International Journal of Pure and Applied Mathematics
Volume 52 No. 3 2009, 401-431

A METHOD FOR CTL MODEL UPDATE, REPRESENTING KRIPKE STRUCTURES AS "TABLE SYSTEMS"

Miguel Carrillo[1], David A. Rosenblueth[2] §
[1,2]Institute for Research in Applied Mathematics and Systems
National Autonomous University of Mexico
Apdo. 20-726, México D.F., 01000, MEXICO
[1]e-mail: mcarrillob@uxmcc2.iimas.unam.mx
[2]e-mail: drosenbl@servidor.unam.mx

Abstract: A model checker determines whether or not a Kripke structure satisfies a given temporal-logic formula. A model updater, by contrast, modifies a Kripke structure in such a way that a given temporal-logic formula holds, if possible. An important component of a model *checker* (such as NuSMV or SPIN) is a language allowing the concise specification of large Kripke structures. This suggests that any practical method for model *update* should employ such languages as well. We study a computation-tree logic (CTL) model updater employing "table systems," a fragment of the structure-specification language of NuSMV. Such a fragment allows our model-updater to produce structures having a similar specification to that of the structure given by the user. Our updater extends an existing, state-by-state method for CTL model update by adding/removing additional transitions than those prescribed by the original method.

AMS Subject Classification: 68T27, 68T05, 68Q60
Key Words: model checking, computation-tree logic (CTL), model update, knowledge revision/update

1. Introduction

Model checking [6, 7] is a collection of tools for establishing whether or not

Received: March 20, 2009

§Correspondence author

a transition system with an ongoing interaction with the environment has a desired property. A model checker typically views a system as a Kripke structure and a property as a temporal-logic formula. In case the Kripke structure does not satisfy the formula, the model checker normally produces a counterexample. Such a counterexample is sometimes helpful in determining how to *manually* modify the Kripke structure so as to satisfy the violated formula. In an attempt to *mechanize* the modification of faulty Kripke structures, a number of algorithms repairing Kripke structures for computation-tree logic (CTL) have recently appeared in the literature [1, 2, 3, 10]. State-by-state update methods [1, 3, 10], explicitly representing each state and transition of a Kripke structure, are important as a first step for studying the update problem. Practical update methods, however, would probably employ a language for succinctly specifying Kripke structures, as standard model checkers do. Our purpose will be to extend an existing state-by-state method [3] for CTL model update to "table systems," a fragment of the language employed by the NuSMV model checker.

An important component of a model checker is a language for concisely describing large Kripke structures. Examples are the languages used by the NuSMV and SPIN model checkers. These languages provide numerous shorthands bridging human capabilities with the specification of large transition systems for Kripke structures. The fact that model checkers employ these languages suggests that practical model updaters should also employ similar techniques.

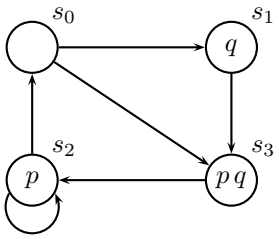
A model updater takes a faulty Kripke structure, which does not satisfy a particular formula, and modifies the given structure in such a way that the new structure does satisfy such a formula, if possible [10]. Hence, both the input and the output are Kripke structures. A problem appears, however, when a model updater employing a language for concisely describing Kripke structures allows arbitrary changes to the faulty structure: such arbitrary changes might destroy the succinctness of the specification given by the user. The reason is that a change to the state-by-state representation of the structure might not correspond to any succinct representation of the structure in the structure-specification language.

This loss of succinctness, in turn, has disadvantages. On the one hand, if model update is viewed as a learning problem [9, pp. 649–653, 675], then applying Ockham's razor will make us prefer concise specifications of Kripke structures. In such a view, a Kripke-structure specification corresponds to a hypothesis, and the formula corresponds to the set of examples to be covered

by the hypothesis. A concise specification of a Kripke structure (in a structure-specification language) corresponds then to a simple hypothesis. Hence, a model updater considering all Kripke structures regardless of the succinctness of their specification would not necessarily conform with Ockham's razor.

On the other hand, a large specification of a Kripke structure resulting from a model-update system could be unreadable by a person, because of demanding an excessive amount of time to understand such a structure. This situation is similar to that of CTL model checking when instead of a confirmation or a counterexample, we wish to know all the states in which a formula is satisfied. Such a set of states could in theory be easily obtained from a state-labelling algorithm [4] for model checking. When the set of states is large, however, it is hard to imagine how to display such a set in a meaningful (perhaps concise) manner, as attested by a designer of NuSMV [8].

These drawbacks suggest that a model updater should modify the *specification* of the input Kripke structure provided by the user rather than its state-by-state representation. It will be convenient, however, to employ a simplified version of one of these languages (that used by NuSMV). Consider the following (nondeterministic) Kripke structure $\mathcal{M}_0$ with four states, $s_0, \ldots, s_3$, and two propositional variables, p and q.



As is usual in the model-checking literature, we only show the positive literals labelling each state. The accessibility relation would be:

p	q	p'	q'
0	0	$*$	1
0	1	1	1
1	0	$*$	0
1	1	1	0

We use 0, 1, and $*$ when a propositional variable is false, true, and nondeterministically either false or true, respectively. Note that if q is false in a state, then regardless of the value of p in that state, p is nondeterministically either

false or true in the next state. Likewise, if q is true in a state, then p is also true in the next state. At the same time, there is a similar redundancy for the behavior of q: If p is false in a state, then regardless of the value of q in that state, q is true in the next state. Likewise, if p is true in a state, then q is false in the next state. This allows us to represent this Kripke structure as the following two smaller tables:

q	p'
0	$*$
otherwise	1

p	q'
0	1
otherwise	0

where the absent variable on the left-hand side of a table takes either truth value. In general, we will view the accessibility relation as a set of Boolean functions f_i, one for each propositional variable v_i, which defines the next value of v_i in terms of the current values of all the variables. To capture nondeterminism, the codomain of each function f_i is $2^{\{0,1\}}$ (the use of "otherwise" here might seem exaggerated in this simple example, but occurs here because it is part of our definition of table and is clearly useful in tables with more variables).

Like NuSMV's structure-specification language, table systems define the next value of a variable in terms of the current values of variables. In such a language, however, the user can also define the next value of a variable in terms of the next values of variables. An important simplification in table systems will be to ignore this last feature of NuSMV.

Instead of starting from the model-update problem afresh, we will build upon a simplified version of an existing state-by-state CTL model updater [3]. The simplification is twofold. On the one hand, the state-by-state algorithm in [3] can alter a faulty Kripke structure with the addition/removal of transitions and labels of states, as well as the addition of states. We will consider instead that a state is determined by its labels, i.e., if $L(s)$ is the set of variables labelling a state s, then for all states s and s', $L(s) = L(s')$ implies $s = s'$. Hence, we omit the addition of states. In doing so, we deviate from the standard logical approach of Kripke semantics, and follow symbolic model checkers instead.

On the other hand, [3] gives two versions of the update with respect to the operator **AX**: a coarse version, convenient for explanation purposes, and a finer one, following Zhang and Ding's Kripke structure ordering [10]. For simplicity and brevity, we have preferred to use the coarser version of **AX**, especially because the finer version could be easily incorporated.

The rest of this paper is structured as follows. After establishing the notation and some definitions in Section 2, we summarize the original state-by-state

method in Section 3. We define table systems in Section 4 and give the updater for such systems in Section 5. An example in Section 6 illustrates this method. We cover related work in Section 7 and conclude in Section 8.

2. Technical Preliminaries

This section gives some introductory definitions and fixes the notation and terminology. We assume some familiarity with CTL model checking and refer the reader to [6, 7] for more thorough treatments.

We slightly deviate from other approaches in the following points. It is usual to label a state with positive literals only, containing exactly all atoms which are true in that state. For us, however, it will be more convenient to redundantly label a state both with positive and negative literals.

In addition, we restrict our update method to CTL formulas in "negation normal form," where negation is only applied to propositional variables (i.e., atomic formulas), and view negation applied to nonatomic formulas as a shorthand.

2.1. Atoms, States, and Terms

We assume that $Atoms$ is a fixed, nonempty, finite set of n atoms (or propositional variables), $Atoms = \{v_1, v_2, \ldots, v_n\}$ (henceforth, we assume that n is fixed). The elements of the set $Atoms$ represent observable Boolean variables of a given system. We use v, p, q, and r to denote arbitrary atoms. The variables of $Atoms$ range over the set $Boolean = \{0, 1\}$.

We define a *state* s as a tuple of n Boolean values, $s = (s_1, s_2, \ldots, s_n)$, where each component s_i represents a value of the variable v_i. The set of states is $States = Boolean^n$.

A *truth assignment* for $Atoms$ is a function $\sigma : Atoms \to Boolean$ that assigns a Boolean value to every atom, and we use $Boolean^{Atoms}$ to denote the set of truth assignments for $Atoms$.

There is an evident bijective correspondence between $States$ and truth assignments. The truth assignment corresponding to a state s is determined by the function: $A : States \to Boolean^{Atoms}$; $A(s)(v_i) = s_i$.

We collect atoms and negated atoms into a set of *literals* denoted by $Lit = Atoms \cup \{\neg v \mid v \in Atoms\}$, and we use ℓ to denote an arbitrary literal. The

complement of a literal ℓ, is defined by $\bar{\ell} = \neg\ell$ if $\ell \in Atoms$, and by $\bar{\ell} = v$ if $\ell = \neg v$.

A *consistent set* of literals is a set that, for every literal ℓ, includes exactly one of ℓ or $\bar{\ell}$. The class of *consistent sets* of literals is: $\mathcal{P}^+(Atoms) = \{X \subseteq Lit \mid (\forall \ell \in X : \bar{\ell} \notin X)$ and $(\forall p \in Atoms : (p \in X$ or $\bar{p} \in X))\}$.

If $t \subseteq Lit$ we say that t is a *propositional term*. We identify a term t with the conjunction over t, and we adopt the convention that $\bigwedge \emptyset = \top$, where $\top$ denotes the Boolean constant 1. We use *Terms* for the set of propositional terms.

