Detecting Feature Interactions in Telecommunication Services with a SAT Solver

Tatsuhiro Tsuchiya
Osaka University
t-tutiya@ist.osaka-u.ac.jp

Masahide Nakamura
Nara Institute of Science & Technology
masa-n@is.aist-nara.ac.jp

Tohru Kikuno
Osaka University
kikuno@ist.osaka-u.ac.jp

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 $\langle U, V, P, E, R, s_{init}\rangle$, 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 in the form $r : pre{-}condition\ [event]\ post{-}condition$.

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 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 $\theta = \langle x_1|a_1, \ldots, x_n|a_n\rangle$ $(a_i \in U, a_i \neq a_j (i \neq j))$ 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

Proceedings of the 2002 Pacific Rim International Symposium on Dependable Computing (PRDC'02)
0-7695-1852-4/02 $17.00 © 2002 IEEE

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

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 $\hat{P}re[r\theta]$ 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 $Pre[r\theta] \cup \bigcup_{p\in\hat{P}re[r\theta]}\{\neg p\}$.

A *state* is defined as a set of *instances of predicates* $p(a_1, \ldots, a_k)$'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 $\hat{P}re[r\theta]$ 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 each instance t of a rule, we define a relation $\xrightarrow{t}$ over states 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 $0 \leq i < k$, (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.

For example, suppose that r is $pots4$ in Figure 1, θ = $\langle x|A, y|B\rangle$, and s = $\{dialtone\ (A), dialtone(B)\}$. Then $Pre[r\theta]$ = $\{dialtone(A)\}$, $\hat{P}re[r\theta]$ = $\{idle(B)\}$, $Post[r\theta]$ = $\{busytone(A)\}$, and $r\theta$ is enabled for event $dial(A, B)$. If subscriber A dials B, that is, this event happens, then a state transition occurs, resulting in the next state s' = $\{busytone(A), dialtone(B)\}$.

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 iff 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_{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 $\mathcal{P} = \{p_1, \cdots, p_m\}$ 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 $f(s) = true$ 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 $t \in \mathcal{T}$ is enabled is

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

Any relation over states can be similarly encoded since they are simply sets of tuples. For example, the relation $\xrightarrow{t}$ is represented as Boolean function $T_t(s, s')$ as follows.

$$E_t(s) \wedge \bigwedge_{p_i\in Post[t]\backslash Pre[t]} b'_i \ \wedge \bigwedge_{p_i\in Pre[t]\backslash Post[t]} \neg b'_i$$
$$\wedge \bigwedge_{p_i\in(\mathcal{P}\backslash(Pre[t]\cup Post[t]))\cup(Pre[t]\cap Post[t])} (b_i \leftrightarrow b'_i).$$

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

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:

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

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

Clearly, $I(s_0)\wedge T(s_0, s_1)\wedge T(s_1, s_2) \cdots \wedge T(s_{k-1}, s_k) = true$ iff $s_0, s_1, \cdots, s_k$ is a computation from 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

$$T_{t_1}(s, s') \vee \cdots \vee T_{t_n}(s, s') \\ \vee((\bigwedge_{p_i \in \mathcal{P}} b_i \leftrightarrow b_i') \wedge \neg E_{t_1}(s) \wedge \cdots \wedge \neg E_{t_n}(s))$$

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 $d_t(s, s') = T_t(s, s') \vee \bigwedge_{p_i \in \mathcal{P}}(b_i \leftrightarrow b_i')$. Then $d_t(s, s')$ is

$$((\bigwedge_{p_i \in Pre[t]} b_i \wedge \bigwedge_{p_i \in \hat{P}re[t]} \neg b_i \\ \wedge \bigwedge_{p_i \in Post[t] \setminus Pre[t]} b_i' \wedge \bigwedge_{p_i \in Pre[t] \setminus Post[t]} \neg b_i') \\ \vee \bigwedge_{p_i \in (Pre[t] \cup Post[t]) \setminus (Pre[t] \cap Post[t])}(b_i \leftrightarrow b_i')) \\ \wedge \bigwedge_{p_i \in (\mathcal{P} \setminus (Pre[t] \cup Post[t])) \cup (Pre[t] \cap Post[t])}(b_i \leftrightarrow b_i')$$

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. Note that this is contrast to the traditional encoding, where a disjunction of $T_t(S, S')$ is used to represent one step. Specifically, our proposed scheme uses the following formula φ for the verification.

