant. Let:

$$\begin{aligned}\chi_{14}(0) \quad = \quad & \mathtt{A}_a\mathtt{F}\alpha_0 \wedge \mathtt{A}_a\mathtt{F}\alpha_1 \wedge \neg\alpha_0 \wedge \neg\alpha_1 \wedge \\ & \mathtt{A}_a\mathtt{G}(\alpha_0 \rightarrow \neg(\alpha_1 \vee \mathtt{E}_a\mathtt{X}\alpha_1 \vee \mathtt{E}_{\overline{a}}\mathtt{X}\alpha_1)) \wedge \\ & \mathtt{A}_a\mathtt{G}(\alpha_1 \rightarrow \neg(\alpha_0 \vee \mathtt{E}_a\mathtt{X}\alpha_0 \vee \mathtt{E}_{\overline{a}}\mathtt{X}\alpha_0))\end{aligned}$$

then it is generalized into a parameterized formula: $\chi_{14}(n) = \bigwedge_{1\leq i\leq n}(\mathtt{A}_a\mathtt{F}\alpha_i \wedge \neg\alpha_i) \wedge \bigwedge_{0\leq i\leq n}\mathtt{A}_a\mathtt{G}(\alpha_i \rightarrow \neg(\gamma_{n,i} \vee \delta_{n,i,a} \vee \delta_{n,i,\overline{a}}))$ where $\gamma_{i,j} = \alpha_0 \vee \cdots \alpha_{j-1} \vee \alpha_{j+1} \cdots \vee \alpha_i$ and $\delta_{i,j,m} = \mathtt{E}_m\mathtt{X}\alpha_0 \vee \cdots \vee \mathtt{E}_m\mathtt{X}\alpha_{j-1} \vee \mathtt{E}_m\mathtt{X}\alpha_{j+1} \vee \cdots \vee \mathtt{E}_m\mathtt{X}\alpha_i$. □

In practice, system specifications for model checking rarely contain such complicated formulae with deeply nested temporal operators such as $\chi_2(n)$ or $\chi_{10}(n)$. However, the abstraction method for pointer manipulation systems proposed by the authors [10], needs to judge the satisfiability of such complicated formulae.

Table 1: Characteristics of parameterized formulae in Figure 2

| name | sat | AP | depth | Mod | inverse | length |
|---|---|---|---|---|---|---|
| test1 | n | c | c | c | n | linear |
| test2 | n | i | c | c | n | linear |
| test3 | n | i | c | c | n | linear |
| test4 | n | c | i | c | n | linear |
| test5 | n | c | c | i | n | linear |
| test6 | n | i | c | i | n | linear |
| test7 | n | i | i | c | n | linear |
| test8 | n | c | i | i | n | linear |
| test9 | n | c | i | i | n | quadratic |
| test10 | n | i | i | i | n | linear |
| test11 | n | c | c | c | y | linear |
| test12 | n | i | c | c | y | linear |
| test13 | n | c | i | c | y | exp |
| test14 | n | c | i | c | y | quadratic |
| test15 | n | c | c | i | y | linear |
| test16 | n | i | c | i | y | linear |
| test17 | n | i | i | c | y | linear |
| test18 | n | c | i | i | y | linear |
| test19 | n | i | i | i | y | linear |
| test20 | y | c | i | c | n | quadratic |
| test21 | y | i | i | i | y | exp |
| test22 | y | c | c | i | n | linear |
| test23 | y | i | c | c | y | linear |
| test24 | y | c | i | i | n | quadratic |

# 5 Experiments

In Figure 2, we show a set of parameterized formulae generated systematically by our proposed methods in Section 4. The parameterized formulae from $\chi_1$ to $\chi_{14}$ are given in the previous section. For example, $\chi_1$ in `test4` is introduced in Example 1, and `test4`(0) is $\neg(p \wedge q \rightarrow p)$. The parameterized formulae from `test1` to `test19` are unsatisfiable, i.e. all generated formulae are obtained by negating valid formulae. The parameterized formula `test3` is introduced in Example 3. The parameterized formulae `test6` and `test16` are derived from `test3`. Parameterized formulae from `test20` to `test24` are satisfiable. The parameterized formula `test24` is a combination of $\chi_{11}$ and $\chi_{13}$, that is, it is $\chi_{13}$ with $\alpha_0 = \chi_{11}$. Table 1 shows characteristics of these parameterized formulae, where `sat` is satisfiability, 'y' means satisfiable and 'n' means unsatisfiable, `AP` is the number of atomic propositions, 'i' means the number increases according to the parameter and 'c' means constant, `depth` is the depth of nested temporal operators, 'i' means the depth increases and 'c' means constant, `Mod` is the number of modality types, 'i' means the number increases and 'c' means constant, and `inverse` is occurrence of backward modalities, 'y' means backward modalities occur and 'n' means no occurrence. In Table 1, 'linear' ('quadratic', and 'exp') means the length of a formula increases linearly (quadratically, and