We say that a state s *satisfies* an atom p, $s \models p$, iff the truth assignment corresponding to s makes p true, i.e., $A(s)(p) = 1$. Analogously, s *satisfies* the literal $\neg p$, $s \models \neg p$, iff $A(s)(p) = 0$. Finally, a state s satisfies a term t, $s \models t$, iff $\forall \ell \in t : s \models \ell$.

We associate with each state s a propositional term, $L(s)$, defined by $L(s) = \{\ell \in Lit \mid s \models \ell\}$. Let us remark that $L(s)$ is a consistent set of literals.

2.2. Temporal Logic

We use the temporal logic CTL [6] with negation normal form (NNF), which is a formulation that limits the application of negation to atoms. **CTL** formulas (abbreviated Φ) have the following BNF syntax:

$$\Phi ::= \top \mid \bot \mid p \mid (\neg p) \mid (\Phi \vee \Phi) \mid (\Phi \wedge \Phi) \mid (\mathbf{EX}\,\Phi) \mid (\mathbf{AX}\,\Phi) \mid \mathbf{E}[\Phi\,\mathbf{U}\,\Phi] \mid \mathbf{A}[\Phi\,\mathbf{U}\,\Phi] \mid \mathbf{E}[\Phi\,\mathbf{R}\,\Phi] \mid \mathbf{A}[\Phi\,\mathbf{R}\,\Phi]\,.$$

The semantics of **CTL** resorts to Kripke structures. A *Kripke structure* $\mathcal{M} = \langle S^{\mathcal{M}}, R^{\mathcal{M}}, L^{\mathcal{M}} \rangle$ consists of: a nonempty finite set of *states*, $S^{\mathcal{M}}$; a total *accessibility relation*, $R^{\mathcal{M}} \subseteq S^{\mathcal{M}} \times S^{\mathcal{M}}$ (i.e., for every $s \in S^{\mathcal{M}}$ there exists $t \in S^{\mathcal{M}}$ such that $(s,t) \in R^{\mathcal{M}}$); and a *labelling function*, $L^{\mathcal{M}} : S \to P^+(Atoms)$. We use K_{States} for the class of Kripke structures $\mathcal{M}$ such that $S^{\mathcal{M}} = States$.

A *path* of a structure $\mathcal{M}$ is an infinite sequence $s_0, s_1, \ldots$ of states $s_i \in S$, such that for all $i \in \mathbb{N}$, $(s_i, s_{i+1}) \in R^{\mathcal{M}}$.

The intuition for the **CTL** syntax is as follows: the letters **E** and **A** denote "there *Exists* a path" and "for *All* paths," respectively; the letters **X**, **U**, and **R**, in turn, abbreviate "*neXt* state," "*Until*," and "*Release*," respectively.

Throughout, unless otherwise noted, possibly subscripted $\mathcal{M}$ denotes a Kripke structure, S a set of states, R an accessibility relation, L a labelling function, and s a state. In addition, Greek letters denote formulas.

Given a **CTL** formula φ, $\mathcal{M} \in K_{States}$, and $s_0 \in States$, we define the satisfiability notion for **CTL** using structural induction on φ. We say that $\mathcal{M}$ *satisfies* φ at state s_0, which we write as $\mathcal{M} \models_{s_0} \varphi$:

1. $\mathcal{M} \models_{s_0} \top$.
2. $\mathcal{M} \not\models_{s_0} \bot$.
3. $\mathcal{M} \models_{s_0} \ell$ iff $\ell \in L(s_0)$.
4. $\mathcal{M} \models_{s_0} \alpha \vee \beta$ iff $\mathcal{M} \models_{s_0} \alpha$ or $\mathcal{M} \models_{s_0} \beta$.
5. $\mathcal{M} \models_{s_0} \alpha \wedge \beta$ iff $\mathcal{M} \models_{s_0} \alpha$ and $\mathcal{M} \models_{s_0} \beta$.
6. $\mathcal{M} \models_{s_0} \mathbf{EX}\,\alpha$ iff there exists a path $s_0, s_1, \ldots$, such that $\mathcal{M} \models_{s_1} \alpha$.
7. $\mathcal{M} \models_{s_0} \mathbf{AX}\,\alpha$ iff for all paths $s_0, s_1, \ldots$, we have $\mathcal{M} \models_{s_1} \alpha$.
8. $\mathcal{M} \models_{s_0} \mathbf{E}[\alpha\,\mathbf{U}\,\beta]$ iff there exist a path $s_0, s_1, \ldots$, and a $j \geq 0$ such that $\mathcal{M} \models_{s_j} \beta$ and for all $i < j$, we have $\mathcal{M} \models_{s_i} \alpha$.
9. $\mathcal{M} \models_{s_0} \mathbf{A}[\alpha\,\mathbf{U}\,\beta]$ iff for all paths $s_0, s_1, \ldots$, there exists a $j \geq 0$ such that $\mathcal{M} \models_{s_j} \beta$ and for all $i < j$, we have $\mathcal{M} \models_{s_i} \alpha$.
10. $\mathcal{M} \models_{s_0} \mathbf{E}[\alpha\,\mathbf{R}\,\beta]$ iff there exists a path $s_0, s_1, \ldots$, such that either there is some $j \geq 0$ such that $\mathcal{M} \models_{s_j} \alpha$ and for all $0 \leq i \leq j$ we have $\mathcal{M} \models_{s_i} \beta$, or for all $k \geq 0$, we have $\mathcal{M} \models_{s_k} \beta$.
11. $\mathcal{M} \models_{s_0} \mathbf{A}[\alpha\,\mathbf{R}\,\beta]$ iff for all paths $s_0, s_1, \ldots$, either there is some $j \geq 0$ such that $\mathcal{M} \models_{s_j} \alpha$ and for all $0 \leq i \leq j$ we have $\mathcal{M} \models_{s_i} \beta$, or for all $k \geq 0$, we have $\mathcal{M} \models_{s_k} \beta$.

We single out the *next-fragment* of **CTL**, denoted by **XCTL**, that excludes all the temporal operators except **EX** and **AX**. The following *fixed-point equivalences* [6, p. 63], combined with the fragment **XCTL**, are instrumental in our update methods:

$$\mathbf{E}[\alpha\,\mathbf{U}\,\beta] \equiv \beta \vee (\alpha \wedge \mathbf{EX}\ \mathbf{E}[\alpha\,\mathbf{U}\,\beta]) \tag{1}$$

$$\mathbf{A}[\alpha\,\mathbf{U}\,\beta] \equiv \beta \vee (\alpha \wedge \mathbf{AX}\ \mathbf{A}[\alpha\,\mathbf{U}\,\beta]) \tag{2}$$

$$\mathbf{E}[\alpha\,\mathbf{R}\,\beta] \equiv \beta \wedge (\alpha \vee \mathbf{EX}\ \mathbf{E}[\alpha\,\mathbf{R}\,\beta]) \tag{3}$$

$$\mathbf{A}[\alpha\,\mathbf{R}\,\beta] \equiv \beta \wedge (\alpha \vee \mathbf{AX}\ \mathbf{A}[\alpha\,\mathbf{R}\,\beta]) \tag{4}$$

The following shorthands allow using known temporal operators not included in the definition above:

1. negation:

$$\neg\top \equiv \bot$$
$$\neg\bot \equiv \top$$
$$\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$$

$$\begin{aligned}
\neg(\alpha \wedge \beta) &\equiv \neg\alpha \vee \neg\beta \\
\neg\mathbf{EX}\,\alpha &\equiv \mathbf{AX}\,\neg\alpha \\
\neg\mathbf{AX}\,\alpha &\equiv \mathbf{EX}\,\neg\alpha \\
\neg\,\mathbf{E}[\alpha\,\mathbf{U}\,\beta] &\equiv \mathbf{A}[\neg\alpha\,\mathbf{R}\,\neg\beta] \\
\neg\,\mathbf{A}[\alpha\,\mathbf{U}\,\beta] &\equiv \mathbf{E}[\neg\alpha\,\mathbf{R}\,\neg\beta] \\
\neg\,\mathbf{E}[\alpha\,\mathbf{R}\,\beta] &\equiv \mathbf{A}[\neg\alpha\,\mathbf{U}\,\neg\beta] \\
\neg\,\mathbf{A}[\alpha\,\mathbf{R}\,\beta] &\equiv \mathbf{E}[\neg\alpha\,\mathbf{U}\,\neg\beta]
\end{aligned}$$

2. "either now or in the future" (**F**), and "always" or "globally" (**G**):

$$\begin{aligned}
\mathbf{EF}\,\alpha &\equiv \mathbf{E}[\top\,\mathbf{U}\,\alpha] \\
\mathbf{AF}\,\alpha &\equiv \mathbf{A}[\top\,\mathbf{U}\,\alpha] \\
\mathbf{EG}\,\alpha &\equiv \mathbf{E}[\bot\,\mathbf{R}\,\alpha] \\
\mathbf{AG}\,\alpha &\equiv \mathbf{A}[\bot\,\mathbf{R}\,\alpha]
\end{aligned}$$

In the structure $\mathcal{M}_0$ above, for example, $\mathcal{M}_0 \models_{s_0} \mathbf{EX}\,q$ and $\mathcal{M}_0 \models_{s_0} \mathbf{EF}\,p$ hold, but $\mathcal{M}_0 \models_{s_0} p$ does not. In addition, $\mathcal{M}_0 \models_{s_1} \neg\mathbf{EG}\,p$ and $\mathcal{M}_0 \models_{s_0} \mathbf{AF}\,(p \wedge q)$ hold, but $\mathcal{M}_0 \models_{s_0} \mathbf{AG}\,p$ does not.

3. A State-by-State Update Method

This section gives a short account of our state-by-state method for **CTL** model update. For a more detailed description of this method we refer the reader to [3].

As a first approximation to this method, take for example the formula $\varphi = \mathbf{EX}\,p$. If φ does not hold at state s_0, then p does not hold at any successor of s_0. Thus, one way of making φ hold at s_0 would be by *adding* a transition going out from s_0 to a state where p holds.

