Model Checking for Aspect-Oriented Software Evolution

WUTTIPONG RUANTHONG AND PORNSIRI MUENCHAISRI
Department of Computer Engineering, Faculty of Engineering
Chulalongkorn University
254 Phyathai Road, Patumwan, Bangkok
THAILAND

Abstract: Model checking is the verification approach for proving a satisfaction of desired properties on a finite state system model. Whenever a new feature (aspect) which is developed as a separated unit is composed to the original software for evolving to the next generation, the properties which held on the model of the original software should be re-checked for their preservation. The re-verification of those properties with the traditional method is impractical because the state space of that software model is increased after the evolution. We use model checking for verifying the evolving software model based on the aspect-oriented concepts. The proposed technique for the preservation checking called *"certainty-aware technique"* can reduce time and state space in the process of preservation checking. Therefore, the verification process of model checker which utilizes our technique can be completed faster than the verification process of traditional model checker.

Key-Words: Software Algorithm, Software Verification, Model Checking, Software Evolution and Aspect-Oriented Software Development

1 Introduction

Aspect-oriented concepts are useful for facilitating a process of software maintenance and software evolution because a software complexity can be managed by a decomposition mechanism which separates software into manageable units [6]. Each separated unit is a part of software that is responsible for a specific requirement. In the process of software evolution, the separated units can be separately developed and incrementally added to the original software as the additional features. There are many approaches which implement the aspect-oriented concepts, such as Collaboration-based Design [7], MDSOC [3], AOP [4] and AHEAD [5]. Although these dominant approaches have their own distinctive features, but there is a commonality in their main ideas at the decomposition and composition mechanism. The separated units are called differently by their different approach, e.g. "concerns", "features" or "collaborations", but the term we used equivalently in this paper is "aspects".

Since the model of original software verified by the model checking approach [2] is incrementally changed after the addition of aspects, the properties which held on that model must be re-checked for their preservation. We call this re-checking process as *"preservation checking"*. The preservation checking with the traditional method is more complex than the previous verification because the state space of software model is increased after the evolution.

The contribution of this paper is the *"certainty-aware technique"* which addresses the above problem. The proposed technique which consists of two processes, *certainty analysis* and *re-verification*, can be generally applied to any evolving forms of software model. In the evolution process, the aspect model may either introduce or remove some elements of the base model. Since the computation paths of base model are changed, properties which held on that base model might be uncertain. The certainty analysis process is operated for checking the certainty of those properties. If some sub-formulas of those properties are considered as uncertain, then they must be re-verified in the re-verification process. We demonstrate that our technique can reduce the time and state space of preservation checking.

Section 2 refers to some related works. The basic models for aspect-oriented software are defined in section 3. Section 4 details the proposed technique. Section 5 demonstrates our technique with a simple example. Section 6 issues the idea of property evolution. The conclusion is in section 7.

2 Related Works

Fisler and Krishnamurthi [8] originally proposed the preservation checking technique for collaboration-based software design (equally well to aspect-oriented form). They utilized a *"compositional reasoning technique"* for confirming the preservation of properties when the base collaboration and the extension collaboration were composed together. In their basic models, the interface states were used as

This work was supported by "Chulalongkorn-Industry Linkage Research Fund".

the joints for the composition of both collaborations. The preservation constraints were derived from sub-formulas in the labels of base collaboration's interface states. The modular verification was performed on the extension collaboration. The properties were preserved if sub-formulas in the labels of extension collaboration's interface states were consistent to the preservation constraints. The complexity of the preservation checking was performed within a scope of extension collaboration rather than the entire composite system.

Later, Thang and Katayama [9] enhanced the basic model of original work for a more general of the collaboration-based software. They showed the weakness of the original work's technique with the circumstances that the compositional reasoning cannot correctly apply but the concrete solution for that weakness was not proposed.

3 Basic Models for Aspect-Oriented Software

We define three basic models (the base model, the aspect model and the composite model) within definitions 1 to 3 as a representation of aspect-oriented software. Our basic models have a key difference from [8, 9] that the interface states are not explicitly presented for the composition of the base model and the additional aspect. The aspect model can either introduce some states and/or transitions to the base model or remove them from the base model while the composition with interface states can be only applied for the addition of individual aspect to the base model. In this paper, we present only the basic models for a system with single actor.

Definition 1: The base model $\mathcal{M} = \langle S, \Sigma, \Delta, s_0, \mathcal{R}, \mathcal{L} \rangle$ where S is a set of states in $\mathcal{M}$, Σ is a set of atomic propositions for input events, Δ is a set of all sub-formulas of the verified properties, $s_0 \in S$ is an initial state of $\mathcal{M}$, $\mathcal{R} \subseteq S \times \mathcal{PL}(\Sigma) \times S$ is the set of transitions between two states (where $\mathcal{PL}(\Sigma)$ denotes the set of propositions expressed over Σ) and $\mathcal{L}: S \rightarrow 2^{\Delta}$ indicates the set of sub-formulas that are true on each state.