$$
\begin{array}{lcl}
\mathtt{test1}(n) & = & \neg\left(\bigwedge_{0\le i\le n}(p) \to \bigwedge_{0\le i\le n}(p)\right)\ (n\ge 0)\\
\mathtt{test2}(0) & = & \neg(\mathtt{AF}P \wedge \mathtt{AG}(\neg S) \to \mathtt{absence}(P,S))\\
\mathtt{test2}(n) & = & \neg\chi_6(n),\\
\mathtt{test3}(n) & = & \neg((\mathtt{EX}(u\vee v)\wedge \bigwedge_{1\le i\le n}\mathtt{A}[p_i\ \mathtt{U}\ q]) \to (\mathtt{EX}u \vee \mathtt{EX}v \wedge \bigwedge_{1\le i\le n}(q\vee \mathtt{AXA}[p_i\ \mathtt{U}\ q])))\\
\mathtt{test4}(n) & = & \neg\chi_1(n)\ (n\ge 1),\quad \neg(p\wedge q\to p)\ (n=0)\\
\mathtt{test5}(0) & = & \neg((p\wedge \mathtt{E}_{a0}\mathtt{X}p)\to \mathtt{E}_{a0}\mathtt{X}p)\ (n=0)\\
\mathtt{test5}(n) & = & \neg(\varphi_n\to\psi_n)\ (n\ge 1), \varphi_n = (p\wedge \mathtt{E}_{a_n}\mathtt{X}p)\wedge\varphi_{n-1}, \psi_n = \mathtt{E}_{a_n}\mathtt{X}p\\
\mathtt{test6}(n) & = & \neg((\mathtt{E}_{a_0}\mathtt{X}(u\vee v)\wedge \bigwedge_{1\le i\le n}\mathtt{A}_{a_i}[p_i\ \mathtt{U}\ q]) \to\\
 & & (\mathtt{E}_{a_0}\mathtt{X}u\vee \mathtt{E}_{a_0}\mathtt{X}v\wedge \bigwedge_{1\le i\le n}(q\vee \mathtt{A}_{a_i}\mathtt{X}\mathtt{A}_{a_i}[p_i\ \mathtt{U}\ q])))\\
\mathtt{test7}(n) & = & \neg\chi_5(n)(n\ge 0)\\
\mathtt{test8}(n) & = & \neg\chi_7(n)(n\ge 1),\ \neg(\mathtt{E}[p\ \mathtt{U}\ q]\to \mathtt{EF}q)\ (n=0)\\
\mathtt{test9}(0) & = & \neg(\mathtt{E}[p\ \mathtt{U}\ q]\to \mathtt{EF}q)\\
\mathtt{test9}(n) & = & \neg(\varphi_n\to\psi_n)\ (n\ge 1), \varphi_i = \mathtt{E}_{a_i}\mathtt{F}\varphi_{i-1}, \psi_i = \mathtt{E}_{a_i}\mathtt{F}\psi_{i-1}\\
\mathtt{test10}(n) & = & \neg(\varphi_n\to\psi_n)\ (n\ge 0), \varphi_0=\alpha_0, \psi_0=\beta_0,\\
 & & \varphi_n = \alpha_n[\varphi_{n-1}/c_n]\ (n\ge 1), \psi_{n\ge 1} = \beta_n[\psi_{n-1}/c_n]\ (n\ge 1),\\
 & & \alpha_n = \mathtt{A}_{a_n}[c_n\ \mathtt{U}\ p_n]\wedge\quad \mathtt{A}_{a_n}\mathtt{G}(p_n\to \mathtt{A}_{a_n}\mathtt{G}(\neg c_n\wedge c_{n+1}))\ (n\ge 0),\\
 & & \beta_n = \mathtt{A}_{a_n}\mathtt{F}\mathtt{A}_{a_n}\mathtt{G}(\neg c_n\wedge c_{n+1})\ (n\ge 0)\\
\mathtt{test11}(n) & = & \neg(\bigwedge_{0\le i\le n}(p\to \mathtt{E}_a\mathtt{X}\mathtt{E}_{\overline{a}}\mathtt{X}p)\to\\
 & & (\bigwedge_{0\le i\le n}(p\to \mathtt{E}_a\mathtt{X}\mathtt{E}_{\overline{a}}\mathtt{X}p))\ (n\ge 0)\\
\mathtt{test12}(n) & = & \neg(\bigvee_{0\le i\le n}(p\wedge \mathtt{E}_a\mathtt{G}q_n)\to\\
 & & \bigvee_{0\le i\le n}(q_n\to \mathtt{E}_{\overline{a}}\mathtt{F}p))\ (n\ge 0)\\
