PAPER *Special Issue on Dependable Computing*

Feature Interaction Detection by Bounded Model Checking

Tomoyuki YOKOGAWA[†], ***Nonmember***, **Tatsuhiro TSUCHIYA**[††], **Masahide NAKAMURA**[†††], ***and*** **Tohru KIKUNO**[††], ***Regular Members***

SUMMARY Feature interaction is the term used in telephony systems to refer to inconsistent conflict between multiple communication services. Feature interaction is considered a major obstacle to developing reliable telephony systems and many approaches have been explored to resolve it. In this paper we present an automatic method for detecting latent feature interaction in service specifications. This method uses bounded model checking as its basis. The basic idea behind bounded model checking is to reduce the detection problem to the propositional satisfiability (SAT) decision problem. For asynchronous systems like telecommunication systems, however, traditional bounded model checking does not work well because resulting propositional formulas tend to become very large. We propose a new encoding scheme to overcome this problem and show the effectiveness through comparative experiments with traditional bounded model checking and other model checking methods.

key words: *bounded model checking, SAT, feature interaction*

1. Introduction

Feature interaction refers to situations where a combination of different services behaves differently than expected from the single services' behaviors. For example, consider a situation where user A has subscribed to the service *Originating Call Screening* (OCS) and does not want calls to user C to be put through, and user B has activated the service *Call Forwarding* (CF) to user C. In this situation, if A calls B, the intention of OCS not to be connected to C is invalidated since the call is put through to C by way of B (Fig. 1).

In today's intelligent telecommunication networks, feature interaction is considered a major obstacle to the introduction of new features and the provision of reliable services. In practical service development, however, the analysis of interactions has often been conducted in an ad hoc manner. This leads to time-consuming service design and testing without any interaction-free guarantee.

To overcome this situation, we propose a formal approach, aimed at detecting latent feature interaction in given communication service specifications. Although formal approaches have been well studied [7], ours is different in that it uses *bounded model checking.*

Model checking is a well-known formal approach for verifying systems that are modeled as a finite state machine. For realistic designs, however, the number of states of the system can be very large and the explicit traversal of the state space may become infeasible. This problem is usually called the state explosion problem.

Symbolic model checking [9] is one of the most successful approaches to state explosion. This method alleviates the problem by symbolically representing the state space by Boolean functions. Many symbolic model checking tools use *Binary Decision Diagrams* (BDDs) as the data structure to manipulate Boolean functions efficiently. Since Boolean functions can often be represented by BDDs very compactly, the symbolic model checking method can reduce the memory and time required for analysis.

Bounded model checking [2], [15] is a new symbolic model checking method which does not use BDDs. The central idea behind this method is to reduce the model checking problem to the propositional satisfiability (SAT) checking problem and to look for counterexamples that are shorter than some fixed length k for a given property. The formula to be checked is constructed by unwinding the transition relation of the system k times such that truth assignments satisfying the formula correspond to counterexamples.

In the literature, it has been reported that bounded model checking can work efficiently, especially for the verification of digital circuits. An advantage of this method is that it works efficiently even when compact BDD representation cannot be obtained. It is also an advantage that it can exploit recent advances in decision procedures of satisfiability.

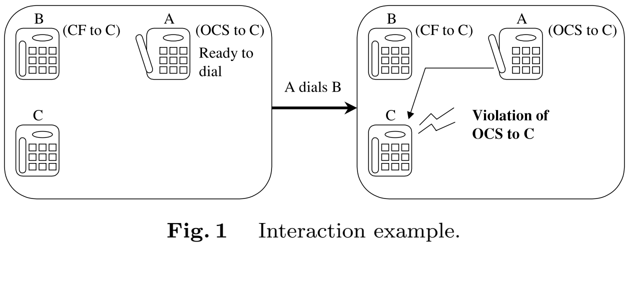

Fig. 1 Interaction example.

Manuscript received March 28, 2003.

[†]The author is with the Graduate School of Engineering Science, Osaka University, Toyonaka-shi, 560–8531 Japan.

[††]The authors are with the Graduate School of Information Science and Technology, Osaka University, Toyonaka-shi, 560–8531 Japan.

[†††]The author is with the Graduate School of Information Science, Nara Institute of Science & Technology, Ikoma-shi, 630–0101 Japan.

In contrast, this method does not work well for asynchronous systems, because the encoding scheme into propositional formulas is not suited for such systems. When applying this technique to asynchronous systems, a large formula would be required to represent the transition relation, thus resulting in large execution time and low scalability.

To overcome this problem we have been working on a new encoding. In this paper we describe the encoding scheme. The new encoding reduces the size of the resultant formula by exploiting the property that usually only small fraction of state variables take in part of each state transition. Interestingly, as a side-effect, the new scheme often explores a larger state space than the existing bounded model checking does for the same k. By applying the proposed scheme and other model checking methods to feature interaction detection, we show the effectiveness of the proposed method.

