Abstract Model Repair

George Chatzieleftheriou[1], Borzoo Bonakdarpour[2], Scott A. Smolka[3], and Panagiotis Katsaros[1]

[1] Department of Informatics, Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
[2] School of Computer Science, University of Waterloo
200 University Avenue West Waterloo N2L3G1, Canada
[3] Department of Computer Science, Stony Brook University
Stony Brook, NY 11794-4400, USA

Abstract. Given a Kripke structure M and CTL formula φ, where $M \not\models \varphi$, the problem of *Model Repair* is to obtain a new model M' such that $M' \models \varphi$. Moreover, the changes made to M to derive M' should be minimal with respect to all such M'. As in model checking, *state explosion* can make it virtually impossible to carry out model repair on models with infinite or even large state spaces. In this paper, we present a framework for model repair that uses *abstraction refinement* to tackle state explosion. Our model-repair framework is based on Kripke Structures, a 3-valued semantics for CTL, and Kripke Modal Transition Systems (KMTSs), and features an abstract-model-repair algorithm for KMTSs. Application to an Automatic Door Opener system is used to illustrate the practical utility of abstract model repair.

Keywords: Model Repair, Model Checking, Abstraction Refinement

1 Introduction

Given a model M and temporal-logic formula φ, *model checking* is the problem of determining if $M \models \varphi$. When this is not the case, a model checker will typically provide a *counterexample* in the form of an execution path along which ϕ is violated. The user should then process the counterexample manually to correct the model.

An extended version of the model-checking problem is that of *model repair*: given a model M and temporal-logic formula φ, where $M \not\models \varphi$, obtain a new model M' such that $M' \models \phi$. The problem of Model Repair was introduced for the first time in the context of Kripke structures and the CTL temporal logic in [4].

State explosion is a well known problem in automated formal methods, such as model checking and model repair, which limits their applicability to systems having large or even infinite state spaces. Different techniques have been developed to cope with this problem. In the case of model checking, *abstraction* is used to create a smaller, more abstract version $\hat{M}$ of the initial concrete model

M, and model checking is performed on this smaller model. For this technique to work as advertised, it should be the case that $\hat{M} \models \varphi$ iff $M \models \varphi$.

Motivated by the success of abstraction-based model checking, we present in this paper a new framework for Model Repair that uses *abstraction refinement* to tackle state explosion. The resulting *Abstract Model Repair* (AMR) methodology makes it possible to repair models with large state spaces, and to speed-up the repair process through the use of smaller abstract models. The major contributions of our work are as follows:

– We provide an AMR framework that uses Kripke structures (KSs) for the concrete model, Kripke Modal Transition Systems (KMTSs) for the abstract model, and a 3-valued semantics for interpreting CTL over KMTSs. An abstract KMTS model is refined whenever the 3-valued CTL model-checking problem returns a value of undefined. Repair is initiated on the KMTS when a value of false is returned.
– We strengthen the Model Repair problem by additionally taking into account the following *minimality* criterion (refer to the definition of Model Repair above): the changes made to M to derive M' should be minimal with respect to all M' satisfying φ. To handle the minimality constraint, we define a metric space over KSs that quantifies the structural differences between KSs.
– A key feature of our Abstract Model Repair framework is a repair algorithm for KMTSs, which takes into account the minimality criterion.
– We illustrate the utility of our approach by applying it to the repair of an Automatic Door Opener system [1].

The rest of this paper is organized as follows. Sections 2 and 3 introduce KS, KMTSs, and the concepts of abstraction and refinement for a 3-valued semantics for CTL. Section 4 defines a metric space for KSs and gives the problem statement for Model Repair. Section 5 presents our framework for Abstract Model Repair, while Section 6 highlights our model-repair algorithm for KMTSs. Section 7 considers related work, while Section 8 offers our concluding remarks.

2 Kripke Modal Transition Systems

Let AP be a set of *atomic propositions*. Also, the set Lit of *literals* is given by:

$$Lit = AP \cup \{\neg p : p \in AP\}$$

Definition 1. *A* Kripke Structure *(KS) is a quadruple $M = (S, S_0, R, L)$, where:*

1. *S is a finite set of* states*.*
2. *$S_0 \subseteq S$ is the set of* initial states*.*
3. *$R \subseteq S \times S$ is a* transition relation *that must be total; i.e., $\forall s \in S$, $\exists s' \in S$ such that $R(s, s')$.*
4. *$L : S \to 2^{Lit}$ is a state* labeling function *such that $\forall s \in S$, $\forall p \in AP$, $p \in L(s) \Leftrightarrow \neg p \notin L(s)$.*

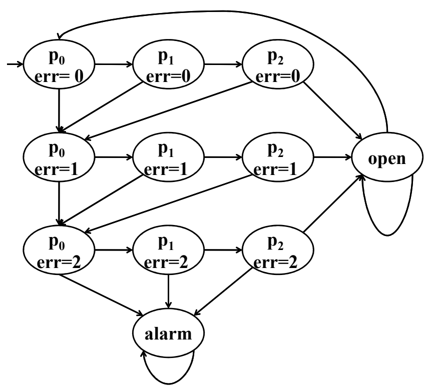


Fig. 1: The Automatic Door Opener (ADO) System.

The fourth condition in Def. 1 ensures that an atomic proposition $p \in AP$ has one and only one truth value at any state.
Example. We use the Automatic Door Opener system (ADO) of [1] as a running example throughout the paper. The system, given as a KS in Fig 1, requires a three-digit code (p_0, p_1, p_2) to open a door, allowing for a wrong digit to be entered at most twice. Variable *err* counts the number of errors, and an alarm is rung if its value exceeds two. For the purposes of our paper, we use a simpler version of the ADO system, given as the KS M in Fig. 2a, where the set of atomic propositions $AP = \{q\}$, $q \equiv (open = true)$.