Definition 2: The aspect model $\mathcal{A} = \langle \mathcal{M}, S^+, \Sigma^+, \Delta^+, s^+_{opt}, \mathcal{R}^+, \mathcal{L}^+, \alpha, \beta \rangle$ where $\mathcal{M}$ is a specific base model that this aspect applies to, S^+ is a set of additional states, Σ^+ is a set of additional atomic propositions for input events, Δ^+ is a set of additional sub-formulas, $s^+ \in (S \cup S^+)$ is a new initial state (s^+ is an optional, if s^+ is undefined then s_0 of $\mathcal{M}$ still be an initial state), $\mathcal{R}^+ \subseteq ((S \cup S^+) \times \mathcal{PL}(\Sigma \cup \Sigma^+) \times (S \cup S^+) - \mathcal{R})$ is the set of additional transitions, $\mathcal{L}^+: S^+ \rightarrow 2^{\Delta \cup \Delta^+}$ indicates the set of sub-formulas that are true on each additional state, $\alpha \subseteq S$ is a set of states in the base model and $\beta \subseteq \mathcal{R}$ is a set of transitions in the base model that will be removed from the base model after the composition.

Definition 3: A composite model $\mathcal{M}' = \langle S', \Sigma', \Delta', s', \mathcal{R}', \mathcal{L}' \rangle$ is composed from the base model $\mathcal{M}$ and the additional aspect $\mathcal{A}$, denoted $\mathcal{M}' = \mathcal{A}(\mathcal{M})$, where $S' = (S \cup S^+) - \alpha, \Sigma' = \Sigma \cup \Sigma^+, \Delta' = \Delta \cup \Delta^+$, $s' = s^+$ if s^+ is defined in $\mathcal{A}$ or $s' = s_0$ if s^+ is undefined in $\mathcal{A}$, $\mathcal{R}' = (\mathcal{R} \cup \mathcal{R}^+) - \beta$ and $\mathcal{L}'(s) = \mathcal{L}(s)$ for $\forall s \in S$ and $\mathcal{L}'(t) = \mathcal{L}^+(t)$ for $\forall t \in S^+$.

4 The Preservation Checking with Certainty-Aware Technique

After the composition, sub-formulas which are true on each state of base model might be uncertain because of the following two conditions. First, sub-formulas specified with a path quantifier are uncertain whenever some computation paths are changed. Second, sub-formula that its truth value depends on the truth value of other sub-formulas is also uncertain if some dependent sub-formula is uncertain.

We should check the certainty of all sub-formulas in the label of each state after the composition. If those sub-formulas are certainly true on that state, then we can conclude that the property is preserved. However, if some sub-formulas are uncertain, then a re-verification of those sub-formulas must be performed for evaluating their actual truth values. Our technique consists of two processes: *certainty analysis* and *re-verification* which are detailed in sections 4.1 and 4.2 respectively.

4.1 Certainty analysis

We represent the dependent relation of sub-formulas as the graph called "*Sub-formulas Dependence Graph* (SDG)" in definition 4.

Definition 4: Sub-formulas Dependence Graph (SDG) is $\langle v, d \rangle$ where v is the set of sub-formulas that are true on any states of base model $\mathcal{M}$ and $d \subseteq v \times v$ is the dependent relation of sub-formulas in the set v. The set v and the relation d are symbolically defined as follows: $v = \{f_s \bullet (f, s) \in \Delta \times S_{\mathcal{M}} \mid f \in \mathcal{L}(s)\}$ and $d = \{(f_s, f'_{s'}) \bullet (f_s, f'_{s'}) \in v \times v \mid \mathcal{V}(f,s) \rightarrow \mathcal{V}(f',s')\}$ where $\mathcal{V}: \Delta \times S_{\mathcal{M}} \rightarrow \{\text{TRUE, FALSE}\}$ is represented for the verification algorithm which is evaluating the sub-formula f on state s. This algorithm can be implemented with a recursive function-call. The function-call is symbolized as $\mathcal{V}(f,s) \rightarrow \mathcal{V}(f',s')$ denoted for the dependency of f_s to $f'_{s'}$ because the verification of f_s requires the truth values from the verification of $f'_{s'}$.