\mathtt{test13}(n) & = & \neg\chi_3(n)\ (n\ge 1),\ \neg(\mathtt{A}_{a,b}\mathtt{G}p\to \mathtt{A}_a\mathtt{G}p)\ (n=0)\\
\mathtt{test14}(0) & = & \neg(\mathtt{AX}p\wedge \mathtt{AX}q\to \mathtt{AX}(p\wedge q))\\
\mathtt{test14}(n) & = & \neg\chi_8(n)\ (n\ge 1),\\
\mathtt{test15}(n) & = & \neg(\bigvee_{0\le i\le n}(p\wedge \mathtt{E}_{a_i}\mathtt{X}\mathtt{E}_{\overline{a_i}}\mathtt{X}p)\to\\
 & & \bigvee_{0\le i\le n}(q\to \mathtt{E}_{\overline{a_i}}\mathtt{F}p))\ (n\ge 0)\\
\mathtt{test16}(n) & = & \neg((\mathtt{E}_{a_0}\mathtt{X}(u\vee v)\wedge \bigwedge_{1\le i\le n}(p\wedge \mathtt{E}_{a_i}\mathtt{X}p))\to\\
 & & (\mathtt{E}_{a_0}\mathtt{X}u\vee \mathtt{E}_{a_0}\mathtt{X}v\wedge \bigwedge_{1\le i\le n}\mathtt{E}_{a_i}\mathtt{X}\mathtt{E}_{\overline{a_i}}\mathtt{X}p))\\
\mathtt{test17}(n) & = & \neg(\varphi_n\to\psi_n)\ (n\ge 0), \varphi_0=\alpha_0,\ \psi_0=\beta_0,\\
 & & \varphi_n = \varphi_{n-1}[\alpha_n/c_n]\ (n\ge 1), \psi_n = \phi_{n-1}[\beta_n/c_n]\ (n\ge 1),\\
 & & \alpha_{n\ge 0} = \mathtt{A}_a[c_n\ \mathtt{U}\ p_n]\wedge\quad \mathtt{A}_a\mathtt{G}(p_n\to \mathtt{A}_a\mathtt{G}(\neg c_n\wedge c_{n+1}))\ (n\ge 0),\\
 & & \beta_n = \mathtt{A}_a\mathtt{G}((\neg c_n\wedge c_{n+1})\to \mathtt{E}_{\overline{a}}\mathtt{F}p_0)\ (n\ge 0)\\
\mathtt{test18}(n) & = & \neg(\varphi_n\to\psi_n)\ (n\ge 0), \varphi_0=\alpha_0,\ \psi_0=\beta_0,\\
 & & \varphi_n = \varphi_{n-1}[\alpha_n/q]\ (n\ge 1), \psi_n = \psi_{n-1}[\beta_n/q]\ (n\ge 1),\\
 & & \alpha_n = p\wedge \mathtt{E}_{a_n}[\neg p\ \mathtt{U}\ q]\ (n\ge 0), \beta_n = q\to \mathtt{E}_{\overline{a_n}}\mathtt{F}p\ (n\ge 0)\\
\mathtt{test19}(n) & = & \neg(\varphi_n\to\psi_n)\ (n\ge 0), \varphi_0=\alpha_0,\ \psi_0=\beta_0,\\
 & & \varphi_n = \varphi_{n-1}[\alpha_n/q_{n-1}]\ (n\ge 1), \psi_n = \psi_{n-1}[\beta_n/q_{n-1}]\ (n\ge 1),\\
 & & \alpha_n = p_n\wedge \mathtt{E}_{a_n}[\neg p_n\ \mathtt{U}\ q_n]\ (n\ge 0), \beta_n = q_n\to \mathtt{E}_{\overline{a_n}}\mathtt{F}p_n\ (n\ge 0)\\
\mathtt{test20}(n) & = & \chi_{11}(n)\ (n\ge 1),\quad p\wedge q\ (n=0)\\
\mathtt{test21}(n) & = & \chi_{12}(n)\ (n\ge 0)\\
\mathtt{test22}(0) & = & \mathtt{A}_a\mathtt{G}(\mathtt{E}_a\mathtt{X}(p\wedge q)\wedge \mathtt{E}_a\mathtt{X}(\neg q\wedge p))\wedge \mathtt{A}_{\overline{a}}\mathtt{F}\neg p\\
\mathtt{test22}(n) & = & \chi_{13}(n)\ (n\ge 1)\\
\mathtt{test23}(n) & = & \chi_{14}(n)\ (n\ge 0)\\
\mathtt{test24}(n) & = & \chi_{13}(n)[\chi_{11}(n)/\alpha_0]\ (n\ge 1),\ \chi_{11}(0)\ (n=0)
\end{array}
$$

