Symbolic model checking of interactions in sequence diagrams with combined fragments by SMV

Yuka KAWAKAMI†, Tomoyuki YOKOGAWA†, Hisashi MIYAZAKI†‡, Sousuke AMASAKI†, Yoichiro SATO†, and Michiyoshi HAYASE†

Abstract—In this paper, we proposed a method for detecting consistency violation between state machine diagrams and a sequence diagram defined in UML 2.0 using SMV. We extended a method expressing these diagrams defined in UML 1.0 with boolean formulas so that it can express a sequence diagram with combined fragments introduced in UML 2.0. This extension made it possible to represent three types of combined fragment: alternative, option and parallel. As a result of experiment, we confirmed that the proposed method could detect consistency violation correctly with SMV.

Keywords—UML, model checking, SMV, sequence diagram.

I. INTRODUCTION

Unified Modeling Language (UML)[1] is a formal language used to describe structure and behavior of a software system and is widely used in software development. In software development using UML, dynamic behavior of a system is modeled by state machine diagram and sequence diagram. The state machine diagram focuses on state transitions of an element in regard to various events and the sequence diagram focuses on message interchanges between elements along with a time sequence. Because these two diagrams are used to represent different perspectives of a system separately, consistency between them can easily be violated. Such an inconsistency leads to errors in the latter stages of development.

Because it is difficult to detect inconsistency with human review, automatic methods for verifying consistency of UML diagrams were proposed[2], [3], [4], [5]. We have also developed a method[6] for verifying consistency between state machine diagrams and a sequence diagram using symbolic model checker SMV[7]. This method modeled message interchanges in a sequence diagram as a transition system using symbolic representation. However, these methods can not verify UML diagrams with combined fragments introduced in UML 2.0 which are used to describe complex structure in a sequence diagram.

In this paper, we proposed a method for verifying consistency of state machine diagrams and a sequence diagram with combined fragments. We extended the method we have proposed[6] so that it can model a sequence diagram with combined fragments. This method can model three types of combined fragment. We applied this method to an example and confirmed it could detect inconsistency correctly using SMV.

II. CONSISTENCY OF UML DIAGRAMS

The consistency of UML diagrams is classified into static and dynamic consistency. The static consistency means correspondence of elements of diagrams and the dynamic consistency means correspondence of behaviour of diagrams. In this paper, we focused on verification of the dynamic consistency.

For example, consider a system described by three UML diagrams in Figure 1. State machine diagrams in Figure 1(a) and (b) describe that Obj1 sends a message m1 and Obj2 receives it. However, sequence diagram in Figure 1(c) describes that Obj2 sends m1 and Obj1 receives it. Thus behaviours of these diagrams are inconsistent.

![Fig. 1. An example of consistency violation](image)

III. SYMBOLIC REPRESENTATION

A. State machine diagram

The state machine diagram models transition relations of elements in a system. In the proposed method, a transition in a state machine diagram is expressed as a boolean formula equivalent to execution of that transition.

A transition \( t_i \in T \) is represented as a formula \( act(t_i) \) which evaluates true iff \( t_i \) is executed. \( act(t_i) \) is a conjunction of the following three formulas: \( pr(t_i) \) representing pre-condition of \( t_i \), \( post(t_i) \) representing post-condition of \( t_i \), and \( in(t_i) \) representing condition for unchanged elements of \( t_i \).

The pre-condition of \( t_i \) is a conjunction of the following three conditions: (1) source state of \( t_i \) is active, (2) event of \( t_i \) is activated, and (3) guard condition of \( t_i \) evaluates true. Hence \( pr(t_i) \) is represented as the following boolean formula:

\[
pr(t_i) \equiv src_i \land evt_i \land \text{grd}_i,
\]
where srcₖ is a boolean variable representing that source state of tᵢ is active, evtᵢ is a boolean variable representing that event of tᵢ is activated and 
ɾdᵢ is a predicate which evaluates true if guard condition of tᵢ is satisfied.

The post-condition of tᵢ is a conjunction of the following three conditions: (1) target state of tᵢ is active, (2) event of tᵢ is not activated, and (3) operation by the action of tᵢ is performed. Hence post(tᵢ) is represented as the following boolean formula:

\[ post(tᵢ) ≡ tgtᵢ ∧ ¬evtᵢ ′ ∧ actᵢ ∧ ¬srcᵢ ′ \]

where tgtᵢ is a boolean variable representing that target state of tᵢ is active, actᵢ is a predicate which evaluates true if operation by action of tᵢ is performed. x' represents a value of variable represented by a symbol x in a state after a transition.

