A Method to Generate Formulae for Temporal Logic Satisfiability Checkers

Toshifusa Sekizawa¹,², Toshinori Takai¹, Yoshinori Tanabe¹,³, and Koichi Takahashi¹

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 complex-
ity, 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 \cite{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 \cite{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 \cite{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 \cite{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 \cite{12} developed by the authors.

2 Backgrounds

Balsiger et al. proposed seven postulates \cite{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 forthcoming 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 \( \text{EXPTIME} \) [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 \( \mathcal{AP} \) be a set of atomic propositions, and \( \mathcal{Mod} \) be a set of modalities. For each modality \( a \in \mathcal{Mod} \), we assume that there exists a modality \( \overline{a} \in \mathcal{Mod} \) such that \( \overline{a} = a \). The syntax of two-way CTL is defined as:

\[
\varphi ::= p | \neg \varphi | \varphi \lor \varphi | E_A \varphi | E_A [\varphi \land \varphi] | E_A [\varphi \lor \varphi] | E_A [\varphi \land \varphi]
\]

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 \( \varphi[p_1, \ldots, p_n] \) for the formula \( \varphi \) in which \( p_1, \ldots, p_n \) are replaced by \( \psi_1, \ldots, \psi_n \), respectively. We write \( \varphi[p_1, \ldots, \psi_n, \psi_n/p_n] \) for the formula obtained by the same replacement to \( \varphi[p_1, \ldots, \psi_n] \). For the formula obtained by the replacements \( m \) times, we write \( \varphi[p_1, \ldots, \psi_n, \psi_n/p_n]^m \).

3.2 Semantics of two-way CTL

A tuple \( \mathcal{M} = (M, \{R_a \mid a \in \mathcal{Mod}\}, \lambda) \) 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) \( \lambda: \mathcal{AP} \rightarrow 2^M \). We call an element of \( M \) a state, and \( \lambda \) a labelling function. For labelling function \( \lambda: \mathcal{AP} \rightarrow 2^M \), atomic proposition \( x \in \mathcal{AP} \), and \( X \subseteq M \), we write \( \lambda(x \rightarrow X) \) 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 \( \mathcal{M}\{x \rightarrow X\} \) for \( (M, R, \lambda(x \rightarrow X)) \). In this paper, we do not require a Kripke structure to be total. For set of modalities \( A \subseteq \mathcal{Mod} \), 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 \( \text{len}(\sigma) \), and the \( i \)-th element \( (i \geq 0) \) of \( \sigma \) is denoted \( \sigma_i \). When the length of \( \sigma \) is infinite, we regard \( i < \text{len}(\sigma) \) holds for any natural number \( i \). (2) For any natural number \( i \) such that \( i + 1 < \text{len}(\sigma) \), 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 \( (a_{\text{len}(\sigma)-1}, s) \in R_a \).

For a Kripke structure \( \mathcal{M} = (M, \{R_a \mid a \in \mathcal{Mod}\}, \lambda) \), 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 \( p \in \mathcal{AP} \) 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 \( \varphi_1 \lor \varphi_2 \), \( \mathcal{M}, s \models \varphi_1 \) or \( \mathcal{M}, s \models \varphi_2 \), then \( \mathcal{M}, s \models \varphi_1 \lor \varphi_2 \).

When \( \varphi \) is of the form \( E_A \varphi' \), 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 \( \mathcal{M}, s \models E_A \varphi' \).

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

In the following, we use symbols \( \land, \neg, \leftrightarrow, \top \) and \( \bot \) with their usual meanings. Additionally, we use the following abbreviations:

- \( A_A X \varphi \overset{\text{def}}{=} \neg E_A X \neg \varphi \)
- \( A_A[\varphi_1 U \varphi_2] \overset{\text{def}}{=} \neg E_A[\neg \varphi_1 U \neg \varphi_2] \)
- \( A_A[\varphi_1 R \varphi_2] \overset{\text{def}}{=} \neg E_A[\neg \varphi_1 R \neg \varphi_2] \)
- \( A_A G \varphi \overset{\text{def}}{=} A_A[\top U \varphi] \)
- \( A_A F \varphi \overset{\text{def}}{=} A_A[\top U \varphi] \)
- \( E_A G \varphi \overset{\text{def}}{=} E_A[\top R \varphi] \)
- \( E_A F \varphi \overset{\text{def}}{=} E_A[\top U \varphi] \)

