Abstract

Feature interaction is a kind of inconsistent conflict between multiple communication services and considered an obstacle to developing reliable telephony systems. In this paper we present an automatic method for detecting feature interactions in service specifications. This method uses bounded model checking, a SAT-based automatic verification technique.

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 will be invalidated since the call is put through to C by way of B. 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.

Many techniques have been explored to overcome this situation [4]. Among them, formal approaches have received much attention as a means of detecting feature interactions in communication service specifications. In this paper we propose a new formal approach which uses bounded model checking [1, 7]. The central idea behind bounded model checking 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.

In the literature, it has been reported that this method can work efficiently, especially for the verification of digital circuits. On the other hand, it 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 is required to represent the transition relation, thus resulting in large execution time and low scalability.

Recently we developed an alternative encoding method for 1-bounded Petri nets [8]. However, this method cannot be directly applied to feature interaction detection, because it does not consider inhibitor arcs (which correspond to negations of predicates in preconditions of rules in specifications). In this paper we extend the method to deal with the interaction detection problem.

2. Model

2.1. Specifications

In this paper we adopt a variant of State Transition Rules (STR) [3] to formally describe services.

A service specification is defined as 6-tuple \((U, V, P, E, R, s_{\text{init}})\), 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 \(s_{\text{init}}\) is the (initial) state. Each rule \(r \in R\) is in the form \(r : \text{pre-condition} \ [\text{event}] \text{post-condition}\).

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 1 shows an example of a specification. This specification describes the Plain Old Telephone Service (POTS). Additional communication features can be described by modifying this specification (for example, adding rules or predicate symbols). We assume that at the initial state, all users are idle and no user subscribes to any service yet.

An STR specification specifies the state transition system defined as follows. For \(r \in R\), let \(x_1, \ldots, x_n\) \((x_i \in V)\) be variables appearing in \(r\), and let \(\theta = (x_1 | a_1, \ldots, x_n | a_n)\) \((a_i \in U, a_i \neq a_j (i \neq j))\) be a substitution replacing each \(x_i\) in \(r\) with \(a_i\). Then, an instance of \(r\) based on \(\theta\) (denoted by \(r\theta\)) is defined as a rule obtained...
\(U = \{A, B\}\)
\(V = \{x, y\}\)
\(P = \{idle(x), dialtone(x), busytone(x), calling(x, y), path(x, y)\}\)
\(E = \{onhook(x), offhook(x), dial(x, y)\}\)
\(R = \{
\begin{align*}
pots1 : & \text{idle}(x) [offhook(x)] \text{dialtone}(x). \\
pots2 : & \text{dialtone}(x) [onhook(x)] \text{idle}(x). \\
pots3 : & \text{dialtone}(x), \text{idle}(y) [\text{dial}(x, y)] \text{calling}(x, y). \\
pots4 : & \text{dialtone}(x), \neg \text{idle}(y) [\text{dial}(x, y)] \text{busytone}(x). \\
pots5 : & \text{calling}(x, y) [\text{onhook}(x)] \text{idle}(x), \text{idle}(y). \\
pots6 : & \text{calling}(x, y) [\text{offhook}(y)] \text{path}(x, y), \text{path}(y, x). \\
pots7 : & \text{path}(x, y), \text{path}(y, x) [\text{onhook}(x)] \text{idle}(x), \text{busytone}(y). \\
pots8 : & \text{busytone}(x) [\text{onhook}(x)] \text{idle}(x). \\
pots9 : & \text{dialtone}(x) [\text{dial}(x, x)] \text{busytone}(x).
\end{align*}
\}
\(s_{\text{init}} = \{\text{idle}(A), \text{idle}(B)\}\)

**Figure 1. Rule-based specification for POTS.**

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 \(Pre[r \theta]\) the set of predicates in the pre-condition and the set of predicates whose negations are in the pre-conditions. Hence the pre-condition of an instance \(r \theta\) of a rule is \(Pre[r \theta] \cup \bigcup_{r \in Pre[r \theta]} \neg p\).

A state is defined as a set of instances of predicates \(p(a_1, \ldots, a_n)\)'s where \(p \in P\) and \(a_i \in U\). We think of each state as representing those that hold in the state.

