IEICE TRANS. INF. & SYST., VOL.E92–D, NO.6 JUNE 2009

PAPER

Feature Interaction Verification Using Unbounded Model Checking with Interpolation

Takafumi MATSUO[†a)], ***Student Member***, **Tatsuhiro TSUCHIYA**[†b)], ***Member***, ***and*** **Tohru KIKUNO**[†c)], ***Fellow***

SUMMARY In this paper, we propose an unbounded model checking method for feature interaction verification for telecommunication systems. Unbounded model checking is a SAT-based verification method and has attracted recent attention as a powerful approach. The interpolation-based approach is one of the most promising unbounded model checking methods and has been proven to be effective for hardware verification. However, the application of unbounded model checking to asynchronous systems, such as telecommunication systems, has rarely been practiced. This is because, with the conventional encoding, the behavior of an asynchronous system can only be represented as a large propositional formula, thus resulting in large computational cost. To overcome this problem we propose to use a new scheme for encoding the behavior of the system and adapt the unbounded model checking algorithm to this encoding. By exploiting the concurrency of an asynchronous system, this encoding scheme allows a very concise formula to represent system's behavior. To demonstrate the effectiveness of our approach, we conduct experiments where 21 pairs of telecommunication services are verified using several methods including ours. The results show that our approach exhibits significant speed-up over unbounded model checking using the traditional encoding.

key words: *unbounded model checking, interpolation, feature interaction, verification*

1. Introduction

Model checking [1] is a well-known formal approach for verifying systems that are modeled as state machines. For realistic systems, however, the number of states of the system model can be very large, making the model checking problem intractable. This problem is called the state explosion problem.

Bounded model checking [2], [3] is one of the successful solutions to this problem. The main idea of bounded model checking is to look for counterexamples that are shorter than some fixed length k for a given property. This limitation allows one to reduce the model checking problem to the satisfiability (SAT) checking problem for a formula of some logic such that its satisfiability implies the existence of a counterexample. Thus if the formula turns out to be satisfiable, then it is possible to conclude that the violation of the property occurs in the system.

Although effective in detecting property violation, bounded model checking cannot be directly used for proving the absence of violation. To cope with this disadvantage, McMillan proposed *unbounded model checking* [4], which combines bounded model checking and *interpolation*.

The key observation used in his method is that when bounded model checking fails to find a counterexample, in which case the formula is unsatisfiable, an over-approximation of the state set reachable in one step can be derived from the unsatisfiability proof produced by the SAT solver. Technically this over-approximation is obtained in the form of an *interpolant* of the tested formula, using the interpolation procedure. By repeatedly executing the interpolant procedure, an over-approximation of the reachable state set can be obtained. If this over-approximation contains no state violating a given property, then it is ensured that the system meets that property.

However, the application of unbounded model checking to asynchronous software systems has rarely been practiced. Indeed we are not aware of any application to telecommunication systems. This can be explained by the fact that with the conventional encoding, the behavior of an asynchronous system can only be represented as a large formula, thus resulting in large computational cost.

To overcome this problem, in this paper, we propose an interpolation-based unbounded model checking method that can be used for the verification of feature interactions in telecommunication services. In our method, we use a new scheme for encoding the behavior of the system. By exploiting the concurrency of the telecommunication system, this encoding scheme allows a very concise representation of system's behavior. By adapting unbounded model checking to this encoding, we obtain our model checking method. The effectiveness of our approach is demonstrated through experiments.

Feature interaction also occurs in more complex systems, such as building control systems and home network systems where the state explosion problem becomes more serious. The techniques for treating feature interaction problem in these systems have been developed based on those for treating feature interactions in telecommunication systems. Hence, this paper shows the applicability of unbounded model checking to such more complex systems.

Previous attempts to improve the performance of unbounded model checking include, for example, [5]–[7]. In [5], the method of reusing interpolants is proposed to efficiently obtain an over-approximation of the reachable state set. In [6], hybridization of interpolation and abstraction refinement is studied. In [7], a new interpolation algorithm

Manuscript received November 4, 2008.
Manuscript revised February 16, 2009.

[†]The authors are with the Graduate School of Information Science and Technology, Osaka University, Suita-shi, 565–0871 Japan.
a) E-mail: t-matuo@ist.osaka-u.ac.jp
b) E-mail: t-tutiya@ist.osaka-u.ac.jp
c) E-mail: kikuno@ist.osaka-u.ac.jp

DOI: 10.1587/transinf.E92.D.1250

which is based on linear programming is proposed. These studies aim to improve the interpolation procedure but do not focus on the representation of the behavior of the system. The central idea behind our encoding can also be seen in our earlier work [8], [9]. In [9] a similar encoding is proposed in the context of the verification of safe Petri nets. The encoding proposed in [8] is used to represent telecommunication systems, as is done in this paper. However transition ordering, which will be explained in Sect. 3.2.1 is not applied in the encoding of [8]. More importantly, in contrast to this paper where unbounded model checking is discussed, these early attempts only deal with bounded model checking.

2. Telecommunication Services

2.1 Examples of Services

In this paper we consider seven telecommunication services taken from ITU-U recommendation [10] and Bellcore's feature standard [11].

Call Waiting (CW): This service allows the subscriber to receive a second incoming call while he or she is already talking.

Call Forwarding (CF): This service allows the subscriber to have his or her incoming calls forwarded to another address.

Originating Call Screening (OCS): This service allows the subscriber to specify that outgoing calls be either restricted or allowed according to a screening list.

Terminating Call Screening (TCS): This service allows the subscriber to specify that incoming calls be either restricted or allowed according to a screening list.

Denied Origination (DO): This service allows the subscriber to disable any call originating from the terminal. Only terminating calls are permitted.

Denied Termination (DT): This service allows the subscriber to disable any call terminating at the terminal. Only originating calls are permitted.

Directed Connect (DC): This service is a so-called hot line service. Suppose that x subscribes to DC and that x specifies y as the destination address. Then, by only off-hooking, x is directly calling y. It is not necessary for x to dial y.

2.2 Feature Interaction

Two types of feature interaction are considered. The freedom from these types of interaction can be viewed as safety properties. In order to detect these types of interactions, it suffices to check the reachability the initial state to undesirable states where feature interaction occurs.

