ABSTRACT REPAIR OF TRANSITION SYSTEMS

George Chatzieleftheriou

ARISTOTLE
UNIVERSITY OF
THESSALONIKI

A thesis submitted for the degree of

Doctor of Philosophy

June 2018

Examiners committee

- Panagiotis Katsaros, Assistant Professor, Department of Informatics, Aristotle University of Thessaloniki (*advisor*)
- Scott A. Smolka, Distinguished Professor, Department of Computer Science, Stony Brook University (*advisory commitee member*)
- Ioannis Stamelos, Professor, Department of Informatics, Aristotle University of Thessaloniki (*advisory commitee member*)
- Joost-Pieter Katoen, Distinguished Professor, Department of Computer Science, RWTH Aachen University
- David Parker, Reader, School of Computer Science, University of Birmingham
- George Rahonis, Associate Professor, Department of Mathematics, Aristotle University of Thessaloniki
- Kostas Tsichlas, Assistant Professor, Department of Informatics, Aristotle University of Thessaloniki

Abstract

Given a transition system M and a specification formula ϕ, the problem of model checking is to determine if M satisfies φ. An extended problem of model checking is that of *model repair*. In the case that M violates 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.

This thesis examines the problem of model repair for (i) Kripke structures and Computation Tree Logic and, (ii) probabilistic systems and reachability temporal logic properties. For Kripke structures, this thesis presents a framework for model repair that uses *abstraction refinement* to tackle state space explosion. The proposed framework aims to repair Kripke Structure models based on a Kripke Modal Transition System abstraction and a 3-valued semantics for CTL. An abstract-model-repair algorithm is introduced for which soundness and semi-completeness are proven, and its complexity class is studied. 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.

For probabilistic systems, this thesis presents a framework based on abstraction and refinement, which reduces the state space of the probabilistic system to repair at the price of obtaining an approximate solution. A metric space is defined over the set of DTMCs, in order to measure the differences between the initial and the repaired models. For the repair, this thesis introduces an algorithm and discusses its important properties, such as soundness and complexity. As a proof of concept, experimental results are provided for probabilistic systems with diverse structures of state spaces, including the well-known Craps game, the IPv4 Zeroconf protocol, a message authentication protocol and the gambler's ruin model.

Dedication

This thesis is dedicated to my lovely and funny wife Dora

Publications

- 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.
- George Chatzieleftheriou, Borzoo Bonakdarpour, Panagiotis Katsaros, and Scott A. Smolka. Abstract Model Repair. *Logical Methods in Computer Science*, Volume 11, Issue 3, Sep 2015.
- George Chatzieleftheriou, Panagiotis Katsaros. Abstract model repair for probabilistic systems. *Information and Computation*, Volume 259, Part 1, Pages 142-160, 2018.

Acknowledgements

I feel gratitude for my supervisor Prof. Panagiotis Katsaros, for all his guidance, encouragement and support during all these years of my PhD studies. He has been a great mentor and partner. I truly want to thank him because he has given me the freedom to pursue my research interests together with his invaluable insights and advice.

I am honored for Prof. Scott A. Smolka being a member of the supervisory committee of this thesis. A significant portion of this work has been inspired by the discussions with him at the first years of my PhD studies.

I want to thank my beloved parents, Nikos and Maria, for all their efforts for me since I was a child. I hope they are proud for their son.

To my wife, life companion, and mother of my child, Dr. Dora Matzari, I want only to thank you for making my life as beautiful as it could possibly be, despite the difficult circumstances faced. Your support and understanding for my research goals and the way you take care of our little daughter, Nikoleta, have been priceless all these years.

Περίληψη

Στη σύχρονη εποχή, τα συστήματα υπολογιστών είναι ενσωματωμένα σχεδόν σε όλες τις μηχανικές κατασκευές και εμφανίζονται σε κάποιες από τις πιο δημοφιλείς βιομηχανίες της εποχής μας, όπως στην αυτοκινητοβιομηχανία, στην αεροπορική βιομηχανία, στην υγεία, στις μεταφορές, στην ενέργεια κ.α.. Αυτά τα συστήματα πολλές φορές χρειάζεται να προσαρμοζονται σε μεταβαλλόμενες προδιαγραφες και σε διαφορετικές λειτουργικές απαιτήσεις.

Τα συστήματα υπολογιστών ακόμη σχεδιάζονται και υλοποιούνται με βάση την ανθρώπινη διαίσθηση και αυτό το γεγονός τα καθιστά ευάλωτα σε λάθη. Αυτό μπορεί να προκαλέσει σημαντικά προβλήματα σε αρκετές οικονομικές και κοινωνικές δραστηριότητες και ακόμα σε κάποιες περιπτώσεις να κοστίσει και ανθρώπινες ζωές. Για παράδειγμα, μια ελαττωματικη σχεδίαση ενός αυτόματου συστήματος οδήγησης ενός αυτοκινήτου μπορεί να προκαλέσει την σύγκρουση του αυτοκινήτου ή ακόμη και την παράσυρση κάποιου πεζού. Πολλές προσπάθειες έχουν γίνει τα τελευταία 30 χρόνια τόσο από την ακαδημαική πλευρά όσο και από την πλευρά της βιομηχανίας για την υιοθέτηση τεχνικών οι οποίες θα επαληθεύσουν αν το σύστημα ικανοποιεί όλες τις προδιαγραφές. Η συνήθης πρακτική είναι η δοκιμή των συστημάτων μέσω συγκεκριμένων τιμών εισόδου και ο έλεγχος αν θα καταλήξουν σε μια κατάσταση αποτυχίας. Αυτή η μέθοδος είναι εκ θεμελίων ανεπαρκής, υπό την έννοια ότι το σύνολο των μονοπατιών εκτέλεσης είναι άπειρο. Τα τελευταία χρόνια, ένα μεγάλο τμήμα της κοινότητας της επιστήμης των υπολογιστών αντιλήφθηκε ότι ο στόχος της απάλειψης των σφαλμάτων των υπολογιστικών συστημάτων μπορεί να επιτευχθεί με τη χρήση τεχνικών επαλήθευσης θεμελιωμένων σε μαθηματικές μεθόδους. Το σύνολο τέτοιων τεχνικών είναι γνωστό ώς *τυπικές μέθοδοι.*

Η πιο υποσχόμενη τυπική μέθοδος των τελευταίων 20 χρόνων είναι ο *έλεγχος μοντέλου* [32]. Στον έλεγχο μοντέλου, ένα υπολογιστικό σύστημα περιγράφεται από ένα σύστημα μεταβάσεων, και μια ιδιότητα διατυπώνεται ως μια φόρμουλα χρονικής λογικής. Δοθέντος ενός συστήματος μεταβάσεων M και μιας ιδιότητας φ, το πρόβλημα του ελέγχου μοντέλου είναι να βρεθεί αν η ιδιότητα ικανοποιείται απο το μοντέλο. Στις μέρες μας, ένας μεγάλος αριθμός δοκιμασμένων αλγορίθμων ελέγχου μοντέλου υπάρχει, τόσο για πιθανοκρατικά όσο και για μη πιθανοκρατικά μοντέλα. Αν μια ιδιότητα δεν ικανοποιείται, κάποιοι αλγόριθμοι επιστρέφουν μια αιτία για την μη επαλήθευση της ιδιότητας η οποία είναι γνωστή ως αντιπαράδειγμα.

Το πρόβλημα της *επιδιόρθωσης μοντέλου* είναι μια επέκταση του προβλήματος του ελέγχου μοντέλου για την περίπτωση στην οποία η ιδιότητα δεν επαληθεύεται. Πιο συγκεκριμένα, ο σκοπός της επιδιόρθωσης μοντέλου είναι να βρεθούν οι ελάχιστες αλλαγές που απαιτούνται στο μοντέλο, ώστε η ιδιότητα φ η οποία παραβιάζεται στο αρχικό μοντέλο να ικανοποιείται.

Η *έκρηξη του χώρου των καταστάσεων* είναι ένας πολύ γνωστός περιορισμός των αυτοματοποιημένων τυπικών μεθόδων, όπως ο έλεγχος μοντέλου και η επιδιόρθωση μοντέλου, η οποία εμποδίζει την εφαρμογή τους σε συστήματα με μεγάλο χώρο καταστάσεων. Το πρόβλημα ενυπάρχει στον έλεγχο μοντέλων και εμποδίζει την εφαρμογη του σε μεγάλα μοντέλα. Το πρόβλημα είναι παρόν σε ακόμα μεγαλύτερο βαθμό στις υπάρχουσες τεχνικές επιδιόρθωσης μοντέλου, οι οποίες στοχεύουν στην απευθείας αλλαγή του μοντέλου που πρόκειται να επιδιορθωθεί. Για παράδειγμα στο πλαίσιο των πιθανοκρατικών συστημάτων, στο [13], οι συγγραφεις μετατρέπουν το πρόβλημα της επιδιόρθωσης σε ένα μη γραμμικό πρόβλημα βελτιστοποίησης κάνοντας χρήση παραμετρικού ελεγχου μοντέλου, και ο χρόνος που χρειάζεται για να υπολογιστεί ένα επιδιορθωμένο μοντέλο αυξάνει ραγδαία σε σχέση με το μέγεθος του χώρου των καταστάσεων.

Ένα σύνολο διαφορετικών τεχνικών έχει αναπτυχθεί για την αντιμετώπιση της έκρηξης του χώρου των καταστάσεων, αλλά η κύρια μέθοδος για τον έλεγχο μοντέλου είναι η χρήση τεχνικών *αφαίρεσης*. Πιο συγκεκριμένα, στην περίπτωση του ελέγχου μοντέλου, η αφαίρεση [34, 73, 55, 40, 53] χρησιμοποιείται για να δημιουργηθεί μία μικρότερη, πιο αφαιρετική έκδοση του αρχικού πραγματικού μοντέλου, και ο έλεγχος μοντέλου πραγματοποιείται σε αυτό το μικρότερο μοντέλο. Για να δουλέψει αυτή η τεχνική, πρέπει να ισχύει ότι αν η ιδιότητα ικανοποιείται στο αφαιρετικό μοντέλο, τότε επίσης ικανοποιείται και στο πραγματικό μοντέλο.

Έχοντας ως κίνητρο την επιτυχία του αφαιρετικού ελέγχου μοντέλου, παρουσιάζω σε αυτή τη διατριβή καινούρια πλαίσια επιδιόρθωσης συστημάτων μεταβάσεων, τα οποία κάνουν χρήση αφαίρεσης για την αντιμετώπιση της έκρηξης του χώρου των καταστάσεων. Στο πρώτο μέρος της διατριβής, επικεντρώνομαι στις δομές Κρίπκε και στην επιδιόρθωσή τους σε σχέση με ιδιότητες οι οποίες εκφράζονται στην Λογική Υπολογιστικού Δέντρου (ΛΥΔ), ενόσω στο δεύτερο μέρος, επικεντρώνομαι στα πιθανοκρατικά μοντέλα στη μορφή των Αλυσίδων Μαρκοφ Διακριτού Χρόνου (ΑΜΔΧ) και στην επιδιόρθωσή τους σε σχέση με ιδιότητες πρόσβασης χρονικής λογικής.

Πιο αναλυτικά, το πλαίσιο επιδιόρθωσης μοντέλου για μη πιθανοκρατικά μοντέλα το οποίο παρουσιάζεται στο πρώτο μέρος της διατριβής, βασίζεται στις δομές Κριπκε, σε μια σημασιολογία τριών τιμών για την ΛΥΔ και στα Τροπικά Συστήματα Μεταβάσεων Κριπκε (ΤΣΜΚ) τα οποία παίζουν το ρόλο του αφαιρετικού μοντέλου. Το πλαίσιο περιλαμβάνει έναν αφαιρετικό αλγόριθμο επιδιόρθωσης ο οποίος εφαρμόζεται πάνων στα ΤΣΜΚ. Αποδεικνύω ότι ο αλγόριθμος είναι ορθός, έχει πολυωνυμική πολυπλοκότητα στο μέγεθος του αφαιρετικού μοντέλου και είναι πλήρης για ένα μεγάλο υποσύνολο της ΛΥΔ. Για να δείξω την πρακτική εφαρμογή του αλγορίθμου, εφήρμοσα το πλαίσιο σε ένα σύστημα πόρτας αυτόματου ανοίγματος και στο πρωτόκολλο συστήματος αρχείων Andrew.

Σχετικά με τα πιθανοκρατικά μοντέλα, παρουσιάζω ένα πλαίσιο βασισμένο στην αφαίρεση για την επιδιόρθωση των ΑΜΔΧ σε σχέση με ιδιότητες πρόσβασης χρονικής λογικής. Στόχος του πλαισίου είναι να αντιμετωπίσουν την έκρηξη του χώρου των καταστάσεων και να προσφέρουν μιά λύση η οποία θα είναι εφαρμόσιμη σε προβλήματα επιδιόρθωσης με πολύ μεγάλο χώρο καταστάσεων. Μετά την αναλυτική περιγραφή του πλαισίου, η διατριβή παρουσιάζει τα οφέλη απόδοσης που προκύπτουν από την συγκεκριμένη προσέγγιση σε σχέση με την απευθείας επιδιόρθωση του πραγματικού μοντέλου, συμπεριλαμβανομένης και της συζήτησης για ειδικές στρατηγικές επιδιόρθωσης βάσει συγκεκριμένων πεδίων εφαρμογής. Η πρακτική χρησιμότητα του πλαισίου επιδεικνύεται με την επιδιόρθωση τεσσάρων διαφορετικών πιθανοκρατικών μοντέλων με ποικιλομορφία στη δομή του χώρου των καταστάσεων.

Contents

Chapter 1

Introduction

1.1 Model Repair

Nowadays, computer systems are incorporated in almost all engineering designs and appear in some of the most mainstream industries of our era, such as automotive, aerospace, healthcare, transportation, energy, etc.. These systems may also need to adjust to modifying specifications and different functional requirements.

Computer systems are still designed and implemented based on human intuition and this makes them vulnerable to errors. This could cause serious problems in several economic and social activities and even cost human lives. For example, a faulty design of a system controlling a drive-by-wire car may lead the car to crash or even kill a pedestrian. Many efforts have been done in the last 30 years from both academia and industry to adopt techniques which will verify if the system fulfill all its specifications. The current practice is to test the systems for specific inputs to check if it will end up with a failure state. This method is fundamentally unsatisfactory, in the sense that all the execution paths of a system are infinite. In recent years, a large part of the computer science community understood that the target for eliminating faults in computer systems can be accomplished by the use of verification techniques formally grounded in mathematical methods. The set of such techniques is known as *formal methods*.

The most promising formal method of the last 20 years is that of *model checking* [32]. In model checking, the system is described by a transition system, and the property is formulated as a temporal logic formula. Given a transition system M and a property φ, the problem of model checking is to find if the property is satisfied by the model. Today, a number of mature model checking algorithms exist, for probabilistic and non probabilistic models. If the property is not satisfied, some algorithms return a cause for the refutation of the property known as counterexample.

The problem of *model repair* is an extension of the model checking problem for the case where the property is refuted. More specifically, the aim of model repair is to find the minimal changes to the model, such that the property φ that is violated in the original model will be satisfied. The problem of model repair for Kripke structures and Computation Tree Logic (CTL) [49] properties was first introduced in [23], while it has been examined in the probabilistic setting for the first time in [13].

In recent years, a number of interesting applications of the model repair problem came in the limelight. Repair of concurrent programs is examined in [8], while the addition of fault tolerance to distributed systems is studied in [76]. Authors in [17] propose a framework for a knowledge-based automated repair of authentication protocols. Another interesting application of the model repair problem is presented in [3], where the objective is the automated fine tuning of probabilistic self-stabilizing algorithms. Finally, a repair application is discussed in [81] and refers to a least-violating control synthesis for autonomous systems.

1.2 Motivation and Contributions

The *state space 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. The problem is inherent in model checking and makes its application infeasible to large models. The problem is actually present to a greatest extent than model checking in existing model repair techniques, which aim to directly change the model under repair. For example, in the context of probabilistic systems, in [13], the authors transform the repair problem to a non-linear optimization problem using parametric model checking, and the time needed for computing a repaired model increases rapidly with respect to the size of the state space.

Different techniques have been developed to cope with the state space explosion problem, but the main method for fighting this problem in model checking is the use of *abstraction* techniques. More specifically, in the case of model checking *abstraction* [34, 73, 55, 40, 53] is used to create a smaller, more abstract version of the initial concrete model, and model checking is performed on this smaller model. For this technique to work as advertised, it should be the case that if a property is satisfied in the abstract model, then it is also satisfied in the concrete model too.

Motivated by the success of abstraction-based model checking, we present in this thesis new frameworks for model repair of transition systems, which use abstraction and refinement to tackle the state space explosion problem.

At the first part of the thesis, we focus on Kripke Structures and their repair with respect to properties expressed in the Computational Tree Logic (CTL), while on the second part, we focus on probabilistic models of Discrete Time Markov Chains (DTMCs) and their repair regarding reachability temporal logic properties.

The major contributions of this thesis related to the non-probabilistic models are:

- The thesis provides 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 [61]. 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.

- The model repair problem is strengthened 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, a metric space is defined over KSs that quantifies the structural differences between them.

- The thesis introduces an Abstract Model Repair algorithm for KMTSs, which takes into account the aforementioned minimality criterion.

- The soundness of the Abstract Model Repair algorithm for the full CTL and the completeness for a major fragment of it are proved. Moreover, the algorithm's complexity is analyzed with respect to the model size of the abstract KMTS, which can be much smaller than the concrete KS.

- The utility of our approach is illustrated through a prototype implementation used to repair a flawed Automatic Door Opener system [11] and the Andrew File System 1 protocol. The experimental results show significant improvement in efficiency compared to a concrete model repair solution.

The main contributions of this thesis regarding the repair of probabilistic models are as follows:

- The thesis introduces a framework for the repair of a DTMC with respect to a (not-nested) Probabilistic Computation Tree Logic (PCTL) reachability formula, using an Abstract Discrete Time Markov Chain (ADTMC) for the given DTMC and the 3-valued semantics of PCTL over ADTMC

  s. Based on a strong preservation theorem, if a PCTL property is refuted or satisfied in the ADTMC (abstract model), t hen the same also holds for the concrete DTMC [64].

- A metric space is defined over the DTMCs with the same state labeling, in order to measure the distance of repaired DTMCs from the original DTMC.

- A Probabilistic Abstract Model Repair (PAMR) algorithm is introduced that transforms the DTMC repair problem to a non-linear minimization problem for the state space of the abstract model, instead of the concrete one. If a solution is found, the repaired DTMC is returned, which corresponds to an approximate (not the optimal) solution; otherwise, the algorithm is iteratively applied to refined ADTMCs until a solution is found. The refinement can be potentially adapted by the analyst, for implementing alternative repair strategies.

- The thesis analyzes the PAMR computational gains and more specifically the achieved reduction in the expensive non-linear optimization and linear equation solving problems, which are involved respectively in the concrete model repair and model checking techniques.

- As a proof of concept, experimental results are provided for the DTMCs of extended versions of the Craps game, the IPv4 Zeroconf protocol, a message authentication protocol and the gambler's ruin model.

1.3 Thesis structure

The thesis is organized as follows. Chapter 2 introduces all the preliminary definitions and background information on which the thesis is grounded. More specifically, Sections 2.1, 2.2 and 2.3 introduce KSs, KMTSs, as well as abstraction and refinement based on a 3-valued semantics for CTL. In Section 2.4, the notion of DTMC is introduced which is the formalism for the concrete model in our probabilistic framework. Section 2.5 discusses how an ADTMC can serve as an abstraction of a DTMC and how a PCTL reachability formula can be verified in an ADTMC for CTL.

In Chapter 3, the abstract model repair framework for Kripke structures is presented. In particular, Section 3.1 defines a metric space for KSs and formally defines the problem of model repair. Section 3.2 presents our framework for Abstract Model Repair, while Section 3.3 introduces the abstract-model-repair algorithm for KMTSs and discusses its soundness, completeness and complexity properties. Section 3.4 presents the experimental evaluation of our method through its application to the Andrew File System 1 protocol (AFS1). Section 3.5 compares our approach with the related work.

The abstract model repair framework for probabilistic systems is presented in Chapter 4. In Section 4.1, the model repair problem for probabilistic systems is formulated together with a metric space for DTMCs. We present the abstract model repair process for probabilistic systems in Section 4.2 together with the basic model repair operations. The PAMR algorithm is described in Section 4.3. The algorithm's steps are illustrated using an application in Section 4.4, where we also elaborate on the method's efficiency gains, its cost in terms of the solution's optimality and its flexibility perspectives. In Section 4.5, we present the experimental results for extended models with progressively larger state spaces of the Craps game, the IPv4 Zeroconf protocol, a message authentication protocol and the gambler's ruin model. The related work is reviewed in Section 4.6.

The conclusions and the future prospects of the thesis are presented in Chapter 5. Namely, Section 5.1 concludes with a review of the overall approach and a summary of contributions, while Section 5.2 pinpoints directions for future work.

Chapter 2

Preliminaries

2.1 Kripke structures

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 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. 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 [11] as a running example throughout the thesis for Kripke structures. The system, given as a KS in Fig 2.1, 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 the thesis, we use a simpler version of the ADO system, given as the KS M in Fig. 2.3a, where the set of atomic propositions is $AP = \{q\}$ and $q \equiv (open = true)$.

