Artificial Intelligence 211 (2014) 51–74

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

CTL update of Kripke models through protections ☆

Miguel Carrillo, David A. Rosenblueth *

Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, Universidad Nacional Autónoma de México, Apdo. 20-126, 01000 México D.F., México

ARTICLE INFO

Article history:
Received 22 September 2012
Received in revised form 24 February 2014
Accepted 24 February 2014
Available online 2 March 2014

Keywords:
CTL model update
Model checking
Automatic synthesis

ABSTRACT

We present a nondeterministic, recursive algorithm for updating a Kripke model so as to satisfy a given formula of computation-tree logic (CTL). Recursive algorithms for model update face two dual difficulties: (1) *Removing* transitions from a Kripke model to satisfy a *universal* subformula may dissatisfy some *existential* subformulas. Conversely, (2) *adding* transitions to satisfy an *existential* subformula may dissatisfy some *universal* subformulas. To overcome these difficulties, we employ protections of the form $\langle E, A, L\rangle$, recording information about the satisfaction of subformulas previously treated by the algorithm. Intuitively, (1) E is the set of transitions that we *cannot remove* without compromising the satisfaction of previously treated subformulas. Conversely, (2) A is the set of transitions that we *can add*. Hence, update proceeds without diminishing E and without augmenting A. Finally, (3) L is a set of literals protecting the model labels. We illustrate our algorithm through several examples: Emerson and Clarke's mutual-exclusion problem, Clarke et. al.'s microwave-oven example, synchronous counters, and randomly generated models and formulas. In addition, we compare our method with other update approaches for either CTL or fragments of CTL. Lastly, we provide proofs of soundness and completeness and a complexity analysis.

1. Introduction

A computation-tree logic (CTL) model checker is an automated tool that usually has as input (1) a Kripke model $\mathcal{M}$ formalizing a system, (2) a CTL formula φ expressing a desirable property of this system, and (3) a set of initial states. The output is either a confirmation or a denial that $\mathcal{M}$ satisfies φ at all initial states, meaning that the system respectively has, or does not have, the required property.[1] In case of denial, model checkers often produce a *counterexample*, consisting of an error trace. This counterexample is intended as a guide for manually updating or repairing $\mathcal{M}$, or a high-level description of $\mathcal{M}$, so that the unsatisfied property is fulfilled. We believe that even a partial automation of such a repairing process could have a significant impact on the use of the model checking technique. Besides, the problem of model update is closely related to that of reasoning about actions and change (e.g., [14]). Model update is therefore an important subject in Artificial Intelligence, namely that of modifying transition systems.

☆ This article extends a version presented at ATVA 2011 [5] with soundness and completeness proofs, complexity analysis, benchmarks on larger models, and a comparison with related approaches.

* Corresponding author.
E-mail addresses: miguel.mcb@gmail.com (M. Carrillo), drosenbl@unam.mx (D.A. Rosenblueth).

[1] The CTL model checking problem is sometimes also defined as the problem of computing the set of all states s such that $\mathcal{M}$ satisfies φ at s.

http://dx.doi.org/10.1016/j.artint.2014.02.005

In spite of its relevance, the problem of mechanically updating a Kripke model has been little studied. As far as we know, Buccafurri et al. [1] proposed the first work on CTL model update: an updater for repairing, through abduction, Kripke models with the addition and removal of transitions. In a later work, Calzone et al. [3] gave a method for updating Kripke models with the addition and removal not only of transitions, but of labels as well, dependent on biases determined by the application domain (biochemical networks). More recently, Zhang and Ding [22] have devised a repair algorithm producing "admissible" models.

Model update methods based on counterexamples produced by a model checker have a drawback. Model repair focused on invalidating one particular counterexample to a *universal* property does not guarantee that the repaired model satisfies this property, because there may be more than one counterexample, and all counterexamples must be treated simultaneously. Furthermore, if the defective model does not satisfy an *existential* property, the model checker does not provide a useful counterexample, because all traces are counterexamples. Hence, this drawback invites us to consider a method based on a concept other than counterexamples.

We present a recursive method for repairing models in a nondeterministic way, based on the preservation of the satisfaction of subformulas via a mechanism of "protections". To update a model with respect to a formula φ, our method recursively updates the model to satisfy the subformulas of φ. Every time our method updates a model to satisfy a subformula α, the satisfaction of α is protected. This protection ensures that if α is a proper subformula of β, then an update to satisfy β causes no loss of the satisfaction of α achieved by a previous update. To facilitate the treatment of negation, our algorithm requires that CTL formulas be in *negation normal form* and written with a set of operators *closed under duality*.

A previous version [5] of this article showed the behavior of a software tool implementing our algorithm using Emerson and Clarke's mutual-exclusion problem [15]. We have now extended that study to include more examples: Clarke et al.'s microwave-oven model [11] illustrating a specification, a scalable counter exhibiting larger models, and randomly generated models and formulas also showing larger models. In addition, [5] only outlined soundness and completeness proofs of our method, whereas we now give more detailed proofs, and perform a complexity analysis of our algorithm. Finally, the cursory comparison with two methods in [5] has now been detailed and extended to more approaches for either CTL or fragments of CTL.

In another previous work [4], we extended a former version of the method presented here so as to modify a concise representation of a Kripke model instead of the Kripke model itself. The former version, however, employed a more primitive form of universal protection (covered in Section 6). This kind of protection, unlike the one used here, is unsatisfactory because it is unclear if the resulting algorithm is complete.

After fixing the notation in Section 2, we treat CTL model update in Section 3. Next, we devote Section 4 to nondeterminism elimination. Examples illustrating the behavior of a software tool implementing our algorithm appear in Section 5; comparisons with other approaches occur in Section 6. Section 7 gives a summary and outlines possible future directions for research. The soundness and completeness proofs are in Appendix A; the complexity analysis is in Appendix B.

2. Technical preliminaries

This section gives introductory definitions and fixes the notation. We assume some familiarity with CTL model checking and refer the reader to [11] for a more thorough treatment.

A *signature* $\Sigma = \langle S, V\rangle$ consists of a pair of nonempty finite sets. We call the elements of S and V, *states*, and *(propositional) variables*, respectively. Unless otherwise stated, we assume that $\Sigma = \langle S, V\rangle$ is *an arbitrary fixed signature*. If $\mathcal{E} = \langle c_1, \ldots, c_n\rangle$, $c_i^{\mathcal{E}}$ is the *i-th component* of $\mathcal{E}$. The set of *literals* over V^Σ is $\mathrm{Lit}(V^\Sigma) = V^\Sigma \cup \{\neg p \mid p \in V^\Sigma\}$. The *complement* of literals is defined by $\overline{p} = \neg p$ and $\overline{\neg p} = p$, $\forall p \in V^\Sigma$. If $X \subseteq \mathrm{Lit}(V^\Sigma)$ then: $\overline{X} = \{\overline{\ell} \mid \ell \in X\}$; X is *consistent* if $\forall \ell \in X,\ \overline{\ell} \notin X$; and X is V^Σ*-maximal* if for all $p \in V^\Sigma$, $p \in X$ or $\neg p \in X$. If $R \subseteq S^\Sigma \times S^\Sigma$ then R is *total* if $\forall s \in S^\Sigma$, $\exists t \in S^\Sigma$ such that $(s,t) \in R$. The set of *successors of* s under R is $R[s] = \{t \mid (s,t) \in R\}$. If $A \cap B = \emptyset$, $f : A \to C$, and $g : B \to D$, $f \cup g$ is the function $f \cup g : A \cup B \to C \cup D$ such that $f \cup g(t) = f(t)$ if $t \in A$ and $f \cup g(t) = g(t)$ if $t \in B$. We use I_D to denote the identity on D, and C_D to denote a constant function such that $\forall t \in D,\ C_D(t) = C$. To denote the symmetric difference of C and D, we use $\Delta(C, D)$.

Definition 2.1 *(Kripke Σ-models).* We say that $\mathcal{M} = \langle S, R, L\rangle$ is a *Kripke Σ-model* if $S = S^\Sigma$, $R \subseteq S^2$ is total, and $L : S \to \mathcal{P}(\mathrm{Lit}(V^\Sigma))$ is such that for all $t \in S$, $L(t)$ is consistent and V^Σ-maximal.

If $(s,t) \in R$ we call (s,t) a *transition from s to t* and we abbreviate this to sRt. We call L the *labeling function* of $\mathcal{M}$. If $\ell \in \mathrm{Lit}(V^\Sigma)$ and $s \in S$, then $L[s \oplus \ell]$ denotes the labeling function such that $L[s \oplus \ell](s) = (L(s) \cup \{\ell\}) - \{\overline{\ell}\}$ and $L[s \oplus \ell](t) = L(t)$ for $t \neq s$. $\mathbf{K}_\Sigma$ denotes the *set of Σ-models*.

We define the *distance* between $\mathcal{M}$ and $\mathcal{M}' \in \mathbf{K}_\Sigma$, as $d(\mathcal{M}, \mathcal{M}') = |\Delta(S^{\mathcal{M}}, S^{\mathcal{M}'})| + |\Delta(R^{\mathcal{M}}, R^{\mathcal{M}'})| + \sum_{t \in S^{\mathcal{M}}} (|\Delta(L^{\mathcal{M}}(t), L^{\mathcal{M}'}(t))|/2)$. It can be proved that d is a metric on $\mathbf{K}_\Sigma$.

Models are often represented graphically as in Section 5, writing only positive literals as labels of the states.

Next, we define an extension of CTL. We use a base of operators *closed under duality* with the following pairs of *dual operators*: (F, T), $(\vee, \wedge)$, $(\mathbf{EX}, \mathbf{AX})$, $(\mathbf{EU}, \mathbf{AR})$, and $(\mathbf{AU}, \mathbf{ER})$. We restrict formulas to a *negation normal form* (**NNF**), by limiting the application of negation to variables. However, since we use a base of operators closed under duality, other instances of

negation may be considered as shorthand. We add to CTL *out-degree formulas* $\mathbf{OD}_{\leqslant} n$, and their duals $\mathbf{OD}_{>} n$, to limit the number of transitions going out of a state.

Definition 2.2 *(Σ-CTL and Σ-XCTL). Formulas of computation-tree logic of signature Σ, Σ-CTL* (abbreviated Φ), have the following syntax:

$$\Phi ::= \mathsf{F} \mid \mathsf{T} \mid \ell \mid (\Phi \vee \Phi) \mid (\Phi \wedge \Phi) \mid (\mathbf{EX}\,\Phi) \mid (\mathbf{AX}\,\Phi) \mid \mathbf{OD}_{\leqslant} n \mid \mathbf{OD}_{>} n$$
$$\mid \mathbf{E}[\Phi\,\mathbf{U}\,\Phi] \mid \mathbf{A}[\Phi\,\mathbf{U}\,\Phi] \mid \mathbf{E}[\Phi\,\mathbf{R}\,\Phi] \mid \mathbf{A}[\Phi\,\mathbf{R}\,\Phi]$$

where ℓ stands for any literal in $\mathrm{Lit}(V^{\Sigma})$ and $0 < n \leqslant |S^{\Sigma}|$.

We will single out *the modal fragment of Σ-CTL. Σ-XCTL formulas* (abbreviated Ψ) have the following syntax:

$$\Psi ::= \mathsf{F} \mid \mathsf{T} \mid \ell \mid (\Psi \vee \Psi) \mid (\Psi \wedge \Psi) \mid (\mathbf{EX}\,\Psi) \mid (\mathbf{AX}\,\Psi) \mid \mathbf{OD}_{\leqslant} n \mid \mathbf{OD}_{>} n$$

We use $\varphi \in \Sigma$-CTL to indicate that φ is a Σ-CTL formula. The size of $\varphi \in \Sigma$-CTL, written $|\varphi|$, is defined by $|\mathsf{F}| = |\mathsf{T}| = 0$, $|\ell| = |\mathbf{OD}_{\leqslant} n| = |\mathbf{OD}_{>} n| = 1$, $|\mathbf{EX}\alpha| = |\mathbf{AX}\alpha| = |\alpha| + 1$, and $|\alpha \vee \beta| = |\alpha \wedge \beta| = |\mathbf{E}[\alpha\,\mathbf{U}\,\beta]| = |\mathbf{A}[\alpha\,\mathbf{U}\,\beta]| = |\mathbf{E}[\alpha\,\mathbf{R}\,\beta]| = |\mathbf{A}[\alpha\,\mathbf{R}\,\beta]| = |\alpha| + |\beta| + 1$.

If φ is built from *atomic formulas* (F, T, or ℓ) by using $\vee$ or $\wedge$, we say that φ is *propositional*; otherwise we say that φ is a *non-propositional formula*. We view other propositional operators ($\rightarrow$, $\leftrightarrow$, exclusive or $\veebar$) and temporal operators (**EF**, **AF**, **EG**, **AG**) as abbreviations: $\mathbf{EF}\alpha = \mathbf{E}[\mathsf{T}\,\mathbf{U}\,\alpha]$, $\mathbf{AF}\alpha = \mathbf{A}[\mathsf{T}\,\mathbf{U}\,\alpha]$, $\mathbf{EG}\alpha = \mathbf{E}[\mathsf{F}\,\mathbf{R}\,\alpha]$, $\mathbf{AG}\alpha = \mathbf{A}[\mathsf{F}\,\mathbf{R}\,\alpha]$.

The above CTL syntax is a key point in the simplicity of our model update algorithm. Any syntax that has a base of operators similar to those used in model checking [11], e.g., $\{\neg, \wedge, \mathbf{EX}, \mathbf{AF}, \mathbf{EU}\}$, complicates model update w.r.t. formulas $\neg\alpha$, $\mathbf{AF}\alpha$, and $\mathbf{E}[\alpha\,\mathbf{U}\,\beta]$. Another key point is that our algorithm focuses on Σ-XCTL formulas, and deals with path operators (**EU**, **AU**, **ER**, and **AR**) through their fixed-point characterizations [11].

Next, we provide basic definitions for the protection mechanism.

Definition 2.3 *(Σ-Protections).* $P = \langle E, A, L\rangle$ is a *Σ-protection* if:

1. $E \subseteq A \subseteq S^{\Sigma} \times S^{\Sigma}$, and
2. $L : S^{\Sigma} \to \mathcal{P}(\mathrm{Lit}(V^{\Sigma}))$ is such that $\forall t \in S^{\Sigma}$, $L(t)$ is consistent.

Protections are the main key point of our update algorithm. Intuitively, a protection $P = \langle E, A, L\rangle$ records information about the satisfaction of previously treated subformulas by our algorithm. In P, E is the set of transitions (s,t) such that (s,t) *cannot be removed* without compromising the satisfaction of previously treated subformulas. Analogously, A is the set of transitions (s,t) such that (s,t) *can be added*. Finally, L is the set of literals ℓ such that ℓ *cannot be changed to* $\bar{\ell}$ without compromising the satisfaction of previously treated subformulas. Transitions related to the satisfaction of universal (existential) subformulas are recorded in A (E). Thus, to preserve the satisfaction of previously treated subformulas, we proceed without diminishing E or augmenting A, and without complementing literals in L.

We refer to states in $E[s]$ ($A[s]$) as the *existentially (universally) protected successors* of s. We will use $\mathbf{P}_{\Sigma}$ to denote the set of Σ-protections.

Definition 2.4 *(Protected models, $P_{\mathcal{M}}$, and $P_{\perp}$).* Let $\mathcal{M} \in \mathbf{K}_{\Sigma}$ and $P \in \mathbf{P}_{\Sigma}$. We say that $\mathcal{M}$ *is protected by* P, and we write $\mathcal{M} \rhd P$ if:

1. $E^P \subseteq R^{\mathcal{M}} \subseteq A^P$, and
2. $\forall t \in S^{\mathcal{M}}$: $L^{\mathcal{M}}(t) \supseteq L^P(t)$.

We say that $(\mathcal{M}, P)$ *is a protected Σ-model* if $\mathcal{M} \rhd P$, and we use $\mathbf{KP}_{\Sigma}$ to denote the set of *protected Σ-models*. The *full Σ-protection of $\mathcal{M}$* is $P_{\mathcal{M}} = \langle R^{\mathcal{M}}, R^{\mathcal{M}}, L^{\mathcal{M}}\rangle$. The *empty Σ-protection* is $P_{\perp} = \langle \emptyset, S^{\Sigma} \times S^{\Sigma}, L_{\perp}\rangle$, where $L_{\perp}(s) = \emptyset$ for all $s \in S^{\Sigma}$.

If $R \subseteq S^2$ and $s \in S$, a *path* in R *beginning at* s, is a sequence $\pi : \mathbb{N} \to S$, such that $\pi(0) = s$ and $\forall n \in \mathbb{N}$, $\pi(n) R \pi(n+1)$. We write π_n instead of $\pi(n)$ and we use $\Pi_{R,s}$ to denote *the set of paths* in R beginning at s.

Definition 2.5 *(Σ-CTL protected semantics).* If $(\mathcal{M}, P) \in \mathbf{KP}_{\Sigma}$ is a protected Σ-model, $s \in S^{\mathcal{M}}$ and $\varphi \in \Sigma$-CTL, then we say that $(\mathcal{M}, P)$ *satisfies* φ at s, and we write $(\mathcal{M}, P), s \models \varphi$, if:

1. $(\mathcal{M}, P), s \not\models \mathsf{F}$. $(\mathcal{M}, P), s \models \mathsf{T}$. $(\mathcal{M}, P), s \models \ell$ if $\ell \in L^P(s)$.
2. $(\mathcal{M}, P), s \models \alpha \vee \beta$ if $(\mathcal{M}, P), s \models \alpha$ or $(\mathcal{M}, P), s \models \beta$.
3. $(\mathcal{M}, P), s \models \alpha \wedge \beta$ if $(\mathcal{M}, P), s \models \alpha$ and $(\mathcal{M}, P), s \models \beta$.

4. $(\mathcal{M}, P), s \models \mathbf{EX}\alpha$ if $\exists t \in E^P[s]$ such that $(\mathcal{M}, P), t \models \alpha$.
5. $(\mathcal{M}, P), s \models \mathbf{AX}\alpha$ if $\forall t \in A^P[s]$, $(\mathcal{M}, P), t \models \alpha$.
6. $(\mathcal{M}, P), s \models \mathbf{E}[\alpha\ \mathbf{U}\ \beta]$ if $\exists \pi \in \Pi_{E^P,s}$, and $\exists j \in \mathbb{N}$ such that $(\mathcal{M}, P), \pi_j \models \beta$ and $\forall i \in \mathbb{N}$, $i < j \rightarrow (\mathcal{M}, P), \pi_i \models \alpha$.
7. $(\mathcal{M}, P), s \models \mathbf{A}[\alpha\ \mathbf{U}\ \beta]$ if $\forall \pi \in \Pi_{A^P,s}$, $\exists j \in \mathbb{N}$ such that $(\mathcal{M}, P), \pi_j \models \beta$ and $\forall i \in \mathbb{N}$, $i < j \rightarrow (\mathcal{M}, P), \pi_i \models \alpha$.
8. $(\mathcal{M}, P), s \models \mathbf{E}[\alpha\ \mathbf{R}\ \beta]$ if $\exists \pi \in \Pi_{E^P,s}$ such that either
   (a) $\forall k \in \mathbb{N}$, $(\mathcal{M}, P), \pi_k \models \beta$, or
   (b) $\exists j \in \mathbb{N}$. $(\mathcal{M}, P), \pi_j \models \alpha$ and $\forall i \in \mathbb{N}$, $i \leqslant j \rightarrow (\mathcal{M}, P), \pi_i \models \beta$.
9. $(\mathcal{M}, P), s \models \mathbf{A}[\alpha\ \mathbf{R}\ \beta]$ if $\forall \pi \in \Pi_{A^P,s}$, either
   (a) $\forall k \in \mathbb{N}$, $(\mathcal{M}, P), \pi_k \models \beta$, or
   (b) $\exists j \in \mathbb{N}$. $(\mathcal{M}, P), \pi_j \models \alpha$ and $\forall i \in \mathbb{N}$, $i \leqslant j \rightarrow (\mathcal{M}, P), \pi_i \models \beta$.
10. $(\mathcal{M}, P), s \models \mathbf{OD}_{\leqslant} n$ if $|A^P[s]| \leqslant n$.
11. $(\mathcal{M}, P), s \models \mathbf{OD}_{>} n$ if $|E^P[s]| > n$.

