Logical Methods in Computer Science
Vol. 11(3:11)2015, pp. 1–43
www.lmcs-online.org

Submitted Jul. 2, 2014
Published Sep. 17, 2015

ABSTRACT MODEL REPAIR *

GEORGE CHATZIELEFTHERIOU[a], BORZOO BONAKDARPOUR[b], PANAGIOTIS KATSAROS[c], AND SCOTT A. SMOLKA[d]

[a]Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece
e-mail address: gchatzie@csd.auth.gr

[b]Department of Computing and Software, McMaster University, 1280 Main Street West, Hamilton, ON L8S 4L7, Canada
e-mail address: borzoo@mcmaster.ca

[c]Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece
e-mail address: katsaros@csd.auth.gr

[d]Department of Computer Science, Stony Brook University, Stony Brook, NY 11794-4400, USA
e-mail address: sas@cs.sunysb.edu

Abstract. Given a Kripke structure M and CTL formula ϕ, where M does not satisfy ϕ, the problem of *Model Repair* is to obtain a new model M' such that M' satisfies ϕ. Moreover, the changes made to M to derive M' should be minimum 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 framework aims to repair Kripke Structure models based on a Kripke Modal Transition System abstraction and a 3-valued semantics for CTL. We introduce an abstract-model-repair algorithm for which we prove soundness and semi-completeness, and we study its complexity class. Moreover, a prototype implementation is presented to illustrate the practical utility of abstract-model-repair on an Automatic Door Opener system model and a model of the Andrew File System 1 protocol.

1. Introduction

Given a model M and temporal-logic formula ϕ, *model checking* [16] is the problem of determining whether or not $M \models \phi$. 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 M.

An extended version of the model-checking problem is that of *model repair*: given a model M and temporal-logic formula ϕ, where $M \not\models \phi$, obtain a new model M', such that

2012 ACM CCS: [**Theory of computation**]: Logic—Verification by model checking / Abstraction; [**Software and its Engineering**]: Software organization and properties—Software functional properties—Formal methods—Model checking.

Key words and phrases: Model Repair, Model Checking, Abstraction Refinement.

* A preliminary version of the paper has appeared in [15].

LOGICAL METHODS IN COMPUTER SCIENCE DOI:10.2168/LMCS-11(3:11)2015

$M' \models \phi$. The problem of Model Repair for Kripke structures and Computation Tree Logic (CTL) [28] properties was first introduced in [12].

State explosion is a well known limitation of automated formal methods, such as model checking and model repair, which impedes their application 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* [18, 42, 33, 23, 31] 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 if $\hat{M} \models \phi$ then $M \models \phi$.

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 M, Kripke Modal Transition Systems (KMTSs) for the abstract model $\hat{M}$, and a 3-valued semantics for interpreting CTL over KMTSs [38]. An iterative refinement of the abstract KMTS model takes place whenever the result of the 3-valued CTL model-checking problem is undefined. If the refinement process terminates with a KMTS that violates the CTL property, this property is also falsified by the concrete KS M. Then, the repair process for the refined KMTS is initiated.
- 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 minimum 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 them.
- We introduce an Abstract Model Repair algorithm for KMTSs, which takes into account the aforementioned minimality criterion.
- We prove the soundness of the Abstract Model Repair algorithm for the full CTL and the completeness for a major fragment of it. Moreover, the algorithm's complexity is analyzed with respect to the abstract KMTS model size, which can be much smaller than the concrete KS.
- We illustrate the utility of our approach through a prototype implementation used to repair a flawed Automatic Door Opener system [5] and the Andrew File System 1 protocol. Our experimental results show significant improvement in efficiency compared to a concrete model repair solution.

Organization. The rest of this paper is organized as follows. Sections 2 and 3 introduce KSs, KMTSs, as well as abstraction and refinement based on a 3-valued semantics for CTL. Section 4 defines a metric space for KSs and formally defines the problem of Model Repair. Section 5 presents our framework for Abstract Model Repair, while Section 6 introduces the abstract-model-repair algorithm for KMTSs and discusses its soundness, completeness and complexity properties. Section 7 presents the experimental evaluation of our method through its application to the Andrew File System 1 protocol (AFS1). Section 8 considers related work, while Section 9 concludes with a review of the overall approach and pinpoints directions for future work.

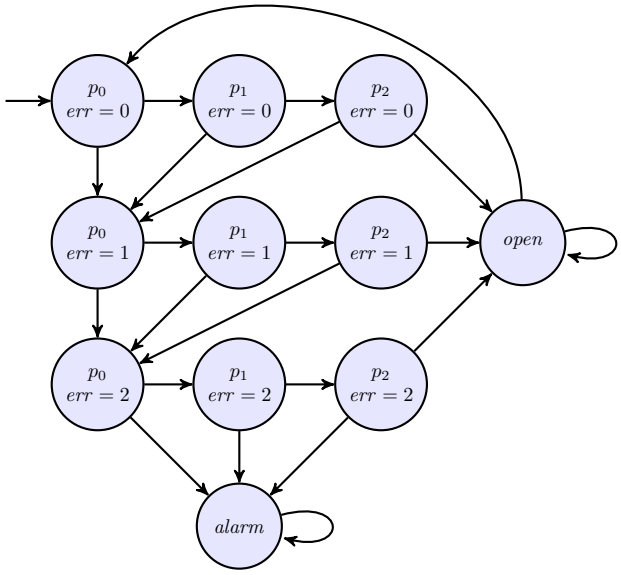


FIGURE 1. The Automatic Door Opener (ADO) System.

2. Kripke Modal Transition Systems

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

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

Definition 2.1. 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.,

$$\forall s \in S : \exists s' \in S : R(s, s').$$

(4) $L : S \to 2^{Lit}$ is a state *labeling function*, such that

$$\forall s \in S : \forall p \in AP : p \in L(s) \Leftrightarrow \neg p \notin L(s).$$ □

The fourth condition in Def. 2.1 ensures that any 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 [5] 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 one and only one 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. 3a, where the set of atomic propositions is $AP = \{q\}$ and $q \equiv (open = true)$.

Definition 2.2. 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) $\hat{L} : \hat{S} \to 2^{Lit}$ 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. Must-transitions are also may-transitions. The "at most one" condition in the fourth part of Def. 2.2 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 outcome ($\perp$). We use the *3-valued semantics* [38] of a CTL formula ϕ at a state $\hat{s}$ of KMTS $\hat{M}$.

Definition 2.3. [38] Let $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$ be a KMTS. The 3-valued semantics of a CTL formula ϕ at a state $\hat{s}$ of $\hat{M}$, denoted as $(\hat{M}, \hat{s}) \models^3 \phi$, is defined inductively as follows:

- If $\phi = \mathit{false}$
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{false}$
- If $\phi = \mathit{true}$
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{true}$
- If $\phi = p$ where $p \in AP$
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{true}$, iff $p \in \hat{L}(\hat{s})$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{false}$, iff $\neg p \in \hat{L}(\hat{s})$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \perp$, otherwise.
- If $\phi = \neg\phi_1$
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{true}$, iff $[(\hat{M}, \hat{s}) \models^3 \phi_1] = \mathit{false}$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{false}$, iff $[(\hat{M}, \hat{s}) \models^3 \phi_1] = \mathit{true}$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \perp$, otherwise.
- If $\phi = \phi_1 \vee \phi_2$
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{true}$, iff $[(\hat{M}, \hat{s}) \models^3 \phi_1] = \mathit{true}$ or $[(\hat{M}, \hat{s}) \models^3 \phi_2] = \mathit{true}$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{false}$, iff $[(\hat{M}, \hat{s}) \models^3 \phi_1] = \mathit{false}$ and $[(\hat{M}, \hat{s}) \models^3 \phi_2] = \mathit{false}$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \perp$, otherwise.
- If $\phi = \phi_1 \wedge \phi_2$
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{true}$, iff $[(\hat{M}, \hat{s}) \models^3 \phi_1] = \mathit{true}$ and $[(\hat{M}, \hat{s}) \models^3 \phi_2] = \mathit{true}$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{false}$, iff $[(\hat{M}, \hat{s}) \models^3 \phi_1] = \mathit{false}$ or $[(\hat{M}, \hat{s}) \models^3 \phi_2] = \mathit{false}$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \perp$, otherwise.
- If $\phi = AX\phi_1$
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{true}$, iff for all $\hat{s}_i$ such that $(\hat{s}, \hat{s}_i) \in R_{may}$, $[(\hat{M}, \hat{s}_i) \models^3 \phi_1] = \mathit{true}$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{false}$, iff there exists some $\hat{s}_i$ such that $(\hat{s}, \hat{s}_i) \in R_{must}$ and $[(\hat{M}, \hat{s}_i) \models^3 \phi_1] = \mathit{false}$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \perp$, otherwise.
- If $\phi = EX\phi_1$
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{true}$, iff there exists $\hat{s}_i$ such that $(\hat{s}, \hat{s}_i) \in R_{must}$ and $[(\hat{M}, \hat{s}_i) \models^3 \phi_1] = \mathit{true}$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{false}$, iff for all $\hat{s}_i$ such that $(\hat{s}, \hat{s}_i) \in R_{may}$, $[(\hat{M}, \hat{s}_i) \models^3 \phi_1] = \mathit{false}$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \perp$, otherwise.

- If $\phi = AG\phi_1$
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{true}$, iff for all may-paths $\pi_{may} = [\hat{s}, \hat{s}_1, \hat{s}_2, ...]$ and for all $\hat{s}_i \in \pi_{may}$ it holds that $[(\hat{M}, \hat{s}_i) \models^3 \phi_1] = \mathit{true}$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{false}$, iff there exists some must-path $\pi_{must} = [\hat{s}, \hat{s}_1, \hat{s}_2, ...]$, such that for some $\hat{s}_i \in \pi_{must}$, $[(\hat{M}, \hat{s}_i) \models^3 \phi_1] = \mathit{false}$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \bot$, otherwise.
- If $\phi = EG\phi_1$
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{true}$, iff there exists some must-path $\pi_{must} = [\hat{s}, \hat{s}_1, \hat{s}_2, ...]$, such that for all $\hat{s}_i \in \pi_{must}$, $[(\hat{M}, \hat{s}_i) \models^3 \phi_1] = \mathit{true}$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{false}$, iff for all may-paths $\pi_{may} = [\hat{s}, \hat{s}_1, \hat{s}_2, ...]$, there is some $\hat{s}_i \in \pi_{may}$ such that $[(\hat{M}, \hat{s}_i) \models^3 \phi_1] = \mathit{false}$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \bot$, otherwise.
- If $\phi = AF\phi_1$
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{true}$, iff for all may-paths $\pi_{may} = [\hat{s}, \hat{s}_1, \hat{s}_2, ...]$, there is a $\hat{s}_i \in \pi_{may}$ such that $[(\hat{M}, \hat{s}_i) \models^3 \phi_1] = \mathit{true}$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{false}$, iff there exists some must-path $\pi_{must} = [\hat{s}, \hat{s}_1, \hat{s}_2, ...]$, such that for all $\hat{s}_i \in \pi_{must}$, $[(\hat{M}, \hat{s}_i) \models^3 \phi_1] = \mathit{false}$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \bot$, otherwise.
- If $\phi = EF\phi_1$
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{true}$, iff there exists some must-path $\pi_{must} = [\hat{s}, \hat{s}_1, \hat{s}_2, ...]$, such that there is some $\hat{s}_i \in \pi_{must}$ for which $[(\hat{M}, \hat{s}_i) \models^3 \phi_1] = \mathit{true}$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{false}$, iff for all may-paths $\pi_{may} = [\hat{s}, \hat{s}_1, \hat{s}_2, ...]$ and for all $\hat{s}_i \in \pi_{may}$, $[(\hat{M}, \hat{s}_i) \models^3 \phi_1] = \mathit{false}$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \bot$, otherwise.
- If $\phi = A(\phi_1 \, U \, \phi_2)$
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{true}$, iff for all may-paths $\pi_{may} = [\hat{s}, \hat{s}_1, \hat{s}_2, ...]$, there is $\hat{s}_i \in \pi_{may}$ such that $[(\hat{M}, \hat{s}_i) \models^3 \phi_2] = \mathit{true}$ and $\forall j < i : [(\hat{M}, \hat{s}_j) \models^3 \phi_1] = \mathit{true}$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{false}$, iff there exists some must-path $\pi_{must} = [\hat{s}, \hat{s}_1, \hat{s}_2, ...]$, such that
    i. for all $0 \leq k < |\pi_{must}|$:
       $(\forall j < k : [(\hat{M}, \hat{s}_j) \models^3 \phi_1] \neq \mathit{false}) \Rightarrow ([(\hat{M}, \hat{s}_k) \models^3 \phi_2] = \mathit{false})$
    ii. (for all $0 \leq k < |\pi_{must}| : [(\hat{M}, \hat{s}_k) \models^3 \phi_2] \neq \mathit{false}) \Rightarrow |\pi_{must}| = \infty$
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \bot$, otherwise.
- If $\phi = E(\phi_1 U \phi_2)$
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{true}$, iff there exists some must-path $\pi_{must} = [\hat{s}, \hat{s}_1, \hat{s}_2, ...]$ such that there is a $\hat{s}_i \in \pi_{must}$ with $[(\hat{M}, \hat{s}_i) \models^3 \phi_2] = \mathit{true}$ and for all $j < i, [(\hat{M}, \hat{s}_j) \models^3 \phi_1] = \mathit{true}$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{false}$, iff for all may-paths $\pi_{may} = [\hat{s}, \hat{s}_1, \hat{s}_2, ...]$
    i. for all $0 \leq k < |\pi_{may}|$:
       $(\forall j < k : [(\hat{M}, \hat{s}_j) \models^3 \phi_1] \neq \mathit{false}) \Rightarrow ([(\hat{M}, \hat{s}_k) \models^3 \phi_2] = \mathit{false})$
    ii. (for all $0 \leq k < |\pi_{may}| : [(\hat{M}, \hat{s}_k) \models^3 \phi_2] \neq \mathit{false}) \Rightarrow |\pi_{may}| = \infty$
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \bot$, otherwise. □

From the 3-valued CTL semantics, it follows that must-transitions are used to check the truth of existential CTL properties, while may-transitions are used to check the truth of universal CTL properties. This works inversely for checking the refutation of CTL properties. In what follows, we use $\models$ instead of $\models^3$ in order to refer to the 3-valued satisfaction relation.

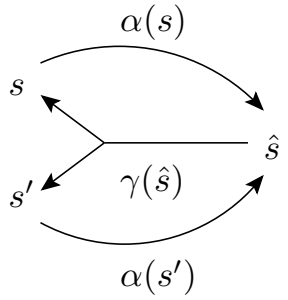


FIGURE 2. Abstraction and Concretization.

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 result of model checking a property ϕ in the abstract model is preserved in the concrete model. This can be achieved if the abstract model is built with certain requirements [18, 31].

Definition 3.1. Given a KS $M = (S, S_0, R, L)$ and a pair of total functions $(\alpha : S \to \hat{S}, \gamma : \hat{S} \to 2^S)$ such that

$$\forall s \in S : \forall \hat{s} \in \hat{S} : (\alpha(s) = \hat{s} \Leftrightarrow s \in \gamma(\hat{s}))$$

the KMTS $\alpha(M) = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$ 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 $\forall s \in \gamma(\hat{s}) : lit \in L(s)$
(3) $R_{must} = \{(\hat{s_1}, \hat{s_2}) \mid \forall s_1 \in \gamma(\hat{s_1}) : \exists s_2 \in \gamma(\hat{s_2}) : (s_1, s_2) \in R\}$
(4) $R_{may} = \{(\hat{s_1}, \hat{s_2}) \mid \exists s_1 \in \gamma(\hat{s_1}) : \exists s_2 \in \gamma(\hat{s_2}) : (s_1, s_2) \in R\}$ □

For a given KS M and a pair of abstraction and concretization functions α and γ, Def. 3.1 introduces the KMTS $\alpha(M)$ defined over the set $\hat{S}$ of *abstract states*. In our AMR framework, we view M as the *concrete model* and the KMTS $\alpha(M)$ as the *abstract model*. Any two concrete states s_1 and s_2 of M are abstracted by α to a state $\hat{s}$ of $\alpha(M)$ if and only if s_1, s_2 are elements of the set $\gamma(\hat{s})$ (see Fig 2). A state of $\alpha(M)$ is initial *if and only if* at least one of its concrete states is initial as well. An atomic proposition in an abstract state is true (respectively, false), *only if* it is also true (respectively, false) in all of its concrete states. This means that the value of an atomic proposition may be unknown at a state of $\alpha(M)$. A must-transition from $\hat{s_1}$ to $\hat{s_2}$ of $\alpha(M)$ exists, if and only if there are transitions from all states of $\gamma(\hat{s_1})$ to at least one state of $\gamma(\hat{s_2})$ ($\forall\exists - condition$). Respectively, a may-transition from $\hat{s_1}$ to $\hat{s_2}$ of $\alpha(M)$ exists, if and only if there is at least one transition from some state of $\gamma(\hat{s_1})$ to some state of $\gamma(\hat{s_2})$ ($\exists\exists - condition$).

Definition 3.2. Given a pair of total functions $(\alpha : S \to \hat{S}, \gamma : \hat{S} \to 2^S)$ such that

$$\forall s \in S : \forall \hat{s} \in \hat{S} : (\alpha(s) = \hat{s} \Leftrightarrow s \in \gamma(\hat{s}))$$

and a KMTS $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$, the set of KSs $\gamma(\hat{M}) = \{M \mid M = (S, S_0, R, L)\}$ is defined such that for all $M \in \gamma(\hat{M})$ the following conditions hold:

(1) $s \in S_0$ iff $\alpha(s) \in \hat{S}_0$
(2) $lit \in L(s)$ if $lit \in \hat{L}(\alpha(s))$

(3) $(s_1, s_2) \in R$ iff
- $\exists s_1' \in \gamma(\alpha(s_1)) : \exists s_2' \in \gamma(\alpha(s_2)) : (\alpha(s_1), \alpha(s_2)) \in R_{may}$ and,
- $\forall s_1' \in \gamma(\alpha(s_1)) : \exists s_2' \in \gamma(\alpha(s_2)) : (\alpha(s_1), \alpha(s_2)) \in R_{must}$ □

For a given KMTS $\hat{M}$ and a pair of abstraction and concretization functions α and γ, Def. 3.2 introduces a set $\gamma(\hat{M})$ of *concrete* KSs. A state s of a KS $M \in \gamma(\hat{M})$ is initial if its abstract state $\alpha(s)$ is also initial. An atomic proposition in a concrete state s is true (respectively, false) if it is also true (respectively, false) in its abstract state $\alpha(s)$. A transition from a concrete state s_1 to another concrete state s_2 exists, if and only if

- there are concrete states $s_1' \in \gamma(\alpha(s_1))$ and $s_2' \in \gamma(\alpha(s_2))$, where $(\alpha(s_1), \alpha(s_2)) \in R_{may}$, and
- there is at least one concrete state $s_2' \in \gamma(\alpha(s_2))$ such that for all $s_1' \in \gamma(\alpha(s_1))$ it holds that $(\alpha(s_1), \alpha(s_2)) \in R_{must}$.

Abstract Interpretation. A pair of abstraction and concretization functions can be defined within an *Abstract Interpretation* [20, 21] framework. Abstract interpretation is a theory for a set of abstraction techniques, for which important properties for the model checking problem have been proved [23, 24].

Definition 3.3. [23, 32] Let $M = (S, S_0, R, L)$ be a concrete KS and $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$ 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 is exists $\hat{s}' \in \hat{S}$ such that $r_{may} = (\hat{s}, \hat{s}') \in R_{may}$ and $(s', \hat{s}') \in H$.
- if $r_{must} = (\hat{s}, \hat{s}') \in R_{must}$, then there exists $s' \in S$ such that $r = (s, s') \in R$ and $(s', \hat{s}') \in H$. □

The abstraction function α of Def. 3.1 is a mixed simulation for the KS M and its abstract KMTS $\alpha(M)$.