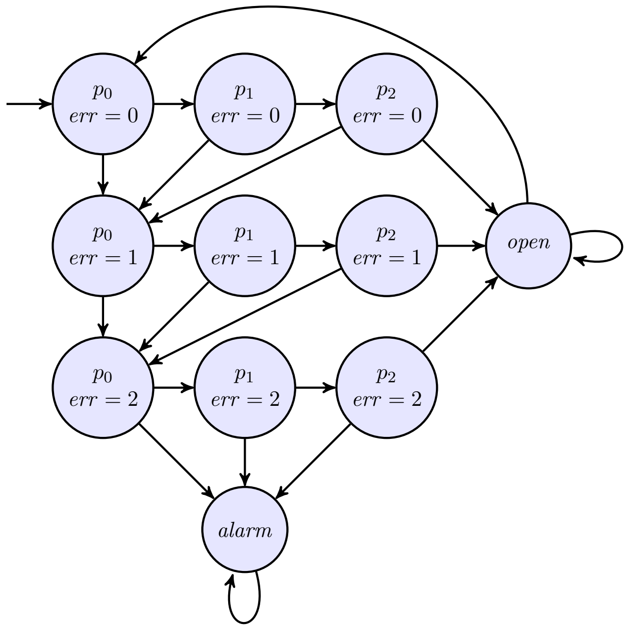


Figure 2.1: The Automatic Door Opener (ADO) System.

2.2 Kripke Modal Transition Systems

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

1. *$\hat{S}$ is a finite set of* states.

2. *$\hat{S}_0 \subseteq \hat{S}$ is the set of* initial states.

3. *$R_{must} \subseteq \hat{S} \times \hat{S}$ and $R_{may} \subseteq \hat{S} \times \hat{S}$ are* transition relations *such that $R_{must} \subseteq R_{may}$.*

4. *$\hat{L} : \hat{S} \to 2^{Lit}$ 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 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* [61] of a CTL formula ϕ at a state $\hat{s}$ of KMTS $\hat{M}$.

Definition 3. [61] *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 $\phi = false$*

  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \mathit{false}$

- *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* $\phi = \phi_1 \wedge \phi_2$

- $[(\hat{M},\hat{s}) \models^3 \phi] = true$, *iff* $[(\hat{M},\hat{s}) \models^3 \phi_1] = true$ *and* $[(\hat{M},\hat{s}) \models^3 \phi_2] = true$.
- $[(\hat{M},\hat{s}) \models^3 \phi] = false$, *iff* $[(\hat{M},\hat{s}) \models^3 \phi_1] = false$ *or* $[(\hat{M},\hat{s}) \models^3 \phi_2] = false$.
- $[(\hat{M},\hat{s}) \models^3 \phi] = \bot$, *otherwise.*

- *If* $\phi = AX\phi_1$
  - $[(\hat{M},\hat{s}) \models^3 \phi] = true$, *iff for all* $\hat{s}_i$ *such that* $(\hat{s},\hat{s}_i) \in R_{may}$, $[(\hat{M},\hat{s}_i) \models^3 \phi_1] = true$.
  - $[(\hat{M},\hat{s}) \models^3 \phi] = false$, *iff there exists some* $\hat{s}_i$ *such that* $(\hat{s},\hat{s}_i) \in R_{must}$ *and* $[(\hat{M},\hat{s}_i) \models^3 \phi_1] = false$.
  - $[(\hat{M},\hat{s}) \models^3 \phi] = \bot$, *otherwise.*

- *If* $\phi = EX\phi_1$
  - $[(\hat{M},\hat{s}) \models^3 \phi] = true$, *iff there exists* $\hat{s}_i$ *such that* $(\hat{s},\hat{s}_i) \in R_{must}$ *and* $[(\hat{M},\hat{s}_i) \models^3 \phi_1] = true$.
  - $[(\hat{M},\hat{s}) \models^3 \phi] = false$, *iff for all* $\hat{s}_i$ *such that* $(\hat{s},\hat{s}_i) \in R_{may}$, $[(\hat{M},\hat{s}_i) \models^3 \phi_1] = false$.
  - $[(\hat{M},\hat{s}) \models^3 \phi] = \bot$, *otherwise.*

- *If* $\phi = AG\phi_1$
  - $[(\hat{M},\hat{s}) \models^3 \phi] = true$, *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* $[(\hat{M},\hat{s}_i) \models^3 \phi_1] = true$.

  - $[(\hat{M}, \hat{s}) \models^3 \phi] = false$, *iff there exists some must-path* $\pi_{must} = [\hat{s}, \hat{s}_1, \hat{s}_2, ...]$, *such that for some* $\hat{s}_i \in \pi_{must}$, $[(\hat{M}, \hat{s}_i) \models^3 \phi_1] = false$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \bot$, *otherwise.*

- *If* $\phi = EG\phi_1$
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = true$, *iff there exists some must-path* $\pi_{must} = [\hat{s}, \hat{s}_1, \hat{s}_2, ...]$, *such that for all* $\hat{s}_i \in \pi_{must}$, $[(\hat{M}, \hat{s}_i) \models^3 \phi_1] = true$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = false$, *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* $[(\hat{M}, \hat{s}_i) \models^3 \phi_1] = false$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \bot$, *otherwise.*

- *If* $\phi = AF\phi_1$
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = true$, *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* $[(\hat{M}, \hat{s}_i) \models^3 \phi_1] = true$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = false$, *iff there exists some must-path* $\pi_{must} = [\hat{s}, \hat{s}_1, \hat{s}_2, ...]$, *such that for all* $\hat{s}_i \in \pi_{must}$, $[(\hat{M}, \hat{s}_i) \models^3 \phi_1] = false$.
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = \bot$, *otherwise.*

- *If* $\phi = EF\phi_1$
  - $[(\hat{M}, \hat{s}) \models^3 \phi] = true$, *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* $[(\hat{M}, \hat{s}_i) \models^3 \phi_1] = true$.

  - $[(\hat{M},\hat{s}) \models^3 \phi] = false$, *iff for all may-paths* $\pi_{may} = [\hat{s},\hat{s}_1,\hat{s}_2,...]$ *and for all* $\hat{s}_i \in \pi_{may}$, $[(\hat{M},\hat{s}_i) \models^3 \phi_1] = false$.
  - $[(\hat{M},\hat{s}) \models^3 \phi] = \perp$, *otherwise.*

- *If* $\phi = A(\phi_1\, U\, \phi_2)$
  - $[(\hat{M},\hat{s}) \models^3 \phi] = true$, *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* $[(\hat{M},\hat{s}_i) \models^3 \phi_2] = true$ *and* $\forall j < i$: $[(\hat{M},\hat{s}_j) \models^3 \phi_1] = true$.
  - $[(\hat{M},\hat{s}) \models^3 \phi] = false$, *iff there exists some must-path* $\pi_{must} = [\hat{s},\hat{s}_1,\hat{s}_2,...]$, *such that*
    i. *for all* $0 \le k < |\pi_{must}|$:
    $(\forall j < k : [(\hat{M},\hat{s}_j) \models^3 \phi_1] \neq false) \Rightarrow ([(\hat{M},\hat{s}_k) \models^3 \phi_2] = false)$
    ii. $($*for all* $0 \le k < |\pi_{must}| : [(\hat{M},\hat{s}_k) \models^3 \phi_2] \neq false) \Rightarrow |\pi_{must}| = \infty$
  - $[(\hat{M},\hat{s}) \models^3 \phi] = \perp$, *otherwise.*

- *If* $\phi = E(\phi_1 U \phi_2)$
  - $[(\hat{M},\hat{s}) \models^3 \phi] = true$, *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* $[(\hat{M},\hat{s}_i) \models^3 \phi_2] = true$ *and for all* $j < i, [(\hat{M},\hat{s}_j) \models^3 \phi_1] = true$.
  - $[(\hat{M},\hat{s}) \models^3 \phi] = false$, *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. $(\text{for all } 0 \leq k < |\pi_{may}| : [(\hat{M}, \hat{s}_k) \models^3 \phi_2] \neq \mathit{false}) \Rightarrow |\pi_{may}| = \infty$

– $[(\hat{M}, \hat{s}) \models^3 \phi] = \perp$, *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.

2.3 Abstraction and Refinement for 3-Valued CTL

2.3.1 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 [34, 53].

Definition 4. *Given a KS $M = (S, S_0, R, L)$ and a pair of total functions $(\alpha : S \rightarrow \hat{S}, \gamma : \hat{S} \rightarrow 2^S)$ 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. 4 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 2.2). 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

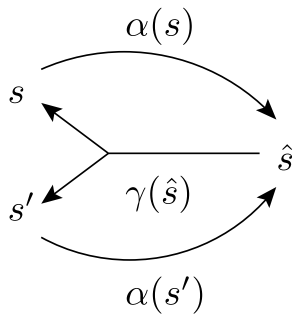


Figure 2.2: Abstraction and Concretization.

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 5. *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. 5 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* [36, 37] framework. Abstract interpretation is a theory for a set of abstraction techniques, for which important properties for the model checking problem have been proved [40, 41].

Definition 6. [40, 54] *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* $(s', \hat{s}') \in H$*.* $\square$

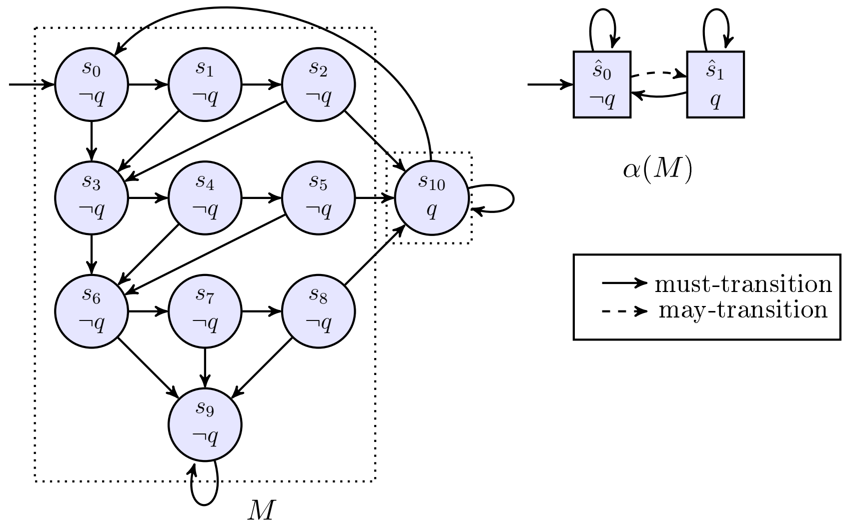
The abstraction function α of Def. 4 is a mixed simulation for the KS M and its abstract KMTS $\alpha(M)$.

Theorem 1. [54] *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*

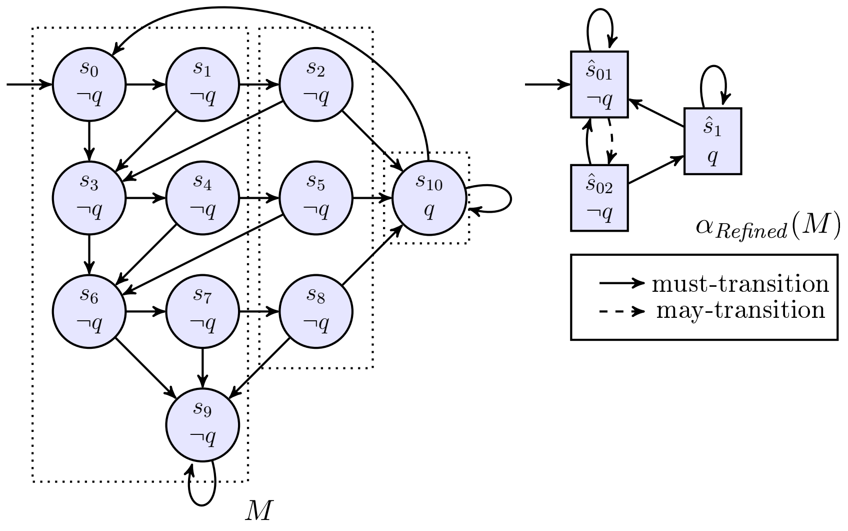
$$[(\hat{M}, \hat{s}) \models \phi] \neq \perp \Rightarrow [(M, s) \models \phi] = [(\hat{M}, \hat{s}) \models \phi]$$

Theorem 1 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 $\perp$ 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. 2.3a, where all the states labeled by q are grouped together, as are all states labeled by $\neg q$.



(a) The KS and initial KMTS.



(b) The KS and refined KMTS.

Figure 2.3: The KS and KMTSs for the ADO system.

2.3.2 Refinement

When the outcome of verifying a CTL formula ϕ on an abstract model using the 3-valued semantics is $\perp$, 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 [33, 35, 38], as well as a number of techniques for the 3-valued CTL model checking [53, 78, 79, 57]. The refinement technique that we adopt is an automated two-step process based on [33, 78]:

1. Identify a *failure state* in $\alpha(M)$ using the algorithms in [33, 78]; 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 $\alpha_{\mathit{Refined}}(M)$, where $\alpha_{\mathit{Refined}}$ is a new abstraction function as in Def. 4, 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 [78], where the authors propose the use of Generalized KMTSs, which ensure monotonicity of the refinement process. GKMTSs introduce a new type of must-transitions which are called *hyper-must transitions* and their refinement process ensures that if a property is satisfied from a GKMTS produced at a refinement step it will also be satisfied in any GKMTS produced at any succeeding refinement step.

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. 2.3a, the absence of a must-transition from $\hat{s}_0$ to $\hat{s}_1$, where $[(\alpha(M), \hat{s}_1) \models q] = \mathit{true}$, in conjunction with the existence of a may-transition from $\hat{s}_0$ to $\hat{s}_1$, i.e. to a state where $[(\alpha(M), \hat{s}_1) \models q] = \mathit{true}$, 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 $\alpha_{\mathit{Refined}}(M)$ it holds that $[\alpha_{\mathit{Refined}}(M), \hat{s}_1) \models \phi] = \mathit{false}$. The initial KS and the refined KMTS $\alpha_{\mathit{Refined}}(M)$ are shown in Fig. 2.3b.

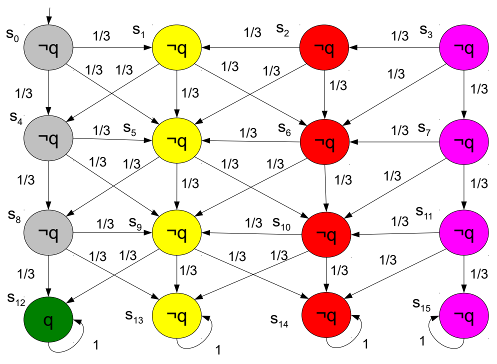


Figure 2.4: A Discrete Time Markov Chain.

2.4 Probabilistic systems

Definition 7. *A (labeled)* Discrete Time Markov Chain *(DTMC) is a 4-tuple* $M = (S, s_{init}, P, L)$*, where:*

1. S *is a finite set of states;*
2. $s_{init} \in S$ *is the initial state;*
3. $P : S \times S \to [0, 1]$ *is a transition probability function with* $\sum_{s' \in S} P(s, s') = 1$ *for all* $s \in S$*;*
4. $L : S \to 2^{Lit}$ *is a state* labeling function *such that* $\forall s \in S$, $\forall p \in AP$, $p \in L(s) \Leftrightarrow \neg p \notin L(s)$.

A DTMC is a transition system with labelled states and probabilities assigned to its transitions.

Example. (ROBOT) We use a robot system as a running example for probabilistic systems. The DTMC for the robot system is shown in Fig. 2.4.

The robot moves among states with different color and tries to reach a state, where the color will be *green*. The robot is "fair", in the sense that for each state the transition probabilities to the directly accessible states are equal $\left(\frac{1}{3}\right)$. The set of atomic propositions for the robot DTMC is $AP = \{q\}$ with $q \equiv (color = green)$. $\square$

Definition 8. *A* path π *in a DTMC* $M = (S, s_{init}, P, L)$ *is an (infinite) sequence* $\pi = [s_0, s_1.s_2, ...]$*; we denote by* $\pi[i]$ *the (i+1)-th state of* π*. A* path fragment $\rho = [s_0, s_1.s_2, ..., s_n]$ *is a finite prefix of a path.*

Definition 9. *Let* $Path_s^M$ *be the set of all paths starting from state* s*. Let* Ω *be the set* $Path^M = \bigcup_{s \in S} Path_s^M$ *and* $Cyl(s_0, ..., s_k)$ *be the (*cylinder) set of states $s_0, ..., s_k$, i.e. the set of all paths in $Path^M$ with prefix $s_0, ..., s_k$. For any such Cyl, a probability measure x is defined such that:

$$x(Cyl(s_0, ..., s_k)) = \prod_{i=0}^{k-1} P(s_i, s_{i+1})$$

The cylinder sets include paths in DTMCs, and their probability measure is used to evaluate the probability of reachability properties, which can be expressed in Probabilistic Computation Tree Logic (PCTL).

Definition 10. *The syntax of the reachability fragment of PCTL over a set of atomic propositions AP is given by the following grammar:*

$$\varphi ::= P_{\bowtie p}[F\psi] \mid P_{\bowtie p}[F^{\leq k}\psi]$$

$$\psi ::= true|q|\neg\psi|\psi \vee \psi|\psi \wedge \psi$$

where $q \in AP$, $p \in [0,1]$, $k \in \mathbb{N}$ *and,* $\bowtie \in \{<, \leq, \geq, >\}$

Def. 10 is focused on reachability properties of PCTL without nested probabilistic quantifiers. The model checking problem for reachability properties is reduced to the solution of a system of equations, as it is shown in Def. 11 for unbounded properties and in Def. 12 for bounded properties .

Definition 11. *Let* $M = (S, s_{init}, P, L)$ *be a DTMC. The probability measure* x_s *of a state* $s \in S$ *to satisfy the property* $F\psi$ *is defined as:*

$$x_s = \begin{cases} 1 & \textit{if } (M,s) \models \psi \\ 0 & \textit{if there is no path from } s \textit{ to a state satisfying } \psi \\ \sum_{t \in S} P(s,t) \cdot x_t & \textit{otherwise} \end{cases}$$

Definition 12. *Let* $M = (S, s_{init}, P, L)$ *be a DTMC. The probability measure* x_s^k *of a state* $s \in S$ *to satisfy the property* $F^{\leq k}\psi$ *with* $k \in \mathbb{N}$ *is defined as:*

$$x_s^k = \begin{cases} 1 & \textit{if } (M,s) \models \psi \\ 0 & \textit{if there is no path from } s \textit{ to a state satisfying } \psi \\ & \textit{or there is a path from } s \textit{ to a state satisfying } \psi \\ & \textit{and } k = 0 \\ \sum_{\mathrm{t \in S}} \mathrm{P(s,t) \cdot x_t^{k-1}} & \textit{otherwise} \end{cases}$$

2.5 Abstract Discrete Time Markov Chains

Definition 13. *Given a transition probability function $P : S \times S \to [0,1]$ defined over a finite set of states S, the* extended *transition probability function $\hat{P} : S \times 2^S \to [0,1]$ is defined, such that $\forall s \in S, S' \subseteq S$, $\hat{P}(s, S') = \sum_{s' \in S'} P(s, s')$.*

Definition 14 ([64])**.** *An* Abstract Discrete Time Markov Chain *(ADTMC) is a 5-tuple $\hat{M} = (\hat{S}, \hat{s}_{init}, P^\ell, P^u, \hat{L})$, where:*

1. *$\hat{S}$ is a finite set of* states*;*

2. *$\hat{s}_{init} \in \hat{S}$ is an initial state;*

3. *$P^\ell : \hat{S} \times \hat{S} \to [0,1]$ and $P^u : \hat{S} \times \hat{S} \to [0,1]$ are two transition probability functions, such that for all $\hat{s} \in \hat{S}$: $\hat{P}^\ell(\hat{s}, \hat{S}) \leq 1 \leq \hat{P}^u(\hat{s}, \hat{S})$ and $P^\ell(\hat{s}, \hat{s}') \leq P^u(\hat{s}, \hat{s}')$ for all $\hat{s}' \in \hat{S}$;*

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 $\neg p$.*

In an ADTMC, the exact transition probability between any two states is not defined explicitly, but instead an upper and a lower bound for this probability is given. We therefore call the P^ℓ and P^u as *lower* and *upper* transition probability functions. Consequently, the verification of a PCTL formula φ over an ADTMC may yield an indefinite answer ($\perp$). The semantics of a reachability PCTL formula φ at a state $\hat{s}$ of an ADTMC $\hat{M}$ is given in Def. 15.

Definition 15 ([64])**.** *Let $\hat{M} = (\hat{S}, \hat{s}_{init}, P^\ell, P^u, \hat{L})$ be a ADTMC. The semantics of a reachability PCTL formula $\varphi = P_{\bowtie p}[F\psi]$ at a state $\hat{s}$ of $\hat{M}$, denoted as $[\![(\hat{M}, \hat{s}) \models^3 \varphi]\!]$, is defined as follows:*

- *If $\varphi = P_{\leq p}[F\psi]$*
  - $[\![(\hat{M}, \hat{s}) \models^3 \varphi]\!] = $ *true, if* $x^u(\{\pi \in Path_{\hat{s}}^{\hat{M}} \mid \exists i \geq 0.\pi[i] \models \psi\}) \leq p$.
  - $[\![(\hat{M}, \hat{s}) \models^3 \varphi]\!] = $ *false, if* $x^\ell(\{\pi \in Path_{\hat{s}}^{\hat{M}} \mid \exists i \geq 0.\pi[i] \models \psi\}) > p$.
  - $[\![(\hat{M}, \hat{s}) \models^3 \varphi]\!] = \perp$, *otherwise.*
- *If $\varphi = P_{\geq p}[F\psi]$*
  - $[\![(\hat{M}, \hat{s}) \models^3 \varphi]\!] = $ *true, if* $x^\ell(\{\pi \in Path_{\hat{s}}^{\hat{M}} \mid \exists i \geq 0.\pi[i] \models \psi\}) \geq p$.
  - $[\![(\hat{M}, \hat{s}) \models^3 \varphi]\!] = $ *false, if* $x^u(\{\pi \in Path_{\hat{s}}^{\hat{M}} \mid \exists i \geq 0.\pi[i] \models \psi\}) < p$.
  - $[\![(\hat{M}, \hat{s}) \models^3 \varphi]\!] = \perp$, *otherwise.*

where x^ℓ, x^u are the probability measures for the lower *and* upper *transition probability functions.* □

From the 3-valued PCTL semantics over ADTMCs, it follows that the truth of $P_{\geq p}[F\psi]$ is checked based on the paths with lower bound probabilities, as opposed to $P_{\leq p}[F\psi]$, for which the paths with the upper bound probabilities are used. For checking the refutation of $P_{\geq p}[F\psi]$, the upper bound probabilities are used, whereas for the refutation of $P_{\leq p}[F\psi]$ the check is based on the lower bound probabilities. The 3-valued semantics for a bounded reachability property in the form $P_{\bowtie p}[F^{\leq k}\psi]$ differ only in that only the finite paths with length k are taken into account.

