Model Updating in Action

Maria V. de Menezes and **Leliane N. de Barros**
Department of Computer Science
IME-USP

Silvio do L. Pereira
Department of Information Technology
FATEC-SP\CEETEPS

Abstract

Model updating is a formal approach to automatically correct a system model $\mathcal{M}$ with respect to some property φ not satisfied by $\mathcal{M}$. The well known model updating approaches are based on Computational Tree Logic (CTL), a branch time temporal logic which does not take into account the actions behind the state transitions. In previous work we have proposed a model checker and a planner based on α-CTL – a temporal logic whose semantics is based on actions – to solve extended reachability goals. In this paper, we present a model updating approach based on α-CTL that can be used to automatically suggest modifications in a state transition model induced by a set of actions and also is able to suggest changes directly in the action specification.

Introduction

Errors are common during the design of systems and their late detection and correction can be one of the major reasons for a high cost design. However, it can be reduced if the designer is able to early detect them, i.e., during system specification. By using formal methods to specify a system behavior, we can apply *model checking* techniques (Müller-Olm, Schmidt, and Steffen 1999) to automatically detect not met requirements. In order to understand how model checking works, let us consider the well-known *microwave oven* scenario presented by (Clarke E. 1999), which represents two main microwave usage processes: food heating and cooking.

The system designer starts by defining which properties will be used to describe the current state of a system. In the microwave oven example, the state properties are: *started* (indicating the microwave is operating), *closed* (indicating the microwave door is closed), *heated* (indicating the food inside microwave oven is heated) and *cooked* (describing that the food is cooked). Those properties are propositional atoms used to describe what is true in the system state and, their negation, describes what is false. Additionally, a state property *error* indicates the error detected during system operation (in our example, an error occurs in the situation where the oven starts and the door is open). Furthermore, system designer has to specify the actions that cause state transitions, that are: *start*, *finish*, *open-door*, *close-door*, *warm up*, *cook* and *reset*. Figure 1 shows a preliminary design model of a microwave oven given by a state transition diagram.

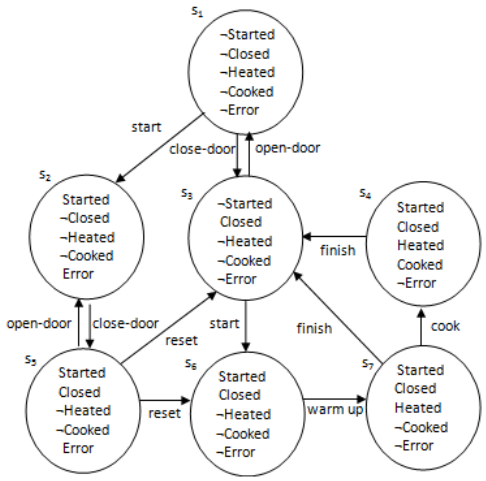


Figure 1: Formal model of a microwave oven, adapted from (Clarke E. 1999).

In the initial state s_1, a user can select one of the two actions: *start* and *close-door*. The *start* action takes the system to state s_2, which indicates the existence of an error. Notice that an error will persist even if the action *close-door* is selected (transition from s_2 to s_5). In this situation, the user has to reset (action *reset*) the microwave which can take the system to one of the two states: s_3 and s_6, both without error (note that *reset* is a non-deterministic action: an action with uncertain effects). If the *close-door* action is selected in the initial state s_1, the error does not occur and the food inside the microwave oven can be heated and\or cooked without making the user to reset the oven.

Suppose the designer wants to verify if the system specification in Figure 1 satisfies the temporal formula φ defined as: " *once the microwave oven is started, the food inside will be heated in some future state*". That means, φ is satisfied in a system model with no state where both *started* and ¬*heated* properties are true. The paths $[s_1, s_2, s_5, s_3, s_1, \cdots]$, $[s_1, s_2, s_5, s_2, \cdots]$ are examples where φ is not satisfied. The rationale behind φ can be "*the user should not have to reset*

Figure 2: Model checker.

the oven in order to be able to heat and\or cook his food".

Although the microwave example seems to be easy to model and verify temporal properties, having defined 5 state properties (fluents or state variables) implies in 32 (2^5) possible states and 1024 (2^{10}) possible state transitions. Thus, to design a system that allows a user to achieve its intended goals (e.g. to heat or cook a meal) and yet guaranteeing to hold some temporal properties, is a complex task that gets harder with the size of state space (i.e. the number of state variables).

Model checking consists of automatically solving the problem $\mathcal{K} \vDash \varphi$, where $\mathcal{K}$ is a formal model of a system and φ is a formal specification of a temporal property to be verified in this system. Essentially, a model checker (Figure 2) is an algorithm that receives a pair $(\mathcal{K}, \varphi)$ as input and systematically visits the states of the model $\mathcal{K}$, in order to verify if the property φ holds. When all states in $\mathcal{K}$ satisfy property φ, the model checker returns success; otherwise, it returns a counter-example (e.g., a state in the model $\mathcal{K}$ where the property φ is violated). One of the limitations of model checking is that, when the property is violated, it only returns a counter-example leaving the task of modifying the system model to the designer.

Model updating is a technique that extends model checking functions in order to support the repair of a faulty system. Zhang and Ding(2008) proposed a model update algorithm that takes a given *Kripke model* $\mathcal{K}$ (Kripke 1963) - a state transition model without action specification - with respect to an arbitrary CTL formula φ and generates an updated model $\mathcal{K}'$ that: (1) satisfies φ and (2) has a minimal change with respect to the original model. To generate $\mathcal{K}'$, this approach uses *primitive update operations* such as: *add a relation element*, *remove a relation element*, *change labelling function on one state*, *add a state* and *remove a state*. For example, a possible correction in the model of Figure 1 is "*remove the transition between the states s_1 and s_2*". This means that "*It is not allowed to start the microwave oven with the opened door*".

Motivation: KE for planning vs. model updating

The microwave formal model from Figure 1 can be seen as a set of plans, specified by the system designer to achieve the goals: heat and\or cook a meal. Each plan is supposed to be executed by an user (according with a "system manual").

In artificial intelligence planning area, the task is to automatically generate a plan of actions given a goal specification and a system model (e.g., a factory or a robot environment model). A planning domain is specified in terms of a set of action schema which can be used to induce the system model. Nevertheless, it is very difficult, even for a simple planning domain, to specify a correct set of actions. Although automatic planning has been the subject of extensive study in the AI community since the early 1970s, low effort has been given to the task of modeling and verification of planning domains.