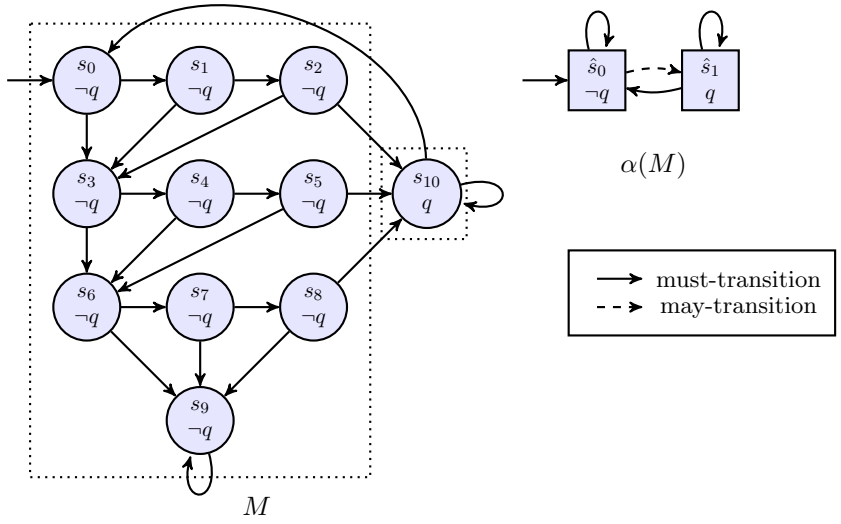
Theorem 3.4. [32] *Let $H \subseteq S \times \hat{S}$ be a mixed simulation from a KS $M = (S, S_0, R, L)$ to a KMTS $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$. Then, for every CTL formula ϕ and every $(s, \hat{s}) \in H$ it holds that*

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

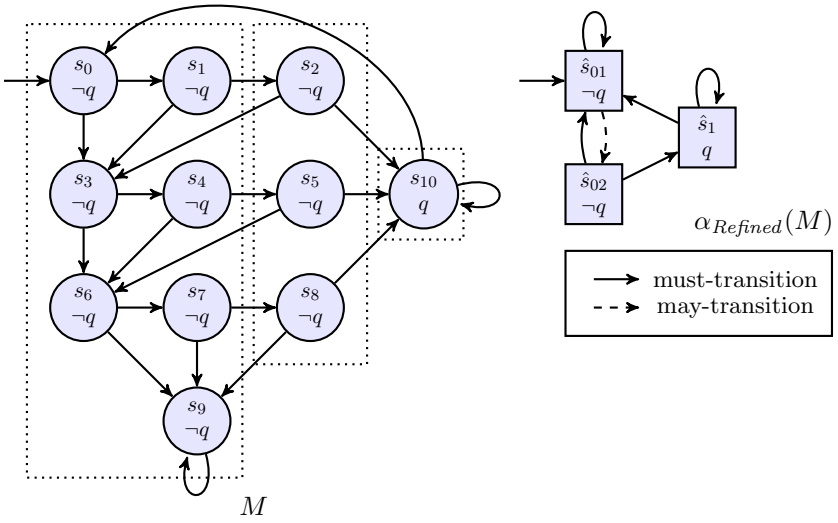
Theorem 3.4 ensures that if a CTL formula ϕ has a definite truth value (i.e., true or false) in the abstract KMTS, then it has the same truth value in the concrete KS. When we get $\bot$ from the 3-valued model checking of a CTL formula ϕ, the result of model checking property ϕ on the corresponding KS can be either true or false.

Example. An abstract KMTS $\hat{M}$ is presented in Fig. 3a, where all the states labeled by q are grouped together, as are all states labeled by $\neg q$.

3.2. **Refinement.** When the outcome of verifying a CTL formula ϕ on an abstract model using the 3-valued semantics is $\bot$, then a *refinement* step is needed to acquire a more *precise* abstract model. In the literature, there are refinement approaches for the 2-valued CTL semantics [17, 19, 22], as well as a number of techniques for the 3-valued CTL model



(A) The KS and initial KMTS.



(B) The KS and refined KMTS.

FIGURE 3. The KS and KMTSs for the ADO system.

checking [31, 46, 47, 35]. The refinement technique that we adopt is an automated two-step process based on [17, 46]:

(1) Identify a *failure state* in $\alpha(M)$ using the algorithms in [17, 46]; the cause of failure for a state $\hat{s}$ stems from an atomic proposition having an undefined value in $\hat{s}$, or from an outgoing may-transition from $\hat{s}$.

(2) Produce the abstract KMTS $\alpha_{Refined}(M)$, where $\alpha_{Refined}$ is a new abstraction function as in Def. 3.1, such that the identified failure state is refined into two states. If the cause of failure is an undefined value of an atomic proposition in $\hat{s}$, then $\hat{s}$ is split into states $\hat{s}_1$ and $\hat{s}_2$, such that the atomic proposition is true in $\hat{s}_1$ and false in $\hat{s}_2$. Otherwise, if the cause of failure is an outgoing may-transition from $\hat{s}$, then $\hat{s}$ is split into states $\hat{s}_1$ and $\hat{s}_2$, such that there is an outgoing must-transition from $\hat{s}_1$ and no outgoing may- or must-transition from $\hat{s}_2$.

The described refinement technique does not necessarily converge to an abstract KMTS with a definite model checking result. A promising approach in order to overcome this restriction is by using a different type of abstract model, as in [46], where the authors propose the use of Generalized KMTSs, which ensure monotonicity of the refinement process.

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 the door is required. This property can be expressed in CTL as $\phi = AGEXq$. Observe that in $\alpha(M)$ of Fig. 3a, the absence of a must-transition from $\hat{s}_0$ to $\hat{s}_1$, where $[(\alpha(M), \hat{s}_1) \models q] = true$, in conjunction with the existence of a may-transition from $\hat{s}_0$ to $\hat{s}_1$, i.e. to a state where $[(\alpha(M), \hat{s}_1) \models q] = true$, results in an undefined model-checking outcome for $[(\alpha(M), \hat{s}_0) \models \phi]$. Notice that state $\hat{s}_0$ is the failure state, and the may-transition from $\hat{s}_0$ to $\hat{s}_1$ is 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 $\hat{s}_1$. Thus, the may-transition which caused the undefined outcome is eliminated and for the refined KMTS $\alpha_{Refined}(M)$ it holds that $[\alpha_{Refined}(M), \hat{s}_1) \models \phi] = false$. The initial KS and the refined KMTS $\alpha_{Refined}(M)$ are shown in Fig. 3b.

4. The Model Repair Problem

In this section, we formulate the problem of Model Repair. A metric space over Kripke structures is defined to quantify their structural differences. This allows us taking into account the *minimality of changes* criterion in Model Repair.
Let π be a function on the set of all functions $f : X \to Y$ such that:

$$\pi(f) = \{(x, f(x)) \mid x \in X\}$$

A *restriction operator* (denoted by $\restriction$) for the domain of function f is defined such that for $X_1 \subseteq X$,

$$f \restriction_{X_1} = \{(x, f(x)) \mid x \in X_1\}$$

By S^C, we denote the complement of a set S.

Definition 4.1. For any two $M = (S, S_0, R, L)$ and $M' = (S', S'_0, R', L')$ in the set K_M of all KSs, 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 $R_{IN} \subseteq R^C$, $R_{OUT} \subseteq R$,
$L' = S' \to 2^{LIT}$,

the *distance function* d over K_M is defined as follows:

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

with $A \,\Delta\, B$ representing 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 spaces, the number of differences $|R \,\Delta\, R'|$ in their transition relation and the number of common states with altered labeling.

Proposition 4.2. *The ordered pair (K_M, d) is a metric space.*

Proof. We use the fact that the cardinality of the symmetric difference between any two sets is a distance metric. It holds that:

(1) $|S\Delta S'| \geq 0$, $|R\Delta R'| \geq 0$ and $|\pi(L \restriction_{S\cap S'})\Delta\pi(L' \restriction_{S\cap S'})| \geq 0$ (non-negativity)
(2) $|S\Delta S'| = 0$ iff $S = S'$, $|R\Delta R'| = 0$ iff $R = R'$ and $|\pi(L \restriction_{S\cap S'})|\Delta|\pi(L' \restriction_{S\cap S'})| = 0$ iff $\pi(L \restriction_{S\cap S'}) = \pi(L' \restriction_{S\cap S'})$ (identity of indiscernibles)
(3) $|S\Delta S'| = |S'\Delta S|$, $|R\Delta R'| = |R'\Delta R|$ and $|\pi(L \restriction_{S\cap S'})\Delta\pi(L' \restriction_{S\cap S'})| = |\pi(L' \restriction_{S\cap S'})\Delta\pi(L \restriction_{S\cap S'})|$(symmetry)
(4) $|S'\Delta S''| \leq |S'\Delta S| + |S\Delta S''|$, $|R'\Delta R''| \leq |R'\Delta R| + |R\Delta R''|$, $|\pi(L' \restriction_{S'\cap S''})\Delta\pi(L''|_{S'\cap S''})| \leq |\pi(L' \restriction_{S'\cap S})\Delta\pi(L \restriction_{S'\cap S})| + |\pi(L \restriction_{S\cap S''})\Delta\pi(L''|_{S\cap S''})|$ (triangle inequality)

We will prove that d is a metric on K_M. Suppose $M, M', M'' \in K_M$

- It easily follows from (1) that $d(M, M') \geq 0$ (non-negativity)
- From (2), $d(M, M') = 0$ iff $M = M'$ (identity of indiscernibles)
- Adding the equations in (3), results in $d(M, M') = d(M', M)$ (symmetry)
- If we add the inequalities in (4), then we get $d(M', M'') \leq d(M', M) + d(M, M'')$ (triangle inequality)

So, the proposition is true. □

Definition 4.3. For any two $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$ and $\hat{M}' = (\hat{S}', \hat{S_0}', R'_{must}, R'_{may}, \hat{L}')$ in the set $K_{\hat{M}}$ of all KMTSs, where

$\hat{S}' = (\hat{S} \cup \hat{S}_{IN}) - \hat{S}_{OUT}$ for some $\hat{S}_{IN} \subseteq \hat{S}^C$, $\hat{S}_{OUT} \subseteq \hat{S}$,
$\hat{R}'_{must} = (\hat{R}_{must} \cup \hat{R}_{IN}) - \hat{R}_{OUT}$ for some $\hat{R}_{IN} \subseteq \hat{R}^C_{must}$, $\hat{R}_{OUT} \subseteq \hat{R}_{must}$,
$\hat{R}'_{may} = (\hat{R}_{may} \cup \hat{R}'_{IN}) - \hat{R}'_{OUT}$ for some $\hat{R}'_{IN} \subseteq \hat{R}^C_{may}$, $\hat{R}'_{OUT} \subseteq \hat{R}_{may}$,
$\hat{L}' = \hat{S}' \to 2^{LIT}$,

the *distance function* $\hat{d}$ over $K_{\hat{M}}$ is defined as follows:

$$\hat{d}(M, M') = |\hat{S} \,\Delta\, \hat{S}'| + |\hat{R}_{must} \,\Delta\, \hat{R}'_{must}| + |(\hat{R}_{may} - \hat{R}_{must}) \,\Delta\, (\hat{R}'_{may} - \hat{R}'_{must})| + \frac{|\pi(\hat{L} \restriction_{\hat{S}\cap\hat{S}'}) \,\Delta\, \pi(\hat{L}' \restriction_{\hat{S}\cap\hat{S}'})|}{2}$$

with $A\Delta B$ representing the symmetric difference $(A - B) \cup (B - A)$.

We note that $\hat{d}$ counts the differences between $\hat{R}'_{may}$ and $\hat{R}_{may}$, and those between $\hat{R}'_{must}$ and $\hat{R}_{must}$ separately, while avoiding to count the differences in the latter case twice (we remind that must-transitions are also included in $\hat{R}_{may}$).

Proposition 4.4. *The ordered pair $(K_{\hat{M}}, \hat{d})$ is a metric space.*

Proof. The proof is done in the same way as in Prop. 4.2. □

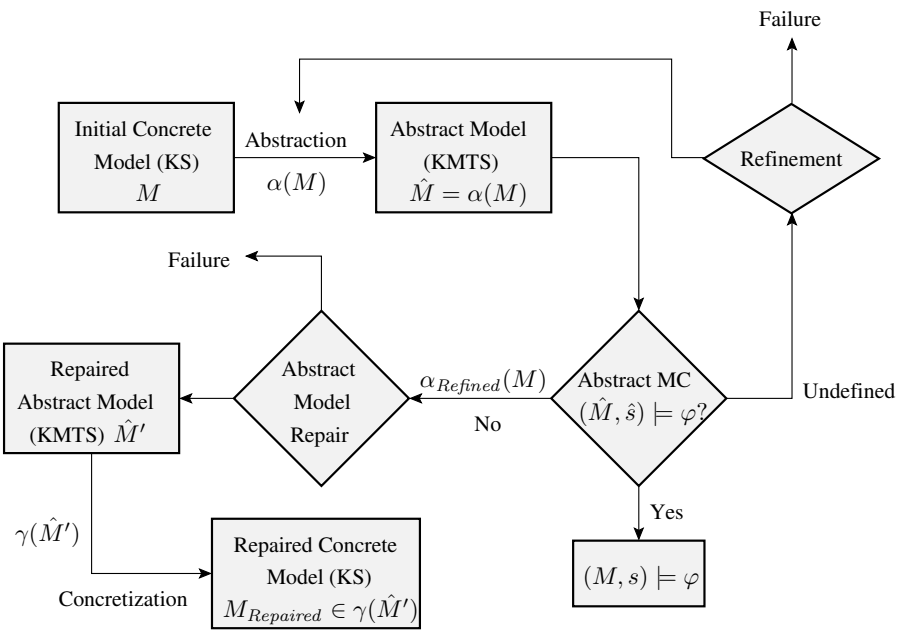


FIGURE 4. Abstract Model Repair Framework.

Definition 4.5. Given a KS M and a CTL formula ϕ where $M \not\models \phi$, the Model Repair problem is to find a KS M', such that $M' \models \phi$ and $d(M, M')$ is minimum with respect to all such M'.

The Model Repair problem aims at modifying a KS such that the resulting KS satisfies a CTL formula that was violated before. The distance function d of Def. 4.1 features all the attractive properties of a distance metric. Given that no quantitative interpretation exists for predicates and logical operators in CTL, d can be used in a model repair solution towards selecting minimum changes to the modified KS.

5. The Abstract Model Repair Framework

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

5.1. The Abstract Model Repair Process.

The process steps shown in Fig. 4 rely on the KMTS abstraction of Def. 3.1. 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. 3.1.

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

Case 1.: If the result is *true*, then, according to Theorem 3.4, $(M, s) \models \phi$ and there is no need to repair M.

Case 2.: If the result is *undefined*, then a refinement of $\hat{M}$ takes place, and:
Case 2.1.: If an $\hat{M}_{Refined}$ is found, the control is transferred to Step 2.
Case 2.2.: If a refined KMTS cannot be retrieved, the repair process terminates with a failure.
Case 3.: If the result is *false*, then, from Theorem 3.4, $(M, s) \not\models \phi$ and the repair process is enacted; the control is transferred to Step 3.
Step 3.: The *AbstractRepair* algorithm is called for the abstract KMTS ($\hat{M}_{Refined}$ or $\hat{M}$ if no refinement has occurred), the state $\hat{s}$ and the property ϕ.
Case 1.: *AbstractRepair* returns an $\hat{M}'$ for which $(\hat{M}', \hat{s}) \models \phi$.
Case 2.: *AbstractRepair* fails to find an $\hat{M}'$ for which the property holds true.
Step 4.: If *AbstractRepair* returns an $\hat{M}'$, then the process ends with selecting the subset of KSs from $\gamma(\hat{M}')$, with elements whose distance d from the KS M is minimum with respect to all the KSs in $\gamma(\hat{M}')$.

5.2. **Basic Repair Operations.** We decompose the KMTS repair process into seven basic repair operations:

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

5.2.1. *Adding a must-transition.*

Definition 5.1 (AddMust)**.** For a given KMTS $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$ and $\hat{r}_n = (\hat{s}_1, \hat{s}_2) \notin R_{must}$, $AddMust(\hat{M}, \hat{r}_n)$ is the KMTS $\hat{M}' = (\hat{S}, \hat{S}_0, R'_{must}, R'_{may}, \hat{L})$ such that $R'_{must} = R_{must} \cup \{\hat{r}_n\}$ and $R'_{may} = R_{may} \cup \{\hat{r}_n\}$. □

Since $R_{must} \subseteq R_{may}$, $\hat{r}_n$ must also be added to R_{may}, resulting in a new may-transition if $\hat{r}_n \notin R_{may}$. Fig. 5 shows how the basic repair operation *AddMust* modifies a given KMTS. The newly added transitions are in bold.

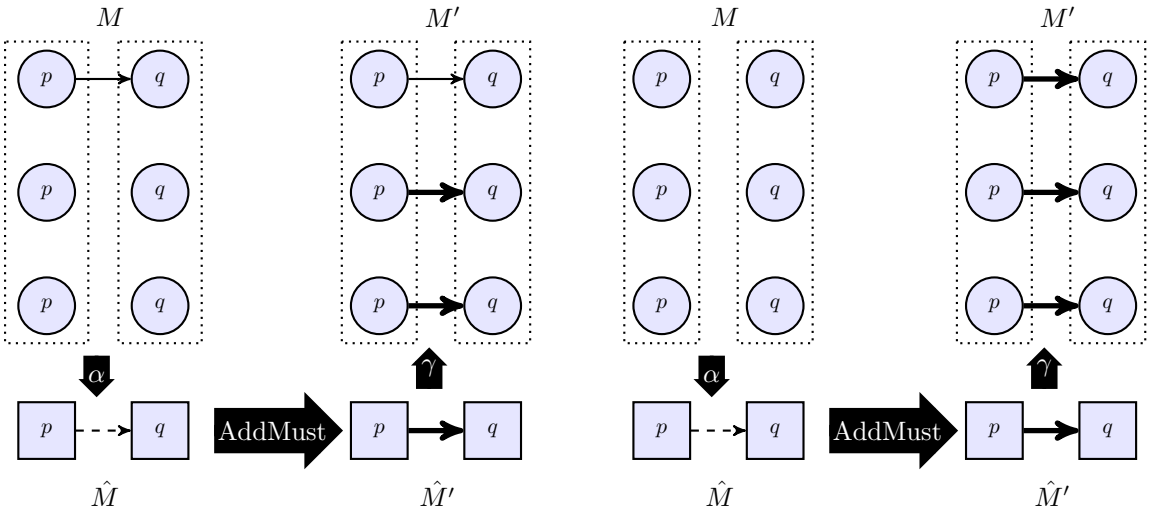
Proposition 5.2. *For any $\hat{M}' = AddMust(\hat{M}, \hat{r}_n)$, it holds that $\hat{d}(\hat{M}, \hat{M}') = 1$.* □

Definition 5.3. Let $M = (S, S_0, R, L)$ be a KS and let $\alpha(M) = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$ be the abstract KMTS derived from M as in Def. 3.1. Also, let $\hat{M}' = AddMust(\alpha(M), \hat{r}_n)$ for some $\hat{r}_n = (\hat{s}_1, \hat{s}_2) \notin R_{must}$. The set $K_{min} \subseteq \gamma(\hat{M}')$ with all KSs, whose distance d from M is minimized is:

$$K_{min} = \{M' \mid M' = (S, S_0, R \cup R_n, L)\} \tag{5.1}$$

where R_n is given for one $s_2 \in \gamma(\hat{s}_2)$ as follows:

$$R_n = \bigcup_{s_1 \in \gamma(\hat{s}_1)} \{(s_1, s_2) \mid \nexists s \in \gamma(\hat{s}_2) : (s_1, s) \in R\}$$ □



(A) May-transition exists (B) May-transition does not exist

FIGURE 5. *AddMust*: Adding a new must-transition

Def. 5.3 implies that when the *AbstractRepair* algorithm applies *AddMust* on the abstract KMTS $\hat{M}$, then a set of KSs is retrieved from the concretization of $\hat{M}'$. The same holds for all other basic repair operations and consequently, when *AbstractRepair* finds a repaired KMTS, one or more KSs can be obtained for which property ϕ holds.