In Def. 16, given a DTMC M and a pair of total functions α (abstraction) and γ (concretization), an ADTMC $\alpha(M)$ is defined with state space the set $\hat{S}$, derived by abstracting the concrete state space S of M. The ADTMC $\alpha(M)$ serves as the abstract model and the DTMC M as the concrete model in our framework.

Definition 16 ([64])**.** *Given a DTMC $M = (S, s_{init}, P, L)$ 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 ADTMC $\alpha(M) = (\hat{S}, \hat{s}_{init}, P^{\ell}, P^{u}, \hat{L})$ is defined as follows:

1. *$\hat{s}_{init} = \alpha(s_{init})$;*
2. *for all $\hat{s}$, $lit \in \hat{L}(\hat{s})$ only if for all $s \in \gamma(\hat{s})$, $lit \in L(s)$;*
3. *for all $\hat{s}_1, \hat{s}_2$, $P^{\ell}(\hat{s}_1, \hat{s}_2) = \inf_{s\in\gamma(\hat{s}_1)} \hat{P}(s, \gamma(\hat{s}_2))$;*
4. *for all $\hat{s}_1, \hat{s}_2$, $P^{u}(\hat{s}_1, \hat{s}_2) = \min(1, \sup_{s\in\gamma(\hat{s}_1)} \hat{P}(s, \gamma(\hat{s}_2)))$.*

A state $\hat{s}$ of $\alpha(M)$ is an abstract state of some $s \in S$, if and only if $\hat{s} = \alpha(s)$ (equivalently $s \in \gamma(\alpha(s))$). The abstract state $\hat{s}_{init}$ of the concrete state s_{init} of the DTMC is initial in $\alpha(M)$. Proposition $q \in AP$ is true (resp. false) in $\hat{s} = \alpha(s)$, if and only if q is true (resp. false) in all $s \in \gamma(\hat{s})$. Otherwise,

q is unknown ($\perp$) at $\hat{s}$. $P^{\ell}(\hat{s}_1, \hat{s}_2)$ for two abstract $\hat{s}_1$, $\hat{s}_2$ is defined as the infinum of the transition probabilities from each $s_i \in \gamma(\hat{s}_1)$ to all $s_j \in \gamma(\hat{s}_2)$. On the other hand, $P^u(\hat{s}_1, \hat{s}_2)$ for $\hat{s}_1$, $\hat{s}_2$ is defined as the minimum of 1 and the supremum of the transition probabilities from each $s_i \in \gamma(\hat{s}_1)$ to all $s_j \in \gamma(\hat{s}_2)$.

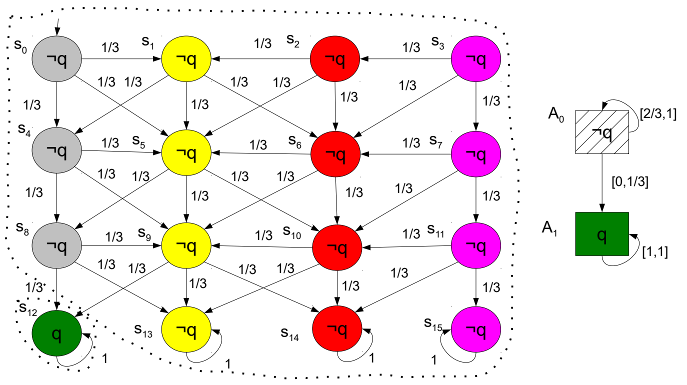


Figure 2.5: The abstraction of a Discrete Time Markov Chain.

Example. (ROBOT) In Fig. 2.5, an abstract DTMC of the robot system is presented where the green state (labeled with q) and the non-green states (labeled with $\neg q$) are abstracted using one abstract state for each case. The lower bound of the outgoing transition from the non-green abstract state A_0 to A_1 is 0, because there are no concrete non-green states with outgoing transition of probability greater than 0 to the concrete green state. The upper bound for the same transition is $\left(\frac{1}{3}\right)$, because the greatest transition probability from a concrete non-green state to the concrete green state is $\left(\frac{1}{3}\right)$. All other lower and upper bound probabilities in Fig. 2.5 are calculated in the same way. $\square$

Theorem 2 ([64]). *Let $M = (S, s_{init}, P, L)$ a DTMC and $\alpha(M) = (\hat{S}, \hat{s}_{init}$ P^{ℓ}, $P^{u}, \hat{L})$ the ADTMC as in Def. 16. Then for every PCTL formula φ and every $s \in S, \hat{s} \in \hat{S}$ with $\hat{s} = \alpha(s)$ we have:*

$$[\![(\alpha(M), \hat{s}) \models \varphi]\!] \neq \perp \Rightarrow [\![(M, s) \models \varphi]\!] = [\![(\alpha(M), \hat{s}) \models \varphi]\!]$$

From Theorem 2, it follows that if a PCTL formula is satisfied (or violated) in the ADTMC, then it is also satisfied (or violated) in the DTMC. If the answer of model checking a PCTL formula over the ADTMC is unknown, then there is no definite conclusion for the same formula over the DTMC. To be concise, in the rest of the thesis we will denote $[\![(\alpha(M), \hat{s}) \models \varphi]\!] = \mathit{false}$ by just writing $(\alpha(M), \hat{s}) \not\models \varphi$.

Chapter 3

Abstract Model Repair of Kripke structures

3.1 The Model Repair Problem for Kripke Structures

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 17. *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 3. *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 18. *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:*

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

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

Proof. The proof is done in the same way as in Prop. 3. □

Definition 19. *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. 17 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.

3.2 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.

3.2.1 The Abstract Model Repair Process

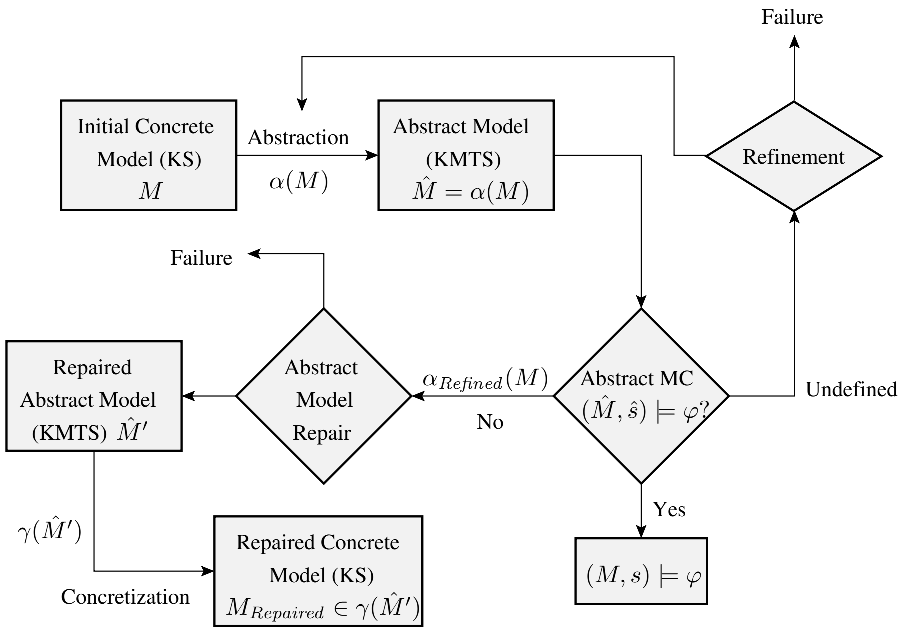

Failure
Initial Concrete Model (KS) M
Abstraction
$\alpha(M)$
Abstract Model (KMTS) $\hat{M} = \alpha(M)$
Refinement
Failure
Repaired Abstract Model (KMTS) $\hat{M}'$
Abstract Model Repair
$\alpha_{Refined}(M)$
No
Abstract MC $(\hat{M}, \hat{s}) \models \varphi$?
Undefined
Yes
$\gamma(\hat{M}')$
Repaired Concrete Model (KS) $M_{Repaired} \in \gamma(\hat{M}')$
Concretization
$(M, s) \models \varphi$

Figure 3.1: Abstract Model Repair Framework.

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

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 1, $(M, s) \models \phi$ 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 1, $(M, s) \not\models \phi$ 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 $\gamma(\hat{M}')$.

3.2.2 Basic Repair Operations

We decompose the ADTMC repair process into two 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

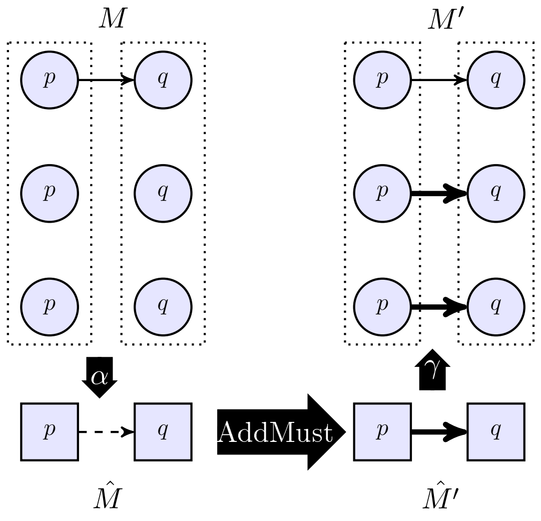
AddState Adding a new KMTS state

RemoveState Removing a disconnected KMTS state

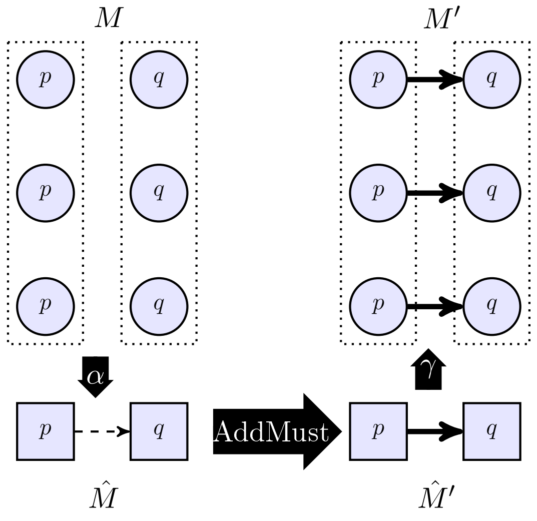
Adding a must-transition

Definition 20 (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. 3.2 shows how the basic repair operation *AddMust* modifies a given KMTS. The newly added transitions are in bold.



(a) May-transition exists



(b) May-transition does not exist

Figure 3.2: *AddMust*: Adding a new must-transition

Proposition 5. *For any* $\hat{M}' = AddMust(\hat{M}, \hat{r}_n)$*, it holds that* $\hat{d}(\hat{M}, \hat{M}') = 1$. □

Definition 21. *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. 4. 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:

$$K_{min} = \{M' \mid M' = (S, S_0, R \cup R_n, L)\} \tag{3.1}$$

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

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

□

Def. 21 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 6. *For all $M' \in K_{min}$, it holds that $1 \leq d(M, M') \leq |S|$.*

Proof. Recall that

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

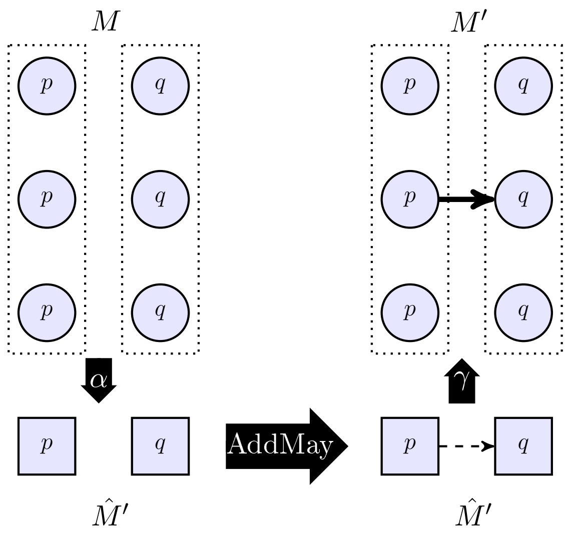
Since $|S \Delta S'| = 0$ and $|\pi(L \upharpoonright_{S \cap S'}) \Delta \pi(L' \upharpoonright_{S \cap S'})| = 0$, $d(M, M') = |R \Delta R'| = |R - R'| + |R' - R| = 0 + |R_n|$. Since $|R_n| \geq 1$ and $|R_n| \leq |S|$, it is proved that $1 \leq d(M, M') \leq |S|$. $\square$

From Prop. 6, we conclude that a lower and upper bound exists for the distance between M and any $M' \in K_{min}$.

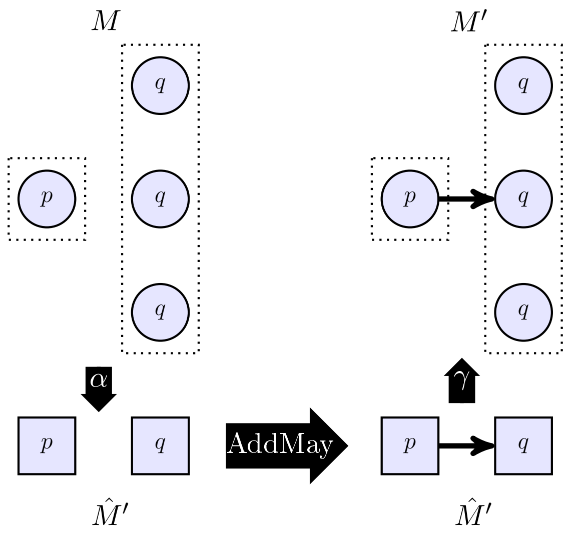
Adding a may-transition

Definition 22 (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* $R'_{may} = R_{may} \cup \{\hat{r}_n\}$*.* $\square$

From Def. 22, 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. 3.3 displays the two different cases of applying basic repair operation *AddMay* to a KMTS.



(a) Only may-transition is added



(b) Must-transition is also added

Figure 3.3: *AddMay*: Adding a new must-transition

Proposition 7. *For any $\hat{M}' = AddMay(\hat{M}, \hat{r}_n)$, it holds that $\hat{d}(\hat{M}, \hat{M}') =$* 1. $\square$

Definition 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. 4. 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:*

$$K_{min} = \{M' \mid M' = (S, S_0, R \cup \{r_n\}, L)\} \tag{3.2}$$

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

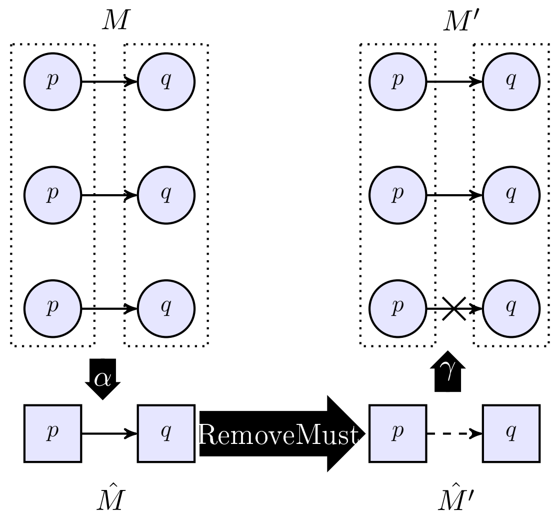
Proposition 8. *For all $M' \in K_{min}$, it holds that $d(M, M') = 1$.*

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 $d(M, M') = 1$. □

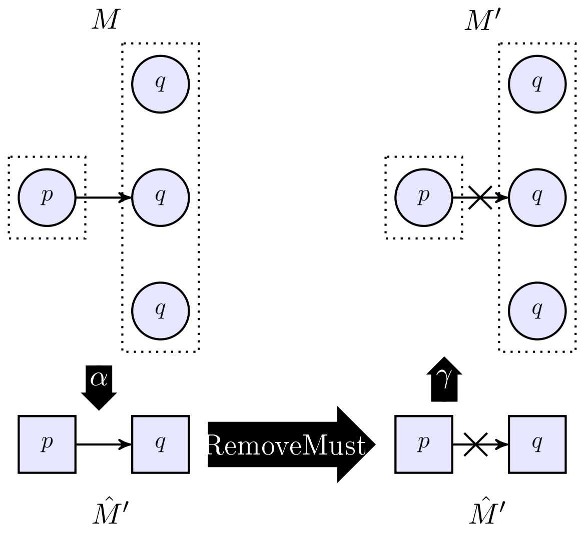
Removing a must-transition

Definition 24 (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. 3.4.



(a) May-transition is not removed



(b) May-transition is also removed

Figure 3.4: *RemoveMust*: Removing an existing must-transition

Proposition 9. *For any $\hat{M}' = RemoveMust(\hat{M}, \hat{r}_m)$, it holds that $\hat{d}(\hat{M}, \hat{M}')$ $= 1$.* □

Definition 25. *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. 4. 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:*

$$K_{min} = \{M' \mid M' = (S, S_0, R - \{R_m\}, L)\} \tag{3.3}$$

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

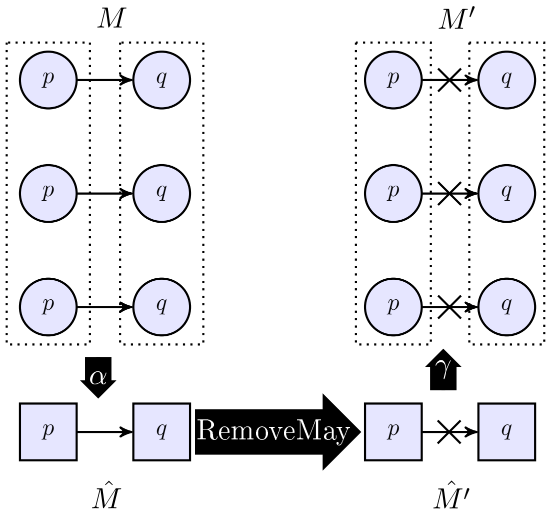
Proof. $d(M, M') = |S \Delta S'| + |R \Delta R'| + \frac{|\pi(L \restriction_{S \cap S'}) \Delta \pi(L' \restriction_{S \cap S'})|}{2}$. Because $|S \Delta S'| = 0$ and $|\pi(L \restriction_{S \cap S'}) \Delta \pi(L' \restriction_{S \cap S'})| = 0$, $d(M, M') = |R \Delta R'| = |R - R'| + |R' - R| = |R_m|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

Removing a may-transition

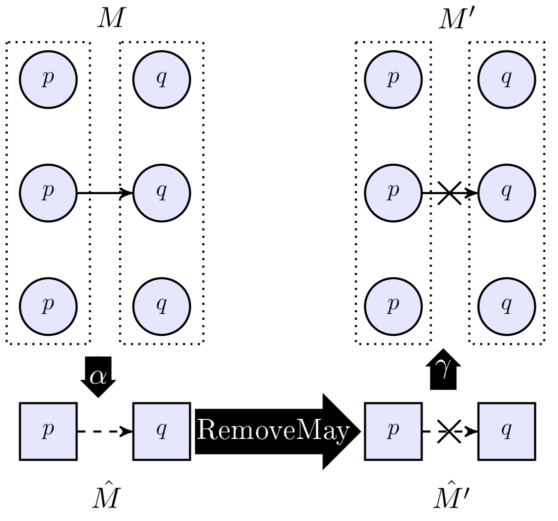
Definition 26 (RemoveMay)**.** *For a given KMTS $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$ and $\hat{r}_m = (\hat{s}_1, \hat{s}_2) \in R_{may}$, $RemoveMay(\hat{M}, \hat{r}_m)$ is the KMTS $\hat{M}' =$*

$(\hat{S}, \hat{S}_0, R'_{must}, R'_{may}, \hat{L})$ *such that* $R'_{must} = R_{must} - \{\hat{r}_m\}$ *and* $R'_{may} = R_{may} - \{\hat{r}_m\}$. □

Def. 26 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. 3.5 shows how function *RemoveMay* works in both cases.



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



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

Figure 3.5: *RemoveMay*: Removing an existing may-transition

Proposition 11. *For any $\hat{M}' = RemoveMay(\hat{M}, \hat{r}_m)$, it holds that $\hat{d}(\hat{M}, \hat{M}') = 1$.* □

Definition 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. 4. 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:

$$M' = (S, S_0, R - R_m, L\} \tag{3.4}$$

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

Proposition 12. *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 $1 \leq d(M, M') \leq |S|^2$. □

Changing the labeling of a KMTS state

Definition 28 (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. 3.6 presents the application of *ChangeLabel* in a graphical manner.