2.2.1 Invariant Violation

It is usually the case that services require some specific properties to be satisfied at any time. For example, the OCS service requires that if x specifies y in the screening list, then x is never calling y at any time. Such a property is generally referred to as an *invariant*. However, combining multiple services can result in violation of this property. Consider a situation where user A has subscribed to OCS service and specified user B in the screening list while user C has activated CF service to B. In this situation, if A calls C, the call is forwarded to B by the CF service. As a result, the invariant property of OCS is violated.

2.2.2 Nondeterminism

Nondeterminism is one of the best known types of feature interactions [12], [13]. Nondeterminism refers to a situation where a single event can simultaneously activate two or more functionalities of different services, and as a result, it cannot be determined exactly which functionality should be activated. For example, this type of interaction can occur between the CW service and the CF service. Suppose that user A subscribes both services. Now consider the situation where A is talking with user B while user C is in A's forwarding address list. If user D dials A, then either the call from D to A may be received by A because of CW, or the call may be forwarded to C because of CF.

2.3 System Model

We use State Transition Rules (STR) [14] to describe services and to model the behavior of the system. A *service* is defined as a 6-tuple $\langle U, V, P, E, R, s_{init}\rangle$, where U is a finite set of service users, V is a finite set of variables, P is a set of predicates, E is a finite set of events, R is a finite set of rules, and s_{init} is the initial state. A predicate $p \in P$ is of the form $p(x_1, x_2, \dots)$ where $x_i \in V$. An event $e \in E$ is of the form $e(x_1, x_2, \dots)$ where $x_i \in V$. A rule $r \in R$ is of the form:

$$r : \textit{pre-condition}\ [\textit{event}]\ \textit{post-condition}$$

The pre-condition is a set of predicates or negations of predicates, or both, while the post-condition is a set of predicates.

Figure 1 shows an example of a service specification expressed in STR. This specification describes the Plain Old Telephone Service (POTS). Additional communication features can be described by modifying this specification (for example, by adding new rules).

A predicate (or an event or a rule) is *instantiated* by substituting a user $a \in U$ for each variable $x \in V$ occurring in the predicate (event or rule, respectively) such that no two variables are substituted by the same user. That is, given a predicate $p(x_1, x_2, \dots,) \in P$ and a substitution $\theta = \langle x_1|a_1, x_2|a_2, \dots\rangle$, $\forall i, j, i \neq j : a_i \neq a_j$, we have a predicate instance $p(a_1, a_2, \dots)$. An event instance or a rule instance is defined similarly. We let $\mathcal{P} = \{p_1, \dots, p_m\}$ denote the set of all predicate instances and m denote the number of the predicate instances (i.e. $m = |\mathcal{P}|$). Also we denote by $\mathcal{R} = \{t_1, t_2, \cdots, t_n\}$ the set of all rule instances and by n the number of the rule instances (i.e. $n = |\mathcal{R}|$).

A state is defined as a set of predicate instances and is

$U = \{A, B\}$
$V = \{x, y\}$
$P = \{idle(x), dialtone(x), busytone(x),$
$calling(x, y), path(x, y)\}$
$E = \{onhook(x), offhook(x), dial(x)\}$
$R = \{$
$pots1 : \{idle(x)\}[offhook(x)]\{dialtone(x)\}.$
$pots2 : \{dialtone(x)\}[onhook(x)]\{idle(x)\}.$
$pots3 : \{dialtone(x), idle(y)\}[dial(x,y)]\{calling(x,y)\}.$
$pots4 : \{dialtone(x), \neg idle(y)\}[dial(x,y)]\{busytone(x)\}.$
$pots5 : \{calling(x,y)\}[onhook(x)]\{idle(x), idle(y)\}.$
$pots6 : \{calling(x,y)\}[offhook(y)]\{path(x,y), path(y,x)\}.$
$pots7 : \{path(x,y), path(y,x)\}[onhook(x)]\{idle(x), busytone(y)\}.$
$pots8 : \{busytone(x)\}[onhook(x)]\{idle(x)\}.$
$pots9 : \{dialtone(x)\}[dial(x,x)]\{busytone(x)\}.$
$\}$
$s_{init} = \{idle(A), idle(B)\}$

Fig. 1 Rule-based specification for POTS.

regarded as representing the predicate instances that hold in that state. We denote by S the set of states, that is, $S = 2^{\mathcal{P}}$.

The execution semantics is as follows. For a rule instance $t \in \mathcal{R}$, let $Pre[t]$ denote the set of predicate instances in the pre-condition of t and $\hat{P}re[t]$ denote the set of predicate instances whose negations are in the pre-condition. Also let $Post[t]$ denote the set of the predicate instances in the post-condition of t and $e[t]$ denote the event instance of t. t is *enabled* for $e[t]$ in a state s iff all instances in $Pre[t]$ hold and no instance in $hatPre[t]$ hold in s; that is, $Pre[t] \subseteq s$ and $\hat{P}re[t] \cap s = \emptyset$. Exactly one enabled rule instance is selected for execution at a time. The execution of an enabled rule t causes a state transition from s to the next state s', by deleting all instances in $Pre[t]$ from s and adding all instances in $Post[t]$; that is $s' = (s \setminus Pre[t]) \cup Post[t]$.

Example 1: Consider the specification of POTS in Fig. 1. Let t be the rule instance of $r = pots1$ based on $\theta = \langle x|A\rangle$. Then $Pre[t] = \{idle(A)\}$, $\hat{P}re[t] = \emptyset$, and $Post[t] = \{dialtone(A)\}$. In state $s = \{idle(A), idle(B)\}$ t is enabled for event $offhook(A)$ — the event that subscriber A picks up the phone. If t is executed in s, then a transition to state $s' = \{dialtone(A), idle(B)\}$ occurs.

In general a state transition system is represented as (S, T, I) where S is the set of states, $T \subseteq S \times S$ is the transition relation, and $I \subseteq S$ is the set of initial states. Now, for each rule instance t, let $T_t \subseteq S \times S$ be the relation over states such that $(s, s') \in T_t$ iff t is enabled in s for some event instance and its execution causes a state transition to s'. The state transition system (S, T, I) defined by an STR specification $SS = \langle U, V, P, E, R, s_{init}\rangle$ is such that $T = \bigcup_{t\in\mathcal{R}} T_t$ and $I = \{s_{init}\}$. We denote by $s \xrightarrow{t} s'$ iff $(s, s') \in T_t$.

3. Proposed Method

3.1 Symbolic Representation

We propose a propositional SAT-based unbounded model checking method. In order to use propositional SAT solvers for model checking, it is essential to encode the state space and the transition relation with Boolean variables.