We assume that \( \text{Mod} \) includes a modality \( a \). We omit the suffix \( \{a\} \) in formulae, for example we write \( A X \varphi \) for \( A_{\{a\}} X \varphi \). For an integer \( n \geq 0 \), we write \( E X^n \varphi \) for the formula that \( \varphi \) is preceded by \( n \) \( EX \)s, i.e., \( E X^n \varphi \overset{\text{def}}{=} \varphi \) and \( E X^{n+1} \varphi \overset{\text{def}}{=} E X(E X^n \varphi) \).

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 $\phi$ if the negation operator precedes no occurrences of $x$ in $\phi$. Let $\phi$ be a formula, $x$ be an atomic proposition, $M = (M, R, \lambda)$ be a Kripke structure, and $X, X' \subseteq M$ be sets of states. If $x$ occurs positively in $\phi$ and $X \subseteq X'$, then we have the following relation:

$$[\phi]_{M_{\{x\rightarrow X'_1}}} \subseteq [\phi]_{M_{\{x\rightarrow X'_2}}}$$

(1)

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

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 $\alpha_1 \rightarrow \beta_1$ and $\alpha_0 \rightarrow \beta_0$ are valid, then so is formula $\alpha_1[\alpha_0/x] \rightarrow \beta_1[\beta_0/x]$.

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

$$[\alpha_1[\alpha_0/x]_M = [\alpha_1]_{M_{\{x\rightarrow X_1}}} \subseteq [\beta_1]_{M_{\{x\rightarrow X_2}}} = [\beta_1[\beta_0/x]]_M$$

(2)

(3)

(4)

where $X_1 = [\alpha_0]_M$ and $X_2 = [\beta_0]_M$, (3) is proved by the fact formula $\alpha_1 \rightarrow \beta_1$ is valid and (1), (4) is proved because $\alpha_0 \rightarrow \beta_0$ is valid. $\square$
Let $p$.

First we consider the case in which for any system for CTL [4]:

$$EX(\beta \leq (1 \alpha \text{lae, we need seeds of the form show some simple ways to give such formulae. To obtain parameterized formu-}}$$

Follows:

$$\chi$$ generated by $A$ satisfiability checker which knows the deduction rule can easily solve formulae

To apply the corollaries, we need a number of formulae of the form other logical systems.

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 $\alpha \rightarrow \beta$. We show some simple ways to give such formulae. To obtain parameterized formulae, we need seeds of the form $\alpha_0 \rightarrow \beta_0$, 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 $\alpha = EXx$, and $\beta = EXx$. By applying Corollary 1 to a valid formula $EXx \rightarrow EXx$, we obtain the following parameterized formula $\chi_1$:

$$\chi_1(n) = EX^n \alpha_0 \rightarrow EX^n \beta_0$$

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

$$\frac{\gamma \rightarrow \delta}{EX\gamma \rightarrow EX\delta}$$

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:

$$\chi_2(n) = \bigvee_{1 \leq i \leq n} (EX^i(\alpha_0) \rightarrow EX^n(\beta_0))$$

This complication corresponds to the complication $k_d4_p$ introduced in [1]. In our methods, any valid formula of the form $\alpha \rightarrow \beta$ can be used to construct parameterized formulae. For example, $\alpha_i = \alpha_0$, $\beta_i = \beta_0$ (i: even), $\alpha_0 = \alpha_0$, $\beta_i = \beta_0$ (i: odd), $\alpha_0 = x \land EXx$, $\beta_0 = E_{x}X^{\top}X(x)$, $\alpha_0 = x \land EXx$, and $\beta_0 = E_{x}X^{\top}X(x)$. Note that both formulæ $x \land EXx \rightarrow E_{x}X^{\top}X(x)$ and $x \land EXx \rightarrow E_{x}X^{\top}X(x)$ are valid. Consequently, the obtained parameterized formula can be written as follows:

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

where $\varphi_0 = \alpha_0$, $\psi_0 = \beta_0$, $\varphi_i = \varphi_i \land EXx \varphi_{i-1}$ (i is odd), $\varphi_i = \varphi_i \land EXx \varphi_{i-1}$ (i is even), $\psi_i = E_{x}X^{\top}X(\varphi_{i-1})$ (i is odd), $\psi_i = E_{x}X^{\top}X(\varphi_{i-1})$ (i is even), and $\alpha_0$ and $\beta_0$ are any formula such that $\alpha_0 \rightarrow \beta_0$ 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 $\alpha$’s are identical except atomic propositions and so are $\beta$’s. For example, let $\alpha_i = E[x \cup q_i]$, and $\beta_i = E[x \cup q_i]$ ($1 \leq i \leq n$), where $q_i$ is an atomic proposition and $\alpha_i \rightarrow \beta_i$ ($1 \leq i \leq n$) is valid. Then we obtain a parameterized formula $\chi_4$ by applying Corollary 1:

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

In the formula above, $\varphi_0 = \alpha_0$, $\psi_0 = \beta_0$, $\varphi_n = E[\varphi_{n-1} \cup q_n]$, and $\psi_n = E[\psi_{n-1} \cup q_n]$. 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 $A[c_0 \cup p_0] \land AG(p_0 \rightarrow AG(\neg c_0 \land x)) \rightarrow AFAG(\neg c_0 \land x)$ is valid. If we use this formula as $\alpha_i \rightarrow \beta_i$, we obtain a parameterized formula $\chi_5$: $\chi_5(n) = \varphi_n \rightarrow \psi_n$, where $\varphi_n = A[c_n \cup p_n] \land AG(p_n \rightarrow AG(\neg c_n \land x))$, and $\psi_n = AFAG(\neg c_n \land x)$. □

Example 3. We give a parameterized formula in which the depth of temporal operators is constant. We use formulae $x \land A[p_i \cup q_i]$ and $x \land (q \lor (p \land AXA[p_i \cup q_i]))$ ($1 \leq i \leq n$) for $\alpha_i$ and $\beta_i$ respectively in Corollary 1. Note that $A[p \cup q] \rightarrow q \lor (p \land AXA[p \cup q])$ is valid. Then we obtain a parameterized formula:

$$\chi_6(n) = \bigwedge_{1 \leq i \leq n} A[p_i \cup q_i] \rightarrow \bigwedge_{1 \leq i \leq n} (q \lor (p \land AXA[p_i \cup q_i]))$$

□

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 $\alpha_i = A_{a_i} \land \forall x$, and $\beta_i = A_{q_i} \land \forall x$ ($1 \leq i \leq n$). Since $\alpha_i \rightarrow \beta_i$ is valid, we can obtain a parameterized formula by the same way as in Example 2. Then we have a parameterized formula $\chi_7$: $\chi_7(n) = \varphi_n \rightarrow \psi_n$, where $\varphi_i = A_{a_i} \land \forall x$, and $\psi_i = A_{q_i} \land \forall x$ ($0 \leq i \leq n$). 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 $\alpha' = x \land E_x X \forall x$, and $\beta' = E_x X \forall E \exists x$. Since formula $x \land E_x X \forall x \rightarrow E_x X \forall E \exists x$ is valid, we can apply Corollary 2, and obtain a parameterized formula $\chi_8$:

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

where $\varphi'_0 = \alpha_0$, $\psi'_0 = \beta_0$, $\varphi'_n = E[\varphi'_{n-1} \cup (p \land E_x X \forall x)]$, and $\psi'_n = E[\psi'_{n-1} \cup E_x X \forall E \exists x]$. □