After the composition of the base model and the aspect model, the certainty of each sub-formula in the label of any states can be analyzed from the out-going transitions which are added to or removed from those states. Let γ is the set of states in base model that some out-going transitions are removed from those states, γ^* is the set of states in base model that all out-going transitions are removed from those states and δ is the set of states in base model that some out-going transitions are added to those states. The decision rule for certainty analysis is specified in definition 5. This decision rule is used for classifying sub-formulas in a label of each state into the set of certain sub-formulas and the set of uncertain sub-formulas.

Definition 5: ξ is the set of sub-formulas which are considered as uncertain (*Note*: $\xi \subseteq v$). Sub-formula $f_s \in v$ is considered as uncertain, then $f_s \in \xi$, if one of the following conditions is true:

1.) **case** f is a sub-formula specified with a CTL operator (e.g. EX, AX, EU or AU) and $s \in \gamma^*$.

2.) **case** f=EX(f')

$s \in \gamma \vee \forall f'_t \in Succ(f_s)\ [f'_t \in \xi]$.

3.) **case** f=AX(f')

$s \in \delta \vee \exists f'_t \in Succ(f_s)\ [f'_t \in \xi]$.

4.) **case** f=E(f'Uf'')

$(f_t \in Succ(f_s) \wedge s \in \gamma) \vee (f'_s \in Succ(f_s) \wedge f'_s \in \xi) \vee (f''_s \in Succ(f_s) \wedge f''_s \in \xi) \vee (\forall f'_t \in Succ(f_s)\ [f'_t \in \xi])$.

5.) **case** f=A(f'Uf'')

$(f_t \in Succ(f_s) \wedge s \in \delta) \vee (f'_s \in Succ(f_s) \wedge f'_s \in \xi) \vee (f''_s \in Succ(f_s) \wedge f''_s \in \xi) \vee (\exists f'_t \in Succ(f_s)\ [f'_t \in \xi])$.

Note: t is an immediate successor of state s in state-transition graph and $Succ: v \rightarrow 2^v$ indicates the set of all immediate successors of each node in SDG.

After the analysis, if f is certain on state s ($f_s \notin \xi$), then f is preserved on state s because f are still considered as the member of its label. Otherwise, f is uncertain on state s ($f_s \in \xi$), the re-verification of f on state s must be performed for its actual truth value.

4.2 Re-verification

The state space of base model can be reduced before the composition with the additional aspect. We classify states in the base model into the following three sets: the set of re-verified states S_{rv} , the set of border states S_{bd} and the set of reducible states S_{rd}. The re-verified state is a state that there exists an uncertain sub-formula in its label. The border state is a state that all sub-formulas in its label are certain but there exists a transition from the re-verified states to it. The reducible state is a state that all sub-formulas in its label are certain and there is no connection from the re-verified states. These sets can be symbolized as follows: $S_{rv} = \{s \bullet s \in S_{\mathcal{M}} \mid \exists f \in \mathcal{L}(s)[f_s \in \xi]\}$, $S_{bd} = \{s \bullet s \in S_{\mathcal{M}} \mid \forall f \in \mathcal{L}(s)[f_s \notin \xi] \wedge \exists t \in S_{rv}\ [(t, *, s) \in \mathcal{R}_{\mathcal{M}}]\}$ (where $*$ is unimportant) and $S_{rd} = S_{\mathcal{M}} - (S_{rv} \cup S_{bd})$. States in the set S_{rd} and their connecting transitions can be reduced. Thus, there are only states in the sets S_{rv}, S_{bd} and $S^+_{\mathcal{A}}$ that must be generated.

On the re-verification process, Sub-formulas that must be checked on the additional states can be considered as uncertain because their truth values are not yet known. Therefore, the re-verification algorithm must check them together with sub-formulas in set ξ. We define sub-formulas that must be re-verified on the additional states as the members of set u, symbolized as $u = \{f_s \bullet (f, s) \in \Delta \times S^+_{\mathcal{A}} \mid f$ must be re-verified on state s by the re-verification process$\}$. Definition 6 specifies sub-formulas that have to be re-verified and sub-formulas that need not be re-verified.

Definition 6: $\Upsilon = u \cup \xi$ is the set of re-verified sub-formulas. f_s is considered as a re-verified sub-formula, denoted $\{f_s\}$, whenever f_s is the member of set Υ, otherwise f_s is a certain sub-formula, denoted $\lfloor f_s \rfloor$, which need not be re-verified.

Definition 7 is the re-verification algorithm which is represented as a recursively defined function. The operation of this algorithm is similar to the original algorithm [1] except that this algorithm concerns about the certainty of each re-verified sub-formula. Suppose sub-formula f_s is re-verified, then we can infer after the termination of re-verification process with the following condition: f is preserved on state s of $\mathcal{M}'$, denoted $\mathcal{M}', s \models f$, if and only if $\lfloor f_s \rfloor$ and $f \in \mathcal{L}'(s)$.