In order to model and verify a planning domain, a designer should start by describing a preliminary set of actions to be further refined. This refinement can be done using, e.g.: (i) a set of plan examples, for a given class of goals, specified in an ad hoc way by a domain expert (or possibly generated by an automatic planner); (ii) a state transition model induced by the semantics of the preliminary set of action schema for a small problem.

Notice that in the system model represented by Figure 1 the state transitions are labelled with actions. Traditional model updating approaches based on CTL (e.g (Zhang and Ding 2008)) do not take into account the actions behind the transitions and therefore can not be applied to update (refine) planning domains. Actions are not part of a CTL *Kripke* model, which is the formalism used in most of the model checking and updating approaches (Buccafurri et al. 1999; Harris and Ryan 2003; Zhang and Ding 2008). In this paper we show that by representing actions in the formal model of a system we can extend model updating techniques to be used as an important support tool for knowledge acquisition, modeling and verification of planning domains.

Since actions are not part of *Kripke* structure, we can only represent them using a temporal logic whose semantics is based on actions. Pereira and Barros (2008) proposed an extension of CTL, called α-CTL, whose semantics considers the transition actions. They have also developed a model checker based on this logic, named α-CTL *model checker*. This work presents a model updating approach based on α-CTL that can be used to automatically suggest modifications in a state transition model induced by a set of actions and therefore is able to suggest changes directly in the actions specification. We also define a criterion of minimal change for α-CTL model updating.

The remainder of this article is organized as follows: we first show the basic concepts of CTL model checking and CTL model updating; then we define a labelled transition system and a model checker based on α-CTL . Finally, we show how to perform model update, based on α-CTL , that can suggest modification in the set of actions of a planning domain.

CTL Model Checking and Update

In this section, we introduce the basic concepts of CTL model checking and CTL model update.

The branching time temporal logic CTL (COMPUTATION TREE LOGIC) (Clarke and Emerson 1982) allows us to reason about alternative time lines (i.e., alternative futures). In CTL the temporal operators must be preceded by some quantifier: $\exists$ (Figure 3) or $\forall$ (Figure 4).

- $\forall\bigcirc$ (in *all next* states)
- $\exists\bigcirc$ (at *some next* state)
- $\forall\Box$ (*invariant*, in *all future* state)
- $\exists\Box$ (*invariant*, at *some* future state)
- $\forall\diamond$ (*finally*, in *all* future states)
- $\exists\diamond$ (*finally*, at *some* the future state)

- $\forall[\varphi_1 \sqcup \varphi_2]$ (*since*, in *all* future states)
- $\exists[\varphi_1 \sqcup \varphi_2]$ (*since*, at *some* future state)

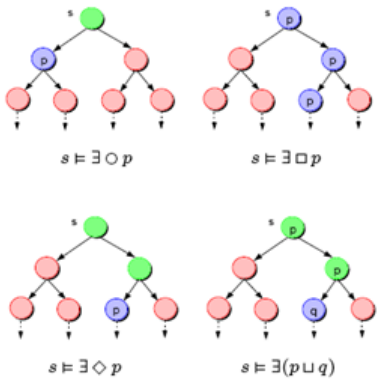


Figure 3: Semantics of the CTL temporal operators preceded by existential quantifier.

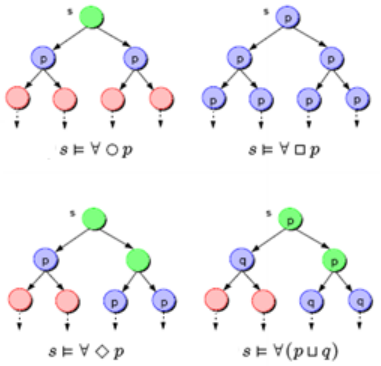


Figure 4: Semantics of the CTL temporal operators preceded by universal quantifier.

The CTL formulas are composed by atomic propositions, propositional operators and temporal operators. The symbols $\circ$ (next), $\Box$ (invariantly), $\Diamond$ (finally) and $\sqcup$ (until), combined with the quantifiers $\exists$ and $\forall$, are used to compose the temporal operators of this logic.

The syntax of CTL is inductively defined as:
$\varphi \doteq p \in \mathbb{P} \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \exists \circ \varphi_1 \mid \forall \circ \varphi_1 \mid \exists \Box \varphi_1 \mid \forall \Box \varphi_1 \mid \exists(\varphi_1 \sqcup \varphi_2) \mid \forall(\varphi_1 \sqcup \varphi_2)$.

The temporal operators $\exists\Diamond$ and $\forall\Diamond$ are defined as:

- $\exists \Diamond \varphi \doteq \exists(\top \sqcup \varphi)$
- $\forall \Diamond \varphi \doteq \forall(\top \sqcup \varphi)$

The semantics of CTL is defined over a *Kripke structure* $\mathcal{K} = \langle S, L, T \rangle$, where S is a set of states, $L : S \mapsto 2^{\mathbb{P}}$ is a state labelling relation and $T \subseteq S \times S$ is a transition relation. A path in $\mathcal{K}$ is a sequence of states $[s_0, s_1, \cdots]$ such that $s_i \in S$ and $(s_i, s_{i+1}) \in T$, for all $i \geqslant 0$.

Given a *Kripke structure* $\mathcal{K}$ and a state $s_0 \in S$, the CTL satisfiability relation is defined as:

Figure 5: Model updater receives $\mathcal{K}$ and φ and returns an updated model $\mathcal{K}$'

