Foundations of Tree-like Local Model Updates

Yan Zhang, Michael Kelly and Yi Zhou
University of Western Sydney, Australia
Email: {yan,mkelly,yzhou}@scm.uws.edu.au

Abstract. Model update is an approach to enhance model checking functions by providing computer aided modifications in system development [2, 9]. It has been observed that one major obstacle restricting the application of this approach, e.g. CTL model update [15], is that the update has to take the entire system model into account, and that is usually not feasible for large scale domains. In this paper, we develop a tree-like local model update approach under the framework of ACTL - a widely used fragment of CTL in property specification. We define a bisimulation based minimal change principle on tree-like local model update, reveal its relationship to traditional belief update and provide essential semantic characterizations. We also investigate primary semantic and computational properties in relation to tree-like local model update. Finally we briefly describe the update system prototype that we have implemented and summarize our experimental results.

1 Introduction

Model checking has demonstrated its promising effectiveness for complex system verification [1]. Although counterexamples generated from model checking procedures provide important information for system designers to specifically identify the errors, model checking itself does not directly fix the system faults. Recently, some approaches have been proposed towards a computer aided system modification, such as the work presented in [8, 11].

Among these various approaches, one important idea is to integrate AI techniques, such as belief revision and model update, into model checking to develop effective system repairing tools, e.g. [2, 9]. The recent work presented in [15] demonstrated how a general CTL model update method can be developed for this purpose. The idea is quite clear: when a CTL specification property fails over a Kripke model through a model checking, the underlying Kripke model will be updated in an automated way, and the result is a new Kripke model which satisfies the specification property. From this result, a candidate modification for the original Kripke model may be derived by the system designer. However, one major obstacle restricting its applications is that the update prototype has to take the entire system model (e.g. a complete Kripke model) as the input. This, obviously, is not generally feasible for large scale domains.

On the other hand, it is well understood that counterexamples generated from a model checking procedure play an essential role in system repairing, because a counterexample usually localizes certain information that reveals how the system fails the specification property, e.g. [7]. Therefore, one natural way to overcome the difficulty of model update approach mentioned above is that we should develop a local model update that only applies to counterexamples and effectively generates candidate modifications for the original system.

In this paper, we present a local model update approach that effectively overcomes this difficulty. We focus on a widely used fragment of CTL - ACTL, and develop the counterexample guided ACTL local model update approach, in which an update is performed on the tree-like ACTL counterexample. We define a minimal change principle for such update problem based on the concept of bisimulation on tree-like Kripke structures. We then reveal the relationship between our local model update and traditional belief update, and provide essential semantic characterizations. We also investigate primary computational properties in relation to our update approach. Finally, we briefly describe the local model update prototype that we have implemented and present a summary of our experimental results.

2 ACTL and Tree-like Structures

ACTL is a fragment of Computation Tree Logic (CTL) and has attracted considerable studies from researchers, e.g. [3, 4]. Besides Boolean connectives, ACTL provides both linear time operators X, F, G and U and the branching time operator A. The linear time operators allow one to express properties of a particular evaluation of the systems given by a series of events in time, and the branching time operator takes into account the multiple possible future scenarios starting from a given state at certain time. ACTL has the following syntax given in Backus-Naur form:

$\phi ::= \top \mid \bot \mid p \mid \neg p \mid \phi \wedge \psi \mid \phi \vee \psi \mid \mathrm{AX}\phi \mid \mathrm{AG}\phi \mid \mathrm{AF}\phi \mid$
$\mathrm{A}[\phi \mathrm{U} \psi]$, where p is any propositional atom (variable).

Let AP be a set of propositional variables. A *Kripke structure* M over AP is a triple $M = (S, R, L)$, where S is a finite set of states, $R \subseteq S \times S$ is a binary relation representing state transitions, and $L : S \rightarrow 2^{AP}$ is a labeling function that assigns each state with a set of propositional variables. A *path* is an infinite sequence of states. We assume that R is *total*, that is, all states have positive out-degree. Therefore, each finite path can be extended into an infinite path.

Let $M = (S, R, L)$ be a Kripke structure and $s_0 \in S$. A *path* in M starting from s_0 is denoted as $\pi = [s_0, \cdots, s_i, s_{i+1}, \cdots]$, where $(s_i, s_{i+1}) \in R$ holds for all $i \geq 0$. If $\pi = [s_0, s_1, \cdots, s_i, \cdots, s_j, \cdots]$ and $i < j$, we denote $s_i < s_j$. For any $s \in S$, the satisfaction relation between (M, s), which we call a *Kripke model induced from* M (or simply *Kripke model*), and an ACTL formula ϕ, denoted by $(M, s) \models \phi$, is defined in a standard way as described in [10].

Definition 1 *Let $M = (S, R, L)$ be a Kripke structure. Given any s in S, we define whether an ACTL formula ϕ holds in M at state s. We write this by $(M, s) \models \phi$. The satisfaction relation $\models$ is defined by structural induction on all ACTL formulas*[1]*:*

[1] Here we follow [10]'s style to define ACTL semantics.

