Open Incremental Model Checking (Extended Abstract)

Nguyen Truong Thang Takuya Katayama
School of Information Science
Japan Advanced Institute of Science and Technology
email: {thang, katayama}@jaist.ac.jp

Categories and Subject Descriptors
D.2.4 [**Software Engineering**]: Software/Program Verification—*formal methods, model checking*

1. INTRODUCTION
Separation of concerns is the core of successful software [5]. One of the most prominent form of concerns are *hyperslices* [5] or features. A system is structured by composing several separate features. The terms feature, hyperslice and component are used interchangeably henceforth.

This paper focuses on the interaction between two components: base and extension. Specifically, the extension refines or modifies the base, i.e. the interferences of the base and extension execution paths occur. Unlike traditional modular model checking methods which treat systems as static, a new method of model checking, called *open incremental model checking* (OIMC) in our opinion, is proposed to address the changes to systems [1]. Given a base component, an extension component is attached such that the extension does not violate some property inherent to the base. A primitive model and a simple verification procedure are suggested to ensure the consistency between two components [1]. The model checking is executed in an *incremental* manner within the extension component only. This approach is also *open* for various kinds of changes. This paper is quite different the work of [1] in several key points such as proposing a generalized model with overriding capability (Section 3), an explicit consistency condition among components (Section 4). More importantly, we also examine key issues not addressed in [1] such as the soundness (Section 5.1) and scalability (Section 5.2). Discussion about the contribution of this paper, its future and related work are presented in Section 6.

2. BACKGROUND
CTL is a restricted subset of CTL* in which each temporal operator among **X** ("next"), **F** ("eventually"), **G** ("always"), **U** ("until") and **R** ("release") must be preceded by a quantifier of **A** ("for all paths") and **E** ("for some path"). With respect to CTL, the incremental verification method has been attempted by [1]. Its key ideas are:

- Proposing a simple formal model whose interface is fixed with single *exit* and single *reentry*. Importantly, this model is additive, i.e. the extension is not allowed to override any behavior of the base.
- Presenting a verification algorithm to check whether a property continues to hold at those exit states. Fundamentally, the algorithm is based on the assumption about labels at all reentry states. From the assumption, the conclusion about the extension component with respect to the property is drawn. The unproven assumption is a weakness of [1] in terms of soundness.

In this paper, the formal interface is generalized to accommodate multiple exit and reentry points with overriding capability. The soundness of OIMC with respect to this generalized model is then tackled in two aspects: proving assumptions at reentry states instead of simply assuming them (1); and based on the proven facts at reentry states, proving the property preservation in the base component (2). Later, the scalability problem of OIMC is discussed with respect to the reality that many subsequent extension components will be incorporated into the newly evolved system. We investigate the complexity of OIMC under such a situation to preserve the property of the original base component.

3. FEATURE SPECIFICATION MODEL
Each component is separately modeled via a state transition model. Let AP be a set of atomic propositions.

DEFINITION 1. *A state transition model M is a tuple $\langle S, \Sigma, s_0, R, L\rangle$ where S is a set of states, Σ is the set of input events, $s_0 \in S$ is the initial state, $R \subseteq S \times PL(\Sigma) \rightarrow S$ is the transition function (where $PL(\Sigma)$ denotes the set of propositional logic expressions over Σ), and $L : S \rightarrow 2^{AP}$ labels each state with the set of atomic propositions true in that state.*

Typically, there are two components to consider: a base and an extension. Between the base and its extension is an interface consisting of *exit* and *reentry* states. An exit state is the state where control is passed to the extension. On the

other hand, a reentry state is the point at which the base regains control. A *base* is expressed by a transition model B and an *interface* I, where $B = \langle S_B, \Sigma_B, s_{o_B}, R_B, L_B \rangle$. An interface is a tuple of two state sets $I = \langle exit, reentry \rangle$, where $exit, reentry \subseteq S_B$ and $exit, reentry \neq \emptyset$. On the other hand, an *extension* is represented by a model $E = \langle S_E, \Sigma_E, \smile, R_E, L_E \rangle$, if considered separately from the base B. $\smile$ denotes no-care value. The interface of E is $J = \langle in, out \rangle$.