- $(\mathcal{K}, s_0) \vDash p$ iff $p \in L(s_0)$;
- $(\mathcal{K}, s_0) \vDash \neg\varphi$ iff $(\mathcal{K}, s_0) \nvDash \varphi$;
- $(\mathcal{K}, s_0) \vDash \varphi_1 \wedge \varphi_2$ iff $(\mathcal{K}, s_0) \vDash \varphi_1$ and $(\mathcal{K}, s_0) \vDash \varphi_2$;
- $(\mathcal{K}, s_0) \vDash \varphi_1 \vee \varphi_2$ iff $(\mathcal{K}, s_0) \vDash \varphi_1$ or $(\mathcal{K}, s_0) \vDash \varphi_2$;
- $(\mathcal{K}, s_0) \vDash \exists \circ \varphi$ iff for some path $[s_0, s_1, \cdots]$ in $\mathcal{K}$, $(\mathcal{K}, s_1) \vDash \varphi$;
- $(\mathcal{K}, s_0) \vDash \forall \circ \varphi$ iff for every path $[s_0, s_1, \cdots]$ in $\mathcal{K}$, $(\mathcal{K}, s_1) \vDash \varphi$;
- $(\mathcal{K}, s_0) \vDash \exists \Box \varphi$ iff for some path $[s_0, s_1, \cdots]$ in $\mathcal{K}$, for $i \geq 0$, $(\mathcal{K}, s_i) \vDash \varphi$;
- $(\mathcal{K}, s_0) \vDash \forall \Box \varphi$ iff for every path $[s_0, s_1, \cdots]$ in $\mathcal{K}$, for $i \geq 0$, $(\mathcal{K}, s_i) \vDash \varphi$;
- $(\mathcal{K}, s_0) \vDash \exists(\varphi_1 \sqcup \varphi_2)$ iff for some path $[s_0, s_1, \cdots]$ in $\mathcal{K}$, there exists $i \geq 0$ such that $(\mathcal{K}, s_i) \vDash \varphi_2$ and, for $0 \leq j < i$, $(\mathcal{K}, s_j) \vDash \varphi_1$;
- $(\mathcal{K}, s_0) \vDash \forall(\varphi_1 \sqcup \varphi_2)$ iff for every path $[s_0, s_1, \cdots]$ in $\mathcal{K}$, there exists $i \geq 0$ such that $(\mathcal{K}, s_i) \vDash \varphi_2$ and, for $0 \leq j < i$, $(\mathcal{K}, s_j) \vDash \varphi_1$.

Model update framework

Let $\mathcal{K}$ be a formal model of a system and φ be a formal specification of a property that is not satisfied in this system, i.e., $\mathcal{K} \nvDash \varphi$. *Model update* (Zhang and Ding 2008) consists of generating a new model $\mathcal{K}'$ that satisfies the input formula ($\mathcal{K}' \vDash \varphi$) and has a minimal change with respect to the original model $\mathcal{K}$. Then, a *model updater* (Figura 5) is an algorithm that receives a pair $(\mathcal{K}, \varphi)$, where $\mathcal{K} \nvDash \varphi$, and returns a new model $\mathcal{K}'$, where $\mathcal{K}' \vDash \varphi$. The updated model $\mathcal{K}'$ can be viewed as a possible correction on the original system specification.

Zhang and Ding (2008) proposed a formal framework for CTL model update, defining primitive operations and specifying a minimal change principle for CTL model updating. Below, we list the Zhang and Ding (2008) primitive operations:

PU1: Adding one relation element. Let be $\mathcal{K} = \langle S, L, T \rangle$, its updated model $\mathcal{K}' = \langle S', L', T' \rangle$ is obtained from $\mathcal{K}$ by adding only one new relation element. That is, $S' = S, L' = L, T' = T \cup (s_i, s_j)$, where $s_i, s_j \in S$ and $(s_i, s_j) \notin T$.

PU2: Removing one relation element. Let be $\mathcal{K} = \langle S, L, T \rangle$, its updated model $\mathcal{K}' = \langle S', L', T' \rangle$ is obtained from $\mathcal{K}$ by removing only one existing relation element. That is, $S' = S, L' = L, T' = T - (s_i, s_j)$, where $(s_i, s_j) \in T$ for two states $s_i, s_j \in S$.

PU3: Changing labelling function on one state. Let be $\mathcal{K} = \langle S, L, T\rangle$, its updated model $\mathcal{K}' = \langle S', L', T'\rangle$ is obtained from $\mathcal{K}$ by changing labelling function on a particular state. That is, $S' = S, T' = T, \forall s \in (S - s^*), s^* \in S, L'(s) = L(s)$ and $L'(s^*)$ is a set of true variable assigned in s^* where $L'(s^*) \neq L(s^*)$.

PU4: Adding one state. Let be $\mathcal{K} = \langle S, L, T\rangle$, its updated model $\mathcal{K}' = \langle S', L', T'\rangle$ is obtained from $\mathcal{K}$ by adding only one new state. That is, $S' = S \cup s^*, s^* \notin S$, $T' = T$ and $\forall s \in S, L'(s) = L(s)$ and $L'(s^*)$ is a set of true variables assigned in s^*.

PU5: Removing one isolated state. Let be $\mathcal{K} = \langle S, L, T\rangle$, its updated model $\mathcal{K}' = \langle S', L', T'\rangle$ is obtained from $\mathcal{K}$ by removing only one isolated state. That is, $S' = S - s^*$, where $s^* \in S$ and $\forall s \in S$ such that $s \neq s^*$, neither (s, s^*) nor (s^*, s) is not in T, $T' = T$ and $\forall s \in S', L'(s) = L(s)$.

Model update should obey minimal change rules. Although, the primitive operations PUi where $i = 1, 2, ..5$ can be used to define minimal change criterion for CTL model update, within this framework it is not possible to make updates considering the actions of the system specification, as we propose in the next sections.

Labelled transition system

Let $\mathbb{P} \neq \varnothing$ be a finite set of atomic propositions, denoting properties of a system, and $\mathbb{A} \neq \varnothing$ be a finite set of actions, representing the events of a system.

Definition 1. *A labelled transition system with signature $(\mathbb{P}, \mathbb{A})$ is defined by $\mathcal{M} = \langle S, L, T\rangle$, where:*

- $S \neq \varnothing$ *is a finite set of states;*
- $L : \mathcal{S} \mapsto 2^{\mathbb{P}}$ *is a state labelling function;*
- $T : S \times \mathbb{A} \times S$ *is a state transition relation.*

A labelled transition system with signature $(\mathbb{P}, \mathbb{A})$ can be represented as a *transition graph*, where states are labelled with subsets of $\mathbb{P}$ and transitions are labelled with elements of $\mathbb{A}$. Set $\mathcal{S}$ has all possible states of a model, state labelling function designs for each state $s \in S$ a proposition set $L(s) \in 2^{\mathbb{P}}$ and labelling function T designs for each transition $t \in T$ an action $a \in \mathbb{A}$. Given two states $s_i, s_j \in S$ and an action $a \in \mathbb{A}$, a transition between s_i and s_j is represented by $(s_i, a, s_j) \in T$.

