World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:4, No:11, 2010

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.

†Graduate School of Systems Engineering, Okayama Prefectural University, Kuboki 111, Soja-shi, Okayama, 719–1197 Japan (e-mail: kawakami, miyazaki@circuit.cse.oka-pu.ac.jp, {t-yokoga, amasaki, sato, hayase}@cse.oka-pu.ac.jp).

‡Kawasaki University of Medical Welfare, 288 Matsushima, Kurashiki-shi, Okayama 701-0193, Japan (e-mail: miyazaki@me.kawasaki-m.ac.jp).

Manuscript received September, 2010; revised October 31, 2010.

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 Obj_1 sends a message m_1 and Obj_2 receives it. However, sequence diagram in Figure 1(c) describes that Obj_2 sends m_1 and Obj_1 receives it. Thus behaviours of these diagrams are inconsistent.

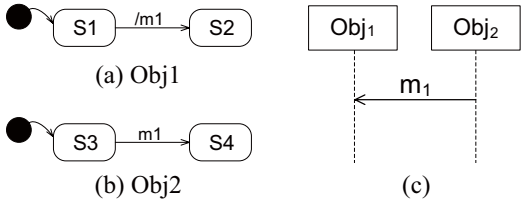


Fig. 1. An example of consistency violation

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: $pre(t_i)$ representing pre-condition of t_i, $post(t_i)$ representing post-condition of t_i, and $inv(t_i)$ 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 $pre(t_i)$ is represented as the following boolean formula:

$$pre(t_i) \equiv \mathsf{src_i} \wedge \mathsf{evt_i} \wedge grd_i,$$

where $\mathsf{src_i}$ is a boolean variable representing that source state of t_i is active, $\mathsf{evt_i}$ is a boolean variable representing that event of t_i is activated and grd_i is a predicate which evaluates true if guard condition of t_i is satisfied.

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

$$post(t_i) \equiv \mathsf{tgt_i} \wedge \neg \mathsf{evt_i}' \wedge act_i \wedge \neg \mathsf{src_i}',$$

where $\mathsf{tgt_i}$ is a boolean variable representing that target state of t_i is active, act_i is a predicate which evaluates true if operation by action of t_i 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 t_i is represented as the following formula:

$$inv(t_i) \equiv \bigwedge_{\mathsf{v} \in Unchange(t_i)} (\mathsf{v}' = \mathsf{v}),$$

where $Unchange(t_i)$ is a set of variables representing elements which do not change by t_i.

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

$$act(t_i) \equiv pre(t_i) \wedge post(t_i) \wedge inv(t_i).$$

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

$$B \equiv \bigvee_{t_i \in T} act(t_i).$$

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 $o_{i\cdot j} \in O$, which is the jth occurrence of lifeline l_i, is represented as a formula $act(o_{i\cdot j})$ which evaluates true iff $o_{i\cdot j}$ is executed. $act(o_{i\cdot j})$ is a conjunction of the following three formulas: $pre(o_{i\cdot j})$ representing pre-condition of $o_{i\cdot j}$, $post(o_{i\cdot j})$ representing post-condition of $o_{i\cdot j}$, and $inv(o_{i\cdot j})$ representing condition for unchanged elements of $o_{i\cdot j}$. These three conditions are represented with boolean representations for sending and receiving occurrence.

1) Representation of sending occurrence: The pre-condition of sending occurrence $o_{i\cdot j}$ is a conjunction of the following three conditions: (1) precedent occurrence $o_{i\cdot j-1}$ is executed, (2) $o_{i\cdot j}$ is not executed, and (3) message of $o_{i\cdot j}$ is not activated. Hence $pre(o_{i\cdot j})$ is represented as the following boolean formula:

$$pre(o_{i\cdot j}) \equiv \mathsf{o_{i\cdot j-1}} \wedge \neg \mathsf{o_{i\cdot j}} \wedge \neg \mathsf{m_{i\cdot j}},$$

where $\mathsf{o_{i\cdot j}}$ is a boolean variable representing that occurrence $o_{i\cdot j}$ is executed and $\mathsf{m_{i\cdot j}}$ is a boolean variable representing that message of $o_{i\cdot j}$ is activated.