Proposition 5.4. *For all $M' \in K_{min}$, it holds that $1 \leq d(M, M') \leq |S|$.*

Proof. Recall that

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

Since $|S \Delta S'| = 0$ and $|\pi(L \restriction_{S \cap S'}) \Delta \pi(L' \restriction_{S \cap S'})| = 0$, $d(M, M') = |R \Delta R'| = |R - R'| + |R' - R| = 0 + |R_n|$. Since $|R_n| \geq 1$ and $|R_n| \leq |S|$, it is proved that $1 \leq d(M, M') \leq |S|$. □

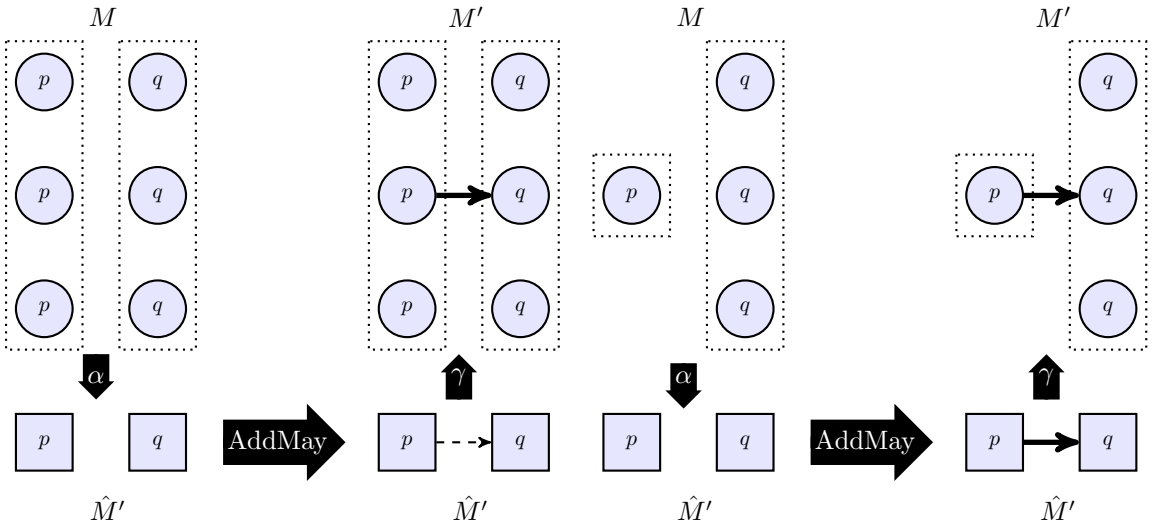
From Prop. 5.4, we conclude that a lower and upper bound exists for the distance between M and any $M' \in K_{min}$.

5.2.2. *Adding a may-transition.*

Definition 5.5 (AddMay)**.** For a given KMTS $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$ and $\hat{r}_n = (\hat{s}_1, \hat{s}_2) \notin R_{may}$, $AddMay(\hat{M}, \hat{r}_n)$ is the KMTS $\hat{M}' = (\hat{S}, \hat{S}_0, R'_{must}, R'_{may}, \hat{L})$ such that $R'_{must} = R_{must} \cup \{\hat{r}_n\}$ if $|S_1| = 1$ or $R'_{must} = R_{must}$ if $|S_1| > 1$ for $S_1 = \{s_1 \mid s_1 \in \gamma(\hat{s}_1)\}$ and $R'_{may} = R_{may} \cup \{\hat{r}_n\}$. □

From Def. 5.5, we conclude that there are two different cases in adding a new may-transition $\hat{r}_n$; adding also a must-transition or not. In fact, $\hat{r}_n$ is also a must-transition if and only if the set of the corresponding concrete states of $\hat{s}_1$ is a singleton. Fig. 6 displays the two different cases of applying basic repair operation *AddMay* to a KMTS.

Proposition 5.6. *For any $\hat{M}' = AddMay(\hat{M}, \hat{r}_n)$, it holds that $\hat{d}(\hat{M}, \hat{M}') = 1$.* □



(A) Only may-transition is added (B) Must-transition is also added

FIGURE 6. *AddMay*: Adding a new must-transition

Definition 5.7. Let $M = (S, S_0, R, L)$ be a KS and let $\alpha(M) = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$ be the abstract KMTS derived from M as in Def. 3.1. Also, let $\hat{M}' = AddMay(\alpha(M), \hat{r}_n)$ for some $\hat{r}_n = (\hat{s}_1, \hat{s}_2) \notin R_{may}$. The set $K_{min} \subseteq \gamma(\hat{M}')$ with all KSs, whose structural distance d from M is minimized is given by:

$$K_{min} = \{M' \mid M' = (S, S_0, R \cup \{r_n\}, L)\} \tag{5.2}$$

where $r_n \in R_n$ and $R_n = \{r_n = (s_1, s_2) \mid s_1 \in \gamma(\hat{s}_1), s_2 \in \gamma(\hat{s}_2) \text{ and } r_n \notin R\}$. □

Proposition 5.8. *For all $M' \in K_{min}$, it holds that $d(M, M') = 1$.*

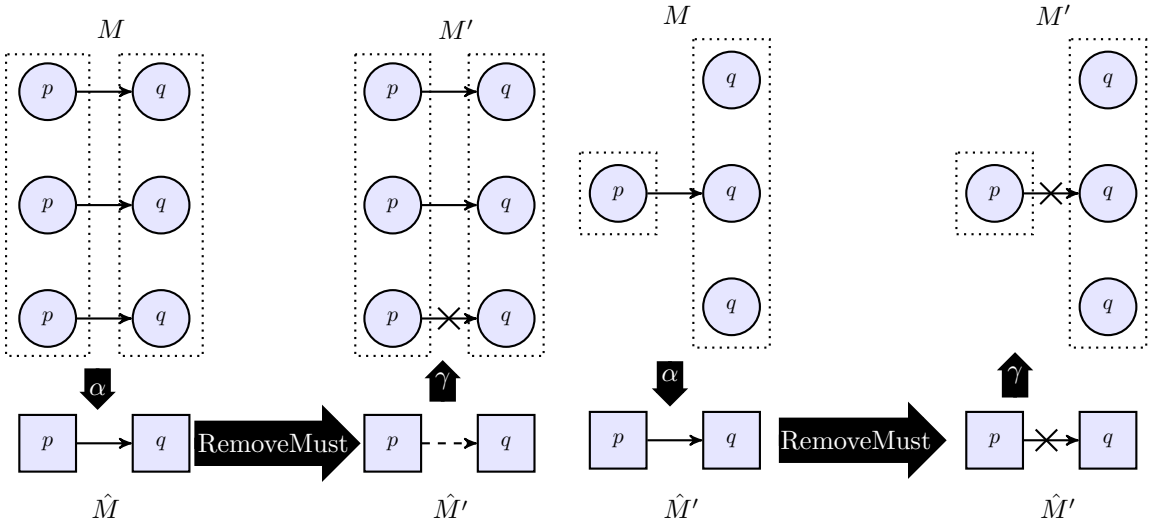
Proof. $d(M, M') = |S \Delta S'| + |R \Delta R'| + \frac{|\pi(L \restriction_{S \cap S'}) \Delta \pi(L' \restriction_{S \cap S'})|}{2}$. Because $|S \Delta S'| = 0$ and $|\pi(L \restriction_{S \cap S'}) \Delta \pi(L' \restriction_{S \cap S'})| = 0$, $d(M, M') = |R \Delta R'| = |R - R'| + |R' - R| = 0 + |\{r_n\}| = 1$. So, we prove that $d(M, M') = 1$. □

5.2.3. *Removing a must-transition.*

Definition 5.9 (RemoveMust)**.** For a given KMTS $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$ and $\hat{r}_m = (\hat{s}_1, \hat{s}_2) \in R_{must}$, $RemoveMust(\hat{M}, \hat{r}_m)$ is the KMTS $\hat{M}' = (\hat{S}, \hat{S}_0, R'_{must}, R'_{may}, \hat{L})$ such that $R'_{must} = R_{must} - \{\hat{r}_m\}$ and $R'_{may} = R_{may} - \{\hat{r}_m\}$ if $|S_1| = 1$ or $R'_{may} = R_{may}$ if $|S_1| > 1$ for $S_1 = \{s_1 \mid s_1 \in \gamma(\hat{s}_1)\}$. □

Removing a must-transition $\hat{r}_m$, in some special and maybe rare cases, could also result in the deletion of the may-transition $\hat{r}_m$ as well. In fact, this occurs if transitions to the concrete states of $\hat{s}_2$ exist only from one concrete state of the corresponding ones of $\hat{s}_1$. These two cases for function *RemoveMust* are presented graphically in Fig. 7.

Proposition 5.10. *For any $\hat{M}' = RemoveMust(\hat{M}, \hat{r}_m)$, it holds that $\hat{d}(\hat{M}, \hat{M}') = 1$.* □



(A) May-transition is not removed (B) May-transition is also removed

FIGURE 7. *RemoveMust*: Removing an existing must-transition

Definition 5.11. Let $M = (S, S_0, R, L)$ be a KS and let $\alpha(M) = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$ be the abstract KMTS derived from M as in Def. 3.1. Also, let $\hat{M}' = RemoveMust(\alpha(M), \hat{r}_m)$ for some $\hat{r}_m = (\hat{s}_1, \hat{s}_2) \in R_{must}$. The set $K_{min} \subseteq \gamma(\hat{M}')$ with all KSs, whose structural distance d from M is minimized is given by:

$$K_{min} = \{M' \mid M' = (S, S_0, R - \{R_m\}, L)\} \tag{5.3}$$

where R_m is given for one $s_1 \in \gamma(\hat{s}_1)$ as follows:

$$R_m = \bigcup_{s_2 \in \gamma(\hat{s}_2)} \{(s_1, s_2) \in R\}$$ □

Proposition 5.12. *For M', it holds that $1 \leq d(M, M') \leq |S|$.*

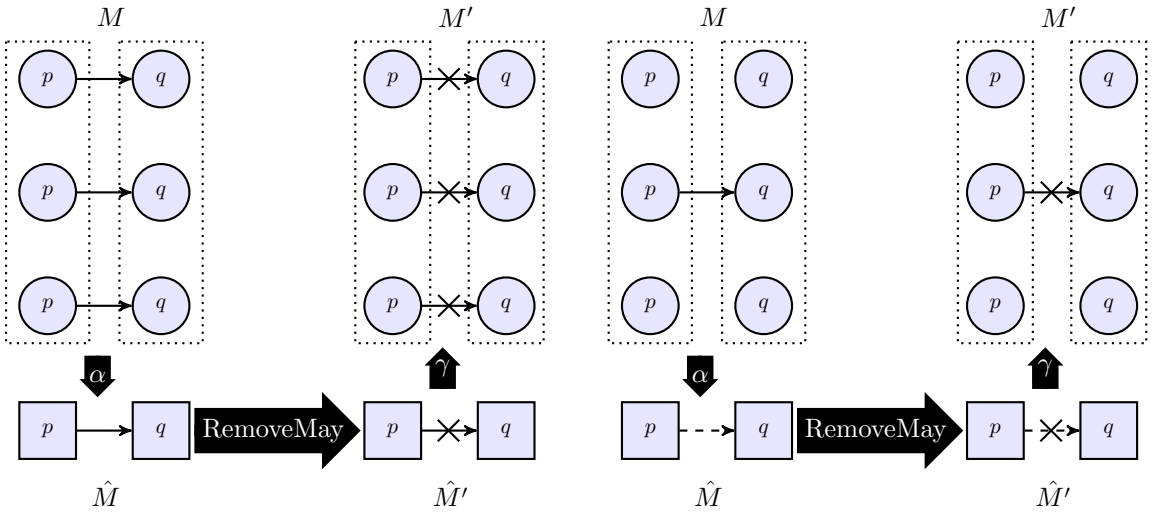
Proof. $d(M, M') = |S \Delta S'| + |R \Delta R'| + \frac{|\pi(L \restriction_{S \cap S'}) \Delta \pi(L' \restriction_{S \cap S'})|}{2}$. Because $|S \Delta S'| = 0$ and $|\pi(L \restriction_{S \cap S'}) \Delta \pi(L' \restriction_{S \cap S'})| = 0$, $d(M, M') = |R \Delta R'| = |R - R'| + |R' - R| = |R_m| + 0 = |R_m|$. It holds that $|R_m| \geq 1$ and $|R_m| \leq |S|$. So, we proved that $1 \leq d(M, M') \leq |S|$. □

5.2.4. *Removing a may-transition.*

Definition 5.13 (RemoveMay)**.** For a given KMTS $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$ and $\hat{r}_m = (\hat{s}_1, \hat{s}_2) \in R_{may}$, $RemoveMay(\hat{M}, \hat{r}_m)$ is the KMTS $\hat{M}' = (\hat{S}, \hat{S}_0, R'_{must}, R'_{may}, \hat{L})$ such that $R'_{must} = R_{must} - \{\hat{r}_m\}$ and $R'_{may} = R_{may} - \{\hat{r}_m\}$. □

Def. 5.13 ensures that removing a may-transition $\hat{r}_m$ implies the removal of a must-transition, if $\hat{r}_m$ is also a must-transition. Otherwise, there are not any changes in the set of must-transitions R_{must}. Fig. 8 shows how function *RemoveMay* works in both cases.

Proposition 5.14. *For any $\hat{M}' = RemoveMay(\hat{M}, \hat{r}_m)$, it holds that $\hat{d}(\hat{M}, \hat{M}') = 1$.* □



(A) May-transition is also a must-transition

(B) May-transition is not a must-transition

FIGURE 8. *RemoveMay*: Removing an existing may-transition

Definition 5.15. Let $M = (S, S_0, R, L)$ be a KS and let $\alpha(M) = (\hat{S}, \hat{S_0}, R_{must}, R_{may}, \hat{L})$ be the abstract KMTS derived from M as in Def. 3.1. Also, let $\hat{M}' = RemoveMay(\alpha(M), \hat{r}_m)$ for some $\hat{r}_m = (\hat{s}_1, \hat{s}_2) \in R_{may}$ with $\hat{s}_1, \hat{s}_2 \in \hat{S}$. The KS $M' \in \gamma(\hat{M}')$, whose structural distance d from M is minimized is given by:

$$M' = (S, S_0, R - R_m, L\} \tag{5.4}$$

where $R_m = \{r_m = (s_1, s_2) \mid s_1 \in \gamma(\hat{s}_1), s_2 \in \gamma(\hat{s}_2) \text{ and } r_m \in R\}$. □

Proposition 5.16. *For M', it holds that $1 \leq d(M, M') \leq |S|^2$.*

Proof. $d(M, M') = |S \Delta S'| + |R \Delta R'| + \frac{|\pi(L \restriction_{S \cap S'}) \Delta \pi(L' \restriction_{S \cap S'})|}{2}$. Because $|S \Delta S'| = 0$ and $|\pi(L \restriction_{S \cap S'}) \Delta \pi(L' \restriction_{S \cap S'})| = 0$, $d(M, M') = |R \Delta R'| = |R - R'| + |R' - R| = 0 + |R_m| = |R_m|$. It holds that $|R_m| \geq 1$ and $|R_m| \leq |S|^2$. So, we proved that $1 \leq d(M, M') \leq |S|^2$. □

5.2.5. *Changing the labeling of a KMTS state.*

Definition 5.17 (ChangeLabel)**.** For a given KMTS $\hat{M} = (\hat{S}, \hat{S_0}, R_{must}, R_{may}, \hat{L})$, a state $\hat{s} \in \hat{S}$ and an atomic CTL formula ϕ with $\phi \in 2^{LIT}$, $ChangeLabel(\hat{M}, \hat{s}, \phi)$ is the KMTS $\hat{M}' = (\hat{S}, \hat{S_0}, R_{must}, R_{may}, \hat{L}')$ such that $\hat{L}' = (\hat{L} - \{\hat{l}_{old}\}) \cup \{\hat{l}_{new}\}$ for $\hat{l}_{old} = (\hat{s}, lit_{old})$ and $\hat{l}_{new} = (\hat{s}, lit_{new})$ where $lit_{new} = \hat{L}(\hat{s}) \cup \{lit \mid lit \in \phi\} - \{\neg lit \mid lit \in \phi\}$. □

Basic repair operation *ChangeLabel* gives the possibility of repairing a model by changing the labeling of a state, thus without inducing any changes in the structure of the model (number of states or transitions). Fig. 9 presents the application of *ChangeLabel* in a graphical manner.

Proposition 5.18. *For any $\hat{M}' = ChangeLabel(\hat{M}, \hat{s}, \phi)$, it holds that $\hat{d}(\hat{M}, \hat{M}') = 1$.* □

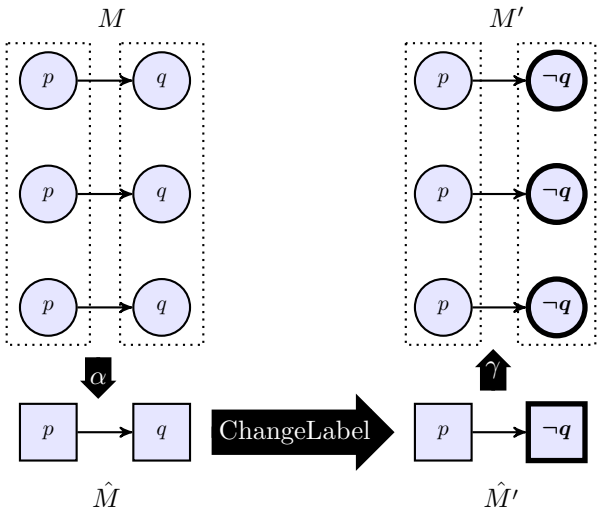


FIGURE 9. *ChangeLabel*: Changing the labeling of a KMTS state

Definition 5.19. Let $M = (S, S_0, R, L)$ be a KS and let $\alpha(M) = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$ be the abstract KMTS derived from M as in Def. 3.1. Also, let $\hat{M}' = ChangeLabel(\alpha(M), \hat{s}, \phi)$ for some $\hat{s} \in \hat{S}$ and $\phi \in 2^{LIT}$. The KS $M' \in \gamma(\hat{M}')$, whose structural distance d from M is minimized, is given by:

$$M' = (S, S_0, R, L - L_{old} \cup L_{new}\} \tag{5.5}$$

where

$$L_{old} = \{l_{old} = (s, lit_{old}) \mid s \in \gamma(\hat{s}), s \in S, \neg lit_{old} \notin \phi \text{ and } l_{old} \in L\}$$
$$L_{new} = \{l_{new} = (s, lit_{new}) \mid s \in \gamma(\hat{s}), s \in S, lit_{new} \in \phi \text{ and } l_{new} \notin L\}$$

□

Proposition 5.20. *For M', it holds that $1 \leq d(M, M') \leq |S|$.*

Proof. $d(M, M') = |S \Delta S'| + |R \Delta R'| + \frac{|\pi(L\restriction_{S \cap S'}) \Delta \pi(L'\restriction_{S \cap S'})|}{2}$. Because $|R \Delta R'| = 0$ and $|R \Delta R'| = 0$, $d(M, M') = \frac{|\pi(L\restriction_{S \cap S'}) \Delta \pi(L'\restriction_{S \cap S'})|}{2} = \frac{|L_{old}| + |L_{new}|}{2} = |L_{old}| = |L_{new}|$. It holds that $L_{new} \geq 1$ and $L_{new} \leq |S|$. So, we prove that $1 \leq d(M, M') \leq |S|$. □

5.2.6. *Adding a new KMTS state.*

Definition 5.21 (AddState)**.** For a given KMTS $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$ and a state $\hat{s}_n \notin \hat{S}$, $AddState(\hat{M}, \hat{s}_n)$ is the KMTS $\hat{M}' = (\hat{S}', \hat{S}_0, R_{must}, R_{may}, \hat{L}')$ such that $\hat{S}' = \hat{S} \cup \{\hat{s}_n\}$ and $\hat{L}' = \hat{L} \cup \{\hat{l}_n\}$, where $\hat{l}_n = (\hat{s}_n, \perp)$. □

The most important issues for function *AddState* is that the newly created abstract state $\hat{s}_n$ is isolated, thus there are no ingoing or outgoing transitions for this state, and additionally, the labeling of this new state is $\perp$. Another conclusion from Def. 5.21 is the fact that the inserted stated is not permitted to be initial. Application of function *AddState* is presented graphically in Fig. 10.