The condition for unchanged elements of tᵢ is represented as the following formula:

\[ inv(tᵢ) ≡ \bigwedge_{v ∈ U-change(tᵢ)} (v' = v), \]

where U-change(tᵢ) is a set of variables representing elements which do not change by tᵢ.

As stated above, act(tᵢ) is a conjunction of the above three formulas:

\[ act(tᵢ) ≡ pre(tᵢ) ∧ post(tᵢ) ∧ inv(tᵢ). \]

The boolean formula B representing transition relations of a state machine diagram is a disjunction of each act(tᵢ).

\[ B ≡ \bigvee_{tᵢ ∈ T} act(tᵢ). \]

B. Sequence diagram

The sequence diagram represents order relations of message processing. This paper only focuses on sequence diagram with asynchronous messages. As in the case of state machine diagram, message processing of a sequence diagram is expressed using boolean formulas.

An occurrence oᵢ₋₁ is O, which is the jth occurrence of lifeline lᵢ, is represented as a formula act(oᵢ₋₁) which evaluates true iff oᵢ₋₁ is executed. act(oᵢ₋₁) is a conjunction of the following three formulas: pre(oᵢ₋₁) representing pre-condition of oᵢ₋₁, post(oᵢ₋₁) representing post-condition of oᵢ₋₁, and inv(oᵢ₋₁) representing condition for unchanged elements of oᵢ₋₁. These three conditions are represented with boolean representations for sending and receiving occurrence.

1) Representation of sending occurrence: The pre-condition of sending occurrence oᵢ₋₁ is a conjunction of the following three conditions: (1) precedent occurrence oᵢ₋₁ is executed, (2) oᵢ₋₁ is not executed, and (3) message of oᵢ₋₁ is not activated. Hence pre(oᵢ₋₁) is represented as the following boolean formula:

\[ pre(oᵢ₋₁) ≡ oᵢ₋₁ ∧ \neg oᵢ₋₁ ∧ \neg mᵢ₋₁. \]

The post-condition of sending occurrence oᵢ₋₁ is a conjunction of the following conditions: (1) oᵢ₋₁ is executed, and (2) message of oᵢ₋₁ is activated. Hence post(oᵢ₋₁) is represented as the following boolean formula:

\[ post(oᵢ₋₁) ≡ oᵢ₋₁ ∧ mᵢ₋₁. \]

The condition for unchanged elements of oᵢ₋₁ is represented as the following formula:

\[ inv(oᵢ₋₁) ≡ \bigwedge_{o ∈ O \setminus oᵢ₋₁} σ' = o. \]

2) Representation of receiving occurrence: The pre-condition of receiving occurrence oᵢ₋₁ is a conjunction of the following three conditions: (1) precedent occurrence oᵢ₋₁ is executed, (2) oᵢ₋₁ is not executed, and (3) a sender of message oᵢ₋₁ (call it oⱼ₋₁) is executed, and (4) message of oᵢ₋₁ is activated. Hence pre(oᵢ₋₁) is represented as the following boolean formula:

\[ pre(oᵢ₋₁) ≡ oᵢ₋₁ ∧ \neg oᵢ₋₁ ∧ mᵢ₋₁. \]

The post-condition of receiving occurrence oᵢ₋₁ is a conjunction of the following conditions: (1) oᵢ₋₁ is executed, and (2) message of oᵢ₋₁ is not activated. Hence post(oᵢ₋₁) is represented as the following boolean formula:

\[ post(oᵢ₋₁) ≡ oᵢ₋₁ ∧ mᵢ₋₁. \]

The condition for unchanged elements of oᵢ₋₁ is represented as the following formula:

\[ inv(oᵢ₋₁) ≡ \bigwedge_{o ∈ O \setminus oᵢ₋₁} σ' = o. \]

As stated above, act(oᵢ₋₁) is a conjunction of the above three formulas:

\[ act(oᵢ₋₁) ≡ pre(oᵢ₋₁) ∧ post(oᵢ₋₁) ∧ inv(oᵢ₋₁). \]

The boolean formula S representing order relations of message processing of a sequence diagram is a disjunction of each act(oᵢ₋₁).

\[ S ≡ \bigvee_{oᵢ₋₁ ∈ O} act(oᵢ₋₁). \]

C. Combined fragment

1) alternative: Alternative combined fragment describes branching operation in a sequence diagram. Interactions in a sub-fragment are taken only when a guard condition of that sub-fragment is satisfied. Figure 2 showed an example of alternative combined fragment. Note that precedent occurrence oⱼ₋₂ is not o₂₋₁ but o₂₋₀. This is because either o₂₋₁ or o₂₋₂ is executed in this alternative combined fragment.