Conversely, if $\varphi' = \mathbf{AX}\,p$ does not hold at s_0, then p does not hold at all the successors of s_0. Hence, one way of making φ' hold at s_0 would be by *removing* transitions from s_0 to states where p does not hold.

We extend this basic idea to **CTL** as follows. First, we preserve the truth value of a subformula via a "protection," interpreted as constraints on modifications to a structure. Protections allow the treatment of conjunctions by first making the leftmost conjunct hold, and then making the rest of the conjunction hold while preserving the truth value of the already-treated conjunct. Associated with each computed structure there is a protection.

Second, formulas are assumed to be in NNF. NNF facilitates the treatment of negation, as the protection of negated literals is simply interpreted as reversing the interpretation of positive-literal protection. The treatment of negation applied to other subformulas would involve computing the set difference w.r.t. the set of all structures making a subformula true (as happens in the treatment of negation in the state-labelling algorithms for **CTL** model checking [4, 7]). The vast number of such structures would render the computation of this set difference intolerably inefficient.

Finally, the semantic construction of the **CTL** temporal operators is made about **EX** and **AX** as *primitive* operators, using fixed-point equivalences [6] (1)–(4) for the rest of the operators. This semantic construction results in only two primitive temporal operators (in contrast with the state-labelling algorithms for **CTL** model checking [7], which typically employ three), but involves a cycle-detection mechanism. Such a mechanism avoids nonterminating cycles with a test for membership of a (state, formula)-pair in the set of already-visited such pairs.

The method in [3] modifies Kripke structures with the addition/removal of labels and transitions, as well as the addition of states. For simplicity, however, we will take here a variant of [3] assuming that a state is determined by its labels, i.e., for all states s and s', $L(s) = L(s')$ implies $s = s'$. Hence, we will modify Kripke structures only with the addition/removal of transitions. In doing so, we deviate from the standard logical approach of Kripke semantics, and follow symbolic model checkers instead.

Similarly, for conciseness we will use here the simpler of two versions of the treatment of the **AX** operator.

We will give the topmost level of our state-by-state updater restricted to **XCTL**. Thus, we will name the main function $Update^X$. Note that in general there will be multiple (minimal) ways of updating a given structure. Hence, $Update^X$ will have as value a *set* of pairs, each of which consists of a structure and its associated protection.

Suppose we are given a structure $\mathcal{M}$, its associated protection t, a state s_0, and a formula φ. We determine the set of updated structures and their associated protections by a case analysis. If φ is:

— $\top$: then $\mathcal{M}$ already satisfies φ, and we return the set consisting of the pair $(\mathcal{M}, t)$.

— $\bot$: then it is not possible to satisfy φ, and we return the empty set.

— ℓ: then if ℓ labels s_0, then $\mathcal{M}$ already satisfies φ; otherwise it is not

possible to satisfy φ.

— $\alpha \vee \beta$: then all structures satisfying either α or β also satisfy φ.

— $\alpha \wedge \beta$: then we first compute the structures satisfying α and then update each such resulting structure w.r.t. β (constrained by the protection of α).

— $\mathbf{EX}\,\alpha$: then we first add (if necessary) a transition from s_0 to any state s, and then update s w.r.t. α, if possible (i.e., this addition is constrained by the given protection t).

— $\mathbf{AX}\,\alpha$: then we first remove the transitions from s_0 to all nonprotected successors of s_0, if possible (i.e., this removal is constrained by the given protection t). If there were no protected successors (i.e., after the removal s_0 has no successors), then we make any state in which α can be made true a successor of s_0. Otherwise, we update the structure in all protected successors w.r.t. α.

We are now in a position to give more details about protections. First, note that we must be able to refer to individual transitions. The reason is that the formulas of the form $\mathbf{EX}\,\alpha$ or $\mathbf{AX}\,\alpha$ hold at a state s_0 because of α holding at successors s_i of s_0. Hence, we need a way of representing all structures having at least the transitions (s_0, s_i) (equivalently, we can think of the transitions (s_0, s_i) as being protected in subsequent updates). Consequently, a protection must consist of (at least) a set of transitions: $R^t \subseteq S^{\mathcal{M}} \times S^{\mathcal{M}}$.

Next, observe that whereas $\mathbf{EX}\,\alpha$ holds in s_0 if α holds in any successor of s_0, for $\mathbf{AX}\,\alpha$ to hold in s_0, α must hold in all successors of s_0. Therefore, further additions of successors to s_0 must be constrained so that the truth value of $\mathbf{AX}\,\alpha$ at s_0 is preserved. We do so by adding a second component to protections, registering the fact that $\mathbf{AX}\,\alpha$ must hold at s_0: $S^t \subseteq S^{\mathcal{M}} \times \mathbf{XCTL}$.

We say that t is an **XCTL** *protection* (or *term*) iff $t = \langle S^t, R^t \rangle$, where $S^t \subseteq S^{\mathcal{M}} \times \mathbf{XCTL}$ and $R^t \subseteq S^{\mathcal{M}} \times S^{\mathcal{M}}$. We say that a state s is *universally protected* in a protection t iff $(s, \psi) \in S^t$ for some ψ.

The following function, having as value the set of all the successors of a given state, will be useful in the sequel:

$$Suc(R, s_0) = \{s_i \mid (s_0, s_i) \in R\}\,.$$

The states in $Suc(R^t, s_0)$ will be said to be the *protected successors* of s_0. The states in $Suc(R^{\mathcal{M}}, s_0) - Suc(R^t, s_0)$ will be said to be the *nonprotected successors* of s_0.

Before giving the topmost level of the state-by-state **CTL** updater, we will posit the availability of a number of primitive operations. First, we need to be able to remove the transitions to the nonprotected successors of a state: If

$s_0 \in S^{\mathcal{M}}$, and $R^t \subseteq S^{\mathcal{M}} \times S^{\mathcal{M}}$, then $\mathcal{M}[R^t, s_0]$ is the *restriction* of $\mathcal{M}$ by R^t in s_0:

$$\mathcal{M}[R^t, s_0] = \langle S^{\mathcal{M}}, R^{\mathcal{M}} - \{(s_0, s) \in R^{\mathcal{M}} \mid s \notin Suc(R^t, s_0)\}, L^{\mathcal{M}} \rangle .$$

We must also be able to add a (state, formula)-pair to the first component of a protection:

$$t[s_0, \alpha] = \langle S^t \cup \{(s_0, \alpha)\}, R^t \rangle$$

as well as add a transition either to a structure or to a protection:

$$\begin{aligned} \mathcal{M}[s_0, s] &= \langle S^{\mathcal{M}}, R^{\mathcal{M}} \cup \{(s_0, s)\}, L^{\mathcal{M}} \rangle , \\ t[s_0, s] &= \langle S^t, R^t \cup \{(s_0, s)\} \rangle . \end{aligned}$$

These operations allow us to define addition and removal of transitions constrained by a protection:

$$Add_e(\mathcal{M}, t, s_0, s) = Update^X(\mathcal{M}[s_0, s], t[s_0, s], s, \bigwedge\{\psi \,|\, (s_0, \psi) \in S^t\})$$

(when adding a transition, we preserve the universal protection of s_0 (if any), hence the call to $Update^X$).

$$Rem_e(\mathcal{M}, t, s_0, s_i) = \begin{cases} \emptyset & \text{if } (s_0, s_i) \in R^t \\ \{(\langle S^{\mathcal{M}}, R^{\mathcal{M}} - \{(s_0, s_i)\}, L^{\mathcal{M}} \rangle, \langle S^t, R^t \rangle)\} & \text{otherwise} \end{cases}$$

(we only remove a transition if it is not protected).

The topmost level of the state-by-state **CTL** updater is:

$$\begin{aligned} Update^X(\mathcal{M}, t, s_0, \top) &= \{(\mathcal{M}, t)\} \\ Update^X(\mathcal{M}, t, s_0, \bot) &= \emptyset \\ Update^X(\mathcal{M}, t, s_0, \ell) &= \begin{cases} \{(\mathcal{M}, t)\} & \text{if } \ell \in L^{\mathcal{M}}(s_0) \\ \emptyset & \text{otherwise} \end{cases} \\ Update^X(\mathcal{M}, t, s_0, \alpha \vee \beta) &= Update^X(\mathcal{M}, t, s_0, \alpha) \cup \\ &\quad\; Update^X(\mathcal{M}, t, s_0, \beta) \\ Update^X(\mathcal{M}, t, s_0, \alpha \wedge \beta) &= Update^{X*}(Update^X(\mathcal{M}, t, s_0, \alpha), s_0, \beta) \\ Update^X(\mathcal{M}, t, s_0, \mathbf{EX}\,\alpha) &= \bigcup_{s \in S^{\mathcal{M}}} Update^{X*}(Add_e(\mathcal{M}, t, s_0, s), s, \alpha) \\ Update^X(\mathcal{M}, t, s_0, \mathbf{AX}\,\alpha) &= \begin{cases} Update^X(\mathcal{M}[R^t, s_0], t[s_0, \alpha], s_0, \\ \qquad \mathbf{EX}\,\alpha) \qquad \text{if } Suc(R^t, s_0) = \emptyset \\ UpdateSts(\{(\mathcal{M}[R^t, s_0], t[s_0, \alpha])\}, \\ \qquad Suc(R^t, s_0), \alpha) \text{ otherwise} \end{cases} \end{aligned}$$

$$Update^{X*}(mts, s_0, \varphi) = \bigcup_{(\mathcal{M}, t) \in mts} Update^X(\mathcal{M}, t, s_0, \varphi)$$

$$\begin{aligned} \mathit{UpdateSts}(\mathit{mts}, \emptyset, \varphi) &= \mathit{mts}, \\ \mathit{UpdateSts}(\mathit{mts}, \{s\} \cup \mathit{ss}, \varphi) &= \mathit{UpdateSts}(\mathit{Update}^{X*}(\mathit{mts}, s, \varphi), \\ &\qquad \mathit{ss} - \{s\}, \varphi) \end{aligned}$$