Definition 2.6 *(Σ-CTL semantics).* If $\mathcal{M}$ is a Σ-model, $s \in S^{\mathcal{M}}$ and $\varphi \in \Sigma$-CTL, we say that $\mathcal{M}$ *satisfies* φ at s, $\mathcal{M}, s \models \varphi$, if $(\mathcal{M}, P_{\mathcal{M}}), s \models \varphi$.

We extend Σ-CTL protected semantics to sets of states and sets of formulas. If $S \subseteq S^{\mathcal{M}}$ then $(\mathcal{M}, P), S \models \varphi$ if for all $s \in S$, $(\mathcal{M}, P), s \models \varphi$. If $\Gamma \subseteq \Sigma$-CTL then $(\mathcal{M}, P), s \models \Gamma$ if for all $\varphi \in \Gamma$, $(\mathcal{M}, P), s \models \varphi$. If φ_1, φ_2 are two Σ-CTL formulas, we say that φ_1 and φ_2 are *logically equivalent*, and we write $\varphi_1 \equiv \varphi_2$, if for all $\mathcal{M} \in \mathbf{K}_\Sigma$ and all $s \in S^{\mathcal{M}}$: $\mathcal{M}, s \models \varphi_1$ if $\mathcal{M}, s \models \varphi_2$.

In Section 5, we use the equivalence $\mathbf{AG}\alpha \wedge \mathbf{AG}\beta \equiv \mathbf{AG}(\alpha \wedge \beta)$ to factor conjunctions of **AG** formulas. Our algorithm uses the following equivalences, known as fixed-point characterizations, to recursively compute a model update in the respective cases:

$$\mathbf{E}[\alpha\ \mathbf{U}\ \beta] \equiv \beta \vee \big(\alpha \wedge \mathbf{EX}\mathbf{E}[\alpha\ \mathbf{U}\ \beta]\big)$$
$$\mathbf{A}[\alpha\ \mathbf{U}\ \beta] \equiv \beta \vee \big(\alpha \wedge \mathbf{AX}\mathbf{A}[\alpha\ \mathbf{U}\ \beta]\big)$$
$$\mathbf{E}[\alpha\ \mathbf{R}\ \beta] \equiv \beta \wedge \big(\alpha \vee \mathbf{EX}\mathbf{E}[\alpha\ \mathbf{R}\ \beta]\big)$$
$$\mathbf{A}[\alpha\ \mathbf{R}\ \beta] \equiv \beta \wedge \big(\alpha \vee \mathbf{AX}\mathbf{A}[\alpha\ \mathbf{R}\ \beta]\big)$$

A model update w.r.t. **EU** and **AU** formulas can be computed with a least fixed-point (lfp) operator, while a model update w.r.t. **ER** and **AR** can be computed with a greatest fixed-point (gfp) operator [11, p. 63].

3. CTL update algorithms

In this section, we describe an algorithm that uses protected models for model update w.r.t. Σ-CTL formulas. After a brief explanation of the pseudo-code notational conventions and basic update operations, we describe two model-update algorithms for Σ-XCTL formulas. The first one is a *direct* algorithm that we include here for the purpose of emphasizing the main features of the model-update problem. The second one implements our method for model update using protected models. Finally, we describe how to extend our method to all formulas in Σ-CTL, and how to add states to the signature Σ if the input formula is not satisfiable by any Σ-model. Appendix A includes a proof of the correctness and completeness of XUPD$_{prot}$, and Appendix B provides a complexity analysis of UPD$_{prot}$.

3.1. Nondeterministic pseudo-code

We use a *nondeterministic pseudo-code*. Besides usual control statements and procedures, our pseudo-code has the following commands.

Given a finite multiset A, the command "**guess** $x \in A$" computes a *nondeterministic choice* of an element u from A and assigns u to x; if $A = \emptyset$, *the computation fails*. The command "$x \leftarrow e$" is a *nondeterministic assignment* equivalent to "**guess** $x \in A_e$", where A_e is the multiset of values produced by the nondeterministic computation of e.

We say that P is a *nondeterministic procedure* if P is a procedure and there is some occurrence of **guess** in the commands defining P. If P is a nondeterministic procedure, and $P(a_1, \ldots, a_n)$ is *a call to* P with arguments $a_1, \ldots, a_n$, the *computation of* $P(a_1, \ldots, a_n)$ consists of a set of execution paths. Each occurrence of a command **guess** $x \in A$ allows to continue a computation on different *execution paths*, one path for each element of A.

We use two commands to indicate the *end of an execution path* in a nondeterministic procedure P. "**return** r" halts the execution and, as one of the results computed by P, returns r. "**fail**" halts the execution without returning any result.

Execution paths ending with **return** are *successful paths*, and those ending with **fail** are *unsuccessful paths*. The *multiset of results* computed by $P(a_1, \ldots, a_n)$, is denoted by $P[a_1, \ldots, a_n]$. Thus, $P[a_1, \ldots, a_n] = \emptyset$ means that, for the given arguments, all execution paths of P are unsuccessful; in this case we say that P *fails*.

```
XUPD1(M, s, φ) % Update M at s w.r.t. φ
INPUT: M ∈ K_Σ, s ∈ S^M, φ ∈ Σ-XCTL
OUTPUT: Modif(M, s, φ)
case φ of
    F      : fail
    T      : M' ← M
    ℓ      : M' ← L^u(M, s, ℓ)
    α ∨ β  : {guess δ ∈ {α, β}; M' ← XUPD1(M, s, δ)}
    α ∧ β  : {M_α ← XUPD1(M, s, α); M' ← XUPD1(M_α, s, β)}
    EX α   : {guess s' ∈ S^M; M' ← XUPD1(T^+(M, s, s'), s', α)}
    AX α   : {guess S' ∈ 2^{S^M} − {∅};
              M' ← XUPD1*(T^u(M, s, S'), S', α)}
    OD_⊙ n : {guess S' ∈ {X ∈ 2^{S^M} − {∅} | |X| ⊙ n};
              M' ← T^u(M, s, S')} % for ⊙ ∈ {⩽, >}
if M', s ⊨ φ then return M' else fail
```

Fig. 1. XUPD$_1$.

3.2. Modification of models

We build our algorithm for model update by using a few basic operations to gradually change the input model.

Definition 3.1 *(Update operations).* Let $\mathcal{M} \in \mathbf{K}_\Sigma$, $s, s' \in S^\mathcal{M}$, $\ell \in \mathrm{Lit}(V^\Sigma)$, $S' \subseteq S^\mathcal{M}$, and $S'' \neq \emptyset$ a finite set, such that $S' \neq \emptyset$ and $S'' \cap S^\mathcal{M} = \emptyset$. The *update operations on Kripke models* are:

1. $\mathbf{L}^u(\mathcal{M}, s, \ell) = \langle S^\mathcal{M}, R^\mathcal{M}, L^\mathcal{M}[s \oplus \ell]\rangle$. Add ℓ to $L^\mathcal{M}(s)$, and remove $\bar{\ell}$.
2. $\mathbf{T}^+(\mathcal{M}, s, s') = \langle S^\mathcal{M}, R^\mathcal{M} \cup \{(s, s')\}, L^\mathcal{M}\rangle$. Add (s, s') to $R^\mathcal{M}$.
3. $\mathbf{T}^u(\mathcal{M}, s, S') = \langle S^\mathcal{M}, (R^\mathcal{M} - (\{s\} \times R^\mathcal{M}[s])) \cup (\{s\} \times S'), L^\mathcal{M}\rangle$. Replace the successors of s, $R^\mathcal{M}[s]$, by S'.
4. $\mathbf{S}^+(\mathcal{M}, S'') = \langle S^\mathcal{M} \cup S'', R^\mathcal{M} \cup I_{S''}, L^\mathcal{M} \cup \overline{V}_{S''}\rangle$. Add S'' to $S^\mathcal{M}$, add $I_{S''}$ to $R^\mathcal{M}$, and label all $t \in S''$ with $\overline{V}$.

Successive applications of the operations $\mathbf{L}^u$, $\mathbf{T}^+$, and $\mathbf{T}^u$ are sufficient to transform a given Σ-model into any other Σ-model. Besides, these operations are instrumental to achieve that a modification of a model satisfies literals, **EX** formulas, and **AX** formulas, respectively. Note that $\mathbf{S}^+(\mathcal{M}, S'')$ is not a model over Σ, i.e., $\mathbf{S}^+(\mathcal{M}, S'') \notin \mathbf{K}_\Sigma$.

Below, we give a precise definition of what we regard as an acceptable modification of a model $\mathcal{M}$ w.r.t. a formula φ.

Definition 3.2 *(Modifications of $\mathcal{M}$ w.r.t. φ).* If $\mathcal{M}$ is a Σ-model, $s \in S^\mathcal{M}$ and $\varphi \in \Sigma$-XCTL, we define, by recursion on φ, *the set of modifications of $\mathcal{M}$ w.r.t. φ at s*, $\mathit{Modif}(\mathcal{M}, s, \varphi)$:

1. $\mathit{Modif}(\mathcal{M}, s, \mathsf{F}) = \emptyset$ and $\mathit{Modif}(\mathcal{M}, s, \mathsf{T}) = \{\mathcal{M}\}$
2. $\mathit{Modif}(\mathcal{M}, s, \ell) = \{\mathbf{L}^u(\mathcal{M}, s, \ell)\}$
3. $\mathit{Modif}(\mathcal{M}, s, \alpha \vee \beta) = \mathit{Modif}(\mathcal{M}, s, \alpha) \cup \mathit{Modif}(\mathcal{M}, s, \beta)$
4. $\mathit{Modif}(\mathcal{M}, s, \alpha \wedge \beta) = \{\mathcal{M}' \in \mathbf{K}_\Sigma \mid \exists \mathcal{M}_\alpha \in \mathit{Modif}(\mathcal{M}, s, \alpha).\ \mathcal{M}' \in \mathit{Modif}(\mathcal{M}_\alpha, s, \beta)\ \&\ \mathcal{M}', s \models \alpha \wedge \beta\}$
5. $\mathit{Modif}(\mathcal{M}, s, \mathbf{EX}\alpha) = \{\mathcal{M}' \in \mathbf{K}_\Sigma \mid \exists s' \in S^\mathcal{M}.\ \mathcal{M}' \in \mathit{Modif}(\mathbf{T}^+(\mathcal{M}, s, s'), s', \alpha)\ \&\ \mathcal{M}', s \models \mathbf{EX}\alpha\}$
6. $\mathit{Modif}(\mathcal{M}, s, \mathbf{AX}\alpha) = \{\mathcal{M}' \in \mathbf{K}_\Sigma \mid \exists S' \in 2^{S^\mathcal{M}} - \{\emptyset\}.\ \mathcal{M}' \in \mathit{Modif}^*(\mathbf{T}^u(\mathcal{M}, s, S'), S', \alpha)\ \&\ \mathcal{M}', s \models \mathbf{AX}\alpha\}$
7. $\mathit{Modif}(\mathcal{M}, s, \mathbf{OD}_\odot n) = \{\mathcal{M}' \in \mathbf{K}_\Sigma \mid \exists S' \in 2^{S^\mathcal{M}} - \{\emptyset\}.\ |S'| \odot n\ \&\ \mathcal{M}' = \mathbf{T}^u(\mathcal{M}, s, S')\},\ \text{for } \odot \in \{\leqslant, >\}$

where $\mathit{Modif}^*(\mathcal{M}, S', \varphi)$ *extends* $\mathit{Modif}(\mathcal{M}, s, \varphi)$ *to a set of states* $S' \subseteq S^\mathcal{M}$:

$$\mathit{Modif}^*(\mathcal{M}, S', \varphi) = \begin{cases} \{\mathcal{M}\} & \text{if } S' = \emptyset \\ \{\mathcal{M}' \in \mathbf{K}_\Sigma \mid \exists t \in S'.\ \exists \mathcal{M}_t \in \mathit{Modif}(\mathcal{M}, t, \varphi).\ \mathcal{M}' \in \mathit{Modif}^*(\mathcal{M}_t, S' - \{t\}, \varphi)\} & \text{if } S' \neq \emptyset \end{cases}$$

Observe that for all $\mathcal{M}' \in \mathit{Modif}(\mathcal{M}, s, \varphi)$, $\mathcal{M}', s \models \varphi$.

3.3. A direct update algorithm for Σ-XCTL

We now define a direct algorithm, XUPD$_1$, that we use as reference to compare our algorithm that updates protected models. For simplicity, we temporarily ignore some concerns that will be considered in the following sections. We now focus on Σ-XCTL, we do not include an operation for adding states, and we are not concerned about efficiency. XUPD$_1$ (see Fig. 1) is an algorithm similar to generate-and-test methods. First, using basic update operations, XUPD$_1$ generates models to satisfy the simplest subformulas of the given formula. Then, XUPD$_1$ modifies these models to satisfy more complex subformulas. Finally, the produced models are tested to verify whether or not they satisfy the whole formula. The case $\varphi = \mathbf{AX}\alpha$ uses XUPD$_1^*$, a procedure implementing $\mathit{Modif}^*(\mathcal{M}, S', \varphi)$.

By using fixed-point characterizations (Section 2) and a mechanism for detecting loops (see Section 3.5), $\text{XUPD}_1(\mathcal{M}, s, \varphi)$ may be extended to an algorithm UPD_1 to compute $\mathit{Modif}(\mathcal{M}, s, \varphi)$ for $\varphi \in \Sigma$-CTL. This extension is analogous to that of Section 3.5.

Note that line (12) of XUPD_1 is necessary to guarantee that the returned model, $\mathcal{M}'$, meets the requirement $\mathcal{M}', s \models \varphi$ in three cases of φ [5].

3.4. An algorithm for updating protected models

Intuitively, a call to $\text{XUPD}_{prot}((\mathcal{M}, P), s, \varphi)$ gradually transforms $\mathcal{M}$ attempting to satisfy the subformulas of φ. Models $\mathcal{M}'$ produced by XUPD_{prot} to satisfy a subformula ψ are accompanied by a *protection* P' containing a part of $\mathcal{M}'$ sufficient to satisfy ψ. In this case, we will say that $\mathcal{M}'$ *is protected by* P'. A key feature of XUPD_{prot} is that if $\mathcal{M}$ is protected by P and $\text{XUPD}_{prot}((\mathcal{M}, P), s, \psi)$ produces $(\mathcal{M}', P')$, then $\mathcal{M}'$ is protected by P' and P' is a protection *greater than or equal to* P (Definition A.1). Hence, XUPD_{prot} preserves satisfaction of previously treated subformulas.

Definition 3.3 *(Protected update operations).* Let $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s, s' \in S^{\mathcal{M}}$, $\ell \in \text{Lit}(V^\Sigma)$, $S' \subseteq S^{\mathcal{M}}$, and $S'' \neq \emptyset$ a finite set. The *update operations on protected models* are:

1. Update ℓ and protect ℓ.
   $\mathbf{L}^u((\mathcal{M}, P), s, \ell) = (\mathbf{L}^u(\mathcal{M}, s, \ell), \langle E^P, A^P, L^P[s \oplus \ell]\rangle)$ if $\bar{\ell} \notin L^P(s)$.
2. Add (s, s') to $\mathcal{M}$ and add (s, s') to E^P.
   $\mathbf{T}^+_\exists((\mathcal{M}, P), s, s') = (\mathbf{T}^+(\mathcal{M}, s, s'), \langle E^P \cup \{(s, s')\}, A^P, L^P\rangle)$ if $s' \in A^P[s]$.
3. Replace $R^{\mathcal{M}}[s]$ and $A^P[s]$ by S'.
   $\mathbf{T}^u_\forall((\mathcal{M}, P), s, S') = (\mathbf{T}^u(\mathcal{M}, s, S'), \langle E^P, A^P - (\{s\} \times (A^P[s] - S')), L^P\rangle)$ if $E^P[s] \subseteq S' \subseteq A^P[s]$ and $S' \neq \emptyset$.
4. Replace $R^{\mathcal{M}}[s]$ and $E^P[s]$ by S'.
   $\mathbf{T}^u_\exists((\mathcal{M}, P), s, S') = (\mathbf{T}^u(\mathcal{M}, s, S'), \langle E^P \cup (\{s\} \times (S' - E^P[s])), A^P, L^P\rangle)$ if $E^P[s] \subseteq S' \subseteq A^P[s]$ and $S' \neq \emptyset$.
5. Add S'' to $\mathcal{M}$ and extend A^P and L^P.
   $\mathbf{S}^+_\forall((\mathcal{M}, P), S'') = (\mathbf{S}^+(\mathcal{M}, S''), \langle E^P, A^P \cup (S^{\mathcal{M}} \times S'') \cup (S'' \times (S^{\mathcal{M}} \cup S'')), L^P \cup \emptyset_{S''}\rangle)$ if $S'' \cap S^{\mathcal{M}} = \emptyset$.

Since the above are partial functions, we assume that their application in cases not covered by the corresponding definitions produces an undefined value ($\perp$). Note that all above operations, except $\mathbf{S}^+_\forall$, preserve the signature Σ.

We extend to protected models our definition of modification of a model.

Definition 3.4 *(Modifications of $(\mathcal{M}, P)$ w.r.t. φ).* If $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$ is a protected model, $s \in S^{\mathcal{M}}$ and $\varphi \in \Sigma$-XCTL, we define, by recursion on φ, *the set of modifications of* $(\mathcal{M}, P)$ *w.r.t.* φ *at* s, $\mathit{Modif}((\mathcal{M}, P), s, \varphi)$:

1. $\mathit{Modif}((\mathcal{M}, P), s, \mathsf{F}) = \emptyset$ and $\mathit{Modif}((\mathcal{M}, P), s, \mathsf{T}) = \{(\mathcal{M}, P)\}$
2. $\mathit{Modif}((\mathcal{M}, P), s, \ell) = \{\mathbf{L}^u((\mathcal{M}, P), s, \ell)\}$
3. $\mathit{Modif}((\mathcal{M}, P), s, \alpha \vee \beta) = \mathit{Modif}((\mathcal{M}, P), s, \alpha) \cup \mathit{Modif}((\mathcal{M}, P), s, \beta)$
4. $\mathit{Modif}((\mathcal{M}, P), s, \alpha \wedge \beta) = \{(\mathcal{M}', P') \in \mathbf{KP}_\Sigma \mid \exists (\mathcal{M}_\alpha, P_\alpha) \in \mathit{Modif}((\mathcal{M}, P), s, \alpha).\ (\mathcal{M}', P') \in \mathit{Modif}((\mathcal{M}_\alpha, P_\alpha), s, \beta)\}$
5. $\mathit{Modif}((\mathcal{M}, P), s, \mathbf{EX}\alpha) = \{(\mathcal{M}', P') \in \mathbf{KP}_\Sigma \mid \exists s' \in A^P[s].\ (\mathcal{M}', P') \in \mathit{Modif}(\mathbf{T}^+_\exists((\mathcal{M}, P), s, s'), s', \alpha)\}$
6. $\mathit{Modif}((\mathcal{M}, P), s, \mathbf{AX}\alpha) = \{(\mathcal{M}', P') \in \mathbf{KP}_\Sigma \mid \exists S' \subseteq A^P[s].\ E^P[s] \subseteq S' \neq \emptyset\ \ \&\ (\mathcal{M}', P') \in \mathit{Modif}^*(\mathbf{T}^u_\forall((\mathcal{M}, P), s, S'), S', \alpha)\}$
7. $\mathit{Modif}((\mathcal{M}, P), s, \mathbf{OD}_\odot n) = \{(\mathcal{M}', P') \in \mathbf{KP}_\Sigma \mid \exists S' \subseteq A^P[s].\ E^P[s] \subseteq S' \neq \emptyset\ \&\ |S'| \odot n\ \&\ (\mathcal{M}', P') = \mathbf{T}^u_Q((\mathcal{M}, P), s, S')\}$
   for $\odot \in \{\leqslant, >\}$; $Q = \forall$ if $\odot \in \{\leqslant\}$, and $Q = \exists$ if $\odot \in \{>\}$

where $\mathit{Modif}^*((\mathcal{M}, P), S', \varphi)$ *extends* $\mathit{Modif}((\mathcal{M}, P), s, \varphi)$ *to a set of states* $S' \subseteq S^{\mathcal{M}}$ and is defined analogously to $\mathit{Modif}^*(\mathcal{M}, S', \varphi)$.