Definition 7: $\mathcal{RV}: \Delta \times S'_{\mathcal{M}'} \rightarrow \{\text{TRUE, FALSE}\}$ is the representation of re-verification algorithm. This function evaluates the truth value of sub-formula $f \in \Delta$ on state $s \in S'_{\mathcal{M}'}$. We define this algorithm as a recursively defined function with the following description:

case $\lfloor f_s \rfloor$:

if $f \in \mathcal{L}'(s)$ then

$\mathcal{RV}(f,s)$ = TRUE, otherwise $\mathcal{RV}(f,s)$ = FALSE.

case $\{f_s\}$:

if $\mathcal{T}(f)$ = NO and $Sub(f) \in \mathcal{L}'(s)$ then

$\mathcal{RV}(f,s)$ = TRUE, otherwise $\mathcal{RV}(f,s)$ = FALSE.

if $\mathcal{T}(f)$ = EX ;where f is EX(f_0) then

$\mathcal{RV}(f,s) = \exists t\ [\mathcal{RV}(f_0,t)$=TRUE].

if $\mathcal{T}(f)$ = AX ;where f is AX(f_0) then

$\mathcal{RV}(f,s) = \forall t\ [\mathcal{RV}(f_0,t)$=TRUE].

if $\mathcal{T}(f)$ = EU ;where f is E(f_1Uf_2) then

$\mathcal{RV}(f,s)$ = ($\mathcal{RV}(f_2,s)$=TRUE) $\vee$ ($\mathcal{RV}(f_1,s)$ = TRUE $\wedge \exists t$ [$\mathcal{RV}(f,t)$=TRUE]).

if $\mathcal{T}(f)$ = AU ;where f is A(f_1Uf_2) then

$\mathcal{RV}(f,s)$ = ($\mathcal{RV}(f_2,s)$=TRUE) $\vee$ ($\mathcal{RV}(f_1,s)$ = TRUE $\wedge \forall t$ [$\mathcal{RV}(f,t)$=TRUE]).

Note:

- $t \in \mathcal{S}'_{\mathcal{M}'}$ is an immediate successor of state s in the state-transition graph.

- Sub: $\Delta \rightarrow 2^{\Delta}$ indicates all sub-formulas of an assigned formula.

- $\mathcal{T}$:$\Delta \rightarrow$ {NO, EX, AX, EU, AU} maps an assigned sub-formula to its corresponding type of CTL operator, such as EX, AX, EU or AU. If the assigned sub-formula is a proposition without CTL operator, then the NO is mapped.

- Before the starting of re-verification process, f is removed from $\mathcal{L}'(s)$ if f_s is $\{f_s\}$.

- After the completion of re-verification process, $\{f_s\}$ is changed to $|f_s|$ and f is added to $\mathcal{L}'(s)$ if the evaluated result of $\mathcal{RV}(f,s)$ is TRUE.

4.3 Dealing with the circular re-verification

The re-verification process of algorithm in definition 7 cannot terminate if the circular function-call occur during the re-verification. For instance, let f is uncertain on state s, denoted as $\{f_s\}$. Suppose the re-verification of $\{f_s\}$ requires the evaluated result from the re-verification of $\{f_t\}$, symbolized as $\mathcal{RV}(f,s) \rightarrow \mathcal{RV}(f,t)$, where t is the additional state that is introduced as the immediate successor of s but the re-verification of $\{f_t\}$ also requires the evaluated result from the re-verification of $\{f_s\}$, $\mathcal{RV}(f,t) \rightarrow \mathcal{RV}(f,s)$, then the circular function-call occurs.

We address this problem by the following idea. First, we insert a method for detecting a circular function-call into the re-verification algorithm. The circular function-call will be immediately terminated when it is detected. Then, we define the decision rules for determining the actual re-verification result which can be inferred from the amount of detected circular function-calls. The characteristic of circular function-call is specified in definition 8. The decision rules used for determining the re-verification result are derived from theorems 1 and 2. The proofs of these theorems are presented in Appendix.