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

If the formula φ is satisfiable, then we can conclude that there is a state in G that can be reached in at most $k * n$ steps, because φ evaluates to true iff (i) $s_0 = s_{init}$, (ii) for any $0 \leq i < k * n$, $s_i \xrightarrow{t} s_{i+1}$ for some $t \in \mathcal{T}$ or $s_i = s_{i+1}$, and (iii) $s_{k*n} \in G$. Therefore s_{k*n}, which belongs to G, is reachable from s_{init} in at most $k * n$ 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, φ can be converted into a much succinct formula that is not logically equivalent but has the same satisfiability. Let $s_i = (b_{1,i}, b_{2,i}, \cdots, b_{m,i})$. The key is that all variables that correspond to the predicates in $\mathcal{P} \setminus (Pre[t] \cup Post[t])$ 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 $d_t(s_j, s_{j+1})$ in φ, term $(b_{i,j} \leftrightarrow b_{i,j+1})$ can be omitted for all $p_i \in (\mathcal{P} \setminus (Pre[t] \cup Post[t])) \cup (Pre[t] \cap Post[t])$ by quantifying $b_{i,j+1}$ away. That is, d_t in ϕ can be replaced with

$$(\bigwedge_{p_i \in Pre[t]} b_i \wedge \bigwedge_{p_i \in \hat{P}re[t]} \neg b_i \\ \wedge \bigwedge_{p_i \in Post[t] \setminus Pre[t]} b_i' \wedge \bigwedge_{p_i \in Pre[t] \setminus Post[t]} \neg b_i') \\ \vee \bigwedge_{p_i \in (Pre[t] \cup Post[t]) \setminus (Pre[t] \cap Post[t])}(b_i \leftrightarrow b_i')$$

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 will be $(dialtone(A) \wedge \neg idle(B) \wedge busytone(A)' \wedge \neg dialtone(A)') \vee ((dialtone(A) \leftrightarrow dialtone(A)') \wedge (busytone(A) \leftrightarrow busytone(A)'))$.

3.4. Representing Nondeterministic States

The remaining problem is to represent states where nondeterminism occurs by Boolean function $f_G(s)$. When two rules, $r1$ and $r2$, triggered by the same event e are given, the set of states where they are enabled simultaneously is represented by

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

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

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

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.

	$k=1$	$k=2$	SVAL	length
CF+DT	1.8	-	2.7	5
CF+OCS	0.5	0.5	10.7	13
CF+TCS	1.8	-	9.0	5
DC+DO	0.2	-	0.2	2
DT+OCS	0.2	-	0.1	3
DT+TCS	0.1	-	0.1	2
OCS+TCS	0.1	-	0.1	2

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

[1] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proceedings of Tools and Algorithms for the Analysis and Construction of Systems (TACAS'99), number 1579 in LNCS*, pages 193–207, 1999.

[2] A. Gammelgaard and E. J. Kristensen. Interaction detection, a logical approach. In *Proceedings of Second Workshop on Feature Interactions in Telecommunications Systems*, pages 178–196, 1994.

[3] Y. Hirakawa and T. Takenaka. Telecommunication service description using state transition rules. In *Proceedings of IEEE Int'l Workshop on Software Specification and Design*, pages 140–147, October 1991.

[4] D. O. Keck and P. J. Kuehn. The feature and service interaction problem in telecommunications systems: A survey. *IEEE Transactions on Software Engineering*, 24(10):779–796, October 1998.

[5] A. Khoumsi. Detection and resolution of interactions between services of telephone networks. In *Proceedings of Fourth Workshop on Feature Interactions in Telecommunications Systems*, pages 78–92, 1997.

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

[7] M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a SAT-solver. In *Proc. of International Conference on Formal Methods in Computer-Aided Design (FMCAD 2000), LNCS 1954*, pages 108–125, 2000.

[8] T. Tsuchiya and T. Kikuno. A SAT-based model checking method for asynchronous concurrent systems. *IEICE Technical Report*, 2002 (to appear).