In Fig. 2, we define an algorithm for updating protected models. Intuitively, $\text{XUPD}_{prot}((\mathcal{M}, P), s, \varphi)$ finds models for φ at state s by modifying $\mathcal{M}$ and respecting the protection P. The initial call to XUPD_{prot} may use the empty protection $P_\perp$. The case $\varphi = \mathbf{AX}\alpha$ uses XUPD^*_{prot}, a procedure implementing $\mathit{Modif}^*((\mathcal{M}, P), S', \varphi)$.

Observe that XUPD_{prot} does not need a verification similar to the verification in line (12) of XUPD_1. Protections guarantee that the returned model, $(\mathcal{M}', P')$, meets the requirement $(\mathcal{M}', P'), s \models \varphi$. First, such a requirement is fulfilled when XUPD_{prot} is applied to basic formulas (T, ℓ, $\mathbf{OD}$). Then, subsequent applications of XUPD_{prot} use the current protection to preserve the satisfaction of previously treated subformulas. For a comparison between XUPD_{prot} and XUPD_1, see Section 5.

3.5. Model update for Σ-CTL and addition of states

We extend XUPD_{prot} to formulas that use the operators **EU**, **AU**, **ER**, and **AR**. These operators are replaced by their fixed-point characterizations (Section 2) and then treated by a mechanism for detecting loops. This loop-detecting mechanism is

$\text{XUPD}_{prot}((\mathcal{M}, P), s, \varphi)$ **%** Find $(\mathcal{M}', P')$ w.r.t. φ at s.
INPUT: $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s \in S^{\mathcal{M}}$, and $\varphi \in \Sigma$-XCTL
OUTPUT: $Modif((\mathcal{M}, P), s, \varphi)$

if $(\mathcal{M}, P), s \models \varphi$ **then** $(\mathcal{M}', P') \leftarrow (\mathcal{M}, P)$
else case φ **of**
  F : **fail**
  T : $(\mathcal{M}', P') \leftarrow (\mathcal{M}, P)$
  ℓ : **if** $\bar{\ell} \in L^P(s)$ **then fail**
    else $(\mathcal{M}', P') := \mathbf{L}^u((\mathcal{M}, P), s, \ell)$
  $\alpha \vee \beta$: {**guess** $\delta \in \{\alpha, \beta\}$;
    $(\mathcal{M}', P') \leftarrow \text{XUPD}_{prot}((\mathcal{M}, P), s, \delta)$}
  $\alpha \wedge \beta$: {$(\mathcal{M}_\alpha, P_\alpha) \leftarrow \text{XUPD}_{prot}((\mathcal{M}, P), s, \alpha)$;
    $(\mathcal{M}', P') \leftarrow \text{XUPD}_{prot}((\mathcal{M}_\alpha, P_\alpha), s, \beta)$}
  $\mathbf{EX}\alpha$: {**guess** $s' \in A^P[s]$;
    $(\mathcal{M}', P') \leftarrow \text{XUPD}_{prot}(\mathbf{T}^+_\exists((\mathcal{M}, P), s, s'), s', \alpha)$}
  $\mathbf{AX}\alpha$: {**guess** $S' \in \{X \subseteq A^P[s] \mid E^P[s] \subseteq X \ \ \& \ X \neq \emptyset\}$;
    $(\mathcal{M}', P') \leftarrow \text{XUPD}^*_{prot}(\mathbf{T}^u_\forall((\mathcal{M}, P), s, S'), S', \alpha)$}
  $\mathbf{OD}_\odot n$: {**guess** $S' \in \{X \subseteq A^P[s] \mid E^P[s] \subseteq X \ \ \& \ X \neq \emptyset \ \& \ |X| \odot n\}$; **%** $\odot \in \{\leqslant, >\}$
    if $\odot \in \{\leqslant\}$ **then** $Q \leftarrow \forall$ **else** $Q \leftarrow \exists$;
    $(\mathcal{M}', P') \leftarrow \mathbf{T}^u_Q((\mathcal{M}, P), s, S')$}
return $(\mathcal{M}', P')$

Fig. 2. XUPD_{prot}.

$\text{UPD}_{prot}((\mathcal{M}, P, Q), s, \varphi)$ **%** Find $(\mathcal{M}', P', Q')$ w.r.t. φ at s.
INPUT: $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $Q \subseteq S^{\mathcal{M}} \times \Sigma$-CTL, $s \in S^{\mathcal{M}}$, and $\varphi \in \Sigma$-CTL.
OUTPUT: $Modif((\mathcal{M}, P, Q), s, \varphi)$

if $(\mathcal{M}, P), s \models \varphi$ **then** $(\mathcal{M}', P', Q') \leftarrow (\mathcal{M}, P, Q)$
else case φ **of**
  .
  .
  .
  E$[\alpha \mathbf{U} \beta]$: **if** $(s, \varphi) \in Q$ **then fail %** default for lfp
    else $(\mathcal{M}', P', Q') \leftarrow \text{UPD}_{prot}((\mathcal{M}, P, Q \cup \{(s, \varphi)\}), s, \beta \vee (\alpha \wedge \mathbf{EX}\varphi))$
  A$[\alpha \mathbf{R} \beta]$: **if** $(s, \varphi) \in Q$ **then** $(\mathcal{M}', P', Q') \leftarrow (\mathcal{M}, P, Q)$ **%** default for gfp
    else $(\mathcal{M}', P', Q') \leftarrow \text{UPD}_{prot}((\mathcal{M}, P, Q \cup \{(s, \varphi)\}), s, \beta \wedge (\alpha \vee \mathbf{AX}\varphi))$
return $(\mathcal{M}', P', Q')$

Fig. 3. UPD_{prot}.

$\text{XUPD}_{S+}((\mathcal{M}, P), s, \varphi)$ **%** Add states until $\text{XUPD}_{prot}[(\mathcal{M}, P), s, \varphi] \neq \emptyset$.
INPUT: $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s \in S^{\mathcal{M}} = \{s_1, \ldots, s_m\}$, $\varphi \in \Sigma$-XCTL
OUTPUT: $Modif(\mathbf{S}^+_\forall((\mathcal{M}, P), S^{\Sigma'} - S^{\Sigma}), s, \varphi)$
Where, assuming $n = |\varphi|$, Σ' is the minimal signature such that: $\Sigma' \supseteq \Sigma$
and $(Modif(\mathbf{S}^+_\forall((\mathcal{M}, P), S^{\Sigma'} - S^{\Sigma}), s, \varphi) \neq \emptyset$ or $|S^{\Sigma'}| > n8^n)$.

$n \leftarrow |\varphi|$; $m' \leftarrow m$; **%** Remember that $S^{\mathcal{M}} = \{s_1, \ldots, s_m\}$
while $\text{XUPD}_{prot}[(\mathcal{M}, P), s, \varphi] = \emptyset$ **and** $m' \leqslant n8^n$ **do**
  {$m' \leftarrow |S^M| + 1$;
  $s' \leftarrow s_{m'}$; **%** s' is a new state, $s' \notin S^M$
  $(\mathcal{M}, P) \leftarrow \mathbf{S}^+_\forall((\mathcal{M}, P), \{s'\})$}
$(\mathcal{M}', P') \leftarrow \text{XUPD}_{prot}((\mathcal{M}, P), s, \varphi)$; **%** "$\leftarrow$" fails if XUPD_{prot} fails
return $(\mathcal{M}', P')$

Fig. 4. XUPD_{S+}.

controlled by a parameter $Q \subseteq S^{\mathcal{M}} \times \Sigma$-CTL. The intuition behind Q is that if $(s, \psi) \in Q$, then state s has already been visited while updating w.r.t. ψ. UPD_{prot}, depicted in Fig. 3, illustrates how to modify the pseudo-code in the cases of **EU** and **AR**; the modifications for **AU** and **ER** are similar. The initial call to UPD_{prot} uses an empty set of visited states.

A state addition is only necessary when a model update fails to find a model with a given signature. For example, if $\Sigma = \langle\{s_0\}, \{p\}\rangle$, $(\mathcal{M}, P)$ is any protected Σ-model, and $\varphi = \mathbf{EX}\, p \wedge \mathbf{EX} \neg p$, then $\text{XUPD}_{prot}[(\mathcal{M}, P), s_0, \varphi] = \emptyset$. In this case, and assuming that $n = |\varphi|$, we can extend our model update procedure so as to be able to add states to $\mathcal{M}$ until $\text{XUPD}_{prot}[(\mathcal{M}, P), s, \varphi] \neq \emptyset$ or $|S^{\mathcal{M}}| > n8^n$. This bound on the number of states is justified by a *small model theorem for CTL* [16, p. 9]: if φ is satisfiable then φ is satisfiable in a model of size less than or equal to $n8^n$, where the *size* of a model $\mathcal{M}$ is $|S^{\mathcal{M}}|$ [16, p. 4]. In Fig. 4, XUPD_{S+} shows how to add states in the update process.

4. Heuristics and search strategies

A significant feature of UPD_{prot} (and UPD_1) is nondeterminism. Direct deterministic implementations of nondeterministic algorithms, however, may suffer from inefficiency. To improve the efficiency of UPD_{prot}, we propose heuristics based on an order on the generation of nondeterministic choices from: (1) a set of states, (2) a set of sets of states, and (3) a set of two CTL formulas. To find minimal solutions we propose a search strategy similar to iterative deepening.

For practical purposes, we make some assumptions. We assume that a finite set A is *represented by a list*, $[A]$, containing the elements of A in an arbitrary order, e.g., $[A] = [a_1, a_2, \ldots, a_n]$. We also assume that the command **guess** $x \in [A]$ produces nondeterministic choices in the order of $[A]$, i.e., the first choice is a_1, then a_2 and so on. We use $+$ and Σ to represent the concatenation of two lists and the concatenation of a finite sequence of lists, respectively. Under these assumptions, we show how to improve the efficiency of $\text{XUPD}_1(\mathcal{M}, s, \varphi)$ in cases where $\varphi = \mathbf{EX}\alpha$, $\varphi = \mathbf{AX}\alpha$, or $\varphi = \alpha \vee \beta$ (improvements for XUPD_{prot} are analogous).

4.1. Ordering nondeterministic choices of states

To improve the efficiency of $\text{XUPD}_1(\mathcal{M}, s, \mathbf{EX}\alpha)$ we use an order on the generation of nondeterministic choices of states. In line (7) of XUPD_1, we replace the command "**guess** $s' \in S^{\mathcal{M}}$" by "**guess** $s' \in [R^{\mathcal{M}}[s]] + [S^{\mathcal{M}} - R^{\mathcal{M}}[s]]$". Thus, the first nondeterministic choices will be elements of $R^M[s]$.

4.2. Ordering nondeterministic choices of sets of states

To improve the efficiency of $\text{XUPD}_1(\mathcal{M}, s, \mathbf{AX}\alpha)$ we use an order on the generation of nondeterministic choices of sets of states. Let $m = |S^{\mathcal{M}}|$, and for $i = 0, \ldots, m$, let $Q_i(\mathcal{M}, s) = \{S' \subseteq S^{\mathcal{M}} \mid S' \neq \emptyset \ \ \& \ \ |\Delta(S', R^{\mathcal{M}}[s])| = i\}$. Therefore, if $S' \in Q_i(\mathcal{M}, s)$ then $d(T^u(\mathcal{M}, s, S'), \mathcal{M}) = i$.

It can be shown that the elements of $\sum_{i=0}^{m}[Q_i(\mathcal{M}, s)]$ are lists representing the nonempty subsets of $S^{\mathcal{M}}$. Besides, if S_1 and S_2 are two elements of the list $\sum_{i=0}^{m}[Q_i(\mathcal{M}, s)]$ and S_1 occurs before S_2 in such a list, then $d(T^u(\mathcal{M}, s, S_1), \mathcal{M}) \leqslant d(T^u(\mathcal{M}, s, S_2), \mathcal{M})$.

In line (8) of XUPD_1, we replace the command "**guess** $S' \in 2^{S^{\mathcal{M}}} - \{\emptyset\}$" by "**guess** $S' \in \sum_{i=0}^{m}[Q_i(\mathcal{M}, s)]$".

4.3. Ordering nondeterministic choices of formulas

To improve the efficiency of $\text{XUPD}_1(\mathcal{M}, s, \alpha \vee \beta)$ we use an order on the generation of nondeterministic choices of CTL formulas. In line (5) of XUPD_1, we replace the command "**guess** $\delta \in \{\alpha, \beta\}$" by "**guess** $\delta \in [\delta_1, \delta_2]$", where $\delta_1 = \alpha$, $\delta_2 = \beta$ if $|\alpha| \leqslant |\beta|$, and $\delta_1 = \beta$, $\delta_2 = \alpha$ otherwise.

4.4. Limiting the number of changes

Besides improving efficiency, the above heuristics help UPD_{prot} to produce first models close to the input model. The above heuristics, however, do not ensure that UPD_{prot} produces first minimal solutions. Therefore, for cases requiring a minimal solution, we propose a variant of UPD_{prot}, UPD_{min}, that uses a search strategy similar to iterative deepening.

We get an implementation of UPD_{min} by modifying UPD_{prot} as follows. First, we add to UPD_{prot} a parameter, n, to record the number of *changes applied* to the input model. We also include a variable, w, to fix the maximum number of *changes allowed*. Second, we modify UPD_{prot} to compute the number of *changes required*, r, before applying an update operation. Third, we modify UPD_{prot} to halt if $n + r > w$, i.e., UPD_{prot} halts if the number of changes already made, plus the number of changes to apply, is greater than the number of changes allowed. Finally, to produce first the minimal solutions, we initialize w to zero and we iterate UPD_{prot}, increasing w by 1, until UPD_{prot} produces a solution.

5. Behavior of UPD_{prot}

We illustrate the behavior of UPD_{prot} with several examples, including a classical mutual exclusion problem and models with hundreds of states. We show other examples at the web site of UPD_{prot}, http://turing.iimas.unam.mx/ctl_upd3/form1.prl.

5.1. Synthesizing a model of the mutual exclusion problem

Emerson and Clarke [15] present a method for automatically synthesizing *synchronization skeletons*, from a CTL specification, through a decision procedure that constructs a model of the specification. We use Emerson and Clarke's mutual-exclusion example to show that our CTL update method can also be used as a decision procedure for satisfiability of CTL formulas.

The specification of the mutual exclusion problem by Emerson and Clarke uses a variant of CTL *with processes* [15]. We adapt such a specification to a specification Γ in Σ-CTL. The specification Γ is the result of factoring **AG** in the conjunction of the following formulas, where $i, j \in \{1, 2\}$ and $i \neq j$:

Table 1
UPD$_1$ vs. UPD$_{prot}$ in the repair of faulty models of mutual exclusion.

N	Removed parts (✗)							UPD$_1$ time [s]		UPD$_{prot}$ time [s]	
	s_0s_1	s_1s_3	s_3s_7	s_2s_5	n_1s_0	n_2s_1	D	-cN	-c?	-cN	-c?
1	✗							1.9	–	0.6	1.7
3	✗	✗	✗					67.0	–	0.8	1.8
5	✗	✗	✗	✗	✗			1274.8	–	0.9	8.9
6	✗	✗	✗	✗	✗	✗		–	–	1.2	9.1
55	✗	✗	✗	✗	✗	✗	✗	–	–	14.9	16.4

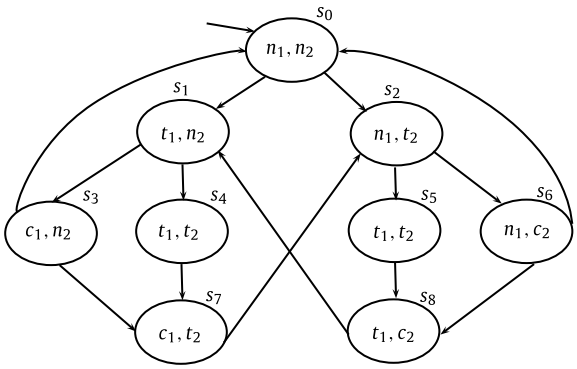

Fig. 5. Model produced by UPD$_{prot}$ (variables $turn_i$ are omitted).

1. Start state. Both processes are in their noncritical region: $(n_1 \wedge n_2)$
2. Each process i is always exactly in one of the three code regions:
   $\mathbf{AG}((n_i \vee t_i \vee c_i) \wedge (n_i \rightarrow \neg(t_i \vee c_i)) \wedge (t_i \rightarrow \neg(n_i \vee c_i)) \wedge (c_i \rightarrow \neg(n_i \vee t_i)))$
3. Any move that process i makes from its noncritical (critical) region is into its trying (noncritical) region, and such a move is always possible:
   $\mathbf{AG}((n_i \rightarrow ((\mathbf{AX}(t_i \vee n_i)) \wedge (\mathbf{EX}\, t_i))) \wedge (c_i \rightarrow ((\mathbf{AX}(n_i \vee c_i)) \wedge (\mathbf{EX}\, n_i))))$
4. Any move that process i makes from its trying region is into its critical region and such a move is possible when it is the turn of process i:
   $\mathbf{AG}((t_i \rightarrow (\mathbf{AX}(c_i \vee n_i))) \wedge ((t_i \wedge turn_i) \rightarrow (\mathbf{EX}\, c_i)))$
5. A transition by one process cannot cause the other process to move. If process i is in region $r_i \in \{n_i, t_i, c_i\}$ and process j moves, then i remains in r_i:
   $\mathbf{AG}(((r_i \wedge n_j) \rightarrow (\mathbf{AX}(t_j \rightarrow r_i))) \wedge ((r_i \wedge t_j) \rightarrow (\mathbf{AX}(c_j \rightarrow r_i))) \wedge ((r_i \wedge c_j) \rightarrow (\mathbf{AX}(n_j \rightarrow r_i))))$
6. Some process can always move. If some process is in its noncritical region then both processes can move; otherwise only one process can move:
   $\mathbf{AG}(((n_1 \vee n_2) \rightarrow (turn_1 \wedge turn_2)) \wedge ((\neg n_1 \wedge \neg n_2) \rightarrow (turn_1 \veebar turn_2)))$
7. Each transition is due to the movement of exactly one process:
   $\mathbf{AG}(((turn_1 \wedge turn_2) \rightarrow (\mathbf{OD}_{\leqslant} 2)) \wedge ((\neg turn_1 \vee \neg turn_2) \rightarrow (\mathbf{OD}_{\leqslant} 1)))$
8. Split state $s = (t_1, t_2, turn_1, turn_2)$ into states $(t_1, t_2, turn_1, \neg turn_2)$ and $(t_1, t_2, \neg turn_1, turn_2)$ and separate the transitions going towards s. This requirement reflects a preference of Emerson and Clarke to distinguish all states by their propositional labels [15, p. 258]:
   $\mathbf{AG}((t_i \wedge n_j) \rightarrow (\mathbf{EX}(t_j \wedge turn_i)))$

Note that, in the specification Γ, we only use formulas to specify the *local structure* of the system, through the operators **AX** and **EX** and an outermost operator **AG**. We do not include formulas to specify *global behavior* of the system by using operators **F**, **G**, **U**, or **R**. Global behavior formulas, for example $\mathbf{AG}(t_i \rightarrow \mathbf{EF}\, c_i)$ and $\mathbf{AG}\neg\mathbf{EF}(c_1 \wedge c_2)$, are generally expected to be a consequence of formulas specifying local structure. Intuitively, it is relatively easier to synthesize a model from formulas specifying local structure than to synthesize a model from formulas specifying global behavior. Global behavior formulas can be used instead to update a faulty model, presumably close to a correct model. A difference between model synthesis and model update is related to the difference between these two kinds of formula.