2. Services and Interaction

2.1 Communication Services

From ITU-T recommendation [6] (*ITU-T Recommendations Q.1200 Series* - Intelligent Network Capability Set 1 (CS1)) and Bellcore's feature standards [1] (*Bellcore* - LSSGR Features Common to Residence and Business Customers I, II, III), we selected the following seven services (features) to consider:

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.

Direct 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 offhooking, x is directly calling y. It is not necessary for x to dial y.

2.2 Feature Interaction

In this paper we consider two types of feature interaction. As shown below, the properties of the absence of these types of interaction can be viewed as safety properties, and hence detecting these types of interactions involves checking reachability from the initial state to undesirable states.

2.2.1 Nondeterminism

The first type we consider is *nondeterminism.* Nondeterminism is one of the best known types of feature interactions [3], [4], [8], [12], [13]. Nondeterminism refers to a situation where an 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.

It is known that this type of interaction occurs between CW and CF. Suppose that A subscribes both services. Now consider the situation where (1) A is talking with B, (2) C is ready to dial, and (3) D is in A's forwarding address list and is idle. In this situation, if C dials A, then either the call from C to A may be received by A because of A's CW feature, or it may be forwarded to D by the CF feature.

This type of interaction can be detected by checking reachability from the initial state to the states that cause nondeterminism. We call such states *nondeterministic states.*

2.2.2 Invariant Violation

The next type of interaction we consider is *invariant violation.* It is usually the case that services require some specific properties to be satisfied at any time. For example, for OCS service, the service designer may describe 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 property. It is known that combining multiple services can result in violation of invariant properties. The OCS plus CF example described in the first section falls in this type.

This type of feature interaction can also be detected by checking reachability from the initial state to the undesirable states where the invariant properties are violated.

3. Model

3.1 State Transition Rules

In this paper we adopt a variant of State Transition Rules (STR) [5], [13] to describe services and model the behavior of the system in a rigorous fashion.

A service specification is defined as 6-tuple $\langle U, V, P, E, R, s_{init}\rangle$, where U is a set of constants representing service users, V is a set of variables, P is a set of predicates, E is a set of events, R is a set of rules, and s_{init} is the (*initial*) state. Each rule $r \in R$ is defined as follows:

r : *pre−condition* [*event*] *post−condition*.

A *predicate* is of the form $p(x_1, \ldots, x_k)$ where $p \in P$ and $x_i \in V$. *Pre-condition* consists of predicates or negations of predicates, or both, while *Post-condition* consists of predicates only. An *event* is of the form $e(x_1, \ldots, x_k)$, where $e \in E$ and $x_i \in V$.

Figure 2 shows an example of a specification. This specification describes the Plain Old Telephone Service (POTS). Additional communication features, such as those described in the previous subsection, can be described by modifying this specification (for example, adding rules or predicate symbols). Specifications for the above services are shown in [11]. In all these specifications, it is assumed that at the initial state, all users are idle and no user subscribes to any service yet.

3.2 State Transition Model

Here we define the state transition system specified by the rule-based specification. Let $\langle U, V, P, E, R, s_{init}\rangle$ be a service specification. For $r \in R$, let $x_1, \ldots, x_n$ ($x_i \in V$) be variables appearing in r, and let $\theta = \langle x_1|a_1, \ldots, x_n|a_n\rangle$ ($a_i \in U, a_i \neq a_j$ ($i \neq j$)) be a substitution replacing each x_i in r with a_i. Then, an *instance* of r based on θ (denoted by $r\theta$) is defined as a rule obtained from r by applying $\theta = \langle x_1|a_1, \ldots, x_n|a_n\rangle$ to r. We represent the event and the post-condition of an instance $r\theta$ of a rule as $e[r\theta]$ and $Post[r\theta]$, respectively. In addition, we denote by $Pre[r\theta]$ and $\hat{Pre}[r\theta]$ the set of predicates in the pre-condition and the set of predicates whose negations are in the pre-conditions. Hence the precondition of an instance $r\theta$ of a rule is $Pre[r\theta] \cup \{\neg p \mid p \in \hat{Pre}[r\theta]\}$.

A *state* is defined as a set of *instances of predicates* $p(a_1, \ldots, a_k)$ where $p \in P$ and $a_i \in U$. We think of each state represents the instances of predicates that hold in that states.