1. $(M, s) \models \top$ *and* $(M, s) \not\models \bot$ *for all* $s \in S$.
2. $(M, s) \models p$ *iff* $p \in L(s)$.
3. $(M, s) \models \neg p$ *iff* $p \notin L(s)$.
4. $(M, s) \models \phi_1 \wedge \phi_2$ *iff* $(M, s) \models \phi_1$ *and* $(M, s) \models \phi_2$.
5. $(M, s) \models \phi_1 \vee \phi_2$ *iff* $(M, s) \models \phi_1$ *or* $(M, s) \models \phi_2$.
6. $(M, s) \models \mathrm{AX}\phi$ *iff for all* s_1 *such that* $(s, s_1) \in R$, $(M, s_1) \models \phi$.
7. $(M, s) \models \mathrm{AG}\phi$ *iff for all paths* $\pi = [s_0, s_1, s_2, \cdots]$ *where* $s_0 = s$ *and* $\forall s_i,\ s_i \in \pi$, $(M, s_i) \models \phi$.
8. $(M, s) \models \mathrm{AF}\phi$ *iff for all paths* $\pi = [s_0, s_1, s_2, \cdots]$ *where* $s_0 = s$ *and* $\exists s_i,\ s_i \in \pi$, $(M, s_i) \models \phi$.
9. $(M, s) \models \mathrm{A}[\phi_1 \mathrm{U} \phi_2]$ *iff for all paths* $\pi = [s_0, s_1, s_2, \cdots]$ *where* $s_0 = s$, $\exists s_i \in \pi$, $(M, s_i) \models \phi_2$ *and* $\forall j < i$, $(M, s_j) \models \phi_1$.

For simplicity reason, in the rest of this paper, unless we explicitly mention, we will only consider satisfiable ACTL formulas in the context. Now we introduce the concept of *tree-like Kripke structures* [4]. Let G be a directed graph. A *strongly connected component* (SCC) C in G is a maximal subgraph of G such that every node in C is reachable from every other node in C. C is *nontrivial* iff either it has more than one node or it contains one node with a self-loop. The *component graph* $c(G)$ of G is the graph where the vertices are given by the SCCs of G, and where two vertices of $c(G)$ are connected by an edge if there exists an edge between vertices in the corresponding SCCs. Then we say a graph G is *tree-like* if (1) all its SCCs are cycles; and (2) $c(G)$ is a directed tree. We should note that condition (1) is non-trivial because some SCCs may not be cycles. For instance, in a graph $G = (V, E)$, where $V = \{s_1, s_2, s_3\}$ and $E = \{(s_1, s_2), (s_2, s_3), (s_3, s_3), (s_3, s_2)\}$, the subgraph $G' = (\{s_2, s_3\}, \{(s_2, s_3), (s_3, s_3), (s_3, s_2)\})$ is a SCC, but it is not a cycle because edge (s_3, s_3) also forms a self-loop.

Consider a Kripke model (M, s_0), where $M = (S, R, L)$, $s_0 \in S$. We say that (M, s_0) is a *tree-like Kripke model* if its corresponding graph $G(M) = (S, R)$ is tree-like. In this case, we call the initial state s_0 the *root* of this tree-like model. Since a tree-like Kripke model may not be a strict tree (e.g. it may contain some cycles along a branch), we cannot follow the traditional notions of *child and parent* in a tree-like model. Instead, we define the following new concepts. We say state s is an *ancestor* of state s', if there is a path $\pi = [s_0, \cdots, s, \cdots, s', \cdots]$ where s' does not occur in the part $[s_0, \cdots, s]$. s is a *parent* of s' if s is an ancestor of s' and $(s, s') \in R$. In this case, we also call s' is a *successor* of s. A state s is called *leaf* if it is not an ancestor of any other states. A tree-like Kripke model (M', s') is called a *local model* of (M, s), if $M' = (S', R', L')$, where $s' = s$, $s' \in S'$, $S' \subseteq S$, $R' \subseteq R$, for all $s^* \in S'$, $L'(s^*) = L(s^*)$. Note that a local model must share the same root with the original model.

Since a counterexample may just contain a finite path, to use a tree-like Kripke model to precisely represent such counterexample, we will allow a tree-like Kripke model to contain finite paths that cannot be extended into paths (i.e. infinite sequences of states). Figure 1 from [4] shows an example of a tree-like model that represents a counterexample for a specific ACTL formula.

Clarke et al. [4] proved an important result regarding ACTL model checking stating that if an ACTL formula is not satisfied in a Kripke structure, then this Kripke structure must contain a tree-like counterexample with respect to this formula.

Theorem 1 *[4] ACTL has tree-like counterexamples.*

3 Tree-like Local Model Update

As we mentioned earlier, one major obstacle restricting the application of CTL model update in practical domains is that very often we have to deal with a big Kripke model, and the update may also lead to a model explosion [15]. This motivates us to focus on the tree-like counterexample update. Since we can view a tree-like counterexample as a partial Kripke structure which is usually small, the update on this counterexample actually provides a computer aided approach for effective system modifications.

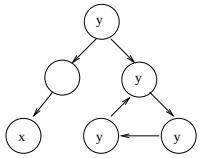

Figure 1. A counterexample for AG$\neg x \vee$AF$\neg y$.

Intuitively, we would like our update approach to obey the following general principles: (1) retain the original structure as much as possible; (2) do not change the structure components that are irrelevant to satisfy the property; (3) allow changes on both transition relations and states in the structure. Principles 1 and 2 are quite obvious: whenever possible, we always like to change the structure as little as possible to make it satisfy the property. Principle 3, on the other hand, means that our update should be flexible enough in order to represent rational modifications in system repairs.

For convenience, in the rest of the paper, we will call (M, s) a *tree-like Kripke model* without explicitly mentioning the corresponding tree-like Kripke structure $M = (S, R, L)$ where $s \in S$. We also define $\mathsf{Diff}(X, Y) = (X \setminus Y) \cup (Y \setminus X)$ where X, Y are two sets.