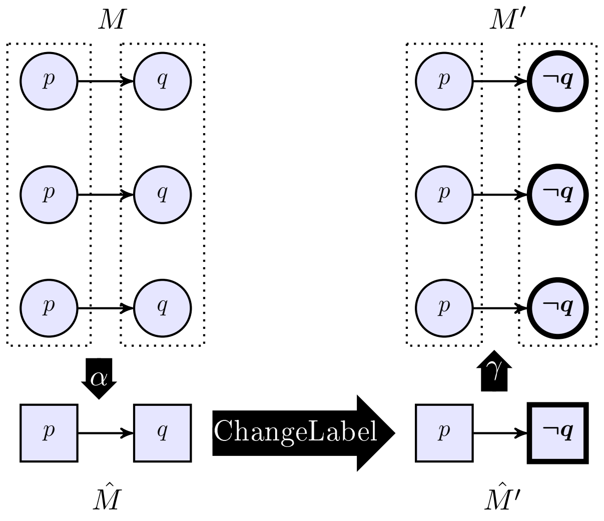


Figure 3.6: *ChangeLabel*: Changing the labeling of a KMTS state

Proposition 13. *For any $\hat{M}' = ChangeLabel(\hat{M}, \hat{s}, \phi)$, it holds that $\hat{d}(\hat{M}, \hat{M}') = 1$.* □

Definition 29. *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. 4. 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:*

$$M' = (S, S_0, R, L - L_{old} \cup L_{new}\} \tag{3.5}$$

where

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

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

$\square$

Proposition 14. *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 $1 \leq d(M, M') \leq |S|$. $\square$

Adding a new KMTS state

Definition 30 (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 $\hat{l}_n = (\hat{s}_n, \perp)$.* $\square$

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. 30 is the fact that the inserted stated is not permitted to be initial. Application of function *AddState* is presented graphically in Fig. 3.7.

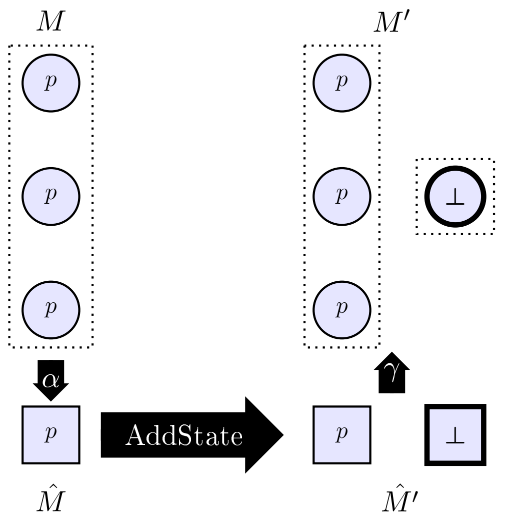


Figure 3.7: *AddState*: Adding a new KMTS state

Proposition 15. *For any $\hat{M}' = AddState(\hat{M}, \hat{s}_n)$, it holds that $\hat{d}(\hat{M}, \hat{M}') =$ 1.* □

Definition 31. *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. 4. 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:*

$$M' = (S \cup \{s_n\}, S_0, R, L \cup \{l_n\}) \tag{3.6}$$

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

Proposition 16. *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 $d(M, M') = 1$. $\square$

Removing a disconnected KMTS state

Definition 32 (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 $\hat{l}_r = (\hat{s}_r, lit) \in \hat{L}$.* $\square$

From Def. 32, 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. 3.8 presents the application of *RemoveState* in a graphical manner.

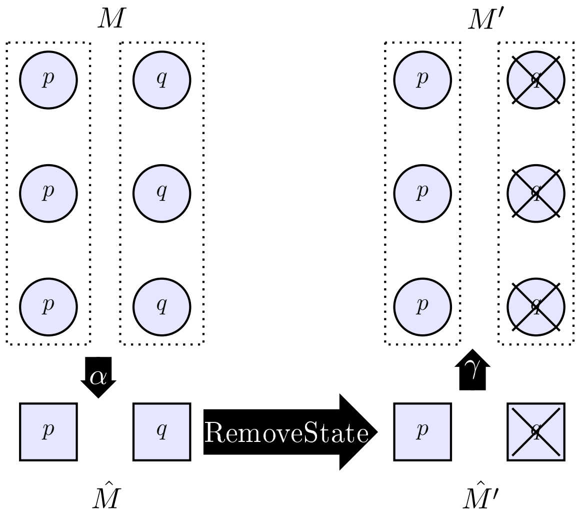


Figure 3.8: *RemoveState*: Removing a disconnected KMTS state

Proposition 17. *For any $\hat{M}' = RemoveState(\hat{M}, \hat{s}_r)$, it holds that $\hat{d}(\hat{M}, \hat{M}') = 1$.* □

Definition 33. *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. 4. 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:*

$$M' = (S', S'_0, R', L') \text{ s.t. } S' = S - S_r, S'_0 = S_0 - S_r, R' = R, L' = L - L_r \quad (3.7)$$

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

Proposition 18. *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 $1 \leq d(M, M') \leq |S|$. □

Minimality Of Changes Ordering For Basic Repair Operations

The distance metric d of Def. 17 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. 6 and all the respective results for the other basic repair operations, we introduce the partial ordering shown in Fig. 3.9. 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.

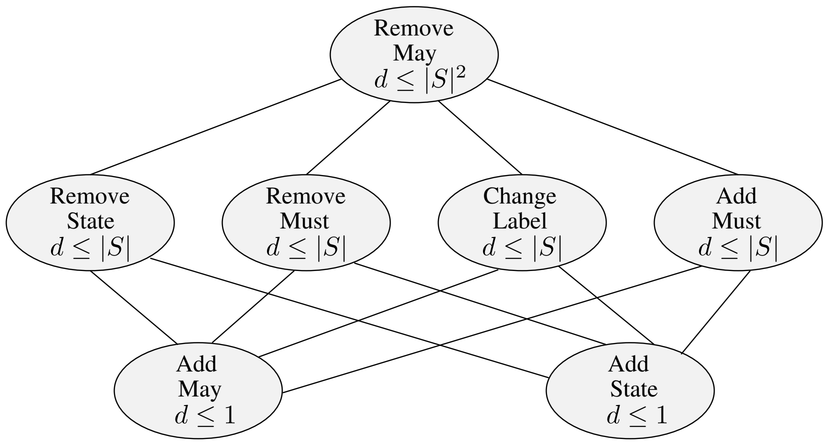


Figure 3.9: Minimality of changes ordering of the set of basic repair operations

3.3 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 [62] and a number of model repair solutions applied to concrete KSs [89, 26]. 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

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 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 $\phi$ 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 $\phi$ is $\phi_1 \wedge \phi_2$ then
6:      return $AbstractRepair_{AND}(\hat{M}, \hat{s}, \phi, C)$
7:  else if $\phi$ is $\phi_1 \vee \phi_2$ then
8:      return $AbstractRepair_{OR}(\hat{M}, \hat{s}, \phi, C)$
9:  else if $\phi$ is $OPER\phi_1$ then
10:     return $AbstractRepair_{OPER}(\hat{M}, \hat{s}, \phi, C)$
11:     where $OPER \in \{AX, EX, AU, EU, AF, EF, AG, EG\}$
```

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

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 $\phi$ where $\phi$
    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 $\phi_{c_i}$ 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
```

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 3 $AbstractRepair_{AND}$

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.

```
 1: $RET_1 := AbstractRepair(\hat{M}, \hat{s}, \phi_1, C)$
 2: $RET_2 := AbstractRepair(\hat{M}, \hat{s}, \phi_2, C)$
 3: $C_1 := C \cup \{(\hat{s}, \phi_1)\}$, $C_2 := C \cup \{(\hat{s}, \phi_2)\}$
 4: $RET'_1 := FAIURE$, $RET'_2 := FAIURE$
 5: if $RET_1 \neq FAILURE$ then
 6:    $\hat{M}_1 := RET_1$
 7:    $RET'_1 := AbstractRepair(\hat{M}_1, \hat{s}, \phi_2, C_1)$
 8:    if $RET'_1 \neq FAILURE$ then
 9:       $\hat{M}'_1 := RET'_1$
10: if $RET_2 \neq FAILURE$ then
11:    $\hat{M}_2 := RET_2$
12:    $RET'_2 := AbstractRepair(\hat{M}_2, \hat{s}, \phi_1, C_2)$
13:    if $RET'_2 \neq FAILURE$ then
14:       $\hat{M}'_2 := RET'_2$
15: if $RET'_1 \neq FAILURE$ && $RET'_2 \neq FAILURE$ then
16:    $\hat{M}' := MinimallyChanged(\hat{M}, \hat{M}'_1, \hat{M}'_2)$
17: else if $RET'_1 \neq FAILURE$ then
18:    $\hat{M}' := RET'_1$
19: else if $RET'_2 \neq FAILURE$ then
20:    $\hat{M}' := RET'_2$
21: else
22:    return FAILURE
23: return $\hat{M}'$
```

Algorithm 4 $AbstractRepair_{AG}$

```
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 $\phi_{c_i}$ 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 $(\hat{M}, \hat{s}) \not\models \phi_1$ then
2:   $RET := AbstractRepair(\hat{M}, \hat{s}, \phi_1, C)$
3:   if $RET == FAILURE$ then
4:     return FAILURE
5:   else
6:     $\hat{M}' := RET$
7: else
8:   $\hat{M}' := \hat{M}$
9: 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
10:   $RET := AbstractRepair(\hat{M}', \hat{s}_k, \phi_1, C)$
11:   if $RET == FAILURE$ then
12:     return FAILURE
13:   else
14:     $\hat{M}' := RET$
15: if $\hat{M}' \models C$ then
16:   return $\hat{M}'$
17: return FAILURE
```

3.3.1 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 5), while conjunctive formulas are handled by the algorithm with the use of constraints (Algorithm 3).

Algorithm 5 $AbstractRepair_{OR}$

```
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 $\phi_{c_i}$ 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 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.

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.

```
1: if there exists $\hat{s}_1 \in \hat{S}$ such that $(\hat{M}, \hat{s}_1) \models \phi_1$ then
2:    for all $\hat{s}_i \in \hat{S}$ such that $(\hat{M}, \hat{s}_i) \models \phi_1$ do
3:       $\hat{r}_i := (\hat{s}, \hat{s}_i)$, $\hat{M}' := AddMust(\hat{M}, \hat{r}_i)$
4:       if $\hat{M}' \models C$ then
5:          return $\hat{M}'$
6: else
7:    for all direct must-reachable states $\hat{s}_i$ from $\hat{s}$ such that $(\hat{M}, \hat{s}_i) \not\models \phi_1$
      do
8:       $RET := AbstractRepair(\hat{M}, \hat{s}_i, \phi_1, C)$
9:       if $RET \neq FAILURE$ then
10:         $\hat{M}' := RET$
11:         return $\hat{M}'$
12:   $\hat{M}' := AddState(\hat{M}, \hat{s}_n)$, $\hat{r}_n := (\hat{s}, \hat{s}_n)$, $\hat{M}' := AddMust(\hat{M}', \hat{r}_n)$
13:   $\hat{r}_n := (\hat{s}_n, \hat{s}_n)$
14:   $\hat{M}' := AddMay(\hat{M}', \hat{r}_n)$
15:   $RET := AbstractRepair(\hat{M}', \hat{s}_n, \phi_1, C)$
16:   if $RET \neq FAILURE$ then
17:      $\hat{M}' := RET$
18:      return $\hat{M}'$
19: return FAILURE
```

$AbstractRepair_{EX}$ presented in Algorithm 6 is the primitive function for handling properties of the form $EX\phi_1$ for some state $\hat{s}$. At first, the KMTS is attempted to be repaired by $AbstractRepair_{EX}$ 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 (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 *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 $\phi_{c_i}$ 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: $RET := FAILURE$
3: for all direct may-reachable states $\hat{s}_i$ from $\hat{s}$ with $(\hat{s}, \hat{s}_i) \in R_{may}$ do
4:   if $(\hat{M}', \hat{s}_i) \not\models \phi_1$ then
5:     $RET := AbstractRepair(\hat{M}', \hat{s}_i, \phi_1, C)$
6:     if $RET == FAILURE$ then
7:       BREAK
8:     $\hat{M}' := RET$
9: if $RET \neq FAILURE$ then
10:   return $\hat{M}'$
11: $\hat{M}' := \hat{M}$
12: 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
13:   if $(\hat{M}', \hat{s}_i) \not\models \phi_1$ then
14:     $\hat{M}' := RemoveMay(\hat{M}', \hat{r}_i)$
15: if there exists direct may-reachable state $\hat{s}_1$ from $\hat{s}$ such that $(\hat{s}, \hat{s}_1) \in R_{may}$ then
16:   if $\hat{M}' \models C$ then
17:     return $\hat{M}'$
18: else
19:   for all $\hat{s}_j \in \hat{S}$ such that $(\hat{M}', \hat{s}_j) \models \phi_1$ do
20:     $\hat{r}_j := (\hat{s}, \hat{s}_j)$, $\hat{M}' := AddMay(\hat{M}', \hat{r}_j)$
21:     if $\hat{M}' \models C$ then
22:       return $\hat{M}'$
23:   $\hat{M}' := AddState(\hat{M}, \hat{s}_n)$
24:   if $\hat{s}_n$ is a dead-end state then
25:     $\hat{r}_n := (\hat{s}_n, \hat{s}_n)$, $\hat{M}' := AddMay(\hat{M}', \hat{r}_n)$
26:   $RET := AbstractRepair(\hat{M}', \hat{s}_n, \phi_1, C)$
27:   if $RET \neq FAILURE$ then
28:     $\hat{M}' := RET$, $\hat{r}_n := (\hat{s}, \hat{s}_n)$, $\hat{M}' := AddMay(\hat{M}', \hat{r}_n)$
29:     if $\hat{M}' \models C$ then
30:       return $\hat{M}'$
31: return FAILURE
```

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.

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.

```
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:    $\hat{M}_1 := RET$
7: 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
8:    $\hat{r}_1 := (\hat{s}, \hat{s}_1)$, $\hat{M}' := AddMust(\hat{M}_1, \hat{r}_1)$
9:    if $\hat{M}' \models C$ then
10:      return $\hat{M}'$
11: 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
12:    $\hat{M}' := \hat{M}_1$
13:    for all $\hat{s}_i \in \pi_{must}$ do
14:       if $(\hat{M}_1, \hat{s}_i) \not\models \phi_1$ then
15:          $RET := AbstractRepair(\hat{M}', \hat{s}_i, \phi_1, C)$
16:          if $RET \neq FAILURE$ then
17:             $\hat{M}' := RET$
18:          else
19:             continue to next path
20:    return $\hat{M}'$
21: $\hat{M}' := AddState(\hat{M}_1, \hat{s}_n)$
22: $RET := AbstractRepair(\hat{M}', \hat{s}_n, \phi_1, C)$
23: if $RET \neq FAILURE$ then
24:    $\hat{M}' := RET$
25:    $\hat{r}_n := (\hat{s}, \hat{s}_n)$, $\hat{M}' := AddMust(\hat{M}', \hat{r}_n)$
26:    if $\hat{s}_n$ is a dead-end state then
27:       $\hat{r}_n := (\hat{s}_n, \hat{s}_n)$, $\hat{M}' := AddMust(\hat{M}', \hat{r}_n)$
28:    if $\hat{M}' \models C$ then
29:       return $\hat{M}'$
30: return FAILURE
```

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 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.

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 $\phi_{c_i}$ 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 $\hat{M}'$
```

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.

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
```

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.

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 $\hat{M}'$
```

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 $\hat{r}_m := (\hat{s}_m, \hat{s}_n) \in R_{may}$ 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.

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
```

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 $\pi_{must} := [\hat{s}_1, ..., \hat{s}_m]$ 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.

3.3.2 Properties of the Algorithm

AbstractRepair is *well-defined* [82], 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 [25, 89] that entail nondeterministic behavior.

Soundness

Lemma 19. *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 $\mathit{AbstractRepair}(\hat{M}, \hat{s}, \phi, C)$ returns FAILURE at line 2 of Algorithm 1 and the lemma is also trivially true.
- if $\phi = p \in AP$, $\mathit{AbstractRepair}_{ATOMIC}(\hat{M}, \hat{s}, p, C)$ is called at line 4 of Algorithm 1 and an $\hat{M}' = \mathit{ChangeLabel}(\hat{M}, \hat{s}, p)$ 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 $\mathit{AbstractRepair}(\hat{M}, \hat{s}, \phi_1, C)$ returns a KMTS $\hat{M}'$, then $(\hat{M}', \hat{s}) \models \phi_1$ and $\hat{M}' \models C$.

Inductive Step:

- if $\phi = \phi_1 \vee \phi_2$, then $\mathit{AbstractRepair}(\hat{M}, \hat{s}, \phi, C)$ calls $\mathit{AbstractRepair}_{OR}$ $(\hat{M}, \hat{s}, \phi_1 \vee \phi_2, C)$ at line 8 of Algorithm 1. From the induction hypothesis, if a KMTS $\hat{M}_1$ is returned by $\mathit{AbstractRepair}(\hat{M}, \hat{s}, \phi_1, C)$ at line 1 of Algorithm 5 and a KMTS $\hat{M}_2$ is returned by $\mathit{AbstractRepair}(\hat{M}, \hat{s}, \phi_2$ $, C)$ respectively, then $(\hat{M}_1, \hat{s}) \models \phi_1$, $\hat{M}_1 \models C$ and $(\hat{M}_2, \hat{s}) \models \phi_1$, $\hat{M}_2 \models C$. $\mathit{AbstractRepair}_{OR}(\hat{M}, \hat{s}, \phi_1 \vee \phi_2, C)$ 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 $\mathit{AbstractRepair}_{AND}$ $(\hat{M}, \hat{s}, \phi_1 \wedge \phi_2, C)$ is called from $\mathit{AbstractRepair}(\hat{M}, \hat{s}, \phi, C)$ at line 6 of Algorithm 1. From the induction hypothesis, if at line 1 of Algorithm 3 $\mathit{AbstractRepair}(\hat{M}, \hat{s}, \phi_1, C)$ 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 $\mathit{AbstractRepair}(\hat{M}_1, \hat{s}, \phi_2, C_1)$ 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 3 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 $C_2 = C \cup (\hat{s}, \phi_2)$.

  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 $\phi = EX\phi_1$, $\mathit{AbstractRepair}_{EX}(\hat{M}, \hat{s}, EX\phi_1, C)$ is called 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 $\hat{M}' = \mathit{AddMust}(\hat{M}, (\hat{s}, \hat{s}_1))$ 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 $\hat{M}' = \mathit{AbstractRepair}(\hat{M}, \hat{s}_1, \phi_1, C)$. 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 $\hat{M}' = \mathit{AbstractRepair}(\mathit{AddMust}(\hat{M}, (\hat{s}, \hat{s}_n)), \hat{s}_n, \phi_1, C)$. 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 $\phi = AG\phi_1$, $\mathit{AbstractRepair}(\hat{M}, \hat{s}, \phi, C)$ calls $\mathit{AbstractRepair}_{AG}(\hat{M}, \hat{s}, AG\phi_1, C)$ at line 10 of Algorithm 1. If $(\hat{M}, \hat{s}) \not\models \phi_1$ and *Abstract Repair*$(\hat{M}, \hat{s}, \phi_1, C)$ returns a KMTS $\hat{M}_0$ at line 2 of Algorithm 4, 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 4 returns a $\hat{M}'$ at line 16, then $\hat{M}' \models C$ and $\hat{M}'$ is the result of successive $\mathit{AbstractRepair}(\hat{M}_i, \hat{s}_k, \phi_1, C)$ calls with $\hat{M}_i = \mathit{AbstractRepair}(\hat{M}_{i-1}, \hat{s}_k, \phi_1, C)$ 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, $(\hat{M}', \hat{s}_k) \models \phi_1$ 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. □

Corrolary 1 (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 19 in the inductive step for $\phi_1 \wedge \phi_2$.

Base Case:

- if $\phi = \top$, Theorem 1 is trivially true, because $(\hat{M}, \hat{s}) \models \phi$.
- if $\phi = \bot$, then $\mathit{AbstractRepair}(\hat{M}, \hat{s}, \bot, \emptyset)$ returns FAILURE at line 2 of Algorithm 1 and the theorem is also trivially true.
- if $\phi = p \in AP$, $\mathit{AbstractRepair}_{ATOMIC}(\hat{M}, \hat{s}, p, \emptyset)$ is called at line 4 of Algorithm 1 and an $\hat{M}' = \mathit{ChangeLabel}(\hat{M}, \hat{s}, p)$ 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 $\mathit{AbstractRepair}(\hat{M}, \hat{s}, \phi, \emptyset)$ returns a KMTS $\hat{M}'$, then $(\hat{M}', \hat{s}) \models \phi_1$.

Inductive Step:

- if $\phi = \phi_1 \vee \phi_2$, then $\mathit{AbstractRepair}(\hat{M}, \hat{s}, \phi, \emptyset)$ calls $\mathit{AbstractRepair}_{OR}$ $(\hat{M}, \hat{s}, \phi_1 \vee \phi_2, \emptyset)$ 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 5 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*$_{AND}(\hat{M}, \hat{s}, \phi_1 \wedge \phi_2, \emptyset)$ is called from *AbstractRepair*$(\hat{M}, \hat{s}, \phi, \emptyset)$ at line 6 of Algorithm 1. From the induction hypothesis, if at line 1 of Algorithm 3 *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 19 $(\hat{M}_1', \hat{s}) \models \phi_2$ and $\hat{M}_1' \models C_1$.

  Likewise, if the calls at lines 2 and 12 of Algorithm 3 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 19 $(\hat{M}_2', \hat{s}) \models \phi_1$, $\hat{M}_2' \models C_2$ 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 $\hat{M}' = \mathit{AddMust}(\hat{M}, (\hat{s}, \hat{s}_1))$. 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 $\hat{M}' = \mathit{Abstract}$ $\mathit{Repair}(\hat{M}, \hat{s}_1, \phi_1, \emptyset)$. 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 $\hat{M}' = \mathit{AbstractRepair}(\mathit{AddMust}(\hat{M}, (\hat{s}, \hat{s}_n)), \hat{s}_n, \phi_1, \emptyset)$. 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 $\phi = AG\phi_1$, $\mathit{AbstractRepair}(\hat{M}, \hat{s}, \phi, \emptyset)$ calls $\mathit{AbstractRepair}_{AG}(\hat{M}, \hat{s}, AG\phi_1, \emptyset)$ at line 10 of Algorithm 1. If $(\hat{M}, \hat{s}) \not\models \phi_1$ and $\mathit{AbstractRepair}$ $(\hat{M}, \hat{s}, \phi_1, \emptyset)$ returns a KMTS $\hat{M}_0$ at line 2 of Algorithm 4, 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 4 returns a $\hat{M}'$ at line 16, this KMTS is the result of successive calls of $\mathit{AbstractRepair}(\hat{M}_i, \hat{s}_k, \phi_1, \emptyset)$ with $\hat{M}_i = \mathit{AbstractRepair}$ $(\hat{M}_{i-1}, \hat{s}_k, \phi_1, \emptyset)$ and $i = 1, \ldots,$ 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 1 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.