Update for the rest of the **CTL** operators can be performed using fixed-point equivalences (1)–(4) and a cycle-detection mechanism mentioned earlier. If **EU**, **AU**, **ER**, and **AR** are seen as predicates [6, p. 63], then the computation of the greatest fixed points of the predicate transformers associated with **ER** and **AR** terminate in true; the computation of the least fixed points of the predicate transformers associated with **EU** and **AU** terminate in false. In terms of sets of (structure, protection)-pairs, updating w.r.t. $\mathcal{M}$ and t, "terminate in true" corresponds to "terminate in $\{(\mathcal{M}, t)\}$," and "terminate in false" corresponds to "terminate in $\emptyset$." We give as example the modification *Update* of Update^X with the parameter V (visited (state, formula)-pairs) for the case of the **EU** operator:

$$\mathit{Update}(\mathcal{M}, t, s_0, \mathbf{E}[\alpha\, \mathbf{U}\, \beta], V)$$
$$= \begin{cases} \emptyset & \text{if } (s_0, \mathbf{E}[\alpha\, \mathbf{U}\, \beta]) \in V, \\ \mathit{Update}(\mathcal{M}, t, s_0, & \\ \qquad \beta \vee (\alpha \wedge \mathbf{EX}\ \mathbf{E}[\alpha\, \mathbf{U}\, \beta]), & \\ \qquad V \cup \{(s_0, \mathbf{E}[\alpha\, \mathbf{U}\, \beta])\}) & \text{otherwise}. \end{cases}$$

Observe that although Update^X computes a set of (structure, protection)-pairs, in practice it is also possible to compute nondeterministically only *one* (structure, term)-pair at a time, instead of the whole set.

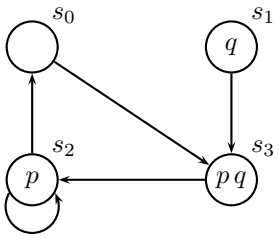
We end this section illustrating this state-by-state method. Consider the Kripke structure $\mathcal{M}_0$ in Section 1, where:

$$\begin{aligned} \mathcal{M}_0 = \langle\ & \{s_0, s_1, s_2, s_3\}, \\ & \{(s_0, s_1), (s_0, s_3), (s_1, s_3), (s_3, s_2), (s_2, s_0), (s_2, s_2)\}, \\ & \{(s_0, \{\neg p, \neg q\}), (s_1, \{\neg p, q\}), (s_2, \{p, \neg q\}), (s_3, \{p, q\})\}\ \rangle \end{aligned}$$

and the formula $\varphi_0 = \mathbf{EX}\, p \wedge \mathbf{AX}\, (p \wedge q)$, which we wish to hold at s_0. Apart from $\mathcal{M}_0$, s_0, and φ_0, the topmost call to Update^X starts with the null protection: $t_0 = \langle \emptyset, \emptyset \rangle$. First, the conjunction case of Update^X calls Update^X w.r.t. the first conjunct $\mathbf{EX}\, p$. This call in turn calls Add_e with a first successor of s_0, which we will assume to be s_1. Next, Add_e "adds" and protects the transition (s_0, s_1). Since such a transition is already present in $\mathcal{M}_0$, $\mathcal{M}_0[s_0, s_1] = \mathcal{M}_0$. The protection, in turn, is $t_0[s_0, s_1] = \langle \emptyset, \{(s_0, s_1)\} \rangle$. Subsequently, Add_e calls Update^X with arguments (1) $\mathcal{M}_0$, (2) $t_0[s_0, s_1]$, (3) s_1, and (4) the identity for conjunction $\top$ (because s_0 is not universally protected), succeeding immedi-

ately. As a result (going back to the **EX** case in $Update^X$), $Update^{X*}$ is called with $\{(\mathcal{M}_0, t_0[s_0, s_1])\}$, which amounts to calling $Update^X$ with $\mathcal{M}_0$, $t_0[s_0, s_1]$, s_0, and p. As p does not label s_1, $Update^X$ returns the empty set. The next (and last) successor of s_0 is s_3. A similar process as that for s_1 occurs, except that this time the treatment of the literal case succeeds (because p does label s_3) returning $\{(\mathcal{M}_0, t_0[s_0, s_3])\}$.

We now get back to the conjunction case, which updates w.r.t. $\beta = \mathbf{AX}\,(p \wedge q)$ the result of having updated w.r.t. $\alpha = \mathbf{EX}\,p$ the structure $\mathcal{M}_0$. Next, $Update^{X*}$ is called with the transition (s_0, s_3) protected. This in turn calls $Update^X$ with $\mathbf{AX}\,(p \wedge q)$. Observe that $R^t = \{(s_0, s_3)\}$. Hence, $Suc(R^t, s_0) = \{s_3\} \neq \emptyset$. The restriction $\mathcal{M}_1$ of $\mathcal{M}_0$ by R^t is then $\mathcal{M}_0$ with the transition (s_0, s_1) removed. Subsequently, a call to *UpdateSts* is performed with arguments (1) a singleton with the pair having $\mathcal{M}_1$ and a universal protection at s_0 w.r.t. $p \wedge q$, (2) the protected successor(s) of s_0 (namely $\{s_3\}$), and (3) the formula $\mathbf{AX}\,(p \wedge q)$. This call to *UpdateSts* amounts to a single call to $Update^X$ with state s_3. After treating the conjunction of literals $(p \wedge q)$, such a call succeeds producing the structure:



4. Table Systems

In this section, we give a detailed description a representation of systems by means of tables. The main purpose of table systems will be to provide a simple language for the succinct representation of some large Kripke structures.

The basic element for the description of systems by means of tables is the notion of "change rule." A change rule attempts to indicate the way to change the value of a variable v in a given state s, in order to produce the new value of v for a successor of s. Therefore, a change rule consists of two components, a change condition t, and a set of values B, with the following meaning: at the state s, the variable v nondeterministically changes to a value in B when s satisfies the condition t.

Definition 1. (Change Rule) We say that c is a *change rule* iff $c = (t^c, B^c)$ where $t^c \in \mathit{Terms}$ and $B^c \subseteq \mathit{Boolean}$. We use *Rules* for denoting the set of change rules.

To express several change conditions, we can accumulate several change rules into a "change table," and consider the application of a default function when none of the rules is applicable. It will be useful to have two kinds of default function, one which preserves the value of some component of the current state, and another one which takes a constant value. In practice, we represent the default of a table T with "otherwise v_j" for the first kind or with "otherwise B" for the second kind.

Definition 2. (Default) We say that $d : \mathit{States} \to 2^{\mathit{Boolean}} - \{\emptyset\}$ is a default iff either:

1. $\forall s \in \mathit{States} : d(s) = \{s_j\}$, where s_j is the jth component of s, for some $1 \leq j \leq n$, or

2. $\forall s \in \mathit{States} : d(s) = B$, for some $B \in 2^{\mathit{Boolean}} - \{\emptyset\}$.

From now on, we assume that, for $i = 1, \ldots, n$, d_i is a function that intuitively represents the default behavior of the variable v_i.

Definition 3. (Change Table and *rSat*) Let $T \subseteq \mathit{Rules}$, and d_i a default. We say that T is a *change table* for the variable v_i with default d_i. We use $\mathit{rSat}(s, T)$ for denoting the set of *rules satisfied by* s in T:

$$\mathit{rSat}(s, T) = \{c \in T \mid s \models t^c\}\,.$$

Finally, a table system is a set of n change tables, with exactly one table for each variable v_i:

Definition 4. (Table System and *rSat*) We say that H is a *table system* on *Atoms* iff $H = \{T_1^H, T_2^H, \ldots, T_n^H\}$ where, for $i = 1, \ldots, n$, T_i^H is a change table for the variable v_i with default d_i. We extend the notation $\mathit{rSat}(s, T)$ to a system H, i.e., $\mathit{rSat}(s, H) = \bigcup\{\mathit{rSat}(s, T_i^H) \mid i = 1, \ldots, n\}$. We use *TSystems* to denote the set of table systems on *Atoms*.

We are now in a position to describe the dynamic behavior of table systems by means of a next-states function. Let us begin by defining a next value of a component of a state w.r.t. a change table T.

Definition 5. (next_T) Let T be a change table for the variable v_i with default d_i, and $s \in \mathit{States}$. The set of *next values* for the ith component for s,

w.r.t. T is given by:

$$next_T(s) = \begin{cases} \bigcup\{B^c \mid c \in rSat(s,T)\} & \text{if } rSat(s,T) \neq \emptyset\,, \\ d_i(s) & \text{if } rSat(s,T) = \emptyset\,. \end{cases}$$

Let us observe that the second case of $next_T$ resorts to the value provided by the default d_i. Therefore, if we take $d_i(s) = \{s_i\}$, we get a deterministic effect in $next_T(s)$ that keeps s_i without change. In contrast, if we take $d_i(s) = Boolean$, we get a nondeterministic effect in $next_T(s)$ representing the absence of knowledge about the behavior of v_i at state s.

Definition 6. ($next_H$) Let $H = \{T_q^H, \ldots, T_n^H\}$ be a table system. The *next-states function* for H, $next_H : States \to 2^{States}$, computes the set of *successors of a state s*:

$$next_H(s) = \{s' = (s'_1, \ldots, s'_n) \in States \mid \forall i = 1, \ldots, n, s'_i \in next_{T_i^H}(s)\}\,.$$

Since the semantics of **CTL** is defined by Kripke structures, with the purpose of having a semantics of **CTL** by means of table systems, we associate a structure $\mathcal{M}_H$ with a table system H.

Definition 7. ($\mathcal{M}_H$) For a table system H, $\mathcal{M}_H$ is the Kripke structure associated with H iff $\mathcal{M}_H = \langle S, R, L\rangle$ where:

1. $S = States$.
2. $R = \{(s, s') \in States^2 \mid s' \in next_H(s)\}$.
3. $L : States \to \mathcal{P}^+(Atoms)$; $L(s) = \{\ell \in Lit \mid s \models \ell\}$.

Let us define the *size* of some of the above objects: terms, change tables, table systems, and Kripke structures.