In Table 1, we compare UPD$_1$ and UPD$_{prot}$ in the repair of faulty models of mutual exclusion w.r.t. Γ. Column N indicates the number of changes needed to repair the model. Input models were obtained by removing, according to the marks ✗, transitions (s_i, s_j) and labels n_i in s_j, from the model in Fig. 5. A mark ✗ in column D means that the input model is the dummy model in Fig. 6. Columns -cN restrict the search to a maximum of N changes. Columns -c? record searches with an unlimited number of changes. Entries "–" mean "no answer after one hour". The last four columns show the time, in seconds, necessary to produce the first solution on a PC with a dual-core processor at 2.0 GHz and 2 GB of RAM.

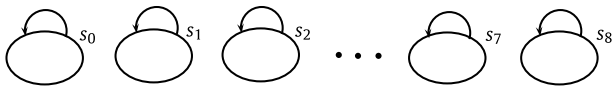

Fig. 6. A dummy model of nine states.

Table 1 shows tests where we gradually removed some parts of the model in Fig. 5. Observe that the more parts were removed, the more time UPD$_1$ required to produce a solution. The option -cN helped both UPD$_1$ and UPD$_{prot}$ to limit the search for solutions to models that result from applying at most N changes to the input model. Note that even with the aid of the switch -cN, UPD$_1$ was incapable of computing a first solution after one hour.

By using the specification Γ, the first solution produced by UPD$_{prot}$ from the dummy model in Fig. 6 is the model in Fig. 5, having the expected structure [15, Fig. 11, p. 259]. This happened after 16.4 s (Table 1).

5.2. Updating a microwave oven model

In Section 5.1, we showed that the application of UPD$_{prot}$ to a dummy model can build a model that satisfies a given formula. We now assume that the non-dummy model in Fig. 7, $\mathcal{M}_{oven}$, is given. Clarke et al. [11] use $\mathcal{M}_{oven}$ to illustrate the model-checking technique and, following Zhang and Ding [22], we use $\mathcal{M}_{oven}$ to show the application of UPD$_{prot}$ in the update of (modifications of) $\mathcal{M}_{oven}$ w.r.t. some given formulas.

The intuition behind $\mathcal{M}_{oven}$ is an informal specification of the oven operation:

1. Open the door and put the food inside ($\rightarrow s_0$).
2. Close the door ($s_0 \rightarrow s_2$).
3. Press START ($s_2 \rightarrow s_5$) to begin the warming of the oven ($s_5 \rightarrow s_6$).
4. After warming, cooking starts automatically ($s_6 \rightarrow s_3$).
5. When cooking is finished, the heat is turned off ($s_3 \rightarrow s_2$).
6. While cooking, the door may be opened ($s_3 \rightarrow s_0$).
7. Pressing START while the door is open produces an error ($s_0 \rightarrow s_1$).
8. On error, close the door ($s_1 \rightarrow s_4$) and press RESET ($s_4 \rightarrow s_2$).

We formalize the above informal specification of the microwave oven by a CTL formula Ψ. The formula Ψ is the result of factoring **AG** in the conjunction of the following formulas:

1. Initial state:
   $\Psi_1 = (\neg start \wedge \neg close \wedge \neg heat \wedge \neg error)$
2. At any time, if the door is open, it may be closed:
   $\Psi_2 = \mathbf{AG}(\neg close \rightarrow \mathbf{EX}\, close)$
3. If the heat is off, START may be pressed:
   $\Psi_3 = \mathbf{AG}((\neg start \wedge \neg heat) \rightarrow \mathbf{EX}\, start)$
4. If START is pressed and there is no error, warming may start:
   $\Psi_4 = \mathbf{AG}((start \wedge close \wedge \neg heat \wedge \neg error) \rightarrow \mathbf{EX}(start \wedge heat))$
5. After warming, cooking may begin:
   $\Psi_5 = \mathbf{AG}((start \wedge heat) \rightarrow \mathbf{EX}(\neg start \wedge heat))$
6. While cooking, the heat may be turned off (by elapsed time):
   $\Psi_6 = \mathbf{AG}((\neg start \wedge heat) \rightarrow \mathbf{EX}(close \wedge \neg heat))$
7. While cooking, the door may be opened:
   $\Psi_7 = \mathbf{AG}((\neg start \wedge heat) \rightarrow \mathbf{EX} \neg close)$
8. If START is pressed and the door is open, an immediate error occurs:
   $\Psi_8 = \mathbf{AG}((start \wedge \neg close) \rightarrow error)$
9. On error, if the door is open, the error persists:
   $\Psi_9 = \mathbf{AG}((error \wedge \neg close) \rightarrow \mathbf{AX}\, error)$
10. On error, if the door is closed, the oven may be reset:
    $\Psi_{10} = \mathbf{AG}((error \wedge close) \rightarrow \mathbf{EX}(\neg start \wedge close))$

Unlike the example in Section 5.1, $\mathcal{M}_{oven}$ (Fig. 7) is not characterized by the specification Ψ, but it can be proved that $\mathcal{M}_{oven}$ satisfies Ψ at s_0. Besides, $\mathcal{M}_{oven}$ does not satisfy at s_0 a formula expressing that if START is pressed, then the heat will eventually be turned on, $\rho_1 = \mathbf{AG}(start \rightarrow \mathbf{AF}\, heat)$.

In Table 2, we show results of tests comparing UPD$_1$ and UPD$_{prot}$ in the repair of modifications of $\mathcal{M}_{oven}$. Input models for these tests were obtained by modifying $\mathcal{M}_{oven}$ according to the changes indicated in column "Modifications". We use t, c, h, and e as shorthand for variables *start*, *close*, *heat* and *error*, respectively. Given $x \in \{t, c, h, e\}$, a modification ix complements the label x in the state s_i. A modification ij adds the transition (s_i, s_j) to $\mathcal{M}_{oven}$ if $(s_i, s_j) \notin R^{\mathcal{M}_{oven}}$, otherwise ij removes

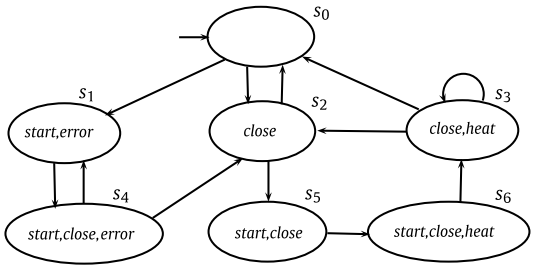

Fig. 7. $\mathcal{M}_{oven}$, a Kripke model for a microwave oven.

Table 2
UPD$_1$ vs. UPD$_{prot}$ in the repair of modifications of $\mathcal{M}_{oven}$.

Input		UPD$_1$ time [s]			UPD$_{min}$ time [s]		UPD$_{prot}$ time [s]		
Modifications	φ	-cm	-c?	n	-c?	m	-cm	-c?	n
None	ρ_1	0.1	0.1	4	0.1	1	0.1	0.1	4
0h, 2c	φ_2	0.1	—	—	0.1	3	0.1	0.1	7
0h, 2c, 5t	φ_3	2.0	—	—	0.4	4	0.1	0.1	10
0h, 6h, 1h, 02, 3h	φ_7	36.5	—	—	2.2	5	0.3	0.1	11
6t,0t,2c,4t,4e,3t,6e	φ_{10}	72.5	—	—	2.0	5	0.1	0.1	15

(s_i, s_j) from $\mathcal{M}_{oven}$. Column φ shows the formula used for updating the input model. For $j \geqslant 0$, $\varphi_j = \rho_1 \wedge (\bigwedge_{i=1}^{j} \Psi_i)$. We designed modifications so that the modification of $\mathcal{M}_{oven}$ in row φ_j does not satisfy $\bigwedge_{i=1}^{j} \Psi_i$. Columns -c$m$ and -c? show time (in seconds) necessary to produce a first solution. Columns -cm show solutions with a search restricted to a maximum of m changes. Columns -c? record solutions of searches with an unlimited number of changes, i.e., without using m. Columns n indicate the number of changes needed to generate a first solution with the option -c?. Column m, computed by UPD$_{min}$, is the minimum number of changes needed to repair the input model. Entries "—" mean "no answer after 600 s". The initial protection in the execution of UPD$_{min}$ and UPD$_{prot}$ was $P_\perp$. These tests were performed on a PC with a quad-core processor at 2.0 GHz and 8 GB of RAM.

We gradually removed some parts of $\mathcal{M}_{oven}$ and gradually augmented the complexity of the input formula. Observe that the more parts were removed, UPD$_1$ required more (or the same) time to produce a solution. Intuitively, the option -cm helps both UPD$_1$ and UPD$_{prot}$ because it restricts the search space to the set of models that result from applying at most m changes.

Remark that in the update of $\mathcal{M}_{oven}$ w.r.t. ρ_1, both UPD$_1$ and UPD$_{prot}$ produced a first solution in less than one second, even without the aid provided by a limit on the number of changes (Table 2). Thus, UPD$_1$ and UPD$_{prot}$ had a similar performance when updating $\mathcal{M}_{oven}$ w.r.t. ρ_1. Unlike UPD$_{prot}$, however, an update through UPD$_1$ may cause $\mathcal{M}_{oven}$ to lose some of its properties, for example satisfying (parts of) the specification Ψ. Indeed, the first solution of UPD$_1(\mathcal{M}_{oven}, s_0, \rho_1)$ does not satisfy Ψ. Even worse, many of the first solutions of UPD$_1(\mathcal{M}_{oven}, s_0, \rho_1)$ do not satisfy Ψ, as can be inferred from the running time of UPD$_1(\mathcal{M}_{oven}, s_0, \varphi_{10})$ (Table 2).

In practice, a formula used to update a given model is usually more complex than ρ_1. A formula for a model update may include subformulas expressing properties that motivated the design of the model, i.e., a model specification (e.g., Ψ).

Note that ρ_1 expresses an undesirable property. If $\mathcal{M}_{oven}$ satisfied ρ_1, then the heat could be turned on with the door open and thus harm the user. Following [22], we considered an update of $\mathcal{M}_{oven}$ to satisfy ρ_1 at s_0.

Column -c? of UPD$_1$ of Table 2 shows that, without the help of m, UPD$_1$ can produce an answer in less than 600 s only in the case of formula ρ_1. In contrast, all the corresponding answers of UPD$_{prot}$ are produced in less than one second.

5.3. Updating the model of a counter

The next example exhibits the behavior of UPD$_{prot}$ in a readily scalable problem: a synchronous counter. Nondeterminism is reflected as an external input making the counter either count or halt. Each state has exactly two successors, intended to represent a system with two successors per state on average. Instead of making the counter halt immediately, we ask it to eventually halt, making our example more representative of typical model-checking uses, where a property is required to eventually hold.

A specification of an n-bit synchronous counter is the conjunction of the formulas below, where $i \in \{0, \ldots, n-1\}$:

1. Each state has exactly two successors: $\mathbf{OD}_{\leqslant} 2 \wedge \mathbf{OD}_{>} 1$.
2. If the input is "count" ($\neg h$) and either the bit i is on or the i least-significant bits are on, then the bit i will be on. Otherwise the bit i will be off (two formulas for each i):

Table 3
UPD$_{prot}$ applied to faulty models of synchronous counters.

states	trans. removed	loops added	time	states	trans. removed	loops added	time
64	64	0	1	256	256	0	55
	96	32	7		384	128	1355
	128	**64**	**13**		**512**	**256**	**2662**
128	128	0	8	512	512	0	140
	192	64	77		576	64	950
	256	**128**	**155**		640	128	3776

Table 4
UPD$_{prot}$ applied to a random model M and a random formula φ.

M: 10 var, 1024 sts, 2048 trn			φ: 10var, 5$\vee$, 5$\wedge$, 2X, 2U, 2R, 2F, 2G			
formula	chg	time	model states-trans.		chg	time
10$\vee$, 10$\wedge$, 20X	4	7	128	256	2	1
+2U	2	5	256	512	22	4
+2U, 2R	2	23	512	1024	49	36
+2U, 2R, 2F	3	411	1024	2048	48	157
+2U, 2R, 2F, 2G	–	–	2048	3072	–	–

$(\neg h \wedge (x_i \veebar (x_0 \wedge \cdots \wedge x_{i-1}))) \rightarrow \mathbf{AX}\, x_i$
$\neg(\neg h \wedge (x_i \veebar (x_0 \wedge \cdots \wedge x_{i-1}))) \rightarrow \mathbf{AX}\, \neg x_i$
By convention $(x_0 \veebar (x_0 \wedge \cdots \wedge x_{0-1})) = \neg x_0$.

3. If a state having all its bits off is reached when not counting (h), then the counter remains there:
   $(h \wedge \neg x_0 \wedge \cdots \wedge \neg x_{n-1}) \rightarrow \mathbf{AX}(\neg x_0 \wedge \cdots \wedge \neg x_{n-1})$
4. When not counting, the counter eventually reaches a state having all its bits off:
   $h \rightarrow \mathbf{A}[h\, \mathbf{U}\, \neg x_0 \wedge \cdots \wedge \neg x_{n-1}]$

In this example, we will assume that we wish the state labels to remain fixed, and will update the transitions only. This can be readily achieved in UPD$_{prot}$ with an initial protection protecting all the labels. In UPD$_1$, we can simply inhibit the fragment code that modifies labels. Note also that the specification must hold at all states.

Input models were obtained by intentionally modifying a model satisfying the specification with either the removal of correct transitions of the addition of wrong transitions. The wrong transitions added are self-loops, so that if all correct transitions are removed and all states have loops, we obtain the dummy model as input model, representing the worst of cases.

Table 3 shows the time, in seconds, taken by UPD$_{prot}$ to obtain a synchronous counter from the specification for different faulty models. (UPD$_1$ took more than one hour even for the simplest (i.e., the first) instance on the table.) The columns labeled "states" have the number of states resulting from considering the bits of the counter itself in addition to the control bit h. The columns labeled "trans. removed" indicate the number of transitions removed from the correct model. The columns labeled "loops added" indicate the number of loops added. The lines in boldface refer to dummy models.

The tests were done on a PC with a dual-core processor at 2.0 GHz and 2 GB of RAM.

Although current model checking practice often employs models with a vast number of states, the fact that UPD$_{prot}$ can handle some examples with hundreds of states shows that our method could at least be useful for abstractions of systems.

5.4. UPD$_{prot}$ applied to random models and formulas

Here we show the behavior of UPD$_{prot}$ when applied to randomly generated models and formulas. Table 4 (left) shows the application of UPD$_{prot}$ to a random model, M, generated with 10 variables, 1024 states and 2048 transitions. We applied UPD$_{prot}$ to update M w.r.t. random formulas that were generated with the operators listed in the first column. All formulas in this column were generated with a set of operators containing 10 $\vee$, 10 $\wedge$, 10 **EX**, and 10 **AX**. The formula in the second row was generated by using two additional **U** operators, one **EU** and one **AU**. The formulas in the other rows were similarly generated. Columns "chg" and "time" of Table 4 (left and right) respectively show the number of changes made by UPD$_{prot}$ and the running time in seconds. Entries "–" mean "no answer after 600 seconds". We discarded running times greater than 600 seconds because the formula (left) or the model (right) along the respective model or formula, could be a case of *unsatisfiable formula* or a case of *hard complexity*, respectively.

Table 4 (right) shows the application of UPD$_{prot}$ to a random formula, φ, generated with 10 variables, five operators $\vee$, five $\wedge$, and one of all other operators (**EX**, **AX**, **EU**, **AU**, **ER**, **AR**, **EF**, **AF**, **EG**, **AG**). We applied UPD$_{prot}$ to update, w.r.t. φ, models that were generated using 10 variables and the number of states and transitions listed respectively in the first two columns.

Note that the running times of UPD$_{prot}$ in Table 4 are for *specific instances* of the update problem tackled by UPD$_{prot}$. The running times cannot be as good for *all instances* because the worst-case complexity of UPD$_{prot}$ is exponential w.r.t. the size of φ (see Appendix B). The results in Table 4, however, show that UPD$_{prot}$ may have an acceptable performance when applied to models with hundreds of states, without a particular structure, w.r.t. non-simple CTL formulas.

5.5. UPD$_{prot}$ applied to models with few states

Finally, we mention an example comparing UPD$_1$ and UPD$_{prot}$ with a single-state model. This example can be found at the website of UPD$_{prot}$. The formula used in this example was built as follows. First, by using 25 variables, we built a propositional formula φ resulting from the conjunction of 105 clauses with three literals. By construction, φ is satisfiable only by one truth assignment. Then, we transformed φ into a temporal formula φ' by prefixing every literal of φ with **EX**. We observe that UPD$_{prot}$ updated a dummy one-state model w.r.t. φ' in less than 30 seconds, while UPD$_1$ did not produce any model after more than one hour. When we changed the model to a two-state dummy model, UPD$_{prot}$ produced a solution in less than one second. Of course, a SAT solver can be efficiently solve these problems by using φ instead of φ'. This example, however, shows two points. First, even CTL-update instances with few states can be hard for a CTL updater. Second, an increase in the number of states may reduce the time required by a CTL updater.

6. Related work

In this section, we first give a simple example of the structure of counterexamples for CTL formulas with references to the main work on this issue. Then, we compare our work with other approaches, both for full CTL and for CTL fragments.

6.1. Structure of counterexamples

Some of the existing methods for CTL model update are wholly or partially based on counterexamples. Thus, we include here some of the main concepts of counterexamples.

In general, existential properties cannot be disproved by counterexamples. Therefore, counterexamples for CTL formulas are limited to ACTL, the universal fragment of CTL where only the path quantifier **A** is allowed, and negation is restricted to atomic subformulas.

Buccafurri et al. [1, p. 25] observe that in many cases a counterexample for an ACTL formula is (essentially) a single path, but there are cases in which a counterexample tree is required. Hence, these authors extend the concept of counterexample presented by Clarke et al. [10] through the introduction of *multi-paths* as a suitable formalization of counterexample trees.

We illustrate multi-paths with a simple example. Suppose $\mathcal{M}$ is a Kripke model such that $S^{\mathcal{M}} = \{s_0, s_1, s_2\}$, $R^{\mathcal{M}} = \{(s_0, s_1), (s_0, s_2), (s_1, s_1), (s_2, s_2)\}$, $L^{\mathcal{M}}(s_0) = \emptyset$, $L^{\mathcal{M}}(s_1) = \{p\}$ and $L^{\mathcal{M}}(s_2) = \{q\}$. If $\varphi_1 = \mathbf{AF}\, p$, then $\pi_1 = [s_0, s_2, s_2, \ldots]$ is a path having all information needed to explain why $\mathcal{M}, s_0 \not\models \varphi_1$. Thus, the single path π_1 is a *(linear) counterexample* for φ_1. Conversely, if $\varphi_2 = \mathbf{AF}\,\mathbf{AG}\, p$, then π_1 may help to track down causes of $\mathcal{M}, s_0 \not\models \varphi_2$, but π_1 does not have all the information needed to provide an explanation for this failure. A detailed explanation of why $\mathcal{M}, s_0 \not\models \varphi_2$ is contained in a multi-path $\pi_2 = [\pi_1, \pi_3, \pi_3, \ldots]$, where $\pi_3 = [s_2, s_2, \ldots]$. Each element of π_2 is a path, π, explaining why $\mathcal{M}, \pi(0) \not\models \mathbf{AG}\, p$. Hence, unlike the single path π_1, the multi-path π_2 is a *counterexample* for φ_2.

Linear counterexamples are preferred for their simplicity and efficient generation through widely used algorithms. Unfortunately, these algorithms are incomplete w.r.t. ACTL because only properties in $ACTL \cap LTL$ have linear counterexamples. In addition, it is PSPACE-hard to decide whether a given ACTL formula always admits a linear counterexample [2]. Thus, to construct linear counterexamples, an ACTL model checker must solve an unfeasible problem for detecting those cases where this construction is possible.

In a later work, Clarke et al. [9] propose *tree-like counterexamples* as a class of counterexamples not bearing the inadequacies of linear counterexamples. The class of tree-like counterexamples is complete for $A\Omega$, a logic containing ACTL, i.e., each violation of an ACTL formula is witnessed by a suitable tree-like counterexample [9]. Clarke et al. also provide an efficient algorithm to generate tree-like counterexamples for $A\Omega$.

We conclude that, in addition to considering that a formula may have more than one counterexample, update methods based on counterexamples should take into account the structure of the counterexamples.