Let s be a state. We say that an instance of rule, $r\theta$, is *enabled* for $e(r\theta)$ at s iff all instances in $Pre[r\theta]$ hold and no instances in $\hat{Pre}[r\theta]$ hold at s. The execution of the enabled rule causes the *next state* s' of s by deleting all instances in $Pre[r\theta]$ from s and adding all instances in $Post[r\theta]$ to s; that is,

$$s' = (s \backslash Pre[r\theta]) \cup Post[r\theta].$$

For example, suppose that $r = pots4$ in Fig. 2, $\theta = \langle x|A, y|B\rangle$, and $s = \{dialtone(A),\ dialtone(B)\}$. Then $Pre[r\theta] = \{dialtone(A)\}$, $\hat{Pre}[r\theta] = \{idle(B)\}$, $Post[r\theta] = \{busytone(A)\}$, and rule *pots4* with substitution θ is enabled for event $dial(A, B)$. If subscriber A dials B, that is, this event happens, then a state transition occurs, resulting in $s' = \{busytone(A),\ dialtone(B)\}$. Figure 3 shows the state transition diagram that is obtained from the STR specification shown in Fig. 2. In this diagram each circle represents a state and each arc between two states represents a state transition caused by execution of a rule instance. States that are not reachable from the initial

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

Fig. 2 Rule-based specification for POTS.

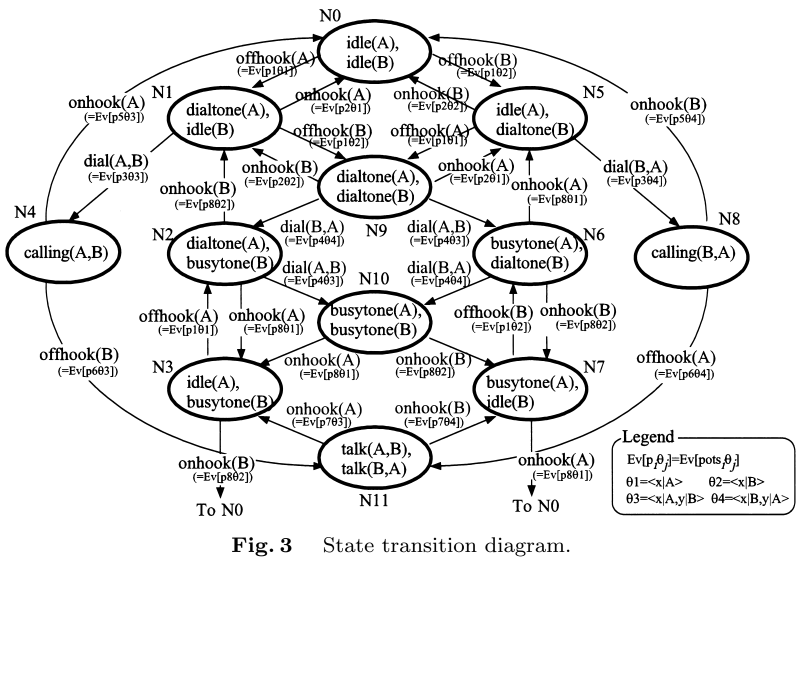

Fig. 3 State transition diagram.

state $\{idle(A), idle(B)\}$ are omitted in the diagram.

Let V denote the set of states. For each instance t of a rule, we define a relation $\xrightarrow{t}$ over states ($\xrightarrow{t} \subseteqq V \times V$) as follows: $s \xrightarrow{t} s'$ iff the execution of t causes s' from s. We also define a *computation* as a sequence of states $s_0 s_1 \cdots s_k$. such that for each $0 \leqq i < k$, (i) $s_i \xrightarrow{t} s_{i+1}$ for some t, or (ii) no rule is enabled at s_i and $s_i = s_{i+1}$. We think of the length of the computation as k.

4. Bounded Model Checking

Bounded model checking has received recent attention as an efficient verification method [2]. The basic idea of this method is to reduce the model checking problem to the propositional satisfiability decision problem.

For asynchronous systems, however, the existing bounded model checking does not work well because the propositional formula to be checked tends to become very large for such systems. Because of the asynchronous nature of telecommunication systems, it is thus not practical to apply the original method to feature interaction detection.

In order to avoid this problem we develop a new encoding scheme. We describe the scheme in detail in this section.

4.1 Symbolic Representation

To apply bounded model checking to service specifications, it is necessary to encode the state space and the transition relation by Boolean functions.

Let $\mathcal{P} = \{p_1, \cdots, p_m\}$ be the set of all instances of predicates and let $\mathcal{T} = \{t_1, \cdots, t_n\}$ be the set of all instances of rules ($m = |\mathcal{P}|$ and $n = |\mathcal{T}|$). A state s can then be viewed as a Boolean vector $s = (b_1, \cdots, b_m)$ such that $b_i = true$ iff an instance p_i of a predicate holds in that state.

Any set of states can be represented as a Boolean function such that

$$f(s) = \begin{cases} true & s \in \text{the set} \\ false & \text{otherwise.} \end{cases}$$

We say that f is a *characteristic function* of the state set.