Definition 8. Let t be a term, T a change table, $H = \{T_1^H, T_2^H, \ldots, T_n^H\}$ a table system, and $\mathcal{M} = \langle S, R, L\rangle$ a Kripke structure; *size* is defined by cases:

1. $size(t) = |t|$, the cardinality of t.
2. $size(T) = \sum_{c \in T} size(t^{c_i})$.
3. $size(H) = \sum_{i=1}^{n} size(T_i^H)$.
4. $size(\mathcal{M}) = |S| + |R| + |L|$.

If we assume that $S = States$, then $|S|$ is exponential in the number of atoms, $|States| = 2^n$; therefore for all $\mathcal{M} \in K_{States}$, $size(\mathcal{M})$ is $\mathcal{O}(2^n)$. In contrast, for some table systems H the order of $size(H)$ is polynomial in n, although $size(\mathcal{M}_H)$ is $\mathcal{O}(2^n)$.

In the worst case, the order of $size(H)$ is exponential in n, but in practice

(in NuSMV, for instance), it is common to describe table systems in which $size(H)$ is polynomial in n.

If $H = \{T_1^H, T_2^H, \ldots, T_n^H\}$ is a table system such that for $i = 1, \ldots, n$, the order of $size(T_i^H)$ is polynomial in n, then the order of $size(H)$ is also polynomial in n. This means that table systems are a succinct representation of a subset of K_{States}.

5. An Update Method for Table Systems

5.1. Auxiliary Functions

Our model update algorithm is based on two basic operations over rules: adding (Add_r) and replacing (Rep_r) a rule.

Definition 9. (Add_r and Rep_r) Let T be a table for the variable v_i with default d_i, $s \in States$, $b \in Boolean$, and $B \subseteq Boolean$. We define the functions $Add_r : States \times Boolean \times Tables \to Tables$, for *adding the rule* $(L(s), \{b\})$ to T, and $Rep_r : States \times 2^{Boolean} \times Tables \to Tables$, for *replacing the rule* $(L(s), \_)$ with $(L(s), B)$ in the table T, respectively:

$$\begin{aligned} Add_r(s, b, T) &= T \cup \{(L(s), \{b\})\} \\ Rep_r(s, B, T) &= (T - rSat(s, T)) \\ &\quad \cup \{(L(s), B)\} \\ &\quad \cup \{(L(s'), B^c) \mid s' \in States - \{s\} \ \& \\ &\qquad\qquad \exists\, c \in rSat(s, T) : s' \models c\}\,. \end{aligned}$$

Add_r simply adds a new rule. Rep_r, by contrast, requires an explanation. Observe first that if state s satisfies no rule, $rSat(s, T) = \emptyset$, then a new rule is added: $Rep_r(s, B, T) = T \cup \{(L(s), B)\}$. If s satisfies some rules, $rSat(s, T) \neq \emptyset$, we expand the rules satisfied by s and replace those corresponding to the condition $L(s)$ with the rule $(L(s), B)$, and keep the rest of the rules.

Finally, remark that Add_r and Rep_r may produce tables that depend on all the variables of $Atoms$. Note that these functions do not have an effect on the default value of the tables.

To illustrate the application of these functions, let us revisit the example of Section 1. Let s be the state $(0, 0)$, and $d_1, d_2 : States \to 2^{Boolean} - \{\emptyset\}$ such that for all $s \in States$: $d_1(s) = \{1\}$ and $d_2(s) = \{0\}$. And let $T_1 = \{(\{\neg q\}, \{0, 1\})\}$ and $T_2 = \{(\{\neg p\}, \{1\})\}$ be tables for the variables p and q, with defaults d_1 and

d_2, respectively:

q	p'
0	$*$
otherwise	1

p	q'
0	1
otherwise	0

The application of $Add_r(s, 0, T_2)$ transforms the table T_2 by adding the rule $(L(s), \{0\}) = (\{\neg p, \neg q\}, \{0\})$, which we will abbreviate as "$(00, 0)$." Thus, the diagram of $Add_r((0,0), 0, T_2)$ (rewriting the original row of T_2 as $(0*, 1)$) is:

$p\ q$	q'
$0\ *$	1
$0\ 0$	0
otherwise	0

In a similar way, the application of $Rep_r(s, \{1\}, T_1)$ transforms T_1 as follows. The set of rules satisfied by $s = (0,0)$ in T_1 is $rSat(s, T_1) = \{(\{q\}, \{0,1\})\} \neq \emptyset$, i.e., $rSat(s, T_1)$ contains exactly the only row of T_1, $rSat(s, T_1) = T_1$. Therefore,

$$\begin{aligned} Rep_r(s, \{1\}, T_1) &= (T_1 - rSat(s, T_1)) \\ &\cup \{(L(s), \{1\})\} \\ &\cup \{(L(s'), B^c) \mid s' \in States - \{s\} \ \& \\ &\qquad \exists\, c \in rSat(s, T_1) : s' \models c\} \\ &= \emptyset \\ &\cup \{(\{\neg p, \neg q\}, \{1\})\} \\ &\cup \{(L(s'), B^c) \mid s' \in States - \{s\} \ \& \\ &\qquad \exists\, c \in rSat(s, T_1) : s' \models c\}\,. \end{aligned}$$

To compute the last term of this union, observe that the only rule in $rSat(s, T_1)$ is $(*0, *)$ and the only state different from $(0,0)$ that satisfies $*0$ is $(1,0)$, therefore:

$$\begin{aligned} Rep_r(s, \{1\}, T_1) &= \emptyset \cup \{(\{\neg p, \neg q\}, \{1\})\} \cup \{(L((1,0)), \{0,1\})\} \\ &= \{(\{\neg p, \neg q\}, \{1\})\} \cup \{(\{p, \neg q\}, \{0,1\})\} \\ &= \{(\{\neg p, \neg q\}, \{1\}), (\{p, \neg q\}, \{0,1\})\}\,. \end{aligned}$$

Thus, for the table $Rep_r(s, \{1\}, T_1)$ we have the diagram:

$p\ q$	p'
$0\ 0$	1
$1\ 0$	$*$
otherwise	1

For the description of our method we will need to be able to add transitions to a system H. The following definition gives such an operation as the multiple application of Add_r.

Definition 10. (Addition of a Transition) If $H = \{T_1^H, T_2^H, \ldots, T_n^H\}$ is a table system and $s, s' \in States$, the table system resulting from adding the transition (s, s') to H is:

$$H[s, s'] = \{Add_r(s, s'_1, T_1^H), Add_r(s, s'_2, T_2^H), \ldots, Add_r(s, s'_n, T_n^H)\}\,.$$

To see an example of $H[s, s']$, let T_1 and T_2 be the tables of the above example and let $H = \{T_1, T_2\}$. If $s = (0, 0)$ and $s' = (0, 0)$ then:

$$\begin{aligned} H[s, s'] &= \{Add_r(s, s'_1, T_1^H), Add_r(s, s'_2, T_2^H)\} \\ &= \{Add_r((0,0), 0, T_1^H), Add_r((0,0), 0, T_2^H)\} \\ &= \{T_1^H \cup \{(L((0,0)), \{0\})\}, T_2^H \cup \{(L((0,0)), \{0\})\}\} \\ &= \{T_1^H \cup \{(\{\neg p, \neg q\}, \{0\})\}, T_2^H \cup \{(\{\neg p, \neg q\}, \{0\})\}\} \end{aligned}$$

Hence, the diagrams of the tables of the system $H[(0, 0), (0, 0)]$ are:

p q	p'
$*$ 0	$*$
0 0	0
otherwise	1

p q	q'
0 $*$	1
0 0	0
otherwise	0

We will need to be able to restrict a table system H by a set of protected transitions given by a protection t.

Definition 11. (Restriction of H by R^t) Let $H = \{T_1^H, T_2^H, \ldots, T_n^H\}$ be a table system, $s \in States$, and t a protection such that $Suc(R^t, s) = \{s_1, s_2, \ldots, s_m\}$ and, for $i = 1, \ldots, n$, let B_i be the set of the ith components of these states, $B_i = \{s_{1i}, s_{2i}, \ldots, s_{mi}\} \subseteq Boolean$. The table system that restricts H by R^t at s is defined by:

$$H[R^t, s] = \{Rep_r(s, B_1, T_1^H), Rep_r(s, B_2, T_2^H), \ldots, Rep_r(s, B_n, T_n^H)\}\,.$$

It is important to note that the replacement process that produces the system $H[R^t, s]$, possibly eliminates several transitions of the system H. However, the source of all the transitions that this process eliminates is the state s. $H[R^t, s]$ preserves, by definition, all the protected transitions with source s. Therefore, this replacement process only eliminates unprotected transitions.

Let us continue with the above example, where $H = \{T_1, T_2\}$ and $s = (0, 0)$. Assume $R^t = \{((0, 0), (1, 0))\}$ so that $B_1 = \{1\}$, $B_2 = \{0\}$, are the first and

second components of state $(1,0)$, respectively

$$H[R^t, s] = \{Rep_r(s, \{1\}, T_1), Rep_r(s, \{0\}, T_2)\}.$$

We already computed $Rep_r(s, \{1\}, T_1)$, and after a simple computation we get:

$$\begin{aligned} Rep_r(s, \{0\}, T_2) &= \emptyset \cup \{(\{\neg p, \neg q\}, \{0\})\} \cup \{(L((0,1)), \{1\})\} \\ &= \{(\{\neg p, \neg q\}, \{0\})\} \cup \{(\{\neg p, q\}, \{1\})\} \\ &= \{(\{\neg p, \neg q\}, \{0\}), (\{\neg p, q\}, \{1\})\}. \end{aligned}$$

Thus, $H[R^t, s]$ is given by the tables:

p	q	p'
0	0	1
1	0	$*$
otherwise		1

p	q	q'
0	0	0
0	1	1
otherwise		0

We remark that the time necessary to compute $H[s, s']$ is polynomial in n, but the time complexity for $H[R^t, s]$ is exponential in n (in the worst case, the set of states that satisfy a row of a table is equal to *States* and therefore Rep_r produces a table of exponential size).