The post-condition of sending occurrence $o_{i\cdot j}$ is a conjunction of the following conditions: (1) $o_{i\cdot j}$ is executed, and (2) message of $o_{i\cdot j}$ is activated. Hence $post(o_{i\cdot j})$ is represented as the following boolean formula:

$$post(o_{i\cdot j}) \equiv \mathsf{o_{i\cdot j}}' \wedge \mathsf{m_{i\cdot j}}'.$$

The condition for unchanged elements of $o_{i\cdot j}$ is represented as the following formula:

$$inv(o_{i\cdot j}) \equiv \bigwedge_{o \in O \setminus o_{i\cdot j}} \mathsf{o}' = \mathsf{o}.$$

2) Representation of receiving occurrence: The pre-condition of receiving occurrence $o_{i\cdot j}$ is a conjunction of the following three conditions: (1) precedent occurrence $o_{i\cdot j-1}$ is executed, (2) $o_{i\cdot j}$ is not executed, (3) a sender of message of $o_{i\cdot j}$ (call it $o_{k\cdot l}$) is executed, and (4) message of $o_{i\cdot j}$ is activated. Hence $pre(o_{i\cdot j})$ is represented as the following boolean formula:

$$pre(o_{i\cdot j}) \equiv \mathsf{o_{i\cdot j-1}} \wedge \neg \mathsf{o_{i\cdot j}} \wedge \mathsf{o_{k\cdot l}} \wedge \mathsf{m_{i\cdot j}}.$$

The post-condition of receiving occurrence $o_{i\cdot j}$ is a conjunction of the following conditions: (1) $o_{i\cdot j}$ is executed, and (2) message of $o_{i\cdot j}$ is not activated. Hence $post(o_{i\cdot j})$ is represented as the following boolean formula:

$$post(o_{i\cdot j}) \equiv \mathsf{o_{i\cdot j}}' \wedge \neg \mathsf{m_{i\cdot j}}'.$$

The condition for unchanged elements of $o_{i\cdot j}$ is represented as the following formula:

$$inv(o_{i\cdot j}) \equiv \bigwedge_{o \in O \setminus o_{i\cdot j}} \mathsf{o}' = \mathsf{o}.$$

As stated above, $act(o_{i\cdot j})$ is a conjunction of the above three formulas:

$$act(o_{i\cdot j}) \equiv pre(o_{i\cdot j}) \wedge post(o_{i\cdot j}) \wedge inv(o_{i\cdot j}).$$

The boolean formula S representing order relations of message processing of a sequence diagram is a disjunction of each $act(o_{i\cdot j})$.

$$S \equiv \bigvee_{o_{i\cdot j} \in O} act(o_{i\cdot j}).$$

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 of $o_{2.2}$ is not $o_{2.1}$ but $o_{2.0}$. This is because either $o_{2.1}$ or $o_{2.2}$ 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 $o_{1.1}$ is a conjunction of the following conditions: (1) $o_{1.0}$ is executed, (2) $o_{1.1}$ is not executed, (3)

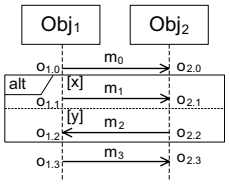


Fig. 2. An example of alternative combined fragment

m_1 is not activated, and (4) a boolean variable x evaluates true. Hence $pre(o_{1.1})$ is represented as the formula:

$$pre(o_{1.1}) \equiv \mathsf{o_{1.0}} \wedge \neg \mathsf{o_{1.1}} \wedge \neg \mathsf{m_1} \wedge \mathsf{x}.$$

The pre-condition of $o_{2.2}$ is also represented as the formula:

$$pre(o_{2.2}) \equiv \mathsf{o_{2.0}} \wedge \neg \mathsf{o_{2.2}} \wedge \neg \mathsf{m_2} \wedge \mathsf{y}.$$

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 $x \vee y$ 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:

$$\begin{aligned} pre(o_{1.3}) \quad \equiv \quad & (\mathsf{o_{1.1}} \wedge \mathsf{x} \vee \mathsf{o_{1.2}} \wedge \mathsf{y} \vee \mathsf{o_{1.0}} \wedge \neg(\mathsf{x} \vee \mathsf{y})) \\ & \wedge \neg \mathsf{o_{1.3}} \wedge \neg \mathsf{m_3}. \end{aligned}$$