Definition 2. *A* Kripke Modal Transition System *(KMTS) is a 5-tuple* $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$*, where:*

1. $\hat{S}$ *is a finite set of* states*.*
2. $\hat{S}_0 \subseteq \hat{S}$ *is the set of* initial states*.*
3. $R_{must} \subseteq \hat{S} \times \hat{S}$ *and* $R_{may} \subseteq \hat{S} \times \hat{S}$ *are* transition relations *such that* $R_{must} \subseteq R_{may}$.
4. $\hat{L} : \hat{S} \rightarrow 2^{Lit}$ *is a state-labeling such that* $\forall \hat{s} \in \hat{S}$, $\forall p \in AP$, $\hat{s}$ *is labeled by at most one of* p *and* $\neg p$*.*

A KMTS has two types of transitions: *must-transitions*, which exhibit *necessary* behavior, and *may-transitions*, which exhibit *possible* behavior. The "at most one" condition in the fourth part of Def. 2 makes it possible for the truth value of an atomic proposition at a given state to be *unknown*. This relaxation of truth values in conjunction with the existence of may-transitions in a KMTS constitutes a *partial modeling* formalism.

Verifying a CTL formula ϕ over a KMTS may result in an undefined answer ($\perp$). We use the *3-valued semantics* [13] of a CTL formula ϕ at a state $\hat{s}$ of KMTS $\hat{M}$ (denoted $[(\hat{M}, \hat{s}) \models^3 \phi]$). From the 3-valued semantics, it follows that must-transitions (under-approximation) are used to check the truth of existential CTL properties, while may-transitions (over-approximation) are used to check the truth of universal CTL properties. This works inversely for checking the refutation of CTL properties. When we get $\perp$ from the 3-valued model checking of a CTL formula ϕ on a KMTS, the result of model checking property ϕ on the corresponding KS can be either true or false. In the rest of the paper, we use $\models$ instead of $\models^3$ in order to refer to 3-valued satisfaction relation.

3 Abstraction and Refinement for 3-Valued CTL

3.1 Abstraction

Abstraction is a state-space reduction technique that produces a smaller abstract model from an initial *concrete* model, so that the models behave similarly. In order for the result of verifying an abstract model to hold for its concrete model, the abstract model should be produced with certain requirements [7, 10].

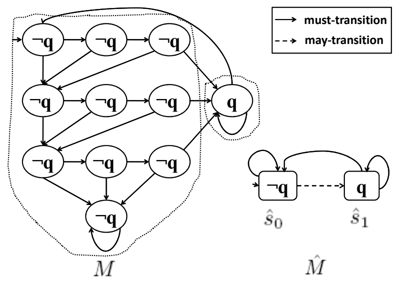
Definition 3. *Let $M = (S, S_0, R, L)$ be a KS. For any pair of total functions $\Re = (\alpha : S \to \hat{S}, \gamma : \hat{S} \to 2^S)$, where $\forall s \in S$, $\hat{s} \in \hat{S}$, $\alpha(s) = \hat{s}$ if and only if $s \in \gamma(\hat{s})$, a KMTS $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$ is defined as follows:*

1. *$\hat{s} \in \hat{S}_0$ iff $\exists s \in \gamma(\hat{s})$ such that $s \in S_0$*
2. *$lit \in \hat{L}(\hat{s})$ only if $\forall s \in \gamma(\hat{s})$ it holds that $lit \in L(s)$*
3. *$R_{must} = \{(\hat{s_1}, \hat{s_2}) \mid \forall s_1 \in \gamma(\hat{s_1})\ \exists s_2 \in \gamma(\hat{s_2})$ such that $R(s_1, s_2)\}$*
4. *$R_{may} = \{(\hat{s_1}, \hat{s_2}) \mid \exists s_1 \in \gamma(\hat{s_1})\ \exists s_2 \in \gamma(\hat{s_2})$ such that $R(s_1, s_2)\}$*

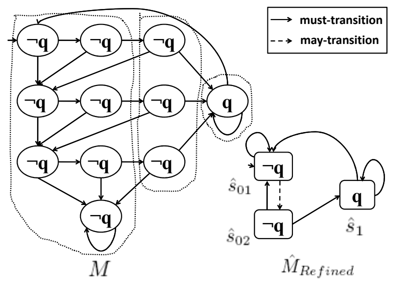
For a given KS and pair of abstraction and concretization functions, Def. 3 introduces a KMTS with a set $\hat{S}$ of *abstract states*. In our AMR framework, we view the given KS as the *concrete model* and the derived KMTS as the *abstract model*. A state of the abstract KMTS is initial *if and only if* at least one of its concrete states is initial. An atomic proposition is true (or false) in an abstract state, *only if* this atomic proposition is true (or false) in all of its concrete states. *Only if* allows for the value of an atomic proposition to be unknown at a KMTS state. Between two abstract states $\hat{s_1}$,$\hat{s_2}$, there exists a must-transition if there are transitions from all the concrete states of $\hat{s_1}$ to at least one concrete state of $\hat{s_2}$ ($\forall\exists - condition$), while on the other side, there exists a may-transition if there is a transition from at least one concrete state of $\hat{s_1}$ to at least one concrete state of $\hat{s_2}$ ($\exists\exists - condition$).