α-CTL model checking

An example where representing actions may allow for a more rational model checking and updating is shown in Figure 6. Suppose that φ is a desired property: "*from the initial state s_0, all transitions take to state in which p is true*". A traditional model checker (MC) would represent the system model by Figure 6(a), i.e., by a *Kripke* model. Since the transition (s_0, s_2) does not satisfy φ, the CTL based MC would detect this error and a model update would indicate " *remove the relation* (s_0, s_2)". However, if we do represent the transition actions (Figure 6(b)) a model checker would not detect an error, since there is an action a that takes the system to state s_2 that satisfies φ. Figure 6(c) shows another example of how model checking can be more rational when we represent the actions in the state transition model: by knowing that the two transitions correspond to non-deterministic effects of action a, removing one transition implies removing the other.

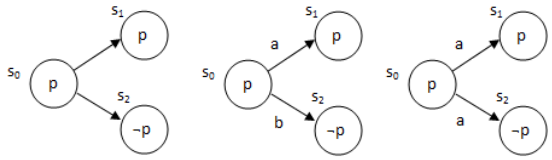


Figure 6: (a) Kripke structure. (b) and (c) Labelled transition model.

In this section, we present the branching time temporal logic α-CTL and a model checker based on this new logic.

The new temporal logic α-CTL

Differently from CTL, the branching time temporal logic α-CTL, proposed in (Pereira and de Barros 2008), can discern among various actions that produce state transitions.

Syntax of α-CTL In CTL, a formula $\forall \circ \varphi$ holds on a state s if and only if it holds on all successors of s, independently of the actions labeling the transitions from s to its successors. In α-CTL , to enforce that actions play an important role in its semantics, we use a different set of "dotted" symbols to represent temporal operators: $\odot$ (next), $\boxdot$ (invariantly), $\diamondsuit$ (finally) and $\sqcup$ (until).

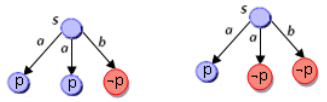
Definition 2. *Let $p \in \mathbb{P}$ be an atomic proposition. The syntax of α-CTL is inductively defined as:*

$\varphi ::= p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \exists \odot \varphi \mid \forall \odot \varphi \mid \exists \boxdot \varphi \mid \forall \boxdot \varphi \mid \exists(\varphi_1 \sqcup \varphi_2) \mid \forall(\varphi_1 \sqcup \varphi_2)$

According to the α-CTL syntax, well-formed formulas are in *negative normal form*, where the scope of negation is restricted to the atomic propositions. Furthermore, all temporal operators are prefixed by a path quantifier ($\exists$ or $\forall$). The temporal operators derived from $\diamondsuit$ are defined as: $\exists \diamondsuit \varphi_2 \doteq \exists(\top \sqcup \varphi_2)$ and $\forall \diamondsuit \varphi_2 \doteq \forall(\top \sqcup \varphi_2)$.

Semantics of α-CTL Let $\mathbb{P} \neq \varnothing$ be a finite set of atomic propositions and $\mathbb{A} \neq \varnothing$ be a finite set of actions. An α-CTL *temporal model* over $(\mathbb{P}, \mathbb{A})$ is a transition graph where states are labelled with subsets of $\mathbb{P}$ and transitions are labelled with elements of $\mathbb{A}$.

Intuitively, a state s in a *temporal model* $\mathcal{M}$ satisfies a formula $\forall \odot \varphi$ (or $\exists \odot \varphi$) (Figure 7) if there exists an action α that, when executed in s, *necessarily* (or *possibly*) reaches an immediate successor of s which satisfies the formula φ. In other words, the modality $\odot$ represents the set of α-successors of s, for some particular action $\alpha \in \mathbb{A}$; the quantifier $\forall$ requires that all these α-successors satisfy φ; and quantifier $\exists$ requires that some of these α-successors satisfy φ.



(a) $s \models \forall \odot p$ (b) $s \models \exists \odot p$

Figure 7: Semantics of the temporal operator $\odot$. (a) p is true for all sucessors of s troughout action a. (b) p is true for some sucessor of s troughout action a.

Before we can give a formal definition of the α-CTL semantics, we need to define the concept of *preimage* of a set of states.

Definition 3. *Let be $Y \subseteq S$ a set of states. The weak preimage of Y, denoted by $\mathcal{I}^-_\exists(Y)$, is a set $\{s \in S : a \in \mathbb{A}$ and $\exists(s,a,s') \in T, s' \in S, s' \subseteq Y\}$; and the strong preimage of Y, denoted by $\mathcal{I}^-_\forall(Y)$, is the set $\{s \in \mathcal{S} : a \in \mathbb{A}$ and $\forall(s,a,s') \in T, s' \in S, s' \subseteq Y\}$*

The semantics of the global temporal operators ($\exists\boxdot$, $\forall\boxdot$, $\exists\sqcup$ and $\forall\sqcup$) is derived from the semantics of the local temporal operators ($\exists\odot$ and $\forall\odot$), by using least (μ) and greatest (ν) fixpoint operations.

Definition 4. *Let $\mathcal{M} = \langle S, L, T\rangle$ be a temporal model with signature $(\mathbb{P}, \mathbb{A})$ and $p \in \mathbb{P}$ be an atomic proposition. The intention of an α-CTL formula φ in $\mathcal{M}$ (or the set of states satisfying φ in $\mathcal{M}$, denoted by $[\![\varphi]\!]_\mathcal{M}$, is defined as:*

- $[\![p]\!]_\mathcal{M} = \{s \in S : p \in L(s)\}$
- $[\![\neg p]\!]_\mathcal{M} = S \setminus [\![p]\!]_\mathcal{M}$
- $[\![\varphi_1 \wedge \varphi_2]\!]_\mathcal{M} = [\![\varphi_1]\!]_\mathcal{M} \cap [\![\varphi_2]\!]_\mathcal{M}$
- $[\![\varphi_1 \vee \varphi_2]\!]_\mathcal{M} = [\![\varphi_1]\!]_\mathcal{M} \cup [\![\varphi_2]\!]_\mathcal{M}$
- $[\![\exists \odot \varphi_1]\!]_\mathcal{M} = \mathcal{T}^-_\exists([\![\varphi_1]\!]_\mathcal{M})$
- $[\![\forall \odot \varphi_1]\!]_\mathcal{M} = \mathcal{T}^-_\forall([\![\varphi_1]\!]_\mathcal{M})$
- $[\![\exists \boxdot \varphi_1]\!]_\mathcal{M} = \nu Y.([\![\varphi_1]\!]_\mathcal{M} \cap \mathcal{T}^-_\exists(Y))$
- $[\![\forall \boxdot \varphi_1]\!]_\mathcal{M} = \nu Y.([\![\varphi_1]\!]_\mathcal{M} \cap \mathcal{T}^-_\forall(Y))$
- $[\![\exists(\varphi_1 \sqcup \varphi_2)]\!]_\mathcal{M} = \mu Y.([\![\varphi_2]\!]_\mathcal{M} \cup ([\![\varphi_1]\!]_\mathcal{M} \cap \mathcal{T}^-_\exists(Y)))$
- $[\![\forall(\varphi_1 \sqcup \varphi_2)]\!]_\mathcal{M} = \mu Y.([\![\varphi_2]\!]_\mathcal{M} \cup ([\![\varphi_1]\!]_\mathcal{M} \cap \mathcal{T}^-_\forall(Y)))$