5.2. Table Update for CTL Formulas

Having defined the auxiliary functions, the state-by-state updater of Section 3 can readily be rewritten for table systems:

$$\begin{aligned} TUpdate^X(H, t, s_0, \top) &= \{(H, t)\} \\ TUpdate^X(H, t, s_0, \bot) &= \emptyset \\ TUpdate^X(H, t, s_0, \ell) &= \begin{cases} \{(H, t)\} & \text{if } \ell \in L(s_0) \\ \emptyset & \text{otherwise} \end{cases} \\ TUpdate^X(H, t, s_0, \alpha \vee \beta) &= TUpdate^X(H, t, s_0, \alpha) \cup \\ &\quad TUpdate^X(H, t, s_0, \beta) \\ TUpdate^X(H, t, s_0, \alpha \wedge \beta) &= TUpdate^{X*}(TUpdate^X(H, t, s_0, \alpha), s_0, \\ &\quad \beta) \\ TUpdate^X(H, t, s_0, \mathbf{EX}\,\alpha) &= \bigcup_{s \in States} TUpdate^{X*}(TAdd_e(H, t, s_0, s), \\ &\quad t, s, \alpha) \end{aligned}$$

$$TUpdate^X(H,t,s_0,\mathbf{AX}\,\alpha) = \begin{cases} TUpdate^X(H[R^t,s_0],t[s_0,\alpha],s_0, \\ \quad \mathbf{EX}\,\alpha) & \text{if } Suc(R^t,s_0)=\emptyset \\ TUpdateSts(\{(H[R^t,s_0],t[s_0,\alpha])\}, \\ \quad Suc(R^t,s_0),\alpha) & \text{otherwise} \end{cases}$$

where $TAdd_e(H,t,s_0,s)$ allows adding the transition (s_0,s) to the table system H considering the protection t:

$$TAdd_e(H,t,s_0,s) = TUpdateSts(\{(H[s_0,s],t[s_0,s])\}, next_{H[s_0,s]}(s_0), \bigwedge\{\psi \mid (s_0,\psi)\in S^t\}).$$

Unlike Add_e of the state-by-state updater, $TAdd_e$ needs to update a set of states because $H[s_0,s]$ not only adds the transition from s_0 to s, but other, additional ones, whose targets are considered in $next_{H[s_0,s]}(s_0)$.

$TUpdate^{X*}$ and $TUpdateSts$ are auxiliary functions for updating several tables and states, respectively:

$$TUpdate^{X*}(hts,s_0,\varphi) = \bigcup_{(H,t)\in hts} TUpdate^X(H,t,s_0,\varphi)$$

$$TUpdateSts(hts,\emptyset,\varphi) = hts,$$

$$TUpdateSts(hts,\{s\}\cup ss,\varphi) = TUpdateSts(TUpdate^{X*}(hts,s,\varphi), ss-\{s\},\varphi).$$

Table update for the operators $\mathbf{AU},\mathbf{EU},\mathbf{AR}$, and $\mathbf{ER}$ can be performed in the same way that the state-by-state method, i.e., using fixed-point equivalences (1)–(4) and a cycle-detection mechanism, testing for membership of a (state, formula)-pair in a list of visited (state, formula)-pairs.

We now state the soundness of $TUpdate$. Before that, however, we need an adequate notion of consistency.

Definition 12. (Consistent With) A Kripke structure $\mathcal{M}$ is *consistent with* a protection t, which we write as $\mathcal{M}$ **Con** t, iff:

1. $\forall(s,\alpha)\in S^t : \mathcal{M}\models_s \mathbf{AX}\,\alpha$ and
2. $R^t\subseteq R^{\mathcal{M}}$.

We write $\mathcal{M}$ **Con** t,t' to express that $\mathcal{M}$ is *consistent with* both protections t and t'.

Theorem 13. (Soundness) *If $H\in TSystems$, $\varphi\in\mathbf{XCTL}$, $s\in States$, and t is a protection such that $\mathcal{M}_H$* **Con** *t, then $\forall(H',t')\in TUpdate^X(H,t,s,\varphi)$: $\mathcal{M}_{H'}\models_s\varphi$ and $\mathcal{M}_{H'}$* **Con** *t,t'.*

We give a proof of this theorem in the appendix using structural induction

on φ.

6. An Example of CTL Table Update

For the sake of illustrating our method, we apply *TUpdate* to a simple answering-machine model with four variables: *del* (the "delete" button is pressed), *play* (the "play" button is pressed), m (there is a message), and p (the answering machine is playing a message back or giving an announcement). This answering machine can only remember one message at a time. We assume that the order of the variables is $(del, play, m, p)$ and the model consists of four tables, $H = \{T_1, T_2, T_3, T_4\}$.

Since *del* and *play* are considered exogenous variables, the tables that describe their behavior, T_1 and T_2, respectively, are trivial tables:

del	*play*	m	p	del'
otherwise				$*$

del	*play*	m	p	$play'$
otherwise				$*$

For the behavior of the variable m we have the table T_3:

del	*play*	m	m'	Intuitive meaning
1	$*$	1	0	-- "delete" erases the current message
0	1	1	1	-- (if not *del*) "play" does not erase m
$*$	$*$	0	$*$	-- a message may or may not arrive
otherwise			m	-- any other condition preserves m's value

For the behavior of the variable p we have the table T_4:

del	*play*	p	p'	Intuitive meaning
0	1	$*$	1	-- (if not *del*) "play" initiates playing
1	$*$	1	0	-- "delete" stops playing
$*$	0	0	$*$	-- "play" may or may not be pressed
otherwise			p	-- any other condition preserves p's value

Let s be the state $s = (1, 0, 1, 0)$ and let ψ be any of the properties of the list below. We can verify, using a **CTL** model checker (for example NuSMV), that the model associated with the above table system H satisfies ψ at s, i.e., $\mathcal{M}_H \models_s \psi$:

1. The "delete" button may erase the current message: $del = 1 \rightarrow \mathbf{EX}\, m = 0$.

2. The "play" button does not erase the current message: $del = 0$ & $play = 1$ & $m = 1 \rightarrow \mathbf{AX}\, m = 1$.

3. A message may or may not arrive: $m = 0 \rightarrow \mathbf{EX}\, m = 1$ & $\mathbf{EX}\, m = 0$.

4. The "play" button initiates playing (if "delete" is not pressed): $play = 1$ & $del = 0 \rightarrow \mathbf{AX}\, p = 1$.

5. The "delete" button stops playing: $del = 1$ & $p = 1 \rightarrow \mathbf{AX}\, p = 0$.

6. The "play" button may or may not be pressed: $play = 0$ & $p = 0 \rightarrow \mathbf{EX}\, play = 1$ & $\mathbf{EX}\, play = 0$.

7. A message may eventually arrive: $\mathbf{EF}\, m = 1$.

8. A message may eventually not arrive: $\mathbf{EF}\, m = 0$.

9. The "play" button may eventually be pressed: $\mathbf{EF}\, play = 1$.

10. The "play" button may eventually not be pressed: $\mathbf{EF}\, play = 0$.

If we look carefully at the first property of this list, we see that it does not include the case of a new message arriving just at the moment of pressing the "delete" button. In this case, it would be possible that $m = 1$ after pressing the "delete" button. The next formula φ captures this property:

$$\varphi \equiv (del = 1 \rightarrow \mathbf{EX}\, m = 1)$$

but the model $\mathcal{M}_H$ does not satisfy φ at s, i.e., $\mathcal{M}_H \not\models_s \varphi$.

We are going to give an example of the application $TUpdate^X$ in order to update H at s w.r.t. φ.

Let us rewrite φ into a form appropriate for $TUpdate^X$:

$$\varphi \equiv (del = 0 \vee \mathbf{EX}\, m = 1) \equiv (\overline{del} \vee \mathbf{EX}\, m)\,.$$

Since $s = (1, 0, 1, 0)$, $L(s) = \{del, \overline{play}, m, \bar{p}\}$. We begin the application of $TUpdate^X$ with the null protection t, so let $t = \langle \emptyset, \emptyset \rangle$. Then:

$$TUpdate^X(H, t, s, \varphi) = TUpdate^X(H, t, s, \overline{del} \vee \mathbf{EX}\, m),$$

therefore,

$$TUpdate^X(H, t, s, \varphi) = TUpdate^X(H, t, s, \overline{del}) \cup TUpdate^X(H, t, s, \mathbf{EX}\, m)$$

But

$$TUpdate^X(H, t, s, \overline{del}) = \emptyset$$

because $\overline{del} \notin L(s)$, so:

$$TUpdate^X(H, t, s, \varphi) = TUpdate^X(H, t, s, \mathbf{EX}\, m).$$

To compute $TUpdate^X(H, t, s, \mathbf{EX}\, m)$, we proceed in a nondeterministic way

with the formula:

$$TUpdate^X(H,t,s,\mathbf{EX}\,m) = \bigcup_{s'\in States} TUpdate^{X*}(TAdd_e(H,t,s,s'),t,s',m)\,,$$

i.e., we guess a state $s' = (s'_1, s'_2, s'_3, s'_4)$. A good nondeterministic choice of s' is $s' = s = (1,0,1,0)$, which gives us:

$$\begin{aligned} TUpdate^{X*}(TAdd_e(H,t,s,s'),t,s',m) &= TUpdate^{X*}(\{H[s,s']\},t,\\ &\qquad\qquad s',m)\\ &= \{(H[s,s'],t[s,s'])\} \end{aligned}$$

because:

$$\begin{aligned} TAdd_e(H,t,s,s') &= TUpdateSts(\{(H[s,s'],t[s,s'])\}, next_{H[s,s']}(s),\\ &\qquad\qquad \bigwedge\{\alpha \mid (s,\alpha)\in S^t\})\\ &= TUpdateSts(TUpdate^{X*}(\{(H[s,s'],t[s,s'])\},s'',\\ &\qquad\qquad \bigwedge\{\alpha \mid (s,\alpha)\in\emptyset\}),\\ &\qquad\qquad next_{H[s,s']}(s)-\{s''\},\\ &\qquad\qquad \bigwedge\{\alpha \mid (s,\alpha)\in\emptyset\}) \end{aligned}$$

$$\begin{aligned} TAdd_e(H,t,s,s') &= TUpdateSts(TUpdate^{X*}(\{(H[s,s'],t[s,s'])\},s'',\\ &\qquad\qquad \top),\\ &\qquad\qquad next_{H[s,s']}(s)-\{s''\},\top)\\ &= TUpdateSts(\{(H[s,s'],t[s,s'])\},\\ &\qquad\qquad next_{H[s,s']}(s)-\{s''\},\top)\\ &\vdots\\ &= TUpdateSts(\{(H[s,s'],t[s,s'])\},\emptyset,\top)\\ &= \{(H[s,s'],t[s,s'])\,,\} \end{aligned}$$

where $s'' \in next_{H[s,s']}(s) = \{(1010),(1000),(1001),(1011)\}$, so that we repetitively remove elements from $next_{H[s,s']}(s)$.