For example, the characteristic function $E_t(s)$ of the set of states where $t \in \mathcal{T}$ is enabled is

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

Any relation R over states can be similarly encoded since they are simply sets of tuples.

$$F(s, s') = \begin{cases} true & sRs' \\ false & \text{otherwise.} \end{cases}$$

Now consider representing the relation $\xrightarrow{t}$ by Boolean function $T_t(s, s')$. Since execution of t causes (i) predicate instances in *Post*[t] to hold, (ii) those in *Pre*[t] but not in *Post*[t] not to hold, and (iii) those in neither *Pre*[t] nor *Post*[t] to be unchanged, we have

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

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

For example, consider the specification shown in Fig. 2. Let t be the instance of the rule '*pots*4 : $dialtone(x), \neg idle(y)[dial(x,y)]busytone(x)$' with substitution $(x, y) = (A, B)$. Since $Pre[t] = \{dialtone(A)\}$, $\hat{Pre}[t] = \{idle(B)\}$, and $Post[t] = \{busytone(A)\}$, we have $T_t(s, s') = dialtone(A) \wedge \neg idle(B) \wedge busytone(A)' \wedge \neg dialtone(A)' \wedge (idle(A) \leftrightarrow idle(A)') \wedge (idle(B) \leftrightarrow idle(B)') \wedge (dialtone(B) \leftrightarrow dialtone(B)') \wedge (busytone(B) \leftrightarrow busytone(B)') \wedge (calling(A, B) \leftrightarrow calling(A, B)') \wedge (calling(B, A) \leftrightarrow calling(B, A)') \wedge (path(A, B) \leftrightarrow path(A, B)') \wedge (path(B, A) \leftrightarrow path(B, A)')$.[†]

4.2 Existing Scheme

Let G denote the set of states whose reachability is to be decided and let $f_G(S)$ be the characteristic function for G. Although there are some variations [15], the basic formula used for checking reachability in bounded model checking is as follows.

$$\begin{aligned} &I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \cdots \wedge T(s_{k-1}, s_k) \\ &\wedge (f_G(s_0) \vee \cdots \vee f_G(s_k)) \end{aligned}$$

where $I(S)$ is the characteristic function of the set of the initial states, and

$$T(s, s') = \begin{cases} true & s' \text{ is reachable from } s \text{ in one step, or } s \text{ has no next states and } s = s'. \\ false & \text{otherwise.} \end{cases}$$

Clearly, $I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \cdots \wedge T(s_{k-1}, s_k) = true$ iff $s_0, s_1, \cdots, s_k$ is a computation from the initial states. Hence the above formula is satisfiable iff there is a state that is in G and reachable from one of the initial states in at most k steps. By checking the satisfiability of the formula, therefore, the verification can be carried out.

In practice, the formula often needs to be transformed into conjunctive normal form (CNF), since most of SAT solvers require for input formulas to be in that

[†]The same symbol is used to denote each predicate and its corresponding Boolean variable, since this is convenient and causes no confusion.

form. However the logically equivalent CNF formula can be exponential with respect to the size of the original formula. To avoid this, it is usual to use *structure preserving transformation* [14], which guarantees that the size of the resulting CNF formula is linear with the original formula. Thus the efficiency of verification critically depends on the size of the original formula in textual form.

Since we assume that exactly one rule is executed at a time, $T(s,s')$ will be

$$\begin{aligned}T(s,s') &= T_{t_1}(s,s') \vee \cdots \vee T_{t_n}(s,s')\\ &\vee (\bigwedge_{p_i \in \mathcal{P}} (b_i \leftrightarrow b'_i) \wedge \neg E_{t_1}(s) \wedge \cdots \wedge \neg E_{t_n}(s)).\end{aligned}$$

It should be noted that this formula would be very large in size in practice. Since T_t contains at least m literals, the total number of the literals in T is greater than $m * n$ literals.

4.3 Proposed Scheme

4.3.1 Encoding

Our proposed scheme alleviates the above problem with a new encoding. Let $Chng[t]$ denote the set of predicate instances that change as a result of execution of rule instance t; that is, $Chng[t] = (Post[t] \backslash Pre[t]) \cup (Pre[t] \backslash Post[t])$. Then T_t can be transformed as follows:

$$\begin{aligned}T_t(s,s') \;=\; & \bigwedge_{p_i \in Pre[t]} b_i \wedge \bigwedge_{p_i \in \hat{P}re[t]} \neg b_i\\ \wedge & \bigwedge_{p_i \in Post[t]\backslash Pre[t]} b'_i \wedge \bigwedge_{p_i \in Pre[t]\backslash Post[t]} \neg b'_i\\ \wedge & \bigwedge_{p_i \in \mathcal{P}\backslash Chng[t]} (b_i \leftrightarrow b'_i).\end{aligned}$$

Now let $D_t(s,s')$ be defined as follows.

$$\begin{aligned}D_t(s,s') \;=\; & T_t(s,s') \vee \bigwedge_{p_i \in \mathcal{P}} (b_i \leftrightarrow b'_i)\\ =\; & ((\bigwedge_{p_i \in Pre[t]} b_i \wedge \bigwedge_{p_i \in \hat{P}re[t]} \neg b_i\\ & \wedge \bigwedge_{p_i \in Post[t]\backslash Pre[t]} b'_i \wedge \bigwedge_{p_i \in Pre[t]\backslash Post[t]} \neg b'_i)\\ & \vee \bigwedge_{p_i \in Chng[t]} (b_i \leftrightarrow b'_i)) \wedge \bigwedge_{p_i \in \mathcal{P}\backslash Chng[t]} (b_i \leftrightarrow b'_i).\end{aligned}$$