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:

$$pre(o_{2.1}) \equiv \mathsf{o_{2.0}} \wedge \neg \mathsf{o_{2.1}} \wedge \mathsf{o_{1.1}} \wedge \mathsf{m_1} \wedge \mathsf{x}.$$

The pre-condition of $o_{1.2}$ and $o_{2.3}$ are represented as the formulas:

$$\begin{aligned} pre(o_{1.2}) \quad \equiv \quad & \mathsf{o_{1.0}} \wedge \neg \mathsf{o_{1.2}} \wedge \mathsf{o_{2.2}} \wedge \mathsf{m_2} \wedge \mathsf{y}, \\ pre(o_{2.3}) \quad \equiv \quad & (\mathsf{o_{1.1}} \wedge \mathsf{x} \vee \mathsf{o_{1.2}} \wedge \mathsf{y} \vee \mathsf{o_{1.0}} \wedge \neg(\mathsf{x} \vee \mathsf{y})) \\ & \wedge \neg \mathsf{o_{2.3}} \wedge \mathsf{o_{1.3}} \wedge \mathsf{m_3}. \end{aligned}$$

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.

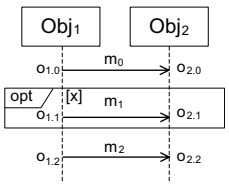


Fig. 3. An example of option combined fragment

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_1 is not activated, and (4) x evaluates true. Hence $pre(o_{1.1})$ is represented as the formula:

$$pre(o_{1.1}) \equiv \mathsf{o_{1.0}} \wedge \neg \mathsf{o_{1.1}} \wedge \neg \mathsf{m_1} \wedge \mathsf{x}.$$

The pre-condition of $o_{1.2}$ 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:

$$pre(o_{1.3}) \equiv (\mathsf{o_{1.1}} \wedge \mathsf{x} \vee \mathsf{o_{1.0}} \wedge \neg \mathsf{y}) \wedge \neg \mathsf{o_{1.3}} \wedge \neg \mathsf{m_3}.$$

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:

$$pre(o_{2.1}) \equiv \mathsf{o_{2.0}} \wedge \neg \mathsf{o_{2.1}} \wedge \mathsf{o_{1.1}} \wedge \mathsf{m_1} \wedge \mathsf{x}.$$

The pre-condition of $o_{2.2}$ is represented as the formula:

$$pre(o_{2.2}) \equiv (\mathsf{o_{2.1}} \wedge \mathsf{x} \vee \mathsf{o_{2.0}} \wedge \neg \mathsf{x}) \wedge \neg \mathsf{o_{2.2}} \wedge \mathsf{o_{1.2}} \wedge \mathsf{m_2}.$$

3) parallel: Parallel combined fragment describes parallel operations in a sequence diagram. Interactions in sub-fragments 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.

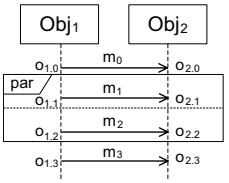


Fig. 4. An example of parallel combined fragment

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_1 is not activated. Hence $pre(o_{1.1})$ is represented as the formula:

$$pre(o_{1.1}) \equiv \mathsf{o_{1.0}} \wedge \neg \mathsf{o_{1.1}} \wedge \neg \mathsf{m_1}.$$

The pre-condition of $o_{1.2}$ is represented as the formula:

$$pre(o_{1.1}) \equiv \mathsf{o_{1.0}} \wedge \neg \mathsf{o_{1.2}} \wedge \neg \mathsf{m_2}.$$

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:

$$pre(o_{1.3}) \equiv \mathsf{o_{2.1}} \wedge \mathsf{o_{2.2}} \wedge \neg \mathsf{o_{1.3}} \wedge \neg \mathsf{m_3}.$$

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:

$$pre(o_{2.1}) \equiv \mathsf{o}_{2.0} \wedge \neg \mathsf{o}_{2.1} \wedge \mathsf{o}_{1.1} \wedge \mathsf{m}_1.$$