E can be inserted to B via the *compatible* interface states according to the following.

- An exit state $ex \in exit$ of B can be matched to an in-state $i \in in$ if $L_E(i) \subseteq L_B(ex)$.
- A reentry state $re \in reentry$ of B can be matched to an out-state $o \in out$ if $L_B(re) \subseteq L_E(o)$.

Subsequently, states ex and re will be used in place of i and o whenever interface states are referred.

DEFINITION 2. *Composing the base B with the extension E, through the interfaces I and J produces a composition model $C = \langle S_C, \Sigma_C, s_{0_C}, R_C, L_C \rangle$. C is defined from $B = \langle S_B, \Sigma_B, s_{0_B}, R_B, L_B \rangle$ and $E = \langle S_E, \Sigma_E, \smile, R_E, L_E \rangle$.*

- $S_C = S_B \cup S_E$; $\Sigma_C = \Sigma_B \cup \Sigma_E$; $s_{0_C} = s_{0_B}$;
- *R_C is defined from R_B and R_E. For each $s \in S_C$, let $\vee_s^E = \bigvee pl_i$ where $(s, pl_i) \in Dom(R_E)$,*
  - $\forall (s, pl_i) \in Dom(R_E)$: $R_C(s, pl_i) = R_E(s, pl_i)$
  - $\forall (s, pl_B) \in Dom(R_B)$: $R_C(s, pl_B \wedge \neg \vee_s^E) = R_B(s, pl_B \wedge \neg \vee_s^E)$
- $\forall s \in S_B, s \notin I.exit \cup I.reentry : L_C(s) = L_B(s)$;
- $\forall s \in S_E, s \notin J.in \cup J.out : L_C(s) = L_E(s)$;
- $\forall s \in I.exit \cup I.reentry : L_C(s) = L_B(s) \cup L_E(s)$;

The propositional logic expressions between different transitions from the same state are disjoint. $\vee_s^E$ represents the union of all events directing to the extension from a state s. In the composition definition above, a transition (s, pl_B, s') in B can be partially or completely overridden by E. In case of being overridden, s is certainly an exit state because overriding only occurs at exit states. The transition is completely removed from C if $pl_B \wedge \neg \vee_s^E = false$. Otherwise, it is partially overridden. This is another key difference between our model and the former [1] in which E is not allowed to override any transition in B. The former is called *additive-only* composition, while ours is *limited overriding*.

DEFINITION 3. *The* closure *of a property p, $cl(p)$, is the set of all sub-formulae of p, including itself.*

DEFINITION 4. *The truth values of state s with respect to a set of CTL properties ps within a model $M = \langle S, \Sigma, s_0, R, L \rangle$, denoted $\mathcal{V}_M(s, ps)$, is a function: $S \times 2^{CTL} \rightarrow 2^{CTL}$ defined according to the following:*

- $\mathcal{V}_M(s, \emptyset) = \emptyset$
- $\mathcal{V}_M(s, \{p\} \cup ps) = \mathcal{V}_M(s, \{p\}) \cup \mathcal{V}_M(s, ps)$
- $\mathcal{V}_M(s, \{p\}) = \begin{cases} \{p\} & \text{if } M, s \models p \\ \{\neg p\} & \text{otherwise} \end{cases}$

CTL denotes the set of all CTL properties. Hereafter, $\mathcal{V}_M(s, \{p\}) = \{p\}$ (or $\{\neg p\}$) is written in the shorthand form as $\mathcal{V}_M(s, p) = p$ (or $\neg p$) for individual property p.

In the subsequent discussion, incremental model checking is represented by an *assumption model checking* [4] in E only rather than in C. The *assumption function* for that model checking is a function $As : I.reentry \rightarrow 2^{CTL}$. In such a situation, the reentry states re in E are assumed with truth values seeded from B, $\mathcal{V}_B(re, cl(p))$, namely $As(re) = \mathcal{V}_B(re, cl(p))$.