Semi-completeness

Definition 34 (*mr*-CTL)**.** *Given a set AP of atomic propositions, we define the syntax of a CTL fragment inductively via a Backus Naur Form:*

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

where p ranges over AP.

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

Theorem 20 (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 20 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 $\mathit{AbstractRepair}_{ATOMIC}(\hat{M}, \hat{s}, p, \emptyset)$ 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 $(\hat{M}'', \hat{s}) \models \phi_1$, $\mathit{AbstractRepair}(\hat{M}, \hat{s}, \phi_1, \emptyset)$ 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 $(\hat{M}''_1, \hat{s}) \models \phi_1$, $\mathit{AbstractRepair}(\hat{M}, \hat{s}, \phi_1, \emptyset)$ at line 1 of Algorithm 5 returns a KMTS $\hat{M}'_1$ such that $(\hat{M}'_1, \hat{s}) \models \phi_1$. Respectively, *Abstract*

Repair$(\hat{M}, \hat{s}, \phi_2, \emptyset)$ at line 2 of Algorithm 5 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 5 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 4 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 $\hat{M}_i =$ *Abstract Repair*$(\hat{M}_{i-1}, \hat{s}_k, \phi_1, \emptyset)$, $i = 1, \ldots$, 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 20 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.

3.3.3 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 [19], 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 $O(|\phi| \cdot (|\hat{S}| + |R_{may}|))$ [53], (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 $O(|\hat{S}|^2 \cdot |\phi_1| \cdot (|\hat{S}| + |R_{may}|))$.

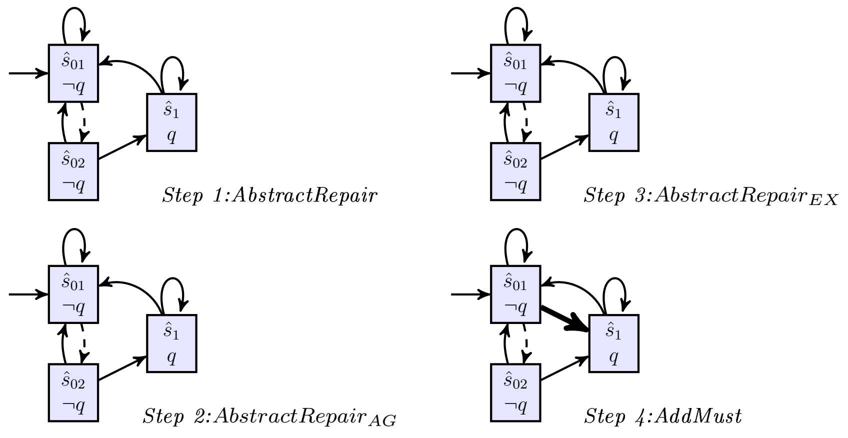
In the same way, it is easy to show that: (i) if $\phi = EG\phi_1$, then *Abstract Repair*$_{EG}$ is in $O(|\hat{S}| \cdot |\phi_1| \cdot (|\hat{S}| + |R_{must}|)$, (ii) if $\phi = AG\phi_1$, then *Abstract Repair*$_{AG}$ is in $O(|\hat{S}| \cdot |\phi_1| \cdot (|\hat{S}| + |R_{may}|))$, (iii) if $\phi = E(\phi_1 U \phi_2)$, 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 $O(|\hat{S}|^2 \cdot |\phi|^2 \cdot (|\hat{S}| + |R_{may}|))$.

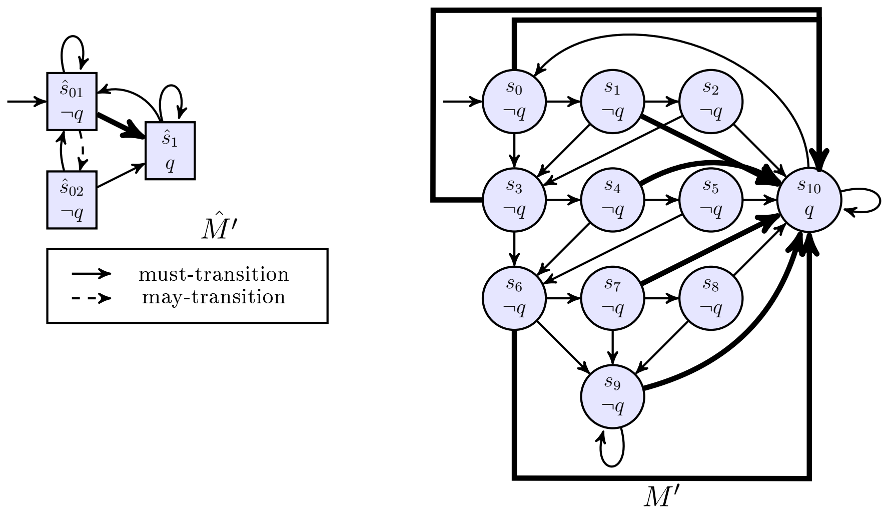
3.3.4 Application

We present the application of *AbstractRepair* on the ADO system from Section 2.1. After the first two steps of our repair process, *AbstractRepair* is called for the KMTS $\alpha_{Refined}(M)$ that is shown in Fig. 2.3b, 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, $Abstract\ Repair_{EX}$ 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. 21). The execution steps of *AbstractRepair* and the obtained repaired KMTS and KS are shown in Fig. 3.10a and Fig. 3.10b respectively.



(a) Application of *AbstractRepair*.



(b) The repaired KMTS and KS.

Figure 3.10: Repair of ADO system using abstraction.

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.

3.3.5 Discussion

AMR algorithm has a strong efficiency upper hand with respect to other concrete model repair methods, because all the operations needed to acquire the repaired solution take place on the abstract model which may be significantly smaller than the state space of the concrete model.

The refinement process which is a main part of the AMR framework always tries to find a bigger and potentially more precise KMTS, in the case that a repair solution cannot be retrieved from the current abstract KMTS. Despite the fact that the current refinement method is not monotonic, i.e. it does not ensure that it will always terminate with an abstract model which will give a definite answer to the model checking problem, it will always converge to the initial concrete model. Consequently, our AMR framework does not reduce the possibility of obtaining a repaired KS compared to the direct concrete model repair method. All cases, having in mind that the use of GKMTSs [78] as the abstract model can lead to monotonic refinement method, it makes the potential of creating an AMR framework based on GKMTSs far more intriguing.

The distance of the repair solution found from the AMR framework from the initial concrete model may be greater than the distance of the repair solution found from a direct concrete model repair method to the initial concrete model. This is the price we should pay for the use of abstraction though in practice the price is a benefit in the sense that direct concrete model repair algorithms cannot terminate due to the state space explosion problem for models with large state spaces.

In this thesis, the use of constraints in our AMR algorithm is used only as a means of repairing nested or conjunctive CTL formulas. These constraints can be used in the future in order to ensure that the repair solution returned from the algorithm satisfies a number of major CTL formulas which are also satisfied in the initial concrete model, thus ensuring that crucial properties of the model are not violated in the final repaired KS.

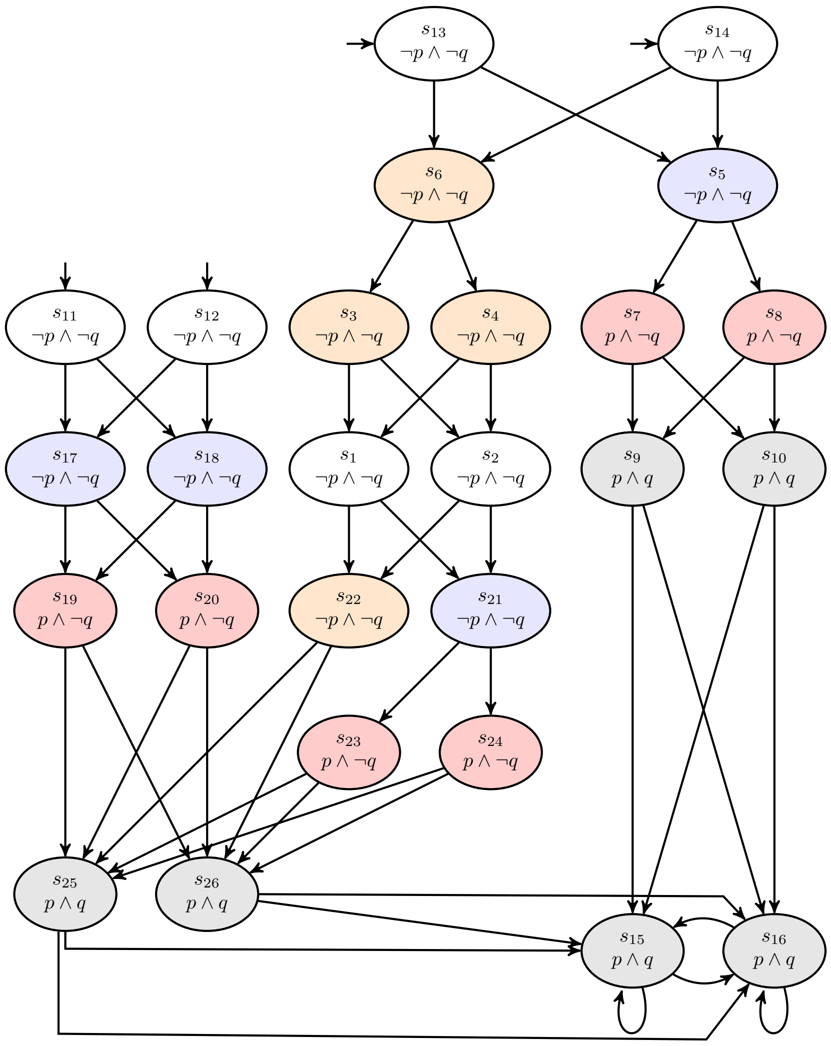
3.4 Experimental Results: The Andrew File System 1 (AFS1) Protocol

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 [89]. The results serve as a proof of concept for the use of abstraction in model repair and demonstrate the practical utility of our approach.

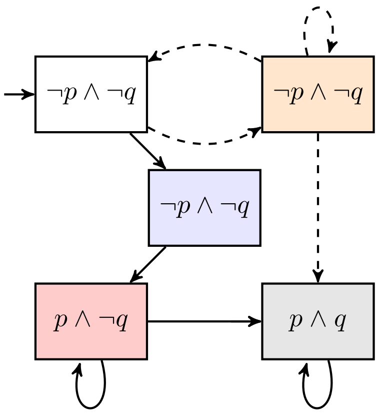
As a model we use a KS for the Andrew File System Protocol 1 (AFS1) [88], which has been repaired for a specific property in [89]. AFS1 is a client-server cache coherence protocol for a distributed file system. Four values are used for the client's belief about a file (nofile, valid, invalid, suspect) 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:

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

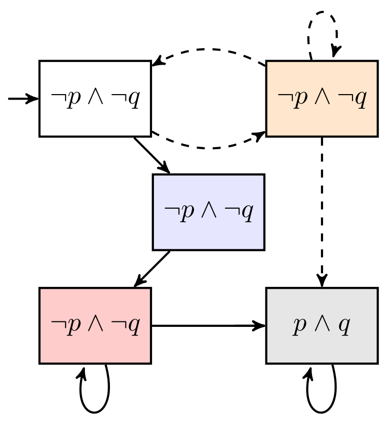

s_{13} $\neg p \wedge \neg q$
s_{14} $\neg p \wedge \neg q$
s_6 $\neg p \wedge \neg q$
s_5 $\neg p \wedge \neg q$
s_{11} $\neg p \wedge \neg q$
s_{12} $\neg p \wedge \neg q$
s_3 $\neg p \wedge \neg q$
s_4 $\neg p \wedge \neg q$
s_7 $p \wedge \neg q$
s_8 $p \wedge \neg q$
s_{17} $\neg p \wedge \neg q$
s_{18} $\neg p \wedge \neg q$
s_1 $\neg p \wedge \neg q$
s_2 $\neg p \wedge \neg q$
s_9 $p \wedge q$
s_{10} $p \wedge q$
s_{19} $p \wedge \neg q$
s_{20} $p \wedge \neg q$
s_{22} $\neg p \wedge \neg q$
s_{21} $\neg p \wedge \neg q$
s_{23} $p \wedge \neg q$
s_{24} $p \wedge \neg q$
s_{25} $p \wedge q$
s_{26} $p \wedge q$
s_{15} $p \wedge q$
s_{16} $p \wedge q$

(a) The KS after the final refinement step.

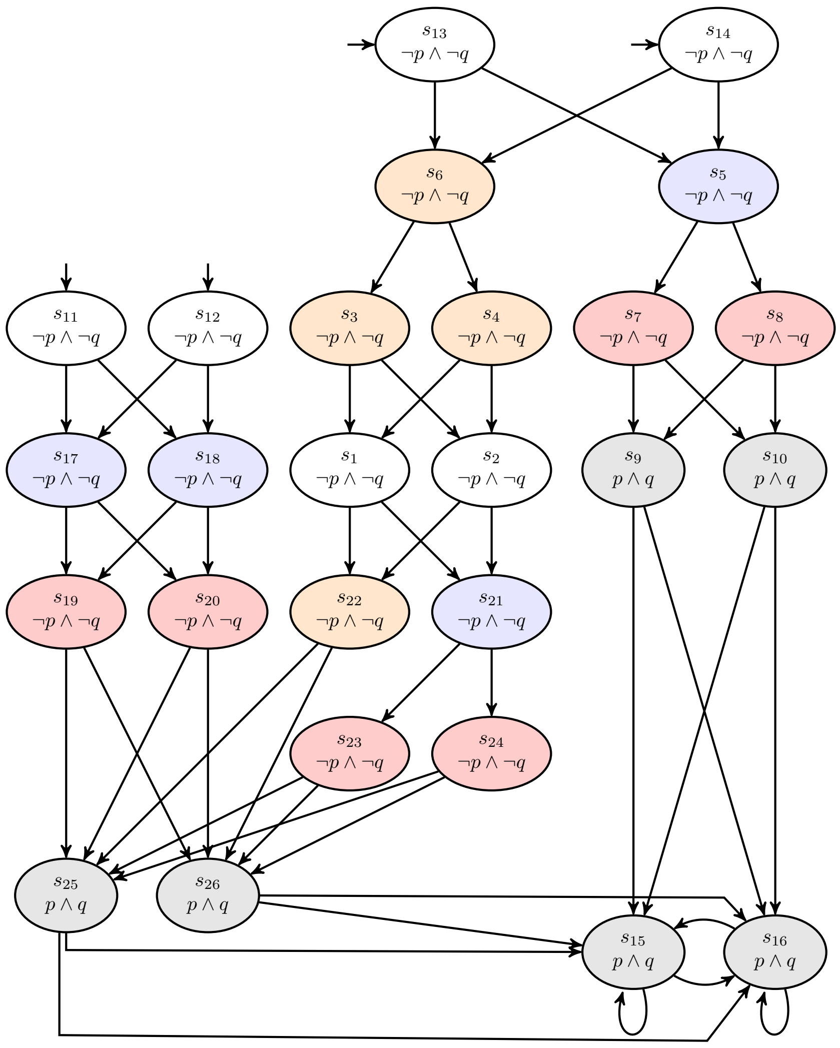


(b) The refined KMTS.

Figure 3.11: The KS and the KMTS of the AFS1 protocol after the 2nd refinement step.



(a) The repaired KMTS.



s_{13} $\neg p \wedge \neg q$
s_{14} $\neg p \wedge \neg q$
s_6 $\neg p \wedge \neg q$
s_5 $\neg p \wedge \neg q$
s_{11} $\neg p \wedge \neg q$
s_{12} $\neg p \wedge \neg q$
s_3 $\neg p \wedge \neg q$
s_4 $\neg p \wedge \neg q$
s_7 $\neg p \wedge \neg q$
s_8 $\neg p \wedge \neg q$
s_{17} $\neg p \wedge \neg q$
s_{18} $\neg p \wedge \neg q$
s_1 $\neg p \wedge \neg q$
s_2 $\neg p \wedge \neg q$
s_9 $p \wedge q$
s_{10} $p \wedge q$
s_{19} $\neg p \wedge \neg q$
s_{20} $\neg p \wedge \neg q$
s_{22} $\neg p \wedge \neg q$
s_{21} $\neg p \wedge \neg q$
s_{23} $\neg p \wedge \neg q$
s_{24} $\neg p \wedge \neg q$
s_{25} $p \wedge q$
s_{26} $p \wedge q$
s_{15} $p \wedge q$
s_{16} $p \wedge q$

(b) The repaired KS.

Figure 3.12: The repaired KMTS and KS of the AFS1 protocol.

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. 3.11a. State colors show how they are abstracted in the KMTS of Fig. 3.11b, which is derived after the 2nd refinement step of our AMR framework (Fig. 3.1). 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 3.1. 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. 3.12.

An interesting observation from the application of the AMR algorithm on the repair of the AFS1 KS is that the distance d (cf. Def. 17) of the repaired KS from the original KS is less than the corresponding distance obtained from the concrete model repair algorithm in [89]. This result demonstrates in practice the effect of the minimality of changes ordering, on which the AMR algorithm is based on (cf. Fig. 3.9).

Models	Concrete States (thousands)	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 3.1: Experimental results of AMR with respect to concrete repair

3.5 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* [84], 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* [9]: 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 [74]. The problem of repairing a Boolean program has been formulated in [80, 63, 56, 85] as the finding of a winning strategy for a game between two players. The only exception is the work reported in [77].

Another line of research on program repair treats the repair as a search problem and applies innovative evolutionary algorithms [6], *behavioral programming* techniques [59] or other informal heuristics [86, 2, 87].

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 [5]. 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 [23]. 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 [89]. 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 [26], 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 [90], 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 [67], the authors propose the use of constraint automata for ACTL formulas, while in [25] 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 [48], 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 [42], an extension of the model repair problem has been studied for Labeled Transition Systems. In [13], 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 [75]. In [15], 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 [16, 18] and for CTL in [58]. Other methods in the area of fault-tolerance include the work in [52], which uses discrete controller synthesis and [50], which employs SMT solving. Another interesting work in this direction is in [47], where the authors present a repair algorithm for fault-tolerance in a fully connected topology, with respect to a temporal specification.

Chapter 4

Abstract Model Repair for Probabilistic systems

4.1 The Model Repair problem for probabilistic systems

In this section, we define a metric space for measuring the distance between DTMCs with the same state labeling, and then the Model Repair problem for DTMCs [13].

Definition 35. *For any two vectors $A = (a_1, ..., a_n)$ and $B = (b_1, ..., b_n)$ with length n, the Manhattan distance d_m is defined as follows:*

$$d_m(A, B) = \sum_{i=1}^{n} |a_i - b_i|$$

. □

Definition 36. *For any two $M = (S, s_{init}, P, L)$ and $M' = (S, s_{init}, P', L)$ in the set D_M of all DTMCs with the same state space S and the same labeling function L, the* distance function d *over D_M is defined as follows:*

$$d(M, M') = d_m(A, B)$$

with $A = (P(s_0, s_0), P(s_0, s_1), \dots, P(s_i, s_j) \dots P(s_n, s_n))$
and $B = (P'(s_0, s_0), P'(s_0, s_1), \dots, P'(s_i, s_j) \dots P'(s_n, s_n))$ for all $s_i, s_j \in S$.

For any two DTMCs with the same state space S and the same labeling function L, the function d measures the differences between the corresponding transition probabilities.

Proposition 21. *The ordered pair (D_M, d) is a metric space.*

Proof 1. *This is inferred from the fact that the Manhattan distance between any two vectors is a distance metric.* □

Definition 37. *Given a DTMC M and a PCTL formula φ where $M \not\models \varphi$, the Model Repair problem is to find a DTMC M', such that $M' \models \varphi$ and $d(M, M')$ is minimum with respect to all such M'.*

The objective of the Model Repair problem in the context of probabilistic systems is to modify the given DTMC, in order to satisfy a PCTL property φ, which is not satisfied. The repair solution should have the smallest possible distance from the initial model compared to all other models satisfying φ. In next sections we show how the model repair problem is transformed into a minimization problem for the changes of the transition probabilities in the DTMC.

4.2 The PAMR framework

The PAMR framework uses 3-valued abstract probabilistic models for the repair of DTMCs with respect to given probabilistic reachability properties (ADTMCs) . The goal of our framework is to provide a method that looks for the repair solution with the smallest possible distance from a given DTMC. This method should be feasible and efficient for models with large state spaces.

The process consists of the following steps, which are shown in Fig. 4.1.

Step 1. For a DTMC M, a state s and a property $\varphi = P_{\leq p}[F\psi]$ or $\varphi = P_{\geq p}[F\psi]$, such that $(M, s) \not\models \varphi$, an ADTMC $\hat{M} = \alpha(M)$ is acquired as in Def. 16.

Step 2. The PAMR algorithm is called with inputs M, $\hat{M}$, $\hat{s} = \alpha(s)$ and the property φ.