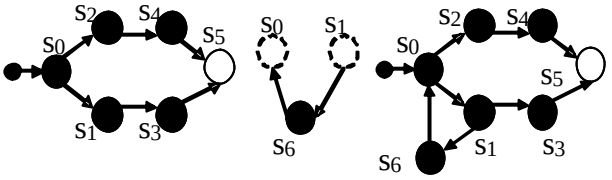
Definition 8: $\mathcal{W}$ is a relation of re-verified sub-formulas in the re-verification process. This relation can be symbolized as follow: $\mathcal{W}$ = {(f_s, $f'_{s'}$) • (f_s,$f'_{s'}$) $\in \Upsilon \times \Upsilon$ | $\mathcal{RV}(f,s) \rightarrow \mathcal{RV}(f',s')$}. The sequence of the re-verified sub-formulas can be enumerated as ($f0_{s0}$, $f1_{s1}$), ($f1_{s1}$,$f2_{s2}$), …, (fn-1$_{sn-1}$, fn$_{sn}$) where $\forall i$[$1 \le i \le n \Rightarrow$ (fi-1$_{si-1}$,fi_{si})$\in \mathcal{W}$]. This sequence is called a re-verification path from $f0_{s0}$ to fn$_{sn}$ and it can be simply shown in the form of $f0_{s0}$, $f1_{s1}$, …, fn-1$_{sn-1}$, fn$_{sn}$. The re-verification path that the first sub-formula and the last sub-formula are the same sub-formula ($f0_{s0}$ = fn$_{sn}$) is called a re-verification cycle.

Theorem 1: Let f is **A**(f_1**U**f_2), if there exists a re-verification cycle starting from f_{s0} which can be enumerated as f_{s0}, f_{s1}, f_{s2}, f_{s3}, …, f_{sn-1}, f_{sn}, f_{s0} then $\mathcal{RV}(f,s_0)$ = FALSE.

Theorem 2: Let f is **E**(f_1**U**f_2), if there are all re-verification cycles starting from f_{s0} which can be enumerated as f_{s0}, f_{s1}, f_{s2}, f_{s3}, …, f_{sn-1}, f_{sn}, f_{s0} then $\mathcal{RV}(f,s_0)$ = FALSE.

5 Example

We exemplify our verification technique by the example in figure 1. The base model $\mathcal{M}$ in figure 1(a) is evolved to the composite model $\mathcal{M}'$ in figure 1(c) by the composition with additional aspect $\mathcal{A}$ in figure 1(b). Sub-formulas "black" is true on state with black color and sub-formula "white" is true on state with white color. The property **A**(black**U**white) holds in the base model, denoted $\mathcal{M},s_0 \models$ **A**(black**U**white).



(a) The base model $\mathcal{M}$. (b) The additional aspect $\mathcal{A}$. (c) The evolving model $\mathcal{M}'$.

Figure 1. The evolution with aspect-oriented concepts.

The SDG of this property is presented in figure 2. From the certainty analysis, we know that the sub-formula **A**(black**U**white)$_{s1}$ is uncertain after the evolution because there is an out-going transition added to state s_1 and the sub-formula **A**(black**U** white)$_{s0}$ is also uncertain because it depends on the truth value of **A**(black**U**white)$_{s1}$ (from the condition 5 of definition 5). The dashed nodes in SDG indicate uncertain sub-formulas which must be re-verified.

The state space of evolving model should be reduced before the re-verification. Considering the base model, since the **A**(black**U**white)$_{s0}$ and the **A**(black**U**white)$_{s1}$ are uncertain sub-formulas then states s_0,s_1 are the re-verified states (s_0,$s_1 \in \mathcal{S}_{rv}$). States s_2,s_3 are the border states (s_2,$s_3 \in \mathcal{S}_{bd}$) because all sub-formulas in their labels are certain and there are transitions from states s_0,s_1 connecting to them. Thus, states s_4,s_5 can be reduced in this circumstance.

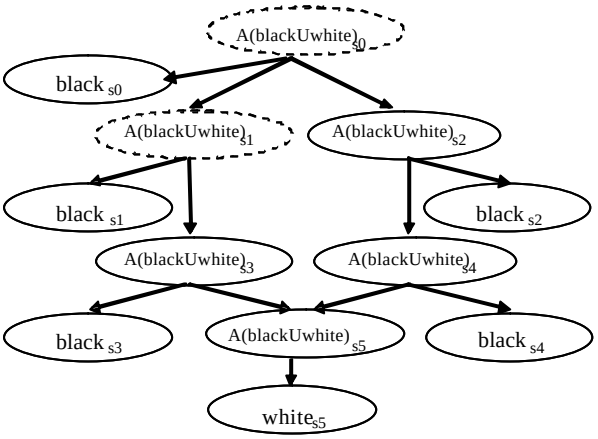


Figure 2. Sub-formula Dependence Graph.

The re-verification process of this example can be viewed as the space of function-call in figure 3. The dashed box indicates a re-verification of uncertain sub-formula while the solid box indicates a re-verification of certain sub-formula. The re-verification of uncertain sub-formula is similar to conventional verification but the re-verification of certain sub-formula is different because the model checker can immediately get the truth value of that sub-formula without a deep re-verification and a modification to the label of state. Thus, the complexity in the re-verification process of our algorithm is less than the process of conventional algorithm.