We modified pre-conditions of sending and receiving occurrences in and after an alternative combined fragment in order to represent that fragment with boolean expression. In the case of Figure 2, they are modified as follows. The pre-condition of sending occurrence o₁₋₁ is a conjunction of the following conditions: (1) o₁₋₀ is executed, (2) o₁₋₁ is not executed, (3)
$m_1$ is not activated, and (4) a boolean variable $x$ evaluates true. Hence $pre(o_{1,1})$ is represented as the formula:

$$pre(o_{1,1}) \equiv o_{1,0} \land \lnot o_{1,1} \land \lnot m_1 \land x.$$  

The pre-condition of $o_{2,2}$ is also represented as the formula:

$$pre(o_{2,2}) \equiv o_{2,0} \land \lnot o_{2,2} \land \lnot m_2 \land y.$$  

Fig. 2. An example of alternative combined fragment

The pre-condition of $o_{1,3}$ is a conjunction of the following conditions: (1) $o_{1,1}$ is executed and $x$ evaluates true or $o_{1,2}$ is executed and $y$ evaluates true or $o_{1,0}$ is executed and $x \lor y$ evaluates false, (2) $o_{1,3}$ is not executed, and (3) $m_3$ is not activated. Hence $pre(o_{1,3})$ is represented as the formula:

$$pre(o_{1,3}) \equiv (o_{1,1} \land x \lor o_{1,2} \lor y \lor o_{1,0} \land \lnot (x \lor y)) \land \lnot o_{1,3} \land \lnot m_3.$$  

The pre-condition of receiving occurrence $o_{2,1}$ is a conjunction of the following conditions: (1) $o_{2,0}$ is executed, (2) $o_{2,1}$ is not executed, (3) $o_{1,1}$ is executed, (4) $m_1$ is activated, and (5) $x$ evaluates true. Hence $pre(o_{2,1})$ is represented as the formula:

$$pre(o_{2,1}) \equiv o_{2,0} \land \lnot o_{2,1} \land o_{1,1} \land m_1 \land x.$$  

The pre-condition of $o_{1,2}$ and $o_{2,3}$ are represented as the formulas:

$$pre(o_{1,2}) \equiv o_{1,0} \land \lnot o_{1,2} \land o_{2,2} \land m_2 \land y,$$

$$pre(o_{2,3}) \equiv (o_{1,1} \land x \lor o_{1,2} \lor y \lor o_{1,0} \land \lnot (x \lor y)) \land \lnot o_{2,3} \land o_{1,3} \land m_3.$$  

2) option: Option combined fragment describes an optional operation in a sequence diagram. If a guard condition of a fragment is unsatisfied, interactions in it are not executed. Figure 3 showed an example of option combined fragment.

As in the case of alternative combined fragment, pre-conditions of occurrences in and after that fragment are modified. In Figure 3 the pre-condition of sending occurrence $o_{1,1}$ is a conjunction of the following conditions: (1) $o_{1,0}$ is executed, (2) $o_{1,1}$ is not executed, (3) $m_3$ is not activated, and (4) $x$ evaluates true. Hence $pre(o_{1,1})$ is represented as the formula:

$$pre(o_{1,1}) \equiv o_{1,0} \land \lnot o_{1,1} \land \lnot m_1 \land x.$$  

The pre-condition of $o_{2,1}$ is a conjunction of the following conditions: (1) $o_{1,1}$ is executed and $x$ evaluates true or $o_{1,0}$ is executed and $x$ evaluates false, (2) $o_{1,3}$ is not executed, and (3) $m_3$ is not activated. Hence $pre(o_{1,3})$ is represented as the formula:

$$pre(o_{1,3}) \equiv (o_{1,1} \land x \lor o_{1,0} \land \lnot y) \land \lnot o_{1,3} \land \lnot m_3.$$  

The pre-condition of receiving occurrence $o_{2,1}$ is a conjunction of the following conditions: (1) $o_{2,0}$ is executed, (2) $o_{2,1}$ is not executed, (3) $o_{1,1}$ is executed, (4) $m_1$ is activated, and (5) $x$ evaluates true. Hence $pre(o_{2,1})$ is represented as the formula:

$$pre(o_{2,1}) \equiv o_{2,0} \land \lnot o_{2,1} \land o_{1,1} \land m_1 \land x.$$  

The pre-condition of $o_{2,2}$ is represented as the formula:

$$pre(o_{2,2}) \equiv (o_{2,1} \land x \lor o_{2,0} \land \lnot x) \land \lnot o_{2,2} \land o_{1,2} \land m_2.$$  

3) parallel: Parallel combined fragment describes parallel operations in a sequence diagram. Interactions in subfragments are executed concurrently. Figure 4 showed an example of parallel combined fragment. Note that precedent occurrence of $o_{1,2}$ is $o_{1,0}$. This is because $m_1$ and $m_2$ are processed concurrently in this parallel combined fragment.