Case 1. If the result is FAILURE, i.e. a repair solution is not found, then a *refinement* step takes place for $\hat{M}$, and:

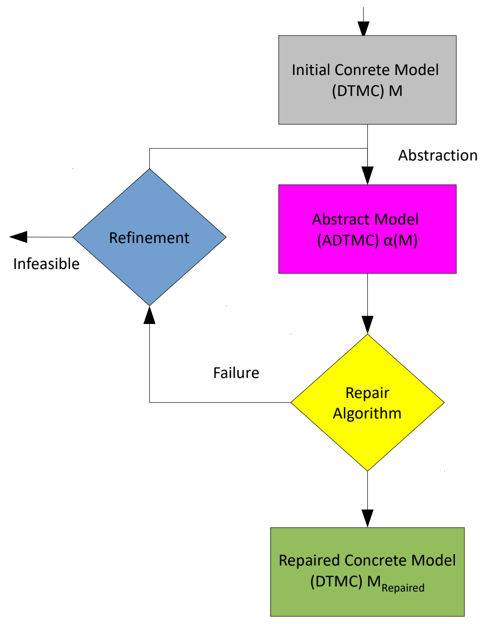


Figure 4.1: Probabilistic Abstract Model Repair Framework.

Case 1.1. If an ADTMC $\hat{M}_{Refined}$ is found, then the control is transfered to Step 2 with $\hat{M}_{Refined}$ as input.

Case 1.2. If it is not feasible to retrieve a new refined ADTMC, the repair process terminates with a FAILURE.

Case 2. A repaired DTMC $M_{Repaired}$ is found.

Example. (ROBOT) The initial abstraction for the DTMC of the robot system is shown in Fig. 2.5. In this case, a predicate abstraction is used where the concrete state s_{12}, which is the only one where q is true, is abstracted to one state, while all the other states which are not labeled with q are abstracted to a different state. Thus, a two-state ADTMC is constructed which is the first abstract model arising from the concrete DTMC M. $\square$

4.2.1 Refinement

The refinement step plays a fundamental role in the PAMR process, since it is the means to obtain a repair solution, if possible, when the PAMR algorithm terminates with a FAILURE result in a previous step. Moreover, even if a repair solution has been already found, it may be possible to apply a refinement step in order to find a more fine-grained repair solution.

In the related bibliography, there are various proposals for the refinement of models for probabilistic systems [60, 65, 66, 39, 68, 28]. The ultimate goal of such a technique is to get an *optimally* refined model with respect to some criteria, which differ in each work. We adopt a simple, yet effective refinement technique, which fulfills the following objective: the refinement procedure monotonically converges to the concrete DTMC with respect to the size of the model's state space. To this end, at each refinement step we split the abstract states invalidating the atomic propositions of the reachability property.

Nevertheless, the PAMR process can be adapted through the adoption of alternative refinement methods according to different criteria (the so-called *repair strategies* [13]).

Example. (ROBOT) From the ADTMC in Fig. 2.5 for the robot system, a refined model can be derived by splitting the abstract state not satisfying q in two states, thus getting a three-state ADTMC. The refined ADTMC and the corresponding partitioning of the DTMC's state space are shown in Fig. 4.2.

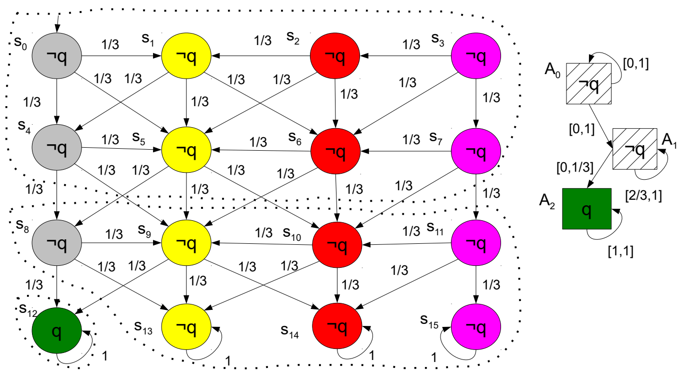


Figure 4.2: DTMC and ADTMC after the first refinement step.

The ADTMC of Fig. 4.2 can be further refined by splitting both abstract states that do not satisfy q in two states. This second refinement and the corresponding partitioning of the DTMC's state space are shown in Fig. 4.3. The result obtained from the third refinement step is shown in Fig. 4.4, from which a subsequent refinement yields the concrete DTMC.

The impact of the refinement method on the PAMR process is better illustrated if we consider the ADTMC obtained from Fig. 2.5 using an alternative refinement. The result is shown in Fig. 4.5, which differs from the ADTMC in Fig. 4.2 and will eventually lead to a different repair solution. □

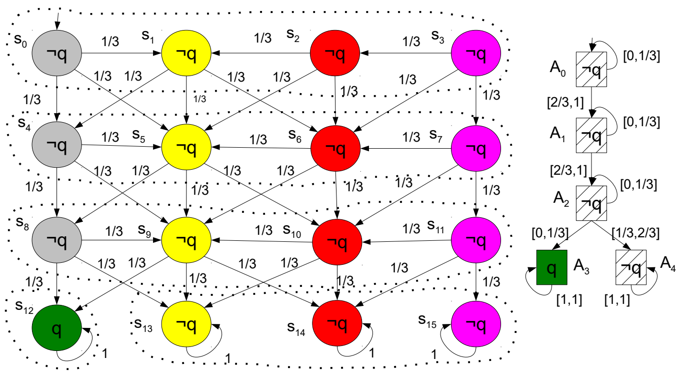


Figure 4.3: DTMC and ADTMC after the second refinement step.

4.2.2 Repair operations for ADTMCs

In the PAMR process, the model repair problem is reduced to a problem addressed through a smaller state space, i.e. that of the abstract ADTMC. Every change to the transition probabilities of the ADTMC by the PAMR algorithm is mapped to the concrete DTMC through the pair of functions (α, γ) in Def. 16.

Repairing the DTMC with respect to the reachability property $\varphi = P_{\geq p}[F\psi]$ means, according to the 3-valued semantics of PCTL in Def. 15, that specific lower bound transition probabilities of the ADTMC should be increased (similarly in the case of $P_{\geq p}[F^{\leq k}\psi]$). When the DTMC is repaired with respect to $\varphi = P_{\leq p}[F\psi]$, specific upper bound transition probabilities should be decreased. We hereby introduce two distinct repair operations applicable to ADTMCs, the IncreaseLowerBound and DecreaseUpperBound.

Definition 38 (IncreaseLowerBound)**.** *Let* $M = (S, s_{init}, P, L)$ *a DTMC and*

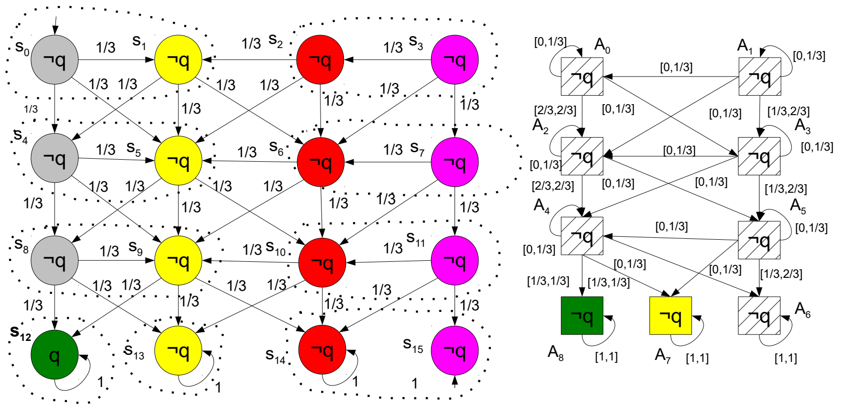


Figure 4.4: DTMC and ADTMC after the third refinement step.

the ADTMC derived from M as in Def. 16 be $\alpha(M) = (\hat{S}, \hat{s}_{init}, P^{\ell}, P^{u}, \hat{L})$. For a given $v_0 \in \mathbb{R}_+$ and a pair $\hat{s}_i, \hat{s}_j \in \hat{S}$, let $prob_{min} = P^{\ell}(\hat{s}_i, \hat{s}_j) + v_0$. Then, $\forall s_m \in S \setminus \gamma(\hat{s}_i), s_n \in S, P'(s_m, s_n) = P(s_m, s_n)$, whereas for all $s_i \in \gamma(\hat{s}_i), s_j \in \gamma(\hat{s}_j), s_k \in S \setminus \gamma(\hat{s}_j)$:

$$P'(s_i, s_j) = \begin{cases} P(s_i, s_j) + \mathit{diff}/card_j & \text{if } \mathit{diff} = prob_{min} - prob_{out} > 0 \\ P(s_i, s_j) & \text{otherwise} \end{cases}$$

$$P'(s_i, s_k) = \begin{cases} P(s_i, s_k) - \mathit{diff}/card_k & \text{if } \mathit{diff} = prob_{min} - prob_{out} > 0 \\ P(s_i, s_k) & \text{otherwise} \end{cases}$$

with $prob_{out} = \sum_{s_i \in \gamma(\hat{s}_i), s_j \in \gamma(\hat{s}_j)} P(s_i, s_j)$, $card_j = |\{(s_i, s_j)\}|$ and $card_k = |\{(s_i, s_k)\}|$.

If P' fulfills the stochastic conditions, thus $P'(s_i, s_j), P'(s_i, s_k) \in [0, 1]$ for

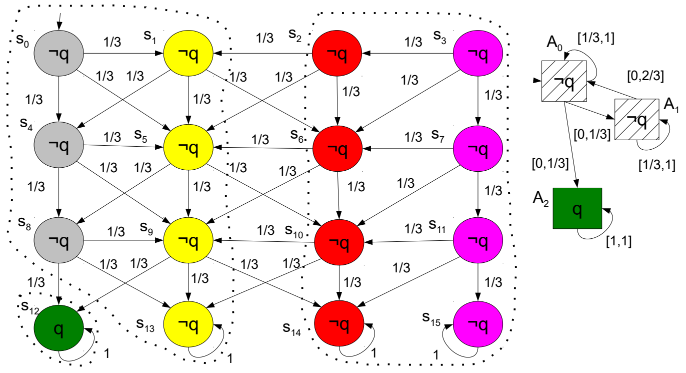


Figure 4.5: DTMC and ADTMC for a different refinement from Fig. 4.2.

all s_i, s_j, s_k*, then the DTMC* $M' = (S, s_{init}, P', L) = IncreaseLowerBound$ $(M, \alpha(M), (\hat{s}_i, \hat{s}_j), v_0)$ *is defined.*

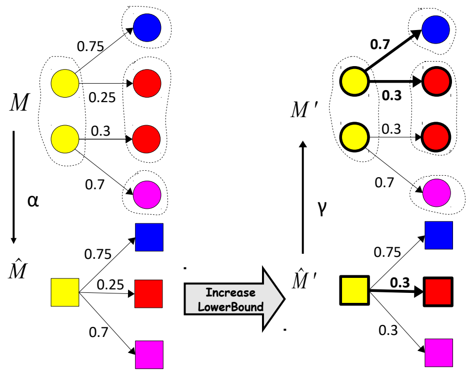


Figure 4.6: Increasing the lower bound of an ADTMC and the resulting concrete DTMC. (The lower bound probabilities of the ADTMC are only shown.)

The IncreaseLowerBound operation, which is illustrated in Fig. 4.6, increases the lower bound probability of a transition $(\hat{s}_i, \hat{s}_j)$ of the ADTMC (yellow to red transition of $\hat{M}$) by some given value $v_0 \in \mathbb{R}_+$ through modifying the DTMC M; this is feasible for the given v_0, if and only if:

$$\hat{P}(s_i, S_j) + v_0 \leq 1, \text{ for all } s_i \in \gamma(\hat{s}_i), \text{ with } S_j = \{s_j : s_j \in \gamma \hat{s}_j\} \tag{4.1}$$

$$\hat{P}(s_i, S_k) - v_0 \geq 0, \text{ for all } s_i \in \gamma(\hat{s}_i), \text{ with } S_k = \{s_k : s_k \in S \setminus \gamma(\hat{s}_j)\} \tag{4.2}$$

i.e. there is a concretization strategy for the given v_0, such that a DTMC exists (stochastic constraints are satisfied). Def. 38 proposes a particular concretization strategy: from all concrete s_i (yellow states of M) of the abstract $\hat{s}_i$, their outgoing transition probabilities to s_j (red states of M) of the abstract $\hat{s}_j$ are increased uniformly to fulfill Def. 16 for the ADTMC with the increased lower bound probability. All other probabilities for outgoing transitions from s_i to s_k (blue state of M) are decreased uniformly, in order to preserve the stochastic conditions; if this is not feasible for the given v_0, then the IncreaseLowerBound operation fails to return a DTMC M'.

In fact, the ADTMC entails multiple DTMCs, and thus, in our case a repaired ADTMC allows for a multitude of ways to repair the DTMC. Apart from the concretization strategy of Def. 38, other strategies could be applied either, (i) interactively by the user, or (ii) automatically, in order to find a DTMC, i.e. the stochastic conditions to be satisfied.

Definition 39 (DecreaseUpperBound)**.** *Let $M = (S, s_{init}, P, L)$ a DTMC and the ADTMC derived from M as in Def. 16 be $\alpha(M) = (\hat{S}, \hat{s}_{init}, P^{\ell}, P^{u}, \hat{L})$. For a given $v_0 \in \mathbb{R}_+$ and a pair $\hat{s}_i, \hat{s}_j \in \hat{S}$, let $prob_{max} = P^u(\hat{s}_i, \hat{s}_j) - v_0$.*

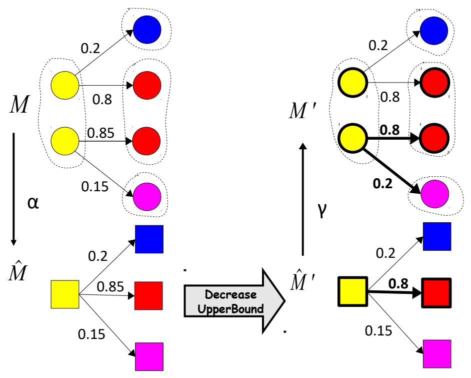


Figure 4.7: Decreasing the upper bound of an ADTMC and the resulting concrete DTMC. (The upper bound probabilities of the ADTMC are only shown.)

Then, $\forall s_m \in S \setminus \gamma(\hat{s}_i), s_n \in S, P'(s_m, s_n) = P(s_m, s_n)$*, whereas for all* $s_i \in \gamma(\hat{s}_i), s_j \in \gamma(\hat{s}_j), s_k \in S \setminus \gamma(\hat{s}_j)$*:*

$$P'(s_i, s_j) = \begin{cases} P(s_i, s_j) - \mathit{diff}/\mathit{card}_j & \text{if } \mathit{diff} = \mathit{prob}_{out} - \mathit{prob}_{max} > 0 \\ P(s_i, s_j) & \text{otherwise} \end{cases}$$

$$P'(s_i, s_k) = \begin{cases} P(s_i, s_k) + \mathit{diff}/\mathit{card}_k & \text{if } \mathit{diff} = \mathit{prob}_{out} - \mathit{prob}_{max} > 0 \\ P(s_i, s_k) & \text{otherwise} \end{cases}$$

with $\mathit{prob}_{out} = \sum_{s_i \in \gamma(\hat{s}_i), s_j \in \gamma(\hat{s}_j)} P(s_i, s_j)$*,* $\mathit{card}_j = |\{(s_i, s_j)\}|$ *and* $\mathit{card}_k = |\{(s_i, s_k)\}|$.

If P' *fulfills the stochastic conditions, thus* $P'(s_i, s_j), P'(s_i, s_k) \in [0, 1]$ *for* *all s_i, s_j, s_k, then the DTMC $M' = (S, s_{init}, P', L) = DecreaseUpperBound$ $(M, \alpha(M), (\hat{s}_i, \hat{s}_j), v_0)$ is defined.*

The DecreaseUpperBound operation, which is illustrated in Fig. 4.7, decreases the upper bound probability of a transition $(\hat{s}_i, \hat{s}_j)$ of the ADTMC (yellow to red transition of $\hat{M}$) by some given value $v_0 \in \mathbb{R}_+$ by modifying the DTMC M; this is feasible for the given v_0 if and only if equations (4.1) and (4.2) hold.

4.3 The PAMR algorithm

Algorithm 13 PAMR

```
Input: $M = (S, s_{init}, P, L)$, $\hat{M} = \alpha(M) = (\hat{S}, \hat{s}_{init}, P^{\ell}, P^{u}, \hat{L})$, $\hat{s} \in \hat{S}$ and a
    property $\varphi = P_{\geq p}[F\psi]$ or $\varphi = P_{\leq p}[F\psi]$ such that $(\hat{M}, \hat{s}) \not\models \varphi$.
Output: $M' = (S, s_{init}, P', L)$ such that $(M', s) \models \varphi$ or FAILURE.
 1: if $\varphi = P_{\geq p}[F\psi]$ then
 2:     $P^{\ell}_{v_0}$ := AddV$_0$LowerToADTMC$(M, \hat{M}, \hat{s}, \varphi)$
 3:     $eq\_set_{v_0}$ := LowerReachabilityEquations$(\hat{M}, P^{\ell}_{v_0}, \hat{s}, \varphi)$
 4:     $v_0$ := NLPSolve$(min(v_0), eq\_set_{v_0}, x \geq p, v_0 > 0)$
 5:     return ConcretizeLowerRepairedModel$(M, \hat{M}, P^{\ell}_{v_0}, \hat{s}, \varphi, v_0)$
 6: else if $\varphi = P_{\leq p}[F\psi]$ then
 7:     $P^{u}_{v_0}$ := SubtractV$_0$UpperToADTMC$(\hat{M}, \hat{s}, \varphi)$
 8:     $eq\_set_{v_0}$ := UpperReachabilityEquations$(\hat{M}, P^{u}_{v_0}, \hat{s}, \varphi)$
 9:     $v_0$ := NLPSolve$(min(v_0), eq\_set_{v_0}, x \leq p, v_0 > 0)$
10:     return ConcretizeUpperRepairedModel$(M, \hat{M}, P^{u}_{v_0}, \hat{s}, \varphi, v_0)$
```

The algorithm is executed at Step 2 of the PAMR process. For brevity, we present the PAMR algorithm for the case of unbounded properties, whereas for the bounded properties there are minor differences that we discuss at the end. The algorithm accepts as input the concrete DTMC M, the abstract ADTMC $\hat{M} = \alpha(M)$, a state $\hat{s} = \alpha(s)$ and a reachability property $\varphi = P_{\geq p}[F\psi]$ or $\varphi = P_{\leq p}[F\psi]$ such that $(M, s) \not\models \varphi$. The main body of the algorithm with two possible execution paths, one for $\varphi = P_{\geq p}[F\psi]$ and the other for $\varphi = P_{\leq p}[F\psi]$, is shown in Algorithm 13.

Algorithm 14 AddV$_0$LowerToADTMC

```
Input: $M = (S, s_{init}, P, L)$, $\hat{M} = (\hat{S}, \hat{s}_{init}, P^{\ell}, P^{u}, \hat{L})$, $\hat{s} \in \hat{S}$ and a property
    $\varphi = P_{\geq p}[F\psi]$.
Output: $P^{\ell}_{v_0}$
1: for all $\hat{s}_k$ such that $(\hat{M}, \hat{s}_k) \models \psi$ do
2:    for all $(\hat{s}_i, \hat{s}_j)$ in shortest maximal paths of the form
      $\pi = [\hat{s}, ..., \hat{s}_i, \hat{s}_j, ..., \hat{s}_k]$ where for all $(\hat{s}_i, \hat{s}_j)$ of $\pi$, $P(s_i, s_j) > 0$ for some
      $s_i, s_j \in S$ with $\hat{s}_i = \alpha(s_i)$, $\hat{s}_j = \alpha(s_j)$ do
3:       $P^{\ell}_{v_0}(\hat{s}_i, \hat{s}_j) := P^{\ell}(\hat{s}_i, \hat{s}_j) + v_0$ with $v_0 > 0$
```

Algorithm 15 LowerReachabilityEquations

```
Input: $\hat{M} = (\hat{S}, \hat{s}_{init}, P^{\ell}, P^{u}, \hat{L})$, $P^{\ell}_{v_0}$, $\hat{s} \in \hat{S}$ and a property $\varphi = P_{\geq p}[F\psi]$.
Output: The equations for the probability measure of $\varphi = P_{\geq p}[F\psi]$.
1: Generate the equations for $\varphi$ from the probabilities in $P^{\ell}_{v_0}$ as in Def. 11.
```

If $\varphi = P_{\geq p}[F\psi]$, then AddV$_0$LowerToADTMC is initially called with M, $\hat{M}$, $\hat{s}$ and φ as arguments and returns a transition probability function $P^{\ell}_{v_0}$. This function modifies $\hat{M}$'s lower bound transition probabilities according to a repair strategy defining with a parameter v_0 which transition probabilities are increased, in order for LowerReachabilityEquations to subsequently generate the nonlinear equations for the probability measure of φ. These equations, together with the objective function, the inequality constraints for v_0 and the probability measure of φ are then passed to NLPSolve. If a solution is returned for v_0, then ConcretizeLowerRepairedModel is called with arguments M, $\hat{M}$, $P^{\ell}_{v_0}$, $\hat{s}, \varphi$ and v_0, and either a repaired model M' is found or else it returns FAILURE. The returned value is eventually reported as the final result of the main PAMR algorithm.

More specifically, the body of AddV$_0$LowerToADTMC is described in Algorithm 14. The P^{ℓ} is modified as follows. For the shortest maximal paths (i.e. paths with transitions that appear at most once) from $\hat{s}$ to a state $\hat{s}_k$ which satisfies ψ, in all transitions $(\hat{s}_i, \hat{s}_j)$ for which there is transition between corresponding concrete states with non-zero probability, the $P^{\ell}(\hat{s}_i, \hat{s}_j)$ is increased by adding the parameter v_0.