Definition 2 *Let* (M, s) *and* (M_1, s_1) *be two tree-like Kripke models. We say that a binary relation* $H \subseteq S \times S_1$ *is a* weak bisimulation *between* (M, s) *and* (M_1, s_1) *if:*

1. $H(s, s_1)$;
2. *given* $v, v' \in S$ *such that* v *is a parent of* v', *for all* $v_1 \in S_1$ *such that* $H(v, v_1)$, *the condition holds: (a) if* v_1 *is not a leaf, then there exists successor* v_1' *of* v_1 *such that* $H(v', v_1')$, *or (b) if* v_1 *is a leaf, then* $H(v', v_1)$ *(forth condition);*
3. *given* $v_1, v_1' \in S_1$ *such that* v_1 *is a parent of* v_1', *for all* $v \in S$ *such that* $H(v, v_1)$, *the condition holds: (a) if* v *is not a leaf, then there exists a successor* v' *of* v *such that* $H(v', v_1')$, *or (b) if* v *is a leaf, then* $H(v, v_1')$ *(back condition).*

Definition 2 is inspired from the concept of bisimulation on Kripke models in classical modal logics.It is observed that for any two tree-like models, there exists at least one weak bisimulation between them. Usually, there are more than one such weak bisimulations.

Definition 3 *Let* (M, s), (M_1, s_1) *and* (M_2, s_2) *be three tree-like models,* H_1 *and* H_2 *be two weak bisimulations between* (M, s) *and* (M_1, s_1) *and between* (M, s) *and* (M_2, s_2) *respectively. We say that* H_1 *is* as similar as H_2, *denoted by* $H_1 \leq H_2$, *if for all nodes* $v \in S$, *the following condition holds:*

1. *there exists an ancestor* v' *of* v *such that for all* $v_1 \in S_1$ *and* $v_2 \in S_2$ *satisfying* $H_1(v', v_1)$ *and* $H_2(v', v_2)$, $\mathsf{Diff}(L(v'), L_1(v_1)) \subset \mathsf{Diff}(L(v'), L_2(v_2))$; *or*
2. *for all* $v_1 \in S_1$ *and* $v_2 \in S_2$ *satisfying* $H_1(v, v_1)$ *and* $H_2(v, v_2)$, $\mathsf{Diff}(L(v), L_1(v_1)) \subseteq \mathsf{Diff}(L(v), L_2(v_2))$.

We write $H_1 < H_2$ *iff* $H_1 \leq H_2$ *but* $H_2 \not\leq H_1$.

Definition 3 specifies how we compare two weak bisimulations among three tree-like models. Intuitively, if H_1 and H_2 are two weak bisimulations between (M, s) and (M_1, s_1), and between (M, s) and (M_2, s_2) respectively, then $H_1 \leq H_2$ means that M_1 represents at least the same information about M as M_2 does under H_1 and H_2 respectively.

Note that if (M_1, s_1) and (M_2, s_2) are identical, then it is still possible to have two different weak bisimulations between (M, s) and (M_1, s_1). Hence, we are always interested in that H_1 where there is no other H_1' between (M, s) and (M', s') such that $H_1' < H_1$. We call such H_1 a *minimal weak bisimulation* between (M, s) and (M', s'), which, as should be noted, is not necessarily unique.

Proposition 1 *$\leq$ defined in Definition 3 is a partial ordering.*

Definition 4 (Tree-like local model update) *Let ϕ be an ACTL formula and (M, s) a tree-like model such that $M \not\models \phi$. A tree-like model (M_1, s_1) is called a result of updating (M, s) with ϕ, if and only if*

1. *$(M_1, s_1) \models \phi$;*
2. *there is a weak bisimulation H_1 between (M, s) and (M_1, s_1) such that there does not exist another tree-like model (M_2, s_2) satisfying $(M_2, s_2) \models \phi$ and a weak bisimulation H_2 between (M, s) and (M_2, s_2) such that $H_2 < H_1$. In this case we say that (M_1, s_1) is an update result* under H_1.

We also use $\mathsf{Res}(\mathsf{Update}^{\mathsf{t}}((M, s), \phi))$ to denote the set of all possible resulting tree-like models of updating (M, s) with ϕ.

Condition 1 in Definition 4 simply states that after the update, the resulting tree-like model should satisfy the updating formula. Condition 2 ensures that the resulting tree-like model is minimal from the original model under some weak bisimulation.

Example 1 Consider a tree-like model M as described in Figure 2, which is a counterexample of AG$\neg x \vee$AF$\neg y$. Then according to Definition 4, it is not hard to verify that (M_1, s_1) is a result of the update of (M, s) with AG$\neg x \vee$ AF$\neg y$, where (M_2, s_2) is not although it also satisfies AG$\neg x \vee$ AF$\neg y$, as (M_2, s_2) represents more changes from (M, s) than (M_1, s_1) does. □

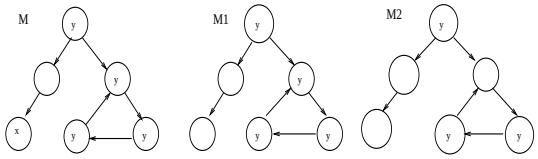

Figure 2. Updating (M, s) with AG$\neg x \vee$ AF$\neg y$.

4 Semantic Characterizations

In this section, we study essential semantic properties of our tree-like local model update. In particular, we will focus our study on two aspects: one is the relationship between our local model update approach and the traditional belief (knowledge system) update, and the other is about the persistence property that we usually expect a local model update to satisfy.