For example, let t be the instance of the rule *pots*4 in Fig. 2 with substitution $(x,y) = (A,B)$. Then $D_t(s,s') = ((dialtone(A) \wedge \neg idle(B) \wedge busytone(A)' \wedge \neg dialtone(A)') \vee ((dialtone(A) \leftrightarrow dialtone(A)') \wedge (busytone(A) \leftrightarrow busytone(A)'))) \wedge (idle(A) \leftrightarrow idle(A)') \wedge (idle(B) \leftrightarrow idle(B)') \wedge (dialtone(B) \leftrightarrow dialtone(B)') \wedge (busytone(B) \leftrightarrow busytone(B)') \wedge (calling(A,B) \leftrightarrow calling(A,B)') \wedge (calling(B,A) \leftrightarrow calling(B,A)') \wedge (path(A,B) \leftrightarrow path(A,B)') \wedge (path(B,A) \leftrightarrow path(B,A)')$.

It is easy to see that $D_t(S,S') = true$ iff $S \xrightarrow{t} S'$ or $S = S'$. In other words, $D_t(S,S')$ differs from $T_t(S,S')$ only in that $D_t(S,S')$ evaluates to true also when $S = S'$. Using this property, a step (or more) can be represented by a conjunction of $D_t(s,s')$ as follows.

$$D_{t_1}(s_0,s_1) \wedge D_{t_2}(s_1,s_2) \wedge \cdots \wedge D_{t_n}(s_{n-1},s_n)$$

Note that this is in contrast to the traditional encoding, where a disjunction of $T_t(S,S')$ is used to represent one step. By definition, $s_0, s_1, \cdots, s_n$ satisfies this formula iff for any $0 \leqq i < n$, $s_i \xrightarrow{t_{i+1}} s_{i+1}$ or $s_i = s_{i+1}$. This means that if the formula 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 $s_0 \xrightarrow{t_i} s'$, it is satisfiable with an assignment such that $s_0 = \cdots = s_{i-1}, s_i = \cdots = s_n = s'$.

As a result, our proposed scheme uses the following formula φ for the verification.

$$\begin{aligned}&I(s_0)\\ &\wedge D_{t_1}(s_0,s_1) \wedge D_{t_2}(s_1,s_2) \wedge \cdots \wedge D_{t_n}(s_{n-1},s_n)\\ &\wedge D_{t_1}(s_n,s_{n+1}) \wedge D_{t_2}(s_{n+1},s_{n+2}) \wedge \cdots \wedge D_{t_n}(s_{2n-1},s_{2n})\\ &\cdots\\ &\wedge D_{t_1}(s_{(k-1)*n},s_{(k-1)*n+1}) \wedge \cdots \wedge D_{t_n}(s_{k*n-1},s_{k*n})\\ &\wedge f_G(s_{k*n})\end{aligned}$$

If the formula φ is satisfiable, then we can conclude that there is a state in G that can be reached from the initial state in at most $k*n$ steps. On the other hand, if the formula φ is unsatisfiable, then there is no state in G that can be reached from the initial state in less than or equal to k steps.

An important observation here is that the method may be able to find a state in G that requires more than k transition executions to reach.

4.3.2 Constructing a Succinct Formula

The most important advantage of our scheme is that φ can be converted into a more succinct formula that is not logically equivalent but has the same satisfiability. Let $s_j = (b_{1,j}, b_{2,j}, \cdots, b_{m,j})$ and $s_{j+1} = (b_{1,j+1}, b_{2,j+1}, \cdots, b_{m,j+1})$. In each $D_t(s_j,s_{j+1})$ in φ, term $(b_{i,j} \leftrightarrow b_{i,j+1})$ for any $p_i \in \mathcal{P}\backslash Chng[t]$ appears as a conjunct. Because of this, φ is satisfiable only if $b_{i,j}$ and $b_{i,j+1}$ have the same value. Hence a shorter formula that maintains the satisfiability is obtained by removing $(b_{i,j} \leftrightarrow b_{i,j+1})$ and replacing $b_{i,j+1}$ with $b_{i,j}$. That is, for each $D_t(s_j,s_{j+1})$ in φ, $(b_{i,j} \leftrightarrow b_{i,j+1})$ for all $p_i \in \mathcal{P}\backslash Chng[t]$ can be removed by quantifying away $b_{i,j+1}$ by applying the formula below.