Proposition 5.22. *For any $\hat{M}' = AddState(\hat{M}, \hat{s}_n)$, it holds that $\hat{d}(\hat{M}, \hat{M}') = 1$.* □

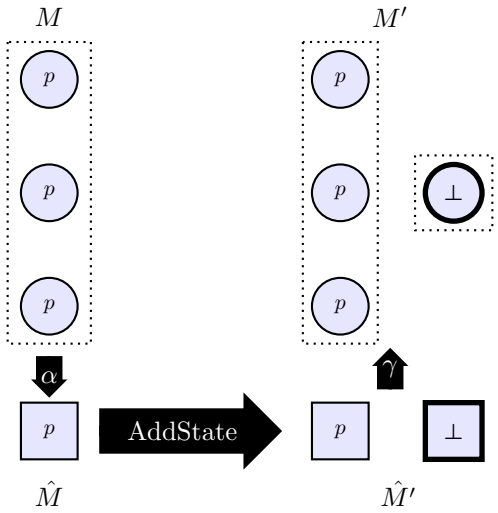


FIGURE 10. *AddState*: Adding a new KMTS state

Definition 5.23. Let $M = (S, S_0, R, L)$ be a KS and let $\alpha(M) = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$ be the abstract KMTS derived from M as in Def. 3.1. Also, let $\hat{M}' = AddState(\alpha(M), \hat{s}_n)$ for some $\hat{s}_n \notin \hat{S}$. The KS $M' \in \gamma(\hat{M}')$, whose structural distance d from M is minimized is given by:

$$M' = (S \cup \{s_n\}, S_0, R, L \cup \{l_n\}) \tag{5.6}$$

where $s_n \in \gamma(\hat{s}_n)$ and $l_n = (s_n, \bot)$. □

Proposition 5.24. *For M', it holds that $d(M, M') = 1$.*

Proof. $d(M, M') = |S \Delta S'| + |R \Delta R'| + \frac{|\pi(L\restriction_{S\cap S'}) \Delta \pi(L'\restriction_{S\cap S'})|}{2}$. Because $|R \Delta R'| = 0$ and $|\pi(L \restriction_{S\cap S'}) \Delta \pi(L' \restriction_{S\cap S'})| = 0$, $d(M, M') = |S \Delta S'| = |S - S'| + |S' - S| = 0 + |\{s_n\}| = 1$. So, we proved that $d(M, M') = 1$. □

5.2.7. *Removing a disconnected KMTS state.*

Definition 5.25 (RemoveState)**.** For a given KMTS $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$ and a state $\hat{s}_r \in \hat{S}$ such that $\forall \hat{s} \in \hat{S} : (\hat{s}, \hat{s}_r) \notin R_{may} \wedge (\hat{s}_r, \hat{s}) \notin R_{may}$, $RemoveState(\hat{M}, \hat{s}_r)$ is the KMTS $\hat{M}' = (\hat{S}', \hat{S}'_0, R_{must}, R_{may}, \hat{L}')$ such that $\hat{S}' = \hat{S} - \{\hat{s}_r\}$, $\hat{S}'_0 = \hat{S}_0 - \{\hat{s}_r\}$ and $\hat{L}' = \hat{L} - \{\hat{l}_r\}$, where $\hat{l}_r = (\hat{s}_r, lit) \in \hat{L}$. □

From Def. 5.25, it is clear that the state being removed should be isolated, thus there are not any may- or must-transitions from and to this state. This means that before using *RemoveState* to an abstract state, all its ingoing or outgoing must have been removed by using other basic repair operations. *RemoveState* are also used for the elimination of dead-end states, when such states arise during the repair process. Fig. 11 presents the application of *RemoveState* in a graphical manner.

Proposition 5.26. *For any $\hat{M}' = RemoveState(\hat{M}, \hat{s}_r)$, it holds that $\hat{d}(\hat{M}, \hat{M}') = 1$.* □

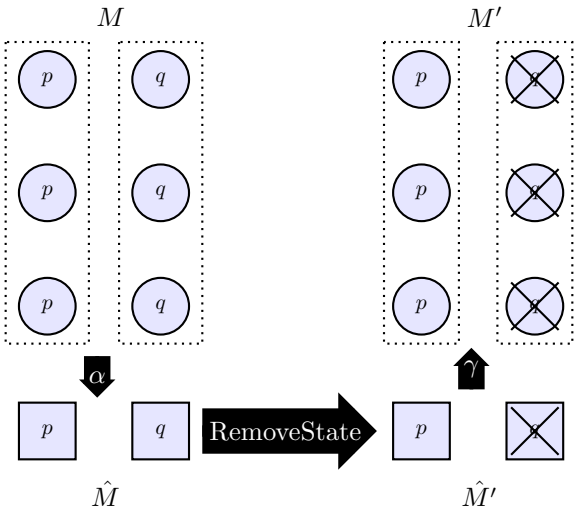


FIGURE 11. *RemoveState*: Removing a disconnected KMTS state

Definition 5.27. Let $M = (S, S_0, R, L)$ be a KS and let $\alpha(M) = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$ be the abstract KMTS derived from M as in Def. 3.1. Also, let $\hat{M}' = RemoveState(\alpha(M), \hat{s}_r)$ for some $\hat{s}_r \in \hat{S}$ with $\hat{l}_r = (\hat{s}_r, lit) \in \hat{L}$. The KS $M' \in \gamma(\hat{M}')$, whose structural distance d from M is minimized, is given by:

$$M' = (S', S'_0, R', L') \text{ s.t. } S' = S - S_r, S'_0 = S_0 - S_r, R' = R, L' = L - L_r \tag{5.7}$$

where $S_r = \{s_r \mid s_r \in S \text{ and } s_r \in \gamma(\hat{s}_r)\}$ and $L_r = \{l_r = (s_r, lit) \mid l_r \in L\}$. □

Proposition 5.28. *For M', it holds that $1 \leq d(M, M') \leq |S|$.*

Proof. $d(M, M') = |S \Delta S'| + |R \Delta R'| + \frac{|\pi(L \restriction_{S \cap S'}) \Delta \pi(L' \restriction_{S \cap S'})|}{2}$. Because $|R \Delta R'| = 0$ and $|\pi(L \restriction_{S \cap S'}) \Delta \pi(L' \restriction_{S \cap S'})| = 0$, $d(M, M') = |S \Delta S'| = |S - S'| + |S' - S| = |S_r| + 0 = |S_r|$. It holds that $|S_r| \geq 1$ and $|S_r| \leq |S|$. So, we proved that $1 \leq d(M, M') \leq |S|$. □

5.2.8. *Minimality Of Changes Ordering For Basic Repair Operations.* The distance metric d of Def. 4.1 reflects the need to quantify structural changes in the concrete model that are attributed to model repair steps applied to the abstract KMTS. Every such repair step implies multiple structural changes in the concrete KSs, due to the use of abstraction. In this context, our distance metric is an essential means for the effective application of the abstraction in the repair process.

Based on the upper bound given by Prop. 5.4 and all the respective results for the other basic repair operations, we introduce the partial ordering shown in Fig. 12. This ordering is used in our *AbstractRepair* algorithm to *heuristically* select at each step the basic repair operation that *generates the KSs with the least changes*. When it is possible to apply more than one basic repair operation with the same upper bound, our algorithm successively uses them until a repair solution is found, in an order based on the computational complexity of their application.

If instead of our approach, all possible repaired KSs were checked to identify the basic repair operation with the minimum changes, this would defeat the purpose of using abstraction. The reason is that such a check inevitably would depend on the size of concrete KSs.

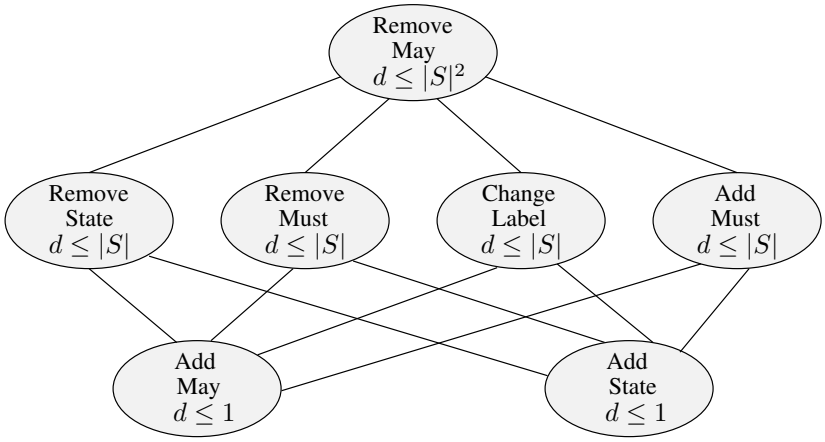


FIGURE 12. Minimality of changes ordering of the set of basic repair operations

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 ϕ in PNF for which $(\hat{M}, \hat{s}) \not\models \phi$, and a set of constraints $C = \{(\hat{s}_{c_1}, \phi_{c_1}), (\hat{s}_{c_2}, \phi_{c_2}), ..., (\hat{s}_{c_n}, \phi_{c_n})\}$ where $\hat{s}_{c_i} \in \hat{S}$ and ϕ_{c_i} 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** ϕ is $false$ **then**
2: **return** FAILURE
3: **else if** $\phi \in LIT$ **then**
4: **return** $AbstractRepair_{ATOMIC}(\hat{M}, \hat{s}, \phi, C)$
5: **else if** ϕ is $\phi_1 \wedge \phi_2$ **then**
6: **return** $AbstractRepair_{AND}(\hat{M}, \hat{s}, \phi, C)$
7: **else if** ϕ is $\phi_1 \vee \phi_2$ **then**
8: **return** $AbstractRepair_{OR}(\hat{M}, \hat{s}, \phi, C)$
9: **else if** ϕ is $OPER\phi_1$ **then**
10: **return** $AbstractRepair_{OPER}(\hat{M}, \hat{s}, \phi, C)$
11: where $OPER \in \{AX, EX, AU, EU, AF, EF, AG, EG\}$

6. The Abstract Model Repair Algorithm

The *AbstractRepair* algorithm used in Step 3 of our repair process is a recursive, syntax-directed algorithm, where the syntax for the property ϕ in question is that of CTL. The same approach is followed by the SAT model checking algorithm in [39] and a number of model repair solutions applied to concrete KSs [55, 14]. In our case, we aim to the repair of an abstract KMTS by successively calling primitive repair functions that handle atomic formulas, logical connectives and CTL operators. At each step, the repair with the least changes for the concrete model among all the possible repairs is applied first.

The main routine of *AbstractRepair* is presented in Algorithm 1. If the property ϕ is not in Positive Normal Form, i.e. negations are applied only to atomic propositions, then we transform it into such a form before applying Algorithm 1.

An initially empty set of constraints $C = \{(\hat{s}_{c_1}, \phi_{c_1}), (\hat{s}_{c_2}, \phi_{c_2}), ..., (\hat{s}_{c_n}, \phi_{c_n})\}$ is passed as an argument in the successive recursive calls of *AbstractRepair*. We note that these constraints can also specify *existing* properties that should be preserved during repair. If C is not empty, then for the returned KMTS $\hat{M}'$, it holds that $(\hat{M}', \hat{s}_{c_i}) \models \phi_{c_i}$ for all $(\hat{s}_{c_i}, \phi_{c_i}) \in C$. For brevity, we denote this with $\hat{M}' \models C$. We use C in order to handle 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.

Algorithm 2 *AbstractRepair*$_{ATOMIC}$

Input: $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$, $\hat{s} \in \hat{S}$, a CTL property ϕ where ϕ is an atomic formula for which $(\hat{M}, \hat{s}) \not\models \phi$, and a set of constraints $C = \{(\hat{s}_{c_1}, \phi_{c_1}), (\hat{s}_{c_2}, \phi_{c_2}), ..., (\hat{s}_{c_n}, \phi_{c_n})\}$ where $\hat{s}_{c_i} \in \hat{S}$ and ϕ_{c_i} 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: $\hat{M}' := ChangeLabel(\hat{M}, \hat{s}, \phi)$
2: **if** $\hat{M}' \models C$ **then**
3: **return** $\hat{M}'$
4: **else**
5: **return** FAILURE

Algorithm 3 *AbstractRepair*$_{OR}$

Input: $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$, $\hat{s} \in \hat{S}$, a CTL property $\phi = \phi_1 \vee \phi_2$ for which $(\hat{M}, \hat{s}) \not\models \phi$, and a set of constraints $C = ((\hat{s}_{c_1}, \phi_{c_1}), (\hat{s}_{c_2}, \phi_{c_2}), ..., (\hat{s}_{c_n}, \phi_{c_n}))$ where $\hat{s}_{c_i} \in \hat{S}$ and ϕ_{c_i} is a CTL formula.

Output: $\hat{M}' = (\hat{S}', \hat{S}'_0, R'_{must}, R'_{may}, \hat{L}')$, $\hat{s} \in \hat{S}'$ and $(\hat{M}', \hat{s}) \models \phi$ or FAILURE.

1: $RET_1 := AbstractRepair(\hat{M}, \hat{s}, \phi_1, C)$
2: $RET_2 := AbstractRepair(\hat{M}, \hat{s}, \phi_2, C)$
3: **if** $RET_1 \neq FAILURE$ && $RET_2 \neq FAILURE$ **then**
4: $\hat{M}_1 := RET_1$
5: $\hat{M}_2 := RET_2$
6: $\hat{M}' := MinimallyChanged(\hat{M}, \hat{M}_1, \hat{M}_2)$
7: **else if** $RET_1 \neq FAILURE$ **then**
8: $\hat{M}' := RET_1$
9: **else if** $RET_2 \neq FAILURE$ **then**
10: $\hat{M}' := RET_2$
11: **else**
12: **return** FAILURE
13: **return** $\hat{M}'$

Algorithm 4 *AbstractRepair*$_{AND}$

Input: $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$, $\hat{s} \in \hat{S}$, a CTL property $\phi = \phi_1 \wedge \phi_2$ for which $(\hat{M}, \hat{s}) \not\models \phi$, and a set of constraints $C = ((\hat{s}_{c_1}, \phi_{c_1}), (\hat{s}_{c_2}, \phi_{c_2}), ..., (\hat{s}_{c_n}, \phi_{c_n}))$ where $\hat{s}_{c_i} \in \hat{S}$ and ϕ_{c_i} is a CTL formula.

Output: $\hat{M}' = (\hat{S}', \hat{S}'_0, R'_{must}, R'_{may}, \hat{L}')$, $\hat{s} \in \hat{S}'$ and $(\hat{M}', \hat{s}) \models \phi$ or FAILURE.

$RET_1 := AbstractRepair(\hat{M}, \hat{s}, \phi_1, C)$
$RET_2 := AbstractRepair(\hat{M}, \hat{s}, \phi_2, C)$
$C_1 := C \cup \{(\hat{s}, \phi_1)\}$, $C_2 := C \cup \{(\hat{s}, \phi_2)\}$
$RET'_1 := FAIURE$, $RET'_2 := FAIURE$
if $RET_1 \neq FAILURE$ **then**
  $\hat{M}_1 := RET_1$
  $RET'_1 := AbstractRepair(\hat{M}_1, \hat{s}, \phi_2, C_1)$
  if $RET'_1 \neq FAILURE$ **then**
    $\hat{M}'_1 := RET'_1$
if $RET_2 \neq FAILURE$ **then**
  $\hat{M}_2 := RET_2$
  $RET'_2 := AbstractRepair(\hat{M}_2, \hat{s}, \phi_1, C_2)$
  if $RET'_2 \neq FAILURE$ **then**
    $\hat{M}'_2 := RET'_2$
if $RET'_1 \neq FAILURE$ && $RET'_2 \neq FAILURE$ **then**
  $\hat{M}' := MinimallyChanged(\hat{M}, \hat{M}'_1, \hat{M}'_2)$
else if $RET'_1 \neq FAILURE$ **then**
  $\hat{M}' := RET'_1$
else if $RET'_2 \neq FAILURE$ **then**
  $\hat{M}' := RET'_2$
else
  return FAILURE
return $\hat{M}'$

6.1. **Primitive Functions.** Algorithm 2 describes *AbstractRepair*$_{ATOMIC}$, which for a simple atomic formula, updates the labeling of the input state with the given atomic proposition. Disjunctive formulas are handled by repairing the disjunct leading to the minimum change (Algorithm 3), while conjunctive formulas are handled by the algorithm with the use of constraints (Algorithm 4).

Algorithm 5 describes the primitive function *AbstractRepair*$_{AG}$ which is called when $\phi = AG\phi_1$. If *AbstractRepair*$_{AG}$ is called for a state $\hat{s}$, it recursively calls *AbstractRepair* for $\hat{s}$ and for all reachable states through may-transitions from $\hat{s}$ which do not satisfy ϕ_1. The resulting KMTS $\hat{M}'$ is returned, if it does not violate any constraint in C.

AbstractRepair$_{EX}$ presented in Algorithm 6 is the primitive function for handling properties of the form $EX\phi_1$ for some state $\hat{s}$. At first, *AbstractRepair*$_{EX}$ attempts to repair the KMTS by adding a must-transition from $\hat{s}$ to a state that satisfies property ϕ_1. If a repaired KMTS is not found, 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 violated, then (i) a new state is added, (ii) *AbstractRepair* is called for the new state and

Algorithm 5 *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 $C = \{(\hat{s}_{c_1}, \phi_{c_1}), (\hat{s}_{c_2}, \phi_{c_2}), ..., (\hat{s}_{c_n}, \phi_{c_n})\}$ where $\hat{s}_{c_i} \in \hat{S}$ and ϕ_{c_i} 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.

if $(\hat{M}, \hat{s}) \not\models \phi_1$ **then**
  $RET := AbstractRepair(\hat{M}, \hat{s}, \phi_1, C)$
  if $RET == FAILURE$ **then**
    return FAILURE
  else
    $\hat{M}' := RET$
else
  $\hat{M}' := \hat{M}$
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**
  $RET := AbstractRepair(\hat{M}', \hat{s}_k, \phi_1, C)$
  if $RET == FAILURE$ **then**
    return FAILURE
  else
    $\hat{M}' := RET$
if $\hat{M}' \models C$ **then**
  return $\hat{M}'$
return FAILURE

(iii) a must-transition from $\hat{s}$ to the new state is added. The resulting KMTS is returned by the algorithm if all constraints of C are satisfied.

Algorithm 7 presents primitive function *AbstractRepair*$_{AX}$ which is used when $\phi = AX\phi_1$. Firstly, *AbstractRepair*$_{AX}$ tries to repair the KMTS by applying *AbstractRepair* for all direct may-successors $\hat{s}_i$ of $\hat{s}$ which do not satisfy property ϕ_1, and in the case that all the constraints are satisfied the new KMTS is returned by the function. If such states do not exist or a constraint is violated, all may-transitions $(\hat{s}, \hat{s}_i)$ for which $(\hat{M}, \hat{s}_i) \not\models \phi_1$, are removed. If there are states $\hat{s}_i$ such that $r_m := (\hat{s}, \hat{s}_i) \in R_{may}$ and all constraints are satisfied then a repaired KMTS has been produced and it is returned by the function. Otherwise, a repaired KMTS results by the application of *AddMay* from $\hat{s}$ to all states $\hat{s}_j$ which satisfy ϕ_1. If any constraint is violated, then the KMTS is repaired by adding a new state, applying *AbstractRepair* to this state for property ϕ_1 and adding a may-transition from $\hat{s}$ to this state. If all constraints are satisfied, the repaired KMTS is returned.