4.1 Relationship to Belief Update

Katsuno and Mendelzon [12] have discovered that the original AGM revision postulates cannot precisely characterize the feature of belief update. They proposed the following alternative update postulates, and argued that any propositional belief update operators should satisfy these postulates. In the following (U1) - (U8) postulates, all occurrences of T, μ, α, etc. are propositional formulas.

(U1) $T \diamond \mu \models \mu$.
(U2) If $T \models \mu$ then $T \diamond \mu \equiv T$.
(U3) If both T and μ are satisfiable then $T \diamond \mu$ is also satisfiable.
(U4) If $T_1 \equiv T_2$ and $\mu_1 \equiv \mu_2$ then $T \diamond \mu_1 \equiv T_2 \diamond \mu_2$.
(U5) $(T \diamond \mu) \wedge \alpha \models T \diamond (\mu \wedge \alpha)$.
(U6) If $T \diamond \mu_1 \models \mu_2$ and $T \diamond \mu_2 \models \mu_1$ then $T \diamond \mu_1 \equiv T \diamond \mu_2$.
(U7) If T is complete (i.e., has a unique model) then $(T \diamond \mu_1) \wedge (T \diamond \mu_2) \models T \diamond (\mu_1 \vee \mu_2)$.
(U8) $(T_1 \vee T_2) \diamond \mu \equiv (T_1 \diamond \mu) \vee (T_2 \diamond \mu)$.

As shown by Katsuno and Mendelzon [12], postulates (U1) - (U8) precisely capture the minimal change criterion for update that is defined based on certain partial ordering on models.

In order to compare our tree-like model update with the traditional knowledge base update, we need to first define a tree-like update operator on ACTL formulas. Let ϕ be an ACTL formula, $M = (S, R, L)$ a tree-like Kripke structure, and $Initial(S) \subseteq S$ the set of *initial states* of M. For some $s \in Initial(S)$, we call (M, s) a *tree-like model* of ϕ if $(M, s) \models \phi$. We use $\mathsf{Mod}^{\mathsf{t}}(\psi)$ to denote the set of all tree-like Kripke models of ψ. We also assume that the underlying language is finite, i.e. the set of propositional variables is finite, because Katsuno and Mendelzon's postulates (U1) - (U8) are based on finite languages [12].

Proposition 2 *If ψ is a satisfiable ACTL formula, then $\mathsf{Mod}^{\mathsf{t}}(\psi) \neq \emptyset$.*

Given two ACTL formulas ψ and ϕ, we specify a tree-like update operator $\diamond_{\mathsf{t}}$ as follows:

$$\mathsf{Mod}^{\mathsf{t}}(\psi \diamond_{\mathsf{t}} \phi) = \bigcup_{(M,s) \in \mathsf{Mod}^{\mathsf{t}}(\psi)} \mathsf{Res}(\mathsf{Update}^{\mathsf{t}}((M, s), \phi)). \quad (1)$$

Theorem 2 *Operator $\diamond_{\mathsf{t}}$ satisfies all Katsuno and Mendelzon update postulates (U1) - (U8), in the sense that for each form $\psi \models \phi$ in the postulates, we replace it as $\mathsf{Mod}^{\mathsf{t}}(\psi) \subseteq \mathsf{Mod}^{\mathsf{t}}(\phi)$.*

Proof: Due to a space limit, here we only prove that $\diamond_{\mathsf{t}}$ satisfies (U5) and (U8), while proofs for other postulates are referred to our full paper. To prove that $\diamond_{\mathsf{t}}$ satisfies (U5), it is sufficient to show that for each $(M, s) \in \mathsf{Mod}^{\mathsf{t}}(\psi)$, $\mathsf{Res}(\mathsf{Update}^{\mathsf{t}}((M, s), \phi)) \cap \mathsf{Mod}^{\mathsf{t}}(\alpha) \subseteq \mathsf{Res}(\mathsf{Update}^{\mathsf{t}}((M, s), \phi \wedge \alpha))$. Consider a tree-like model $(M', s') \in \mathsf{Res}(\mathsf{Update}^{\mathsf{t}}((M, s), \phi)) \cap \mathsf{Mod}^{\mathsf{t}}(\alpha)$. Suppose $(M', s') \notin \mathsf{Res}(\mathsf{Update}^{\mathsf{t}}((M, s), \phi \wedge \alpha))$. Then this implies two cases: (a) $(M', s') \not\models \phi \wedge \alpha$; (b) there exists another tree-like model $(M'', s'') \in \mathsf{Mod}^{\mathsf{t}}(\phi \wedge \alpha)$ such that $H'' < H'$, where H', H'' are the weak bisimulations between (M, s) and (M', s'), (M, s) and (M'', s'') respectively. If it is case (a), then $(M', s') \notin \mathsf{Res}(\mathsf{Update}^{\mathsf{t}}((M, s), \phi)) \cap \mathsf{Mod}^{\mathsf{t}}(\alpha)$, so the result holds. If it is case (b), then it means that $(M', s') \notin \mathsf{Res}(\mathsf{Update}^{\mathsf{t}}((M, s), \phi))$ according to Definition 4, and hence $(M', s') \notin \mathsf{Res}(\mathsf{Update}^{\mathsf{t}}((M, s), \phi)) \cap \mathsf{Mod}^{\mathsf{t}}(\alpha)$. The result also holds.