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 $\text{EX}x \rightarrow \text{EX}x$. 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, $E[\varphi \lor \psi] \rightarrow \psi \lor (\varphi \land \text{EX}E[\varphi \lor \psi])$, $A[p \lor q] \rightarrow \neg E[\neg p \lor q \land \neg \text{EG}(\neg q)]$, etc. Axioms of a formal system for CTL are also useful references [4, 8]. For example, $\text{EX}(\varphi \lor \psi) \leftrightarrow \text{EX}\varphi \lor \text{EX}\psi$, $\text{AX} \varphi \land \text{AX} \psi \rightarrow \text{AX}(\varphi \land \psi)$, etc. In the case of two-way CTL, we can easily see that $A_i \text{G} x \rightarrow A_i \text{G} x$, $x \land E_{\pi} X \top \rightarrow E_{\pi} X E_{\pi} X x$ 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 $A[\neg S \lor (P \lor \text{AG} \neg P)]$. We can use specification patterns to construct simple valid formulae. We name the formula $A[\neg S \lor (P \lor \text{AG} \neg P)]$ absence($P, S$). Then we have a non-trivial valid formula $\text{AFP} \land \text{AG} \rightarrow \text{absence}(P, S)$. 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 occur at some states, and from these states, event $c_1$ permanently holds and event $c_0$ never holds”. This property can be expressed as $\text{AFAG}(\neg c_0 \land c_1)$ in CTL. On the other hand, the models satisfying $A[c_0 \lor p_0] \land \text{AG}(p_0 \rightarrow \text{AG}(\neg c_0 \land c_1))$ also satisfy the property above. That is, $c_1$ 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 $A[c_0 \lor p_0] \land \text{AG}(p_0 \rightarrow \text{AG}(\neg c_0 \land c_1)) \rightarrow \text{AFAG}(\neg c_0 \land c_1)$.

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 $\chi_9(n) = \text{EX}^n(p)$. We show some examples.

Example 6. We show some variations of $\chi_9$. Let $\chi_{10}(n) = \bigwedge_{0 \leq i \leq n} \text{EX}^i(\alpha_0)$ where $\alpha_0$ is an arbitrary formula which does not contain reverse modalities. Another variation is: $\chi_{11}(n) = \bigwedge_{1 \leq i \leq n} (\text{EX}^{-1}(\neg \alpha_0) \land \text{EX}^i(\alpha_0))$. But in this case, we have to choose an appropriate formula $\alpha_0$. \(\square\)
Example 7. Two-way CTL does not have the finite model property. For example, if a Kripke structure satisfies the formula \( x \land A_x X A_0 G(\neg z) \land (A_y G(E_x X p \land A_y F z)) \), 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 \( s \models A_y G(E_x X p) \). 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 \( \neg \), we have \( \sigma_i \models E_x \neg z \), which contradicts \( s \models A_y G(E_x X p \land A_y F z) \). 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:
\[
\chi_{12}(0) = z_0 \land A_{n_0} X A_{\alpha_0} G(\neg z_0) \land A_{\alpha_0} G(E_{n_0} X \sigma_0) \land A_{\alpha_0} G(E_{\alpha_0} X \sigma_0)
\]

This formula can be parameterized as \( \chi_{12}(n) = \alpha_n \land \chi_{12}(n-1) \land \beta_n[\chi_{12}(n-1)/x] \) for \( n \geq 1 \) where
\[
\alpha_i = z_i \land A_{\alpha_i} X A_{\alpha_i} G(\neg z_i) \land A_{\alpha_i} G(E_{\alpha_i} X \sigma_i) \land A_{\alpha_i} G(E_{\alpha_i} X \sigma_i)
\]
\[
\beta_i = E_{\alpha_i} X(p_i) \land A_{\alpha_i} G(E_{\alpha_i} X(p_i) \land A_{\alpha_i} F(z_i) \land x), \text{ and}
\]
\[
\gamma_i = E_{\alpha_i} X(p_i) \land A_{\alpha_i} G(E_{\alpha_i} X(p_i) \land A_{\alpha_i} F(z_i) \land x).
\]
The length of the parameterized formulae increases exponentially. \( \square \)