6.2. Other approaches to CTL model update

We compare our work with other approaches in chronological order of publication: Buccafurri et al. [1], Calzone et al. [3], Zhang and Ding [22], a previous work by us [4], Ding and Hemer [13], Kelly et al. [19] and [20], Guerra and Wassermann [17], and Chatzieleftheriou et al. [7].

6.2.1. Buccafurri et al.

Buccafurri et al. [1] develop an update method, employing multi-path counterexamples to generate fewer plausible modifications than those of a naive method. Their method, based on counterexamples, is limited to ACTL.

Buccafurri et al.'s method modifies programs composed of collections of processes running in parallel. Such programs are modeled as Kripke models with an asynchronous accessibility relation having interleaving and processes that are synchronized with a fair scheduler. These authors assume a mapping between a program and its corresponding Kripke model.

Such a mapping only allows modifications at the Kripke-model level that correspond to modifications at the program level belonging to a repertoire of modifications and respecting the fairness constraints. At the program level, the modifications are of three kinds: (a) negation of the right-hand side of an assignment statement having either "true" or "false" on the right-hand side, (b) change of the variable on the left-hand side of an assignment statement, and (c) the swapping of two consecutive assignment statements. At the Kripke-model level, these modifications correspond to the addition and removal of transitions.

Given a counterexample, only statements corresponding to states occurring in such a counterexample are considered for either right-hand side assignment changes or swapping of statements. Similarly, only variables occurring in counterexamples are considered for left-hand side assignment changes. Buccafurri et al. [1] illustrate their method in a mutual-exclusion example, assuming a single error. A naive repair method makes 77 correction attempts, whereas their method makes only 17.

As a conclusion, Buccafurri et al.'s method is centered around the use of tree-like counterexamples to reduce the number of correction attempts of a naive repair method.

6.2.2. Calzone et al.

Calzone et al. [3] present a modeling system, Biocham, able to translate a biochemical network N into a Kripke model $\mathcal{M}_N$. If φ is a CTL formula expressing a property of N, and $\mathcal{M}_N$ does not satisfy φ, then, in some cases of φ, Biocham can generate an update N' such that $\mathcal{M}_{N'}$ does satisfy φ.

Biocham's update algorithm proceeds according to three kinds of CTL formulas: *universal* formulas contain only non-negated universal operators, *existential* formulas contain only non-negated existential operators, and *unclassified* formulas contain both universal and existential operators.

If φ is universal, then Biocham uses NuSMV [8] to compute a *counterexample*. Next, Biocham generates and tests models by *deleting transitions* occurring in such a counterexample. If φ is existential, then Biocham generates and tests models by *adding transitions with a bias* taken from the application domain. If φ is unclassified, then Biocham treats φ by deleting and adding transitions. However, the deletions (additions) for satisfying universal (existential) formulas may dissatisfy some existential (universal) or unclassified formulas. Therefore, trying to satisfy the three kinds of formulas, Biocham *uses a heuristic*: it first treats the existential formulas, then the unclassified ones, and finally the universal ones. If some formulas are dissatisfied by the last step, the process repeats.

On the one hand, a drawback of Biocham, compared with our method, is that the use of biases and heuristics makes it incomplete. Another disadvantage is that the use of domain-dependent biases makes it a non-general method. On the other hand, since NuSMV represents models with usually compact binary decision diagrams (BDDs), Biocham is able to process large models [6]. This is a significant advantage of Biocham compared to $\textsc{Upd}_{prot}$ or any other extensional updater.

6.2.3. Zhang and Ding

Zhang and Ding [22] devise a recursive and syntax-directed model-update method w.r.t. CTL formulas that employs "constraints". Constraints appear in the treatment of conjunction $\alpha \wedge \beta$, which is performed as follows (see Fig. 8). First Zhang and Ding's algorithm updates the input model w.r.t. α. Then such an algorithm updates the models obtained when treating α to produce models satisfying β, using α as a constraint.

* function $\text{Update}_\wedge((M, s_0), \phi_1 \wedge \phi_2)$ *
input: (M, s_0) and $\phi_1 \wedge \phi_2$, where $M = (S, R, L)$, $s_0 \in S$, and $(M, s_0) \not\models \phi_1 \wedge \phi_2$;
output: (M', s'_0), where $M' = (S', R', L')$, $s'_0 \in S'$ and $(M', s'_0) \models \phi_1 \wedge \phi_2$;
01 **begin**
02 **if** $\phi_1 \wedge \phi_2$ is a propositional formula, **then** $(M', s'_0) = \text{Update}_{prop}((M, s_0), \phi_1 \wedge \phi_2)$;
03 **else** $(M^*, s_0^*) = \text{CTLUpdate}((M, s_0), \phi_1)$;
04 $(M', s'_0) = \text{CTLUpdate}((M^*, s_0^*), \phi_2)$ with constraint ϕ_1;
05 return (M', s'_0);
06 **end**

Fig. 8. Zhang and Ding's [22, p. 141] $\text{Update}_\wedge$.

The treatment of constraints is described in Fig. 9: Candidate models are simply model-checked w.r.t. the input constraints. If a model satisfying the constraints is found, such a model is returned; otherwise, Zhang and Ding's algorithm looks for another model.

"[...] suppose $\mathcal{C}$ is the set of domain constraints for a system specification $M = (S, R, L)$, and we need to update (M, s_0) with formula ϕ, where $s_0 \in S$, and $\mathcal{C} \cup \{\phi\}$ is satisfiable. Then in each function of $CTLUpdate$, we simply add a model checking condition on the candidate model $M' = (S', R', L')$: $(M', s'_0) \models \mathcal{C}$ $(s'_0 \in S')$. The result (M', s'_0) is returned from the function if it satisfies $\mathcal{C}$. Otherwise, the function will look for another candidate model."

Fig. 9. Zhang and Ding's [22, p. 143] Constraint handling.

Let us compare the treatment of conjunction of Zhang and Ding's Update$_\wedge$ (Fig. 8) with that of our generate-and-test XUPD$_1$ (Section 3.3). Observe that Update$_\wedge$ employs the first conjunct α of a conjunction $\alpha \wedge \beta$ as a constraint for repeatedly updating w.r.t. β until a candidate model satisfying α is found. Next note that when dealing with conjunction, our XUPD$_1$ model-checks the candidate models (line 12) resulting from updating w.r.t. a conjunction $\alpha \wedge \beta$ until a model satisfying such a conjunction is found (or failure occurs), hence also model-checking w.r.t. α the models that were obtained when treating β. Consequently, the treatments by Zhang and Ding's Update$_\wedge$ and our XUPD$_1$ of formulas of the form $\alpha \wedge \beta$ are similar to each other. In comparison, UPD$_{prot}$ does not verify that an update w.r.t. β satisfies α.

Zhang and Ding [22, Theorem 8] assert that "CTLUpdate$((M, s_0), \phi)$ terminates and generates an admissible model to satisfy ϕ". However, there is no guarantee that CTLUpdate$((M, s_0), \phi)$ generates all possible "admissible" models [17]. By contrast, we prove that XUPD$_{prot}$ is sound and complete (Appendix A). Besides, Zhang and Ding's algorithm is less clear than UPD$_{prot}$ because it uses an operator base ($\{\mathbf{EX}, \mathbf{AF}, \mathbf{EU}\}$) more appropriate for model checking than for model update. For example, it is not clear how Zhang and Ding's algorithm updates a model to satisfy any of the formulas $\mathbf{AX}\alpha$, $\mathbf{EF}\alpha$, or $\mathbf{A}[\alpha\ \mathbf{U}\ \beta]$.

6.2.4. Carrillo and Rosenblueth

In another work [4], we extend a former version of UPD$_{prot}$ so as to update a concise representation of a Kripke model instead of the Kripke model itself. This former version of UPD$_{prot}$ is also based on protections, but the universal protection is different from the one we developed here. Instead of consisting of the set of transitions that we can add, as happens in UPD$_{prot}$, this former universal protection is a set of (state, formula)-pairs meant for preserving the truth value of already-treated **AX** subformulas. Recall that for $\mathbf{AX}\alpha$ to hold at s_0, α must hold at all successors of s_0. By recording a pair of the form (s_0, α), it is possible to accompany the addition of a successor of s_0 with an invocation to the update algorithm so that α holds at the new successor of s_0. The recording of (s_0, α) hence allows this method to avoid having to update the old successors again, as would happen in a generate-and-test method. The protections used in [4], however, are unsatisfactory because it is unclear if the resulting algorithm is complete.

To achieve a good scaling, this state-by-state method is extended with a concise representation of Kripke models, which can be viewed as a simplification of SMV's model-description language. The next value x_i' of a variable x_i is defined by a function $f_i : S \to \mathcal{P}(\{0, 1\}) - \{\emptyset\}$, which can be written with a number of abbreviations, including a default value. In addition, [4] provides operations modifying such descriptions of Kripke models. It remains to explore the possibility of extending UPD$_{prot}$ to such descriptions possibly achieving a better scaling than UPD$_{prot}$.

6.2.5. Ding and Hemer

Ding and Hemer [13] make an improvement over [12,22] in the treatment of formulas of the form $\mathbf{AG}\varphi$, where φ is a propositional formula. Such an improvement produces fewer candidate models than the method in Ding's thesis [12]. Essentially, the original method of [12] preserves existing paths between an initial state and an arbitrary state. The improved method [13], by contrast, preserves existing paths between any two states. Ding and Hemer apply their improved update method to a cache coherence protocol for the Andrew file system (AFS-1) [21], and generate 125 candidate models, instead of 225.

As [13] is an improvement over [22], we must now evaluate [13]. In the AFS-1 example, Ding [12] gives as input to the updater a Kripke model and a formula:

$$\mathbf{AG}\big((\mathit{Server.belief} = \mathit{valid}) \to (\mathit{Client.belief} = \mathit{valid})\big) \tag{1}$$

which is false in the input Kripke model. Note that this formula represents an undesired property, and is included in [21] only to illustrate SMV's counterexamples. Hence, the produced models have an undesired property. Ding [12, p. 100], however, observes that: "[...] after our model updating, we do not need to consider the logic outcome of the updated models under the false specification property."

Moreover, Ding ignores another formula given in [21]:

$$\mathbf{AG}\big((\mathit{Client.belief} = \mathit{valid}) \to (\mathit{Server.belief} = \mathit{valid})\big) \tag{2}$$

which is true in the input model and represents a desired property. As a result, the produced models may not satisfy (2). If (2) were considered, not only would fewer models be produced, but such models would satisfy a formula representing a desired property. When more properties are given to an updater, even fewer models are generated. For instance, we gave UPD$_{prot}$ a complete CTL specification of the input model, together with (1), reducing the number of produced models to zero (the reason is that (1) is not only false in the model, it is also inconsistent with the specification).

We conclude that the proliferation of models produced by Ding's updater is not an inherent problem of model update, but rather a consequence of ignoring desired properties. Hence, instead of preferring models preserving existing paths (the reason for which, incidentally, is not justified in [12,22]), formulas representing desired properties should be considered.

6.2.6. Kelly et al.

Kelly et al. [19] and Kelly and Zhang [20] present a counterexample-based method for ACTL. They report experiments on the mutual-exclusion problem of [1] and the sliding-window protocol, respectively.

Although the authors do not give their algorithm, we know that "[it] was designed with a top down recursive approach with respect to the given model and properties." [20, p. 16]. In addition, Kelly et al. [19] observe that "when we perform a model update, we may require this update not violate other specified functions (e.g. breaking a deadlock should *not* violate a liveness in a concurrent program)." [19, p. 138]. For this purpose, a Kripke model is augmented with "actions" labeling transitions, as well as a set of two kinds of deterministic finite-state automaton, meant to encode constraints on the values of variables, and constraints on the precedence of actions within a path, respectively. Each such automata has a distinguished "violation" state. Although Kelly et al. [19] do not describe the construction of the automata, we know that such automata encode "complex constraints that are usually not expressible [...] in the form of ACTL (or CTL) formulas." [19, p. 139]. In the experiments of both [19] and [20], the formulas are of the form $\mathbf{AG}\varphi$, where φ is a propositional formula. Kelly and Zhang [20] report updates of models with up to 512 states (though apparently the automata were not used for these experiments).

6.2.7. Guerra and Wassermann

In the next work we cover, Guerra and Wassermann [17] give an algorithm for performing CTL model revision using Zhang and Ding's CTLUpdate. Belief update supposes a dynamic world, and new information represents changes in such a world. Belief revision, by contrast, assumes a static world and the objective is to restore consistency as new information is added.

Guerra and Wassermann give the following algorithm for model revision: Let ψ be a belief base and ϕ a new belief. Let S be a set of models initialized to $\emptyset$. Models satisfying ψ are enumerated. Models satisfying ψ which also satisfy ϕ are added to S. Models satisfying ψ which do not satisfy ϕ are updated with CTLUpdate and the results are added to S. Finally, Guerra and Wassermann remove the models that are not minimal w.r.t. the models satisfying ψ from the set S.

These authors assert that their model-revision algorithm can be more adequate than Zhang and Ding's CTL model update when applied to system modifications in a static context. Since the advantages these authors find over Zhang and Ding's model-update method do not depend on the inner workings of such a method, it seems fair to conclude that Guerra and Wassermann's algorithm would also be preferable to ours for model revision if $\textsc{Upd}_{prot}$ were substituted for CTLUpdate.

6.2.8. Chatzieleftheriou et al.

Finally, Chatzieleftheriou et al. [7] develop a CTL update method employing model abstraction. An abstraction of a Kripke model is determined by a function mapping a set of concrete states to an abstract state. Instead of an accessibility relation, such an abstraction has two relations: R_{must} and R_{may}. There is a transition in R_{must} from $\hat{s}_1$ to $\hat{s}_2$ if there are transitions from all concrete states of $\hat{s}_1$ to some concrete state of $\hat{s}_2$. By contrast, there is a transition in R_{may} from $\hat{s}_1$ to $\hat{s}_2$ if there is a transition from some concrete state of $\hat{s}_1$ to some concrete state of $\hat{s}_2$. A literal labels an abstract state *only if* such a literal labels all concrete states of such an abstract state. Hence, the abstract labeling function is partial. As a result, abstraction uses 3-valued semantics. If the truth value of a formula at an abstract state is undefined, a refinement step is performed so as to get a more concrete abstraction and thus obtain a true or false value.

Like both our technique and Zhang and Ding's (2008), this method is recursive and employs additional, auxiliary information generated by other subformulas. Such information, in this case, is a set of (state, formula)-pairs, also called constraints, and is used as follows. Chatzieleftheriou et al.'s algorithm produces models satisfying both the input formula and constraints as output. At the same time, update at a state s w.r.t. $\alpha \wedge \beta$ proceeds by first treating α with (s, β) added to the set of pairs, then treating β with (s, α) added to the set of pairs, and finally "combining both results appropriately" [7, p. 349].

Let us now assess Chatzieleftheriou et al.'s method. First, although Chatzieleftheriou et al. assert the soundness of their method, these authors do not assert its completeness.

Second, Chatzieleftheriou et al. [7] give a deterministic algorithm, as opposed to our method, which is nondeterministic. We believe that a nondeterministic approach has the advantage of allowing a separation of the essence from the search strategy of the algorithm.

Lastly, Chatzieleftheriou et al. [7] use abstractions, thus introducing a relevant improvement element in terms of efficiency. This suggests that it would be promising to explore the possibility of developing an algorithm combining Chatzieleftheriou et al.'s abstractions with our protections so as to obtain a method having the best of both these methods.

7. Conclusions

Recursive, syntax-directed methods for CTL update confront two dual difficulties: The removal of a transition may dissatisfy an already-treated existential subformula. Similarly, the addition of a transition may dissatisfy an already-treated universal subformula. A direct way of dealing with this situation would be to repetitively model-check (i.e., test) a modified (i.e., generated) model w.r.t. already-treated subformulas, yielding a method similar to generate-and-test techniques. Another way would be to record information (e.g., a protection) of already-treated subformulas allowing only certain changes on the model to be modified so as to preserve the truth value of the already-treated subformulas. We showed that with appropriate protections it is possible to avoid the model-checking stage of the direct approach, thus producing a more efficient method.

We exhibited the behavior of our method, UPD$_{prot}$, on several examples. In all cases UPD$_{prot}$ behaved favorably compared with a generate-and-test method. First, we synthesized the synchronization skeleton of Emerson and Clarke's mutual-exclusion example [15] from a dummy model (having the identity relation as its accessibility relation).

Next, we formalized an informal specification of Clarke et al.'s microwave-oven example [11]. By gradually removing parts of a correct model and gradually augmenting the complexity of the input formula, we obtained different instances of this problem.

To measure the performance of UPD$_{prot}$ on larger examples, we employed two sets of problems. First, we used models representing counters of different sizes that either count or are externally reset. UPD$_{prot}$ was able to completely synthesize (from a dummy model) instances with up to 256 states in less than one hour, and repair a faulty model with 512 states, making 768 changes in the process, in around one hour of CPU time. Fewer changes would mean the possibility of handling larger instances.

Subsequently, we used randomly generated models and formulas. We showed the behavior of UPD$_{prot}$, first fixing the model and varying the formula, and then fixing the formula and varying the model. We obtained results in a reasonable time with up to 1024 states. These two sets of examples (counters and random models and formulas) show the possible usefulness of our method in abstractions of models with many more states.

Finally, we proved the soundness and completeness of XUPD$_{prot}$ (Appendix A), and we provided a complexity analysis of UPD$_{prot}$ (Appendix B).

There are several possible avenues for future research. First, we believe that our protections could potentially benefit from ideas used in other update methods, including LTL update. For example, Jobstmann et al. [18] employ games to perform LTL model update. The system to be corrected is converted to an infinite game played between the system (the protagonist) and the environment (the antagonist).

Second, the close connections between model update and reasoning about actions and change (e.g., [14]) suggest investigating the possibility of adapting our approach to methods developed for such reasoning.

Finally, a better scaling might be achieved by codifying protected models and protections with symbolic techniques, such as BDDs.

Acknowledgments

We gratefully acknowledge the facilities provided by IIMAS, UNAM and the financial support from DGAPA grant PAPIIT IN113013. We should like to thank Sergio Rajsbaum and Pedro Góngora, who gave us useful comments and suggestions, Michael Scott White, who proposed English changes, and the referees, who made insightful observations.

Appendix A. Soundness and completeness

We outline here a proof of soundness and completeness for XUPD$_{prot}$. The application of XUPD$_{prot}$ to a protected model $(\mathcal{M}, P)$ should produce protected models $(\mathcal{M}', P')$ such that P' is "greater than or equal to" P. Therefore, we start by defining a partial order on protections suitable for model update.

Definition A.1 *(Relation $\succcurlyeq$).* If $P, P' \in \mathbf{P}_\Sigma$ are two protections, then we say that P' is *greater than or equal to* P, and we write $P' \succcurlyeq P$ if $E^{P'} \supseteq E^P$, $A^{P'} \subseteq A^P$, and $\forall t \in S^\Sigma\colon\ L^{P'}(t) \supseteq L^P(t)$.

Theorem A.2. *The relation $\succcurlyeq$ is a partial order on $\mathbf{P}_\Sigma$.*

Proof. Note that the relation $\succcurlyeq$ is defined by the partial orders $\subseteq$ and $\supseteq$.

Thus, if $P \in \mathbf{P}_\Sigma$, then $E^P \supseteq E^P$, $A^P \subseteq A^P$, and $\forall t \in S^\Sigma\colon\ L^P(t) \supseteq L^P(t)$. Therefore, $P \succcurlyeq P$, and $\succcurlyeq$ is reflexive. Similarly, we can show that $\succcurlyeq$ is antisymmetric and transitive. □

Protected update operations, when applicable, produce a protected model with a protection greater than or equal to the protection of the input model. That is, these operations do not decrease protections.