A model checker for α-CTL

A model checker for α-CTL can be directly implemented from its semantics. Given a model $\mathcal{M} = \langle S, L, T\rangle$ and an α-CTL formula φ, the model checker computes the set C of states that do not satisfy the formula φ in $\mathcal{M}$; then, if C is the empty set, it returns success; otherwise, it returns C as counter-example.

α – MODELCHECKER$(\varphi, \mathcal{M})$
1 $C \leftarrow S \setminus$ INTENTION$(\varphi, \mathcal{M})$
2 if $C = \varnothing$ then return success
3 else return C

The basic operation on this model checker is implemented by the function INTENTION, that inductively computes the intention of the formula φ in the model $\mathcal{M}$. The efficiency of α – MODELCHECKER can be highly improved with use of BDDs (Bryant 1992), resulting in an extremely efficient symbolic version of this model checker. More details about the α-MODELCHECKER can be found in (Pereira and de Barros 2008).

α-CTL model updating

In this section, we define the basic concepts about the proposed model updating system, which is based on the branching time temporal logic α-CTL (Pereira and de Barros 2008).

First, consider that a *complete model* $\mathcal{M}^\star = \langle S^\star, L^\star, T^\star\rangle$ (eventually induced by a formal specification in $\mathbb{A}$) is a labelled transition system with signature $(\mathbb{P}, \mathbb{A})$. Thus, according to Definition 1:

- $S^\star \neq \varnothing$ is a finite set of states;
- $L^\star : S^\star \mapsto 2^\mathbb{P}$ is a state labelling function;
- $T^\star \subseteq S^\star \times \mathbb{A} \times S^\star$ is a state transition relation.

Moreover, consider that the model which the system designer wants to correct is a *partial model* $\mathcal{M} = \langle S, L, T\rangle$ such that $\mathcal{M} \subseteq \mathcal{M}^\star$ or, more precisely:

- $S \subseteq S^\star$ is a finite set of states;
- $L : S \mapsto 2^\mathbb{P}$ such that, for all $s \in S, L(s) = L^\star(s)$;
- $T \subseteq T^\star$ such that, if $(s_i, a, s_j) \in T$ and $(s_i, a, s_k) \in T^\star$, then $(s_i, a, s_k) \in T$.

In Figure 8, we have a labelled transition system representing a complete model $\mathcal{M}^\star$, where the highlighted substructure is the partial model $\mathcal{M} \subseteq \mathcal{M}^\star$ given by system designer.

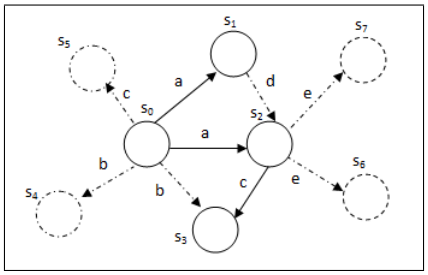


Figure 8: A labelled transition system. Solid lines represent the partial model $\mathcal{M}$ and dashed lines represent the complete model $\mathcal{M}^\star$.

Given a partial model $\mathcal{M} \subseteq \mathcal{M}^\star$, a initial state $s_0 \in S$ and an α-CTL formula φ such that $(\mathcal{M}, s_0) \not\models \varphi$, the α-CTL *model updating problem* consists of generating a new partial model $\mathcal{M}' \subseteq \mathcal{M}^\star$, called *updated model*, such that: (*i*) $(\mathcal{M}', s_0) \models \varphi$ and (*ii*) $\mathcal{M}'$ has a minimal change with respect to the original partial model $\mathcal{M}$.

To update a partial model $\mathcal{M}$, w.r.t. a complete model $\mathcal{M}^\star$, in order to satisfy an α-CTL formula φ, we define the follow primitive operations:

PUA1: Adding transitions induced by an action. Given a partial model $\mathcal{M} = \langle S, L, T\rangle$, a corresponding updated model $\mathcal{M}' = \langle S', L', T'\rangle$, with respect to $\mathcal{M}^\star$, can be obtained from $\mathcal{M}$ by adding a transition between states $s_i, s_j \in S$. In other words:

- $T' = T \cup \{(s_i, a, s) \in T^\star : \exists a \in \mathbb{A}, (s_i, a, s_j) \in T^\star\}$
- $S' = S \cup \{s : \exists a \in \mathbb{A}, (s_i, a, s) \in T'\}$
- $L'(s) = L^\star(s), s \in S'$

Adding a transition between two states s_i and s_j in the partial model is possible only if there is some transition between these states in the complete model. For example, in Figure 8 we cannot add a transition between states s_2 and s_4 in the partial model (solid lines), because there is no transition between s_2 and s_4 in the complete model (dashed lines). In Figure 8, it is possible to add a transition between states s_0 and s_3, since a transition labelled with action b exists in the complete model. However, all the effects of action b must also be added. In Figure 8 in order to add a transition between states s_0 and s_3, using action b, we must also add state s_4 and the transition between s_0 and s_4 in the updated model.

PUA2: Removing transitions induced by an action. Given a partial model $\mathcal{M} = \langle S, L, T\rangle$, a corresponding updated model $\mathcal{M}' = \langle S', L', T'\rangle$, with respect to $\mathcal{M}^\star$, can be obtained from $\mathcal{M}$ by removing an existing transition between states $s_i, s_j \in S$, which is labelled by some action $a \in \mathbb{A}$. More formally:

- $T' = T - \{(s_i, a, s) \in T^\star : (s_i, a, s_j) \in T\}$
- $S' = S$
- $L'(s) = L(s), s \in S'$

It is important to observe that to remove an existing transition between states s_i and s_j labelled by an action a, we must also remove all transitions from s_i using action a, i.e., all non-deterministic effects of the action a in state s_i. For example, to remove the transition between states s_0 and s_1 in the partial model of Figure 8, it is also necessary to remove the transition between states s_0 and s_2.