DEFINITION 5. *The assumption function As is* proper *at a reentry state re if the assumed truth values are exactly those resulted at re from the standard model checking in C, i.e. $\mathcal{V}_B(re, cl(p)) = \mathcal{V}_C(re, cl(p))$.*

4. PROPERTIES PRESERVATION AT BASE STATES

A property p is adhered to the base $B = \langle S_B, \Sigma_B, s_{0_B}, R_B, L_B \rangle$ if it holds for every state in B, i.e. $\forall s \in S_B : B, s \models p$. An extension E is *composable* with B with respect to p if $\forall s \in S_B : C, s \models p$ where C is the composition of B and E.

The key problem this paper tries to deal with is: Given B and p, what are the conditions for E so that B and E are composable with respect to p?

With respect to the generalized model, the soundness issue is very important. It will be discussed in Section 5.1. Another question relates to the scalability of OIMC (Section 5.2) with respect not only to E but also to many future extensions to the composition C.

4.1 A Theorem on Component Consistency

Due to the inside-out characteristic of model checking, during verifying p in B, $\mathcal{V}_B(s, cl(p))$ are recorded at each state s. The truth values $\mathcal{V}_B(ex, cl(p))$ at any exit state ex serve as the conformance for the composition between B and E.

DEFINITION 6. *B and E are in* conformance *at the exit state ex (with respect to $cl(p)$) if $\mathcal{V}_B(ex, cl(p)) = \mathcal{V}_E(ex, cl(p))$.*

THEOREM 7. *Given a base B and a property p, an extension E is attached to B at some interface states. Further, suppose that the assumption function As defined during model checking E is proper. If B and E conform with each other at all exit states, $\forall s \in S_B : \mathcal{V}_B(s, cl(p)) = \mathcal{V}_C(s, cl(p))$.*

The theorem holds regardless of composition type, either additive or overriding. Due to space limitation, the proof of the theorem is skipped. Figure 1 depicts the composition

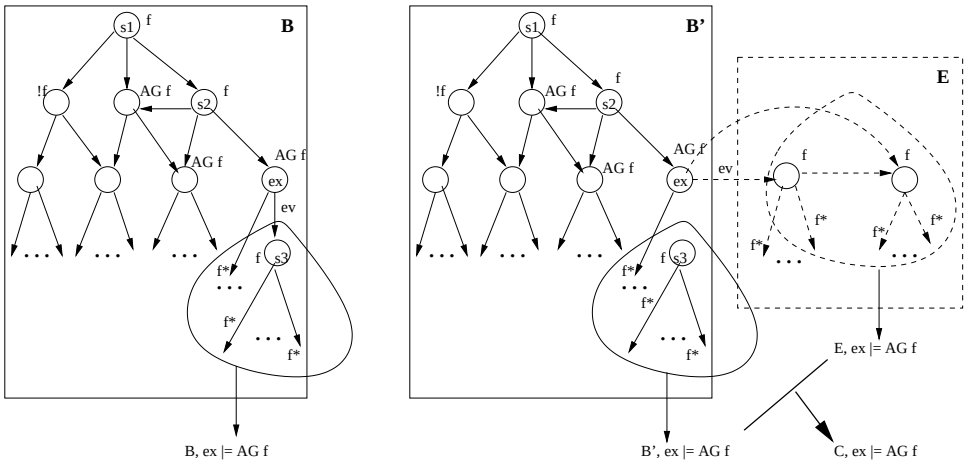

Figure 1: An illustration of base-overriding composition conformance. The truth value with respect to the property $p = \mathbf{AG}\, f$ is preserved at ex as well as all states in B.