The function LowerReachabilityEquations is described in Algorithm 15, where the equations for the probability measure x of φ being true in the ADTMC are generated as in Def. 11. We note here that *the number of generated equations is equal to the number of states of the ADTMC.*

Algorithm 16 NLPSolve

```
Input: An objective function $obj(v_0)$, a set of equations $eq\_set_{v_0}$, an in-
    equality constraint for an unknown in $eq\_set_{v_0}$ and the $v_0$'s range.
Output: The value $v_0$ for which $obj$ is optimized or FAILURE.
 1: Use the Sequential Quadratic Programming (SQP) method to solve the
    non-linear optimization problem for $v_0$.
```

The function NLPSolve is described in Algorithm 16. This function is called in line 4 of Algorithm 13 with $min(v_0)$ as the objective function, the equations $eq\_set_{v_0}$ generated by LowerReachabilityEquations, the inequality constraint $x \geq p$, where p is the given probability bound, and the inequality constraint which ensures positive solution for v_0.

For NLPSolve, the exact solution of constrained nonlinear optimization problems has been proved to be NP-Hard in the worst case [83]. We adopt the Sequential Quadratic Programming (SQP) [14] method, which looks for a *local* solution instead of a *global* one. The complexity of SQP algorithms is bound to the number of equations, which in NLPSolve is equal to the number of the abstract states, and can be exponential in the worst case [72]. *By applying SQP to the state-space of the ADTMC we achieve significant efficiency gains compared to applying it to the state-space of the concrete model M (Table 4.1).*

Algorithm 17 ConcretizeLowerRepairedModel

```
Input: $M = (S, s_{init}, P, L)$, $\hat{M} = (\hat{S}, \hat{s}_{init}, P^{\ell}, P^{u}, \hat{L})$, $P^{\ell}_{v_0}$, $\hat{s} \in \hat{S}$, a property
    $\varphi = P_{\geq p}[F\psi]$ and a $c \in \mathbb{R}_+$.
Output: $M_{Repaired} = (S, s_{init}, P', L)$ or FAILURE.
 1: $M' := M$
 2: for all $(\hat{s}_i, \hat{s}_j) \in \hat{S} \times \hat{S}$ with $P^{\ell}_{v_0}(\hat{s}_i, \hat{s}_j) = P^{\ell}(\hat{s}_i, \hat{s}_j) + v_0$ do
 3:     RET := IncreaseLowerBound$(M', \hat{M}, (\hat{s}_i, \hat{s}_j), c)$
 4:     if RET == FAILURE then
 5:         return FAILURE
 6:     else
 7:         $M'$ := RET
 8: $\hat{M}' := \alpha(M')$
 9: if AbstractModelCheck$(\hat{M}', \varphi)$ == TRUE then
10:     return $M'$
11: return FAILURE
```

The function ConcretizeLowerRepairedModel in Algorithm 17, which finds the repaired DTMC M', is called if NLPSolve computes a solution for v_0 that is passed as argument. The IncreaseLowerBound repair operation is invoked for all $(\hat{s}_i, \hat{s}_j)$, for which the transition probability in $P^{\ell}_{v_0}$ includes v_0. Each invocation of IncreaseLowerBound produces a modified DTMC or a FAILURE result, if the computed value for v_0 cannot yield a valid DTMC because the stochastic conditions are not fulfilled. For the M' produced by the final call of IncreaseLowerBound, the algorithm verifies if it really satisfies φ. This check is essential because IncreaseLowerBound operation alters the transition probabilities of the DTMC in order to preserve the stochastic conditions and these changes might affect the probability measure for φ. As opposed to other model repair approaches [75], which validate the solution by model checking the concrete model, we model check φ over the abstract ADTMC $\hat{M}'$. If the model checking result is true, then M' is returned by the PAMR algorithm as the repair solution.

Algorithm 18 SubtractV$_0$UpperToADTMC

```
Input: $\hat{M} = (\hat{S}, \hat{s}_{init}, P^{\ell}, P^{u}, \hat{L})$, $\hat{s} \in \hat{S}$ and a property $\varphi = P_{\leq p}[F\psi]$.
Output: $P^{u}_{v_0}$
1: for all $\hat{s}_k$ such that $(\hat{M}, \hat{s}_k) \models \psi$ do
2:     for all $(\hat{s}_i, \hat{s}_j)$ in shortest maximal paths of the form
       $\pi = [\hat{s}, ..., \hat{s}_i, \hat{s}_j, ..., \hat{s}_k]$ with $P^{u}(\hat{s}_i, \hat{s}_j) > 0$ for all $(\hat{s}_i, \hat{s}_j)$ of $\pi$ do
3:         $P^{u}_{v_0}(\hat{s}_i, \hat{s}_j) := P^{u}(\hat{s}_i, \hat{s}_j) - v_0$ with $v_0 > 0$
```

Algorithm 19 UpperReachabilityEquations

```
Input: $\hat{M} = (\hat{S}, \hat{s}_{init}, P^{\ell}, P^{u}, \hat{L})$, $P^{u}_{v_0}$, $\hat{s} \in \hat{S}$ and a property $\varphi = P_{\leq p}[F\psi]$.
Output: The equations for the probability measure of $\varphi = P_{\leq p}[F\psi]$.
1: Generate the equations for $\varphi$ from the probabilities in $P^{u}_{v_0}$ as in Def. 11.
```

The execution path of PAMR in Algorithm 13 for $\varphi = P_{\leq p}[F\psi]$ is implemented in an analogous way, with the only difference that instead of increasing the lower bounds, the upper bounds are decreased. The main body of the functions called in this execution path of PAMR are presented in Algorithms 18, 19 and 20.

Algorithm 20 ConcretizeUpperRepairedModel

```
Input: $M = (S, s_{init}, P, L)$, $\hat{M} = (\hat{S}, \hat{s}_{init}, P^{\ell}, P^{u}, \hat{L})$, $P^{u}_{v_0}$, $\hat{s} \in \hat{S}$, a property
    $\varphi = P_{\leq p}[F\psi]$ and a $c \in \mathbb{R}_{+}$.
Output: $M_{Repaired} = (S, s_{init}, P', L)$ or FAILURE
 1: $M' := M$
 2: for all $(\hat{s}_i, \hat{s}_j) \in \hat{S} \times \hat{S}$ with $P^{u}_{v_0}(\hat{s}_i, \hat{s}_j) = P^{u}(\hat{s}_i, \hat{s}_j) - v_0$ do
 3:     RET := DecreaseUpperBound$(M', \hat{M}, (\hat{s}_i, \hat{s}_j), c)$
 4:     if RET == FAILURE then
 5:         return FAILURE
 6:     else
 7:         $M'$ := RET
 8: $\hat{M}' := \alpha(M')$
 9: if AbstractModelCheck$(\hat{M}', \varphi)$ == TRUE then
10:     return $M'$
11: return FAILURE
```

The PAMR algorithm can be easily shown to be *sound* in the sense that if a DTMC M' is returned for the property φ being true at state s, then we have $(M', s) \models \varphi$. The proof is straightforward as a consequence of Theorem 2 and the fact that a DTMC is returned if and only if the result of model checking the ADTMC $\hat{M}'$ that is derived from M' is true.

For bounded properties of the form $P_{\bowtie p}[F^{\leq k}\psi]$, the AddV$_0$LowerToADTMC and SubtractV$_0$UpperToADTMC are modified, such that they work only for the paths of finite length k, whereas LowerReachabilityEquations is modified together with UpperReachabilityEquations, in order to produce the equations of Def. 12 instead of Def. 11.

4.4 Application and discusssion

We present the results of the repair of the DTMC in Fig. 2.4 for the robot system. In the illustrated scenario we are interested to achieving a probability level of at least 0.325 for the robot to reach the *green* state. The PCTL property for the mentioned goal is $\varphi = P_{\geq 0.325}[Fq]$, where q is the atomic proposition for $color = green$. This property is not satisfied in the model M of Fig. 2.4.

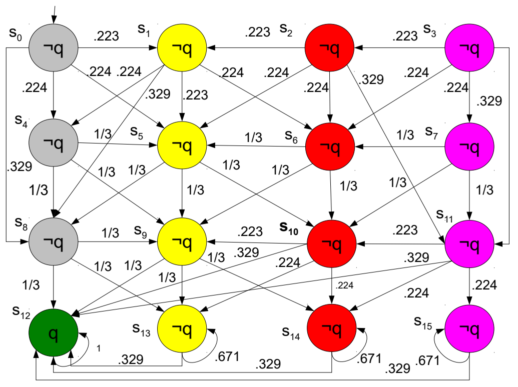


Figure 4.8: Repaired DTMC after the first refinement step.

When the PAMR algorithm is applied to the ADTMC of Fig. 2.5 (initial abstract model) the NLPSolve fails to compute a solution for the parameter v_0 (Case 1 of Step 2 in the PAMR process of Section 4.2). The process continues with a refinement step (Case 1.1 of Step 2 in the PAMR process) and the refined ADTMC of Fig. 4.2 is obtained. The PAMR algorithm generates the following equations for the refined ADTMC and φ:

$$
\begin{aligned}
x_0 &= 0x_0 + (0 + v_0)x_1 \\
x_1 &= \left(\frac{2}{3}\right) x_1 + (0 + v_0)x_2 \\
x_2 &= 1
\end{aligned}
$$

The NLPSolve function is eventually called for the above equations and the inequality constraint $x_0 \geq 0.325$. In this case, a solution is returned for the parameter v_0, which is 0.329. The execution of the PAMR algorithm is completed with the computation of the repaired DTMC shown in Fig. 4.8. Thus, a solution is obtained by reducing the repair problem for a model with 16 states to that for an abstract model with only 3 states.

4.4.1 Discussion

The efficiency advantage of the PAMR algorithm is based on the fact that the non-linear optimization problem is solved over the state space of the abstract ADTMC, which may be orders of magnitude smaller than the state space of the concrete DTMC.

The PAMR process of Section 4.2 does not deteriorate the possibility to obtain a repair solution compared to the concrete model repair case. This happens due to the fact that the refinement process always converges to the concrete DTMC, if in the meantime a repair solution is not found.

If a repair solution M' is found, the distance of M' from the initial DTMC M (Def. 36) may be greater than the distance of a solution derived by directly applying the repair to the concrete model. This happens because minimizing v_0 only represents the best we can do regarding the Manhattan distance minimality at the granularity of precision offered by the specific ADTMC. In essence, this is the price to pay for using abstraction though in practice this may be the only way to obtain a repair solution for models with large or even medium size state space. In our application, the differences between the first PAMR repair solution shown in Fig. 4.8 and the direct repair solution in Fig. 4.9 are noticeable.

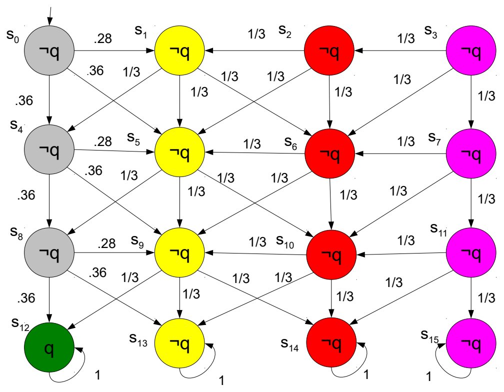


Figure 4.9: Repaired DTMC after applying repair directly to the concrete model.

Whatever the solution is, it is still possible for the refinement process to continue, such that the PAMR algorithm can be applied to ADTMCs with larger state spaces, in which case a more fine grained repair solution could be obtained. This is illustrated through applying the PAMR algorithm in the ADTMCs of Fig. 4.3 and Fig. 4.4, which depict respectively the ADTMCs after the second and the third refinement steps in the model for the robot system. The corresponding repaired DTMCs for these two cases are shown in Fig. 4.10 and Fig. 4.11.

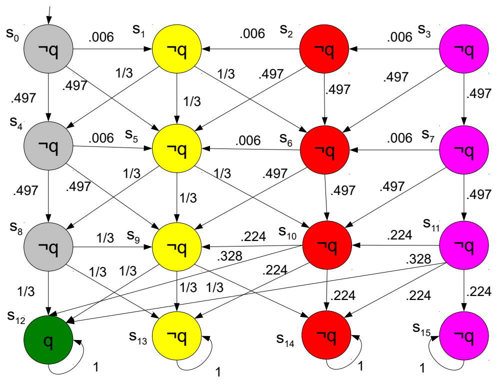


Figure 4.10: Repaired DTMC after the second refinement step.

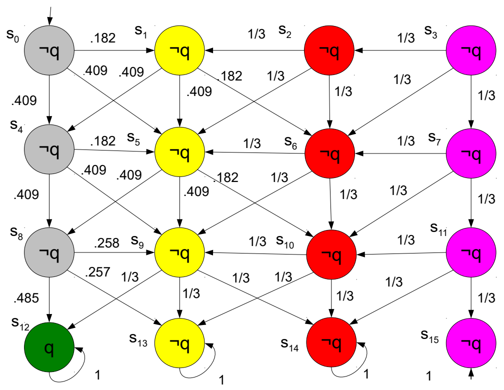


Figure 4.11: Repaired DTMC after the third refinement step.

The PAMR process can be effectively controlled through the refinement method, as well as through specifying how the transition probabilities are modified. These two provide the means to follow alternative repair strategies. The refinement method can be replaced by any other method which may be considered as more adequate for a specific application domain [60, 65, 66, 39, 68, 28]. The solution from applying the PAMR algorithm to the first refined ADTMC of Fig. 4.5 is shown in Fig. 4.12. It is clear that this solution differs from the one in Fig. 4.8 that was obtained through the previous refinement approach.

$\mathsf{AddV_0ToLowerADTMC}$ and $\mathsf{SubtractV_0ToUpperADTMC}$ specify which of the transition probabilities can be changed and how. The implementation of these functions in our algorithm aims to find a repair solution that distributes the changes to many transitions instead of localizing them to just a few, and that the transition probabilities are evenly changed. This repair strategy, as well as other strategies can result in sub-optimal solutions, i.e. the minimal repair according to the distance of Def. 35 may be missed. However, in practice it is not possible to adopt a single strategy for all domains where the repair problem can arise. Therefore, the aforementioned functions can be replaced by analogous user-defined functions, to specify the parameters $(v_0, v_1, ...)$ for modifying the transition probabilities, and which transition probabilities can be modified.

By using the same parameter v_0 for all transitions to be modified and the same implementation of $\mathsf{IncreaseLowerBound}, \mathsf{DecreaseUpperBound}$ for all modified transitions, we ensure that the same parameter can affect multiple transitions in the same way. This is essential if the PAMR framework is to be

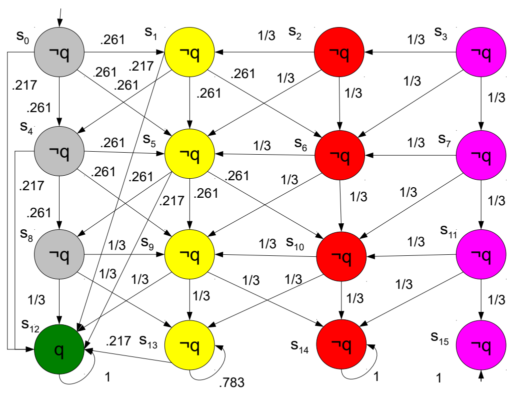


Figure 4.12: Repaired DTMC after the alternative first refinement step.

applied to a probabilistic model specified in a high-level guarded command language like the one used in widely used model checking tools [71, 45].

4.5 Experimental Results

We have implemented the PAMR algorithm using MAPLE [1]. In particular, we utilized specific functions for model checking ADTMCs (the `solve()` linear equations' solver) [70] and for the solution of the constrained non-linear optimization problem (the `NLPSolve()` function called with the `sqp` parameter).

Using these functions we compared the efficiency of the PAMR algorithm with that of the same algorithm when it is directly applied to the DTMC, i.e. without abstraction. In particular, we experimented with four systems from various domains, whose state spaces have significantly different structure. These systems are the Craps game [11], the IPv4 Zeroconf protocol [13, 69], a message authentication protocol [51] and the gambler's ruin model, which demonstrates the worst-case behaviour of the PAMR process in terms of its efficiency gains. The message authentication protocol is mentioned in [31], as a model repair scenario, where the primary aim is not finding the optimal repair solution, but it is sufficient to find some repair solution rapidly (fast model repair problem). Our PAMR framework fits ideally to this context, since a repair solution can be obtained even for very large model sizes, which can be afterwards refined with respect to the needs of the user.

The Craps game model shown in Fig. 4.13 refers to a dice game, where the player wins or loses based on the outcome of the roll of two dice. The outcomes 2, 3, 7, 11, 12 are "craps", i.e. the player loses. On any other outcome the dice are rolled again and the outcome of the come-out roll is remembered (the "point"). The dice are rolled repeatedly until the outcome is 7, in which case the player loses, or the outcome is the point, in which case the player wins.

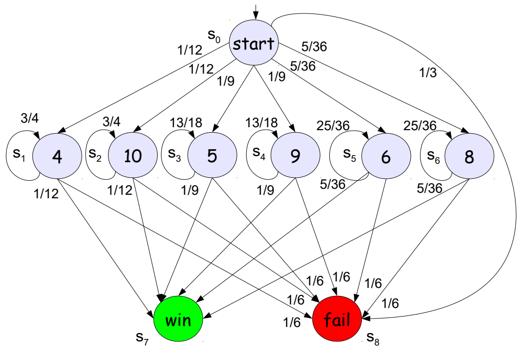


Figure 4.13: The DTMC for a version of the Craps game.

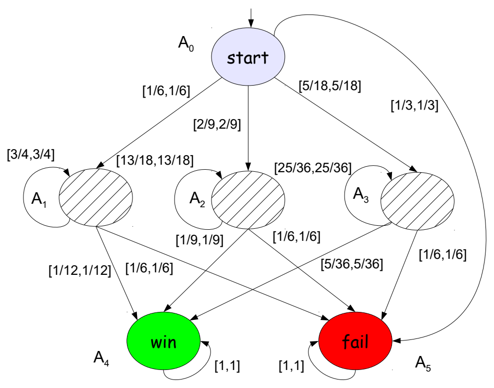


Figure 4.14: The ADTMC after the third refinement step for the Craps game.

The probability for winning the Craps game, i.e. reaching the *Win* state in Fig. 4.13 is approximately 0.27. In order to increase the probability to at least 0.3 for the player to win the game, we would like to find how much the dice should be biased each time they are rolled. Let us consider the atomic proposition q as $Win = true$, in which case the PCTL property of interest is $\varphi = P_{\geq 0.3}[Fq]$. The DTMC of Fig. 4.13 was eventually repaired after three refinement steps, which resulted in the ADTMC shown in Fig. 4.14. The repair solution given by the PAMR algorithm is shown in Fig. 4.15.

Zeroconf is a protocol for assigning IP addresses in a network of hosts. When a new host joins the network it asks the other hosts if the newly selected IP is already in use from any other host. There is a probability that the new host will not get any answer and in this case the query is repeated. The host will assume falsely that the chosen address is valid, if after n tries no answer is received.

The DTMC for an instance of the Zeroconf protocol is shown in Fig. 4.16a,

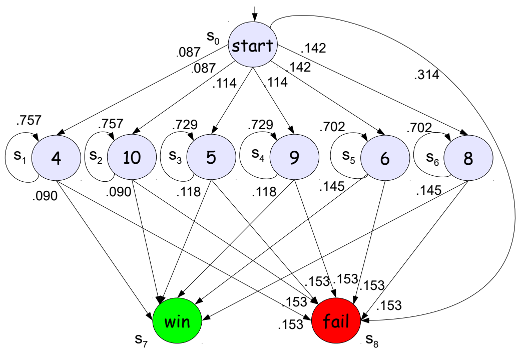
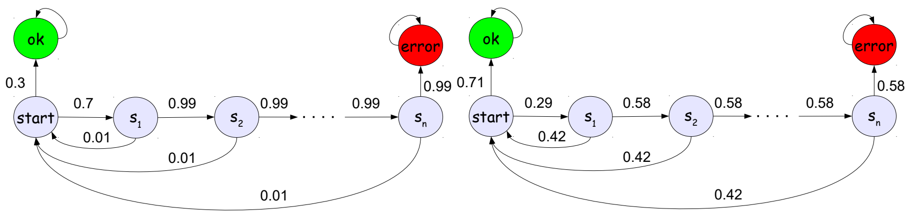


Figure 4.15: The repaired DTMC for the Craps game.

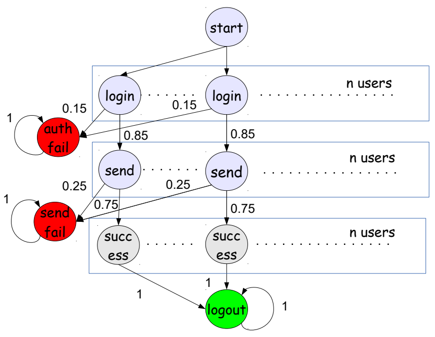


(a) The initial DTMC. (b) The repaired DTMC for $n = 10$.

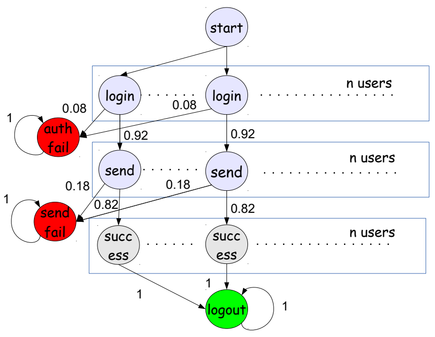
Figure 4.16: The IPv4 Zeroconf protocol

where the probability of not getting an answer is 0.99. We try to repair the DTMC for $n = 10$ tries, in order to ensure that the probability for reaching the OK state (which is 0.32) becomes at least 0.99. We consider the atomic proposition q as $OK = true$, in which case the reachability PCTL property of interest is $\varphi = P_{\geq 0.99}[Fq]$. The application of our PAMR framework resulted, after three refinement steps, to the repair solution presented in Fig. 4.16b.