Since the circular function-call occurs in this re-verification, by theorem 1, then the re-verification algorithm will determine that **A**(black**U**white) is false on state s_0. Therefore, the property **A**(black**U**white) is not preserved in the evolving model $\mathcal{M}'$.

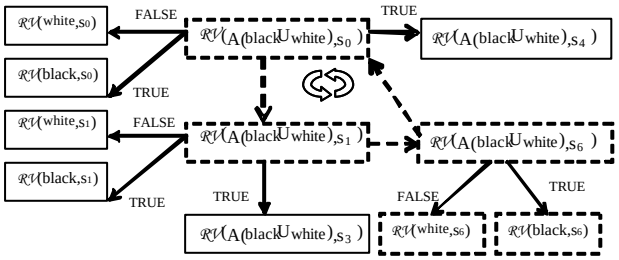


Figure 3. The re-verification process.

6 The Evolution of Property

It is possible that the additional properties defined for responding to the additional features may be evolved from the original properties which hold in the base model. We define the informal semantic of property evolution as "*the property φ′ is evolved from φ if and only if there exists sub-formula of φ within the proposition of φ′*". For instance, let the property φ = **E**(black**U**white) holds on the base model $\mathcal{M}$. The developers introduce the additional aspect to that base model for evolving to $\mathcal{M}'$ and they need to know that the additional property φ′ = "**AG**(**E**(black**U** white)∧**EG**(black))" holds in the model $\mathcal{M}'$ or not. The property φ′ evolves from φ because the **E**(black **U**white) of φ is also within the proposition of φ′. In our approach, if the **E**(black**U**white) is considered as a certain sub-formula, then the verification of φ′ with the re-verification algorithm is completed faster than the traditional algorithm because, rather than verification of all sub-formulas, the **E**(black**U**white) is ignored from the verification. Thus, our approach is very useful for this circumstance.

7 Conclusion

This paper presents the software evolution based on the aspect-oriented concepts. The basic models for aspect-oriented software are proposed. We employ the model checking for a verification of that aspect-oriented software model. The contribution of this paper is the preservation checking technique called *certainty-aware technique*. We suggest that this technique should be utilized by the CTL model checker for reducing the complexity of verification process. This technique consists of two processes: certainty analysis and re-verification. With the certainty analysis, we can determine if sub-formulas are certain or not after the evolution. Only the uncertain sub-formulas must be re-verified in the re-verification process for their actual truth values. As discussed in section 6, this technique is not only useful for the software model evolution, but also the property evolution. Some sub-formulas in the evolving property need not be re-verified if they are considered as the certain sub-formulas. Thus, only the additional sub-formulas should be verified.

We demonstrate the effectiveness of our technique with a simple example. For future works, the actual assessment of our approach should be performed by the experiment on real world applications. It is possible that there is an introduction of new actors after the evolution, but our approach is limited on only the case that the numbers of actors are not changed. Hence, the generality of our approach should be improved and the model checker which utilizes our technique should be implemented.

8 Acknowledgements

We are grateful to Professor Takuya Katayama for his comments of our works.

References:

[1] E. M. Clarke, E. A. Emerson, and A. P. Sistla, Automatic Verification of Finite-State Concurrent Systems using Temporal Logic Specifications, *ACM Transactions on Programming Languages and Systems*, Vol. 8, No. 2, 1986, pp.244-263.

[2] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*, The MIT Press, 2000.

[3] P. Tarr, H. Ossher, W. Harrison, and S. M. Sutton, N-degrees of separation: Multi-dimen sional separation of concerns. In Proc. ICSE, 1999, pp. 109-117.

[4] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin, Aspect-oriented programming, In European Conference on Object-Oriented Programming *(ECOOP'97)*, 1997, pp. 220-242.

[5] D. Batory, J.N. Sarvela, and A. Rauschmayer, Scaling Step-Wise Refinement, In Proc. ICSE, 2003, pp. 187-197.

[6] T. Mens and M. Wermelinger, Separation of concerns for software evolution, *Journal of Software Maintenance and Evolution*, Vol. 14, 2002, pp. 311-315.

[7] Y. Smaragdakis and D. Batory, Mixin layers: an object-oriented implementation technique for refinements and collaboration-based designs, *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 11, No.2, April 2002, pp.215-255.

[8] K. Fisler and S. Krishnamurthi, Modular verification of collaboration-based software designs, In Proc. Symposium on the Foundations of Software Engineering, 2001, pp.152-163.

[9] N. T. Thang and T. Katayama, Towards a Sound Modular Model Checking of Collaboration-Based Software Design, In Proc. The tenth Asia-Pacific Software Engineering Conference (*APSEC'03*), 2003.