preserving the property $p = \mathbf{AG}\, f$ when B and E are in conformance. The composition is done via a single exit state ex. Further, E overrides the transition ex-s_3 whose input event is ev in B. B' is the remainder of B after removing all overridden transitions. f^* denotes that f holds at all intermediate states along the computation path. In the figure, within B, $p = \mathbf{AG}\, f$ holds at s_2, ex and s_3 but not at s_1. As $\mathcal{V}_E(ex, p) = \mathcal{V}_{B'}(ex, p) = \mathcal{V}_B(ex, p) = \mathbf{AG}\, f$, B and E conform at ex. While the edge ex-s_3 is removed, the new paths in E together with the remaining computation tree in B' still preserve p at ex directly; and consequently s_2 indirectly. For s_1, its truth value $\mathcal{V}_C(s_1, p) = \neg p$ is preserved as well. On the other hand, s_3 is not affected by E. In this figure, we do not care about the descendant states in E. Thus, E is intentionally left open-end so that the reentry state re is not explicitly displayed. In this part, what E can deliver at ex is important regardless of ex's descendants. The arguments are still valid when the downstream of E converges to the reentry state re.

From Theorem 7, if there is a conformance at all exit states, all truth values with respect to $cl(p)$, surely including p, at base states are preserved. The following corollary is the answer to the problem earlier prescribed in this section - the key of this paper.

COROLLARY 8. *Given a model B and a CTL property p adhered to it, an extension E is attached to B at some interface states. Further, suppose that the assumption function As is proper. E does not violate p inherent to B if B and E conform with each other at all exit states.*

The properness of As is a major part for the soundness of the incremental verification which is mentioned with in Section 5.1. Instead of assuming As's properness, we need to prove it.

4.2 Open Incremental Model Checking

From Corollary 8, the preservation constraints are required at exit states only. Corresponding to an exit state ex, the algorithm to verify a preservation constraint in E can be briefly described as follows:

1. Seeding all reentry states re with $\mathcal{V}_B(re, cl(p))$.
2. Executing the standard CTL model checking procedure in E from re states backward to ex. The formula to check are $\forall \phi \in cl(p)$.
3. At the end of the the model checking procedure, checking if $\mathcal{V}_E(ex, cl(p)) = \mathcal{V}_B(ex, cl(p))$.
4. Repeating the procedure for other exit states.

At the end of the process, if at all exit states, the truth values with respect to $cl(p)$ are matched respectively. B and E are composable.

5. SOUNDNESS AND SCALABILITY ISSUES

5.1 Soundness Issue

In Section 4, the assumption function As is constructed by copying the truth values at reentry states re in B directly. The copying step implicitly assumes that As is proper at all reentry states. For the soundness of OIMC, this section is mainly concerned with proving As's properness, i.e. checking whether Theorem 7 remains valid if the assumption on the properness of As is dropped. Thus, the soundness problem in essence consists of two parts:

1. Proving that As is proper at all reentry states (This is to make sure that the label seeding steps at reentry states are correct). (Soundness Problem 1)
2. Based on the above As's properness, proving that the truth values with respect to $cl(p)$ are preserved at all exit states and hence at all base states. (Soundness Problem 2)

In OIMC, we are only concerned with the interface states between B and E because at these states, the associated

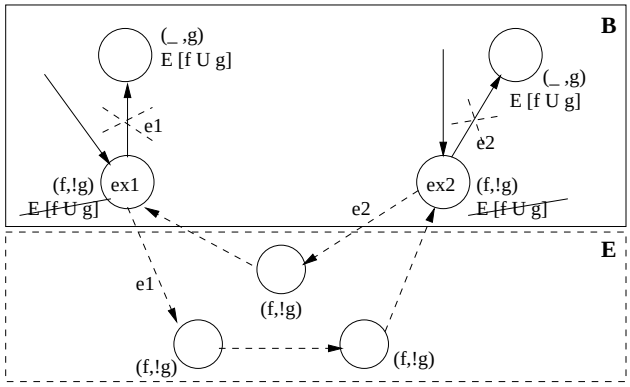

Figure 2: A composition failing to preserve $p = \mathbf{E}[f\,\mathbf{U}\,g]$ in case of extension-only cyclic dependency.

computation trees are first to change, if any. Certainly, the property changes at these states then propagate to ascendant states in B. By an observation, if the truth values with respect to $cl(p)$ are preserved at all interface states, the same thing happens at all base states.