Definition 4. [8, 11] *Let $M = (S, S_0, R, L)$ be a concrete KS, and let $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$ be an abstract KMTS. A relation $H \subseteq S \times \hat{S}$ for M and $\hat{M}$ is called a* mixed simulation*, when $H(s, \hat{s})$ implies:*

- $\hat{L}(\hat{s}) \subseteq L(s)$
- *if $r = (s, s') \in R$, then there exists some $\hat{s'} \in \hat{S}$ such that $r_{may} = (\hat{s}, \hat{s'}) \in R_{may}$ and $(s', \hat{s'}) \in H$.*



(a) The KS and initial KMTS.



(b) The KS and refined KMTS.

Fig. 2: The KS and KMTSs for the ADO system.

– *if* $r_{must} = (\hat{s}, \hat{s'}) \in R_{must}$*, then there exists some* $s' \in S$ *such that* $r = (s, s') \in R$ *and* $(s', \hat{s'}) \in H$.

Abstraction function α in Def. 3 is a mixed simulation for KS M and KMTS $\hat{M}$.

Theorem 1. [11] *Let* $H \subseteq S \times \hat{S}$ *be a mixed simulation from a KS* $M = (S, S_0, R, L)$ *to a KMTS* $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$*. Then, for every CTL formula* φ *and every* $(s, \hat{s}) \in H$ *it holds that*

$$[(\hat{M}, \hat{s}) \models \varphi] \neq \bot \Rightarrow [(M, s) \models \varphi] = [(\hat{M}, \hat{s}) \models \phi]$$

Theorem 1 ensures that if a CTL formula ϕ has a definite truth value (true or false) in the abstract KMTS then it has the same truth value in the concrete KS.
Example. An abstract KMTS $\hat{M}$ is presented in Fig. 2a, where all the states labeled by q are grouped together, as are all states labeled by $\neg q$.

3.2 Refinement

When the answer to verifying a CTL formula φ on an abstract model using the 2-valued semantics is $\bot$, then a *refinement* step is needed to acquire a more precise abstract model. A number of refinement frameworks specialized for 3-valued model checking have been proposed [10, 16]. The refinement technique that we use in our framework is a two-step process: (1) identify a *failure state* in the KMTS, and (2) produce a new abstract KMTS such that this failure state is refined into several states. The cause of failure for a state s stems from an atomic proposition having an undefined value in s, or from an outgoing may-transition from s. In both cases, s is refined in a way that the cause of failure is eliminated.

Example Consider the case where the ADO system requires a mechanism for opening the door from any state with a direct action. This could be an action done by an expert if an immediate opening of a door is required. This property can be expressed in CTL as the formula $\varphi = AGEXq$. Observe that in $\hat{M}$ of Fig. 2a, the absence of a must-transition from $\hat{s}_0$ to $\hat{s}_1$, where $[(\hat{M}, \hat{s}_1) \models q] = true$, in conjunction with the existence of a may-transition from $\hat{s}_0$ to $\hat{s}_1$, thus to a state where $[(\hat{M}, \hat{s}_1) \models q] = true$, results in an undefined answer to the model-checking question for $\hat{M}$ and φ. State $\hat{s}_0$ is identified as the failure state, and the may-transition from $\hat{s}_0$ to $\hat{s}_1$ as the cause of the failure. Consequently, $\hat{s}_0$ is refined into two states, $\hat{s}_{01}$ and $\hat{s}_{02}$, such that the former has no transition to $\hat{s}_1$ and the latter has an outgoing must-transition to $\hat{s}_1$. As such, we eliminate the may-transition which led to the undefined answer of model checking $var\phi$ over $\hat{M}$. The refined KMTS $\hat{M}_{Refined}$ together with the initial KS is shown in Fig. 2b.

4 The Model Repair Problem

In this section, we give the problem statement for Model Repair and define a metric space over Kripke structures to quantify their structural differences such that the *minimality of changes* can be taken into account as a criterion for Model Repair.
Let G be a function on the set of all functions $F : X \to Y$ such that:

$$G(F : X \to Y) = \{(x, F(x)) : x \in X\}$$

Let $F : X \to Y$ be a function defined over a set X. A *restricting operator* ($\upharpoonright$) for the domain of function F can be defined such that

$$F \upharpoonright_{X_1} = \{(x, F(x)) : x \in X_1\}$$

where $X_1 \subseteq X$. Finally, we let S^C denote the complement of a set S.

Definition 5. *Let K_M be the set of all KSs $M' = (S', S'_0, R', L')$ derived from the KS $M = (S, S_0, R, L)$, where $S' = (S \cup S_{IN}) - S_{OUT}$ for some $S_{IN} \subseteq S^C$, $S_{OUT} \subseteq S$, $R' = (R \cup R_{IN}) - R_{OUT}$ for some $R_{IN} \subseteq R^C$, $R_{OUT} \subseteq R$, $L' = S' \to 2^{LIT}$. A* distance function *d can be defined over K_M such that*

$$d(M, M') = |S \Delta S'| + |R \Delta R'| + \frac{|G(L \upharpoonright_{S \cap S'}) \Delta G(L' \upharpoonright_{S \cap S'})|}{2}$$

where $A \Delta B$ represents the symmetric difference $(A - B) \cup (B - A)$.

For any two KSs defined over the same set of atomic propositions AP, function d counts the number of differences $|S \Delta S'|$ in the state space of M, the number of differences $|R \Delta R'|$ in their transition relation and the number of common states with altered labeling.