Now we prove that $\diamond_{\mathsf{t}}$ satisfies (U8). From (1), it is clear that: $\mathsf{Mod}^{\mathsf{t}}((\psi_1 \vee \psi_2) \diamond_{\mathsf{t}} \phi) = \bigcup_{(M,s) \in \mathsf{Mod}^{\mathsf{t}}(\psi_1 \vee \psi_2)} \mathsf{Res}(\mathsf{Update}^{\mathsf{t}}((M, s), \phi)) = \bigcup_{(M,s) \in \mathsf{Mod}^{\mathsf{t}}(\psi_1)} \mathsf{Res}(\mathsf{Update}^{\mathsf{t}}((M, s), \phi)) \cup \bigcup_{(M,s) \in \mathsf{Mod}^{\mathsf{t}}(\psi_2)} \mathsf{Res}(\mathsf{Update}^{\mathsf{t}}((M, s), \phi)) = \mathsf{Mod}^{\mathsf{t}}(\psi_1 \diamond_{\mathsf{t}} \phi) \cup \mathsf{Mod}^{\mathsf{t}}(\psi_2 \diamond_{\mathsf{t}} \phi)$. This means $\diamond_{\mathsf{t}}$ satisfies postulate (U8). □

4.2 Persistence Properties

An essential semantic property we should study in relation to model update is so called *persistence*. That is, for a given tree-like model (M, s) and two ACTL formulas ϕ and ψ, where $(M, s) \not\models \phi$ and $(M, s) \models \psi$, after updating (M, s) with ϕ we obtain a new tree-like local model (M', s') such that $(M', s') \models \phi$. Then we would like to know whether ψ still holds in the new model: $(M', s') \models \psi$? In general updating a model with one formula may affect the satisfaction of other formulas in the resulting model. Study on the persistence property in a local model update is important, because this will provide essential information of how a specific local model update influences other properties that the system originally obeys. The following general result indicates that our update does not affect those irrelevant formulas' satisfactions in the resulting local model.

Proposition 3 *Let (M, s) a tree-like model, ϕ and ψ two ACTL formulas such that $Var(\phi) \cap Var(\psi) = \emptyset$[2], and (M', s') a tree-like model resulting from the update of (M, s) with ϕ. Then $(M', s') \models \psi$ iff $(M, s) \models \psi$.*

However, the situation becomes complicated when ϕ and ψ share some common propositional variables. In general, the persistence property does not hold any more in a local model update when two formulas share common propositional variables. What makes this problem meaningful and challenging is to identify some useful cases for which the local model update preserves the persistence property for a class of ACTL formulas.

Definition 5 (Strict extension) *A tree-like model (M, s) is called a* strict extension *of (M', s') if for each path in (M', s') $\pi' = [s_0, s_1, \cdots]$ ($s_0 = s'$), there is at most one path in (M, s) $\pi = [s_0, \cdots, s_k, s_{k+1}, \cdots]$ ($s_0 = s'$), such that for all $s_i \leq s_k$, $s_i \in \pi'$, and for all $s_k < s_j$, $s_j \notin \pi'$.*

Intuitively, if (M, s) is a strict extension of (M', s'), then (M, s) does not contain any more branches than (M', s') but may contain *longer* paths than (M', s') does. Strict extensions represent some interesting cases in tree-like local model updates. Quite often, an update result may be obtained by only *cutting off* or *extending* some paths of the original local model. The following theorem reveals an important persistence property associated to strict extensions.

Theorem 3 *Let (M, s) be a tree-like model and ϕ an ACTL formula. Then the following results hold:*

1. *If (M', s') is a result of updating (M, s) with ϕ, and it is a strict extension of (M, s), then for any ACTL formula ψ not containing operator* AG, *$(M, s) \models \psi$ implies $(M', s') \models \psi$;*
2. *If (M', s') is a result of updating (M, s) with ϕ, and (M, s) is a strict extension of (M', s'), then for any ACTL formula ψ only containing operator* AG, *$(M, s) \models \psi$ implies $(M', s') \models \psi$.*

Proof: Here we only prove Result 1 here, while result 2 can be proved in a similar style. We prove Result 1 by induction on the structure of formula ψ. More specifically, it is sufficient to prove the cases of propositional formula ψ, AXψ, AFψ, and A[ϕUψ]. Let $M = (S, R, L)$ and $M' = (S', R', L')$. From Definition 5, we know that (M', s') must contain the same branches as (M, s) contains, and $s = s'$. We first consider that ϕ is just a propositional formula. Then we have $(M, s) \models \phi$ iff $L(s) \models \phi$. This means $(M', s') = (M', s) \models \phi$. Now suppose ϕ is of the form AXψ. Since $(M, s) \models$ AXψ, we know that for each $s^* \in S$ such that $(s, s^*) \in R$, $(M, s^*) \models \psi$. Again, because (M', s') is a strict extension of (M, s), all such (s, s^*) are also in R', so $(M', s^*) \models \psi$. Furthermore, there is no other new state $s^\dagger$ such that $(s, s^\dagger) \in R'$ but $(s, s^\dagger) \notin R$. So $(M', s') \models$ AXψ as well. Suppose ϕ is of the form AFψ. From $(M, s) \models$ AFψ, we know that for each path $\pi = [s, \cdots]$ in M, there exists some $s_k \in \pi$ such that $(M, s_k) \models \psi$. Then this path must also be in M', so we also have $(M', s_k) \models \psi$. On the other hand, from the fact that (M', s') is a strict extension of (M, s), we know that in M', there does not exist a path of the form $\pi' = [s', \cdots, s_k, s_{k+1}, \cdots]$ where states $s_k \in S$, and $s_{k+1}, \cdots \in S'$ and another path $\pi'' = [s', \cdots, s_k, s'_{k+1}, \cdots]$ is in M and also in M' ($s_{k+1} \neq s'_{k+1}$). That means, it is not possible that state s_k leads to two different paths in M'. This implies that $(M', s) \models$ AFψ. The case of A[ϕUψ] can be shown similarly. □

[2] $Var(\phi)$ is the set of all propositional variables occurring in ϕ.