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 \(Pre[r \theta]\) 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, \(s' = (s \setminus Pre[r \theta]) \cup Post[r \theta]\). For each instance \(t\) of a rule, we define a relation \(\rightarrow\) over states as follows: \(s \overset{t}{\rightarrow} s'\) iff the execution of \(t\) causes \(s'\) from \(s\). We also define a computation as a sequence of states \(s_0, s_1, \ldots, s_k\) such that for each \(0 \leq i < k\), (i) \(s_i \overset{t}{\rightarrow} s_{i+1}\) 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 \(k\).

For example, suppose that \(r\) is \(pots4\) in Figure 1, \(\theta = \langle x_1 | A, y | B \rangle\), and \(s = \{\text{dialtone}(A), \text{dialtone}(B)\}\). Then \(Pre[r \theta] = \{\text{dialtone}(A)\}\), \(Pre[r \theta] = \{\text{idle}(B)\}\), \(Post[r \theta] = \{\text{busytone}(A)\}\), and \(r \theta\) is enabled for event \(\text{dial}(A, B)\).

If subscriber \(A\) dials \(B\), that is, this event happens, then a state transition occurs, resulting in the next state \(s' = \{\text{busytone}(A), \text{dialtone}(B)\}\).

### 2.2. Feature Interactions

In this paper, we focus primarily on detection of nondeterminism, which is one of the best known types of feature interactions [2, 5, 6]. 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.

In the model, nondeterminism occurs iff the system can reach a state where different rules are simultaneously enabled for the same event. That is, nondeterminism occurs at a state \(s\) if there are two enabled instances \(t, t'\) of rules in \(s\) such that \(e[t] = e[t']\). Thus detecting this type of feature interaction requires to check the reachability to any of the nondeterministic states from the initial state \(s_{\text{init}}\).

### 3. Bounded Model Checking

#### 3.1. 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 \(P = \{p_1, \ldots, p_m\}\) be the set of all instances of predicates and let \(T = \{t_1, \ldots, t_n\}\) be the set of all instances of rules (\(m = |P|\) and \(n = |T|\)). A state \(s\) can then be viewed as a Boolean vector \(s = (b_1, \ldots, b_m)\) such that \(b_i = \text{true}\) 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 \(f(s) = \text{true}\) iff \(s\) is in the set. We say that \(f\) is the characteristic function of the state set.

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

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

Any relation over states can be similarly encoded since they are simply sets of tuples. For example, the relation \(\rightarrow\) is represented as Boolean function \(T_t(s, s')\) as follows.

\[
E_t(s) \land \bigwedge_{p_i \in Post[t] \setminus Pre[t]} b'_i \land \bigwedge_{p_i \in Pre[t] \setminus Post[t]} \neg b'_i \\
\land \bigwedge_{p_i \in \{P \setminus (Pre[t] \cup Post[t])\}} (b_i \leftrightarrow b'_i)
\]

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

#### 3.2. Existing Scheme

Let \(G\) be 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 [7], the basic formula used for checking reachability in bounded model checking is:

\[
I(s_0) \land T(s_0, s_1) \land T(s_1, s_2) \land \cdots \land T(s_{k-1}, s_k) \\
\land (f_G(s_0) \lor \cdots \lor f_G(s_k))
\]

where \(I(S)\) is the characteristic function of the set of the initial states, and \(T(s, s') = \text{true}\) iff (i) \(s'\) is reachable from \(s\) in one step or (ii) \(s\) has no next states and \(s = s'\).

Clearly, \(I(s_0) \land T(s_0, s_1) \land T(s_1, s_2) \land \cdots \land T(s_{k-1}, s_k) = \text{true}\) iff \(s_0, s_1, \ldots, s_k\) is a computation from an initial state.
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.

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

$$T_{i_1}(s, s') \lor \cdots \lor T_{i_n}(s, s')$$

where

$$\forall \left( \bigwedge_{p_i \in P} b_i \leftrightarrow b'_i \right) \land \neg E_{i_1}(s) \land \cdots \land \neg E_{i_n}(s)$$

It should be noted that this formula would be very large in size. This becomes a major obstacle to applying bounded model checking to feature interaction detection, because usually the running time of a SAT procedure critically depends on the size of the input formula in textual form.