Recall that $\mathcal{P} = \{p_1, \ldots, p_m\}$ is the set of predicate instances. A state can be represented as a Boolean m-vector such that it has a *true* in the ith position iff p_i holds in that state. In the following of the paper, we represent states with m Boolean variables $s = (b_1, \ldots, b_m)$; that is, a state is a truth assignment of these m variables.

Any set $S_0 \subseteq S$ of states can be represented as a Boolean function $f : \{true, false\}^m \to \{true, false\}$ such that:

$$f(s) = \begin{cases} true & s \in S_0 \\ false & otherwise \end{cases}$$

For example, the state set where the pre-condition of a rule instance t holds is represented as:

$$E_t(s) = \bigwedge_{p_i \in Pre[t]} b_i \wedge \bigwedge_{p_i \in \hat{P}re[t]} \neg b_i$$

The relation over states is also represented as a Boolean function with $2m$ Boolean variables, since the relation is simply a set of state pairs. We therefore identify a set of states or of transitions with its corresponding Boolean function. For example, T_t is represented as:

$$T_t(s, s') = E_t(s) \wedge \bigwedge_{p_i \in Post[t]} b'_i \wedge \bigwedge_{p_i \in Pre[t] \setminus Post[t]} \neg b'_i \wedge \bigwedge_{p_i \in \mathcal{P} \setminus (Pre[t] \cup Post[t])} (b_i \leftrightarrow b'_i)$$

where $s = (b_1, \cdots, b_m)$ and $s' = (b'_1, \cdots b'_m)$.

Example 2: For simplicity, we use the name of a predicate instance to denote its corresponding Boolean variable. Let t be the instance of the rule $pots1$ in Fig. 1 with substitution $\theta = \langle x|A\rangle$. Then we have $T_t(s, s') = idle(A) \wedge dialtone(A)' \wedge \neg idle(A)' \wedge (idle(B) \leftrightarrow idle(B)') \wedge (dialtone(B) \leftrightarrow dialtone(B)') \wedge (busytone(A) \leftrightarrow busytone(A)') \wedge (busytone(B) \leftrightarrow busytone(B)') \wedge (calling(A, B) \leftrightarrow calling(A, B)') \wedge (calling(B, A) \leftrightarrow calling(B, A)') \wedge (path(A, B) \leftrightarrow path(A, B)') \wedge (path(B, A) \leftrightarrow path(B, A)')$.

The transition relation T is represented as:

$$T(s, s') = \bigvee_{t\in\mathcal{R}} T_t(s, s')$$

For simplicity, assume that we know that T is a *total* relation [1]. Then whether state set G is reachable from the

initial state in k steps can be determined by checking the satisfiability of the following formula:

$$I(s_0) \wedge T(s_0, s_1) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge (G(s_0) \vee \cdots \vee G(s_k))$$

This is the basic formula used in SAT-based model checking.

3.2 Boolean Encoding of Telecommunication Systems

The obstacle to applying SAT-based model checking to asynchronous systems like telecommunication systems is that the transition relation T for such a system can only be represented as a large formula. This can be understood by seeing Example 2: To represent the transitions for each rule instance t, T_t must contain conjuncts $(b_i \leftrightarrow b_i')$ for all Boolean variables b_i that represent predicate instances not engaged in that rule. The behavior of a telecommunication system is represented as many rule instance. The execution of each rule instance only changes the value of a small number of predicate instance. Hence, the transition relation T consists largely of formulas that represent the value of predicate instance does not change.

Our encoding overcomes this disadvantage. The intuitive idea is as follows: We introduce a new semantics for system execution that maintains safety properties of the original model. In this semantics the n rule instances are totally ordered and a step is represented as a sequence of n "micro" steps. The ith micro step is either the state transition by the ith rule instance or a stuttering step. Because only two state transitions are possible at each micro step, this semantics can avoid a blow-up in the formula size, which is inherent to symbolic representation of asynchronous systems.

Let $Chng[t]$ denote the set of predicate instances that change their truth value as a result of the execution of rule instance t; that is, $Chng[t] = (Post[t] \setminus Pre[t]) \cup (Pre[t] \setminus Post[t])$. Our encoding can avoid generating a large subformula related to $\mathcal{P} \setminus Chng[t]$.

Now let $d_t(s, s')$ be defined as follows:

$$\begin{aligned} d_t(s,s') &= T_t(s,s') \vee \bigwedge_{p_i \in \mathcal{P}} (b_i \leftrightarrow b_i') \\ &= \Big(\Big(\bigwedge_{p_i \in Pre[t]} b_i \wedge \bigwedge_{p_i \in \hat{P}re[t]} \neg b_i \\ &\quad \wedge \bigwedge_{p_i \in Post[t] \setminus Pre[t]} b_i' \wedge \bigwedge_{p_i \in Pre[t] \setminus Post[t]} \neg b_i' \Big) \\ &\quad \vee \bigwedge_{p_i \in Chng[t]} (b_i \leftrightarrow b_i') \Big) \\ &\quad \wedge \bigwedge_{p_i \in \mathcal{P} \setminus Chng[t]} (b_i \leftrightarrow b_i') \end{aligned}$$

Example 3: Let t be the rule instance of *pots1* in Fig. 1 with substitution $\theta = \langle x|A \rangle$. Then $Chng[t] = \{idle(A), dialtone(A)\}$ and thus we have $d_t(s, s') = \Big((idle(A) \wedge dialtone(A)' \wedge \neg idle(A)') \vee ((idle(A) \leftrightarrow idle(A)') \wedge (dialtone(A) \leftrightarrow dialtone(A)') \Big) \wedge (idle(B) \leftrightarrow idle(B)') \wedge (dialtone(B) \leftrightarrow dialtone(B)') \wedge (busytone(A) \leftrightarrow busytone(A)') \wedge (busytone(B) \leftrightarrow busytone(B)') \wedge (calling(A, B) \leftrightarrow calling(A, B)') \wedge (calling(B, A) \leftrightarrow calling(B, A)') \wedge (path(A, B) \leftrightarrow path(A, B)') \wedge (path(B, A) \leftrightarrow path(B, A)')$.