Proposition 1. *The ordered pair (K_M, d) is a metric space.*

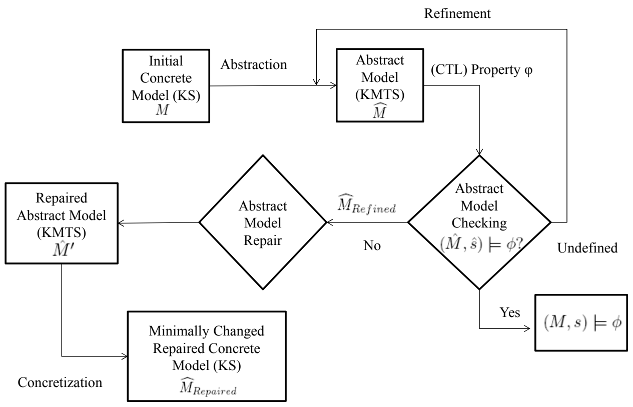

Refinement
Initial Concrete Model (KS) M
Abstraction
Abstract Model (KMTS) $\widehat{M}$
(CTL) Property φ
Abstract Model Checking $(\hat{M}, \hat{s}) \models \phi?$
Undefined
$\widehat{M}_{Refined}$
No
Abstract Model Repair
Repaired Abstract Model (KMTS) $\hat{M}'$
Yes
$(M, s) \models \phi$
Concretization
Minimally Changed Repaired Concrete Model (KS) $\widehat{M}_{Repaired}$

Fig. 3: Abstract Model Repair Framework.

Definition 6. *Given a KS M and a CTL formula φ where $M \not\models \varphi$, the Model Repair problem is to find a KS M', such that $M' \models \varphi$ and $d(M, M')$ is minimal with respect to all such M'.*

The Model Repair problem aims at modifying a KS such that the KS satisfies a CTL formula that it originally does not. We focus on repair with minimal changes to the original KS.

5 Abstract Model Repair Framework and Algorithm

Our AMR framework integrates 3-valued model checking, model refinement, and a new algorithm for ordering the basic repair operations to be performed on the abstract model. The goal of this algorithm is to order the repair operations in such a way that the number of corresponding structural changes applied to the concrete model is minimized. The basis for this algorithm is a partial order over the basic repair operations. This section describes the steps involved in our AMR framework, the basic repair operations, and the operations-ordering algorithm.

5.1 The Abstract Model Repair Process

The process steps shown in Fig. 3 rely on the KMTS abstraction of Def. 3. These are the following:

Step 1. Given a KS M, a state s of M, and a CTL property φ, let us call $\hat{M}$ the KMTS obtained as in Def. 3.

Step 2. For state $\hat{s} = \alpha(s)$ of $\hat{M}$, we check whether $(\hat{M}, \hat{s}) \models \varphi$ by 3-valued model checking.

Case 1. If the result is *true*, then, according to Theorem 1, $(M, s) \models \varphi$ and there is no need for repair.

Case 2. If the result is *undefined*, $\hat{M}$ is refined to an $\hat{M}_{Refined}$ and control is transferred to Step 2.

Case 3. If the result is *false*, then, from Theorem 1, $(M, s) \not\models \varphi$ and the repair process follows.

Step 3. The *AbstractRepair* algorithm is called for the KMTS $\hat{M}$ (or $\hat{M}_{Refined}$ if refinement occurred), the state $\hat{s}$ and the property φ.

Case 1. *AbstractRepair* returns an $\hat{M}'$ for which $(\hat{M}', \hat{s}) \models \varphi$.

Case 2. *AbstractRepair* fails to find an $\hat{M}'$ for which the property holds.

Step 4. If *AbstractRepair* returns an $\hat{M}'$, then the process ends with a set of KSs, resulting from the concretization of $\hat{M}'$, whose structural distance d from the original KS M is minimized.

5.2 Basic Repair Operations

We decompose the repair process of the KMTS into seven basic repair operations:

AddMust. Adding a must-transition
AddMay. Adding a may-transition
RemoveMust. Removing an existing must-transition
RemoveMay. Removing an existing may-transition
ChangeLabel. Changing the labeling of a KMTS state
AddState. Adding a new KMTS state
RemoveState. Removing a disconnected KMTS state

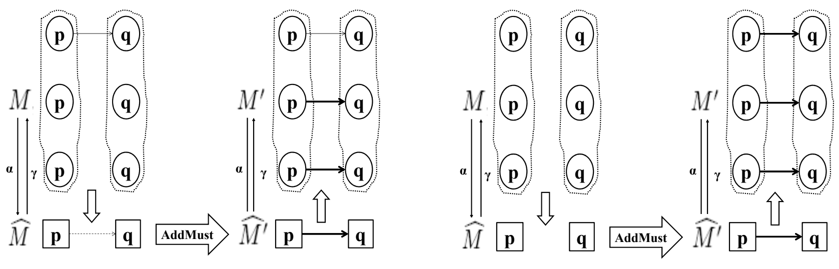
Definition 7 (AddMust). *For a given KMTS* $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$ *and* $\hat{r}_n = (\hat{s}_1, \hat{s}_2) \notin R_{must}$ *with* $\hat{s}_1, \hat{s}_2 \in \hat{S}$*,* $AddMust(\hat{M}, \hat{r}_n)$ *is a KMTS* $\hat{M}' = (\hat{S}', \hat{S}_0', R_{must}', R_{may}', \hat{L}')$ *such that* $\hat{S}' = \hat{S}$*,* $\hat{S}_0' = \hat{S}_0$*,* $R_{must}' = R_{must} \cup \{\hat{r}_n\}$*,* $R_{may}' = R_{may} \cup \{\hat{r}_n\}$ *and* $\hat{L}' = \hat{L}$*.*

Fig. 4 shows how the basic repair operation *AddMust* modifies a given KMTS.