$$\exists b_{i,j+1}(F \wedge (b_{i,j} \leftrightarrow b_{i,j+1})) = F|_{b_{i,j+1} \to b_{i,j}}$$

where F is an intermediate formula obtained from φ

```
for $p_i \in \mathcal{P}$
  $c_i := 0$;
$j := 0$;
$X := I(s)|_{s_i \to s_{i,0} \text{ for all } p_i \in \mathcal{P}}$;
for $step = 1, \cdots, k$ {
  for $t \in \mathcal{T}$ {
    $j := j + 1$;
    $X := X \wedge$
        $((\bigwedge_{p_i \in Pre[t]} b_{i,c_i} \wedge \bigwedge_{p_i \in \hat{P}re[t]} \neg b_{i,c_i}$
          $\wedge \bigwedge_{p_i \in Post[t] \setminus Pre[t]} b_{i,j} \wedge \bigwedge_{p_i \in Pre[t] \setminus Post[t]} \neg b_{i,j})$
          $\vee \bigwedge_{p_i \in Chng[t]} (b_{i,c_i} \leftrightarrow b_{i,j}))$
    for $p_i \in Chng[t]$
      $c_i := j$;
  }
}
$X := X \wedge f_G(S)|_{b_i \to b_{i,c_i} \text{ for all } p_i \in \mathcal{P}}$;
```

Fig. 4 Algorithm for constructing the formula used for verification.

and $F|_{y \to x}$ denotes the formula obtained from F by replacing y with x.

Note that $b_{i,j}$ may also be replaced by a further earlier version of variable $b_{i,j-1}$. In that case $b_{i,j+1}$ is replaced with $b_{i,j-1}$ as a result.

Consequently, D_t in φ can be replaced with

$$
\begin{aligned}
&((\bigwedge_{p_i \in Pre[t]} b_i \wedge \bigwedge_{p_i \in \hat{P}re[t]} \neg b_i \\
&\wedge \bigwedge_{p_i \in Post[t] \setminus Pre[t]} b'_i \wedge \bigwedge_{p_i \in Pre[t] \setminus Post[t]} \neg b'_i) \\
&\vee \bigwedge_{p_i \in Chng[t]} (b_i \leftrightarrow b'_i))
\end{aligned}
$$

by appropriately replacing some variables.

The number of literals occurring in the above formula is $4 * |Pre[t] \setminus Post[t]| + |Pre[t] \cap Post[t]| + 3 * |Post[t] \setminus Pre[t]| + |Post[t] \cap \hat{P}re[t]| + |\hat{P}re[t] \setminus Post[t]|$. T_t, which is the counterpart in the traditional encoding, contains at least $|\mathcal{P}|$ literals. Hence the proposed scheme can exploit its advantage if $Pre[t] \cup \hat{P}re[t] \cup Post[t]$ is a small fraction of the whole set of predicate instances $\mathcal{P}$. This is usually the case for the service specifications we consider and is more likely when the number of users is large.

Figure 4 shows the algorithm that directly constructs the shorter formula for a given k. In the algorithm, variable c_i is used to denote the earlier version of the variable that is substituted for $b_{i,j}$; that is, $b_{i,j}$ will be replaced with b_{i,c_i}.

4.4 Representing States where Interaction Occurs

The remaining problem is to represent states where nondeterminism occurs by Boolean function $f_G(s)$.

4.4.1 Nondeterminism

Nondeterminism occurs at a state s iff two rules, $r1$ and $r2$, can be triggered by the same event e at s. As shown in the previous example, when such $r1$, $r2$, and e are given, the set of states where they are enabled simultaneously is represented by

$$\bigvee_{\{\theta 1, \theta 2\}: e[r1\theta 1] = e[r1\theta 2]} E_{r1\theta 1} \wedge E_{r2\theta 2}.$$

Thus the characteristic function for the set of all states where nondeterminism occurs is

$$\bigvee_{\{r1, r2\}: r1, r2 \in R} \ \bigvee_{\{\theta 1, \theta 2\}: e[r1\theta 1] = e[r2\theta 2]} E_{r1\theta 1} \wedge E_{r2\theta 2}.$$

4.4.2 Invariant Violation

Given an invariant that is intended to be satisfied by a service, whether it is satisfied or not can be decided by checking the reachability to states where the property does not hold. In this case

$$f_G(s) = \neg Inv(s)$$

where $Inv(s)$ is the Boolean function representing the set of states where the invariant property holds.

5. Experimental Results

In order to evaluate the effectiveness of the proposed method, we conducted experimental evaluation for the seven services described in Sect. 2. We used the same ordering as in the given specification in the experiment. Combining two of the seven services, we examined a total of the 21 pairs.

The experiments were performed on a Linux workstation with a 853 MHz Pentium III processor. The number of users was assumed to be four. ZChaff, an implementation of Chaff [10], was used as a SAT solver.

For each problem we incremented k until interaction was detected. Tables 1 and 3 show the value of k for which interaction was first found and the time (in seconds) required by ZChaff to find a satisfying assignment for that value of k.

5.1 Nondeterminism

It has been known that out of a total of the 21 pairs of the seven services, 11 pairs cause nondeterminism. Since the proposed method in itself cannot prove the absence of feature interaction, we evaluated the performance of the detection method for these combinations only.