Example 8. We consider a complication in which the number of modalities increases. Formula \( \chi_{13}(0) = A_{\alpha_1} G(\neg \alpha_0) \land A_{\alpha_2} G(\neg \alpha_0) \land A_{\alpha_3} F(\alpha_0) \) expresses that \( \alpha_0 \) is not reachable through the modality \( \alpha_1 \) only or \( \alpha_2 \) only, but it is reachable if both modalities \( \alpha_1 \) and \( \alpha_2 \) are allowed. A parameterized formula is given as follows:
\[
\chi_{13}(n) = A_{\alpha_1, \ldots, \alpha_{n-2}} G(\neg \alpha_0) \land A_{\alpha_1, \ldots, \alpha_{n-2}, \alpha_n} G(\neg \alpha_0) \land \cdots \land A_{\alpha_2, \ldots, \alpha_n} G(\neg \alpha_0) \land E_{\alpha_1, \ldots, \alpha_n} F(\alpha_0)
\]
This complication is rather simple, but it can be combined with other complications in order to obtain parameterized formulae with increasing modalities. \( \square \)

Example 9. We consider a complication in which the depth of nested temporal operators is constant. Let:
\[
\chi_{14}(0) = A_{\alpha} F \alpha_0 \land A_{\alpha} F \alpha_1 \land \neg \alpha_0 \land \neg \alpha_1 \land A_{\alpha} G(\alpha_0 \rightarrow \neg(\alpha_1 \lor E_{\alpha} X \alpha_1 \lor E_{\alpha} X \alpha_1)) \land A_{\alpha} G(\alpha_1 \rightarrow \neg(\alpha_0 \lor E_{\alpha} X \alpha_0 \lor E_{\alpha} X \alpha_0))
\]
then it is generalized into a parameterized formula:
\[
\chi_{14}(n) = \bigwedge_{1 \leq i \leq n} A_{\alpha} G(\alpha_i \rightarrow \neg(\gamma_{n,i} \lor \delta_{n,i,a} \lor \delta_{n,i,b})) \text{ where } \gamma_{i,j} = \alpha_0 \lor \cdots \lor \alpha_j \lor \cdots \lor \alpha_1 \text{ and } \delta_{i,j,m} = E_{\alpha} X \alpha_0 \lor \cdots \lor E_{\alpha} X \alpha_{j-1} \lor E_{\alpha} X \alpha_{j+1} \lor \cdots \lor E_{\alpha} X \alpha_1.
\]