Definition 8. *Let* M *be a KS,* $\hat{M}$ *be a KMTS derived as in Def. 3, and* $\hat{M}' = AddMust(\hat{M}, \hat{r}_n)$ *for some* $\hat{r}_n = (\hat{s}_1, \hat{s}_2) \notin R_{must}$ *with* $\hat{s}_1, \hat{s}_2 \in \hat{S}$*. The set of KSs, derived from the concretization of* $\hat{M}'$*, whose structural distance* d *from* M *is minimized is given by:*

$$K_{min} = \{M' = (S', S_0', R', L') \mid S' = S, S_0' = S_0, R' = R \cup R_n, L' = L\} \quad (1)$$

where
$R_n = \{r_n = (s_1, s_2) \mid$ *for every* $s_1 \in \gamma(\hat{s}_1)$ *such that* $\nexists s \in \gamma(\hat{s}_2)$ *with* $(s_1, s) \in R$*, and only one* $s_2 \in \gamma(\hat{s}_2)\}$*.*



(a) May-transition exists (b) May-transition doesn't exist

Fig. 4: *AddMust*: Adding a new must-transition

Def. 8 implies that when the *AbstractRepair* algorithm applies *AddMust* on the abstract KMTS $\hat{M}$, then a set of KSs are retrieved from the concretization of $\hat{M}'$. The same holds for the other basic repair operations for which their definition is omitted for the sake of brevity. Consequently when *AbstractRepair* finds a repaired KMTS, one or more KSs can be obtained for which property φ holds.

Proposition 2. *For all $M' \in K_{min}$, it holds that $1 \leq d(M, M') \leq |S|$.*

From Prop. 2, we conclude that a lower and upper bound exists for the distance between M and any $M' \in K_{min}$.

Minimality Of Changes Ordering For Basic Repair Operations Based on the upper bound given by Prop. 2 and the corresponding results for the other basic repair operations, we introduce the ordering shown in Fig. 5. We use this ordering in the *AbstractRepair* algorithm to *heuristically* select at each step

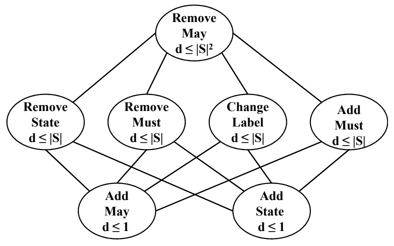


Fig. 5: Minimality of changes ordering of the set of basic operations

Algorithm 1 AbstractRepair

```
Input: $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$, $\hat{s} \in \hat{S}$, a CTL property $\phi$ for which $(\hat{M}, \hat{s}) \not\models \phi$,
    and a set of constraints $C = \{(\hat{s}_{c1}, \phi_{c1}), (\hat{s}_{c2}, \phi_{c2}), ..., (\hat{s}_{cn}, \phi_{cn})\}$ where $\hat{s}_{ci} \in \hat{S}$ and
    $\phi_{ci}$ is a CTL formula.
Output: $\hat{M}' = (\hat{S}', \hat{S}'_0, R'_{must}, R'_{may}, \hat{L}')$ and $(\hat{M}', \hat{s}) \models \phi$ or FAILURE.
 1: $\phi_{pos} := PositiveNormalForm(\phi)$
 2: if $\phi_{pos}$ is $\bot$ then
 3:     return FAILURE
 4: else if $\phi_{pos} \in LIT$ then
 5:     return $AbstractRepair_{ATOMIC}(\hat{M}, \hat{s}, \phi_{pos}, C)$
 6: else if $\phi_{pos}$ is $\phi_1 \wedge \phi_2$ then
 7:     return $AbstractRepair_{AND}(\hat{M}, \hat{s}, \phi_{pos}, C)$
 8: else if $\phi_{pos}$ is $\phi_1 \vee \phi_2$ then
 9:     return $AbstractRepair_{OR}(\hat{M}, \hat{s}, \phi_{pos}, C)$
10: else if $\phi_{pos}$ is $OPER\phi_1$ then
11:     return $AbstractRepair_{OPER}(\hat{M}, \hat{s}, \phi_{pos}, C)$
12:     where $OPER \in \{AX, EX, AU, EU, AF, EF, AG, EG\}$
```

the basic repair operation that generates the KSs with the least changes. The alternative to check at each step all possible repaired KSs in order to identify the proper basic repair operation, would cancel the benefits of using abstraction. The reason is that such a check inevitably depends on the size of the KS.

6 The Abstract Model Repair Algorithm

The *AbstractRepair* algorithm used in Step 3 of our repair process is a recursive, CTL syntax-directed algorithm. The repair of an abstract KMTS is accomplished by successive calls of primitive repair functions that handle atomic formulas, logical connectives and CTL operators.

The main routine of *AbstractRepair* is presented in Algorithm 1. A set of constraints $C = \{(\hat{s}_{c1}, \phi_{c1}), (\hat{s}_{c2}, \phi_{c2}), ..., (\hat{s}_{cn}, \phi_{cn})\}$ which is initially empty is passed as an argument in the successive recursive calls of *AbstractRepair*. If C is not empty, then for the KMTS $\hat{M}'$ returned from *AbstractRepair*, it holds that $(\hat{M}', \hat{s}_{ci}) \models \phi_{ci}$ for all $(\hat{s}_{ci}, \phi_{ci}) \in C$. C is used for handling conjunctive formulas of the form $\phi = \phi_1 \wedge \phi_2$ for some state $\hat{s}$. In this case, *AbstractRepair* is called for the KMTS $\hat{M}$ and property ϕ_1 with $C = \{(\hat{s}, \phi_2)\}$. The same is repeated for property ϕ_2 with $C = \{(\hat{s}, \phi_1)\}$ and the two results are combined appropriately.