Example 2 The tree-like local model (M, s) depicted in the following figure represents a counterexample of ACTL formula A(aUb) $\vee$ AXa. From Definition 4, it can be checked that model (M', s) on the right side in Figure 3 is one possible result of updating (M, s) with formula A[aU($\neg a \wedge b$)] $\vee$ AXa. Clearly, (M', s) is a strict extension of (M, s) in this case. Consider another formula AFAX($a \vee c$). It is observed that $(M, s) \models$ AFAX($a \vee c$). According to Result 1 in Theorem 3, we should have $(M', s) \models$ AFAX($a \vee c$) as well. Indeed it is easy to verify that this is true. □

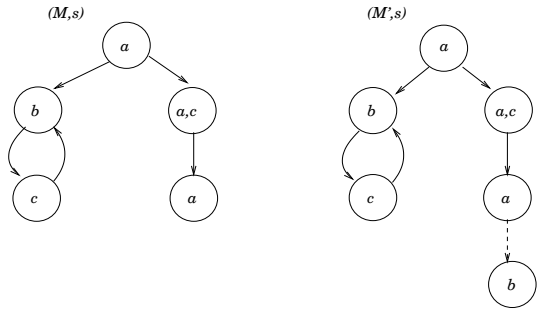

Figure 3. Updating (M, s) with A[aU($\neg a \wedge b$)] $\vee$ AXa preserves the persistence of AFAX($a \vee c$).

5 Computational Properties

In this section we study the computational properties in relation to tree-like local model update. We will first provide the main complexity result of local model update, and then study the computational issues of some typical tree-like local model updates.

5.1 The Main Complexity Result

We first provide the main complexity result of tree-like local model update as follows.

Theorem 4 *Let (M, s) and (M', s') be two tree-like models, H a weak bisimulation between (M, s) and (M', s'), and ϕ an ACTL formula. Deciding whether (M', s') is a result of updating (M, s) with ϕ under H as defined in Definition 4 is co-NP-complete.*

Proof: (Membership). To show the membership, we have first proved the following result:

Result: Given models (M, s), (M', s') and a weak bisimulation H between (M, s) and (M', s'). There exist a model (M'', s'') and a weak bisimulation H'' between (M, s) and (M'', s'') such that $H'' < H$, iff there exist a model $(M^\dagger, s^\dagger)$, whose size is in the polynomial size of (M, s) and (M', s'), and a $H^\dagger$ between (M, s) and $(M^\dagger, s^\dagger)$ such that $H^\dagger < H$.

Suppose $M' = (S', R', L')$ and $s' \in S'$. It is well known that checking whether $(M', s') \models \phi$ can be done in $\mathcal{O}(|\phi| \cdot (|S'| + |R'|))$. Now we need to check (M', s')'s minimality according to Definitions 2 and 3. For doing that, we consider the complement of the problem: checking whether (M', s') is *not* an update result. First, we guess another tree-like model (M'', s'') and a weak bisimulation H'' between (M, s) and (M'', s''). From the above Result, we know that it is sufficient to just guess a (M'', s'') which is in polynomial size of (M, s) and (M', s'). According to Definition 2, on the other hand, guessing a H'' between (M, s) and (M'', s'') can also be done in polynomial time. Then we check whether $(M'', s'') \models \phi$. Obviously, checking whether $(M'', s'') \models \phi$ is in polynomial time. From Definition 3, it is easy to see that deciding whether $H'' < H$ is also in polynomial time. So deciding whether (M', s') is not a result is in NP. That is, the original problem is in co-NP.

(Hardness). It is well known that the validity problem for a propositional formula is co-NP-complete. Given a propositional formula ϕ, we construct a transformation from the problem of deciding ϕ's validity to a tree-like model update in polynomial time. Let X be the set of all variables occurring in ϕ, and a, b two new variables do not occur in X. We denote $\neg X = \bigwedge_{x_i \in X} \neg x_i$. Then, we specify a tree-like Kripke model based on the variable set

$X \cup \{a, b\}$: $M = (\{s_0, s_1\}, \{(s_0, s_1), (s_1, s_1)\}, L)$, where $L(s_0) = \emptyset$ and $L(s_1) = X$.

Now we define a new formula $\mu = \mathrm{AX}(((\phi \rightarrow a) \wedge (\neg X \wedge b)) \vee (\neg\phi \wedge a))$. Clearly, formula $((\phi \rightarrow a) \wedge (\neg X \wedge b)) \vee (\neg\phi \wedge a)$ is satisfiable and $s_1 \not\models ((\phi \rightarrow a) \wedge (\neg X \wedge b)) \vee (\neg\phi \wedge a)$. So $(M, s_0) \not\models \mu$. Now we consider the update of (M, s_0) with μ. We define a new tree-like model $M' = (\{s'_0, s'_1\}, \{(s'_0, s'_1), (s'_1, s'_1)\}, L')$, where $L'(s'_0) = L(s_0)$ and $L'(s'_1) = \{a, b\}$. The weak bisimulation H between (M, s_0) and (M', s'_0) is defined as $H(s_0, s'_0)$ and $H(s_1, s'_1)$. Then it is not difficult to verify that (M', s'_0) is a result of updating (M, s_0) with μ iff ϕ is valid. This completes our proof. □

Theorem 4 provides an essential computational insight for tree-like local model update. It implies that, unless P=NP, it is unlikely to develop a polynomial time algorithm to compute an update result. Indeed, our implementation algorithm for local model update runs in exponential time generally. Furthermore, without giving the specific bisimulation in the input, the model checking complexity as stated in Theorem 4 will be in Π_2^P.