By definition $d_t(s, s') = true$ iff $s \xrightarrow{t} s'$ or $s = s'$. Using this property, a step (or more) can be represented by a conjunction of $d_t(s, s')$ as follows:

$$D(s_0, \cdots, s_n) = \bigwedge_{1 \le i \le n} d_{t_i}(s_{i-1}, s_i)$$

$D(s_0, \cdots, s_n)$ evaluates to true iff any $1 \le i \le n$, $s_{i-1} \xrightarrow{t_i} s_i$ or $s_{i-1} = s_i$. This means that if this function evaluates to true, s_n is reachable from s_0 in at most n steps (including 0 steps), and that if there is at least one t_i such that $s \xrightarrow{t_i} s'$, $D(s_0, \cdots, s_n)$ evaluates to true under an assignment such that $s = s_0 = \cdots = s_{i-1}$, $s_{i-1} \xrightarrow{t_i} s_i$, and $s_i = \cdots = s_n = s'$.

Consequently, the following formula BMC^k can be used for the verification.

$$BMC^k = I(s_0) \wedge \bigwedge_{0 \le j < k} D(s_{j*n}, \cdots, s_{(j+1)*n}) \wedge G(s_{k*n})$$

If BMC^k is satisfiable, then some state in G is reachable from the initial state in at most $k * n$ steps. If BMC^k is unsatisfiable, then no state in G can be reached from the initial state in k steps.

A major benefit of using this formula is that it can be shortened to a considerable extent and thus in turn the run time of SAT solving can be reduced. The idea is as follows. Note that in BMC^k, term $(b_i \leftrightarrow b_i')$ for any $p_i \in \mathcal{P} \setminus Chng[t]$ occurs as a conjunct (See the definition of $d_t(s, s')$). Replacing b_i' with b_i, such a term can safely be removed from BMC^k without altering the satisfiability, because BMC^k is satisfiable only if b_i and b_i' have the same truth value.

The effect of this optimization is significant, since for practical telecommunication services, a rule execution affects only a small fraction of the predicate instances. Compared to the conventional formula shown in Sect. 3.1, a reduction of around 60 to 90 percent in the number of literal occurrences has typically been observed in the examples tested in Sect. 4.

Remark 1: For presentation purpose we explain our model checking method using the original BMC^k; but this optimization is always used in the implementation.

3.2.1 Transition Ordering

In practice, the state transitions represented by D critically depend on the order of rule instances. This can be intuitively explained as follows: Consider two rule instances t_i and t_j and suppose that the execution of t_i cause the precondition of t_j to hold but not vice versa. In this case, if t_i occurs before t_j

```
 1: set Checked := ∅ ; {Keeps predicate instances.}
 2: queue Done := ∅ ; {Keeps ordered rule instances.}
 3: for all p ∈ s_init do
 4:     call CHECK(p);
 5: end for
 6:
 7: Procedure: CHECK(p)
 8:     add p to Checked;
 9:     for all t such that p ∈ Pre[t] do
10:         if t ∉ Done and ∀p' ∈ Pre[t][p' ∈ Checked] then
11:             enqueue t to Done;
12:             for all p' such that p' ∈ Post[t] and p' ∉ Checked do
13:                 call CHECK(p');
14:             end for
15:         end if
16:     end for
17: end Procedure
```

Fig. 2 Algorithm for ordering transitions.

in D, then D can represent the successive execution of t_i and t_j. On the other hand, if t_i comes after t_j, then D can only express the execution of only one of the two rule instances.

We propose a heuristic algorithm for transition ordering, by extending the one in [9], which is proposed in the context of safe Petri nets. The basic idea is to select a rule instance t when each predicate instance in $Pre[t]$ occurs in s_{init} or in the post-condition of an already selected rule instance. Figure 2 shows the algorithm.

The FIFO queue *Done* is used for storing rule instances that have already been ordered. The set *Checked* of predicate instances is used to maintain those occurring in the initial state or in the post-condition of rule instances ordered already. The main part of the algorithm calls procedure CHECK with each predicate instance occurring in the initial state. In the procedure, first p is added to *Checked*. Then, for each rule instance t such that $p \in Pre[t]$, the following is done: If t has not yet been ordered and all predicate instances in $Pre[t]$ have been stored in *Checked*, then t is enqueued and the procedure is recursively called with each predicate instance in $Post[t]$. Since a rule instance t is ordered only after all predicate instances in $Pre[t]$ are stored in *Checked*, for any predicate instance p in $Pre[t]$, it occurs in the initial state or there must be at least one already ordered rule instance t' such that $p \in Post[t']$.

In case a given service specification is ill-formed, this algorithm may fail to order all rule instances. It is easy to show that in such a case, the rule instances not selected for ordering are always unenabled, and thus they can safely be omitted.

Example 4: Consider the service specification in Fig. 1 and let $\theta 1 = \langle x|A, y|B\rangle$ and $\theta 2 = \langle x|B, y|A\rangle$. The algorithm orders a total of 18 rule instances as follows: $(t_1, \cdots, t_{18}) = (pots1\theta 1, pots1\theta 2, pots2\theta 1, pots2\theta 2, pots3\theta 1, pots3\theta 2, pots4\theta 1, pots4\theta 2, pots5\theta 1, pots5\theta 2, pots6\theta 1, pots6\theta 2, pots7\theta 1, pots7\theta 2, pots8\theta 1, pots8\theta 2, pots9\theta 1, pots9\theta 2)$, where $r\theta$ denotes the instance of a rule r with a substitution θ. With this ordering, BMC^1 allows reachability checking for 12 states, including state $\{path(A, B), path(B, A)\}$ which requires $t_1 = pots1\theta 1$, $t_5 = pots3\theta 1$, and $t_{11} = pots6\theta 1$ to occur in this order to be reached. On the other hand, if the rule instances were reversely ordered, the efficiency would be much deteriorated. In this case BMC^1 can check the reachability of only four states: $\{idle(A), idle(B)\}$, $\{dialtone(A), idle(B)\}$, $\{idle(A), dialtone(B)\}$, and $\{dialtone(A), dialtone(B)\}$.

3.3 Unbounded Model Checking

3.3.1 State Exploration Using Interpolants

For two first-order logic formulas A and B, if $A \wedge B$ is unsatisfiable, then the interpolant P for A and B is a formula with the following properties:

- $A \rightarrow P$,
- $P \wedge B$ is unsatisfiable, and
- P refers only to the common variables of A and B.

Several interpolation methods have been proposed, including [4], [7], [15]. Using an interpolation method with a SAT solver, one can simultaneously perform satisfiability checking and, if the formula is unsatisfiable, interpolant generation.