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

<table>
<thead>
<tr>
<th>name</th>
<th>sat</th>
<th>AP</th>
<th>depth</th>
<th>Mod</th>
<th>inverse</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>test1</td>
<td>n</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>n</td>
<td>linear</td>
</tr>
<tr>
<td>test2</td>
<td>n</td>
<td>i</td>
<td>c</td>
<td>c</td>
<td>n</td>
<td>linear</td>
</tr>
<tr>
<td>test3</td>
<td>n</td>
<td>i</td>
<td>c</td>
<td>c</td>
<td>n</td>
<td>linear</td>
</tr>
<tr>
<td>test4</td>
<td>n</td>
<td>c</td>
<td>i</td>
<td>c</td>
<td>n</td>
<td>linear</td>
</tr>
<tr>
<td>test5</td>
<td>n</td>
<td>c</td>
<td>c</td>
<td>i</td>
<td>n</td>
<td>linear</td>
</tr>
<tr>
<td>test6</td>
<td>n</td>
<td>i</td>
<td>c</td>
<td>i</td>
<td>n</td>
<td>linear</td>
</tr>
<tr>
<td>test7</td>
<td>n</td>
<td>i</td>
<td>i</td>
<td>c</td>
<td>n</td>
<td>linear</td>
</tr>
<tr>
<td>test8</td>
<td>n</td>
<td>c</td>
<td>i</td>
<td>i</td>
<td>n</td>
<td>linear</td>
</tr>
<tr>
<td>test9</td>
<td>n</td>
<td>c</td>
<td>i</td>
<td>i</td>
<td>n</td>
<td>quadratic</td>
</tr>
<tr>
<td>test10</td>
<td>n</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>n</td>
<td>linear</td>
</tr>
<tr>
<td>test11</td>
<td>n</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>y</td>
<td>linear</td>
</tr>
<tr>
<td>test12</td>
<td>n</td>
<td>i</td>
<td>c</td>
<td>c</td>
<td>y</td>
<td>linear</td>
</tr>
<tr>
<td>test13</td>
<td>n</td>
<td>c</td>
<td>i</td>
<td>c</td>
<td>y</td>
<td>exp</td>
</tr>
<tr>
<td>test14</td>
<td>n</td>
<td>c</td>
<td>i</td>
<td>c</td>
<td>y</td>
<td>quadratic</td>
</tr>
<tr>
<td>test15</td>
<td>n</td>
<td>c</td>
<td>c</td>
<td>i</td>
<td>y</td>
<td>linear</td>
</tr>
<tr>
<td>test16</td>
<td>n</td>
<td>i</td>
<td>c</td>
<td>i</td>
<td>y</td>
<td>linear</td>
</tr>
<tr>
<td>test17</td>
<td>n</td>
<td>i</td>
<td>i</td>
<td>c</td>
<td>y</td>
<td>linear</td>
</tr>
<tr>
<td>test18</td>
<td>n</td>
<td>c</td>
<td>i</td>
<td>i</td>
<td>y</td>
<td>linear</td>
</tr>
<tr>
<td>test19</td>
<td>n</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>y</td>
<td>linear</td>
</tr>
<tr>
<td>test20</td>
<td>y</td>
<td>c</td>
<td>i</td>
<td>c</td>
<td>n</td>
<td>quadratic</td>
</tr>
<tr>
<td>test21</td>
<td>y</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>y</td>
<td>exp</td>
</tr>
<tr>
<td>test22</td>
<td>y</td>
<td>c</td>
<td>c</td>
<td>i</td>
<td>n</td>
<td>linear</td>
</tr>
<tr>
<td>test23</td>
<td>y</td>
<td>i</td>
<td>c</td>
<td>c</td>
<td>y</td>
<td>linear</td>
</tr>
<tr>
<td>test24</td>
<td>y</td>
<td>c</td>
<td>i</td>
<td>i</td>
<td>n</td>
<td>quadratic</td>
</tr>
</tbody>
</table>