As in the case of alternative combined fragment, pre-conditions of occurrences in and after that fragment are modified. In Figure 4 the pre-condition of sending occurrence $o_{1,1}$ is a conjunction of the following conditions: (1) $o_{1,0}$ is executed, (2) $o_{1,1}$ is not executed, and (3) $m_3$ is not activated. Hence $pre(o_{1,1})$ is represented as the formula:

$$pre(o_{1,1}) \equiv o_{1,0} \land \lnot o_{1,1} \land \lnot m_1.$$  

The pre-condition of $o_{1,2}$ is represented as the formula:

$$pre(o_{1,2}) \equiv o_{1,0} \land \lnot o_{1,2} \land \lnot m_2.$$  

The pre-condition of $o_{1,3}$ is a conjunction of the following conditions: (1) all receiving occurrences in this fragment, that is, $o_{2,1}$ and $o_{2,2}$ are executed, (2) $o_{1,3}$ is not executed, and (3) $m_3$ is not activated. Hence $pre(o_{1,3})$ is represented as the formula:

$$pre(o_{1,3}) \equiv o_{2,1} \land o_{2,2} \land \lnot o_{1,3} \land \lnot m_3.$$  

The pre-condition of receiving occurrence $o_{2,1}$ is a conjunction of the following conditions: (1) $o_{2,0}$ is executed, (2) $o_{2,1}$
is not executed, (3) \( o_{1,1} \) is executed, and (4) \( m_1 \) is activated. Hence \( pre(o_{2,1}) \) is represented as the formula:

\[
pre(o_{2,1}) \equiv o_{2,0} \land \neg o_{2,1} \land o_{1,1} \land m_1.
\]

The pre-condition of \( o_{2,2} \) and \( o_{2,3} \) are represented as the formulas:

\[
pre(o_{2,2}) \equiv o_{2,0} \land \neg o_{2,2} \land o_{1,2} \land m_2,
\]

\[
pre(o_{2,3}) \equiv o_{2,1} \land \neg o_{2,3} \land o_{1,3} \land m_3.
\]

D. Input of SMV

SMV needs two inputs. One is a conjunction of boolean formulas \( B \) and \( S \), \( B \land S \). It models a software system as a transition system which satisfies transition relations in state machine diagrams and order relations of messages in a sequence diagram.

Another is a CTL formula expressing a property to be verified. If a sequence diagram and state machine diagrams of a software system are consistent, all occurrences in this sequence diagram.

\[EF\neg o_{1,0} \land EF\neg o_{1,1} \land EF\neg o_{1,2} \land EF\neg o_{1,3} \land \ldots\]

IV. APPLICATION RESULT

We applied the proposed method to state machine diagrams and a sequence diagram in Figure 5 and verified consistency between them by using SMV. In the system described by the state machine diagrams, either \( x \) or \( y \) becomes true nondeterministically. If \( x \) is true then \( Obj_2 \) sends message \( m_1 \) to \( Obj_2 \), else if \( y \) is true then \( Obj_2 \) sends message \( m_2 \) to \( Obj_3 \).

Figure 6 shows the results of consistency verification. Figure 6(a) indicates the CTL formula evaluates true and that consistency of the diagrams is satisfied. Figure 6(b) indicates the CTL formula evaluates false and that inconsistency between the sequence diagram in Figure 5(d) and state machine diagrams could be detected.

---

(b) consistency of Figure 5(a),(b) and (c)

\[EF\neg o_{1,0} \land EF\neg o_{1,1} \land EF\neg o_{1,2} \land EF\neg o_{1,3} \land \ldots\]

---

(b) consistency of Figure 5(a),(b) and (d)

---

V. CONCLUSION

In this paper, we proposed a method for verifying consistency of UML diagrams with combined fragments. This method can treat three types of combined fragments: alternative, option and parallel. We also confirmed that our method could verify consistency of state machine diagrams and a sequence diagram with a combined fragment.

For the future work, it is necessary to extend the proposed method to other type of combined fragments. In addition, handling synchronous messages is required in practice.

REFERENCES