Theorem A.3 *(Operations do not decrease protections). Given $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s, s' \in S^{\mathcal{M}}$, $\ell \in \mathrm{Lit}(V^\Sigma)$, $S' \subseteq S^{\mathcal{M}}$, and the following conditions:*

1. $\bar{\ell} \notin L^P(s)$ *and* $(\mathcal{M}', P') = \mathbf{L}^u((\mathcal{M}, P), s, \ell)$
2. $s' \in A^P[s]$ *and* $(\mathcal{M}', P') = \mathbf{T}^+_\exists((\mathcal{M}, P), s, s')$
3. $E^P[s] \subseteq S' \subseteq A^P[s]$, $S' \neq \emptyset$, *and* $(\mathcal{M}', P') = \mathbf{T}^u_\forall((\mathcal{M}, P), s, S')$
4. $E^P[s] \subseteq S' \subseteq A^P[s]$, $S' \neq \emptyset$, *and* $(\mathcal{M}', P') = \mathbf{T}^u_\exists((\mathcal{M}, P), s, S')$,

if (1), *or* (2), *or* (3), *or* (4), *then* $(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$ *and* $P' \succcurlyeq P$.

Proof. (According to the above 1–4 conditions)

1. If $\bar{\ell} \notin L^P(s)$ and $(\mathcal{M}', P') = \mathbf{L}^u((\mathcal{M}, P), s, \ell)$, then
$(\mathcal{M}', P') = (\mathbf{L}^u(\mathcal{M}, s, \ell), \langle E^P, A^P, L^P[s \oplus \ell]\rangle)$.
Therefore, $\mathcal{M}' = \mathbf{L}^u(\mathcal{M}, s, \ell) = \langle S^{\mathcal{M}}, R^{\mathcal{M}}, L^{\mathcal{M}}[s \oplus \ell]\rangle$, and
$P' = \langle E^P, A^P, L^P[s \oplus \ell]\rangle$.
Hence, the only difference between $\mathcal{M}'$ and $\mathcal{M}$, and between P' and P is in the third component.
Since $\mathcal{M} \rhd P$, we have $E^P \subseteq R^{\mathcal{M}} \subseteq A^P$, and $\forall t \in S^{\mathcal{M}} : L^{\mathcal{M}}(t) \supseteq L^P(t)$. Thus, to conclude that $\mathcal{M}' \rhd P'$, all we need to show is that $\forall t \in S^{\mathcal{M}} : L^{\mathcal{M}}[s \oplus \ell](t) \supseteq L^P[s \oplus \ell](t)$.
If $t \neq s$ then $L^{\mathcal{M}}[s \oplus \ell](t) = L^{\mathcal{M}}(t) \supseteq L^P(t) = L^P[s \oplus \ell](t)$.
If $t = s$ then $L^{\mathcal{M}}[s \oplus \ell](t) = (L^{\mathcal{M}}(s) \cup \{\ell\}) - \{\bar{\ell}\} \supseteq (L^P(s) \cup \{\ell\}) - \{\bar{\ell}\} = L^P[s \oplus \ell](t)$.
Hence, $\mathcal{M}' \rhd P'$ and $(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$.
Besides, $E^{P'} = E^P \supseteq E^P$, and $A^{P'} = A^P \subseteq A^P$. Thus, to conclude that $P' \succcurlyeq P$, all we need to show is that $\forall t \in S^\Sigma : L^{P'}(t) \supseteq L^P(t)$.
If $t \neq s$ then $L^{P'}(t) = L^P[s \oplus \ell](t) = L^P(t) \supseteq L^P(t)$.
If $t = s$ then $L^{P'}(t) = L^P[s \oplus \ell](t) = (L^P(s) \cup \{\ell\}) - \{\bar{\ell}\} \supseteq L^P(s) = L^P(t)$, because $\bar{\ell} \notin L^P(s)$.
Therefore, $P' \succcurlyeq P$.
2. If $s' \in A^P[s]$ and $(\mathcal{M}', P') = \mathbf{T}^+_\exists((\mathcal{M}, P), s, s')$, then
$(\mathcal{M}', P') = (\mathbf{T}^+(\mathcal{M}, s, s'), \langle E^P \cup \{(s, s')\}, A^P, L^P\rangle)$
Therefore, $\mathcal{M}' = \mathbf{T}^+(\mathcal{M}, s, s') = \langle S^{\mathcal{M}}, R^{\mathcal{M}} \cup \{(s, s')\}, L^{\mathcal{M}}\rangle$, and
$P' = \langle E^P \cup \{(s, s')\}, A^P, L^P\rangle$.
Hence, the only difference between $\mathcal{M}'$ and $\mathcal{M}$, and between P' and P is in the first and second components, respectively.
Since $\mathcal{M} \rhd P$, we have $E^P \subseteq R^{\mathcal{M}} \subseteq A^P$, and $\forall t \in S^{\mathcal{M}}$: $L^{\mathcal{M}}(t) \supseteq L^P(t)$. Thus, to conclude that $\mathcal{M}' \rhd P'$, all we need to show is that $E^{P'} \subseteq R^{\mathcal{M}'} \subseteq A^{P'}$.
Since $s' \in A^P[s]$, $E^{P'} = E^P \cup \{(s, s')\} \subseteq R^{\mathcal{M}} \cup \{(s, s')\} = R^{\mathcal{M}'} \subseteq A^P \cup \{(s, s')\} = A^{P'}$.
Hence, $\mathcal{M}' \rhd P'$ and $(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$.
Besides, we have $A^{P'} = A^P \subseteq A^P$, and $\forall t \in S^\Sigma$: $L^{P'}(t) = L^P(t) \supseteq L^P(t)$. Thus, to conclude that $P' \succcurlyeq P$, all we need to show is that $E^{P'} \supseteq E^P$.
Clearly, $E^{P'} = E^P \cup \{(s, s')\} \supseteq E^P$.
Therefore, $P' \succcurlyeq P$.
3. If $E^P[s] \subseteq S' \subseteq A^P[s]$, $S' \neq \emptyset$, and $(\mathcal{M}', P') = \mathbf{T}^u_\forall((\mathcal{M}, P), s, S')$, then
$(\mathcal{M}', P') = (\mathbf{T}^u(\mathcal{M}, s, S'), \langle E^P, A^P - (\{s\} \times (A^P[s] - S')), L^P\rangle)$.
Therefore, $\mathcal{M}' = \mathbf{T}^u(\mathcal{M}, s, S') = \langle S^{\mathcal{M}}, (R^{\mathcal{M}} - (\{s\} \times R^{\mathcal{M}}[s])) \cup (\{s\} \times S'), L^{\mathcal{M}}\rangle$, and $P' = \langle E^P, A^P - (\{s\} \times (A^P[s] - S')), L^P\rangle$.
Hence, the only difference between $\mathcal{M}'$ and $\mathcal{M}$, and between P' and P is in the second component.
Since $\mathcal{M} \rhd P$, we have $E^P \subseteq R^{\mathcal{M}} \subseteq A^P$, and $\forall t \in S^{\mathcal{M}}$: $L^{\mathcal{M}}(t) \supseteq L^P(t)$. Thus, to conclude that $\mathcal{M}' \rhd P'$, all we need to show is that $E^{P'} \subseteq R^{\mathcal{M}'} \subseteq A^{P'}$.
Since $E^P[s] \subseteq S' \subseteq A^P[s]$, and $S' \neq \emptyset$, we have
$E^{P'} = E^P \subseteq (R^{\mathcal{M}} - (\{s\} \times R^{\mathcal{M}}[s])) \cup (\{s\} \times S') \subseteq A^P - (\{s\} \times (A^P[s] - S')) = A^{P'}$.
Hence, $\mathcal{M}' \rhd P'$ and $(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$.
Besides, we have $E^{P'} = E^P \supseteq E^P$, and $\forall t \in S^\Sigma$: $L^{P'}(t) = L^P(t) \supseteq L^P(t)$. Thus, to conclude that $P' \succcurlyeq P$, all we need to show is that $A^{P'} \subseteq A^P$.
Clearly, $A^{P'} = A^P - (\{s\} \times (A^P[s] - S')) \subseteq A^P$.
Therefore, $P' \succcurlyeq P$.
4. If $E^P[s] \subseteq S' \subseteq A^P[s]$, $S' \neq \emptyset$, and $(\mathcal{M}', P') = \mathbf{T}^u_\exists((\mathcal{M}, P), s, S')$, then
$(\mathcal{M}', P') = (\mathbf{T}^u(\mathcal{M}, s, S'), \langle E^P \cup (\{s\} \times (S' - E^P[s])), A^P, L^P\rangle)$.
In a manner analogous to the above case, we can show that $(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$ and $P' \succcurlyeq P$. □

As a consequence of Theorem A.3, we get one of the key features of the modification of protected models: The protection of a modification is greater than or equal to the protection of the input model.

Theorem A.4 *(Modifications do not decrease protections). Let $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s \in S^{\mathcal{M}}$, $S' \subseteq S^{\mathcal{M}}$, and $\varphi \in \Sigma$-XCTL.*
If $(\mathcal{M}', P') \in Modif((\mathcal{M}, P), s, \varphi)$ or $(\mathcal{M}', P') \in Modif^*((\mathcal{M}, P), S', \varphi)$ then $P' \succcurlyeq P$.

Proof. If $(\mathcal{M}', P') \in Modif((\mathcal{M}, P), s, \varphi)$ or $(\mathcal{M}', P') \in Modif^*((\mathcal{M}, P), S', \varphi)$, then either $(\mathcal{M}', P') = (\mathcal{M}, P)$, or $(\mathcal{M}', P')$ is obtained from $(\mathcal{M}, P)$ by the application of protected update operations. Therefore, by Theorem A.3, $P' \succcurlyeq P$. □

Theorem A.5 *($\succcurlyeq$ preserves satisfiability). Let $(\mathcal{M}, P), (\mathcal{M}', P') \in \mathbf{KP}_\Sigma$, $s \in S^{\mathcal{M}}$, and $\varphi \in \Sigma$-XCTL. If $P' \succcurlyeq P$ and $(\mathcal{M}, P), s \models \varphi$ then $(\mathcal{M}', P'), s \models \varphi$.*

Proof. Suppose $P' \succcurlyeq P$ and $(\mathcal{M}, P), s \models \varphi$. We proceed by induction on the structure of φ.
Basis.

1. Cases $\varphi = \mathsf{F}$ and $\varphi = \mathsf{T}$ are trivial.
2. If $\varphi = \ell$, then $\ell \in L^P(s) \subseteq L^{P'}(s)$. Thus, $(\mathcal{M}', P'), s \models \ell$.
3. Cases of $\varphi = \mathbf{OD}_{\leqslant} n$ and $\varphi = \mathbf{OD}_{>} n$ are similar to the case $\varphi = \mathbf{AX}\alpha$ (see below).

Induction. Let IH be the induction hypothesis.

1. If $\varphi = \alpha \vee \beta$, then w.l.o.g. $(\mathcal{M}, P), s \models \alpha$.
   Therefore, by IH, $(\mathcal{M}', P'), s \models \alpha$. Thus, $(\mathcal{M}', P'), s \models \alpha \vee \beta$.
2. The case $\varphi = \alpha \wedge \beta$ is similar to the case $\varphi = \alpha \vee \beta$.
3. If $\varphi = \mathbf{EX}\alpha$, then $\exists t \in E^P[s] \subseteq E^{P'}[s]$ such that $(\mathcal{M}, P), t \models \alpha$. Therefore, by IH, $(\mathcal{M}', P'), t \models \alpha$. Thus, $(\mathcal{M}', P')$, $s \models \mathbf{EX}\alpha$.
4. If $\varphi = \mathbf{AX}\alpha$, then $\forall t \in A^P[s] \supseteq A^{P'}[s]$, $(\mathcal{M}, P), t \models \alpha$. Therefore, by IH, $\forall t \in A^{P'}[s]$, $(\mathcal{M}', P'), t \models \alpha$. Thus, $(\mathcal{M}', P')$, $s \models \mathbf{AX}\alpha$. □

Next, we show in three steps that *Modif* and *Modif** are sound. First, given $\varphi \in \Sigma$-XCTL, $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, and $S' \subseteq S^{\mathcal{M}}$, we show that *Modif** is conditionally sound: assuming that *Modif* is sound for φ and $(\mathcal{M}, P)$, we show that *Modif** is sound by induction on $|S'|$. Second, using the conditional soundness of *Modif**, we show that *Modif* is sound by induction on φ. Finally, as a corollary, we obtain the soundness of *Modif**.

Theorem A.6 *(Conditional soundness of Modif***). Let $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$ and $\varphi \in \Sigma$-XCTL such that*

$$\forall s \in S^{\mathcal{M}},\ \text{if } \big(\mathcal{M}', P'\big) \in \mathit{Modif}\big((\mathcal{M}, P), s, \varphi\big) \text{ then } \big(\mathcal{M}', P'\big), s \models \varphi \tag{A.1}$$

Then $\forall S' \subseteq S^{\mathcal{M}}$, if $(\mathcal{M}', P') \in \mathit{Modif}^*((\mathcal{M}, P), S', \varphi)$ then $(\mathcal{M}', P'), S' \models \varphi$.

Proof. Induction on $|S'|$.
Basis. If $|S'| = 0$, then $S' = \emptyset$. Therefore, $(\mathcal{M}', P'), S' \models \varphi$.
Induction. Suppose $0 < m \in \mathbb{N}$ and assume that

$$|S'| < m \Rightarrow \Big(\big(\mathcal{M}', P'\big) \in \mathit{Modif}^*\big((\mathcal{M}, P), S', \varphi\big) \Rightarrow \big(\mathcal{M}', P'\big), S' \models \varphi\Big) \tag{A.2}$$

We show that if $|S'| = m$ and $(\mathcal{M}', P') \in \mathit{Modif}^*((\mathcal{M}, P), S', \varphi)$, then $(\mathcal{M}', P'), S' \models \varphi$.

Since $|S'| = m > 0$, $S' \neq \emptyset$. Thus, by Definition 3.4, if $(\mathcal{M}', P') \in \mathit{Modif}^*((\mathcal{M}, P), S', \varphi)$, then $\exists t \in S^{\mathcal{M}}$ and $\exists(\mathcal{M}_t, P_t) \in \mathit{Modif}((\mathcal{M}, P), t, \varphi)$, such that $(\mathcal{M}', P') \in \mathit{Modif}^*((\mathcal{M}_t, P_t), S' - \{t\}, \varphi)$. By (A.2), $(\mathcal{M}', P'), S' - \{t\} \models \varphi$ and, by (A.1), $(\mathcal{M}_t, P_t), t \models \varphi$. Besides, by Theorem A.4, $P' \succcurlyeq P_t$ and $P_t \succcurlyeq P$. Thus, $P' \succcurlyeq P$ and $(\mathcal{M}', P'), t \models \varphi$. Hence, $(\mathcal{M}', P'), S' \models \varphi$. □

Theorem A.7 *(Soundness of Modif). For all $\varphi \in \Sigma$-XCTL, $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, and $s \in S^{\mathcal{M}}$: $(\mathcal{M}', P') \in \mathit{Modif}((\mathcal{M}, P), s, \varphi) \Rightarrow (\mathcal{M}', P'), s \models \varphi$.*

Proof. Induction on the structure of φ.
Let $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s \in S^{\mathcal{M}}$, and suppose $(\mathcal{M}', P') \in \mathit{Modif}((\mathcal{M}, P), s, \varphi)$.
Basis.

1. Cases $\varphi = \mathsf{F}$ and $\varphi = \mathsf{T}$ are trivial.
2. If $\varphi = \ell$, we have two cases: $\bar{\ell} \in L^P[s]$ and $\bar{\ell} \notin L^P[s]$.
   If $\bar{\ell} \in L^P[s]$, then $\mathit{Modif}((\mathcal{M}, P), s, \ell) = \emptyset$.
   If $\bar{\ell} \notin L^P[s]$, then $\mathit{Modif}((\mathcal{M}, P), s, \ell)$ uses $\mathbf{L}^u$ to generate exactly one model, $(\mathcal{M}', P')$, by adding ℓ to the protected labels of s. Thus, $\ell \in L^{P'}(s)$, $(\mathcal{M}', P'), s \models \ell$, and $\forall s \in S^{\mathcal{M}}$, $Q(s, \ell)$.
3. If $\varphi = \mathbf{OD}_{\leqslant} n$, then $\exists S' \subseteq A^P[s]$ such that $E^P[s] \subseteq S' \neq \emptyset$, $|S'| \leqslant n$, and $(\mathcal{M}', P') = \mathbf{T}^u_\forall((\mathcal{M}, P), s, S')$. Therefore, by the definition of $\mathbf{T}^u_\forall$, $A^{P'}[s] \subseteq S'$. Thus, $|A^{P'}[s]| \leqslant |S'| \leqslant n$ and $(\mathcal{M}', P'), s \models \mathbf{OD}_{\leqslant} n$.
4. The case $\varphi = \mathbf{OD}_{>} n$ is similar to the case $\varphi = \mathbf{OD}_{\leqslant} n$.

Induction. Let IH be the induction hypothesis, i.e., IH states that the theorem is true for $\varphi = \alpha, \beta$.

1. If $\varphi = \alpha \vee \beta$, then, w.o.l.g. $(\mathcal{M}', P') \in \mathit{Modif}((\mathcal{M}, P), s, \alpha)$. By IH, $(\mathcal{M}', P'), s \models \alpha$. Therefore, $(\mathcal{M}', P'), s \models \alpha \vee \beta$.
2. If $\varphi = \alpha \wedge \beta$, then $\exists(\mathcal{M}_\alpha, P_\alpha) \in \mathit{Modif}((\mathcal{M}, P), s, \alpha)$ such that
   $(\mathcal{M}', P') \in \mathit{Modif}((\mathcal{M}_\alpha, P_\alpha), s, \beta)$.
   By IH, $(\mathcal{M}_\alpha, P_\alpha), s \models \alpha$ and $(\mathcal{M}', P'), s \models \beta$. Since $P' \succcurlyeq P_\alpha$, by Theorem A.5, $(\mathcal{M}', P'), s \models \alpha$. Therefore, $(\mathcal{M}', P')$, $s \models \alpha \wedge \beta$.

3. If $\varphi = \mathbf{EX}\alpha$, then $\exists t \in A^P[s]$ such that
   $(\mathcal{M}', P') \in \mathit{Modif}(\mathbf{T}^+_\exists((\mathcal{M}, P), s, t), t, \alpha)$.
   By IH, $(\mathcal{M}', P'), t \models \alpha$. Besides, if $(\mathcal{M}_t, P_t) = \mathbf{T}^+_\exists((\mathcal{M}, P), s, t)$, then $t \in E^{P_t}[s]$, and by Theorem A.4 $P' \succcurlyeq P_t$. Thus, $t \in E^{P'}[s]$ and $(\mathcal{M}', P'), t \models \alpha$. Therefore, $(\mathcal{M}', P'), s \models \mathbf{EX}\alpha$.
4. If $\varphi = \mathbf{AX}\alpha$, then $\exists S' \subseteq A^P[s]$ such that $E^P[s] \subseteq S' \neq \emptyset$ and $(\mathcal{M}', P') \in \mathit{Modif}^*((\mathcal{M}_{S'}, P_{S'}), S', \alpha)$, where $(\mathcal{M}_{S'}, P_{S'}) = \mathbf{T}^u_\forall((\mathcal{M}, P), s, S')$.
   By IH, $\forall s \in S^{\mathcal{M}}$, if $(\mathcal{M}'', P'') \in \mathit{Modif}((\mathcal{M}_{S'}, P_{S'}), s, \alpha)$ then
   $(\mathcal{M}'', P''), s \models \alpha$.
   Therefore, by Theorem A.6, $(\mathcal{M}', P'), S' \models \alpha$. By the definition of $P_{S'}$, we have $S' = A^{P_{S'}}[s]$, and, by Theorem A.4, $P' \succcurlyeq P_{S'}$. Thus, $S' = A^{P_{S'}}[s] \supseteq A^{P'}[s]$. Since $(\mathcal{M}', P'), S' \models \alpha$ and $A^{P'}[s] \subseteq S'$, $(\mathcal{M}', P'), A^{P'}[s] \models \alpha$. Therefore, $(\mathcal{M}', P'), s \models \mathbf{AX}\alpha$. □

Corollary A.8 *(Soundness of Modif*). For all $\varphi \in \Sigma$-XCTL, $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, and $S' \subseteq S^{\mathcal{M}}$: $(\mathcal{M}', P') \in \mathit{Modif}^*((\mathcal{M}, P), S', \varphi) \Rightarrow (\mathcal{M}', P'), S' \models \varphi$.*

Proof. Let $\varphi \in \Sigma$-XCTL, $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $S' \subseteq S^{\mathcal{M}}$, and suppose $(\mathcal{M}', P') \in \mathit{Modif}^*((\mathcal{M}, P), S', \varphi)$. By Theorem A.7,
$\forall s \in S^{\mathcal{M}}$, $(\mathcal{M}'', P'') \in \mathit{Modif}((\mathcal{M}, P), s, \varphi) \Rightarrow (\mathcal{M}'', P''), s \models \varphi$.
Therefore, by Theorem A.6, $(\mathcal{M}', P'), S' \models \varphi$. □

To prove completeness of *Modif*, it is important to know whether there is a modification of $(\mathcal{M}, P)$ for φ, i.e., whether $(\mathcal{M}, P)$ is "φ-modifiable".