AbstractRepair$_{EG}$ which is presented in Algorithm 8 is the primitive function which is called when input CTL property is in the form of $EG\phi_1$. Initially, if ϕ_1 is not satisfied at $\hat{s}$ *AbstractRepair* is called for $\hat{s}$ and ϕ_1, and a KMTS $\hat{M}_1$ is produced. At first, a must-transition is added from $\hat{s}$ to a state $\hat{s}_1$ of a maximal must-path (i.e. a must-path in which each transition appears at most once) $\pi_{must} := [\hat{s}_1, \hat{s}_2, ...]$ such that $\forall \hat{s}_i \in \pi_{must}$, $(\hat{M}_1, \hat{s}_i) \models \phi_1$. If all constraints are satisfied, then the repaired KMTS is returned. Otherwise, a KMTS is produced by recursively calling *AbstractRepair* to all states $\hat{s}_i \neq \hat{s}$ of any maximal must-path $\pi_{must} := [\hat{s}_1, \hat{s}_2, ...]$ with $\forall \hat{s}_i \in \pi_{must}$, $(\hat{M}_1, \hat{s}_i) \not\models \phi_1$. If there are violated constraints in C, then a repaired KMTS is produced by adding a new state, calling *AbstractRepair* for this

Algorithm 6 *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 $C = \{(\hat{s}_{c_1}, \phi_{c_1}), (\hat{s}_{c_2}, \phi_{c_2}), ..., (\hat{s}_{c_n}, \phi_{c_n})\}$ where $\hat{s}_{c_i} \in \hat{M}$ and ϕ_{c_i} 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.

if there exists $\hat{s}_1 \in \hat{S}$ such that $(\hat{M}, \hat{s}_1) \models \phi_1$ **then**
  for all $\hat{s}_i \in \hat{S}$ such that $(\hat{M}, \hat{s}_i) \models \phi_1$ **do**
    $\hat{r}_i := (\hat{s}, \hat{s}_i)$, $\hat{M}' := AddMust(\hat{M}, \hat{r}_i)$
    if $\hat{M}' \models C$ **then**
      return $\hat{M}'$
else
  for all direct must-reachable states $\hat{s}_i$ from $\hat{s}$ such that $(\hat{M}, \hat{s}_i) \not\models \phi_1$ **do**
    $RET := AbstractRepair(\hat{M}, \hat{s}_i, \phi_1, C)$
    if $RET \neq FAILURE$ **then**
      $\hat{M}' := RET$
      return $\hat{M}'$
  $\hat{M}' := AddState(\hat{M}, \hat{s}_n)$, $\hat{r}_n := (\hat{s}, \hat{s}_n)$, $\hat{M}' := AddMust(\hat{M}', \hat{r}_n)$
  $\hat{r}_n := (\hat{s}_n, \hat{s}_n)$
  $\hat{M}' := AddMay(\hat{M}', \hat{r}_n)$
  $RET := AbstractRepair(\hat{M}', \hat{s}_n, \phi_1, C)$
  if $RET \neq FAILURE$ **then**
    $\hat{M}' := RET$
    return $\hat{M}'$
return FAILURE

state and property ϕ_1 and calling *AddMust* to insert a must-transition from $\hat{s}$ to the new state. The resulting KMTS is returned by the algorithm, if all constraints in C are satisfied.

AbstractRepair$_{AF}$ shown in Algorithm 9 is called when the CTL formula ϕ is in the form of $AF\phi_1$. While there is maximal may-path $\pi_{may} := [\hat{s}, \hat{s}_1, ...]$ such that $\forall \hat{s}_i \in \pi_{may}$, $(\hat{M}', \hat{s}_i) \not\models \phi_1$, *AbstractRepair*$_{AF}$ tries to obtain a repaired KMTS by recursively calling *AbstractRepair* to some state $\hat{s}_i \in \pi_{may}$. If all constraints are satisfied to the new KMTS, then it is returned as the repaired model.

AbstractRepair$_{EF}$ shown in Algorithm 10 is called when the CTL property ϕ is in the form $EF\phi_1$. Initially, a KMTS is acquired by adding a must-transition from a must-reachable state $\hat{s}_i$ from $\hat{s}$ to a state $\hat{s}_k \in \hat{S}$ such that $(\hat{M}, \hat{s}_k) \models \phi_1$. If all constraints are satisfied then this KMTS is returned. Otherwise, a KMTS is produced by applying *AbstractRepair* to a must-reachable state $\hat{s}_i$ from $\hat{s}$ for ϕ_1. If none of the constraints is violated then this KMTS is returned. At any other case, a new KMTS is produced by adding a new state $\hat{s}_n$, recursively calling *AbstractRepair* for this state and ϕ_1 and adding a must-transition from $\hat{s}$ or from a must-reachable $\hat{s}_i$ from $\hat{s}$ to $\hat{s}_n$. If all constraints are satisfied, then this KMTS is returned as a repaired model by the algorithm.

AbstractRepair$_{AU}$ is presented in Algorithm 11 and is called when $\phi = A(\phi_1 U \phi_2)$. If ϕ_1 is not satisfied at $\hat{s}$, then a KMTS $\hat{M}_1$ is produced by applying *AbstractRepair* to $\hat{s}$ for ϕ_1. Otherwise, $\hat{M}_1$ is same to $\hat{M}$. A new KMTS is produced as follows: for all may-paths $\pi_{may} := [\hat{s}_1, ..., \hat{s}_m]$ such that $\forall \hat{s}_i \in \pi_{may}$, $(\hat{M}_1, \hat{s}_i) \models \phi_1$ and for which there does not

Algorithm 7 *AbstractRepair*$_{AX}$

Input: $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$, $\hat{s} \in \hat{S}$, a CTL property $\phi = AX\phi_1$ for which $(\hat{M}, \hat{s}) \not\models \phi$, and a set of constraints $C = \{(\hat{s}_{c_1}, \phi_{c_1}), (\hat{s}_{c_2}, \phi_{c_2}), ..., (\hat{s}_{c_n}, \phi_{c_n})\}$ where $\hat{s}_{c_i} \in \hat{M}$ and ϕ_{c_i} 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.

$\hat{M}' := \hat{M}$
$RET := FAILURE$
for all direct may-reachable states $\hat{s}_i$ from $\hat{s}$ with $(\hat{s}, \hat{s}_i) \in R_{may}$ **do**
  if $(\hat{M}', \hat{s}_i) \not\models \phi_1$ **then**
    $RET := AbstractRepair(\hat{M}', \hat{s}_i, \phi_1, C)$
    if $RET == FAILURE$ **then**
      BREAK
    $\hat{M}' := RET$
if $RET \neq FAILURE$ **then**
  return $\hat{M}'$
$\hat{M}' := \hat{M}$
for all direct may-reachable states $\hat{s}_i$ from $\hat{s}$ with $\hat{r}_i := (\hat{s}, \hat{s}_i) \in R_{may}$ **do**
  if $(\hat{M}', \hat{s}_i) \not\models \phi_1$ **then**
    $\hat{M}' := RemoveMay(\hat{M}', \hat{r}_i)$
if there exists direct may-reachable state $\hat{s}_1$ from $\hat{s}$ such that $(\hat{s}, \hat{s}_1) \in R_{may}$ **then**
  if $\hat{M}' \models C$ **then**
    return $\hat{M}'$
else
  for all $\hat{s}_j \in \hat{S}$ such that $(\hat{M}', \hat{s}_j) \models \phi_1$ **do**
    $\hat{r}_j := (\hat{s}, \hat{s}_j)$, $\hat{M}' := AddMay(\hat{M}', \hat{r}_j)$
    if $\hat{M}' \models C$ **then**
      return $\hat{M}'$
  $\hat{M}' := AddState(\hat{M}, \hat{s}_n)$
  if $\hat{s}_n$ is a dead-end state **then**
    $\hat{r}_n := (\hat{s}_n, \hat{s}_n)$, $\hat{M}' := AddMay(\hat{M}', \hat{r}_n)$
  $RET := AbstractRepair(\hat{M}', \hat{s}_n, \phi_1, C)$
  if $RET \neq FAILURE$ **then**
    $\hat{M}' := RET$, $\hat{r}_n := (\hat{s}, \hat{s}_n)$, $\hat{M}' := AddMay(\hat{M}', \hat{r}_n)$
    if $\hat{M}' \models C$ **then**
      return $\hat{M}'$
return FAILURE

$\hat{r}_m := (\hat{s}_m, \hat{s}_n) \in R_{may}$ with $(\hat{M}_1, \hat{s}_n) \models \phi_2$, *AbstractRepair* is called for property ϕ_2 for some state $\hat{s}_j \in \pi_{may}$ with $(\hat{M}_1, \hat{s}_j) \not\models \phi_2$. If the resulting KMTS satisfies all constraints, then it is returned as a repair solution.

AbstractRepair$_{EU}$ is called if for input CTL formula ϕ it holds that $\phi = E(\phi_1 U \phi_2)$. *AbstractRepair*$_{EU}$ is presented in Algorithm 12. Firstly, if ϕ_1 is not satisfied at $\hat{s}$, then *AbstractRepair* is called for $\hat{s}$ and ϕ_1 and a KMTS $\hat{M}_1$ is produced for which $(\hat{M}_1, \hat{s}) \models \phi_1$. Otherwise, $\hat{M}_1$ is same to $\hat{M}$. A new KMTS is produced as follows: for a must-path

Algorithm 8 *AbstractRepair*$_{EG}$

Input: $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$, $\hat{s} \in \hat{S}$, a CTL property $\phi = EG\phi_1$ for which $(\hat{M}, \hat{s}) \not\models \phi$, and a set of constraints $C = \{(\hat{s}_{c_1}, \phi_{c_1}), (\hat{s}_{c_2}, \phi_{c_2}), ..., (\hat{s}_{c_n}, \phi_{c_n})\}$ where $\hat{s}_{c_i} \in \hat{S}$ and ϕ_{c_i} 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.

$\hat{M}_1 := \hat{M}$
if $(\hat{M}, \hat{s}) \not\models \phi_1$ **then**
  $RET := AbstractRepair(\hat{M}, \hat{s}, \phi_1, C)$
  if $RET == FAILURE$ **then**
    return FAILURE
  $\hat{M}_1 := RET$
while there exists maximal path $\pi_{must} := [\hat{s}_1, \hat{s}_2, ...]$ such that $\forall \hat{s}_i \in \pi_{must}$ it holds that $(\hat{M}_1, \hat{s}_i) \models \phi_1$ **do**
  $\hat{r}_1 := (\hat{s}, \hat{s}_1)$, $\hat{M}' := AddMust(\hat{M}_1, \hat{r}_1)$
  if $\hat{M}' \models C$ **then**
    return $\hat{M}'$
while there exists maximal path $\pi_{must} := [\hat{s}, \hat{s}_1, \hat{s}_2, ...]$ such that $\forall \hat{s}_i \neq \hat{s} \in \pi_{must}$ it holds that $(\hat{M}_1, \hat{s}_i) \not\models \phi_1$ **do**
  $\hat{M}' := \hat{M}_1$
  for all $\hat{s}_i \in \pi_{must}$ **do**
    if $(\hat{M}_1, \hat{s}_i) \not\models \phi_1$ **then**
      $RET := AbstractRepair(\hat{M}', \hat{s}_i, \phi_1, C)$
      if $RET \neq FAILURE$ **then**
        $\hat{M}' := RET$
      else
        continue to next path
  return $\hat{M}'$
$\hat{M}' := AddState(\hat{M}_1, \hat{s}_n)$
$RET := AbstractRepair(\hat{M}', \hat{s}_n, \phi_1, C)$
if $RET \neq FAILURE$ **then**
  $\hat{M}' := RET$
  $\hat{r}_n := (\hat{s}, \hat{s}_n)$, $\hat{M}' := AddMust(\hat{M}', \hat{r}_n)$
  if $\hat{s}_n$ is a dead-end state **then**
    $\hat{r}_n := (\hat{s}_n, \hat{s}_n)$, $\hat{M}' := AddMust(\hat{M}', \hat{r}_n)$
  if $\hat{M}' \models C$ **then**
    return $\hat{M}'$
return FAILURE

$\pi_{must} := [\hat{s}_1, ..., \hat{s}_m]$ such that $\forall \hat{s}_i \in \pi_{must}$, $(\hat{M}_1, \hat{s}_i) \models \phi_1$ and for a $\hat{s}_j \in \hat{S}$ with $(\hat{M}_1, \hat{s}_j) \models \phi_2$, a must-transition is added from $\hat{s}_m$ to $\hat{s}_j$. If all constraints are satisfied then the new KMTS is returned. Alternatively, a KMTS is produced by adding a new state $\hat{s}_n$, recursively calling *AbstractRepair* for ϕ_2 and $\hat{s}_n$ and adding a must-transition from $\hat{s}$ to $\hat{s}_n$. In the case that no constraint is violated then this is a repaired KMTS and it is returned from the function.

Algorithm 9 *AbstractRepair*$_{AF}$

Input: $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$, $\hat{s} \in \hat{S}$, a CTL property $\phi = AF\phi_1$ for which $(\hat{M}, \hat{s}) \not\models \phi$, and a set of constraints $C = \{(\hat{s}_{c_1}, \phi_{c_1}), (\hat{s}_{c_2}, \phi_{c_2}), ..., (\hat{s}_{c_n}, \phi_{c_n})\}$ where $\hat{s}_{c_i} \in \hat{S}$ and ϕ_{c_i} 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: $\hat{M}' := \hat{M}$
2: **while** there exists maximal path $\pi_{may} := [\hat{s}, \hat{s}_1, ...]$ such that $\forall \hat{s}_i \in \pi_{may}$ it holds that $(\hat{M}', \hat{s}_i) \not\models \phi_1$ **do**
3: **for all** $\hat{s}_i \in \pi_{may}$ **do**
4: $RET := AbstractRepair(\hat{M}', \hat{s}_i, \phi_1, C)$
5: **if** $RET \neq FAILURE$ **then**
6: $\hat{M}' := RET$
7: continue to next path
8: **return** FAILURE
9: **return** $\hat{M}'$

6.2. **Properties of the Algorithm.** *AbstractRepair* is *well-defined* [49], in the sense that the algorithm always proceeds and eventually returns a result $\hat{M}'$ or FAILURE such that $(\hat{M}', \hat{s}) \models \phi$, for any input $\hat{M}$, ϕ and C, with $(\hat{M}, \hat{s}) \not\models \phi$. Moreover, the algorithm steps are well-ordered, as opposed to existing concrete model repair solutions [13, 55] that entail nondeterministic behavior.

6.2.1. *Soundness.*

Lemma 6.1. *Let a KMTS $\hat{M}$, a CTL formula ϕ with $(\hat{M}, \hat{s}) \not\models \phi$ for some $\hat{s}$ of $\hat{M}$, and a set $C = \{(\hat{s}_{c_1}, \phi_{c_1}), (\hat{s}_{c_2}, \phi_{c_2}), ..., (\hat{s}_{c_n}, \phi_{c_n})\}$ with $(\hat{M}, \hat{s}_{c_i}) \models \phi_{c_i}$ for all $(\hat{s}_{c_n}, \phi_{c_n}) \in C$. If $AbstractRepair(\hat{M}, \hat{s}, \phi, C)$ returns a KMTS $\hat{M}'$, then $(\hat{M}', \hat{s}) \models \phi$ and $(\hat{M}', \hat{s}_{c_i}) \models \phi_{c_i}$ for all $(\hat{s}_{c_i}, \phi_{c_i}) \in C$.*

Proof. We use structural induction on ϕ. For brevity, we write $\hat{M} \models C$ to denote that $(\hat{M}, \hat{s}_{c_i}) \models \phi_{c_i}$, for all $(\hat{s}_{c_i}, \phi_{c_i}) \in C$.

Base Case:

- if $\phi = \top$, the lemma is trivially true, because $(\hat{M}, \hat{s}) \models \phi$
- if $\phi = \bot$, then $AbstractRepair(\hat{M}, \hat{s}, \phi, C)$ returns FAILURE at line 2 of Algorithm 1 and the lemma is also trivially true.
- if $\phi = p \in AP$, $AbstractRepair_{ATOMIC}(\hat{M}, \hat{s}, p, C)$ is called at line 4 of Algorithm 1 and an $\hat{M}' = ChangeLabel(\hat{M}, \hat{s}, p)$ is computed at line 1 of Algorithm 2. Since $p \in \hat{L}'(\hat{s})$ in $\hat{M}'$, from 3-valued semantics of CTL over KMTSs we have $(\hat{M}', \hat{s}) \models \phi$. Algorithm 2 returns $\hat{M}'$ at line 3, if and only if $\hat{M}' \models C$ and the lemma is true.

Induction Hypothesis: For CTL formulae ϕ_1, ϕ_2, the lemma is true. Thus, for ϕ_1 (resp. ϕ_2), if $AbstractRepair(\hat{M}, \hat{s}, \phi_1, C)$ returns a KMTS $\hat{M}'$, then $(\hat{M}', \hat{s}) \models \phi_1$ and $\hat{M}' \models C$.

Algorithm 10 *AbstractRepair*$_{EF}$

Input: $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$, $\hat{s} \in \hat{S}$, a CTL property $\phi = EF\phi_1$ for which $(\hat{M}, \hat{s}) \not\models \phi$, and a set of constraints $C = \{(\hat{s}_{c_1}, \phi_{c_1}), (\hat{s}_{c_2}, \phi_{c_2}), ..., (\hat{s}_{c_n}, \phi_{c_n})\}$ where $\hat{s}_{c_i} \in \hat{S}$ and ϕ_{c_i} 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: **for all** must-reachable states $\hat{s}_i$ from $\hat{s}$ with $(\hat{M}, \hat{s}_i) \not\models \phi_1$ or $\hat{s}_i := \hat{s}$ **do**
2: **for all** $\hat{s}_k \in \hat{S}$ such that $(\hat{M}, \hat{s}_k) \models \phi_1$ **do**
3: $\hat{r}_k := (\hat{s}_i, \hat{s}_k)$, $\hat{M}' := AddMust(\hat{M}, \hat{r}_k)$
4: **if** $\hat{M}' \models C$ **then**
5: **return** $\hat{M}'$
6: **for all** must-reachable states $\hat{s}_i$ from $\hat{s}$ with $(\hat{M}, \hat{s}_i) \not\models \phi_1$ **do**
7: $RET := AbstractRepair(\hat{M}, \hat{s}_i, \phi_1, C)$
8: **if** $RET \neq FAILURE$ **then**
9: $\hat{M}' := RET$
10: **return** $\hat{M}'$
11: $\hat{M}_1 := AddState(\hat{M}', \hat{s}_n)$, $RET := AbstractRepair(\hat{M}_1, \hat{s}_n, \phi_1, C)$
12: **if** $RET \neq FAILURE$ **then**
13: $\hat{M}_1 := RET$
14: **for all** must-reachable states $\hat{s}_i$ from $\hat{s}$ with $(\hat{M}, \hat{s}_i) \not\models \phi_1$ or $\hat{s}_i := \hat{s}$ **do**
15: $\hat{r}_i := (\hat{s}_i, \hat{s}_n)$, $\hat{M}' := AddMust(\hat{M}_1, \hat{r}_i)$
16: **if** $\hat{s}_n$ is a dead-end state **then**
17: $\hat{r}_n := (\hat{s}_n, \hat{s}_n)$, $\hat{M}' := AddMust(\hat{M}', \hat{r}_n)$
18: **if** $\hat{M}' \models C$ **then**
19: **return** $\hat{M}'$
20: **return** FAILURE

Inductive Step:

- if $\phi = \phi_1 \vee \phi_2$, then $AbstractRepair(\hat{M}, \hat{s}, \phi, C)$ calls $AbstractRepair_{OR}(\hat{M}, \hat{s}, \phi_1 \vee \phi_2, C)$ at line 8 of Algorithm 1. From the induction hypothesis, if a KMTS $\hat{M}_1$ is returned by $AbstractRepair(\hat{M}, \hat{s}, \phi_1, C)$ at line 1 of Algorithm 3 and a KMTS $\hat{M}_2$ is returned by $AbstractRepair(\hat{M}, \hat{s}, \phi_2, C)$ respectively, then $(\hat{M}_1, \hat{s}) \models \phi_1$, $\hat{M}_1 \models C$ and $(\hat{M}_2, \hat{s}) \models \phi_1$, $\hat{M}_2 \models C$. $AbstractRepair_{OR}(\hat{M}, \hat{s}, \phi_1 \vee \phi_2, C)$ returns at line 8 of Algorithm 1 the KMTS $\hat{M}'$, which can be either $\hat{M}_1$ or $\hat{M}_2$. Therefore, $(\hat{M}', \hat{s}) \models \phi_1$ or $(\hat{M}', \hat{s}) \models \phi_2$ and $\hat{M}' \models C$ in both cases. From 3-valued semantics of CTL, $(\hat{M}', \hat{s}) \models \phi_1 \vee \phi_2$ and the lemma is true.
- if $\phi = \phi_1 \wedge \phi_2$, then $AbstractRepair(\hat{M}, \hat{s}, \phi, C)$ calls $AbstractRepair_{AND}(\hat{M}, \hat{s}, \phi_1 \wedge \phi_2, C)$ at line 6 of Algorithm 1. From the induction hypothesis, if at line 1 of Algorithm 4 $AbstractRepair(\hat{M}, \hat{s}, \phi_1, C)$ returns a KMTS $\hat{M}_1$, then $(\hat{M}_1, \hat{s}) \models \phi_1$ and $\hat{M}_1 \models C$. Consequently, $\hat{M}_1 \models C_1$, where $C_1 = C \cup (\hat{s}, \phi_1)$. At line 7, if $AbstractRepair(\hat{M}_1, \hat{s}, \phi_2, C_1)$ returns a KMTS $\hat{M}'_1$, then from the induction hypothesis $(\hat{M}'_1, \hat{s}) \models \phi_2$ and $\hat{M}'_1 \models C_1$.