### 3.3. Proposed Encoding

Our proposed scheme alleviates the above problem with a new encoding. Let $d_i(s, s') = T_i(s, s') \lor \bigwedge_{p_i \in P} (b_i \leftrightarrow b'_i)$. Then $d_i(s, s')$ is

$$\left( \bigwedge_{p_i \in Pre[t]} b_i \land \bigwedge_{p_i \in \neg Pre[t]} \neg b_i \land \bigwedge_{p_i \in Post[t]} b'_i \land \bigwedge_{p_i \in \neg Post[t]} \neg b'_i \right)$$

$$\lor \bigwedge_{p_i \in \{Pre[t] \cup Post[t]\} \setminus \{Pre[t] \cap Post[t]\}} (b_i \leftrightarrow b'_i)$$

It is easy to see that $d_i(s, s')$ is true iff $s \not\rightarrow S'$ or $S = S'$. In other words, $d_i(s, s')$ differs from $T_i(s, S')$ only in that $d_i(s, S')$ evaluates to true also when $S = S'$. Using this property, a step (or more) can be represented by a conjunction of $d_i$. Note that this is contrast to the traditional encoding, where a disjunction of $T_i(s, S')$ is used to represent one step. Specifically, our proposed scheme uses the following formula $\varphi$ for the verification.

$$I(s_0)$$

$$\land d_{i_1}(s_0, s_1) \land d_{i_2}(s_1, s_2) \land \cdots \land d_{i_n}(s_{n-1}, s_n)$$

$$\land d_{i_1}(s_n, s_{n+1}) \land d_{i_2}(s_{n+1}, s_{n+2}) \land \cdots \land d_{i_n}(s_{2n-1}, s_{2n})$$

$$\cdots$$

$$\land d_{i_k}(s_{k-1} s_k, s_k s_{k+1}) \land \cdots \land d_{i_n}(s_{kn-1}, s_{kn})$$

$$\land \forall G(s_{kn})$$

If the formula $\varphi$ is satisfiable, then we can conclude that there is a state in $G$ that can be reached in at most $k \ast n$ steps, because $\varphi$ evaluates to true iff (i) $s_0 = s_{init}$, (ii) for any $0 \leq i < k \ast n$, $s_i \not\rightarrow s_{i+1}$ for some $t \in T$ or $s_i = s_{i+1}$, and (iii) $s_{kn} \in G$. Therefore $s_{kn}$, which belongs to $G$, is reachable from $s_{init}$ in at most $k \ast n$ 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.

More importantly, $\varphi$ can be converted into a much succinct formula that is not logically equivalent but has the same satisfiability. Let $s_i = (b_{i,1}, b_{i,2}, \cdots, b_{i,m})$. The key is that all variables that correspond to the predicates in $P \setminus (Pre[t] \cup Post[t])$ or in $Pre[t] \cap Post[t]$ can be omitted by substituting an earlier version of the variable (that is, $b_{i,j'}$ for some $j' < j$) for it.

More specifically, for each $d_i(s_j, s_{j+1})$ in $\varphi$, term $b_i \iff b_{i+1}$ can be omitted for all $p_i \in P \setminus (Pre[t] \cup Post[t]) \cup (Pre[t] \cap Post[t])$ by quantifying $b_i \iff b_{i+1}$ away. That is, $d_i$ in $\varphi$ can be replaced with

$$\left( \bigwedge_{p_i \in Pre[t]} b_i \land \bigwedge_{p_i \in \neg Pre[t]} \neg b_i \land \bigwedge_{p_i \in Post[t]} b'_i \land \bigwedge_{p_i \in \neg Post[t]} \neg b'_i \lor \bigwedge_{p_i \in \{Pre[t] \cup Post[t]\} \setminus \{Pre[t] \cap Post[t]\}} (b_i \leftrightarrow b'_i) \right)$$

by appropriately replacing some variables. For example, when $t$ is the instance of the rule $pots4$ in Figure 1 with substitution $(x, y) = (A, B)$, the above formula would be

$$\text{(dialtone}(A) \land \neg \text{idle}(B) \land \text{busytone}(A') \land \neg \text{dialtone}(A')) \lor ((\text{dialtone}(A) \iff \text{dialtone}(A')) \land \text{busytone}(A) \iff \text{busytone}(A')).$$