PUA3: Adding a new state. Given a partial model $\mathcal{M} = \langle S, L, T\rangle$, a corresponding updated model $\mathcal{M}' = \langle S', L', T'\rangle$, with respect to $\mathcal{M}^\star$, can be obtained from $\mathcal{M}$ by adding only one new state. That is:

- $S' = S \cup \{s\}$, for some $s \in S^\star$, such that $s \notin S$
- $L'(s) = L^\star(s), s \in S'$
- $T' = T$

It is possible to add one state s in the partial model if and only if this state exists in the complete model.

PUA4: Removing an isolated state. Given a partial model $\mathcal{M} = \langle S, L, T\rangle$, a corresponding updated model $\mathcal{M}' = \langle S', L', T'\rangle$, with respect to $\mathcal{M}^\star$, can be obtained from $\mathcal{M}$ by removing only one isolated state. That is:

- $S' = S - \{s\}$, for some $s \in S$, such that for all $s_i \in S$, $s_i \neq s$, and $a \in \mathbb{A}$, we have $(s_i, a, s) \notin T$ and $(s, a, s_i) \notin T$
- $L'(s) = L(s), s \in S'$
- $T' = T$

Defining minimal change for α-CTL model updating

Based on the work of Zhang and Ding (2008), we can establish a *minimal change criterion* for a labelled transition system using the primitive update operations (PUA1 – PUA4), defined in previous section.

By using a primitive update operation PUAi, a partial model $\mathcal{M}$ given by a system designer can be updated in different ways. Thus, we need a criterion which allows us to measure the changes in the different possible updated models of $\mathcal{M}$ and choose the one which is more close to the original model $\mathcal{M}$.

Given a labelled transition system $\mathcal{M} = \langle S, L, T\rangle$ and a corresponding updated model $\mathcal{M}' = \langle S', L', T'\rangle$, for each operation PUA$i$, for $i = 1..4$, we use $\mathit{Diff}_{\text{PUA}_i}(\mathcal{M}, \mathcal{M}')$ to denote the differences between these two models.

$$\mathit{Diff}_{\text{PUA}i}(\mathcal{M}, \mathcal{M}') = |T' - T| + |S' - S|$$

We also define $\mathit{Diff}(\mathcal{M}, \mathcal{M}')$ as the following tuple: $(\mathit{Diff}_{\text{PUA1}}(\mathcal{M}, \mathcal{M}'), \mathit{Diff}_{\text{PUA2}}(\mathcal{M}, \mathcal{M}'), \mathit{Diff}_{\text{PUA3}}(\mathcal{M}, \mathcal{M}'), \mathit{Diff}_{\text{PUA4}}(\mathcal{M}, \mathcal{M}'))$.

Now, we can precisely define the ordering $\leq_{\mathcal{M}}$ on labelled transition system.

Definition 5. *(Closeness ordering) - Given $\mathcal{M}$ a partial model and $\mathcal{M}'_1$, $\mathcal{M}'_2$ two corresponding updated models, with respect to a complete model $\mathcal{M}^\star$. We say that $\mathcal{M}'_1$ is at least as close to $\mathcal{M}'_2$, denoted as $\mathcal{M}'_1 \leq_{\mathcal{M}} \mathcal{M}'_2$, if and only if for each set of PUA1 – PUA4 operations that transform $\mathcal{M}$ into $\mathcal{M}'_2$, there exists a set of PUA1 – PUA4 operations that transform $\mathcal{M}$ into $\mathcal{M}'_1$, such that the following condition hold:*

$$\mathit{Diff}_{PUAi}(\mathcal{M}, \mathcal{M}'_1) \leq \mathit{Diff}_{PUAi}(\mathcal{M}, \mathcal{M}'_2), \text{ for } i = 1..4$$

We also denote $\mathcal{M}'_1 <_{\mathcal{M}} \mathcal{M}'_2$ if $\mathcal{M}'_1 \leq_{\mathcal{M}} \mathcal{M}'_2$ and $\mathcal{M}'_2 \not\leq_{\mathcal{M}} \mathcal{M}'_1$. For example, if $\mathcal{M}, \mathcal{M}'_1$ and $\mathcal{M}'_2$ are models such that: $\mathit{Diff}_{\text{PUA1}}(\mathcal{M}, \mathcal{M}'_1) = 5$; $\mathit{Diff}_{\text{PUA2}}(\mathcal{M}, \mathcal{M}'_1) = 2$; $\mathit{Diff}_{\text{PUA3}}(\mathcal{M}, \mathcal{M}'_1) = 4$; $\mathit{Diff}_{\text{PUA4}}(\mathcal{M}, \mathcal{M}'_1) = 6$; $\mathit{Diff}_{\text{PUA1}}(\mathcal{M}, \mathcal{M}'_2) = 3$; $\mathit{Diff}_{\text{PUA2}}(\mathcal{M}, \mathcal{M}'_2) = 1$; $\mathit{Diff}_{\text{PUA3}}(\mathcal{M}, \mathcal{M}'_2) = 1$ and $\mathit{Diff}_{\text{PUA4}}(\mathcal{M}, \mathcal{M}'_2) = 5$. We say that $\mathcal{M}'_2 <_{\mathcal{M}} \mathcal{M}'_1$, i.e., $\mathcal{M}'_2$ is more close to $\mathcal{M}$ than $\mathcal{M}'_1$ is.

Definition 5 presents a measure on the difference between two labelled transition system with respect to a partial model given by designer. Intuitively, we say that model M'_1 is closer to M relative to model M'_2 if M'_1 is obtained from M by applying all primitive update operations that cause fewer changes than those applied to obtain model M_2. Having the ordering specified in Definition 5, we can define a α-CTL model updating formally.

Definition 6. *(Admissible Update) Let be a partial model $M = \langle S, L, T\rangle$, $\mathcal{M} = (M, s_0)$, where $s_0 \in S$, and an α-CTL formula φ, a $Update(\mathcal{M}, \varphi)$ is called an admissible model if the conditions below hold:*

- $Update(\mathcal{M}, \varphi) = (M', s'_0), (M', s'_0) \models \varphi$, *where* $M' = (S', L', T')$ *and* $s'_0 \in S'$;
- *There does nor exists another updated model* $M'' = \langle S'', L'', T''\rangle$ *and* $s''_0 \in S''$ *such that* $(M'', s''_0) \models \varphi$ *and* $M'' <_M M'$.