Table 1 compares the proposed encoding and the traditional one with respect to the running time, in seconds, required to detect nondeterministic states for

Table 1 Performance of bounded model checking for nondeterminism detection.

	k	time	Trad. scheme	length
CW+CF	2	3.02	4934.76	10
CW+DT	3	4.81	212.10	8
CW+OCS	2	2.90	330.15	8
CW+TCS	2	3.80	1470.37	8
CF+DT	2	0.02	53.52	5
CF+OCS	2	0.02	89.32	5
CF+TCS	2	0.02	65.10	5
DC+DO	1	0.02	0.87	2
DT+OCS	2	0.05	1.91	3
DT+TCS	1	0.02	1.86	3
OCS+TCS	1	0.01	1.01	2

Table 2 Performance of SMV and SVAL for nondeterminism detection.

	SMV	SMV(-early)	SVAL	length
CW+CF	12859.40	90473.00	17.45	10
CW+DT	82.12	410.37	3.29	8
CW+OCS	44.23	194.91	3.37	8
CW+TCS	39.28	168.28	9.65	8
CF+DT	12.51	8.21	1.83	5
CF+OCS	22.80	5.55	6.11	5
CF+TCS	27.52	5.55	2.45	5
DC+DO	1.21	0.25	0.31	2
DT+OCS	1.23	0.24	0.06	3
DT+TCS	1.66	0.24	0.11	3
OCS+TCS	1.86	0.29	0.11	2

these specifications. Items in the 'length' column represent the length of the shortest counterexample, that is, the shortest computation from the initial state to a nondeterministic state.

As can be seen in this table, when using the proposed encoding, interaction was detected with k of less than or equal to three for all cases. For CW plus CF case, for example, $k = 2$ was sufficient while the shortest counterexample computation is of length 10. This is because it may be possible to check execution of two or more rules by one formula $D_{t_1} \wedge D_{t_2} \wedge \cdots \wedge D_{t_n}$. In this experiment, we used the same ordering of rules as in the given specification in encoding the formula. Thus if two rules are executed in the order as in the specification, they can be checked by this single formula.

Note that the length of the shortest counterexample coincides with the smallest k value at which the traditional scheme can find such a computation. This resulted in large detection time of the traditional scheme, as shown in this table.

For comparison purposes, we also applied two other model checking tools to the same set of problems. The first one is SMV, which is a well-known BDD-based symbolic model checker. In SMV, properties to be checked are given in the form of Computation Tree Logic (CTL). As stated before, the absence of nondeterminism is a safety property and thus, it can be described as $AG\neg f_G$ in CTL.

Table 2 shows the results of applying SMV to interaction detection. Comparing with Table 1, it is clear that the proposed method detected interaction much more efficiently than SMV. The difference is most clear for CW plus CF. For this case, the running time of the proposed scheme was only three seconds, while SMV required more than three hours to complete detection.

By enabling 'early' option, it is possible to force SMV to work on-the-fly; that is, when using this option, SMV incrementally checks whether or not the property holds in a breadth-first manner, and terminates immediately if it finds that the property can be violated. Table 2 also shows the running time of SMV with this option enabled. As expected, this resulted in short detection time for some service combinations. However, for some cases such as CW+CF, CW+DT, CW+OCS, or CW+TCS, it ended up with larger running time. A common characteristic of these combinations is that the formula representing f_G is very large. Hence the reason is thought to be that the benefit of early termination was diminished by time consumed at each stage of the incremental checking.

The second model checking tool is SVAL, which is a tool which we had developed for feature interaction detection [12]. The SVAL tool employs explicit state enumeration with symmetry and partial order state reduction techniques.

As can be seen in Table 2, the proposed method and SVAL exhibited similar performance for four cases, namely, DC+DO, DT+OCS, DT+TCS, and OCS+TCS. The common characteristic of these cases is that nondeterminism occurs at a state that is very close to the initial state. In these cases, therefore, it is possible to detect interaction by exploring a small number of states, thus resulting in very small detection times of SVAL.

On the other hand, for the cases of CW+CF, CW+OCS, CW+TCS, CF+DT, CF+OCS, and CF+TCS, computations of relatively large length have to be examined to conclude the existence of nondeterministic states. For these cases, the proposed method outperformed the previous method, by efficiently exploring the large state space with symbolic representation.

5.2 Invariant Violation

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 to DO, x never receives dialtone at any time" $(\neg DO(x) \vee \neg dialtone(x))$

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

Tables 3 and 4 show the performance for bounded model checking and SMV, respectively. SVAL is ex-

Table 3 Performance of bounded model checking for invariant violation detection.

	k	time	Trad. scheme	length
CW+DT	3	1.00	1318.41	10
CW+OCS	2	0.24	2795.61	10
CW+TCS	2	0.21	1744.04	10
CF+DT	2	0.01	149.74	6
CF+OCS	2	0.02	173.00	6
CF+TCS	2	0.03	1850.80	6
DC+OCS	2	0.03	3.57	3
DC+TCS	2	0.04	3.68	3
OCS+TCS	2	0.12	3.13	3

Table 4 Performance of SMV for invariant violation detection.