  In the same manner, if the calls at lines 2 and 12 of Algorithm 4 return the KMTSs $\hat{M}_2$ and $\hat{M}'_2$, then from the induction hypothesis $(\hat{M}_2, \hat{s}) \models \phi_2$, $\hat{M}_2 \models C$ and $(\hat{M}'_2, \hat{s}) \models \phi_1$, $\hat{M}'_2 \models C_2$ with $C_2 = C \cup (\hat{s}, \phi_2)$.

Algorithm 11 *AbstractRepair*$_{AU}$

Input: $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$, $\hat{s} \in \hat{S}$, a CTL property $\phi = A(\phi_1 U \phi_2)$ for which $(\hat{M}, \hat{s}) \not\models \phi$, and a set of constraints $C = \{(\hat{s}_{c_1}, \phi_{c_1}), (\hat{s}_{c_2}, \phi_{c_2}), ..., (\hat{s}_{c_n}, \phi_{c_n})\}$ where $\hat{s}_{c_i} \in \hat{S}$ and ϕ_{c_i} 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: $\hat{M}_1 := \hat{M}$
2: **if** $(\hat{M}, \hat{s}) \not\models \phi_1$ **then**
3: $RET := AbstractRepair(\hat{M}, \hat{s}, \phi_1, C)$
4: **if** $RET == FAILURE$ **then**
5: **return** FAILURE
6: **else**
7: $\hat{M}_1 := RET$
8: **while** there exists path $\pi_{may} := [\hat{s}_1, ..., \hat{s}_m]$ such that $\forall \hat{s}_i \in \pi_{may}$ it holds that $(\hat{M}_1, \hat{s}_i) \models \phi_1$ and there does not exist $\hat{r}_m := (\hat{s}_m, \hat{s}_n) \in R_{may}$ such that $(\hat{M}_1, \hat{s}_n) \models \phi_2$ **do**
9: **for all** $\hat{s}_j \in \pi_{may}$ for which $(\hat{M}_1, \hat{s}_j) \not\models \phi_2$ with $\hat{s}_j \neq \hat{s}_1$ **do**
10: $RET := AbstractRepair(\hat{M}_1, \hat{s}_j, \phi_2, C)$
11: **if** $RET \neq FAILURE$ **then**
12: $\hat{M}' := RET$
13: continue to next path
14: **return** FAILURE
15: **return** $\hat{M}'$

The KMTS $\hat{M}'$ at line 6 of Algorithm 1 can be either $\hat{M}'_1$ or $\hat{M}'_2$ and therefore, $(\hat{M}', \hat{s}) \models \phi_1$, $(\hat{M}', \hat{s}) \models \phi_2$ and $\hat{M}' \models C$. From 3-valued semantics of CTL it holds that $(\hat{M}', \hat{s}) \models \phi_1 \wedge \phi_2$ and the lemma is true.

- if $\phi = EX\phi_1$, $AbstractRepair(\hat{M}, \hat{s}, \phi, C)$ calls $AbstractRepair_{EX}(\hat{M}, \hat{s}, EX\phi_1, C)$ at line 10 of Algorithm 1.

  If a KMTS $\hat{M}'$ is returned at line 5 of Algorithm 6, there is a state $\hat{s}_1$ with $(\hat{M}, \hat{s}_1) \models \phi_1$ such that $\hat{M}' = AddMust(\hat{M}, (\hat{s}, \hat{s}_1))$ and $\hat{M}' \models C$. From 3-valued semantics of CTL, we conclude that $(\hat{M}', \hat{s}) \models EX\phi_1$.

  If a $\hat{M}'$ is returned at line 11, there is $(\hat{s}, \hat{s}_1) \in R_{must}$ such that $(\hat{M}', \hat{s}_1) \models \phi_1$ and $\hat{M}' \models C$ from the induction hypothesis, since $\hat{M}' = AbstractRepair(\hat{M}, \hat{s}_1, \phi_1, C)$. From 3-valued semantics of CTL, we conclude that $(\hat{M}', \hat{s}) \models EX\phi_1$.

  If a $\hat{M}'$ is returned at line 18, a must transition $(\hat{s}, \hat{s}_n)$ to a new state has been added and $\hat{M}' = AbstractRepair(AddMust(\hat{M}, (\hat{s}, \hat{s}_n)), \hat{s}_n, \phi_1, C)$. Then, from the induction hypothesis $(\hat{M}', \hat{s}_n) \models \phi_1$, $\hat{M}' \models C$ and from 3-valued semantics of CTL, we also conclude that $(\hat{M}', \hat{s}) \models EX\phi_1$.
- if $\phi = AG\phi_1$, $AbstractRepair(\hat{M}, \hat{s}, \phi, C)$ calls $AbstractRepair_{AG}(\hat{M}, \hat{s}, AG\phi_1, C)$ at line 10 of Algorithm 1. If $(\hat{M}, \hat{s}) \not\models \phi_1$ and $AbstractRepair(\hat{M}, \hat{s}, \phi_1, C)$ returns a KMTS $\hat{M}_0$ at line 2 of Algorithm 5, then from the induction hypothesis $(\hat{M}_0, \hat{s}) \models \phi_1$ and $\hat{M}_0 \models C$. Otherwise, $\hat{M}_0 = \hat{M}$ and $(\hat{M}_0, \hat{s}) \models \phi_1$ also hold true.

  If Algorithm 5 returns a $\hat{M}'$ at line 16, then $\hat{M}' \models C$ and $\hat{M}'$ is the result of successive $AbstractRepair(\hat{M}_i, \hat{s}_k, \phi_1, C)$ calls with $\hat{M}_i = AbstractRepair(\hat{M}_{i-1}, \hat{s}_k, \phi_1, C)$ and $i =$

Algorithm 12 *AbstractRepair*$_{EU}$

Input: $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$, $\hat{s} \in \hat{S}$, a CTL property $\phi = E(\phi_1 U \phi_2)$ for which $(\hat{M}, \hat{s}) \not\models \phi$, and a set of constraints $C = \{(\hat{s}_{c_1}, \phi_{c_1}), (\hat{s}_{c_2}, \phi_{c_2}), ..., (\hat{s}_{c_n}, \phi_{c_n})\}$ where $\hat{s}_{c_i} \in \hat{S}$ and ϕ_{c_i} 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: $\hat{M}_1 := \hat{M}$
2: **if** $(\hat{M}, \hat{s}) \not\models \phi_1$ **then**
3: $RET := AbstractRepair(\hat{M}, \hat{s}, \phi_1, C)$
4: **if** $RET == FAILURE$ **then**
5: **return** FAILURE
6: **else**
7: $\hat{M}_1 := RET$
8: **while** there exists path $\pi_{must} := [\hat{s}_1, ..., \hat{s}_m]$ such that $\forall \hat{s}_i \in \pi_{must}$, $(\hat{M}_1, \hat{s}_i) \models \phi_1$ **do**
9: **for all** $\hat{s}_j \in \hat{S}$ with $(\hat{M}_1, \hat{s}_j) \models \phi_2$ **do**
10: $\hat{r}_j := (\hat{s}_m, \hat{s}_j)$, $\hat{M}' := AddMust(\hat{M}_1, \hat{r}_j)$
11: **if** $\hat{M}' \models C$ **then**
12: **return** $\hat{M}'$
13: $\hat{M}' := AddState(\hat{M}_1, \hat{s}_k)$
14: $RET := AbstractRepair(\hat{M}', \hat{s}_k, \phi_2, C)$
15: **if** $RET \neq FAILURE$ **then**
16: $\hat{M}' := RET$
17: $\hat{r}_n := (\hat{s}, \hat{s}_k)$, $\hat{M}' := AddMust(\hat{M}', \hat{r}_n)$
18: **if** $\hat{s}_k$ is a dead-end state **then**
19: $\hat{r}_k := (\hat{s}_k, \hat{s}_k)$, $\hat{M}' := AddMust(\hat{M}', \hat{r}_k)$
20: **if** $\hat{M}' \models C$ **then**
21: **return** $\hat{M}'$
22: **return** FAILURE

$1, ...$, for all may-reachable states $\hat{s}_k$ from $\hat{s}$ such that $(\hat{M}_0, \hat{s}_k) \not\models \phi_1$. From the induction hypothesis, $(\hat{M}', \hat{s}_k) \models \phi_1$ and $\hat{M}' \models C$ for all such $\hat{s}_k$ and from 3-valued semantics of CTL we conclude that $(\hat{M}', \hat{s}) \models AG\phi_1$.

We prove the lemma for all other cases in a similar manner. □

Theorem 6.2 (Soundness)**.** *Let a KMTS $\hat{M}$, a CTL formula ϕ with $(\hat{M}, \hat{s}) \not\models \phi$, for some $\hat{s}$ of $\hat{M}$. If AbstractRepair$(\hat{M}, \hat{s}, \phi, \emptyset)$ returns a KMTS $\hat{M}'$, then $(\hat{M}', \hat{s}) \models \phi$.*

Proof. We use structural induction on ϕ and Lemma 6.1 in the inductive step for $\phi_1 \wedge \phi_2$.

Base Case:

- if $\phi = \top$, Theorem 6.2 is trivially true, because $(\hat{M}, \hat{s}) \models \phi$.
- if $\phi = \bot$, then *AbstractRepair*$(\hat{M}, \hat{s}, \bot, \emptyset)$ returns FAILURE at line 2 of Algorithm 1 and the theorem is also trivially true.
- if $\phi = p \in AP$, *AbstractRepair*$_{ATOMIC}(\hat{M}, \hat{s}, p, \emptyset)$ is called at line 4 of Algorithm 1 and an $\hat{M}' = ChangeLabel(\hat{M}, \hat{s}, p)$ is computed at line 1. Because of the fact that $p \in \hat{L}'(\hat{s})$

in $\hat{M}'$, from 3-valued semantics of CTL over KMTSs we have $(\hat{M}', \hat{s}) \models \phi$. Algorithm 2 returns $\hat{M}'$ at line 3 because C is empty, and the theorem is true.

Induction Hypothesis: For CTL formulae ϕ_1, ϕ_2, the theorem is true. Thus, for ϕ_1 (resp. ϕ_2), if $AbstractRepair(\hat{M}, \hat{s}, \phi, \emptyset)$ returns a KMTS $\hat{M}'$, then $(\hat{M}', \hat{s}) \models \phi_1$.

Inductive Step:

- if $\phi = \phi_1 \vee \phi_2$, then $AbstractRepair(\hat{M}, \hat{s}, \phi, \emptyset)$ calls $AbstractRepair_{OR}(\hat{M}, \hat{s}, \phi_1 \vee \phi_2, \emptyset)$ at line 8 of Algorithm 1.

  From the induction hypothesis, if $AbstractRepair(\hat{M}, \hat{s}, \phi_1, \emptyset)$ returns a KMTS $\hat{M}_1$ at line 1 of Algorithm 3 and $AbstractRepair(\hat{M}, \hat{s}, \phi_2, \emptyset)$ returns a KMTS $\hat{M}_2$ respectively, then $(\hat{M}_1, \hat{s}) \models \phi_1$ and $(\hat{M}_2, \hat{s}) \models \phi_1$. $AbstractRepair_{OR}(\hat{M}, \hat{s}, \phi_1 \vee \phi_2, \emptyset)$ returns at line 8 of Algorithm 1 the KMTS $\hat{M}'$, which can be either $\hat{M}_1$ or $\hat{M}_2$. Therefore, $(\hat{M}', \hat{s}) \models \phi_1$ or $(\hat{M}', \hat{s}) \models \phi_2$. From 3-valued semantics of CTL, $(\hat{M}', \hat{s}) \models \phi_1 \vee \phi_2$ and the theorem is true.
- if $\phi = \phi_1 \wedge \phi_2$, then $AbstractRepair(\hat{M}, \hat{s}, \phi, \emptyset)$ calls $AbstractRepair_{AND}(\hat{M}, \hat{s}, \phi_1 \wedge \phi_2, \emptyset)$ at line 6 of Algorithm 1. From the induction hypothesis, if at line 1 of Algorithm 4 $AbstractRepair(\hat{M}, \hat{s}, \phi_1, \emptyset)$ returns a KMTS $\hat{M}_1$, then $(\hat{M}_1, \hat{s}) \models \phi_1$. Consequently, $\hat{M}_1 \models C_1$, where $C_1 = \emptyset \cup (\hat{s}, \phi_1)$. At line 7, if $AbstractRepair(\hat{M}_1, \hat{s}, \phi_2, C_1)$ returns a KMTS $\hat{M}_1'$, then from Lemma 6.1 $(\hat{M}_1', \hat{s}) \models \phi_2$ and $\hat{M}_1' \models C_1$.

  Likewise, if the calls at lines 2 and 12 of Algorithm 4 return the KMTSs $\hat{M}_2$ and $\hat{M}_2'$, then from the induction hypothesis $(\hat{M}_2, \hat{s}) \models \phi_2$ and from Lemma 6.1 $(\hat{M}_2', \hat{s}) \models \phi_1$, $\hat{M}_2' \models C_2$ with $C_2 = \emptyset \cup (\hat{s}, \phi_2)$.

  The KMTS $\hat{M}'$ at line 7 of Algorithm 1 can be either $\hat{M}_1'$ or $\hat{M}_2'$ and therefore, $(\hat{M}', \hat{s}) \models \phi_1$ and $(\hat{M}', \hat{s}) \models \phi_2$. From 3-valued semantics of CTL it holds that $(\hat{M}', \hat{s}) \models \phi_1 \wedge \phi_2$ and the lemma is true.
- if $\phi = EX\phi_1$, $AbstractRepair(\hat{M}, \hat{s}, \phi, \emptyset)$ calls $AbstractRepair_{EX}(\hat{M}, \hat{s}, EX\phi_1, \emptyset)$ at line 10 of Algorithm 1.

  If a KMTS $\hat{M}'$ is returned at line 5 of Algorithm 6, there is a state $\hat{s}_1$ with $(\hat{M}, \hat{s}_1) \models \phi_1$ such that $\hat{M}' = AddMust(\hat{M}, (\hat{s}, \hat{s}_1))$. From 3-valued semantics of CTL, we conclude that $(\hat{M}', \hat{s}) \models EX\phi_1$.

  If a $\hat{M}'$ is returned at line 11, there is $(\hat{s}, \hat{s}_1) \in R_{must}$ such that $(\hat{M}', \hat{s}_1) \models \phi_1$ from the induction hypothesis, since $\hat{M}' = AbstractRepair(\hat{M}, \hat{s}_1, \phi_1, \emptyset)$. From 3-valued semantics of CTL, we conclude that $(\hat{M}', \hat{s}) \models EX\phi_1$.

  If a $\hat{M}'$ is returned at line 18, a must transition $(\hat{s}, \hat{s}_n)$ to a new state has been added and $\hat{M}' = AbstractRepair(AddMust(\hat{M}, (\hat{s}, \hat{s}_n)), \hat{s}_n, \phi_1, \emptyset)$. Then, from the induction hypothesis $(\hat{M}', \hat{s}_n) \models \phi_1$ and from 3-valued semantics of CTL, we also conclude that $(\hat{M}', \hat{s}) \models EX\phi_1$.
- if $\phi = AG\phi_1$, $AbstractRepair(\hat{M}, \hat{s}, \phi, \emptyset)$ calls $AbstractRepair_{AG}(\hat{M}, \hat{s}, AG\phi_1, \emptyset)$ at line 10 of Algorithm 1. If $(\hat{M}, \hat{s}) \not\models \phi_1$ and $AbstractRepair(\hat{M}, \hat{s}, \phi_1, \emptyset)$ returns a KMTS $\hat{M}_0$ at line 2 of Algorithm 5, then from the induction hypothesis $(\hat{M}_0, \hat{s}) \models \phi_1$. Otherwise, $\hat{M}_0 = \hat{M}$ and $(\hat{M}_0, \hat{s}) \models \phi_1$, $\hat{M}_0 \models C$ also hold true.

  If Algorithm 5 returns a $\hat{M}'$ at line 16, this KMTS is the result of successive calls of $AbstractRepair(\hat{M}_i, \hat{s}_k, \phi_1, \emptyset)$ with $\hat{M}_i = AbstractRepair(\hat{M}_{i-1}, \hat{s}_k, \phi_1, \emptyset)$ and $i = 1, ...,$

for all may-reachable states $\hat{s}_k$ from $\hat{s}$ such that $(\hat{M}_0, \hat{s}_k) \not\models \phi_1$. From the induction hypothesis, $(\hat{M}', \hat{s}_k) \models \phi_1$ for all such $\hat{s}_k$ and from 3-valued semantics of CTL we conclude that $(\hat{M}', \hat{s}) \models AG\phi_1$.

We prove the theorem for all other cases in the same way. □

Theorem 6.2 shows that *AbstractRepair* is *sound* in the sense that if it returns a KMTS $\hat{M}'$, then $\hat{M}'$ satisfies property ϕ. In this case, from the definitions of the basic repair operations, it follows that one or more KSs can be obtained for which ϕ holds true.

6.2.2. *Semi-completeness.*

Definition 6.3 (*mr*-CTL)**.** Given a set AP of atomic propositions, we define the syntax of a CTL fragment inductively via a Backus Naur Form:

$$\phi ::== \bot \mid \top \mid p \mid (\neg\phi) \mid (\phi \vee \phi) \mid AXp \mid EXp \mid AFp \\ \mid EFp \mid AGp \mid EGp \mid A[p\,U\,p] \mid E[p\,U\,p]$$

where p ranges over AP.

mr-CTL includes most of the CTL formulae apart from those with nested path quantifiers or conjunction.

Theorem 6.4 (Completeness)**.** *Given a KMTS $\hat{M}$, an mr-CTL formula ϕ with $(\hat{M}, \hat{s}) \not\models \phi$, for some $\hat{s}$ of $\hat{M}$, if there exists a KMTS $\hat{M}''$ over the same set AP of atomic propositions with $(\hat{M}'', \hat{s}) \models \phi$, AbstractRepair$(\hat{M}, \hat{s}, \phi, \emptyset)$ returns a KMTS $\hat{M}'$ such that $(\hat{M}', \hat{s}) \models \phi$.*

Proof. We prove the theorem using structural induction on ϕ.

