Preventing Feature Interactions by Constraints
Jihong Zuo, Qianxiang Wang, Hong Mei
Institute of Software
School of Electronics Engineering and Computer Science
Peking University, Beijing 100871, China
{zuojh, wqx, meih}@pku.edu.cn

Abstract
As software systems evolve by adding new extensions some unexpected conflicts may occur, which is known as the Feature Interaction Problem (FIP). FIP is a threat to the dependability of software systems as it can break system security and safety. A major cause for FIP is the non-determinism related with sending or receiving the predefined signals of the base subsystem by extensions. This paper analyzes the problems and proposes to enforce necessary constraints to prevent the FIP. In addition, we present a systematic approach to acquire heuristically constraints from system specifications.

1. Introduction

Nowadays, many systems adopt the extension-based evolution strategy, i.e., adding new extensions (also called features) into a base subsystem, to enhance system functionality. While the strategy supports concurrent development and deployment of different extensions, it also faces the challenge of the Feature Interaction Problem (FIP), that is, different extensions may interfere with each other, causing chaos or strange behaviors in the system [4]. Studies have shown that feature interactions are widely present in many systems such as Email [6], telecommunication systems [8], and Internet telephony [9]. Recently, people begin to notice the phenomena of aspect interactions in the field of Aspect-Oriented Programming (AOP) [11]. If each aspect is a special implementation of an extension, aspect interactions can be seen as a special form of feature interactions. FIP poses a big threat to the dependability of software systems because it can break system security and safety [2, 8].

Formal methods prove to be one of the most effective methods to detect feature interactions. For such methods, the verification is performed on a composite formal model composed of the base model and at least two extension models. And the criteria of identifying a conflict may be system-specific or general-purpose properties. However, the methods also face some problems. One is non-monotonic extension, which means that adding a new extension may necessitate modifications of the original base model, which can invalidate some assumptions made earlier [12]. Another problem is the poor scalability due to the fast growth of extension combinations.

Rather than detecting conflicts directly, we propose to detect conflict-prone behaviors that can lead to non-determinism. Many reactive systems may define a set of signals that carry specific meanings. Sending or receiving such a predefined signal is often associated with some specific condition. However, the associated conditions are often not expressed explicitly. Non-determinism may happen if the signal-related conditions are violated, which is an important source of feature interactions. To eliminate the non-determinism, we need to formulate the signal-related constraints and put them into the base model. Afterwards, we can apply model checking on the composite model composed of the base model and an individual extension model. The verification will reveal the behaviors that violate some constraints, which are considered to be conflict-prone. By correcting the conflict-prone behaviors the possible related FIP can be prevented.

In this paper, we also present a systematic approach to formulate constraints starting from the functional model of the base, which is described as a logical transition system. All the transitions will be analyzed to extract either the post-conditions for receiving signals or the preconditions for sending signals. The conditions for sending or receiving the same signal will be grouped together. Afterwards, designers can heuristically define a constraint for each group of conditions through generalization.

Our contributions lie in two aspects. First, we propose an approach to prevent the feature interactions caused by sending or receiving the predefined signals. The rationale is to eliminate non-determinism using constraints such that feature interactions can be prevented. Compared with other existing works, our approach can provide several benefits: 1) it is more scalable because we only need to verify each individual extension model against the common constraints, which can bypass the trouble of extension combinations; 2) the verification can reveal the causes for potential conflicts even though the conflicts are not present yet; 3) the common constraints can serve
as axioms to keep the stability and validity of the base model.

Second, we propose an approach to systematically formulate constraints about sending and receiving predefined signals. The approach can help base designers extract constraints at an early stage.

The rest of the paper is structured as follows: Section 2 analyzes the non-determinism related with sending and receiving signals; Section 3 presents our approach; Section 4 is about the experiment; Section 5 discusses the related works and concludes the paper.

2. Non-determinism Related with Sending and Receiving Signals

In this section, we will briefly analyze the non-determinism related with sending and receiving signals and discuss how to eliminate such non-determinism.