Model Update in Action

Like in the CTL model update proposed by Zhang and Ding (2008), the primitive updating operations *PUA1-PUA4* can suggest modifications by adding or removing states and transitions but:

- by considering an action labelled transition model, the effects of nondeterministic actions imply in different model updates; and
- the temporal formula φ is expressed in α-CTL which, as shown in Pereira and Barros (2008) can specify more expressive planning goals.

In this section we extend model updating to modify an action specification, i.e., an action precondition and effect. The idea is to add two more updating operations to *PUA1-PUA4*, as we show next.

First, we define how to induce the complete model $\mathcal{M}*$ from a set of actions, described in a language based on its preconditions and nondeterministic effects.

Definition 7. *An action is defined by a triple action(name,pre,pos), where pre is the set of preconditions that must be true in a state s where the action is applied; and pos is the set of nondeterministic effects that becomes true in the state resulting from the execution of the action a on state s. We use $pre(a)$ refering to the preconditions of action a and $pos(a)$ for the effects of action a.*

For example, action $(a, \{p, q\}, \{\{r, s\}, \{u\}\})$ indicates that $pre(a) = \{p, q\}$ and the two possible nondeterministic effects of a are given by $pos(a) = \{\{r, s\}, \{u\}\}$

Given an action set $\mathbb{A}$, the preposition set $\mathbb{P}$ of the *complete model* is determined as:

$$\mathbb{P} = \{p : a \in \mathbb{A} \text{ and } pre(a) \cup pos(a)\}$$

that means, all the proposition symbols involved in the description of the set of actions $\mathbb{A}$ are elements of $\mathbb{P}$.

Definition 8. *The* complete model $\mathcal{M}_A^*$ *induced by a set of actions $\mathbb{A}$ is a structure $\langle S_A^*, L_A^*, T_A^* \rangle$, where:*

- S_A^* *is a finite set of enumerated state symbols such that* $|S| = |2^{\mathbb{P}}|$ *;*
- $L_A^* : S \mapsto 2^{\mathbb{P}}$ *is the labelling function that assigns to each state a set of atomic propositions ;*
- $T_A^* = \{(s_x, a, s_y) : s_x, s_y \in S, a \in \mathbb{A}, pre(a) \subseteq s_x,$ *eff* $\in pos(a),$ *eff* $\subseteq L(s_y)$.

Notice that the semantics of actions defines that a transition labelled with action a is added to the complete model $\mathcal{M}_A^*$ if: $pre(a)$ is satisfied in the state s_x; and $pos(a)$ is also satisfied in state s_y. That means, the induced model $\mathcal{M}_A^*$ contains transitions between states without preserving properties in s_x that are not modified by action a. One can justify this kind of transitions with the occurrence of exogenous events (a possible explanation for nondeterministic actions) or as a relaxed state transition model useful to suggest the updates on the partial model (or in a preliminary set of actions specification, as we will see in the next section). We could also induce a complete model from a set of actions respecting the classical planning assumptions (i.e., the STRIPS-like semantics). However, it would be too restrictive while modeling a new planning domain (specially in the case of a nondeterministic domain).

E.g., let us consider the following set of actions $\mathbb{A}$:

- action$(a, \{p, q\}, \{\{r, s\}, \{u\}\})$
- action$(b, \{r\}, \{t\})$
- action$(c, \{u\}, \{\{t\}\})$
- action$(d, \{p, u\}, \{\{t\}\})$

The set of proposition atoms is given by $\mathbb{P} = \{p, q, r, s, u, t\}$. So, the complete model $\mathcal{M}_A^*$ induced by $\mathbb{A}$ has 64 states and all possible transitions, according with the semantics (Definition 7) of actions in $\mathbb{A}$.

Planning Domain Model Updating

In order to develop a real world planning application, a knowledge engineer must specify a correct set of actions which can guarantee the synthesis of correct plans. Our claim is that the use of a formal method, such as the model update approach presented in previous section, can offer an important support to knowledge acquisition, modelling and verification of planning domains. In this paper we use model update in a planning domain w.r.t. a preliminary set of actions $\mathbb{A}$, by making the following correspondences:

- the complete model $\mathcal{M}_A^*$ is induced by the actions $\mathbb{A}$ according with Definition 8 (we may call this complete model as *weakly induced* by $\mathbb{A}$);
- a *partial model* $\mathcal{M} \subseteq \mathbb{A}$ can be seen as a part of the *complete model* $\mathcal{M}_A^*$ that can correspond to (i) a set of plans for a given class of goals, specified in an ad hoc way by a domain expert (or possibly generated by an automatic nondeterministic planner) or ; (ii) a state transition model induced by a stronger semantics of actions (with frame axioms) and
- an α-CTL temporal formula φ is a planning goal (which can since it is expressed by α-CTL can be more complex than a simple reachability goal).

Formally, given a complete model $\mathcal{M}_A^*$ induced by a set of actions $\mathbb{A}$; a partial model $\mathcal{M} \subseteq \mathcal{M}_A^*$; and a α-CTL formula φ defining a planning goal, the set of primitive updating operations $PUA1 - PUA4$ can be used to refine and validate the partial model $\mathcal{M}$. Plus, in order to perform updates directly on the action specification, we need to define two extra primitive updating operations, named $PU5 - PU6$, as follows.

PUA5: Adding transitions induced by a modified action (precondition change). Given (i) a complete model $\mathcal{M}_A^* = \langle S_A^*, L_A^*, T_A^* \rangle$ induced by a set of actions $\mathbb{A}$; (ii) a partial model $\mathcal{M} = \langle S, L, T \rangle$ such that $\mathcal{M} \subseteq \mathcal{M}_A^*$ and (iii) s_i, $s_j \in S$, the corresponding updated model $\mathcal{M}' = \langle S', L', T' \rangle$ is obtained from $\mathcal{M}$ by adding a transition between states s_i and s_j labelled by action a_{new} which is a modified version of an action $a \in \mathbb{A}$ (where $pre(a)$ not satisfied in s_i needs to be relaxed), generating a new set of actions $\mathbb{A}'$. Formally:

- $\mathbb{A}' = (\mathbb{A} \setminus action(a, pre(a), pos(a))) \cup action(a_{new}, pre(a_{new}), pos(a_{new}))$, eff $\in pos(a)$, eff $\subseteq L(s_j)$, $pre(a_{new}) = L(s_i) \cap pre(a)$, $pos(a_{new}) = pos(a)$
- $\mathcal{M}_{A'}^* = \langle S_{A'}^*, L_{A'}^*, T_{A'}^* \rangle$ is a complete model induced by $\mathbb{A}'$, where $S_{A'}^* = S_A^*$, $L_{A'}^* = L_A^*$ and $T_{A'}^* = T_A^* \cup$

$\{(s_x, a_{new}, s_y) | s_x, s_y \in S^*_{A'}, pre(a_{new}) \subseteq L(s_x)$, eff $\in pos(a_{new})$, eff $\subseteq L(s_y)\}$

- $T' = T \cup \{(s_i, a_{new}, s) \in T^*_{A'}\}$
- $S' = S \cup \{s | (s_i, a_{new}, s) \in T'\}$
- $L'(s) = L^*_{A'}(s), s \in S'$

PUA6: Adding transitions induced by a modified action (postcondition change). Given (i) a complete model $\mathcal{M}^*_A = \langle S^*_A, L^*_A, T^*_A \rangle$ induced by a set of actions $\mathbb{A}$; (ii) a partial model $\mathcal{M} = \langle S, L, T \rangle$ such that $\mathcal{M} \subseteq \mathcal{M}^*_A$ and (iii) $s_i, s_j \in S$, the corresponding updated model $\mathcal{M}' = \langle S', L', T' \rangle$ is obtained from $\mathcal{M}$ by adding a transition between states s_i and s_j labelled by action a_{new} which is a modified version of an action $a \in \mathbb{A}$ (where $pre(a)$ is satisfied in s_i and we want to modify $pos(a0)$ to reach s_j), generating a new set of actions $\mathbb{A}'$. Formally:

- $\mathbb{A}' = (\mathbb{A} \setminus action(a, pre(a), pos(a))) \cup action(a_{new}, pre(a_{new}), pos(a_{new}))$, $pre(a) \subseteq L(s_i), pos(a_new) = pos(a) \cup L(s_j), pre(a_new) = pre(a)$
- $\mathcal{M}^*_{A'} = \langle S^*_{A'}, L^*_{A'}, T^*_{A'} \rangle$ is a complete model induced by $\mathbb{A}'$, where $S^*_{A'} = S^*_A$, $L^*_{A'} = L^*_A$ and $T^*_{A'} = T^*_A \cup \{(s_x, a_{new}, s_y) | s_x, s_y \in S^*_{A'}, pre(a_{new}) \subseteq L(s_x)$, eff $\in pos(a_{new})$, eff $\subseteq L(s_y)\}$
- $T' = T \cup \{(s_i, a_{new}, s) \in T^*_{A'}\}$
- $S' = S \cup \{s | (s_i, a_{new}, s) \in T'\}$
- $L'(s) = L^*_{A'}(s), s \in S'$

The principle of minimal change for $PUA5 - PUA6$ follows the same criterion we have defined for $PUA1 - PUA4$. Notice that $PUA5 - PUA6$ can update a model when a transition between two states s_i and s_j does not exist in the complete model $\mathcal{M}^*_A$, i.e., there is no action $a \in \mathbb{A}$ such that $pre(a) \subseteq L(s_i)$ and $pos(a) \subseteq L(s_j)$ and therefore the only way to update the model is by using the operations $PUA5 - PUA6$. However, when the primitive operations $PUA5 - PUA6$ are used to modify the actions in $\mathbb{A}$ they also imply in modifications on the induced complete model $\mathcal{M}^*_A$ which can eventually cause too many changes in the partial model. Therefore, considering all the primitive updating operations, $PUA1 - PUA6$, it is up to the minimal change criterion to suggest a set of minimal changes to the planning domain designer.

Conclusion

In this work we have presented a model updating approach that considers the actions behind the transitions in a state model. We also formalized the principle of minimal change for α-CTL logic - a previous proposed branching time temporal logic that has been applied to planning based on model checking (Pereira and de Barros 2008). To perform model updating in action (Figure 9), we take (i) a set of actions $\mathbb{A}$ (which is used to induce the complete labeled transition system $\mathcal{M}^*$), (ii) a partial model $\mathcal{M}$ such that $\mathcal{M}^* \subseteq \mathcal{M}^*$ and (iii) a α-CTL formula (i.e. a planning goal) and returns an updated model $\mathcal{M}'$, that has minimal change with respect to the original partial model (an example of plan specification or a more restrictive model induced by a preliminary set of actions).

Figure 9: Model updater in action.

The minimal change principle proposed, as well the primitive operations $PUA1 - PUA4$ extended the work of Zhang and Ding (2008) to perform α-CTL model update. By using the description of actions, we proposed two extra primitive operations $PUA5 - PUA6$, that can be used to change an action specification (its precondition and effect, respectively). These two operations allow us to apply the proposed model updating as an important supporting tool for a planning domain designer.

As a future work we intend to implement an α-CTL model updater, based on our α-CTL model checker implementation and use it to refine some nondeterministic planning domain with complex goals. We also want to make experiments with new planning applications.

Acknowledgments. We thank CNPq and FAPESP (grant 2009/07039-4) for financial support.

References

Bryant, R. E. 1992. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys* 24(3):293–318.

Buccafurri, F.; Eiter, T.; Gottlob, G.; and Leone, N. 1999. Enhancing model checking in verification by AI techniques. *Artif. Intell.* 112(1-2):57–104.

Clarke, E. M., and Emerson, E. A. 1982. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, 52–71. London, UK: Springer-Verlag.

Clarke E., Grumberg O., P. D. 1999. *Model Checking*. San Francisco: MIT Press.

Harris, H., and Ryan, M. 2003. Theoretical foundations of updating systems. In *18th IEEE International Conference on Automated Software Engineering, 2003. Proceedings*, 291–294.

Kripke, S. 1963. Semantical considerations on modal logic. *Acta Philosophica Fennica* 16:83–94.

Müller-Olm, M.; Schmidt, D. A.; and Steffen, B. 1999. Model-checking: A tutorial introduction. In *SAS '99: Proceedings of the 6th International Symposium on Static Analysis*, 330–354. London, UK: Springer-Verlag.

Pereira, S. L., and de Barros, L. N. 2008. A logic-based agent that plans for extended reachability goals. *Autonomous Agents and Multi-Agent Systems* 16(3):327–344.

Zhang, Y., and Ding, Y. 2008. CTL model update for system modifications. *J. Artif. Int. Res.* 31(1):113–155.