5.2 Computing Typical Tree-like Local Model Updates

Although computing local model update is generally expensive, we have observed that many updates with typical ACTL formulas can actually be achieved in a more effective way. In the following, we provide complexity results for these typical updates.

Theorem 5 *Let (M, s) be a tree-like Kripke model, where $M = (S, R, L)$ and $s \in S$, ϕ and ψ two propositional formulas. Then the following results hold.*

1. *If $(M, s) \not\models \mathrm{AX}\phi$, then a resulting tree-like model (M', s') can be computed in time $\mathcal{O}(|R| \cdot 2^{|var(\phi)|})$;*
2. *If $(M, s) \not\models \mathrm{AG}\phi$, then a resulting tree-like model (M', s') can be computed in time $\mathcal{O}(|S| \cdot 2^{|var(\phi)|})$;*
3. *If $(M, s) \not\models \mathrm{AF}\phi$, then a resulting tree-like local model (M', s') can be computed in time $\mathcal{O}(|R| \cdot |S| \cdot 2^{|var(\phi)|})$;*
4. *If $(M, s) \not\models \mathrm{A}(\psi \mathrm{U} \phi)$, then a resulting tree-like local model (M', s') can be computed in time $\mathcal{O}(|R| \cdot |S| \cdot 2^{|var(\phi)|})$;*
5. *If $(M, s) \not\models \mathrm{AG}(\psi \rightarrow \mathrm{AF}\phi)$, then a resulting tree-like model (M', s') can be computed in time $\mathcal{O}(|R| \cdot |S| \cdot 2^{|var(\phi)|})$.*

Proof: (*Sketch*) Here we only give the proof outline of Result 3 due to a space limit. Since (M, s) is a counterexample of AFϕ, then there must be a path in (M, s) $\pi = [s_0, s_1, \cdots]$ ($s_0 = s$) such that for each $s_i \in \pi$, $L(s_i) \not\models \phi$. There are two cases.

Case 1. π is a finite path. Recall that under the weak bisimulation minimal change principle, whenever possible, we try to update the tree-like model at as lowest level as possible. In this case, we simply update the last state s_k in π to s'_k, such that under the new model's labeling function L', $L'(s'_k) \models \phi$ and $\mathsf{Diff}(L(s_k, L'(s'_k))$ is minimal. If there is no any other path in (M, s) violates Fϕ, then it is easy to see the new model (M', s') obtained in this way is a resulting model with respect to Definition 4.

Case 2. π is an infinite path, that is π ends up with a loop: $\pi = [s_0, s_1, \cdots, s_k, s_{k+1}, \cdots, s_{k+h}, s_k, \cdots]$, as shown in Figure 4.

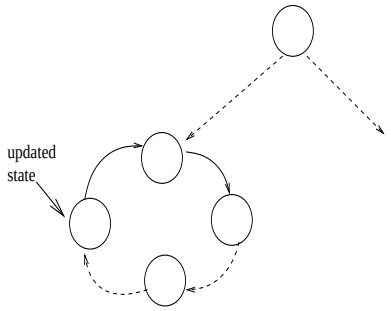

Figure 4. A case of update with AFϕ.

In this case, the lowest level of path π is the state s_{k+h}: the last state in the loop back to the loop entry state s_k (i.e. the arrow pointed state in Figure 4). So we just update s_{k+h} to satisfy formula ϕ minimally. Once we complete the update on the all paths that violate Fϕ, the resulting model will satisfy AFϕ, and is minimal from the original model (M, s).

Now let us consider the complexity of this update procedure. Clearly, for each such path, we need to update exactly one state, which costs time $\mathcal{O}(2^{|var(\phi)|})$[3]. Given the tree-like counterexample (M, s) of formula AFϕ, generating a path of (M, s) and the update on the last state will cost $\mathcal{O}(|R| + 2^{|var(\phi)|})$ at most. Since there are at most $|S|$ states we need to consider for the update, the total cost will be no more than $\mathcal{O}(|R| \cdot |S| \cdot 2^{|var(\phi)|})$. □

6 System Implementation and Experimental Results

We have implemented a system prototype that performs ACTL tree-like local model update. The system takes the counterexample generated from NuSMV model checker as its input, updates this local model with the underlying property, and produces an updated local

[3] Such complexity $\mathcal{O}(2^{|var(\phi)|})$ for individual state minimal update is inevitably inherited from the the classical model based update approach [14].

model that satisfies the property. All these procedures are undertaken in an automatic way.

After getting the updated local model, it is then the designer's task to find an actual fix for the original system. We have observed that for most situations, detailed information about the original system's errors correction has already been revealed through the updated local model. So it is usually not very difficult to find a candidate fix. Once a candidate fix is identified, another procedure of model checking and update should be carried out in order to achieve a final correction. The structure of our update prototype is depicted as follows.

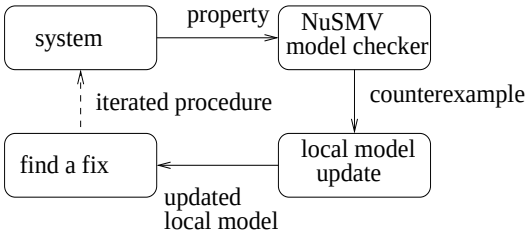

Figure 5. Local model update prototype.