Appendix

Lemma 1: Given a re-verification path that can be enumerated as $f0_{s0}$, $f1_{s1}, f2_{s2}, \ldots, fn\text{-}1_{sn-1}, fn_{sn}$. Suppose fn_{sn} is re-verified by the re-verification function $\mathcal{RV}$ and it can be changed to be certain, then this re-verification path is not a re-verification cycle. *Proof*

Step 1: By definition 7, a re-verification of $\{f_s\}$ that results in either TRUE or FALSE must conform to one of following conditions:

- $\mathcal{T}(f) \notin \{EX, AX, EU, AU\}$, thus f can be evaluated immediately at state s.
- $\mathcal{T}(f) \in \{EX, AX, EU, AU\}$ and $\forall f'_{s'}[\mathcal{RV}(f,s) \rightarrow \mathcal{RV}(f',s') \Rightarrow |f'_{s'}|]$.

Step 2: From the hypothesis, fn_{sn} is uncertain at first then it can be changed to certain after the completion of $\mathcal{RV}(fn, sn)$. Therefore, from step 1, fn_{sn} must conform to one of the following conditions:

- $\mathcal{T}(fn) \notin \{EX, AX, EU, AU\}$; fn can be evaluated immediately at state sn.
- $\mathcal{T}(fn) \in \{EX, AX, EU, AU\}$ and $\forall f'_{s'}[\mathcal{RV}(fn, sn) \rightarrow \mathcal{RV}(f', s') \Rightarrow |f'_{s'}|]$

Step 3: By definition 8, we can enumerate the view of recursive function-call on this re-verification path as $\mathcal{RV}(f0,s0) \rightarrow \mathcal{RV}(f1,s1) \rightarrow \mathcal{RV}(f2,s2) \rightarrow \mathcal{RV}(f3, s3) \rightarrow \ldots \rightarrow \mathcal{RV}(fn\text{-}1, sn\text{-}1) \rightarrow \mathcal{RV}(fn,sn)$. Since there is a recursive function-call from $\mathcal{RV}(f0,s0)$, $f0$ cannot be evaluated immediately at state $s0$, thus $\mathcal{RV}(f0, s0)$ requires the certainty of $f1_{s1}$ from the re-verification result of $\mathcal{RV}(f1, s1)$ and so on.

Step 4: Suppose fn cannot be evaluated immediately at state sn then all $f'_{s'}$ that $\mathcal{RV}(fn, sn) \rightarrow \mathcal{RV}(f', s')$ must be certain (from step 2). Since there is a recursive function-call of $\mathcal{RV}(f0,s0) \rightarrow \mathcal{RV}(f1, s1)$ which indicates that $f1_{s1}$ is uncertain, then $f0_{s0} \neq fn_{sn}$. Therefore, from definition 8, we can prove that the re-verification path is not a re-verification cycle. ¨

Lemma 2: Given a re-verification path that can be enumerated as $f_{s0}, f_{s1}, f_{s2}, \ldots, f_{sm-1}, f_{sm}$ where f is totally $A(f_1 U f_2)$ or f is totally $E(f_1 U f_2)$. Suppose $\exists n[(0 \leq n \leq m) \Rightarrow \mathcal{RV}(f_2,s_n)$=TRUE] then a sequence $f_{s0}, f_{s1}, f_{s2}, \ldots, f_{sn-1}, f_{sn}$ is not a re-verification cycle. *Proof*

From the hypothesis, we can assume $\mathcal{RV}(f_2,s_n)$=TRUE. By definition 7, $\{f_{sn}\}$ is changed to $|f_{sn}|$ whenever $\mathcal{RV}(f,s_n)$ completes the verification process. With the lemma 1, we can deduce that the sequence $f_{s0}, f_{s1}, f_{s2}, f_{s3}, \ldots, f_{sn-1}, f_{sn}$ is not the re-verification cycle because f_{sn} can be changed to a certain sub-formula. ¨

Theorem 1: Let f is $A(f_1 U f_2)$, if there exists a re-verification cycle starting from f_{s0} which can be enumerated as $f_{s0}, f_{s1}, f_{s2}, f_{s3}, \ldots, f_{sn-1}, f_{sn}, f_{s0}$ then $\mathcal{RV}(f, s_0)$ = FALSE. *Proof*