Base Case:

- if $\phi = \top$, Theorem 6.4 is trivially true, because for any KMTS $\hat{M}$ it holds that $(\hat{M}, \hat{s}) \models \phi$.
- if $\phi = \bot$, then the theorem is trivially true, because there does not exist a KMTS $\hat{M}''$ such that $(\hat{M}'', \hat{s}) \models \phi$.
- if $\phi = p \in AP$, there is a KMTS $\hat{M}''$ with $p \in \hat{L}''(\hat{s})$ and therefore $(\hat{M}'', \hat{s}) \models \phi$. Algorithm 1 calls $AbstractRepair_{ATOMIC}(\hat{M}, \hat{s}, p, \emptyset)$ at line 4 and an $\hat{M}' = ChangeLabel(\hat{M}, \hat{s}, p)$ is computed at line 1 of Algorithm 2. Since C is empty, $\hat{M}'$ is returned at line 3 and $(\hat{M}', \hat{s}) \models \phi$ from 3-valued semantics of CTL. Therefore, the theorem is true.

Induction Hypothesis: For *mr*-CTL formulae ϕ_1, ϕ_2, the theorem is true. Thus, for ϕ_1 (resp. ϕ_2), if there is a KMTS $\hat{M}''$ over the same set AP of atomic propositions with $(\hat{M}'', \hat{s}) \models \phi_1$, $AbstractRepair(\hat{M}, \hat{s}, \phi_1, \emptyset)$ returns a KMTS $\hat{M}'$ such that $(\hat{M}', \hat{s}) \models \phi_1$.

Inductive Step:

- if $\phi = \phi_1 \vee \phi_2$, from the 3-valued semantics of CTL a KMTS that satisfies ϕ exists if and only if there is a KMTS satisfying any of the ϕ_1, ϕ_2. From the induction hypothesis, if there is a KMTS $\hat{M}_1''$ with $(\hat{M}_1'', \hat{s}) \models \phi_1$, $AbstractRepair(\hat{M}, \hat{s}, \phi_1, \emptyset)$ at line 1 of Algorithm 3 returns a KMTS $\hat{M}_1'$ such that $(\hat{M}_1', \hat{s}) \models \phi_1$. Respectively, $AbstractRepair(\hat{M}, \hat{s}, \phi_2, \emptyset)$ at line 2 of Algorithm 3 can return a KMTS $\hat{M}_2'$ with $(\hat{M}_2', \hat{s}) \models \phi_2$. In any case, if either $\hat{M}_1'$ or $\hat{M}_2'$ exists, for the KMTS $\hat{M}'$ that is returned at line 13 of Algorithm 3 we have $(\hat{M}', \hat{s}) \models \phi_1$ or $(\hat{M}', \hat{s}) \models \phi_2$ and therefore $(\hat{M}', \hat{s}) \models \phi$.

- if $\phi = EX\phi_1$, from the 3-valued semantics of CTL a KMTS that satisfies ϕ at $\hat{s}$ exists if and only if there is KMTS satisfying ϕ_1 at some direct must-successor of $\hat{s}$.

  If in the KMTS $\hat{M}$ there is a state $\hat{s}_1$ with $(\hat{M}, \hat{s}_1) \models \phi_1$, then the new KMTS $\hat{M}' = AddMust(\hat{M}, (\hat{s}, \hat{s}_1))$ is computed at line 3 of Algorithm 6. Since C is empty $\hat{M}'$ is returned at line 5 and $(\hat{M}', \hat{s}) \models EX\phi_1$.

  Otherwise, if there is a direct must-successor $\hat{s}_i$ of $\hat{s}$, $AbstractRepair(\hat{M}, \hat{s}_i, \phi_1, \emptyset)$ is called at line 8. From the induction hypothesis, if there is a KMTS $\hat{M}''$ with $(\hat{M}'', \hat{s}_i) \models \phi_1$, then a KMTS $\hat{M}'$ is computed such that $(\hat{M}', \hat{s}_i) \models \phi_1$ and therefore the theorem is true.

  If there are no must-successors of $\hat{s}$, a new state $\hat{s}_n$ is added and subsequently connected with a must-transition from $\hat{s}$. *AbstractRepair* is then called for ϕ_1 and $\hat{s}_n$ as previously and the theorem holds also true.
- if $\phi = AG\phi_1$, from the 3-valued semantics of CTL a KMTS that satisfies ϕ at $\hat{s}$ exists, if and only if there is KMTS satisfying ϕ_1 at $\hat{s}$ and at each may-reachable state from $\hat{s}$.

  $AbstractRepair(\hat{M}, \hat{s}, \phi_1, \emptyset)$ is called at line 2 of Algorithm 5 and from the induction hypothesis if there is KMTS $\hat{M}_0'$ with $(\hat{M}_0', \hat{s}) \models \phi_1$, then a KMTS $\hat{M}_0$ is computed such that $(\hat{M}_0, \hat{s}) \models \phi_1$. *AbstractRepair* is subsequently called for ϕ_1 and for all may-reachable $\hat{s}_k$ from $\hat{s}$ with $(\hat{M}_0, \hat{s}_k) \not\models \phi_1$ one-by-one. From the induction hypothesis, if there is KMTS $\hat{M}_i'$ that satisfies ϕ_1 at each such $\hat{s}_k$, then all $\hat{M}_i = AbstractRepair(\hat{M}_{i-1}, \hat{s}_k, \phi_1, \emptyset)$, $i = 1, ...,$ satisfy ϕ_1 at $\hat{s}_k$ and the theorem holds true.

We prove the theorem for all other cases in the same way. □

Theorem 6.4 shows that *AbstractRepair* is *semi-complete* with respect to full CTL: if there is a KMTS that satisfies a *mr*-CTL formula ϕ, then the algorithm finds one such KMTS.

6.3. **Complexity Issues.** AMR's complexity analysis is restricted to *mr*-CTL, for which the algorithm has been proved complete. For these formulas, we show that AMR is upper bounded by a polynomial expression in the state space size and the number of may-transitions of the abstract KMTS, and depends also on the length of the *mr*-CTL formula.

For CTL formulas with nested path quantifiers and/or conjunction, AMR is looking for a repaired model satisfying all conjunctives (constraints), which increases the worst-case execution time exponentially to the state space size of the abstract KMTS. In general, as shown in [10], the complexity of all model repair algorithms gets worse when raising the level of their completeness, but AMR has the advantage of working exclusively over an abstract model with a reduced state space compared to its concrete counterpart.

Our complexity analysis for *mr*-CTL is based on the following results. For an abstract KMTS $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$ and a *mr*-CTL property ϕ, (i) 3-valued CTL model checking is performed in $O(|\phi| \cdot (|\hat{S}| + |R_{may}|))$ [31], (ii) Depth First Search (DFS) of states reachable from $\hat{s} \in \hat{S}$ is performed in $O(|\hat{S}| + |R_{may}|)$ in the worst case or in $O(|\hat{S}| + |R_{must}|)$ when only must-transitions are accessed, (iii) finding a maximal path from $\hat{s} \in \hat{S}$ using Breadth First Search (BFS) is performed in $O(|\hat{S}| + |R_{may}|)$ for may-paths and in $O(|\hat{S}| + |R_{must}|)$ for must-paths.

We analyze the computational cost for each of the AMR's primitive functions:

- if $\phi = p \in AP$, $AbstractRepair_{ATOMIC}$ is called and the operation *ChangeLabel* is applied, which is in $O(1)$.

- if $\phi = EX\phi_1$, then $AbstractRepair_{EX}$ is called and the applied operations with the highest cost are: (1) finding a state satisfying ϕ_1, which depends on the cost of 3-valued CTL model checking and is in $O(|\hat{S}| \cdot |\phi_1| \cdot (|\hat{S}| + |R_{may}|))$, (2) finding a must-reachable state, which is in $O(|\hat{S}| + |R_{must}|)$. These operations are called at most once and the overall complexity for this primitive functions is therefore in $O(|\hat{S}| \cdot |\phi_1| \cdot (|\hat{S}| + |R_{may}|))$.
- if $\phi = AX\phi_1$, then $AbstractRepair_{AX}$ is called and the most costly operations are: (1) finding a may-reachable state, which is in $O(|\hat{S}| + |R_{may}|)$, and (2) checking if a state satisfies ϕ_1, which is in $O(|\phi_1| \cdot (|\hat{S}| + |R_{may}|))$. These operations are called at most $|\hat{S}|$ times and the overall bound class is $O(|\hat{S}| \cdot |\phi_1| \cdot (|\hat{S}| + |R_{may}|))$.
- if $\phi = EF\phi_1$, $AbstractRepair_{EF}$ is called and the operations with the highest cost are: (1) finding a must-reachable state, which is in $O(|\hat{S}| + |R_{must}|)$, (2) checking if a state satisfies ϕ_1 with its bound class being $O(|\phi_1| \cdot (|\hat{S}| + |R_{may}|))$ and (3) finding a state that satisfies ϕ_1, which is in $O(|\hat{S}| \cdot |\phi_1| \cdot (|\hat{S}| + |R_{may}|))$. These three operations are called at most $|\hat{S}|$ times and consequently, the overall bound class is $O(|\hat{S}|^2 \cdot |\phi_1| \cdot (|\hat{S}| + |R_{may}|))$.
- if $\phi = AF\phi_1$, $AbstractRepair_{AF}$ is called and the most costly operation is: finding a maximal may-path violating ϕ_1 in all states, which is in $O(|\hat{S}| \cdot |\phi_1| \cdot (|\hat{S}| + |R_{may}|)$. This operation is called at most $|\hat{S}|$ times and therefore, the overall bound class is $O(|\hat{S}|^2 \cdot |\phi_1| \cdot (|\hat{S}| + |R_{may}|))$.

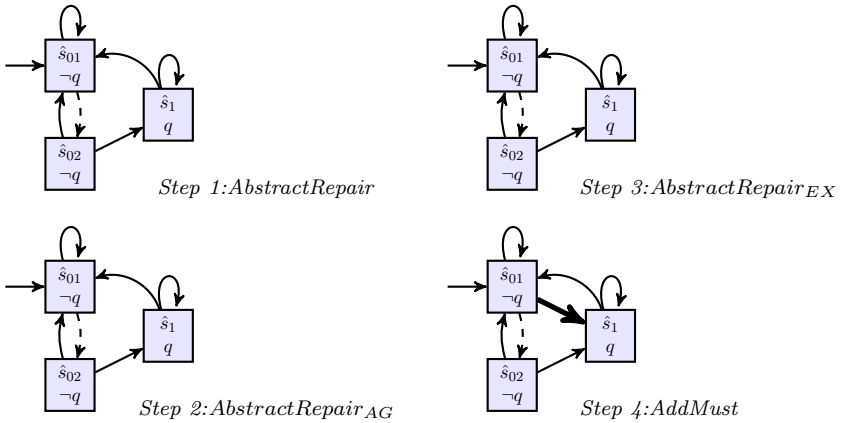
In the same way, it is easy to show that: (i) if $\phi = EG\phi_1$, then $AbstractRepair_{EG}$ is in $O(|\hat{S}| \cdot |\phi_1| \cdot (|\hat{S}| + |R_{must}|)$, (ii) if $\phi = AG\phi_1$, then $AbstractRepair_{AG}$ is in $O(|\hat{S}| \cdot |\phi_1| \cdot (|\hat{S}| + |R_{may}|))$, (iii) if $\phi = E(\phi_1 U \phi_2)$, then the bound class of $AbstractRepair_{EU}$ is $O(|\hat{S}| \cdot |\phi_1| \cdot (|\hat{S}| + |R_{must}|)$, (iv) if $\phi = A(\phi_1 U \phi_2)$ then $AbstractRepair_{AU}$ is in $O(|\hat{S}|^2 \cdot |\phi_1| \cdot (|\hat{S}| + |R_{may}|))$.

For a *mr*-CTL property ϕ, the main body of the algorithm is called at most $|\phi|$ times and the overall bound class of the AMR algorithm is $O(|\hat{S}|^2 \cdot |\phi|^2 \cdot (|\hat{S}| + |R_{may}|))$.

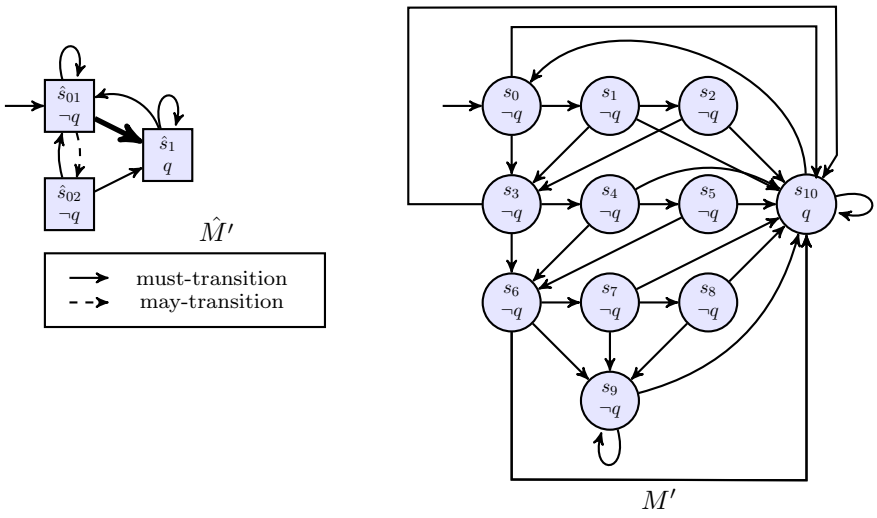
6.4. **Application.** We present the application of *AbstractRepair* on the ADO system from Section 2. After the first two steps of our repair process, *AbstractRepair* is called for the KMTS $\alpha_{Refined}(M)$ that is shown in Fig. 3b, the state $\hat{s}_{01}$ and the CTL property $\phi = AGEXq$.

AbstractRepair calls $AbstractRepair_{AG}$ with arguments $\alpha_{Refined}(M)$, $\hat{s}_{01}$ and $AGEXq$. The $AbstractRepair_{AG}$ algorithm at line 10 triggers a recursive call of *AbstractRepair* with the same arguments. Eventually, $AbstractRepair_{EX}$ is called with arguments $\alpha_{Refined}(M)$, $\hat{s}_{01}$ 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 $\hat{M'}$ (cf. Def. 5.3). The execution steps of *AbstractRepair* and the obtained repaired KMTS and KS are shown in Fig. 13a and Fig. 13b 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, new transitions would have have been inserted from all the states labeled with $\neg$*open* to the one labeled with *open*. In the ADO, we have seven such states, but in a system with a large state space this number can be significantly higher. The repair of such a model without the use of abstraction would be impractical.



(A) Application of *AbstractRepair*.



(B) The repaired KMTS and KS.

FIGURE 13. Repair of ADO system using abstraction.

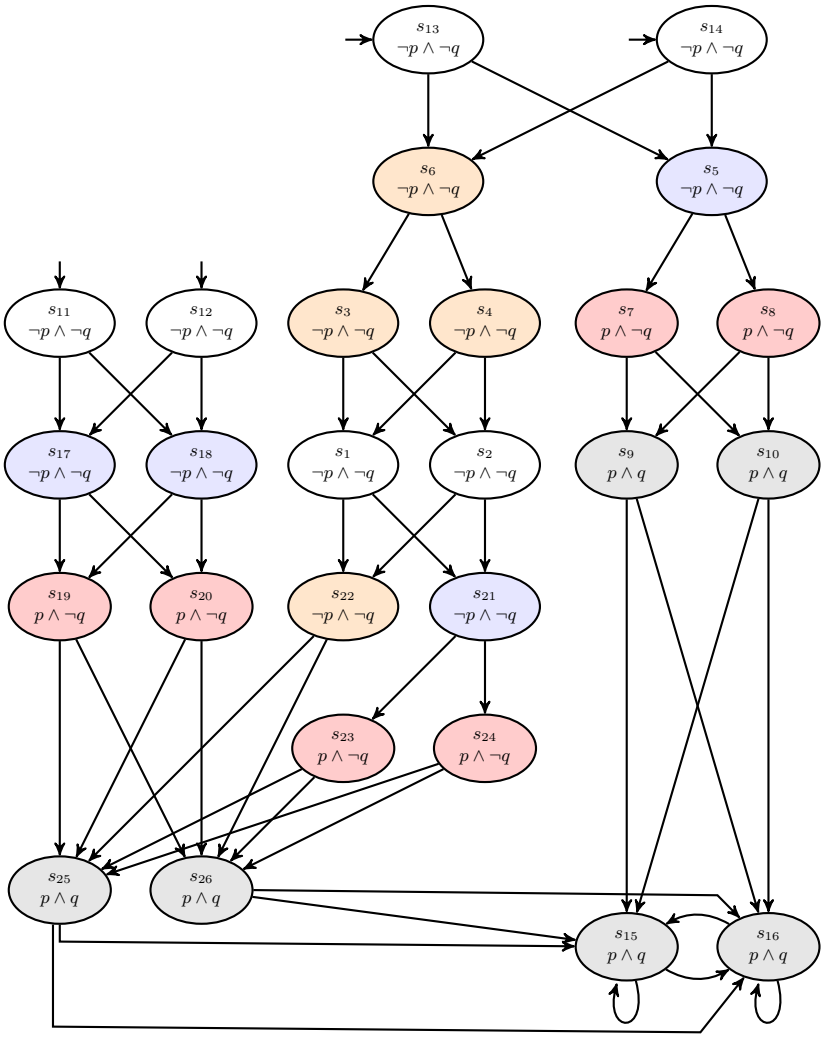
7. Experimental Results: The Andrew File System 1 (AFS1) Protocol

In this section, we provide experimental results for the relative performance of a prototype implementation of our AMR algorithm in comparison with a prototype implementation of a concrete model repair solution [55]. The results serve as a proof of concept for the use of abstraction in model repair and demonstrate the practical utility of our approach.

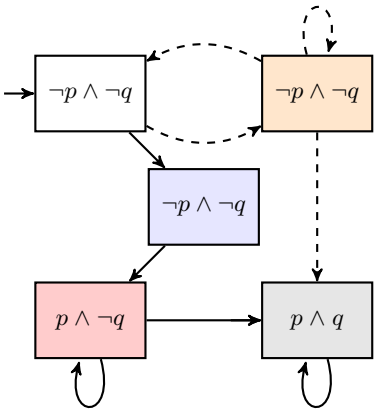
As a model we use a KS for the Andrew File System Protocol 1 (AFS1) [54], which has been repaired for a specific property in [55]. AFS1 is a client-server cache coherence protocol for a distributed file system. Four values are used for the client's belief about a file (nofile, valid, invalid, suspect) and three values for the server's belief (valid, invalid, none).

A property which is not satisfied in the AFS1 protocol in the form of CTL is:

$$AG((Server.belief = valid) \rightarrow (Client.belief = valid))$$

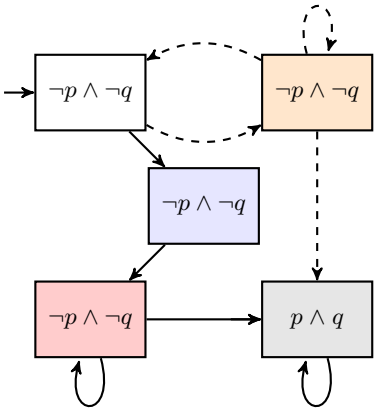


(A) The KS after the final refinement step.