Finally, we need to compute $H[s,s']$:

$$\begin{aligned} H[s,s'] &= \{Add_r(s,s'_1,T_1), Add_r(s,s'_2,T_2), Add_r(s,s'_3,T_3),\\ &\qquad Add_r(s,s'_4,T_4)\}\\ &= \{T'_1,T'_2,T'_3,T'_4\}\,, \end{aligned}$$

where T'_1 and T'_2 are the tables:

del	*play*	*m*	*p*	*del′*
1	0	1	0	1
otherwise				∗

del	*play*	*m*	*p*	*play′*
1	0	1	0	0
otherwise				∗

and the tables T_3' and T_4' are:

del	*play*	*m*	*p*	*m′*
1	∗	1	∗	0
0	1	1	∗	1
∗	∗	0	∗	∗
1	0	1	0	1
otherwise				*m*

del	*play*	*m*	*p*	*p′*
0	1	∗	∗	1
1	∗	∗	1	0
∗	0	∗	0	∗
1	0	1	0	0
otherwise				*p*

The most important change happened in T_3 (T_1 and T_2 simply received a new row, and T_4' is equivalent to T_4 —the new row in T_4' is covered by the third one). In T_3', by contrast, the combined action of the first and the fourth rows produces a new meaning: "the "delete" button erases *m but a new message may simultaneously arrive.*"

7. Related Work

Of the CTL model updaters known to us [1, 2, 3, 10], only that used by the Biocham system [2] employs a concise language for specifying large Kripke structures. Biocham's model updater is designed for modelling biochemical networks. The user describes such networks with rules of a form such as: `A+B=>C+D`, where `A`, `B`, `C`, and `D` denote chemical substances. Biocham, in turn, translates each such rule into four transitions, corresponding to the complete or incomplete consumption of reactants `A` and `B`.

Another important characteristic of Biocham's model updater is that it is built on top of NuSMV. Hence, Biocham can rely on such a powerful model checker. NuSMV generates counterexamples, used by Biocham as a guide to repair faulty Kripke structures.

It is important to observe that counterexamples can only be generated for *universal* properties: existential properties cannot be disproved by counterexamples [5].

Biocham updates a faulty Kripke structure as follows. Assume that $\mathcal{M} \models_s \varphi$ does not hold, and that we wish to modify $\mathcal{M}$ so that it does. If φ contains only nonnegated *universal* temporal operators, a counterexample represents a

path making φ false. Biocham, therefore, deletes transitions occurring in such a path. If the deletion produces an accessibility relation which is not total, leaving a state with no successors, then a self-loop at that state is added.

If φ contains only nonnegated *existential* temporal operators, a counterexample will not be produced. Biocham, however, can add transitions using a bias taken from the application domain.

Formulas that have both universal and existential operators, i.e., "unclassified" formulas, are treated by adding and deleting transitions using a counterexample. Such a counterexample only provides information about the outermost universal operators of the formula. Again Biocham can add transitions using a bias.

Biocham treats first the existential formulas, then the unclassified ones, and finally the universal ones. The deletions for satisfying universal formulas are allowed to dissatisfy some existential or unclassified formulas, in which case such formulas are treated again, this time not dissatisfying the universal formulas already satisfied. Presumably existential formulas are treated first because, as universal formulas are treated with deletions, we would otherwise produce a "poor" structure.

There are two main differences between ours and Biocham's model updater. On the one hand, whereas Biocham is meant for biochemical networks, specified by rules of chemical reactions, we provide a more generic language, which is a fragment of that used by NuSMV. On the other hand, while Biocham employs counterexamples, we use an algorithm consisting primarily of protections. Now, counterexamples only provide information about the outermost universal operators of the formula. Thus, all other occurrences of operators are treated with heuristics. By contrast, our method could be viewed as establishing a search space. Such a search space, in turn, could possibly be traversed with heuristics dependent on a particular application.

Both model updaters are incomplete, but for different reasons. One reason why Biocham's is incomplete is the conversion of the accessibility relation to a total one with a self-loop, without considering other ways in which to make a universal formula hold. Another one is that satisfying an existential or unclassified formula may require the addition of transitions not added by Biocham because of the bias. We, in turn, only allow the update of transition systems which are representable as table systems.

8. Concluding Remarks

We devote this section to summarizing our work and suggesting future directions of research.

8.1. Model Update and Table Systems

Model checking is an approach to formal verification determining the presence or absence of properties (typically represented as temporal-logic formulas) in a system (usually represented as a Kripke structure). On the one hand, model checking contrasts with full-behavior verification methods by concentrating on partial-behavior characterizations. On the other hand, model checking differs from ordinary testing by dealing with properties instead of single input-output pairs. In addition, model checking is normally intended for systems with an ongoing interaction with the environment, unlike both full-behavior verification methods and ordinary testing, generally taking a halting system. From an inference point of view, model checking departs from other formal methods based on deduction, in computing whether or not a single structure (model) satisfies a formula.

Model *update* tries to go one step further than model checking by repairing a Kripke structure not satisfying a given formula. A number of model-update methods for computation-tree logic (CTL) have been recently developed [1, 2, 3, 10]. Some of these model updaters utilize a state-by-state representation of the Kripke structure, where each state and transition occurs explicitly. Practical model *checkers*, such as SPIN or NuSMV, employ a special-purpose language for concisely describing large Kripke structures. This suggests that model *updaters* should also use one such language, especially because both the input and output of a model updater are Kripke structures. To our knowledge, Biocham's [2] is the only CTL model updater providing such a structure-description language. This updater, however, is oriented towards modelling biochemical networks. In an attempt to devise a more general purpose CTL model updater, we have selected instead a simplification of the language used by NuSMV, we have termed "table systems."

Like NuSMV, table systems represent the accessibility relation of a Kripke structure as a set of Boolean functions f_i, one for each propositional variable v_i; we call each f_i a "table." Unlike NuSMV, table systems do not allow references to the next value of a variable in specifying each f_i.

We restrict our update method to table systems (which cannot represent

arbitrary Kripke structures given a fixed set of variables). This restriction can be justified by arguing that we wish the result of an update to be readable by the user. This renders our updater incomplete, in the sense that given a CTL satisfiable formula φ, our updater may not be able to find a structure satisfying φ.

8.2. Protections

The chief principle behind both the earlier state-by-state method [3] and the present one is that of the preservation of a truth value of a subformula via a "protection." A protection is interpreted as a constraint on modifications to a structure, allowing the treatment, for example, of conjunctions and the **AX** operator.

8.3. Future Work

Our motivation for introducing table systems to a state-by-state CTL model updater was that of developing an updater suitable for practical applications. Hence, a natural next step would be to test the resulting updater in one such application.

For brevity, we have presented our updater employing a coarse version [3] of the update for the **AX** operator, removing all nonprotected successors of the current state. In a practical application, however, it would be advisable to utilize instead the more refined version of the update for the **AX** operator in [3], considering the possibility of updating the nonprotected successors of the current state.

Another avenue of research would be to establish a firmer formal basis of our algorithm via an ordering relation. Our method performs "small" changes to the representation of a faulty Kripke structure, but we have not formally defined the set of admissible changes to a structure. A definition of such changes could possibly be stated through an ordering relation of table systems.

Acknowledgments

We are grateful to Julián Argil, Lourdes del Carmen González Huesca, Adalberto Hernández Llarena, and Joel Espinosa Longi for their help with the answering-machine example, as well as to Carlos Velarde for his help with

LaTeX. We are also happy to acknowledge the facilities provided by IIMAS, UNAM.

References

[1] Francesco Buccafurri, Thomas Eiter, Georg Gottlob, Nicola Leone, Enhancing model checking in verification by AI techniques, *Artificial Intelligence*, **112**, No-s: 1-2 (1999), 57-104.

[2] Laurence Calzone, Nathalie Chabrier-Rivier, François Fages, Sylvain Soliman, Machine learning biochemical networks from temporal logic properties, In: *Transactions on Computational Systems Biology VI*, Lecture Notes in Bioinformatics No. 4220 (2006), 68-94.

[3] Miguel Carrillo, David A. Rosenblueth, A state-by-state method for CTL model update, *Technical Report Preimpreso*, No. 146, Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, Universidad Nacional Autónoma de México (2008); url: http://leibniz.iimas.unam.mx/~drosenbl/ctl_update_report.

[4] E.M. Clarke, E.A. Emerson, A.P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications, *ACM Transactions of Programming Languages and Systems*, **8**, No. 2 (1984), 244-263.

[5] Edmund Clarke, Somesh Jha, Yuan Lu, Helmut Veith, Tree-like counterexamples in model checking, In: *IEEE Symposium on Logic in Computer Science (LICS)* (2002).

[6] Edmund M. Clarke, Orna Grumberg, Doron A. Peled, *Model Checking*, MIT Press (1999).

[7] Michael R.A. Huth, Mark D. Ryan, *Logic in Computer Science: Modelling and Reasoning about Systems*, Cambridge University Press, Second Edition (2004).

[8] Marco Roveri, Personal communication (2007).