Figure 2: A set of parameterized formulae generated by proposed methods

Table 2: Measurements results of the maximum parameter by a satisfiability checker

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| `test1` | >20 | `test2` | 7 | `test3` | 11 | `test4` | 15 |
| `test5` | >20 | `test6` | 7 | `test7` | 3 | `test8` | 15 |
| `test9` | 14 | `test10` | 3 | `test11` | >20 | `test12` | >20 |
| `test13` | 14 | `test14` | >20 | `test15` | >20 | `test16` | 13 |
| `test17` | 4 | `test18` | >20 | `test19` | >20 | | |
| `test20` | >20 | `test21` | 3 | `test22` | >20 | `test23` | 6 |
| `test24` | 5 | | | | | | |

Table 3: Environment.

| | |
|---|---|
| OS | Red Hat Enterprise Linux ES3 |
| CPU | Intel Xeon 3.0GHz |
| Memory | 4GB |
| JVM | Java2 1.5.0_03 |

exponentially respectively). Since unsatisfiable formulae are more difficult to solve for satisfiability checkers in general, a fewer number of satisfiable formulae are in the list. We mainly prepare formulae which `length` is linear. For unsatisfiable formulae, the table covers all combinations of the 4 characteristics, `AP`, `depth`, `Mod`, and `inverse`.

We measured a satisfiability checker [12] which has been developed by the authors using the set of formulae in Figure 2. The criterion for the measurement is "the parameter value of the longest formula of which satisfiability is judged within 100 seconds". The measurement results are shown in Table 2, and the environment is shown in Table 3. The criterion for measurement is adopted from the paper [1] by Balsiger et al. Sometimes satisfiability checking times against the length of formulae are plotted in a graph to show results, but drawing such graphs is not suitable for comparing two or more results. Here, 100 seconds is an acceptable time that one can wait and fixing measurement time corresponds to the postulate (6) with respect to evaluation time described in Section 2. Additionally, showing the results by parameter $n$ corresponds to the postulate (7) with respect to summarization.

# 6 Conclusion

In this paper, we proposed systematic methods to generate formulae for two-way CTL satisfiability checkers. We adopted two-way CTL as an example of temporal logics. Our methods are based on valid formulae of the form $\alpha \to \beta$ and can be applied to other logics which have Kripke structures as models.

For unsatisfiable formulae, we proposed an automatic method to obtain complicated formulae, but for satisfiable formulae, we only showed some examples. To check unsatisfiability is difficult in general, and thus unsatisfiable formulae are important for evaluation in satisfiability checking. However, to find methods to generate satisfiable formulae systematically is one of our future work.