We have undertaken extensive testings and case studies to evaluate the performance of this update prototype system. Due to a space limit, in the following, we just provide a brief summary of the experimental results for one particular case study we have done - Sliding Window Protocol (SWP) update.

Sliding Window Protocol (SWP) Update

SWP is a common protocol used for point to point packet-based communication in networked environments [6]. Generally speaking, there are two main components in SWP: the sender and the receiver. The sender sends messages sequentially (each is called a *frame message*) to the receiver through frame channel. When the receiver eventually receives these frame messages, it will acknowledge the sender through the acknowledgement channel.

If all frame messages in the *sending window* received acknowledgement, then the sending window slides forward; similarly, if the receiver received all frame messages in the *receiving window*, then the receiving window slides forward. The size of sending window and receiving window is always the same. Since channels are possibly unreliable, during the transmission, however, messages could be lost, reordered and duplicated.

In recent years, various methods for verifying SWP have been developed, e.g. [5]. Basically, to ensure the correctness of SWP, the verification property, in some way, is of the form $\varphi : \bigwedge_i \mathrm{AG}(sent_i \rightarrow \mathrm{AF}received_i)$, stating if the sender sends a frame message, then it will be eventually received by the message receiver. Here $sent_i$ and $received_i$ are pair of variables associated with the i^{th} frame message in the window.

In our experiment, we encoded SWP and the checking property ϕ via a NuSMV program, and allowed the window sizes vary from 2 to 8 in order to demonstrate the performance on different parameters. The following table shows our experimental results for SWP update at different window sizes[4].

From Figure 6, we can see that with the window size increasing, the state explosion is getting serious, while the size of local model (counterexample size) remains relatively steady. We should also indicate that during the model update, the number of propositional variables that the update prototype has to deal with is $2n$, where n is the size of window. This implies that for each state update, it will take time $\mathcal{O}(2^{2n})$. For window size 8, for example, it will take $\mathcal{O}(2^{16})$ for each state update in the worst case. Since the counterexample size remains quite small comparing to the whole model size, our prototype is able to perform effective updates and assist to derive actual fixes.

[4] Due to a space limit, the detailed procedure of SWP verification and update in our system has to be referred to our full paper.

Window size	Counterexample size (state numbers)	Complete model size (state numbers)	Update time (seconds)
2	7	8	0.344
3	13	27	1.938
4	12	64	5.813
5	19	125	22.531
6	31	216	94.532
7	32	343	278.844
8	33	512	1020.688

Figure 6. Results of Update on Sliding Window Size n = 2..8

7 Concluding Remarks

In this paper we developed an approach for ACTL tree-like local model update. We studied semantic and computational foundations of this approach in details. Our experimental results showed the feasibility of this approach in applications.

We should mention that using our approach, the result produced from a local model update is not an actual fix of the original system itself, instead, it is a local model that the correct system *should* contain in order to satisfy the checking property. This resulting local model usually presents concrete information helping the designer to find an actual fix for the original system. More detailed studies on this aspect have been reported in our another paper [13].

REFERENCES

[1] B. Berard and et al., *System and Software Verifi cation: Model-Checking Techniques and Tools*, Springer, 2001.

[2] F. Buccafurri, T. Eiter, G. Gottlob, and N. Leone, 'Enhancing model checking in verification by ai techniques', *Artifi cial Intelligence*, **112**, 57–105, (1999).

[3] F. Buccafurri, T. Eiter, G. Gottlob, and N. Leone, 'Actl formulas having linear counterexamples', *J. Comp & Sys Sci*, **62**, 463–515, (2001).

[4] E.Jr. Clarke, S. Jha, Y. Lu, and H. Veith, 'Tree-like counterexamples in model checking', in *Proceedings of LICS'02*, pp. 19–29, (2002).

[5] W. Fokkink and et al, 'Verifying a sliding window protocol in μCRL', in *Proceedings of AMAST*, pp. 148–163, (2004).

[6] B. Forouzan, *Data Communications and Networking*, Mc Graw Hill, 2003.

[7] G. Fraser and F. Wotawa, 'Nondeterministic testing with linear model-checking counterexamples', in *Proceedings of the 7th Internaitonal Conference on Quality Software (QSIC 2007)*, (2007).

[8] A. Griesmayer, R. Bloem, and B. Cook, 'Repair of boolean programs with an application to C', in *CAV 2006*, pp. 358–371, (2006).

[9] H. Harris and M. Ryan, 'Theoretical foundations of updating systems', in *Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE 2003)*, pp. 291–298, (2003).

[10] M. Huth and M. Ryan, *Logic in Computer Science: Modelling and Reasoning about Systems (2nd edition)*, Cambridge University Press, 2004.

[11] B. Jobstmann, S. Staber, A. Griesmayer, and R. Bloem, 'Finding and fixing faults', *Journal of Computer and System Sciences*, (2010).

[12] H. Katsuno and A. Mendelzon, 'On the difference between updating a knowledge base and revising it', in *KR'91*, pp. 387–394, (1991).

[13] M. Kelly and Y. Zhang, 'Local model update with an application to sliding window protocol', in *Proceedings of KES 2010*, (2010 (to appear)).

[14] M. Winslett, *Updating Logical Databases*, MIT, 1990.

[15] Y. Zhang and Y. Ding, 'CTL model update for system modifications', *Journal of Artifi cial Intelligence Research*, **31**, 113–155, (2008).