![Figure 1. Sending a signal by the base](image1)

Suppose a process \( p \) sends a signal \( m \) to a process \( q \). And the sender \( p \) may be of two structures. In the first case, as illustrated in Figure 1, \( p \) is only composed of the base, the model of which is denoted by M1. And in the second case, as illustrated in Figure 2, \( p \) is composed of the base and an extension \( E \), the model of which is denoted by M2. If the signal \( m \) is a state signal, i.e., its function is to notify the receiver \( q \) of the sender \( p \)'s current situation, sending \( m \) is usually associated with a pre-condition. The \( \text{idle	signal} \) in the telecom system is such an example that is used to notify the receiver to destroy some connection. And the pre-condition for receiving \( m \) can be denoted as \( P_{M1,m} \) when \( p \) is composed of the base, and \( Q_{M2,m} \) when \( p \) is composed of the base and an extension \( E \). Thus, \( m \) does not know if the signal goes to the base or the extension \( E \), it cannot decide if its assumption of \( P_{M1,m} \) holds or not.

To eliminate the non-determinism we need to define a constraint \( CS_m \) on sending signal \( m \) such that:

\[
M1 \models \text{send}(p, m, q) \rightarrow CS_m
\]

\[
M2 \models \text{send}(p, m, q) \rightarrow CS_m
\]

Here, \( M \models \varphi \) means that the model \( M \) satisfies the formula \( \varphi \), “\( \rightarrow \)” denotes implication, and \( \text{send}(p, m, q) \) represents the process \( p \) sending the signal \( m \) to the process \( q \). From the perspective of the receiver, in whatever the case, it is assured that at least \( CS_m \) holds when the signal \( m \) is sent. Thus, \( CS_m \) can become its reliable assumption.

If the signal \( m \) is a command signal, i.e., its function is to notify the receiver \( p \) to do some specific actions as required by the sender \( q \), receiving \( m \) is usually associated with a post-condition. The \( \text{disconnect} \) signal in the telecom system is such an example that is used to notify the receiver to destroy some connection. And the post-condition for receiving \( m \) can be denoted as \( Q_{M1,m} \) when \( p \) is only composed of the base, and \( Q_{M2,m} \) when \( p \) is composed of the base and an extension \( E \). Thus, \( m \) does not know if the signal \( m \) goes to the base or the extension \( E \), it cannot decide if its assumption of the post-condition \( Q_{M1,m} \) holds or not. To eliminate the non-determinism we need to define the constraint \( CR_m \) on receiving signal \( m \) such that:

\[
M1 \models \text{receive}(p, m, q) \rightarrow CR_m
\]

\[
M2 \models \text{receive}(p, m, q) \rightarrow CR_m
\]

where \( \text{receive}(p, m, q) \) represents that the process \( p \) receives the signal \( m \) from the process \( q \).

3. Approach

To acquire constraints, we first describe the base model as a transition system. A transition system is composed of a set of transitions, which can be specified using temporal logic. Our specification style is inspired by [1]. Each transition has the following form:

\[
S_\alpha \land E_\alpha \rightarrow o(E_\beta \land S_\beta)
\]  

(1)

where \( S_\alpha \) and \( S_\beta \) characterize the current and the next-time system state respectively; \( E_\alpha \) and \( E_\beta \) characterize the current and the next-time event respectively; and the temporal operator “\( o \)” represents “next”. \( S_\alpha \) and \( S_\beta \) are conjunctions of atomic state formulae. For example, state(p, idle) represents that the process \( p \) is in idle state. \( E_\alpha \) and \( E_\beta \) are conjunctions of atomic event formulae, which are used to represent sending or receiving signals. For example, \( \text{send}(p, \text{idle	signal}, q) \) represents that the process \( p \) sends an idle_signal to the process \( q \).