We only showed methods to generate formulae. Our future work includes an evaluation method of satisfiability checkers using sets of parameterized formulae generated by our methods. We suppose that analyzing which characteristic is important in Table 1 is also essential for evaluating satisfiability checkers.


### Acknowledgements

This research was supported by Core Research for Evolutional Science and Technology (CREST) Program "New High-Performance Information Processing Technology Supporting Information-Oriented Society" of Japan Science and Technology Agency (JST).


## References


[1] P. Balsiger, A. Heuerding, and S. Schwendimann, "A benchmark method for the propositional modal logics K,KT,S4," J. of Automated Reasoning, vol. 24, no. 3, pp. 297–317, 2000.

[2] E.M. Clarke, O. Grumberg, and D. Peled, Model Checking, Mit Press, 2000.

[3] M.B. Dwyer, G.S. Avrunin, and J.C. Corbett, "Patterns in property specifications for finite-state verification," Proc. of 21st International Conference on Software Engineering, May, 1999.

[4] E.A. Emerson, "Temporal and modal logic," in Handbook of Theoretical Computer Science, vol. B, chap. 16, pp. 995–1072, Elsevier and MIT Press, 1990.

[5] E.A. Emerson, and E.M. Clarke, "Using branching-time temporal logic to synthesize synchronization skeletons," Science of Computer Programming, vol. 2, no. 3, pp. 241–266, 1982.

[6] M. Hagiya, K. Takahashi, M. Yamamoto, and T. Sato, "Analysis of synchronous and asynchronous cellular automata using abstraction by temporal logic," Proc. of 7th International Symposium on Functional and Logic Programming, LNCS, vol. 2998, pp. 7–21, 2004.

[7] M. Huth, and M. Ryan, Logic in Computer Science: modelling and reasoning about systems, Cambridge University Press, 2004.

[8] M. Lange, and C. Stirling, "Focus games for satisfiability and completeness of temporal logic," Proc. of 16th Annual IEEE Symposium on Logic in Computer Science, pp. 357–365, 2001.

[9] Z. Manna, and P. Wolper, "Synthesis of communicating process from temporal logic specifications," ACM Transactions on Programming Languages and Systems, vol. 6, no. 1, pp. 68–93, 1984.

[10] K. Takahashi, and M. Hagiya, "Abstraction of graph transformation using temporal formulas," Supplemental Volume of International Conference on Dependable Systems and Networks (DSN-2003), pp. W-65–W-66, 2003.

[11] Y. Tanabe, T. Takai, T. Sekizawa, and K. Takahashi, "Preconditions of properties described in CTL for statements manipulating pointers," Supplemental Volume of International Conference on Dependable Systems and Networks (DSN2005), pp.228–234, June 28-July 1, 2005.

[12] Y. Tanabe, K. Takahashi, M. Yamamoto, A. Tozawa, and M. Hagiya, "A Decision Procedure for Satisfiability of Modal Logics Implementable with BDD", Japan Society for Software Science and Technology (JSSST) Workshop on Programming and Programming Language (PPL) 2005, pp.5-16, 2005. (in Japanese)

[13] Y. Tanabe, K. Takahashi, M. Yamamoto, T. Sato, and M. Hagiya, "An Implementation of a Decision Procedure for Satisfiability of Two-way CTL Formulas Using BDD", Computer Software - JSSST Journal, Vol.22, No.3 (2005), pp.154-166, 2005. (in Japanese with English abstract)

[14] A. Tozawa, "On binary tree logic for XML and its satisfiability test," JSSST Workshop on Programming and Programming Language (PPL) 2004, 2004.

時相論理の充足可能性判定器のための論理式生成法
(in English)
(算譜科学研究速報)
発行日：2007 年 2 月 19 日
編集・発行：独立行政法人産業技術総合研究所システム検証研究センター
同連絡先：〒563-8577 大阪府池田市緑丘 1-8-31
e-mail：informatics-inquiry@m.aist.go.jp


A Method to Generate Formulae for Temporal Logic Satisfiability Checkers
(Programming Science Technical Report)
Feb. 19, 2007
Research Center for Verification and Semantics (CVS)
National Institute of Advanced Industrial Science and Technology (AIST)
1-8-31 Midorigaoka, Ikeda, Osaka, 563-8577, Japan
e-mail: informatics-inquiry@m.aist.go.jp