Definition A.9 *(φ-modifiable).* Let $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s \in S^{\mathcal{M}}$, and $\varphi \in \Sigma$-CTL. $(\mathcal{M}, P)$ is *φ-modifiable* at s if $\mathit{Modif}((\mathcal{M}, P), s, \varphi) \neq \emptyset$.

In addition, we need to prove that if φ is "satisfiable", then XUPD$_{prot}$ produces at least one result. Thus, we need to clarify the notion of "satisfiable".

Definition A.10 *(Satisfiable).* Let $\varphi \in \Sigma$-CTL, $P \in \mathbf{P}_\Sigma$, and $s \in S^\Sigma$.

1. φ is *P-satisfiable at s* if $\exists\ (\mathcal{M}', P') \in \mathbf{KP}_\Sigma.\ (\mathcal{M}', P'), s \models \varphi\ \&\ P' \succcurlyeq P$.
2. φ is *$\mathbf{K}_\Sigma$-satisfiable at s* if $\exists\ \mathcal{M}' \in \mathbf{K}_\Sigma.\ \mathcal{M}', s \models \varphi$.

To state completeness of *Modif* it is not enough to reverse the implication of Theorem A.7. We analyze two wrong statements of completeness.

First, the statement
(1) $(\mathcal{M}', P'), s \models \varphi \Rightarrow (\mathcal{M}', P') \in \mathit{Modif}((\mathcal{M}, P), s, \varphi)$ is not true.
For example, if $\bar{\ell}_1 \in L^P(s)$ and $\ell_1 \in L^{P'}(s)$, then $(\mathcal{M}', P') \models \ell_1$ but $(\mathcal{M}', P') \notin \mathit{Modif}((\mathcal{M}, P), s, \ell_1) = \emptyset$.
Second, a weaker statement,
(2) $(\mathcal{M}', P'), s \models \varphi$ and $P' \succcurlyeq P \Rightarrow (\mathcal{M}', P') \in \mathit{Modif}((\mathcal{M}, P), s, \varphi)$, is not true either.
For example, if $\bar{\ell}_1, \bar{\ell}_2 \notin L^P(s)$, and $(\mathcal{M}', P')$ is the result of adding ℓ_1 and ℓ_2 to $(\mathcal{M}, P)$, i.e., $(\mathcal{M}_{\ell_1}, P_{\ell_1}) = \mathbf{L}^u((\mathcal{M}, P), s, \ell_1)$, and $(\mathcal{M}', P') = \mathbf{L}^u((\mathcal{M}_{\ell_1}, P_{\ell_1}), s, \ell_2)$, then
$(\mathcal{M}', P') \models \ell_1$ and $P' \succcurlyeq P$
but $(\mathcal{M}', P') \notin \mathit{Modif}((\mathcal{M}, P), s, \ell_1) = \{(\mathcal{M}_{\ell_1}, P_{\ell_1})\}$.
According to our purposes (e.g., Corollary A.15), a more appropriate statement for completeness of *Modif* is as in Theorem A.12. To prove this theorem, we follow three steps. First, we show that *Modif** is conditionally complete: assuming that *Modif* is complete for given φ and $(\mathcal{M}, P)$, we show that *Modif** is complete by induction on $|S'|$. Second, using the conditional completeness of *Modif**, we show that *Modif* is complete by induction on φ. Finally, as a corollary, we obtain the completeness of *Modif**.

Theorem A.11 *(Conditional completeness of Modif*). Let $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$ and $\varphi \in \Sigma$-XCTL such that*

$$\forall s \in S^{\mathcal{M}},\ \big(\exists(\mathcal{M}', P') \in \mathbf{KP}_\Sigma.(\mathcal{M}', P'), s \models \varphi \wedge P' \succcurlyeq P\big) \Rightarrow \big(\exists(\mathcal{M}_{s,\varphi}, P_{s,\varphi}) \in \mathit{Modif}\big((\mathcal{M}, P), s, \varphi\big).\ P' \succcurlyeq P_{s,\varphi}\big) \tag{A.3}$$

Then $\forall S' \subseteq S^{\mathcal{M}}$, if $\exists(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$ such that $(\mathcal{M}', P'), S' \models \varphi$ and $P' \succcurlyeq P$, then $\exists(\mathcal{M}_{S',\varphi}, P_{S',\varphi}) \in \mathit{Modif}^*((\mathcal{M}, P), S', \varphi)$ such that $P' \succcurlyeq P_{S',\varphi}$.

Proof. Induction on $|S'|$.

Basis. If $|S'| = 0$, then $S' = \emptyset$ and $Modif^*((\mathcal{M}, P), \emptyset, \varphi) = \{(\mathcal{M}, P)\}$. Therefore, if we take $(\mathcal{M}_{S',\varphi}, P_{S',\varphi}) = (\mathcal{M}, P) \in Modif^*((\mathcal{M}, P), \emptyset, \varphi)$, then $P' \succcurlyeq P = P_{S',\varphi}$.

Induction. Suppose $0 < m \in \mathbb{N}$ and assume that

$$\begin{aligned}&\big(S' \subseteq S^{\mathcal{M}} \wedge |S'| < m\big) \quad \Rightarrow \\ &\qquad \big(\big(\exists(\mathcal{M}', P') \in \mathbf{KP}_\Sigma . (\mathcal{M}', P'), S' \models \varphi \wedge P' \succcurlyeq P\big) \quad \Rightarrow \\ &\qquad \big(\exists(\mathcal{M}_{S',\varphi}, P_{S',\varphi}) \in Modif^*\big((\mathcal{M}, P), S', \varphi\big).\ P' \succcurlyeq P_{S',\varphi}\big)\big)\end{aligned} \tag{A.4}$$

We show that if $S' \subseteq S^{\mathcal{M}}$, $|S'| = m$, and $\exists(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$ such that $(\mathcal{M}', P'), S' \models \varphi$ and $P' \succcurlyeq P$, then $\exists(\mathcal{M}_{S',\varphi}, P_{S',\varphi}) \in Modif^*((\mathcal{M}, P), S', \varphi)$ such that $P' \succcurlyeq P_{S',\varphi}$.

Let $S' \subseteq S^{\mathcal{M}}$ such that $|S'| = m > 0$. Suppose $(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$ is such that (1) $(\mathcal{M}', P'), S' \models \varphi$ and (2) $P' \succcurlyeq P$.

Since $|S'| > 0$, $S' \neq \emptyset$. Let $t \in S'$ and $S'_t = S' - \{t\}$.

On the one hand, by (1) and (2), $(\mathcal{M}', P'), t \models \varphi$ and $P' \succcurlyeq P$. Thus, by (A.3), there exists $(\mathcal{M}_{t,\varphi}, P_{t,\varphi}) \in Modif((\mathcal{M}, P), s, \varphi)$ such that $P' \succcurlyeq P_{t,\varphi}$. On the other hand, by (1), $(\mathcal{M}', P'), S'_t \models \varphi$ and $P' \succcurlyeq P_{t,\varphi}$. Thus, by (A.4), $\exists(\mathcal{M}_{S'_t,\varphi}, P_{S'_t,\varphi}) \in Modif^*((\mathcal{M}_{t,\varphi}, P_{t,\varphi}), S'_t, \varphi)$ such that $P' \succcurlyeq P_{S'_t,\varphi}$.

Hence, by using $S'_t = S' - \{t\}$ and Definition 3.4, $(\mathcal{M}_{S'_t,\varphi}, P_{S'_t,\varphi}) \in Modif^*((\mathcal{M}, P), S', \varphi)$ and $P' \succcurlyeq P_{S',\varphi}$. □

Theorem A.12 *(Completeness of Modif). For all $\varphi \in \Sigma$-XCTL, $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, and $s \in S^{\mathcal{M}}$: if $\exists(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$ such that $(\mathcal{M}', P'), s \models \varphi$ and $P' \succcurlyeq P$ then $\exists(\mathcal{M}_{s,\varphi}, P_{s,\varphi}) \in Modif((\mathcal{M}, P), s, \varphi)$ such that $P' \succcurlyeq P_{s,\varphi}$.*

Proof. Induction on the structure of φ.

Let $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s \in S^{\mathcal{M}}$, and suppose $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$ is such that

$$(\mathcal{M}', P'), s \models \varphi \tag{A.5}$$

$$\text{and } P' \succcurlyeq P \tag{A.6}$$

Basis.

1. If $\varphi = \mathsf{F}$, then $\nexists(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$ such that $(\mathcal{M}', P'), s \models \mathsf{F}$ and $P' \succcurlyeq P$.
2. If $\varphi = \mathsf{T}$, take $(\mathcal{M}_{s,\mathsf{T}}, P_{s,\mathsf{T}}) = (\mathcal{M}, P) \in Modif((\mathcal{M}, P), s, \mathsf{T})$, i.e. $(\mathcal{M}_{s,\mathsf{T}}, P_{s,\mathsf{T}}) = \{(\mathcal{M}, P)\}$. Thus, $P' \succcurlyeq P = P_{s,\mathsf{T}}$.
3. If $\varphi = \ell$, then, by (A.5), $\ell \in L^{P'}(s)$. Therefore, $\bar{\ell} \notin L^{P'}(s)$ and, by (A.6), $\bar{\ell} \notin L^{P}(s)$. Let $(\mathcal{M}_{s,\ell}, P_{s,\ell}) = \mathbf{L}^u((\mathcal{M}, P), s, \ell)$, the only element of $Modif((\mathcal{M}, P), s, \ell)$. By Theorem A.3, $(\mathcal{M}_{s,\ell}, P_{s,\ell}) \in \mathbf{KP}_\Sigma$ (and $P_{s,\ell} \succcurlyeq P$). Besides, $P' \succcurlyeq P$, $E^{P_{s,\ell}} = E^P$, $A^{P_{s,\ell}} = A^P$, and $L^{P_{s,\ell}} = L^P[s \oplus \ell]$; therefore $P' \succcurlyeq P_{s,\ell}$. Hence, $(\mathcal{M}_{s,\ell}, P_{s,\ell}) \in Modif((\mathcal{M}, P), s, \ell)$ and $P' \succcurlyeq P_{s,\ell}$.
4. Cases $\varphi = \mathbf{OD}_{\leqslant} n$ and $\varphi = \mathbf{OD}_{>} n$ are similar to the case $\varphi = \mathbf{AX}\alpha$ (see below).

Induction. Let IH be the induction hypothesis, i.e., IH states that the theorem is true for $\varphi = \alpha$ and $\varphi = \beta$.

1. $\varphi = \alpha \vee \beta$. By (A.5), and w.l.o.g. $(\mathcal{M}', P'), s \models \alpha$. Thus, by IH, $\exists(\mathcal{M}_{s,\alpha}, P_{s,\alpha}) \in Modif((\mathcal{M}, P), s, \alpha)$ such that $P' \succcurlyeq P_{s,\alpha}$. Hence, $(\mathcal{M}_{s,\alpha}, P_{s,\alpha}) \in Modif((\mathcal{M}, P), s, \alpha \vee \beta)$ and $P' \succcurlyeq P_{s,\alpha}$.
2. $\varphi = \alpha \wedge \beta$. By (A.5), $(\mathcal{M}', P'), s \models \alpha$. Thus, by IH, $\exists(\mathcal{M}_{s,\alpha}, P_{s,\alpha}) \in Modif((\mathcal{M}, P), s, \alpha)$ such that $P' \succcurlyeq P_{s,\alpha}$. Therefore, $P' \succcurlyeq P_{s,\alpha}$ and, by (A.5), $(\mathcal{M}', P'), s \models \beta$. Thus, by applying IH with $(\mathcal{M}, P) = (\mathcal{M}_{s,\alpha}, P_{s,\alpha})$ and $\varphi = \beta$, we have that $\exists(\mathcal{M}_{s,\beta}, P_{s,\beta}) \in Modif((\mathcal{M}_{s,\alpha}, P_{s,\alpha}), s, \beta)$ such that $P' \succcurlyeq P_{s,\beta}$. Hence, $(\mathcal{M}_{s,\beta}, P_{s,\beta}) \in Modif((\mathcal{M}, P), s, \alpha \wedge \beta)$ and $P' \succcurlyeq P_{s,\beta}$.
3. $\varphi = \mathbf{EX}\alpha$. By (A.5), $\exists s' \in E^{P'}[s]$ such that $(\mathcal{M}', P'), s' \models \alpha$. Let $(\mathcal{M}_{s'}, P_{s'}) = \mathbf{T}^+_\exists((\mathcal{M}, P), s, s')$. Since $s' \in E^{P'}[s]$, by (A.6), $P' \succcurlyeq P_{s'}$. Thus, by applying IH with $(\mathcal{M}, P) = (\mathcal{M}_{s'}, P_{s'})$ and $\varphi = \alpha$, we have that $\exists(\mathcal{M}_{s',\alpha}, P_{s',\alpha}) \in Modif((\mathcal{M}_{s'}, P_{s'}), s', \alpha)$ such that $P' \succcurlyeq P_{s',\alpha}$. Thus, $s' \in E^{P'}[s] \subseteq A^{P'}[s]$ and $(\mathcal{M}_{s',\alpha}, P_{s',\alpha}) \in Modif(\mathbf{T}^+_\exists((\mathcal{M}, P), s, s'), s', \alpha)$. Therefore, $(\mathcal{M}_{s',\alpha}, P_{s',\alpha}) \in Modif((\mathcal{M}, P), s, \mathbf{EX}\alpha)$ and $P' \succcurlyeq P_{s',\alpha}$.
4. $\varphi = \mathbf{AX}\alpha$. By (A.5), $\forall s' \in A^{P'}[s]$, $(\mathcal{M}', P'), s' \models \alpha$. Let $S' = A^{P'}[s]$ and $(\mathcal{M}_{S'}, P_{S'}) = \mathbf{T}^u_\forall((\mathcal{M}, P), s, S')$. Since $S' \subseteq A^{P'}[s]$, by (A.6), $P' \succcurlyeq P_{S'}$. Thus, by applying IH with $(\mathcal{M}, P) = (\mathcal{M}_{S'}, P_{S'})$ and $\varphi = \alpha$, we have that $\exists(\mathcal{M}_{S',\alpha}, P_{S',\alpha}) \in Modif^*((\mathcal{M}_{S'}, P_{S'}), S', \alpha)$ such that $P' \succcurlyeq P_{S',\alpha}$. Since $S' = A^{P'}[s] \supseteq E^{P'}[s]$ and $A^{P'}[s] \supseteq R^{\mathcal{M}'}[s] \neq \emptyset$, we have $S' \subseteq A^{P'}[s]$ and $E^{P'}[s] \subseteq S' \neq \emptyset$. Thus, $S' \subseteq A^{P'}[s]$, $E^{P'}[s] \subseteq S' \neq \emptyset$, and $(\mathcal{M}_{S',\alpha}, P_{S',\alpha}) \in Modif^*(\mathbf{T}^u_\forall((\mathcal{M}, P), s, S'), S', \alpha)$. Therefore, $(\mathcal{M}_{S',\alpha}, P_{S',\alpha}) \in Modif((\mathcal{M}, P), s, \mathbf{AX}\alpha)$ and $P' \succcurlyeq P_{S',\alpha}$. □

Corollary A.13 *(Completeness of Modif*). For all $\varphi \in \Sigma$-XCTL, $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, and $S' \subseteq S^{\mathcal{M}}$: if $\exists(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$ such that $(\mathcal{M}', P'), S' \models \varphi$ and $P' \succcurlyeq P$, then $\exists(\mathcal{M}_{S',\varphi}, P_{S',\varphi}) \in Modif^*((\mathcal{M}, P), S', \varphi)$ such that $P' \succcurlyeq P_{S',\varphi}$.*

Proof. Let $\varphi \in \Sigma$-XCTL, $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $S' \subseteq S^{\mathcal{M}}$, and suppose $(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$ is such that (1) $(\mathcal{M}', P'), S' \models \varphi$ and (2) $P' \succcurlyeq P$.

By Theorem A.12, $\forall s \in S^{\mathcal{M}}$, if $\exists(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$ such that $(\mathcal{M}', P'), s \models \varphi$ and $P' \succcurlyeq P$ then $\exists(\mathcal{M}_{s,\varphi}, P_{s,\varphi}) \in \mathit{Modif}((\mathcal{M}, P), s, \varphi)$ such that $P' \succcurlyeq P_{s,\varphi}$. Therefore, by Theorem A.11, if $\exists(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$ such that $(\mathcal{M}', P'), S' \models \varphi$ and $P' \succcurlyeq P$, then $\exists(\mathcal{M}_{S',\varphi}, P_{S',\varphi}) \in \mathit{Modif}^*((\mathcal{M}, P), S', \varphi)$ such that $P' \succcurlyeq P_{S',\varphi}$. Therefore, by (1) and (2), $\exists(\mathcal{M}_{S',\varphi}, P_{S',\varphi}) \in \mathit{Modif}^*((\mathcal{M}, P), S', \varphi)$ such that $P' \succcurlyeq P_{S',\varphi}$. □

Hence, as stated in the following theorem, "φ is P-satisfiable" is equivalent to "$(\mathcal{M}, P)$ is φ-modifiable".

Theorem A.14 *(P-satisfiable $\approx$ φ-modifiable). Let $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$ and $s \in S^{\mathcal{M}}$. If $\varphi \in \Sigma$-XCTL, then φ is P-satisfiable at s iff $(\mathcal{M}, P)$ is φ-modifiable at s.*

Proof. ($\Rightarrow$). If φ is P-satisfiable at s, then there is $(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$ such that $(\mathcal{M}', P'), s \models \varphi$ and $P' \succcurlyeq P$. Therefore, by Theorem A.12, there exists $(\mathcal{M}_\varphi, P_\varphi) \in \mathit{Modif}((\mathcal{M}, P), s, \varphi)$ such that $P' \succcurlyeq P_\varphi$.

Thus, $\mathit{Modif}((\mathcal{M}, P), s, \varphi) \neq \emptyset$, and $(\mathcal{M}, P)$ is φ-modifiable at s.

($\Leftarrow$). If $(\mathcal{M}, P)$ is φ-modifiable at s, then $\mathit{Modif}((\mathcal{M}, P), s, \varphi) \neq \emptyset$. Let $(\mathcal{M}', P') \in \mathit{Modif}((\mathcal{M}, P), s, \varphi)$. By Theorem A.7, $(\mathcal{M}', P'), s \models \varphi$, and, by Theorem A.4, $P' \succcurlyeq P$. Therefore, φ is P-satisfiable at s. □

Finally, completeness of XUPD$_{prot}$ is a consequence of the next corollary.

Corollary A.15 *($\mathbf{K}_\Sigma$-satisfiable $\approx$ φ-modifiable). For all $\varphi \in \Sigma$-XCTL, $s \in S^\Sigma$, and $\mathcal{M} \in \mathbf{K}_\Sigma$: φ is $\mathbf{K}_\Sigma$-satisfiable at s iff $(\mathcal{M}, P_\perp)$ is φ-modifiable at s.*

Proof. ($\Rightarrow$). Let $\mathcal{M}' \in \mathbf{K}_\Sigma$ such that $\mathcal{M}', s \models \varphi$. Then, by Definition 2.6, $(\mathcal{M}', P_{\mathcal{M}'}), s \models \varphi$. Besides, $P_{\mathcal{M}'} \succcurlyeq P_\perp$. Therefore, φ is $P_\perp$-satisfiable at s and, by Theorem A.14, $(\mathcal{M}, P_\perp)$ is φ-modifiable at s.