For any CTL formula ϕ and KMTS state $\hat{s}$, *AbstractRepair* either outputs a KMTS $\hat{M}'$ for which $(\hat{M}', \hat{s}) \models \phi$ or else returns FAILURE if such a model cannot be found. This is the case when the algorithm handles conjunctive formulas and a KMTS that simultaneously satisfies all conjuncts cannot be found.

6.1 Primitive Functions

For a simple atomic formula, $AbstractRepair_{ATOMIC}$ updates the label of the input state with the given atomic proposition. While conjunctive formulas are

Algorithm 2 *AbstractRepair*$_{AG}$

```
Input: $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$, $\hat{s} \in \hat{S}$, a CTL property $\phi = AG\phi_1$ for which
    $(\hat{M}, \hat{s}) \not\models \phi$, and a set of constraints $C = \{(\hat{s}_{c1}, \phi_{c1}), (\hat{s}_{c2}, \phi_{c2}), ..., (\hat{s}_{cn}, \phi_{cn})\}$ where
    $\hat{s}_{ci} \in \hat{S}$ and $\phi_{ci}$ is a CTL formula.
Output: $\hat{M}' = (\hat{S}', \hat{S}'_0, R'_{must}, R'_{may}, \hat{L}')$ and $(\hat{M}', \hat{s}) \models \phi$ or FAILURE.
 1: if $(\hat{M}, \hat{s}) \not\models \phi_1$ then
 2:   $RET := AbstractRepair(\hat{M}, \hat{s}, \phi_1, C)$
 3:   if $RET == FAILURE$ then
 4:     return FAILURE
 5:   else
 6:     $\hat{M}' := RET$
 7: else
 8:   $\hat{M}' := \hat{M}$
 9: $\hat{M}'' := \hat{M}'$
10: for all reachable states $\hat{s}_k$ through may-transitions from $\hat{s}$ such that $(\hat{M}', \hat{s}_k) \not\models \phi_1$
    do
11:   $RET := AbstractRepair(\hat{M}', \hat{s}_k, \phi_1, C)$
12:   if $RET == FAILURE$ then
13:     BREAK
14:   else
15:     $\hat{M}' := RET$
16: if $\hat{M}' \models \phi$ && $\hat{M}' \models C$ then
17:   return $\hat{M}'$
18: else
19:   $\hat{M}' := \hat{M}''$
20:   for all $\hat{\pi}_{may} := [\hat{s}, \hat{s}_1, ..., \hat{s}_i, \hat{s}_k]$ for which $(\hat{M}', \hat{s}_k) \not\models \phi_1$, $(\hat{M}', \hat{s}_i) \models \phi_1$ and
      $\nexists \hat{s}_j \in \hat{\pi}_{may}$ such that $(\hat{M}', \hat{s}_j) \not\models \phi_1$ and $\hat{s}_j \in Pre_{may}(\hat{s}_i)$ do
21:     $\hat{r}_m := (\hat{s}_i, \hat{s}_k)$, $\hat{M}' := RemoveMay(\hat{M}', \hat{r}_m)$
22:     if $\hat{s}_i$ is a dead-end state then
23:       $\hat{r}_n := (\hat{s}_i, \hat{s}_i)$, $\hat{M}' := AddMay(\hat{M}', \hat{r}_n)$
24:   if $\hat{M}' \models C$ then
25:     return $\hat{M}'$
26:   else
27:     return FAILURE
```

handled by the algorithm with the use of constraints, disjunctive formulas are handled by repairing any of the disjuncts.

Algorithm 2 describes the primitive function *AbstractRepair*$_{AG}$ which is called when $\phi = AG\phi_1$. When *AbstractRepair*$_{AG}$ is called with state $\hat{s}$ as argument, it recursively calls *AbstractRepair* for all states that are reachable from $\hat{s}$ through successive may-transitions and do not satisfy ϕ_1. If the found KMTS $\hat{M}'$ does not violate any constraint in C, then $(\hat{M}', \hat{s}) \models \phi$ and *AbstractRepair*$_{AG}$ returns the found solution. If a KMTS does not satisfy all the constraints in C, then *AbstractRepair*$_{AG}$ tries to repair the input KMTS by removing all may-transitions through which the state violating ϕ_1 is reached.

AbstractRepair$_{EX}$ presented in Algorithm 3 is the primitive function for handling properties of the form $EX\phi_1$ for some state $\hat{s}$. Initially, this function

Algorithm 3 *AbstractRepair*$_{EX}$

```
Input: $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$, $\hat{s} \in \hat{S}$, a CTL property $\phi = EX\phi_1$ for which
    $(\hat{M}, \hat{s}) \not\models \phi$, and a set of constraints $C = \{(\hat{s}_{c1}, \phi_{c1}), (\hat{s}_{c2}, \phi_{c2}), ..., (\hat{s}_{cn}, \phi_{cn})\}$ where
    $\hat{s}_{ci} \in \hat{M}$ and $\phi_{ci}$ is a CTL formula.
Output: $\hat{M}' = (\hat{S}', \hat{S}_0', R_{must}', R_{may}', \hat{L}')$ and $(\hat{M}', \hat{s}) \models \phi$ or FAILURE.
 1: if there exists $\hat{s}_1 \in \hat{S}$ such that $(\hat{M}, \hat{s}_1) \models \phi_1$ then
 2:    for all $\hat{s}_i \in \hat{S}$ such that $(\hat{M}, \hat{s}_i) \models \phi_1$ do
 3:       $\hat{r}_n := (\hat{s}, \hat{s}_i)$, $\hat{M}' := AddMust(\hat{M}, \hat{r}_n)$
 4:       if $\hat{M}' \models C$ then
 5:          return $\hat{M}'$
 6: else
 7:    for all $\hat{s}_i \in Post_{must}(\hat{s})$ do
 8:       $RET := AbstractRepair(\hat{M}, \hat{s}_i, \phi_1, C)$
 9:       if $RET \neq FAILURE$ then
10:          $\hat{M}' := RET$
11:          return $\hat{M}'$
12:    $\hat{M}' := AddState(\hat{M}, \hat{s}_1')$, $\hat{r}_n := (\hat{s}, \hat{s}_1')$, $\hat{M}' := AddMust(\hat{M}', \hat{r}_n)$
13:    if $\hat{s}_1'$ is a dead-end state then
14:       $\hat{r}_n := (\hat{s}_1', \hat{s}_1')$, $\hat{M}' := AddMay(\hat{M}', \hat{r}_n)$
15:    $RET := AbstractRepair(\hat{M}', \hat{s}_1', \phi_1, C)$
16:    if $RET \neq FAILURE$ then
17:       $\hat{M}' := RET$
18:       return $\hat{M}'$
19:    else
20:       return FAILURE
21: return FAILURE
```