The pre-condition of $o_{2.2}$ and $o_{2.3}$ are represented as the formulas:

$$\begin{aligned} pre(o_{2.2}) &\equiv \mathsf{o}_{2.0} \wedge \neg \mathsf{o}_{2.2} \wedge \mathsf{o}_{1.2} \wedge \mathsf{m}_2, \\ pre(o_{2.3}) &\equiv \mathsf{o}_{2.1} \wedge \mathsf{o}_{2.2} \wedge \neg \mathsf{o}_{2.3} \wedge \mathsf{o}_{1.3} \wedge \mathsf{m}_3. \end{aligned}$$

D. Input of SMV

SMV needs two inputs. One is a conjunction of boolean formulas B and S, $B \wedge S$. 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.

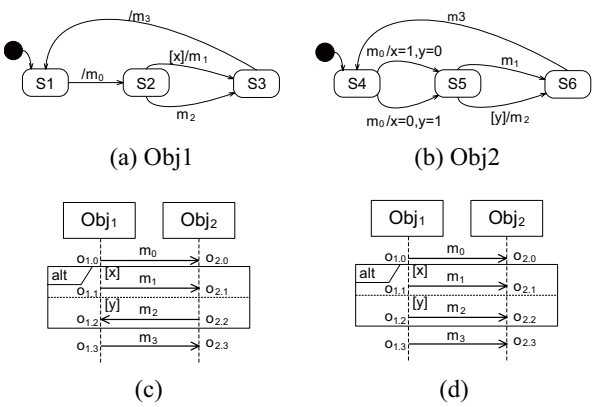
Another is a CTL formula expressing a property to be verified. If a sequence diagram and state machine diagrams of a software system are consistent, all occurrences in this sequence diagram can be executed. This property is expressed as the following CTL formula:

$$\bigwedge_{o_{i \cdot j} \in O} EF \mathsf{o}_{\mathsf{i \cdot j}}.$$

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_1 sends message m_1 to Obj_2, else if y is true then Obj_2 sends message m_2 to Obj_1.

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.



(a) Obj1 (b) Obj2

(c) (d)

Fig. 5. Examples of UML diagrams

```
-- specification EF o1_0 & EF o1_1 & EF o1_2 & EF o1_3 & ...
is true

resources used:
processor time: 0 s,
BDD nodes allocated: 10012
Bytes allocated: 1171452
BDD nodes representing transition relation: 385 + 1
reachable states: 132 (2^7.04439) out of 147456 (2^17.1699)
```

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

```
-- specification EF o1_0 & EF o1_1 & EF o1_2 & EF o1_3 & ...
is false
-- as demonstrated by the following execution sequence
state 1.1:
...


resources used:
processor time: 0 s,
BDD nodes allocated: 9787
Bytes allocated: 1171452
BDD nodes representing transition relation: 360 + 1
reachable states: 13 (2^3.70044) out of 147456 (2^17.1699)
```

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

Fig. 6. Results of the consistency verification by SMV

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

[1] O. M. Group, *Unified Modeling Language*. Object Management Group, 2001, http://www.uml.org.

[2] S. Bernardi, S. Donatelli, and J. Merseguer, "From uml sequence diagrams and statecharts to analysable petrinet models," in *Workshop on Software and Performance*, 2002, pp. 35–45.

[3] B. Litvak, S. S. Tyszberowicz, and A. Yehudai, "Behavioral consistency validation of UML diagrams," in *SEFM*. IEEE Computer Society, 2003, pp. 118–125.

[4] T. Schäfer, A. Knapp, and S. Merz, "Model checking uml state machines and collaborations," *Electr. Notes Theor. Comput. Sci.*, vol. 55, no. 3, 2001.

[5] X. Zhao, Q. Long, and Z. Qiu, "Model checking dynamic UML consistency," in *ICFEM*, ser. Lecture Notes in Computer Science, Z. Liu and J. He, Eds., vol. 4260. Springer, 2006, pp. 440–459.

[6] S. Harada, T. Yokogawa, H. Miyazaki, Y. Sato, and M. Hayase, "A tool support for verifying consistency between UML diagrams by SMV," in *ITC-CSCC*, 2009, pp. 897–900.

[7] K. McMillan, *Symbolic Model Checking*. Kluwer Academic, 1993.