We divide BMC^k into *PREF* and $SUFF^k$ as follows:

$$PREF = I(s_0) \wedge D(s_0, \cdots, s_n)$$
$$SUFF^k = \bigwedge_{1 \le i < k} D(s_{i*n}, \cdots, s_{(i+1)*n}) \wedge G(s_{k*n})$$

If $PREF \wedge SUFF^k$ is satisfiable, then the system can reach a state in G. On the other hand, if $PREF \wedge SUFF^k$ is unsatisfiable, then an interpolant of *PREF* and $SUFF^k$ can be generated.

The interpolant refers only to the common variables of *PREF* and $SUFF^k$. Hence, it is a formula over state s_n. We denote by $Interpolant(s_n)$ this interpolant and by $Interpolant(s_n)\langle s_n|s\rangle$ the formula obtained from $Interpolant(s_n)$ by replacing Boolean variables for s_n with those for s.

Since the interpolant is implied by *PREF*, it follows that the $Interpolant(s_n)\langle s_n|s\rangle$ is true in the initial state and in every state reachable from the initial state in one step. In other words, $Interpolant(s_n)\langle s_n|s\rangle$ is an over-approximation of the state set reachable from the initial state within one step.

In effect this interpolant usually contains more reachable states than those reachable in one step, because D can represent, in addition to all single steps, up to n consecutive steps. This property contributes to effective state exploration of our method.

3.3.2 Overview of the Algorithm

The overview of the algorithm is shown in Fig. 3. The input of the algorithm is a service specification $SS = \langle U, V, P, E, R, s_{init}\rangle$ and the state set G whose reachability is

```
1: Construct M = ⟨I, D, G⟩ from SS and G;
2: while true do
3:     k := 2;
4:     FiniteRun(M, k);
5:     if aborted then
6:         k := k + 1;
7:     else if true is returned then
8:         return Reachable;
9:     else if false then
10:        return Unreachable;
11:    end if
12: end while
```

Fig. 3 Unbounded Model Checking for given (SS, G).

```
1: Function: FiniteRun(M = ⟨I, D, G⟩, k > 1)
2:     R := I;
3:     while true do
4:         M' := ⟨R, D, G⟩;
5:         Generate PREF, SUFF^k from M'
6:         Run SAT on PREF ∧ SUFF^k;
7:         if satisfiable then
8:             if R = I then
9:                 return true; {Can be reached only in the 1st iteration.}
10:            else
11:                abort;
12:            end if
13:        else
14:            Generate interpolant(Interpolant(s_n))
                   of PREF, SUFF^k;
15:            R' := Interpolant(s_n)⟨s_n/s_0⟩;
16:            if R' = R then
17:                return false;
18:            else
19:                R := R';
20:            end if
21:        end if
22:    end while
23: end Function
```

Fig. 4 Function for given (M, k).

to be verified.

First, we build the symbolic representations of the transition system $M = \langle I, D, G\rangle$ from the given service specification $SS = \langle U, V, P, E, R, s_{init}\rangle$.

The function FiniteRun is at the heart of the algorithm. It has two arguments M and k. The function returns true if it determines, by performing SAT solving for BMC^k, that a state in G is reachable and returns false if it determines that no state in G is reachable. In these cases, the algorithm can terminate simply by returning Reachable or Unreachable, according to the result of FiniteRun. FiniteRun aborts if it is impossible to determine whether or not G is reachable by using given k. In this case, the algorithm increases k and calls FiniteRun again.

3.3.3 The Function FiniteRun

The function FiniteRun is shown in Fig. 4. The basic design of this function follows that of [4] but is adapted to subtle but important differences in the encoding of system behaviors, elaborated in Sect. 3.2.

In this function, D is used instead of T which is used in [4] for representing the transitions of the system. This results in the following advantages. First, a concise formula of $PREF \wedge SUFF^k$ can be obtained. This results in speed-up of the satisfiability checking of $PREF \wedge SUFF^k$. Second, an interpolant which represents more reachable states than that of [4] can be obtained. This is because D can represent not only all single steps but also at most n consecutive steps. This contributes to effective state exploration.

This function first checks the satisfiability of $PREF \wedge SUFF^k$. If $PREF \wedge SUFF^k$ is satisfiable, then at least one of the states in G is reachable from the initial state (line 9). Hence, this function returns true. On the other hand, if $PREF \wedge SUFF^k$ is unsatisfiable, then state exploration is performed by repeatedly computing interpolants.

In the function R is used to represent the set of explored states. Initially R only represents the initial state. In each iteration of the while loop, R is updated to the interpolant for $PREF$ and $SUFF^k$ (line 19) and $PREF$ is updated with I being replaced with R (line 5). At the end of the ith execution of the while loop, R represents an over-approximation of a set of states that are reachable within i steps.

As discussed in the appendix in detail, the iteration of the while loop eventually terminates (or aborts) in either of two ways. The first case is where $PREF \wedge SUFF^k$ turns out to be satisfiable. Then the function aborts (line 11), since the satisfiability of $PREF \wedge SUFF^k$ only means the reachability of G from R which may contain unreachable states.

The second case is where R reaches a fixed point — the point from which R will never grow further. At this point R contains all reachable states and thus the unreachability of G is immediately concluded from the unsatisfiability of $PREF \wedge SUFF^k$. If this happens, the function terminates by returning false (line 17).

4. Experiment Results

To evaluate the effectiveness of our proposed method, we conducted experiments. We verified 21 pairs of telecommunication services described in Sect. 2.1 using the proposed unbounded model checking method, McMillan's unbounded model checking method [4] and the model checker SPIN [16]. We implemented these two unbounded model checking methods using McMillan's FOCI tool for both SAT solving and interpolant generation. In the proposed unbounded model checking method, the new proposed encoding and the proposed algorithm are used. On the other hand, McMillan's method uses conventional encoding with T and interpolation procedure proposed in [4].

SPIN, a very well-known model checker, uses explicit state representation, in the sense that it does not employ Boolean state space encoding. We construct Promela models for telecommunication services as follows. We represent each predicate instance by a Boolean variable. We use a single Promela process to represent the behavior of the whole system. The Promela process has, in turn, a single big do statement, in which every rule instance is represented as a guarded command. At any point of time, a single guarded

command whose guard is true is nondeterministically selected for execution.

The experiments were performed on a Linux (kernel 2.4) PC with a 2.8 GHz CPU and about 3 Gbyte memory.