	SMV	SMV(-early)	length
CW+DT	40.29	35.43	10
CW+OCS	23.51	13.41	10
CW+TCS	24.96	13.81	10
CF+DT	10.83	0.97	6
CF+OCS	22.46	1.10	6
CF+TCS	27.34	1.16	6
DC+OCS	1.83	0.32	3
DC+TCS	2.50	0.33	3
OCS+TCS	1.83	0.30	3

cluded because it does not support invariant violation checking. Comparing with Tables 1 and 2, it can be seen that these three methods exhibited similar tendencies.

6. Conclusions

In this paper, we proposed to use bounded model checking to detect feature interactions in telecommunication services. We developed a new encoding scheme that is tailored to this purpose and, by applying it to practical services, demonstrated its effectiveness.

We think of examining the ordering of rules as an interesting topic for future research. We used the same ordering as in the given specification in the experiment; however, the state space explored for a fixed k could be enlarged by using an appropriate ordering technique.

Acknowledgments

This work was in part supported by Grant-in-Aid from the Ministry of Education, Culture, Sports, Science and Technology of Japan (Nos. 13224060 and 14019055).

References

[1] Bellcore, Advanced Intelligent Network (AIN) Release 1, Switching Systems Generic Requirements, Bellcore Technical Advisory TA-NWT-001123, 1991.

[2] A. Biere, A. Cimatti, E.M. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," Proc. Tools and Algorithms for the Analysis and Construction of Systems (TACAS '99), LNCS 1579, pp.193–207, 1999.

[3] R. Dssouli, S. Some, J.W. Guillery, and N. Rico, "Detection of feature interactions with REST," Proc. Fourth Workshop on Feature Interactions in Telecommunications Systems, pp.271–283, 1997.

[4] A. Gammelgaard and E.J. Kristensen, "Interaction detection, a logical approach," Proc. Second Workshop on Feature Interactions in Telecommunications Systems, pp.178–196, 1994.

[5] Y. Hirakawa and T. Takenaka, "Telecommunication service description using state transition rules," Proc. IEEE Int'l Workshop on Software Specification and Design, pp.140–147, Oct. 1991.

[6] ITU-T Recommendations Q.1200 Series, Intelligent Network Capability Set 1 (CS1), ITU-T, Sept. 1990.

[7] D.O. Keck and P.J. Kuehn, "The feature and service interaction problem in telecommunications systems: A survey," IEEE Trans. Softw. Eng., vol.24, no.10, pp.779–796, Oct. 1998.

[8] A. Khoumsi, "Detection and resolution of interactions between services of telephone networks," Proc. Fourth Workshop on Feature Interactions in Telecommunications Systems, pp.78–92, 1997.

[9] K.L. McMillan, Symbolic Model Checking, Kluwer Academic, 1993.

[10] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient sat solver," Proc. 39th Design Automation Conference, 2001.

[11] M. Nakamura, Design and evaluation of efficient algorithms for feature interaction detection in telecommunication services, Ph.D. Dissertation, Osaka University, 1998.

[12] M. Nakamura and T. Kikuno, "Feature interaction detection using permutation symmetry," Proc. Fifth Int'l. Workshop on Feature Interactions in Telecommunication Networks and Distributed Systems (FIW '98), pp.193–207, 1998.

[13] T. Ohta and Y. Harada, "Classification, detection and resolution of service interaction in telecommunication services," Proc. Second Workshop on Feature Interactions in Telecommunications Systems, pp.60–72, 1994.

[14] D.A. Plaisted and S. Greenbaum, "A structure-preserving clause form translation," Journal of Symbolic Computation, vol.2, pp.293–304, Sept. 1986.

[15] M. Sheeran, S. Singh, and G. Stålmarck, "Checking safety properties using induction and a sat-solver," Proc. International Conference on Formal Methods in Computer-Aided Design (FMCAD 2000), LNCS 1954, pp.108–125, 2000.

Tomoyuki Yokogawa received the M.E. degree from Osaka University in 2000. He is currently studying towards the Ph.D. degree in the Graduate School of Engineering Science at the same university. He has been engaged in research on automatic verification.

Tatsuhiro Tsuchiya received the M.E. and Ph.D. degrees in computer science 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.

Masahide Nakamura received the M.E. and Ph.D. degrees in computer science from Osaka University in 1996 and 1999, respectively. He was post doctoral fellow at Ottawa University from 1999 to 2000. He is currently a research associate in the Department of Information Systems at Nara Institute of Science and Technology. His research interests include design, verification and testing of telecommunication services.

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 a symposium chair of the 21st Symposium on Reliable Distributed Systems (SRDS2002).