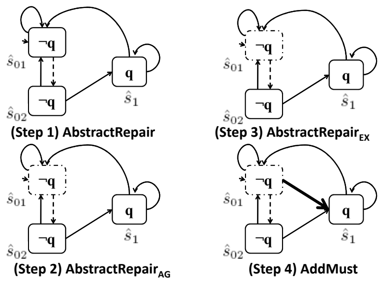
tries to repair the KMTS by adding a must-transition from $\hat{s}$ to a state that satisfies property ϕ_1. If the obtained KMTS does not satisfy all constraints in C, then *AbstractRepair* is recursively called for an immediate successor of $\hat{s}$ through a must-transition, such that ϕ_1 is not satisfied. If a constraint in C is still violated, then (i) a new state is added, (ii) *AbstractRepair* is called for the new state and (iii) a must-transition from $\hat{s}$ to the new state is added.

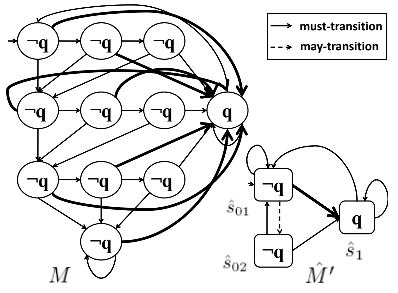
6.2 Well-definedness and Soundness

AbstractRepair is *well-defined*, in the sense that all possible cases are handled and each algorithm step is deterministically defined. This feature distinguishes our approach from related concrete model repair solutions which entail nondeterministic behavior [19, 5].

Theorem 2 (Soundness). *Let $\hat{M}$ be a KMTS and ϕ a CTL formula for which $(\hat{M}, \hat{s}) \not\models \phi$ for some state $\hat{s}$ of $\hat{M}$. If AbstractRepair$(\hat{M}, \hat{s}, \phi)$ returns a KMTS $\hat{M}'$, then $(\hat{M}', \hat{s}) \models \phi$.*

Proof. The proof is done by structural induction over ϕ.





(a) Application of *AbstractRepair*. (b) The repaired KMTS and KS.

Fig. 6: Repair of ADO system using abstraction.

Theorem 2 shows that *AbstractRepair* is *sound* in the sense that if it returns a KMTS $\hat{M}'$, then $\hat{M}'$ satisfies property ϕ. In that case, from Def. 8 it follows that one or more KSs are obtained for which property ϕ holds true.

6.3 Application

We present the application of *AbstractRepair* to the ADO system from Section 2. After the first two steps of our repair process, *AbstractRepair* is called for the KMTS $\hat{M}_{Refined}$ that is shown in Fig. 2b, the state $\hat{s}_{01}$ and the CTL property $\phi = AGEXq$.

AbstractRepair calls *AbstractRepair*$_{AG}$ with arguments $\hat{M}_{Refined}$, $\hat{s}_{01}$ and $AGEXq$. The *AbstractRepair*$_{AG}$ algorithm at line 2 triggers a recursive call of *AbstractRepair* with the same arguments. Eventually, *AbstractRepair*$_{EX}$ is called with arguments $\hat{M}_{Refined}$, $\hat{s}_{01}$ and EXq, that in turn calls *AddMust* at line 3, thus adding a must-transition from $\hat{s}_{01}$ to $\hat{s}_1$. *AbstractRepair* terminates by returning a KMTS $\hat{M}'$ that satisfies $\phi = AGEXq$. The repaired KS M' is the single element in the set of KSs derived by the concretization of $\hat{M}'$. The execution steps of *AbstractRepair* and the obtained repaired KMTS and KS are shown in Fig. 6a and Fig. 6b respectively.

Although the ADO is not a system with a large state space, it is shown that the repair process is accelerated by the proposed use of abstraction. If on the other hand model repair was applied directly to the concrete model, adding transitions to the state labeled with *open* would have to take place for all states with a different labeling. The number of these states is seven but in a system with a large state space this number can be significantly higher. Direct repair of such a model without using abstraction is impractical.

7 Related Work

To the best of our knowledge this is the first work that suggests the use of abstraction as a means to counter the state space explosion in the search for a solution to the Model Repair problem. In [18], abstract interpretation is used in *program synthesis*, a problem related to Model Repair but much different.