We consider invariant properties for four of the seven services as follows:

OCS: If x puts y in the OCS screening list, x is never calling y at any time. ($\neg OCS(x, y) \vee \neg calling(x, y)$)

TCS: If x puts y in the TCS screening list, y is never calling x at any time. ($\neg TCS(x, y) \vee \neg calling(y, x)$)

DO: If x subscribes DO, x never receives dialtone at any time. ($\neg DO(x) \vee \neg dialtone(x)$)

DT: If x subscribes DT, y is never calling x at any time. ($\neg DT(x) \vee \neg calling(y, x)$)

Consequently a total of 18 pairs that contain at least one of the four services are verified against these invariant properties. Because of the symmetry of users, we check the violation with a single variable substitution by users. For example, the invariant of OCS is verified by checking reachability to $G(s) = \neg(\neg OCS(A, B) \vee \neg calling(A, B))$, where $OCS(A, B)$ and $calling(A, B)$ are Boolean variables representing predicate instances $OCS(A, B)$ and $calling(A, B)$.

Nondeterminism occurs in states where two rules simultaneously become enabled for the same event. Due to user symmetry, it suffices to check all event instances obtained from any single variable substitution θ_0 of users. Thus we let:

$$G(s) = \bigvee_{e\theta_0 : e \in E} \bigvee_{\substack{t_1, t_2 \in \mathcal{R}(t_1 \neq t_2): \\ e[t_1] = e[t_2] = e\theta_0}} E_{t_1}(s) \wedge E_{t_2}(s)$$

Here $e\theta_0$ is an event instance obtained by the substitution θ_0 and t_1 and t_2 are two different rule instances that have the same event instance $e\theta_0$. For example, consider the specification shown in Fig. 1 and let $\theta_0 = \langle x|A, y|B\rangle$. Three event instances, namely, $onhook(A)$, $offhook(A)$ and $dial(A, B)$, are shared by more than one rule instance. Specifically, $onhook(A)$ triggers $pots2\theta_0$, $pots5\theta_0$, $pots7\theta_0$ and $pots8\theta_0$, $offhook(A)$ triggers $pots1\theta_0$ and $pots6\theta_0$, and $dial(A, B)$ triggers $pots3\theta_0$ and $pots4\theta_0$. Hence we have $G(s) = \Big((E_{pots2\theta_0}(s) \wedge E_{pots5\theta_0}(s)) \vee (E_{pots2\theta_0}(s) \wedge E_{pots7\theta_0}(s)) \vee (E_{pots2\theta_0}(s) \wedge E_{pots8\theta_0}(s)) \vee (E_{pots5\theta_0}(s) \wedge E_{pots7\theta_0}(s)) \vee (E_{pots5\theta_0}(s) \wedge E_{pots8\theta_0}(s)) \vee (E_{pots7\theta_0}(s) \wedge E_{pots8\theta_0}(s))\Big) \vee \Big(E_{pots1\theta_0}(s) \wedge E_{pots6\theta_0}(s)\Big) \vee \Big(E_{pots3\theta_0}(s) \wedge E_{pots4\theta_0}(s)\Big)$.

Table 1 shows the results of the verification of the violation of invariant properties, while Table 2 shows the results of the verification of nondeterminism. The two leftmost columns represent the combination of services tested and whether feature interaction occurs in that combination.

For all the three methods, the execution time needed for verification is presented. The execution time is the total time that elapsed between when a service specification was input and when the verification was completed. NA means that we could not complete verification since an error was caused by memory overflow when the program was generating an interpolant. (The number inside the parentheses shows the elapsed time till the error occurred.)

For the proposed method and McMillan's method, (k, r) shows the value of k of the finally executed instance of FiniteRun and the number of times of computing an interpolant in that execution of FiniteRun.

For all cases where the verification was completed, our proposed method outperformed the McMillan's ordinary unbounded model checking in execution time. In particular, when a violation required a relatively large number of transitions to occur, the proposed method could conclude the existence of the violation using a much smaller value k (labeled [A] in Table 1 and Table 2).

For example, when CW and DT are activated at the same time, a feature interaction occurs after ten steps from the initial state. As can be seen in Table 1, the interaction was detected with $k = 10$ using the conventional encoding (McMillan's method). On the other hand, our approach can detect the interaction with $k = 2$. This is because that D represents not only one transition but also at most n consecutive transitions. This means that using our encoding scheme the state space can be explored with a lower value of k than the value required by McMillan's method.

This fact can be seen for the cases labeled [B] in Table 1 and Table 2. In these cases, our proposed method can explore all reachable states. On the other hand, McMillan's method can not explore the reachable states even with a greater value of k than the value required for our proposed method. For example, for the case DC + DO in Table 1, the over-approximation of the reachable states can be obtained by our proposed method with $k = 3$ and three times of computation of an interpolant. McMillan's method can not obtain the reachable states with $k = 8$ and five times of computation of an interpolant.

In some cases labeled [C] in Table 2, the same (k, r) is required, but our method outperformed McMillan's method in execution time. This is cased by the fact that our proposed encoding scheme can generate concise formulas to represent system's behavior. For example, in case OCS + TCS in Table 2, the system has 39 predicate instances and 78 rule instances. T consists of about 6,000 literals, and a formula to be checked in McMillan's method has about 12,000 literals. On the other hand, D consists of about 600 literals, and a formula to be checked in the proposed method has about 1,200 literals. The difference of these two formulas results in the difference of the execution time.

SPIN consistently exhibited good performance; but our proposed method outperformed SPIN for many cases labeled [D] in Table 1 and in Table 2. When our method showed lower performance or even aborted, a large k was (or would be) needed for the algorithm to terminate. This is explained by the fact the time needed for checking the satisfiability of the formula and generating an interpolant rapidly increases with the size of the input formula.

One might think that the result is somewhat discouraging; but we think there is still plenty of room for improving

Table 1 Violation of invariant.