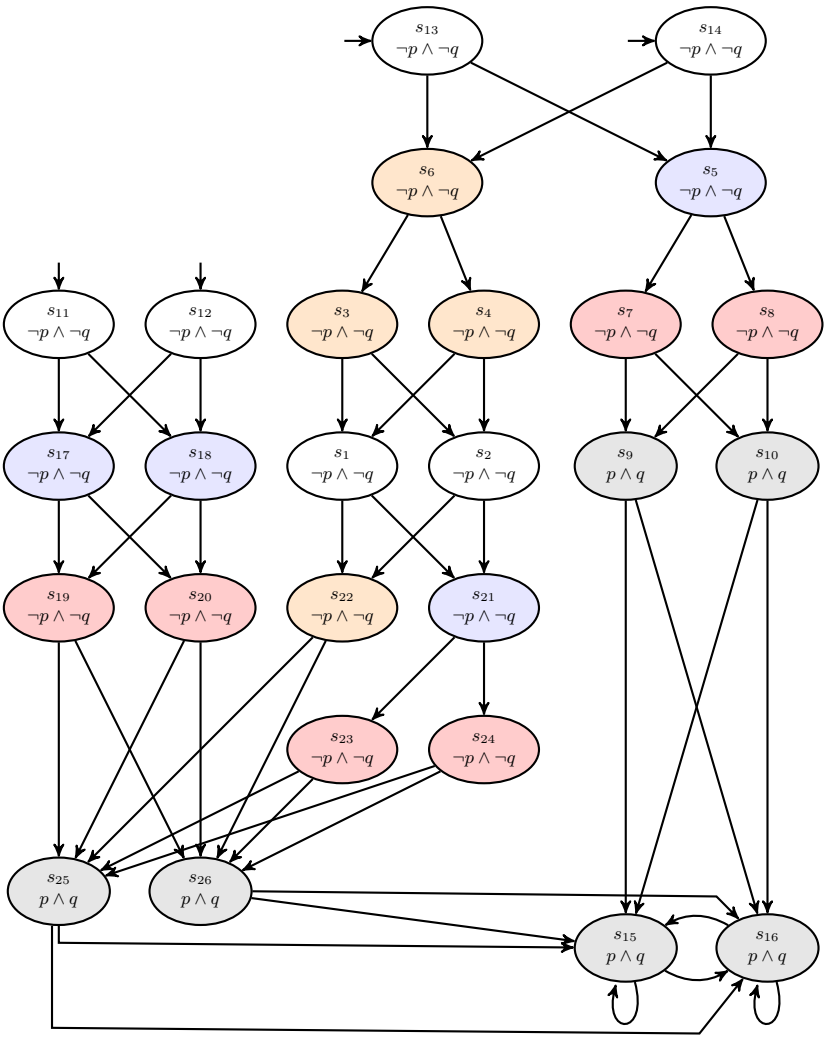


(B) The refined KMTS.

FIGURE 14. The KS and the KMTS of the AFS1 protocol after the 2nd refinement step.



(A) The repaired KMTS.



(B) The repaired KS.

FIGURE 15. The repaired KMTS and KS of the AFS1 protocol.

Models	Concrete States	Concr. Repair (Time in sec.)	AMR (Time in sec.)	Improvement (%)
$AFS1$	26	17.4	0.14	124
$AFS1(Extension1)$	30	24.9	0.14	178
$AFS1(Extension2)$	34	35.0	0.14	250
$AFS1(Extension3)$	38	117.0	0.14	836

TABLE 1. Experimental results of AMR with respect to concrete repair

We define the atomic proposition p as $Server.belief = valid$ and q as $Client.belief = valid$, and the property is thus written as $AG(p \rightarrow q)$. The KS for the AFS1 protocol is depicted in Fig. 14a. State colors show how they are abstracted in the KMTS of Fig. 14b, which is derived after the 2nd refinement step of our AMR framework (Fig. 4). The shown KMTS and the CTL property of interest are given as input in our prototype AMR implementation.

To obtain larger models of AFS1 we have extended the original model by adding one more possible value for three model variables. Three new models are obtained with gradually increasing size of state space.

The results of our experiments are presented in Table 1. The time needed for the AMR prototype to repair the original AFS1 model and its extensions is from 124 to even 836 times less than the needed time for concrete model repair. The repaired KMTS and KS for the original AFS1 model are shown in Fig. 15.

An interesting observation from the application of the AMR algorithm on the repair of the AFS1 KS is that the distance d (cf. Def. 4.1) of the repaired KS from the original KS is less than the corresponding distance obtained from the concrete model repair algorithm in [55]. This result demonstrates in practice the effect of the minimality of changes ordering, on which the AMR algorithm is based on (cf. Fig. 12).

8. 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 search of a Model Repair solution. However, abstraction and in particular abstract interpretation has been used in *program synthesis* [50], a different but related problem to the Model Repair. Program synthesis refers to the automatic generation of a program based on a given specification. Another related problem where abstraction has been used is that of *trigger querying* [4]: given a system M and a formula ϕ, find the set of scenarios that trigger ϕ in M.

The related work in the area of *program repair* do not consider KSs as the program model. In this context, abstraction has been previously used in the repair of data structures [43]. The problem of repairing a Boolean program has been formulated in [48, 40, 34, 51] as the finding of a winning strategy for a game between two players. The only exception is the work reported in [45].

Another line of research on program repair treats the repair as a search problem and applies innovative evolutionary algorithms [3], *behavioral programming* techniques [37] or other informal heuristics [52, 1, 53].

Focusing exclusively on the area of Model Repair without the use of abstraction, it is worth to mention the following approaches. The first work on Model Repair with respect to CTL formulas was presented in [2]. The authors used only the removal of transitions and showed that the problem is NP-complete. Another interesting early attempt to introduce the Model Repair problem for CTL properties is the work in [12]. The authors are based on the AI techniques of abductive reasoning and theory revision and propose a repair algorithm with relatively high computational cost. A formal algorithm for Model Repair in the context of KSs and CTL is presented in [55]. The authors admit that their repair process strongly depends on the model's size and they do not attempt to provide a solution for handling conjunctive CTL formulas.

In [14], the authors try to render model repair applicable to large KSs by using "table systems", a concise representation of KSs that is implemented in the NuSMV model checker. A limitation of their approach is that table systems cannot represent all possible KSs. In [56], tree-like local model updates are introduced with the aim of making the repair process applicable to large-scale domains. However, the proposed approach is only applicable to the universal fragment of the CTL.

A number of works attempt to ensure completeness for increasingly larger fragments of the CTL by introducing ways of handling the constraints associated with conjunctive formulas. In [41], the authors propose the use of constraint automata for ACTL formulas, while in [13] the authors introduce the use of protected models for an extension of the CTL. Both of the two methods are not directly applicable to formulas of the full CTL.

The Model Repair problem has been also addressed in many other contexts. In [27], the author uses a distributed algorithm and the processing power of computing clusters to fight the time and space complexity of the repair process. In [25], an extension of the Model Repair problem has been studied for Labeled Transition Systems. In [6], we have provided a solution for the Model Repair problem in probabilistic systems. Another recent effort for repairing discrete-time probabilistic models has been proposed in [44]. In [7], model repair is applied to the *fault recovery* of component-based models. Finally, a slightly different but also related problem is that of Model Revision, which has been studied for UNITY properties in [8, 9] and for CTL in [36]. Other methods in the area of fault-tolerance include the work in [30], which uses discrete controller synthesis and [29], which employs SMT solving. Another interesting work in this direction is in [26], where the authors present a repair algorithm for fault-tolerance in a fully connected topology, with respect to a temporal specification.

9. Conclusions

In this paper, we have shown how abstraction can be used to cope with the state explosion problem 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. We have proved that our AMR algorithm is sound for the full CTL and complete for a subset of CTL. We have also proved that our AMR algorithm is upper bounded by a polynomial expression in the size of the abstract model for a major fragment of CTL. To demonstrate its practical utility, we applied our framework to an Automatic Door Opener system and to the Andrew File System 1 protocol.

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. A

model repair application of high interest is in the design of fault-tolerant systems. In [11], the authors present an approach for the repair of a distributed algorithm such that the repaired one features fault-tolerance. The input to this model repair problem includes a set of uncontrollable transitions such as the faults in the system. The model repair algorithm used works on concrete models and it can therefore solve the problem only for a limited number of processes. With this respect, we believe that this application could be benefited from the use of abstraction in our AMR framework.

At the level of extending our AMR framework, we aim to search for "better" abstract models, in order to either restrict failures due to refinement or ensure completeness for a larger fragment of the CTL. We will also investigate different notions of minimality in the changes introduced by model repair and the applicability of abstraction-based model repair to probabilistic, hybrid and other types of models.

10. Acknowledgment

This work was partially sponsored by Canada NSERC Discovery Grant 418396-2012 and NSERC Strategic Grants 430575-2012 and 463324-2014. The research was also co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thalis Athens University of Economics and Business - SOFTWARE ENGINEERING RESEARCH PLATFORM.

References

[1] Thomas Ackling, Bradley Alexander, and Ian Grunert. Evolving patches for software repair. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 1427–1434, New York, NY, USA, 2011. ACM.

[2] Marco Antoniotti. *Synthesis and Verification of Discrete Controllers for Robotics and Manufacturing Devices with Temporal Logic and the Control-D System.* PhD thesis, New York University, 1995.

[3] Andrea Arcuri. Evolutionary repair of faulty software. *Appl. Soft Comput.*, 11:3494–3514, June 2011.

[4] Guy Avni and Orna Kupferman. An abstraction-refinement framework for trigger querying. *Formal Methods in System Design*, 44(2):149–175, 2014.

[5] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series).* The MIT Press, 2008.

[6] Ezio Bartocci, Radu Grosu, Panagiotis Katsaros, C. R. Ramakrishnan, and Scott A. Smolka. Model repair for probabilistic systems. In *Proceedings of the 17th international conference on Tools and algorithms for the construction and analysis of systems: part of the joint European conferences on theory and practice of software*, TACAS'11/ETAPS'11, pages 326–340, Berlin, Heidelberg, 2011. Springer-Verlag.

[7] Borzoo Bonakdarpour, Marius Bozga, and Gregor Goessler. A theory of fault recovery for component-based models. In *Proceedings of the 2011 IEEE 30th International Symposium on Reliable Distributed Systems*, SRDS '11, pages 265–270, Washington, DC, USA, 2011. IEEE Computer Society.

[8] Borzoo Bonakdarpour, Ali Ebnenasir, and Sandeep S. Kulkarni. Complexity results in revising UNITY programs. *ACM Trans. Auton. Adapt. Syst.*, 4:5:1–5:28, February 2009.

[9] Borzoo Bonakdarpour and Sandeep S. Kulkarni. Revising distributed UNITY programs is NP-complete. In *Principles of Distributed Systems (OPODIS)*, pages 408–427, 2008.

[10] Borzoo Bonakdarpour and Sandeep S. Kulkarni. Automated model repair for distributed programs. *SIGACT News*, 43(2):85–107, jun 2012.

[11] Borzoo Bonakdarpour, Sandeep S. Kulkarni, and Fuad Abujarad. Symbolic synthesis of masking fault-tolerant programs. *Springer Journal on Distributed Computing*, 25(1):83–108, March 2012.

[12] Francesco Buccafurri, Thomas Eiter, Georg Gottlob, and Nicola Leone. Enhancing model checking in verification by AI techniques. *Artif. Intell.*, 112:57–104, August 1999.

[13] Miguel Carrillo and David Rosenblueth. Nondeterministic update of CTL models by preserving satisfaction through protections. In Tevfik Bultan and Pao-Ann Hsiung, editors, *Automated Technology for Verification and Analysis*, volume 6996 of *Lecture Notes in Computer Science*, pages 60–74. Springer Berlin / Heidelberg, 2011.

[14] Miguel Carrillo and David. A. Rosenblueth. A method for CTL model update, representing Kripke Structures as table systems. *IJPAM*, 52:401–431, January 2009.

[15] George Chatzieleftheriou, Borzoo Bonakdarpour, Scott A. Smolka, and Panagiotis Katsaros. Abstract model repair. In *Proceedings of the 4th international conference on NASA Formal Methods*, NFM'12, pages 341–355, Berlin, Heidelberg, 2012. Springer-Verlag.

[16] Edmund M. Clarke, E. Allen Emerson, and Joseph Sifakis. Model checking: Algorithmic verification and debugging. *Communications of the ACM*, 52(11):74–84, 2009.

[17] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Proceedings of the 12th International Conference on Computer Aided Verification*, CAV '00, pages 154–169, London, UK, 2000. Springer-Verlag.

[18] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. *ACM Trans. Program. Lang. Syst.*, 16:1512–1542, September 1994.

[19] Byron Cook, Andreas Podelski, and Andrey Rybalchenko. Abstraction refinement for termination. In Chris Hankin and Igor Siveroni, editors, *Static Analysis*, volume 3672 of *Lecture Notes in Computer Science*, pages 87–101. Springer Berlin / Heidelberg, 2005.

[20] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL '77*, pages 238–252, New York, NY, USA, 1977. ACM.

[21] Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, POPL '79, pages 269–282, New York, NY, USA, 1979. ACM.

[22] Patrick Cousot, Pierre Ganty, and Jean-Franois Raskin. Fixpoint-guided abstraction refinements. In Hanne Nielson and Gilberto Fil, editors, *Static Analysis*, volume 4634 of *Lecture Notes in Computer Science*, pages 333–348. Springer Berlin / Heidelberg, 2007.

[23] Dennis Dams, Rob Gerth, and Orna Grumberg. Abstract interpretation of reactive systems. *ACM Trans. Program. Lang. Syst.*, 19:253–291, March 1997.

[24] DR. Dams. *Abstract interpretation and partition refinement for model checking*. PhD thesis, Technische Universiteit Eindhoven, 1996.

[25] Maria de Menezes, Silvio do Lago Pereira, and Leliane de Barros. System design modification with actions. In Antnio da Rocha Costa, Rosa Vicari, and Flavio Tonidandel, editors, *Advances in Artificial Intelligence SBIA 2010*, volume 6404 of *Lecture Notes in Computer Science*, pages 31–40. Springer Berlin / Heidelberg, 2011.

[26] Rayna Dimitrova and Bernd Finkbeiner. Synthesis of fault-tolerant distributed systems. In *Proceedings of the 7th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, pages 321–336, 2009.

[27] Ali Ebnenasir. Action-based discovery of satisfying subsets: A distributed method for model correction. *Information and Software Technology*, 2012.

[28] E. A. Emerson and Joseph Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *J. Comput. Syst. Sci.*, 30:1–24, February 1985.

[29] Fathiyeh Faghih and Borzoo Bonakdarpour. SMT-based synthesis of distributed self-stabilizing systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 2015. To appear.

[30] Alain Girault and Éric Rutten. Automating the addition of fault tolerance with discrete controller synthesis. *Formal Methods in System Design (FMSD)*, 35(2):190–225, 2009.

[31] Patrice Godefroid, Michael Huth, and Radha Jagadeesan. Abstraction-based model checking using modal transition systems. In *Proceedings of the 12th International Conference on Concurrency Theory*, CONCUR '01, pages 426–440, London, UK, 2001. Springer-Verlag.

[32] Patrice Godefroid and Radha Jagadeesan. Automatic abstraction using generalized model checking. In *Proceedings of the 14th International Conference on Computer Aided Verification*, CAV '02, pages 137–150, London, UK, UK, 2002. Springer-Verlag.

[33] Susanne Graf and Hassen Saidi. Construction of abstract state graphs with pvs. In Orna Grumberg, editor, *Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 72–83. Springer Berlin / Heidelberg, 1997.

[34] Andreas Griesmayer, Roderick Bloem, and Byron Cook. Repair of boolean programs with an application to C. In Thomas Ball and Robert Jones, editors, *Computer Aided Verification*, volume 4144 of *Lecture Notes in Computer Science*, pages 358–371. Springer Berlin / Heidelberg, 2006.

[35] Orna Grumberg, Martin Lange, Martin Leucker, and Sharon Shoham. When not losing is better than winning: Abstraction and refinement for the full mu-calculus. *Inf. Comput.*, 205:1130–1148, August 2007.

[36] Paulo T. Guerra and Renata Wassermann. Revision of CTL models. In *Proceedings of the 12th Ibero-American Conference on Advances in Artificial Intelligence*, IBERAMIA'10, pages 153–162, Berlin, Heidelberg, 2010. Springer-Verlag.

[37] D. Harel, G. Katz, A. Marron, and G. Weiss. Non-intrusive repair of reactive programs. In *Engineering of Complex Computer Systems (ICECCS), 2012 17th International Conference on*, pages 3–12, july 2012.

[38] Michael Huth, Radha Jagadeesan, and David A. Schmidt. Modal transition systems: A foundation for three-valued program analysis. In *Proceedings of the 10th European Symposium on Programming Languages and Systems*, ESOP '01, pages 155–169, London, UK, 2001. Springer-Verlag.

[39] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, August 2004.

[40] Barbara Jobstmann, Andreas Griesmayer, and Roderick Bloem. Program repair as a game. In Kousha Etessami and Sriram Rajamani, editors, *Computer Aided Verification*, volume 3576 of *Lecture Notes in Computer Science*, pages 287–294. Springer Berlin / Heidelberg, 2005.

[41] Michael Kelly, Fei Pu, Yan Zhang, and Yi Zhou. ACTL local model update with constraints. In *Proceedings of the 14th international conference on Knowledge-based and intelligent information and engineering systems: Part IV*, KES'10, pages 135–144, Berlin, Heidelberg, 2010. Springer-Verlag.

[42] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, S. Bensalem, and David Probst. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6:11–44, 1995.

[43] Razieh Nokhbeh Zaeem, MuhammadZubair Malik, and Sarfraz Khurshid. Repair abstractions for more efficient data structure repair. In Axel Legay and Saddek Bensalem, editors, *Runtime Verification*, volume 8174 of *Lecture Notes in Computer Science*, pages 235–250. Springer Berlin Heidelberg, 2013.

[44] Shashank Pathak, Erika Ábrahám, Nils Jansen, Armando Tacchella, and Joost-Pieter Katoen. A greedy approach for the efficient repair of stochastic models. In Klaus Havelund, Gerard Holzmann, and Rajeev Joshi, editors, *NASA Formal Methods*, volume 9058 of *Lecture Notes in Computer Science*, pages 295–309. Springer International Publishing, 2015.

[45] Roopsha Samanta, Jyotirmoy V. Deshmukh, and E. Allen Emerson. Automatic generation of local repairs for boolean programs. In *Proceedings of the 2008 International Conference on Formal Methods in Computer-Aided Design*, FMCAD '08, pages 27:1–27:10, Piscataway, NJ, USA, 2008. IEEE Press.

[46] Sharon Shoham and Orna Grumberg. Monotonic abstraction-refinement for CTL. In Kurt Jensen and Andreas Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2988 of *Lecture Notes in Computer Science*, pages 546–560. Springer Berlin / Heidelberg, 2004.

[47] Sharon Shoham and Orna Grumberg. A game-based framework for CTL counterexamples and 3-valued abstraction-refinement. *ACM Trans. Comput. Logic*, 9, 2007.

[48] Stefan Staber, Barbara Jobstmann, and Roderick Bloem. Finding and fixing faults. In Dominique Borrione and Wolfgang Paul, editors, *Correct Hardware Design and Verification Methods*, volume 3725 of *Lecture Notes in Computer Science*, pages 35–49. Springer Berlin / Heidelberg, 2005.

[49] Jan Van den Bussche, Dirk Van Gucht, and Stijn Vansummeren. Well-definedness and semantic type-checking for the nested relational calculus. *Theor. Comput. Sci.*, 371(3):183–199, feb 2007.

[50] Martin Vechev, Eran Yahav, and Greta Yorsh. Abstraction-guided synthesis of synchronization. In *Proceedings of the 37th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '10, pages 327–338, New York, NY, USA, 2010. ACM.

[51] Christian von Essen and Barbara Jobstmann. Program repair revisited. Technical Report TR-2012-4, Verimag Research Report, 2012.

[52] Farn Wang and Chih-Hong Cheng. Program repair suggestions from graphical state-transition specifications. In *Proceedings of the 28th IFIP WG 6.1 international conference on Formal Techniques for*

Networked and Distributed Systems, FORTE '08, pages 185–200, Berlin, Heidelberg, 2008. Springer-Verlag.

[53] Yi Wei, Yu Pei, Carlo A. Furia, Lucas S. Silva, Stefan Buchholz, Bertrand Meyer, and Andreas Zeller. Automated fixing of programs with contracts. In *Proceedings of the 19th international symposium on Software testing and analysis*, ISSTA '10, pages 61–72, New York, NY, USA, 2010. ACM.

[54] Jeannette M. Wing and Mandana Vaziri-Farahani. Model checking software systems: a case study. In *Proceedings of the 3rd ACM SIGSOFT symposium on Foundations of software engineering*, SIGSOFT '95, pages 128–139, New York, NY, USA, 1995. ACM.

[55] Yan Zhang and Yulin Ding. CTL model update for system modifications. *J. Artif. Int. Res.*, 31:113–155, January 2008.

[56] Yan Zhang, Michael Kelly, and Yi Zhou. Foundations of tree-like local model updates. In *Proceeding of the 2010 conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, pages 615–620, Amsterdam, The Netherlands, The Netherlands, 2010. IOS Press.