A first attempt for introducing the Model Repair problem in the context of CTL has been done in [4], where a repair algorithm with high computational cost is presented based on the AI techniques of abductive reasoning and theory revision. A formal algorithm for Model Repair in the context of KSs and CTL is presented in [19]. The authors acknowledge that the repair process strongly depends on the size of the model, while they do not implement explicitly in their algorithm how the constraints can be used to handle conjunctive formulas. An effort for making repair applicable to large KSs, is done by the authors of [6]. They use "table systems", a concise representation of KSs, implemented in the NuSMV model checker. A certain limitation for their approach is that table systems cannot represent any KS. In [20], tree-like local model updates are introduced with the aim of making repair process applicable to large scale domains, but their approach is limited to the universal fragment of CTL formulas. For better handling of the constraints in the repair process and thus, ensuring completeness of it, the use of constraint automata for ACTL formulas [14] and the use of protected models for an extension of CTL [5] have been proposed. Both methods are not directly applied to formulas of full CTL. An extension of the Model Repair problem in the context of Labeled Transition Systems has been examined in [9].

The Model Repair problem has been addressed in [2] in the context of probabilistic systems. A slightly different problem, that of Model Revision, has been studied for UNITY properties in [3] and for CTL in [12]. Finally, the *program repair* problem that does not consider KSs as the repair model, has been examined in prior work [17, 15].

8 Conclusions

In this paper, we have shown how abstraction can be used to fight state explosion in Model Repair. Our model-repair framework is based on Kripke Structures, a 3-valued semantics for CTL, and Kripke Modal Transition Systems, and features an abstract-model-repair algorithm for KMTSs. To demonstrate its practical utility, we applied our framework to an Automatic Door Opener system.

As future work, we plan to apply our method to case studies with larger state spaces, and investigate how abstract model repair can be used in different contexts and domains.

Acknowledgments We thank the anonymous reviewers for their valuable comments. Research supported in part by NSF Grants CCF-0926190 and CCF-1018459, and AFOSR Grant FA0550-09-1-0481 . Researchers from Greece were supported by the GSRT/COOPERATION/TRACER Project (09SYN-72-942). The second author's research is supported in part by Canada's NSERC DG 357121-2008, ORF RE03-045, ORE RE04-036, and ORF-RE04-039.

References

1. C. Baier and J.-P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.
2. E. Bartocci, R. Grosu, P. Katsaros, C. R. Ramakrishnan, and S. A. Smolka. Model repair for probabilistic systems. In *TACAS'11*, pages 326–340, Berlin, Heidelberg, 2011. Springer-Verlag.
3. B. Bonakdarpour, A. Ebnenasir, and S. S. Kulkarni. Complexity results in revising UNITY programs. *ACM Trans. Auton. Adapt. Syst.*, 4:5:1–5:28, February 2009.
4. F. Buccafurri, T. Eiter, G. Gottlob, and N. Leone. Enhancing model checking in verification by AI techniques. *Artif. Intell.*, 112:57–104, August 1999.
5. M. Carrillo and D. Rosenblueth. Nondeterministic update of CTL models by preserving satisfaction through protections. In *ATVA*, volume 6996 of *LNCS*, pages 60–74. Springer Berlin / Heidelberg, 2011.
6. M. Carrillo and D. A. Rosenblueth. A method for CTL model update, representing Kripke Structures as table systems. *IJPAM*, 52:401–431, January 2009.
7. E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Trans. Program. Lang. Syst.*, 16:1512–1542, September 1994.
8. D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Trans. Program. Lang. Syst.*, 19:253–291, March 1997.
9. M. de Menezes, S. do Lago Pereira, and L. de Barros. System design modification with actions. In *SBIA 2010*, volume 6404 of *LNCS*, pages 31–40. Springer, 2011.
10. P. Godefroid, M. Huth, and R. Jagadeesan. Abstraction-based model checking using modal transition systems. In *CONCUR '01*, pages 426–440. Springer, 2001.
11. P. Godefroid and R. Jagadeesan. Automatic abstraction using generalized model checking. In *CAV '02*, pages 137–150, London, UK, UK, 2002. Springer-Verlag.
12. P. T. Guerra and R. Wassermann. Revision of CTL models. In *IBERAMIA'10*, pages 153–162, Berlin, Heidelberg, 2010. Springer-Verlag.
13. M. Huth, R. Jagadeesan, and D. A. Schmidt. Modal transition systems: A foundation for three-valued program analysis. In *ESOP '01*, pages 155–169, London, UK, 2001. Springer-Verlag.
14. M. Kelly, F. Pu, Y. Zhang, and Y. Zhou. ACTL local model update with constraints. In *KES'10*, pages 135–144, Berlin, Heidelberg, 2010. Springer-Verlag.
15. R. Samanta, J. V. Deshmukh, and E. A. Emerson. Automatic generation of local repairs for boolean programs. In *FMCAD '08*, pages 27:1–27:10, Piscataway, NJ, USA, 2008. IEEE Press.
16. S. Shoham and O. Grumberg. Monotonic abstraction-refinement for CTL. In *TACAS '04*, pages 546–560. Springer, 2004.
17. S. Staber, B. Jobstmann, and R. Bloem. Finding and fixing faults. In *CHARME '05*, volume 3725 of *LNCS*, pages 35–49. Springer Berlin / Heidelberg, 2005.
18. M. Vechev, E. Yahav, and G. Yorsh. Abstraction-guided synthesis of synchronization. In *POPL '10*, pages 327–338, New York, NY, USA, 2010. ACM.
19. Y. Zhang and Y. Ding. CTL model update for system modifications. *J. Artif. Int. Res.*, 31:113–155, January 2008.
20. Y. Zhang, M. Kelly, and Y. Zhou. Foundations of tree-like local model updates. In *ECAI '10*, pages 615–620, Amsterdam, The Netherlands, 2010. IOS Press.