[9] Stuart Russell, Peter Norvig, *Artificial Intelligence. A Modern Approach*, Prentice Hall, Second Edition (2003).

[10] Yan Zhang, Yulin Ding, CTL model update for system modifications, *Journal of Artificial Intelligence Research*, **31** (2008), 113-155.

9. Appendix A: Proofs

Theorem 13. (Soundness) *If $H \in TSystems$, $\varphi \in$ **XCTL**, $s \in States$, and t is a protection such that $\mathcal{M}_H$ **Con** t, then $\forall (H', t') \in TUpdate^X(H, t, s, \varphi)$: $\mathcal{M}_{H'} \models_s \varphi$ and $\mathcal{M}_{H'}$ **Con** t, t'.*

Proof. (Structural induction on φ.)

1. For the case $\varphi \equiv \bot$ we have $TUpdate^X(H, t, s, \varphi) = \emptyset$, so there is nothing to prove.

2. If $\varphi \equiv \top$, $TUpdate^X(H, t, s, \top) = \{(H, t)\}$ and therefore
$(H', t') \in TUpdate^X(H, t, s, \varphi)$ entails $H' = H \ \& \ t' = t$.

Thus $\mathcal{M}_{H'} \models_s \top$ and $\mathcal{M}_{H'} = \mathcal{M}_H$ **Con** t, t'.

3. For $\varphi \equiv \ell$ we have two possible cases,

$$TUpdate^X(H, t, s, \ell) = \{(H, t)\} \text{ (if } \ell \in L(s)),$$
$$\text{or } TUpdate^X(H, t, s, \ell) = \emptyset \text{ (if } \ell \notin L(s));$$

for the second one there is nothing to prove, and for the first one we have:

$$(H', t') \in TUpdate^X(H, t, s, \ell) \Rightarrow \mathcal{M}_{H'} = \mathcal{M}_H \ \& \ t' = t,$$
$$\therefore \mathcal{M}_{H'} \models_s \ell \iff \ell \in L^{\mathcal{M}_{H'}}(s) \iff s \models \ell \iff \ell \in L(s).$$
$$\therefore \mathcal{M}_{H'} \models_s \ell \ \& \ \mathcal{M}_{H'} \textbf{ Con } t, t'.$$

4. If $\varphi \equiv \alpha \vee \beta$, by the induction hypothesis for α and β, we have
$\forall (H', t') \in TUpdate^X(H, t, s, \alpha) : \ \mathcal{M}_{H'} \models_s \alpha \ \& \ \mathcal{M}_{H'}$ **Con** t, t'
and
$\forall (H', t') \in TUpdate^X(H, t, s, \beta) : \ \mathcal{M}_{H'} \models_s \beta \ \& \ \mathcal{M}_{H'}$ **Con** t, t'.

Hence, if $(H', t') \in TUpdate^X(H, t, s, \varphi) = TUpdate^X(H, t, s, \alpha) \cup TUpdate^X(H, t, s, \beta)$, then

$$(H', t') \in TUpdate^X(H, t, s, \alpha) \text{ or } (H', t') \in TUpdate^X(H, t, s, \beta),$$
$$\therefore (\mathcal{M}_{H'} \models_s \alpha \text{ or } \mathcal{M}_{H'} \models_s \beta) \text{ and } \mathcal{M}_{H'} \textbf{ Con } t, t',$$
$$\therefore \mathcal{M}_{H'} \models_s \alpha \vee \beta \text{ and } \mathcal{M}_{H'} \textbf{ Con } t, t'.$$

5. If $\varphi \equiv \alpha \wedge \beta$, then

$$TUpdate^X(H, t, s, \varphi) = TUpdate^{X*}(TUpdate^X(H, t, s, \alpha), s, \beta)$$
$$= \bigcup_{(H', t') \in A} TUpdate^X(H', t', s, \beta),$$

where $A = TUpdate^X(H, t, s, \alpha)$. Thus, if $(H_\varphi, t_\varphi) \in TUpdate^X(H, t, s, \alpha \wedge \beta)$,

then

$$\exists (H', t') \in \mathit{TUpdate}^X(H, t, s, \alpha) : (H_\varphi, t_\varphi) \in \mathit{TUpdate}^X(H', t', s, \beta).$$

Let $(H_\alpha, t_\alpha) \in \mathit{TUpdate}^X(H, t, s, \alpha)$ with this property. By the induction hypothesis for H, t, s, α, we have:

$$\forall (H', t') \in \mathit{TUpdate}^X(H, t, s, \alpha) : \ \mathcal{M}_{H'} \models_s \alpha \ \&\ \mathcal{M}_{H'} \ \mathbf{Con}\ t, t',$$
$$\therefore \mathcal{M}_{H_\alpha} \models_s \alpha \ \&\ \mathcal{M}_{H_\alpha} \ \mathbf{Con}\ t, t_\alpha.$$

And by the induction hypothesis for $H_\alpha, t_\alpha, s, \beta$, we have:

$$\forall (H', t') \in \mathit{TUpdate}^X(H_\alpha, t_\alpha, s, \beta) : \ \mathcal{M}_{H'} \models_s \beta \ \&\ \mathcal{M}_{H'} \ \mathbf{Con}\ t_\alpha, t',$$
$$\therefore \mathcal{M}_{H_\varphi} \models_s \beta \ \&\ \mathcal{M}_{H_\varphi} \ \mathbf{Con}\ t_\alpha, t_\varphi.$$

Note that: if $\mathcal{M}_{H_\varphi}$ **Con** t_α and $(H_\alpha, t_\alpha) \in \mathit{TUpdate}^X(H, t, s, \alpha)$, then $\mathcal{M}_{H_\varphi} \models_s \alpha$.

$$\therefore \mathcal{M}_{H_\varphi} \models_s \alpha \wedge \beta \ \&\ \mathcal{M}_{H_\varphi} \ \mathbf{Con}\ t_\alpha, t_\varphi.$$

6. If $\varphi \equiv \mathbf{EX}\,\alpha$, then

$$\mathit{TUpdate}^X(H, t, s, \varphi) = \bigcup_{s' \in \mathit{States}} \mathit{TUpdate}^{X*}(\mathit{TAdd}_e(H, t, s, s'), t, s', \alpha)$$
$$\therefore \text{ if } (H'', t'') \in \mathit{TUpdate}^X(H, t, s, \mathbf{EX}\,\alpha) \text{ then}$$
$$\exists s' \in \mathit{States} : (H'', t'') \in \mathit{TUpdate}^{X*}(\mathit{TAdd}_e(H, t, s, s'), t, s', \alpha).$$

But

$$\mathit{TUpdate}^{X*}(\mathit{TAdd}_e(H, t, s, s'), t, s', \alpha) = \bigcup_{(H', t') \in A} \mathit{TUpdate}^X(H', t', s', \alpha)$$
$$\text{where } A = \mathit{TAdd}_e(H, t, s, s')$$

Therefore, if $(H'', t'') \in \mathit{TUpdate}^{X*}(\mathit{TAdd}_e(H, t, s, s'), t, s', \alpha)$ then

$$\exists (H', t') \in \mathit{TAdd}_e(H, t, s, s') : (H'', t'') \in \mathit{TUpdate}^X(H', t', s', \alpha).$$

Let $(H', t') \in \mathit{TAdd}_e(H, t, s, s')$ with this property. Since

$$\mathit{TAdd}_e(H, t, s, s') = \mathit{TUpdateSts}(\{(H[s, s'], t[s, s'])\}, \mathit{next}_{H[s,s']}(s), \bigwedge\{\psi \mid (s, \psi) \in S^t\}),$$

and given that the ψ's are subformulas of φ, we can apply the induction hypothesis. Therefore, given that $(H', t') \in \mathit{TAdd}_e(H, t, s, s')$,
$\mathcal{M}_{H'}$ **Con** $t', t[s, s'],\ \therefore (s, s') \in R^{\mathcal{M}_{H'}}, R^{t'}$.

Thus, by the induction hypothesis, if $(H'', t'') \in TUpdate^X(H', t', s', \alpha)$, then $\mathcal{M}_{H''} \models_{s'} \alpha$ and $\mathcal{M}_{H''}$ **Con** t'', t'.

$\therefore \mathcal{M}_{H''} \models_{s'} \alpha$ and $(s, s') \in R^{t'} \subseteq R^{\mathcal{M}_{H''}}$,

$\therefore \mathcal{M}_{H''} \models_{s} \mathbf{EX}\,\alpha$ and $\mathcal{M}_{H''}$ **Con** t'', t'.

7. If $\varphi \equiv \mathbf{AX}\,\alpha$, then

$$TUpdate^X(H, t, s, \varphi) = \begin{cases} TUpdate^X(H[R^t, s], t[s, \alpha], s, \\ \quad \mathbf{EX}\,\alpha) & \text{if } Suc(R^t, s) = \emptyset\,, \\ TUpdateSts(\{(H[R^t, s], t[s, \alpha])\}, \\ \quad Suc(R^t, s), \alpha) & \text{otherwise}\ . \end{cases}$$

Therefore, if $(H', t') \in TUpdate^X(H[R^t, s], t[s, \alpha], s, \mathbf{EX}\,\alpha)$, then $\mathcal{M}_{H'} \models_s \mathbf{EX}\,\alpha$ and $\mathcal{M}_{H'}$ **Con** $t', t[s, \alpha]$.

But, if $Suc(R^t, s) = \emptyset$ then s has no successors in $H[R^t, s]$, and α holds at all the successors (only one) added to $H[R^t, s]$ by $TUpdate^X(H[R^t, s], t[s, \alpha], s, \mathbf{EX}\,\alpha)$.

$\therefore \mathcal{M}_{H'} \models_s \mathbf{AX}\,\alpha$ and $\mathcal{M}_{H'}$ **Con** $t', t[s, \alpha]$.

$\therefore \mathcal{M}_{H'} \models_s \mathbf{AX}\,\alpha$ and $\mathcal{M}_{H'}$ **Con** t', t.

If $Suc(R^t, s) \neq \emptyset$ then the update of φ amounts to the update of the conjunction of all the protected successors of s.

We conclude that the theorem holds. □