		Proposed method		McMillan's method		SPIN
Service	Interaction	time(s)	(k,r)	time(s)	(k,r)	time(s)
CW + DO [E]		NA (2753.96)	(4,3)	NA (15311.30)	(11,3)	1.52
CW + DT [A] [D]	√	0.81	(2,0)	6595.31	(10,0)	1.43
CW + OCS [A] [D]	√	0.65	(2,0)	3819.62	(10,0)	1.57
CW + TCS [A] [D]	√	0.66	(2,0)	8130.83	(10,0)	1.59
CF + DO [E]		NA (1346.50)	(4,7)	NA (4973.47)	(7,4)	1.53
CF + DT [A] [D]	√	0.64	(2,0)	19.59	(6,0)	1.18
CF + OCS [A] [D]	√	0.47	(2,0)	49.93	(6,0)	1.26
CF + TCS [A] [D]	√	0.47	(2,0)	37.72	(6,0)	1.24
DC + DO [B] [E]		6.60	(3,3)	NA (4266.88)	(8,5)	0.94
DC + DT [D]		0.38	(2,2)	0.66	(2,2)	0.94
DC + OCS [D]	√	0.27	(2,0)	1.26	(3,0)	0.84
DC + TCS [D]	√	0.27	(2,0)	1.25	(3,0)	1.04
DO + DT [B] [E]		1149.39	(4,5)	NA (8892.33)	(8,4)	0.84
DO + OCS [B] [E]		1023.36	(4,6)	NA (6037.83)	(7,4)	0.92
DO + TCS [B] [E]		257.09	(3,6)	NA (1249.24)	(7,4)	0.91
DT + OCS [B] [E]		16.12	(3,5)	NA (29077.82)	(7,4)	0.97
DT + TCS [B] [E]		14.04	(3,4)	NA (13662.32)	(8,4)	1.08
OCS + TCS [D]	√	0.33	(2,0)	1.50	(3,0)	0.95

Table 2 Nondeterminism.

		Proposed method		McMillan's method		SPIN
Service	Interaction	time(s)	(k,r)	time(s)	(k,r)	time(s)
CW + CF [A] [E]	√	3.91	(2,0)	NA (1365.87)	(6,2)	2.10
CW + DC [E]		NA (12326.47)	(4,2)	NA (4776.48)	(7,1)	1.93
CW + DO [E]		NA (6512.17)	(5,1)	NA (4453.44)	(8,1)	1.62
CW + DT [A] [D]	√	3.22	(2,0)	1004.32	(8,0)	1.62
CW + OCS [A] [D]	√	2.03	(2,0)	3049.80	(8,0)	1.82
CW + TCS [A] [D]	√	2.06	(2,0)	3910.24	(8,0)	1.72
CF + DC [E]		NA (1033.90)	(2,4)	NA (7664.26)	(7,2)	2.32
CF + DO [E]		NA (1346.50)	(4,7)	NA (8177.80)	(8,2)	1.48
CF + DT [A] [D]	√	0.67	(2,0)	31.81	(5,0)	1.35
CF + OCS [A] [D]	√	0.51	(2,0)	39.49	(5,0)	1.28
CF + TCS [A] [D]	√	0.50	(2,0)	53.78	(5,0)	1.30
DC + DO [C] [D]	√	0.30	(2,0)	0.38	(2,0)	1.33
DC + DT [B] [E]		25.72	(3,3)	NA (35501.07)	(6,4)	0.93
DC + OCS [B] [E]		74.47	(3,4)	NA (3749.37)	(5,3)	1.16
DC + TCS [B] [E]		45.62	(3,3)	NA (3485.23)	(4,5)	1.15
DO + DT [B] [E]		4.83	(2,4)	NA (7057.75)	(3,8)	0.95
DO + OCS [B] [E]		16.31	(2,5)	NA (2922.07)	(3,7)	0.92
DO + TCS [B] [E]		19.55	(2,5)	NA (1700.26)	(3,6)	0.91
DT + OCS [C] [D]	√	0.28	(2,0)	0.71	(2,0)	0.87
DT + TCS [C] [D]	√	0.28	(2,0)	0.33	(2,0)	0.88
OCS + TCS [C] [D]	√	0.28	(2,0)	0.46	(2,0)	1.01

our method. In our method we use FOCI. FOCI is tool for checking the satisfiability of formulas and generating interpolant using a result of satisfiability checking. However there are faster SAT solver, such as MiniSat [17]. By developing a new tool for computing interpolants using such a faster SAT solver, we may be able to enhance the performance of our method.

5. Conclusion

In this paper, we proposed a verification method for checking whether or not feature interaction occurs in telecommunication services. We used a new encoding scheme that effectively represents the behaviors of asynchronous systems such as telecommunication systems. Based on this encoding, we developed an unbounded model checking method. To show the effectiveness of our method, we conducted experiments. To the best of our knowledge this was the first time to adapt unbounded model checking to the interleaving concurrency of asynchronous systems.

The results of the experiments showed that our proposed method can verify the services much more quickly than the conventional unbounded model checking. Comparing to the SPIN model checker, our method often exhibited better or comparable performance; but in some cases ours could not complete verification, while SPIN solved the verification problem in a few seconds.

There is plenty of room for improving the proposed method. Our implementation is still in its prototype stage. This is in contrast to SPIN, which has been consistently im-

proved for around two decades. Our current implementation uses FOCI for interpolant generation. FOCI supports not only pure propositional logic but also uninterpreted functions or linear arithmetic. By developing a new, faster interpolation procedure tailored to propositional logic, we may be able to enhance the performance of our method. This expectation can also be justified by the facts that the research on interpolation is still in its early stage, and that the performance of SAT solving has been improved by several orders of magnitude in this decade.

Acknowledgments

This work is supported in part by MEXT's GCOE program "Founding Ambient Information Society Infrastructure."