Between these interface states are dependency relations due to CTL model checking, i.e. from a state s to any descendant state d of s. If $\mathcal{V}_C(d, cl(p)) \neq \mathcal{V}_B(d, cl(p))$ then it is likely that $\mathcal{V}_C(s, cl(p)) \neq \mathcal{V}_B(s, cl(p))$. These interface states together define a *dependency structure*. The soundness of Theorem 7 after dropping the assumption on As's properness is examined. The results are as follows:

- The theorem is sound if the dependency structure is acyclic, regardless of composition type (additive or overriding).
- The theorem is sound if the composition is additive, regardless of the dependency structure.
- It may fail in some extreme cases of overriding composition with cyclic dependency.

The failing case is illustrated in Figure 2. Two exit states are mutually reentry states in E, namely cyclic dependency between ex_1 and ex_2. Further, E overrides critical paths rooted at ex_1 and ex_2, whose associated input events are e_1 and e_2, with respect to $p = \mathbf{E}[f\,\mathbf{U}\,g]$. Initially, $p = \mathbf{E}[f\,\mathbf{U}\,g]$ holds at both ex_1 and ex_2. However, after the overriding composition, at these states, the property no longer holds (being crossed out). The assumption function As is not proper at both states. Thus, the result of OIMC within E is not correct due to incorrect label seeding.

5.2 Scalability Issue

So far, we have investigated only one-step extension in which E is attached to B. We are concerned with the application of OIMC for subsequent extensions to C. We consider the n-th version ($C_n = C_{(n-1)} + E_n$) during software evolution as a structure of components B, E_1, E_2, ..., E_n. Here, E_i is the extension component to the $(i-1)$-th evolved version ($C_{(i-1)}$). The initial version is $C_0 = B$. The complexity of verification does not change after adding feature E_n according to Theorem 9 below whose proof is skipped. OIMC maintains its scalability - the incremental characteristic.

THEOREM 9. *Suppose that any pair of base and extension components $C_{(i-1)}$ and E_i respectively conform at all exit states, $i = \overline{1,(n-1)}$. The complexity of the incremental verification for confirming E_n not violating the property p in B only depends on the size of E_n (proportional to the number of states and transitions in E_n).*

6. CONCLUSION

Compared with the earlier work [1], this paper differs in several significant points. They include: a precise and generalized formal model of feature-based software with overriding possibility (1); the soundness problem of OIMC, especially in case of cyclic dependency between interface states, via two sub-problems: the properness of As and properties preservation at exit states (2); an unified condition, $\mathcal{V}_E(ex, cl(p)) = \mathcal{V}_B(ex, cl(p))$, for any legal composition of B and E (3); and the scalability of OIMC (4).

Comparing to the modular verification work [2, 3, 4], there is a fundamental difference in characteristic between those and the work of both [1] and ours. Modular verification in those work are rather closed. Even though it is based on component-based modular model checking, it is not prepared for changes. If a component is added, the whole system of many existing components and the new component is re-checked altogether. On the contrary, the approach in [1] and this paper is incrementally modular. It is also open for future changes. We only check the new system partially within the new component and its interface with the rest of the system. Certainly, this merit comes at the cost of "fixed" conditions at exit states. This "fixed" constraint can cause false negatives to some legal extensions. One of the future work is to relax the conformance condition based on matching truth values with respect to $cl(p)$.

7. REFERENCES

[1] K. Fisler and S. Krishnamurthi. Modular verification of collaboration-based software designs. In *Proc. Symposium on the Foundations of Software Engineering*, September 2001.

[2] O. Grumberg and D. E. Long. Model checking and modular verification. In *International Conference on Concurrency Theory*, volume 527 of *Lecture Notes of Computer Science*. Springer-Verlag, 1991.

[3] O. Kupferman and M. Y. Vardi. Modular model checking. In *Compositionality: The Significant Difference*, volume 1536 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[4] K. Laster and O. Grumberg. Modular model checking of software. In *Conference on Tools and Algorithms for the Constructions and Analysis of Systems*, 1998.

[5] P. Tarr, H. Ossher, W. Harrison, and S. M. Sutton. N-degrees of separation: Multi-dimensional separation of concerns. In *Proc. ICSE*, pages 109 – 117, 1999.