Given a transition in the form of (1), if \( E_\beta \) contains
send(p, m, q) and m is a state signal, we can extract the precondition for send(p, m, q) as follows:

\[ P_m \land S_\beta \rightarrow \Box (S_a \land E_\alpha) \] (2)

where the temporal operator “\( \Box \)” represents “previously”. It means when sending the signal m and if the current process state satisfies \( S_\beta \), then at the previous time the process state should satisfy \( S_a \) and the process should have done the events as specified by \( E_\alpha \). Each send(p, m, q) may be associated with multiple preconditions, which are grouped together as shown below:

\[ \text{send}(p, m, q) \rightarrow \{P_{m,i}\} \]

Similarly, if \( E_\alpha \) contains receive(p, m, q) and m is a command signal, we can extract the post-condition for receive(p, m, q) as follows:

\[ Q_m \land S_\alpha \rightarrow \diamond E_\beta \land S_\gamma \] (3)

It means when receiving the signal m and if the current process state satisfies \( S_\alpha \), then at the next time the process state should satisfy \( S_\gamma \) and \( E_\beta \) and the process should do the events as specified by \( E_\gamma \). Also, each receive(p, m, q) may be associated with multiple post-conditions, which are grouped together as shown below:

\[ \text{receive}(p, m, q) \rightarrow \{Q_{m,j}\} \]

For each group of preconditions or post-conditions, we can define a constraint by generalization such that extensions are conditionally allowed to send or receive the signal in new states. In practice, the generalization usually requires domain-specific knowledge.

Given a precondition set \( \{P_{m,1}\} \subseteq \{P_{m}\} \), to generalize \( \{P_{m,1}\} \) we can heuristically search for \( U \) and \( V \) such that for each \( P \in \{P_{m,1}\} \) that is in the form of (2), the formulae of \( S_\beta \rightarrow U \) and \( S_a \land E_\alpha \rightarrow V \) are valid. Thus, the sub-constraint, i.e., part of a constraint, for \( \{P_{m,1}\} \) is:

\[ U \land \Box V \]

And all the sub-constraints are combined through disjunction to acquire the corresponding constraint. Suppose we have a set \( \{S_{m,1}, \ldots, S_{m,g}\} \) of sub-constraints for \( \{P_{m,1}\} \), then the constraint for send(p, m, q) is:

\[ CS_m: S_{m,1} \lor \ldots \lor S_{m,g} \]

Similarly, given a post-condition set \( \{Q_{m,1}\} \subseteq \{Q_{m}\} \), to generalize \( \{Q_{m,1}\} \) we can heuristically search for \( X \) and \( Y \) such that for each \( Q \in \{Q_{m,1}\} \) that is in the form of (3), the formulae of \( S_\alpha \rightarrow X \) and \( E_\beta \land S_\gamma \rightarrow Y \) are valid. Thus, the sub-constraint for \( \{Q_{m,1}\} \) is:

\[ X \land \diamond Y \]

Suppose we have a set \( \{R_{m,1}, \ldots, R_{m,h}\} \) of sub-constraints for \( \{Q_{m}\} \), then the constraint for receive(p, m, q) is:

\[ CR_m: R_{m,1} \lor \ldots \lor R_{m,h} \]

After acquiring the necessary constraints, the next step is to combine the base model and each individual extension model and verify the composite model against all the constraints. Suppose we have a composite model \( M \) and a set \( M \) of the predefined signals of the base model. Thus, for each \( m \in M \) we need assure the following two formulae hold:

\[ M \models \text{send}(p, m, q) \rightarrow CS_m \]

\[ M \models \text{receive}(p, m, q) \rightarrow CR_m \]

if \( CS_m \) and \( CR_m \) are available. Whenever a constraint violation is detected, it means that the extension has a conflict-prone behavior, which needs to be corrected.