($\Leftarrow$). If $(\mathcal{M}', P') \in \mathit{Modif}((\mathcal{M}, P_\perp), s, \varphi)$, then, by Theorem A.7, $(\mathcal{M}', P'), s \models \varphi$.

Besides, $P_{\mathcal{M}'} \succcurlyeq P'$. Hence, $(\mathcal{M}', P_{\mathcal{M}'}), s \models \varphi$ and, by Definition 2.6, $\mathcal{M}', s \models \varphi$. Therefore, φ is $\mathbf{K}_\Sigma$-satisfiable at s. □

Note that pseudo-code of XUPD$_{prot}$ is an intuitive implementation of *Modif*. If we assume that models computed by XUPD$_{prot}$ are exactly modifications of the input model, i.e., for all $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s \in S^{\mathcal{M}}$, and $\varphi \in \Sigma$-XCTL, XUPD$_{prot}[(\mathcal{M}, P), s, \varphi] = \mathit{Modif}((\mathcal{M}, P), s, \varphi)$, then the following theorem formalizes, in a sense, that XUPD$_{prot}$ is sound and complete.

Theorem A.16 *(XUPD$_{prot}$ is sound and complete). If for all $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s \in S^{\mathcal{M}}$, and $\varphi \in \Sigma$-XCTL, XUPD$_{prot}[(\mathcal{M}, P), s, \varphi] = \mathit{Modif}((\mathcal{M}, P), s, \varphi)$, then for all $\mathcal{M} \in \mathbf{K}_\Sigma$, $s \in S^{\mathcal{M}}$, and $\varphi \in \Sigma$-XCTL:*

1. *If $(\mathcal{M}', P') \in$ XUPD$_{prot}[(\mathcal{M}, P_\perp), s, \varphi]$, then $(\mathcal{M}', P'), s \models \varphi$.*
2. *If φ is $\mathbf{K}_\Sigma$-satisfiable at s, then XUPD$_{prot}[(\mathcal{M}, P_\perp), s, \varphi] \neq \emptyset$.*

Proof. (1). If $(\mathcal{M}', P') \in$ XUPD$_{prot}[(\mathcal{M}, P_\perp), s, \varphi] = \mathit{Modif}((\mathcal{M}, P), s, \varphi)$, then, by Theorem A.7, $(\mathcal{M}', P'), s \models \varphi$. Therefore, XUPD$_{prot}$ is sound.

(2). If φ is $\mathbf{K}_\Sigma$-satisfiable at s, then, by Corollary A.15, $(\mathcal{M}, P_\perp)$ is φ-modifiable at s. Therefore, XUPD$_{prot}[(\mathcal{M}, P_\perp), s, \varphi] = \mathit{Modif}((\mathcal{M}, P_\perp), s, \varphi) \neq \emptyset$. □

Appendix B. Complexity

This section offers a complexity analysis of our model-update method. We measure the complexity of UPD$_{prot}$ with respect to the size of the input formula and the size of the input model. For Σ-CTL formulas, we use the size defined in Section 2.

To define the size of a model in $\mathbf{KP}_\Sigma$, we note that it is common to consider that the states of a model are vectors whose components correspond to variable values. In such case, the number of states is exponential with respect to the number of variables and therefore the size of a model may be estimated by considering only $|S \times S|$ or just $|S|$. In general, however, it is possible that the number of variables is greater than the number of states. In addition, some parts of UPD$_{prot}$ depend not only on the states of input model but also on the model transitions. Therefore, we define the size of a model as follows.

Definition B.1 *(Size of protected models).* The *size* of a protected model $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$ is defined by $|(\mathcal{M}, P)| = \max\{|\mathrm{Lit}(V^\Sigma)|, |S^\Sigma \times S^\Sigma|\}$.

Note that all $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$ have the same size. Besides, update operations in the execution of XUPD$_{prot}((\mathcal{M}, P), s, \varphi)$ preserve Σ. Therefore, we use $m = |(\mathcal{M}, P)|$ as a fixed parameter in the next complexity analysis.

Definition B.2 *(Number of steps).* Let $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s \in S^{\mathcal{M}}$, and $\varphi \in \mathbf{CTL}_\Sigma$. If $m = |(\mathcal{M}, P)|$, $n = |\varphi|$, and $\varphi \in \mathbf{XCTL}_\Sigma$, we use $T_m(n)$ to denote the *number of steps* required in a nondeterministic execution of $\text{XUPD}_{prot}((\mathcal{M}, P), s, \varphi)$.

If $\varphi \in \mathbf{CTL}_\Sigma - \mathbf{XCTL}_\Sigma$, we use $T'_m(n)$ to denote the *number of steps* required in a nondeterministic execution of $\text{UPD}_{prot}((\mathcal{M}, P), s, \varphi)$.

We use $T_m(n)$ to analyze the worst-case time complexity of XUPD_{prot}.

Theorem B.3 *(Complexity of* XUPD_{prot}*).* *Let* $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s \in S^{\mathcal{M}}$, *and* $\varphi \in \mathbf{XCTL}_\Sigma$. *If* $m = |(\mathcal{M}, P)|$, *and* $n = |\varphi|$, *then* $T_m(n)$ *is* $\mathcal{O}(m^n)$.

Proof. We analyze $T_m(n)$ according to the structure of φ.

1. If $\varphi = \mathsf{F}$ or $\varphi = \mathsf{T}$, then, by lines (3–4), $\text{XUPD}_{prot}((\mathcal{M}, P), s, \varphi)$ fails or returns $(\mathcal{M}, P)$. In both cases the execution of $\text{XUPD}_{prot}((\mathcal{M}, P), s, \varphi)$ requires at most one step. Therefore, if $n = |\mathsf{F}| = |\mathsf{T}|$, $T_m(n) = 1$.
2. If $\varphi = \ell$, then, since $|L^P(s)| \leqslant |\text{Lit}(V^\Sigma)| \leqslant m$, computing the test $\overline{\ell} \in L^P(s)$ in line (5) requires at most m steps.
   Next, to compute $\mathbf{L}^u((\mathcal{M}, P), s, \ell)$ in line (6), we need to calculate $L^{\mathcal{M}}[s \oplus \ell]$ and $L^P[s \oplus \ell]$. Thus, to compute $L^{\mathcal{M}}[s \oplus \ell]$ we search for $L^{\mathcal{M}}(s)$ in $L^{\mathcal{M}}$, and we modify $L^{\mathcal{M}}(s)$ by adding ℓ to $L^{\mathcal{M}}(s)$ and removing $\overline{\ell}$ from $L^{\mathcal{M}}(s)$.
   The search for $L^{\mathcal{M}}(s)$ in $L^{\mathcal{M}}$ can be done in at most m steps because $|S^{\mathcal{M}}| \leqslant m$. The addition (without duplication) of ℓ, and the removal of $\overline{\ell}$, can be done in at most $2m$ steps because $|L^{\mathcal{M}}(s)| \leqslant |\text{Lit}(V^\Sigma)| \leqslant m$. Therefore, the computation of $L^{\mathcal{M}}[s \oplus \ell]$ requires at most $3m$ steps.
   Analogously, the computation of $L^P[s \oplus \ell]$ requires at most $3m$ steps. Hence, $\text{XUPD}_{prot}((\mathcal{M}, P), s, \ell)$ needs at most $m + 3m + 3m$ steps.
   Therefore, if $n = |\ell|$, $T_m(n) = 7m$.
3. If $\varphi = \alpha \vee \beta$, then, by lines (7–8), after guessing $\delta \in \{\alpha, \beta\}$ in one step, $\text{XUPD}_{prot}((\mathcal{M}, P), s, \alpha \vee \beta)$ makes a recursive call to
   $\text{XUPD}_{prot}((\mathcal{M}, P), s, \delta)$.
   Therefore, if δ is the guess that requires more execution steps, the number of steps required by $\text{XUPD}_{prot}((\mathcal{M}, P), s, \alpha \vee \beta)$ is 1 plus the number of steps needed by $\text{XUPD}_{prot}((\mathcal{M}, P), s, \delta)$.
   Thus, if $n = |\alpha \vee \beta|$, $T_m(n) = 1 + T_m(n-1)$.
4. If $\varphi = \alpha \wedge \beta$, then lines (9–10) amount to compute
   $\text{XUPD}_{prot}((\mathcal{M}', P'), s, \beta)$
   after guessing $(\mathcal{M}', P') \in \text{XUPD}_{prot}((\mathcal{M}, P), s, \alpha)$.
   Therefore, the number of steps required to compute
   $\text{XUPD}_{prot}((\mathcal{M}, P), s, \alpha \wedge \beta)$ is given by the sum of: $T_m(n-1)$ steps for computing $\text{XUPD}_{prot}((\mathcal{M}, P), s, \alpha)$, 1 step for guessing $(\mathcal{M}', P') \in \text{XUPD}_{prot}((\mathcal{M}, P), s, \alpha)$, and $T_m(n-1)$ steps for computing
   $\text{XUPD}_{prot}((\mathcal{M}', P'), s, \beta)$.
   Thus, if $n = |\alpha \wedge \beta|$, $T_m(n) = 1 + 2T_m(n-1)$.
5. If $\varphi = \mathbf{EX}\alpha$, then line (11) guesses $s' \in A^P[s]$ in one step.
   Now, to compute $\mathbf{T}^+_\exists((\mathcal{M}, P), s, s')$ in line (12), we have to add (s, s') to $R^{\mathcal{M}}$ and protect (s, s') in E^P.
   Line (12) adds transition (s, s') to $(\mathcal{M}, P)$ in $2m$ steps and makes a recursive call to $\text{XUPD}_{prot}((\mathcal{M}'', P''), s', \alpha)$.
   Therefore, the number of steps required to compute
   $\text{XUPD}_{prot}((\mathcal{M}, P), s, \mathbf{EX}\alpha)$ is 1 plus the sum of $2m$ and the number of steps needed to compute $\text{XUPD}_{prot}((\mathcal{M}'', P''), s', \alpha)$.
   Thus, if $n = |\mathbf{EX}\alpha|$, $T_m(n) = 1 + 2m + T_m(n-1)$.
6. If $\varphi = \mathbf{AX}\alpha$, then line (13) guesses $S' \in \{X \subseteq A^P[s] \mid E^P[s] \subseteq X \text{ and } X \neq \emptyset\}$ in one step.
   Now, to compute $\mathbf{T}^u_\forall((\mathcal{M}, P), s, S')$ in line (14), we need to calculate $\mathbf{T}^u(\mathcal{M}, s, S')$ and $(A^P - s \times (A^P[s] - S'))$.
   The computation of $\mathbf{T}^u(\mathcal{M}, s, S')$ requires at most $3m$ steps: m steps to compute $R^{\mathcal{M}}[s]$; m steps to remove $s \times R^{\mathcal{M}}[s]$ from $R^{\mathcal{M}}$; and m steps to add $s \times S'$ to $R^{\mathcal{M}}$.
   The computation of $(A^P - s \times (A^P[s] - S'))$ requires at most $3m$ steps: m to compute s in $A^P[s]$; m to compute $A^P[s] - S'$; and m to remove $s \times (A^P[s] - S')$ from A^P.
   Therefore, the computation of $\mathbf{T}^u_\forall((\mathcal{M}, P), s, S')$ in line (14) requires at most $6m$ steps.
   Then, the call to $\text{XUPD}^*_{prot}(\mathbf{T}^u_\forall((\mathcal{M}, P), s, S'), S', \alpha)$ in line (14) corresponds to $|S'|$ consecutive calls to $\text{XUPD}_{prot}(\_, s', \alpha)$ with all $s' \in S'$. Since $|S'| \leqslant m$ and $|\alpha| = n - 1$, the maximum number of steps required by all these calls to XUPD_{prot} is $mT_m(n-1)$.
   Thus, if $n = |\mathbf{AX}\alpha|$, $T_m(n) = 1 + 6m + mT_m(n-1)$.
7. If $\varphi = \mathbf{OD}_\odot j$, where $\odot \in \{\leqslant, >\}$, then line (15) guesses S' in one step. Analogously to the case $\varphi = \mathbf{AX}\alpha$, the computation of $\mathbf{T}^u_Q((\mathcal{M}, P), s, S')$ in line (17) requires at most $6m$ steps.
   Therefore, if $n = |\mathbf{OD}_\odot j|$, $T_m(n) = 1 + 6m$.

The worst case of $\text{XUPD}_{prot}((\mathcal{M}, P), s, \varphi)$ is when $\varphi = \mathbf{AX}\alpha$. Therefore, a lower bound for the worst-case complexity of XUPD_{prot} is given by a solution of the recurrence relation defined by $T_m(n) = 1 + 6m + mT_m(n-1)$, if $n > 1$, and $T_m(1) = 7m$.

The factor m in $T_m(1)$ and the term $mT_m(n-1)$ in $T_m(n)$ imply that a solution of this recurrence relation has a factor m^n. Therefore, the worst-case time complexity of XUPD$_{prot}$ is $\mathcal{O}(m^n)$. □

The above theorem can be extended to formulas in $\mathbf{CTL}_\Sigma$.

Theorem B.4 *(Complexity of* UPD$_{prot}$*). Let* $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s \in S^{\mathcal{M}}$, *and* $\varphi \in \mathbf{CTL}_\Sigma - \mathbf{XCTL}_\Sigma$. *If* $m = |(\mathcal{M}, P)|$, *and* $n = |\varphi|$, *then* $T'_m(n)$ *is* $\mathcal{O}(m^{n+1})$.

Proof. Let ψ be a subformula of φ such that $\psi \in \mathbf{CTL}$ has the form $\mathbf{E}[\alpha\,\mathbf{U}\,\beta]$ or $\mathbf{A}[\alpha\,\mathbf{U}\,\beta]$ or $\mathbf{E}[\alpha\,\mathbf{R}\,\beta]$ or $\mathbf{A}[\alpha\,\mathbf{R}\,\beta]$.

The evaluation of UPD$_{prot}((\mathcal{M}, P), s, \psi)$ is done replacing ψ by the corresponding fixed-point formula ψ_{fix}. On the one hand, the size of ψ_{fix} is linear w.r.t. the size of ψ, $|\psi_{fix}| = c*|\psi|$ for some $c \in \mathbb{N}$. On the other hand, the application of UPD$_{prot}$ to ψ_{fix} implies the recursive application of XUPD$_{prot}$ to ψ at a state s', where s' is a successor of s (obtained from $R^{\mathcal{M}}$ with cost m). In each recursive call to XUPD$_{prot}$ the pair (ψ, s') is added to a list of visited states with the formula ψ and XUPD$_{prot}$ continues in this manner provided that the pair (ψ, s') does not represent a visited state with ψ. Thus, the maximum number of recursive calls to XUPD$_{prot}$ is bounded by m. Therefore, by Theorem B.3, $T'_m(n) \leqslant m*(m+T_m(n)) \leqslant m^2+m*d*m^n$, for some $d \in \mathbb{N}$. Hence, $T'_m(n) \leqslant k*m^{n+1}$, for some $k \in \mathbb{N}$, and $T'_m(n)$ is $\mathcal{O}(m^{n+1})$. □

References

[1] F. Buccafurri, T. Eiter, G. Gottlob, N. Leone, Enhancing model checking in verification by AI techniques, Artif. Intell. 112 (1999) 57–104.
[2] F. Buccafurri, T. Eiter, G. Gottlob, N. Leone, On ACTL formulas having linear counterexamples, J. Comput. Syst. Sci. 62 (2001) 463–515.
[3] L. Calzone, N. Chabrier-Rivier, F. Fages, S. Soliman, Machine learning biochemical networks from temporal logic properties, in: C. Priami, G.D. Plotkin (Eds.), Transactions on Computational Systems Biology, in: Lecture Notes in Computer Science, vol. 4220, Springer, 2006, pp. 68–94.
[4] M. Carrillo, D.A. Rosenblueth, A method for CTL model update, representing Kripke structures as "table systems", Int. J. Pure Appl. Math. 52 (2009) 401–431.
[5] M. Carrillo, D.A. Rosenblueth, Nondeterministic update of CTL models by preserving satisfaction through protections, in: T. Bultan, P.A. Hsiung (Eds.), Automated Technology for Verification and Analysis (ATVA 2011), in: Lecture Notes in Computer Science, vol. 6996, Springer, Taipei, Taiwan, 2011, pp. 60–74.
[6] N. Chabrier-Rivier, M. Chiaverini, V. Danos, F. Fages, V. Schächter, Modeling and querying biomolecular interaction networks, Theor. Comput. Sci. 325 (2004) 25–44.
[7] G. Chatzieleftheriou, B. Bonakdarpour, S.A. Smolka, P. Katsaros, Abstract model repair, in: A.E. Goodloe, S. Person (Eds.), Proc. 4th NASA Formal Methods (NFM 2012), in: Lecture Notes in Computer Science, vol. 7226, Springer, 2012, pp. 341–355.
[8] A. Cimatti, E.M. Clarke, F. Giunchiglia, M. Roveri, NuSMV: a new Symbolic Model Verifier, in: N. Halbwachs, D. Peled (Eds.), Proc. Eleventh Conference on Computer-Aided Verification (CAV '99), in: Lecture Notes in Computer Science, vol. 1633, Springer, 1999, pp. 495–499.
[9] E. Clarke, S. Jha, Y. Lu, H. Veith, Tree-like counterexamples in model checking, in: 17th IEEE Symposium on Logic in Computer Science (LICS 2002), Springer, 2002, pp. 19–29.
[10] E.M. Clarke, O. Grumberg, D.E. Long, Verification tools for finite-state concurrent systems, in: A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium, Springer-Verlag, London, UK, 1994, pp. 124–175.
[11] E.M. Clarke, O. Grumberg, D.A. Peled, Model Checking, The MIT Press, 1999.
[12] Y. Ding, Model update for system modifications, Ph.D. thesis, School of Computing and Mathematics, University of Western Sydney, Australia, 2007.
[13] Y. Ding, D. Hemer, An optimised algorithm to tackle the model explosion problem in CTL model update, in: B.T. Zhang, M.A. Orgun (Eds.), 11th Pacific Rim International Conference on Artificial Intelligence (PRICAI 2010), in: Lecture Notes in Artificial Intelligence, vol. 6230, Springer, 2010, pp. 589–594.
[14] T. Eiter, E. Erdem, M. Fink, J. Senko, Updating action domain descriptions, Artif. Intell. 174 (2010) 1172–1221.
[15] E.A. Emerson, E.M. Clarke, Using branching time temporal logic to synthesize synchronization skeletons, Sci. Comput. Program. 2 (1982) 241–266.
[16] E.A. Emerson, J.Y. Halpern, Decision procedures and expressiveness in the temporal logic of branching time, J. Comput. Syst. Sci. 30 (1985) 1–24.
[17] P.T. Guerra, R. Wassermann, Revision of CTL models, in: Ibero-American Conference on Artificial Intelligence (IBERAMIA 2010), in: Lecture Notes in Artificial Intelligence, vol. 6433, Springer, 2010, pp. 153–162.
[18] B. Jobstmann, S. Staber, A. Griesmayer, R. Bloem, Finding and fixing faults, J. Comput. Syst. Sci. 78 (2012) 441–460.
[19] M. Kelly, F. Pu, Y. Zhang, Y. Zhou, ACTL local model update with constraints, in: Proc. 14th Knowledge-Based and Intelligent Information and Engineering Systems, Part IV (KES 2010), in: Lecture Notes in Artificial Intelligence, vol. 6279, Springer, 2010, pp. 135–144.
[20] M. Kelly, Y. Zhang, Local model update with an application to sliding window protocol, in: Proc. 14th Knowledge-Based and Intelligent Information and Engineering Systems, Part IV (KES 2010), in: Lecture Notes in Artificial Intelligence, vol. 6279, Springer, 2010, pp. 11–21.
[21] J.M. Wing, M. Vaziri-Farahani, A case study in model checking software, Sci. Comput. Program. 28 (1997) 273–299.
[22] Y. Zhang, Y. Ding, CTL model update for system modifications, J. Artif. Intell. Res. 31 (2008) 113–155.