Step 1: The re-verification cycle that is starting from f_{s0} can be enumerated as $f_{s0}, f_{s1}, f_{s2}, f_{s3}, \ldots, f_{sn-1}, f_{sn}, f_{s0}$. By definition 8, we can illustrate the view of recursive function-call on this re-verification cycle as $\mathcal{RV}(f,s_0) \rightarrow \mathcal{RV}(f,s_1) \rightarrow \mathcal{RV}(f,s_2) \rightarrow \mathcal{RV}(f,s_3) \rightarrow \ldots \mathcal{RV}(f,s_{n-1}) \rightarrow \mathcal{RV}(f,s_n) \rightarrow \mathcal{RV}(f,s_0)$. Since state s_0 is an immediate successor of s_n and state s_i is an immediate successor of s_{i-1} where $1 \leq i \leq n$, then there exists a path $(s_0, s_1, \ldots, s_{n-1}, s_n, s_0)$ in the state-transition graph. (1)

Step 2: The contrapositive of Lemma 2 is that if the sequence $f_{s0}, f_{s1}, f_{s2}, f_{s3}, \ldots, f_{sn-1}, f_{sn}$ is a re-verification cycle then $\forall i[(0 \leq i \leq m) \Rightarrow \mathcal{RV}(f_2,s_i)$ = FALSE] ,where $m \geq n$.

Therefore, $\forall i[(0 \leq i \leq n) \Rightarrow s_i \models \neg f_2]$. (2)

Step 3: From (1) (2) and Lemma 3.2 of [1], we can deduce that $s_0 \models \neg A(f_1 U f_2)$. Thus, we can conclude that $\mathcal{RV}(f,s_0)$ = FALSE where f is $A(f_1 U f_2)$. ¨

Note: Lemma 3.2 (from [1])

Suppose there exists a path $(s_0, s_1, s_2, \ldots, s_n, s_k)$ in the state-transition graph such that $0 \leq k \leq n$ and $\forall i[0 \leq i \leq n \Rightarrow s_i \models \neg f_2]$, then $s_k \models \neg A(f_1 U f_2)$.

Theorem 2: Let f is $E(f_1 U f_2)$, if there are all re-verification cycles starting from f_{s0} which can be enumerated as $f_{s0}, f_{s1}, f_{s2}, f_{s3}, \ldots, f_{sn-1}, f_{sn}, f_{s0}$ then $\mathcal{RV}(f,s_0)$ = FALSE. *Proof*

Let t is an immediate successor of s in a state-transition graph.

From definition 7,

$(\mathcal{RV}(f,s)$=TRUE) $\Rightarrow$ ($\mathcal{RV}(f_2,s)$=TRUE $\vee$ ($\mathcal{RV}(f_1,s)$=TRUE $\wedge$ $\exists t$ [$\mathcal{RV}(f,t)$=TRUE]))

The contrapositive of above statement is

($\mathcal{RV}(f_2,s)$=FALSE $\wedge$ ($\mathcal{RV}(f_1,s)$=FALSE $\vee$ $\forall t$[$\mathcal{RV}(f,t)$= FALSE])) $\Rightarrow$ ($\mathcal{RV}(f,s)$=FALSE)

So, if we can show $\mathcal{RV}(f_2,s_0)$=FALSE and $\forall t$[$\mathcal{RV}(f,t)$=FALSE] where t is an immediate successor of s_0 then we can prove the $\mathcal{RV}(f,s_0)$ = FALSE.

Step 1: Each re-verification cycle that is starting from f_{s0} can be enumerated as $f_{s0}, f_{s1}, f_{s2}, f_{s3}, \ldots, f_{sn-1}, f_{sn}, f_{s0}$. The contrapositive of Lemma 2 is that if the sequence $f_{s0}, f_{s1}, f_{s2}, f_{s3}, \ldots, f_{sn-1}, f_{sn}$ is a re-verification cycle then $\forall i[(0 \leq i \leq m) \Rightarrow \mathcal{RV}(f_2,s_i)$ = FALSE] where $m \geq n$.

Therefore, $\mathcal{RV}(f_2,s_0)$ = FALSE. (1)

Step 2: For each re-verification cycle, $\forall i$ [($1 \leq i \leq n$) $\Rightarrow$ $\mathcal{RV}(f,s_i)$=FALSE]) because the following two reasons:

1.) $\forall i[(1 \leq i \leq n) \Rightarrow \mathcal{RV}(f_2,s_i)$=FALSE] (from above) and

2.) $\forall i$ [($1 \leq i \leq n$) $\Rightarrow$ $\{f_{ti}\}$ cannot change to $|f_{ti}|$] (by the contrapositive of Lemma 1, *if the re-verification path is a re-verification cycle then fn_{sn} cannot change to be certain by the re-verification function* $\mathcal{RV}$). Thus, we can infer that $\forall s_i$ [$\mathcal{RV}(f,s_i)$ =FALSE] where f_{s1} is an immediate successor of f_s in all re-verification cycles. (2)

From (1) and (2), we can prove that $\mathcal{RV}(f,s_0)$ = FALSE. ¨