Note that \( CS_m \) contains the past-time temporal operator “\( \Box \)” But many existing model checking tools do not support past-time LTL directly. As a solution, we managed to transform constraints into assertions in our study. To construct the assertions, the key is to record the values of the relevant state variables and event variables. This can be achieved by inserting some monitoring points within transitions. At each monitoring point, a specific event, e.g., receive(p, m, q) or send(p, m, q), is fired to a monitor process. On receiving the event, the monitor will collect the values of global state variables and check the appropriate assertions if necessary. The detail has to be omitted for limited space.

4. Experiment

We conducted an experiment on the telecom system. The base model is described as a transition system in temporal logic, which consists of 21 transitions. The transitions involve 2 state signals and 7 command signals. With the transitions we produce 7 preconditions for sending state signals and 18 post-conditions for receiving command signals. Based on that, 9 constraints are generated, one constraint for each signal.

![Figure 3. The architecture of the system model](image-url)

The verification is conducted using Spin [7]. As Spin has its own description language Promela, we rewrite the model in it. As illustrated in Figure 3, the system model consists of the following processes:

- **user**
- **monitor**
- **bcm(1)**
- **bcm(2)**
- **bcm(3)**
- **network**
- **f(1)**
1. Three \( \text{bcm} \) processes: A \( \text{bcm} \) process models a user-side process, which responds to user actions and external signals from the network.
2. One network process: The network process relays signals between channels.
3. One user process: The user process models all the user actions.
4. One monitor process: The monitor process observes the behaviors of other processes and checks them against constraints.
5. One feature process: The \( f(1) \) process represents a feature that is added on top of \( \text{bcm}(1) \).

Five features have been implemented and verified and the result of the experiment is illustrated in Table 1, where Call Waiting (CW) is to handle new incoming calls when the subscriber’s line is busy; Three-Way Calling (TWC) is to set up a three-party conference call; Call Forwarding on Busy (CFB) is to forward incoming calls when the subscriber’s line is busy; Originating Call Screening (OCS) is to filter the outgoing calls whose callees are on screen list; and Terminating Call Screening (TCS) is to filter the incoming calls whose callers are on the screen list.

<table>
<thead>
<tr>
<th></th>
<th>CW</th>
<th>TWC</th>
<th>CFB</th>
<th>OCS</th>
<th>TCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num of violations</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

5. Related Work and Conclusion

Nearly all of the existing works using formal methods focus on detecting interactions directly \([1, 3, 5, 8, 10]\). Here, we only introduce three typical works. In \([5]\), features are specified as a collection of temporal logic formulae at a high level. Interactions between two features \( F_1 \) and \( F_2 \) are detected if \((F_1 \land F_2)\) is unsatisfiable with respect to axioms about system behavior. In \([1]\), temporal logic is used to specify the transitions of the base and feature models. The base model and feature models are composed by conjunction. Interactions are detected if some "bad" state is reachable. In \([10]\), the base and features are specified as a set of processes in LOTOS. Features are combined by using a choice operator (i.e., logical disjunction). And a set of temporal properties are specified using \( \mu \)-calculus. Given a network model \( N \), two feature models \( F_1 \) and \( F_2 \), and properties \( \varphi_1 \) and \( \varphi_2 \), an interaction is detected if \( N \varphi F_1 \models \varphi_1 \) and \( N \varphi F_2 \models \varphi_2 \), but \((B \varphi F_1) \varphi F_2 \not\models \varphi_1 \land \varphi_2 \). Here, " \( \varphi \) " represents composition by choice. And \( M \not\models \varphi \) means that the model \( M \) does not satisfy the property \( \varphi \).

In contrast, our approach focuses on detecting the conflict-prone behaviors that can lead to the non-determinism related with sending or receiving signals. Compared with existing methods, this approach is more scalable. And it also helps keep the stability and validity of the base model. In addition, we also present a systematic approach to formulate constraints. The experiment shows that our approach is feasible and efficient.

Acknowledgements

The research is supported by the National Natural Science Foundation of China under Grant No. 60233010, 90412011, the National Basic Research Program (973) of China under Grant No. 2005CB321805, and the National Grand Fundamental Research Program of China under Grant No. 2002CB3120003.

References