References

[1] E.M. Clarke, O. Grumberg, and D.A. Peled, Model Checking, MIT Press, 1999.
[2] A. Biere, A. Cimatti, E.M. Clarke, and Y. Zhu, "Symbolic model checking without bdds," Proc. 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS 1999), pp.193–207, London, UK, 1999.
[3] F. Copty, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M.Y. Vardi, "Benefits of bounded model checking at an industrial setting," Proc. 13th International Conference on Computer Aided Verification (CAV 2001), pp.436–453, London, UK, 2001.
[4] K.L. McMillan, "Interpolation and sat-based model checking," Proc. 15th International Conference on Computer Aided Verification (CAV2003), pp.1–13, Boulder, CO, USA, July 2003.
[5] J. Marques-Silva, "Interpolant learning and reuse in sat-based model checking," Electronic Notes in Theoretical Computer Science, vol.174, no.3, pp.31–43, 2007.
[6] N. Amla and K.L. McMillan, "Combining abstraction refinement and sat-based model checking," Proc. 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2007), pp.405–419, Braga, Portugal, March 2007.
[7] A. Rybalchenko and V. Sofronie-Stokkermans, "Constraint solving for interpolation," Proc. 8th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI 2007), pp.346–362, Nice, France, Jan. 2007.
[8] T. Yokogawa, T. Tsuchiya, M. Nakamura, and T. Kikuno, "Feature interaction detection by bounded model checking," IEICE Trans. Inf. & Syst., vol.E86-D, no.12, pp.2579–2587, Dec. 2003.
[9] S. Ogata, T. Tsuchiya, and T. Kikuno, "Sat-based verification of safe petri nets," Proc. 2nd International Symposium on Automated Technology for Verification and Analysis (ATVA 2004), SAT-Based Verification of Safe Petri Nets, pp.72–92, Nov. 2004.
[10] "ITU-T recomendations Q.1200 series, intelligent network capability set 1 (CS 1)," ITU-T, Sept. 1990.
[11] Bellcore, "Advanced intelligent network (AIN) release 1, switching systems generic requirement," Bellcore Technical Advisory TA-NWT.001123, 1991.
[12] A. Gammelgaard and J.E. Kristensen, "Interaction detection, a logical approach," Proc. 2nd Workshop on Feature Interaction in Telecommunication Systems, pp.178–196, Amsterdam, The Netherlands, May 1994.
[13] A. Khoumsi, "Detection and resolution of interactions between services of telephone networks," Proc. 4th Workshop on Feature Interaction in Telecommunication Systems, pp.78–92, Montréal, Canada, June 1997.
[14] Y. Hirakawa and T. Takenaka, "Telecommunication service description using state transition rules," Proc. 6th international workshop on Software specification and design (IWSSD 1991), pp.140–147, Como, Italy, Oct. 1991.
[15] P. Pudlák, "Lower bounds for resolution and cutting plane proofs and monotone computations," Symbolic Logic, vol.62, no.2, pp.981–998, Sept. 1997.
[16] G.J. Holzmann, "The model checker SPIN," IEEE Trans. Softw. Eng., vol.23, no.5, pp.279–295, 1997.
[17] N. Een and N. Sorensson, "Minisat — A sat solver with conflict-clause minimization," Proc. Theory and Applications of Satisfiability Testing (SAT 05), 2005.

Appendix

A. Correctness of Our Proposed Algorithm

We show the correctness of our proposed algorithm in Fig. 4.

Lemma 1: R is monotonically nondecreasing.
Proof: $D(s_0, s_n)$ evaluates to true when $s_0 = \cdots = s_n$. In other words, an interpolant includes $s_0 \in R$. Hence, R is monotonically nondecreasing.

Lemma 2: FiniteRun(M, k) terminates for every (M, k).
Proof: First, suppose that G is reachable from the initial state. If BMC^k is satisfiable, then the function terminates by returning true. If BMC^k is unsatisfiable, then the function proceeds to the iterative generation of interpolants. In this iterative process, R gradually increases until it reaches a fixed point (from Lemma 1). Such a fixed point must exist since the state space is finite and contains all reachable states. Thus R always grows to the extent where BMC^k is satisfiable, in which case the function aborts (at line 11) since $R \neq I$.

Next, suppose G is unreachable from the initial state. Since BMC^k is unsatisfiable at the first time (at line 6), the function tries to compute the over-approximate set of states reachable from the initial state. During this computation, if BMC^k becomes satisfiable, then the function aborts since $R \neq I$. Otherwise, the process of computing the over-approximation of the reachable state set terminates because the state space is finite. In this case, $R = R'$ holds at line 16 and thus false is returned (at line 17).

Lemma 3: For every M, there exists k such that FiniteRun(M, k) terminates without aborting.
Proof: If G is reachable, then the result of the first run of SAT run of SAT must be "satisfiable" when $k = |S| - 1$, in which case the function returns true and terminates.

Now suppose that G is unreachable and let k_G be the maximum length (the number of transitions) of the shortest path from any state in S to any state in G. Of course such a path only contains unreachable states. Let k be $k_G + 1$. Then $PREF \wedge SUFF^k$ is always unsatisfiable in any run of SAT. This can be explained by showing that R never contains unreachable states. In the first iteration, this trivially holds since $R = I$. In any later iteration, R is an interpolant for $PREF$ and $SUFF^k$, and thus $R(s_n) \wedge SUFF^k$ is unsatisfiable. This implies that R contains no unreachable state, because if an unreachable state existed in R, then $R(s_n) \wedge SUFF^k$ would

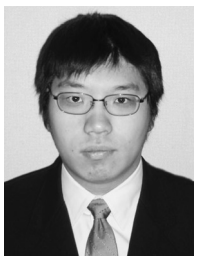
be satisfiable since G can be reached within $k_G = k - 1$ steps from that state. Since the number of states of the system is finite, R eventually reaches a fixed point, at which time the function returns false and terminates.

Lemma 4: When FiniteRun(M, k) terminates, it returns true if G is reachable and false if G is unreachable.

Proof: First suppose the function returns true. This means that BMC^k (at line 6) is satisfiable; hence G is reachable from initial state. Next suppose that the function returns false. In this case, R represents an over-approximation of the set of reachable states and $PREF \wedge SUFF^k$ is unsatisfiable with that R. Since any reachable state is contained in R, the fact that $PREF \wedge SUFF^k$ is unsatisfiable implies that no state in G is reachable.

Theorem 1: The algorithm shown in Fig. 3 terminates. It returns Reachable if G is reachable from the initial state; it returns Unreachable, otherwise.

Proof: The proof straightforwardly follows from Lemmas 2, 3 and 4.



Takafumi Matsuo received the Master of Information Science and Technology degree from Osaka University in 2006. He is currently working towards a Ph.D. degree in the Graduate School of Information Science and Technology at the same university. He is currently doing research on automatic verification of feature interactions in distributed systems.

Tatsuhiro Tsuchiya received the M.E. and Ph.D. degrees in engineering from Osaka University in 1995 and 1998, respectively. He is currently an associate professor in the Department of Information Systems Engineering at Osaka University. His research interests are in the areas of model checking and distributed fault-tolerant systems.

Tohru Kikuno received the M.S. and Ph.D. degrees from Osaka University in 1972 and 1975, respectively. He was with Hiroshima University from 1975 to 1987. Since 1990, he has been a Professor at Osaka University. His research interests include the quantitative evaluation of software development process and the analysis and design of fault-tolerant systems. He served as symposium chair of the 21st Symposium on Reliable Distributed Systems (SRDS 2002).