### 3.4. Representing Nondeterministic States

The remaining problem is to represent states where nondeterminism occurs by Boolean function $f_G(s)$. When two rules, $r_1$ and $r_2$, triggered by the same event $e$ are given, the set of states where they are enabled simultaneously is represented by

$$\bigvee_{\{r_1, r_2\} : e \mid r_1 \in r_2} E_{r_1} \land E_{r_2}$$

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

$$\bigvee_{\{r_1, r_2\} : r_1 \in r_2} \bigvee_{\{r_1, r_2\} : e \mid r_1 \in r_2} E_{r_1} \land E_{r_2}$$

Although we limit our discussion to detecting nondeterminism in this paper, other types of interactions, for example, deadlock or violation of invariant properties, can also be detected by the proposed method by setting $f_G(s)$ appropriately.

### 4. Experimental Results

In order to evaluate the effectiveness of the proposed method, we conducted experimental evaluation for practical services. The experiments were performed on a Linux workstation with a 500 MHz Pentium III processor. The number of users was assumed to be four. ZChaff was used as a SAT solver.

We selected the following seven services (features) to consider: Call Forwarding (CF), Originating Call Screening (OCS), Terminating Call Screening (TCS), Denied Origination (DO), Denied Termination (DT), and Direct Connect (DC).
Table 1. Performance of feature interaction detection.

<table>
<thead>
<tr>
<th></th>
<th>( k = 1 )</th>
<th>( k = 2 )</th>
<th>SVAL</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>CF+DT</td>
<td>1.8</td>
<td>-</td>
<td>2.7</td>
<td>5</td>
</tr>
<tr>
<td>CF+OCS</td>
<td>0.5</td>
<td>0.5</td>
<td>10.7</td>
<td>13</td>
</tr>
<tr>
<td>CF+TCS</td>
<td>1.8</td>
<td>-</td>
<td>9.0</td>
<td>5</td>
</tr>
<tr>
<td>DC+DO</td>
<td>0.2</td>
<td>-</td>
<td>0.2</td>
<td>2</td>
</tr>
<tr>
<td>DT+OCS</td>
<td>0.2</td>
<td>-</td>
<td>0.1</td>
<td>3</td>
</tr>
<tr>
<td>DT+TCS</td>
<td>0.1</td>
<td>-</td>
<td>0.1</td>
<td>2</td>
</tr>
<tr>
<td>OCS+TCS</td>
<td>0.1</td>
<td>-</td>
<td>0.1</td>
<td>2</td>
</tr>
</tbody>
</table>

It has been known that out of a total of the 15 pairs of the six services, seven pairs cause nondeterminism [6]. 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 shows the running time, in seconds, required to detect nondeterministic states in these specifications. In all cases except the CF+OCS case, feature interaction was detected when \( k = 1 \). For the CF+OCS case, a nondeterministic state was detected when \( k = 2 \). The figures written in the \( k = 1 \) and \( k = 2 \) columns are the times required for the SAT solver to decide satisfiability.

For comparison purposes, we also measured the performance of SVAL, a tool which we had developed [6]. The SVAL tool does not use SAT; it employs explicit state enumeration with symmetry and partial order state reduction techniques. The column ‘SVAL’ shows the time required for this tool to find the first nondeterministic state for each case. The column ‘length’ shows the length of the computation to this nondeterministic state, that is, the length of the shortest counterexample showing that the specification is not interaction-free. It should be noted that this length coincides with the value of \( k \) that would be needed for detection if the existing bounded model checking scheme were used.

As can be seen in Table 1, 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 time of SVAL.

On the other hand, for the remaining three cases (that is, 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. 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.

Acknowledgments

The authors wish to thank Mr. Takayuki Hamada and Mr. Kazuhiro Kishigami for their technical supports. This work was in part supported by Grant-in-Aid from the Ministry of Education, Culture, Sports, Science and Technology of Japan (No. 13224060 and No. 14019055).

References