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 $\neg(p \land q \rightarrow p)$. 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
test1(n) = ¬(\(\bigwedge_{0 \leq i \leq n} (p_i \rightarrow \bigwedge_{0 \leq i \leq n} (p_i))\) \((n \geq 0)\))
test2(0) = ¬(AFp \land A\neg(S) \rightarrow \text{absence}(P, S))
test3(n) = ¬((\exists X(u \lor v) \land \bigwedge_{1 \leq i \leq n} A[p_i, U q]) \rightarrow (\exists Xu \lor EXv \land \bigwedge_{1 \leq i \leq n} (q \lor AXA[p_i, U q])))
test4(n) = ¬(\(\chi_1(n) \geq 1\), ¬(p \land q \rightarrow p) \((n = 0)\))
test5(0) = ¬((p \lor E_0 Xp) \rightarrow E_0 Xp) \((n = 0)\))
test5(n) = ¬(\(\varphi_n \rightarrow \psi_n\) \((n \geq 1)\), \(\varphi_n = (p \land E_0 Xp) \land \varphi_{n-1}, \psi_n = E_n Xp\))
test6(n) = ¬((E_n X(u \lor v) \land \bigwedge_{1 \leq i \leq n} A_n[p_i, U q]) \rightarrow (E_n Xu \lor EXn Xu \land \bigwedge_{1 \leq i \leq n} (q \lor A_n XA_n[p_i, U q])))
test7(n) = ¬(\(\chi_3(n)\) \((n \geq 0)\))
test8(0) = ¬(\(\chi_3(n)\) \((n \geq 0)\), ¬(E[p \lor q] \rightarrow EFq) \((n = 0)\))
test9(0) = ¬(E[p \lor q] \rightarrow EFq)
test10(n) = ¬(\(\varphi_n \rightarrow \psi_n\) \((n \geq 1)\), \(\varphi_i = E_n F \varphi_{i-1}, \psi_i = E_n F \psi_{i-1}\))
test11(n) = ¬((\bigwedge_{0 \leq i \leq n} (p \rightarrow E_n XE_p Xp)) \((n \geq 0)\))
test12(n) = ¬((\bigvee_{0 \leq i \leq n} (p \land E_0 Gq_n)) \rightarrow (\bigvee_{0 \leq i \leq n} (q_n \rightarrow E_0 Fp)) \((n \geq 0)\))
test13(n) = ¬(\(\chi_3(n)\) \((n \geq 1)\), ¬(A_n \land Gp \rightarrow A_n Gp) \((n = 0)\))
test14(n) = ¬(AXP \land AXq \rightarrow AX(p \land q))
test15(n) = ¬(\(\chi_4(n)\) \((n \geq 1)\), ¬(\(\chi_5(n)\) \((n \geq 1)\)))
test16(n) = ¬((E_n X(u \lor v) \land \bigwedge_{1 \leq i \leq n} (p \land E_n Xp)) \rightarrow (E_n Xu \lor EXn Xu \land \bigwedge_{1 \leq i \leq n} E_n XE_p Xp))
test17(n) = ¬((\bigwedge_{0 \leq i \leq n} (q_i \land \bigwedge_{0 \leq i \leq n} E_n XE_p Xp))
test18(n) = ¬((\bigwedge_{0 \leq i \leq n} (q_i \land \bigwedge_{0 \leq i \leq n} E_n XE_p Xp))
test19(n) = ¬((\bigwedge_{0 \leq i \leq n} (q_i \land \bigwedge_{0 \leq i \leq n} E_n XE_p Xp))
test20(n) = \chi_{11}(n) \((n \geq 1)\), p \land q \((n = 0)\))
test21(n) = \chi_{12}(n) \((n \geq 0)\))
test22(0) = A_n G(E_n X(p \land q) \land E_n X(\neg q \land p)) \land A_n F \neg p
ntest23(n) = \chi_{13}(n) \((n \geq 1)\)
test24(n) = \chi_{13}(n) [\chi_{11}(n) / \alpha_n] \((n \geq 1)\), \(\chi_{11}(0)\) \((n = 0)\))
Table 2: Measurements results of the maximum parameter by a satisfiability checker

<table>
<thead>
<tr>
<th>test1</th>
<th>&gt;20</th>
<th>test2</th>
<th>7</th>
<th>test3</th>
<th>11</th>
<th>test4</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>test5</td>
<td>&gt;20</td>
<td>test6</td>
<td>7</td>
<td>test7</td>
<td>3</td>
<td>test8</td>
<td>15</td>
</tr>
<tr>
<td>test9</td>
<td>14</td>
<td>test10</td>
<td>3</td>
<td>test11</td>
<td>&gt;20</td>
<td>test12</td>
<td>&gt;20</td>
</tr>
<tr>
<td>test13</td>
<td>14</td>
<td>test14</td>
<td>&gt;20</td>
<td>test15</td>
<td>&gt;20</td>
<td>test16</td>
<td>13</td>
</tr>
<tr>
<td>test17</td>
<td>4</td>
<td>test18</td>
<td>&gt;20</td>
<td>test19</td>
<td>&gt;20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>test20</td>
<td>&gt;20</td>
<td>test21</td>
<td>3</td>
<td>test22</td>
<td>&gt;20</td>
<td>test23</td>
<td>6</td>
</tr>
<tr>
<td>test24</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Environment.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Red Hat Enterprise Linux ES3</td>
</tr>
<tr>
<td>CPU</td>
<td>Intel Xeon 3.0GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>4GB</td>
</tr>
<tr>
<td>JVM</td>
<td>Java2 1.5.0.03</td>
</tr>
</tbody>
</table>

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 \( \alpha \rightarrow \beta \) 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 Evoloutional Science and Technology (CREST) Program “New High-Performance Information Processing Technology Supporting Information-Oriented Society” of Japan Science and Technology Agency (JST).

References


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
本掲載記事の無断転載を禁じます

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
・Reproduction in whole or in part without written permission is prohibited.

Systems and Computers in Japan © Copyright 2007 Wiley Periodicals, Inc.