The DTMC of the message authentication protocol for n users is depicted in Fig. 4.17a. An authentication process takes place for each user, which can fail with a probability of 0.15. After the successful authentication, each user



(a) The initial DTMC.



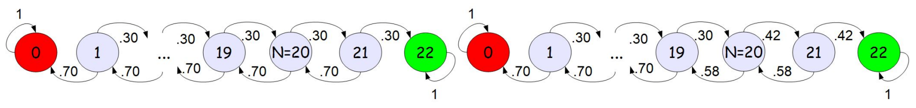
(b) The repaired DTMC for $n = 5$.

Figure 4.17: A message authentication protocol

can send a message with a failure probability of 0.25. The users log out from the system after having successfully sent a message. We applied our PAMR framework for $n = 5$ users, in order to get the repaired DTMC on which the probability for the users to log out is at least 0.75. A repaired DTMC is acquired after four refinement steps and is shown in Fig. 4.17b.

For the gambler's ruin model, let us consider a gambler who starts playing a game with initial wealth N coins and can bet at each time one coin. The gambler can win the bet with a probability p and can lose the bet with probability q. The game stops either when the gambler has no more money to bet or when he has earned a specific amount W. The DTMC for a version of the game with $N = 20$, $W = N \times 10\% = 2$, $p = 0.3$ and $q = 0.7$ is presented in Fig. 4.18a. The gambler's ruin model belongs to a special category of Markov Chains, called *birth-death*, where only a one-step transition to the nearest neighbors is permitted.

The probability for the gambler to earn the amount W is 0.183. After applying our PAMR framework with the aim of finding a repaired DTMC where the probability of the gambler to earn the amount W will be at least 0.25, we acquire the DTMC in Fig 4.18b.



(a) The initial DTMC. (b) The repaired DTMC

Figure 4.18: The gambler's ruin model for initial wealth $N = 20$.

Our experiments include extended versions for all the four models. For the Craps game, the model's state space is expanded by rolling more than two dices and also adapting the conditions for wining/losing the game. For the IPv4 Zeroconf protocol, for the message authentication protocol and for the gambler's ruin model, the state space is expanded by increasing the number of tries, the number of users and the initial wealth respectively. As shown in the Table 4.1, the state space size varies between a few hundred of states and transitions up to several tens of thousands states and transitions.

For all the models and for the same reachability properties, we applied our model repair algorithm directly to the DTMC (concrete model) and we compared its execution time and the repair solution's distance with that of the PAMR algorithm for the ADTMC. For the Craps game and the IPv4 Zeroconf protocol, we get a repair solution after three refinement steps, whereas for the message authentication protocol a solution is obtained after four refinement steps. For the gambler's ruin model the number of needed refinement

Models	Concr. States	Concr. Trans.	Concrete Repair	PAMR	Ref. Steps	Abstr. States	d_{concr}	d_{PAMR}
Craps (Ext.1)	123	483	$0.8s$	$0.1s$	3	6	7.4	10.4
Craps (Ext.2)	243	963	$5.1s$	$0.1s$	3	6	14.6	20.5
Craps (Ext.3)	483	1923	$38.3s$	$0.1s$	3	6	29	40.6
Craps (Ext.4)	963	3843	$5m13s$	$0.1s$	3	6	57.8	80.9
Craps (Ext.5)	1923	7683	$49m43s$	$0.1s$	3	6	115.4	161.6
Craps (Ext.6)	3843	15363	$8h30m23s$	$0.1s$	3	6	230.6	322.9
Craps (Ext.7)	7683	30723	*time-out*	$0.1s$	3	6	–	645.42
Craps (Ext.7)	7683	30723	*time-out*	$1.2s$	8	123	–	608.5
Craps (Ext.7)	7683	30723	*time-out*	$19.1s$	10	483	–	562.4
Craps (Ext.7)	7683	30723	*time-out*	$2h4m34s$	13	3843	–	488.7
Craps (Ext.7)	7683	30723	*time-out*	*time-out*	14	7683	–	–
Zconf (n=100)	103	204	$0.5s$	$0.2s$	3	5	10.1	82.8
Zconf (n=200)	203	404	$2.7s$	$0.2s$	3	5	17.7	164.8
Zconf (n=400)	403	804	$18.1s$	$0.2s$	3	5	28.9	328.8
Zconf (n=800)	803	1604	$2m38s$	$0.2s$	3	5	43.3	656.8
Zconf (n=1600)	1603	3204	$15m29s$	$0.2s$	3	5	51.2	1312.8
Zconf (n=3200)	3203	6404	$2h36m12s$	$0.2s$	3	5	57.6	2624.8
Zconf (n=6400)	6403	12804	*time-out*	$0.2s$	3	5	–	5248.2
Zconf (n=6400)	6403	12804	*time-out*	$2.1s$	10	104	–	4224.6
Zconf (n=6400)	6403	12804	*time-out*	$14.9s$	12	404	–	2048.3
Zconf (n=6400)	6403	12804	*time-out*	$1h36m43s$	15	3204	–	512.1
Zconf (n=6400)	6403	12804	*time-out*	*time-out*	16	6403	–	–
Auth.Prot.(n=30)	154	213	$1.2s$	$0.2s$	4	7	8.4	8.4
Auth.Prot.(n=60)	304	423	$7.8s$	$0.2s$	4	7	16.8	16.8
Auth.Prot.(n=120)	604	843	$57.5s$	$0.2s$	4	7	33.6	33.6
Auth.Prot.(n=240)	1204	1683	$9m43s$	$0.2s$	4	7	67.2	67.2
Auth.Prot.(n=480)	2404	3363	$1h12m11s$	$0.2s$	4	7	134.4	134.4
Auth.Prot.(n=960)	4804	6723	$12h19m52s$	$0.2s$	4	7	268.8	268.8
Auth.Prot.(n=1920)	9604	13443	*time-out*	$0.2s$	4	7	–	537.6
Auth.Prot.(n=1920)	9604	13443	*time-out*	$2.2s$	9	131	–	537.6
Auth.Prot.(n=1920)	9604	13443	*time-out*	$20s$	11	515	–	537.6
Auth.Prot.(n=1920)	9604	13443	*time-out*	$1h54m32s$	14	4099	–	537.6
Auth.Prot.(n=1920)	9604	13443	*time-out*	*time-out*	15	9604	–	–
Gambler(N=100)	111	220	$0.8s$	$0.2s$	4	58	0.38	2.93
Gambler(N=200)	221	440	$4.6s$	$0.9s$	5	113	0.84	5.94
Gambler(N=400)	441	880	$33.3s$	$4.8s$	6	223	2.48	12.28
Gambler(N=800)	881	1760	$4m59s$	$32.7s$	7	443	5.92	25.04
Gambler(N=1600)	1761	3520	$45m12s$	$4m48s.$	8	883	13.44	50.88
Gambler(N=3200)	3521	7040	$8h2m41s$	$45m17s$	9	1763	30.72	103.36
Gambler(n=7400)	7041	14080	*time-out*	$6h58m36s$	10	3523	–	207.36

Table 4.1: Experimental results of PAMR compared to concrete repair (in fourth column the accumulated times are shown, for all iterated refinement steps)

steps varies from 4 to 10 according to the size of the model. The results in Table 4.1 suggest that as the model's state space grows up, the needed time for the concrete model repair is increased dramatically or even a solution cannot be obtained within a time-out period of 14 hours, while the PAMR algorithm returns a repair solution efficiently. It is worth mentioning that the time for the PAMR algorithm remains constant for the Craps game, the IPv4 Zeroconf protocol and the message authentication protocol, because in all extended versions the repair solution was obtained from ADTMCs with the same size of state space after three or four refinement steps respectively. This is not true for the gambler's ruin model, where the repair solution is always acquired in the final refinement step, before converging to the concrete DTMC. These results confirm the fact that the efficiency gains of abstraction in model repair depends - as in model checking - on the model's state space structure and the property.

For all models except for the gambler's ruin model, we continued the application of our PAMR framework, after having found the first repair solution, until the PAMR converged to be applied to the initial concrete model, in which case the ADTMC would be the same with the DTMC. From the results presented in Table 4.1, we deduce that for all the examined models, our method gives a repair solution and it is significantly more efficient than the direct repair even for the ADTMC produced in the penultimate refinement step.

4.6 Related Work

The model repair problem for probabilistic systems has been first introduced in [13]. The authors use parametric model checking and transform the model repair problem to a non-linear optimization problem. The experimental results of their work indicates that their approach suffers from the state space explosion problem even for models with relatively small number of states. In comparison with [13], beyond the efficiency advantage of the PAMR framework, it is true that we do not aim to a direct repair solution, if any, but to gradually approach a suitable solution through a number of consecutive refinements that depends on the available computational resources. In such an iterative process, the designer can interactively develop his repair strategy, as described in Section 4.4.1, whereas in [13] the repair strategy is specified at once through the definition of a so-called controllable DTMC over a set of parameters that has to fulfill the stochasticity condition. However, such a parameterization approach is not easily applied in all state space topologies.

An effort for presenting a scalable method for the repair of DTMCs is also presented in [75]. In that work, the authors present a greedy approach where, starting from an initial parameter assignment, they apply local repair steps by iteratively changing the parameter values. The execution time for the local repairs is reduced with respect to the methods using non-linear optimization, but the fact that this approach includes a model-checking phase eventually increases the total execution time.

Regarding the model repair of Markov Decision Processes (MDPs), there are approximate solution techniques, which have been introduced in [31].

Several research projects consider the model repair problem as part of a solution to the formal verification and *fault recovery* of applications of real-world domains, such as the aerospace systems [21]. The first works on model repair in the context of temporal logics, for non-probabilistic systems, have been presented in [6] and [23]. In [4], the authors attempt three different model repair techniques to efficiently compute the probability distribution for the minimum recovery time of an input randomized distributed self-stabilizing protocol. The first approach is based on solving symbolic linear algebraic equations, in order to identify the fastest state reachability in parametric discrete-time Markov chains. Their second approach applies parameter synthesis techniques to compute the rational function describing the average recovery time, like in [13]. In their third approach, they focus on finding sub-otpimal solutions by computing over- and under-approximations of the result for a given parameter region and iteratively refining the regions with minimal recovery time up to the desired precision. Yet, all of the proposed techniques still suffer from the state space explosion problem.

For the abstraction of probabilistic models, some more techniques have been proposed apart from the adopted abstraction method with 3-valued semantics. In [66], the authors present an abstraction method for Markov Decision processes based on games. In [46], the authors propose a *model aggregation* technique to construct effectively the lumping quotient of a Markov chain. In [10], the authors examine the minimal distance problem of Markov chains using bisimulation metrics, i.e., given a finite DTMC and a positive integer k, they try to find the k-state DTMC with the minimal distance to the original. We note that in our abstraction framework based on [64], the abstract model is connected to the concrete model with a simulation – not with a (more restrictive) bisimulation – relation. A thorough presentation of the abstraction methods used for probabilistic systems is given in [43]. We are not aware of any other model repair approach, which exploits an abstraction technique for probabilistic systems.

However, abstraction has been used in model repair for systems with large state spaces, in the non-probabilistic setting [30]. In this work, the authors present an abstraction-refinement framework based on Kripke Structures as the concrete models, Kripke Modal Transition Systems as the abstract models, CTL as the specification language and 3-valued model checking. Their aim is that of reducing the upper bound complexity class of the repair process, such that it depends on the size of a much smaller abstract model. A more complete journal version of this work is presented in [29].

Some related work exists regarding the *parametric analysis* of Markov models. More specifically, an interesting and related to model repair problem is that of *parameter synthesis*, where the aim is to find ranges of parameter values such that a satisfaction probability of a formula meets a given threshold, is maximized, or minimized. In [12], the authors address the system design problem for Continuous Time Markov Chains (CTMCs), where their aim is to optimize some parameters of a stochastic model to maximize robustness of some given specifications. The state space explosion problem is also inherent in [27], where a tool is introduced using GPU-accelerated parameter synthesis to make the approach scalable. This work addresses the parallel implementation of the parameter space decomposition technique of [22]. A recent work [24] deals with the synthesis problem for CTMCs within the scope of optimizing specific quality criteria. Moreover, a tool called PROPhESY has been presented in [44], for scalable, incremental, and automatic parameter synthesis. Another tool called SEA-PARAM has been presented in [7], where the authors propose a parameter synthesis method for Parametric Markov Decision Processes (PMDPs).

Finally, the authors in [22] have proposed a method for parameter interval decomposition for CSL formulas in CTMCs. This method is a means to reduce the inaccuracy created from a min-max approximation. If this idea could be lifted to DTMCs, it is a promising tool that could be adopted in our abstraction-refinement framework.

Chapter 5

Conclusions and Future Work

5.1 Advancements in relation to the state of the art

In this thesis, we have shown how abstraction can be used to cope with the state explosion problem in model repair. Two frameworks regarding Kripke structures and probabilistic systems have been presented which can give solutions to the model repair problem for models with large state spaces.

The model-repair framework for the non-probabilistic models 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. It is proved that our AMR algorithm is sound for the full CTL and complete for a subset of CTL. It is 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, the framework framework has been applied to an Automatic Door Opener system and to the Andrew File System 1 protocol.

Regarding the probabilistic models, this thesis has presented an abstraction refinement framework and algorithms for the repair of DTMCs with respect to reachability PCTL properties. The so-called PAMR framework and its algorithm aim to confront the state space explosion problem and to provide a solution that is applicable even in model repair problems with very large state spaces. After having described the PAMR framework, this thesis illustrates the efficiency gains of the approach compared to the direct repair of the concrete model, including the discussion of custom repair strategies depending on the particular application domain. The practical utility of the PAMR framework was demonstrated in the repair of four DTMC models with diverse structures of their state space.

The abstract model repair frameworks presented in this thesis outweigh all the existing concrete model repair approaches in terms of efficiency, in the sense that all the repair operations take place in an abstract model with significantly smaller state space. Despite the fact that the main objective of this thesis is to make repair feasible to models with large state spaces, there also other contributions which can make the repair process in the future more effective and flexible.

At first, our abstract model repair framework for probabilistic systems bypasses the use of parametric model checking which has been used in existing model repair approaches and introduced a large efficiency blow-up together with unnecessary complexity. All probabilistic methods can gain from this general principle presented in this thesis.

Moreover, the use of abstraction and refinement in model repair can transform the process for searching of a repair solution from a monolithic procedure to an interactive framework. This can happen because from a repaired abstract model many concrete repaired models can be produced. Additionally, different refinement methods can lead to a set of different repaired models. Thus, the user can interact with the repair process before getting a repair solution and being this way, the one who can choose the repaired model which fits best to his design specifications.

Another contribution of this work which can be crucial in the future works on the model repair problem, is the introduction of constraint properties in the repair algorithm. Having in mind that a strong objective of the model repair problem is the minimality of changes of the repair solution, we can conclude that the support of constraint properties in repair algorithms will guarantee that any repaired model should not violate properties that already satisfied in the initial model. That is to say, this thesis can be the first step for introducing the logical minimality of changes instead of or together with the structural minimality supported up to now from the distance metrics on the concrete models.

5.2 Future research prospects

Many different lines of research can be followed as future prospects of the work presented in this thesis. These prospects contain improvements of the current abstract model repair framework, applications to different domains and extensions to other contexts.

Firstly, I strongly consider that abstract model repair can gain a lot from being implemented in a mainstream model checker such as SMV, PRISM or STORM. This way, abstract model repair could be applied in larger, real-world and industrial systems and the feedback from this application could result in better understanding its performance and effectiveness. For example, a model repair application of high interest could be the design of fault-tolerant systems. In [20], 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 does not include a set of uncontrollable transitions including 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 benefit from the use of abstraction in our framework.

Another very interesting extension of this thesis could be the search for “better" abstract models, in order to either restrict failures due to refinement or ensure completeness for a larger fragment of CTL or PCTL properties. Currently, most of the abstraction frameworks are created for model checking and not for model repair. It would be really interesting to search for abstract models which will be optimized for being components of a repair process.

A very promising approach could be the exploit of the degrees of freedom provided to the user from the PAMR framework, i.e. abstraction, refinement, constraint properties, to transform the model repair methods from monolithic tools to an interactive design framework.

Finally, an attractive prospect for future research should include efforts for adopting abstraction in model repair frameworks for different contexts such as hybrid and real-time systems.

BIBLIOGRAPHY

[1] Maple. Accessed: 2017-06-22.

[2] 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.

[3] Saba Aflaki, Mathias Volk, Borzoo Bonakdarpour, Joost-Pieter Katoen, and Arne Storjohann. Automated fine tuning of probabilistic self-stabilizing algorithms. In *Reliable Distributed Systems (SRDS), 2017 IEEE 36th Symposium on*, pages 94–103. IEEE, 2017.

[4] Saba Aflaki, Matthias Volk, Borzoo Bonakdarpour, Joost-Pieter Katoen, and Arne Storjohann. Automated fine tuning of probabilistic self-stabilizing algorithms. In SRDS, 2017.

[5] 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.

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

[7] Sebastian Arming, Ezio Bartocci, and Ana Sokolova. Sea-param: Exploring schedulers in parametric mdps. *Quantitative Aspects of Programming Languages and Systems 2017*, 2017.

[8] Paul C. Attie, Kinan Dak Al Bab, and Mouhammad Sakr. Model and program repair via sat solving. *ACM Trans. Embed. Comput. Syst.*, 17(2):32:1–32:25, December 2017.

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

[10] Giovanni Bacci, Giorgio Bacci, Kim G. Larsen, and Radu Mardare. On the Metric-Based Approximate Minimization of Markov Chains. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 104:1–104:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

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

[12] Ezio Bartocci, Luca Bortolussi, Laura Nenzi, and Guido Sanguinetti. System design of stochastic models using robustness of temporal properties. *Theoretical Computer Science*, 587:3–25, 2015.

[13] 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.

[14] Paul T Boggs and Jon W Tolle. Sequential quadratic programming. *Acta numerica*, 4:1–51, 1995.

[15] 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.

[16] 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.

[17] Borzoo Bonakdarpour, Reza Hajisheykhi, and Sandeep S. Kulkarni. Knowledge-based automated repair of authentication protocols. In *Pro-*

ceedings of the 19th International Symposium on FM 2014: Formal Methods - Volume 8442, pages 132–147, New York, NY, USA, 2014. Springer-Verlag New York, Inc.

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

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

[20] 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.

[21] Marco Bozzano, Harold Bruintjes, Alessandro Cimatti, Joost-Pieter Katoen, Thomas Noll, and Stefano Tonetta. Formal methods for aerospace systems. In *Cyber-Physical System Design from an Architecture Analysis Viewpoint*, pages 133–159. Springer, 2017.

[22] Luboš Brim, Milan Češka, Sven Dražan, and David Šafránek. Exploring parameter space of stochastic biochemical systems using quantitative model checking. In *Proceedings of the 25th International Conference on Computer Aided Verification*, CAV'13, pages 107–123, Berlin, Heidelberg, 2013. Springer-Verlag.

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

[24] Radu Constantin Calinescu, Milan Ceska, Simos Gerasimou, Marta Kwiatkowska, and Nicola Paoletti. Designing robust software systems through parametric markov chain synthesis. In *IEEE International Conference on Software Architecture (ICSA 2017)*. IEEE, 2017.

[25] 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.

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

[27] Milan Češka, Petr Pilař, Nicola Paoletti, Luboš Brim, and Marta Kwiatkowska. Prism-psy: precise gpu-accelerated parameter synthesis for stochastic systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 367–384. Springer, 2016.

[28] Rohit Chadha and Mahesh Viswanathan. A counterexample-guided abstraction-refinement framework for markov decision processes. *ACM Transactions on Computational Logic (TOCL)*, 12(1):1, 2010.

[29] George Chatzieleftheriou, Borzoo Bonakdarpour, Panagiotis Katsaros, and Scott A. Smolka. Abstract Model Repair. *Logical Methods in Computer Science*, Volume 11, Issue 3, Sep 2015.

[30] 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.

[31] Taolue Chen, Ernst Moritz Hahn, Tingting Han, Marta Kwiatkowska, Hongyang Qu, and Lijun Zhang. Model repair for markov decision processes. In *Theoretical Aspects of Software Engineering (TASE), 2013 International Symposium on*, pages 85–92. IEEE, 2013.

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

[33] 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.

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

[35] 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.

[36] 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.

[37] 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.

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

[39] Pedro R D argenio, Bertrand Jeannet, Henrik E Jensen, and Kim G Larsen. Reachability analysis of probabilistic systems by successive re-

finements. In *Process Algebra and Probabilistic Methods. Performance Modelling and Verification*, pages 39–56. Springer, 2001.

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

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

[42] Maria de Menezes, Silvio do Lago Pereira, and Leliane de Barros. System design modification with actions. In Antônio 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.

[43] Christian Dehnert, Daniel Gebler, Michele Volpato, and David N. Jansen. On abstraction of probabilistic systems. In *Advanced Lectures of the International Autumn School on Stochastic Model Checking. Rigorous Dependability Analysis Using Model Checking Techniques for Stochastic Systems - Volume 8453*, ROCKS 2012, pages 87–116, New York, NY, USA, 2014. Springer-Verlag New York, Inc.

[44] Christian Dehnert, Sebastian Junges, Nils Jansen, Florian Corzilius, Matthias Volk, Harold Bruintjes, Joost-Pieter Katoen, and Erika Abraham. Prophesy: A probabilistic parameter synthesis tool. In *Inter-*

national Conference on Computer Aided Verification, pages 214–231. Springer, 2015.

[45] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, pages 592–600, 2017.

[46] Salem Derisavi, Holger Hermanns, and William H. Sanders. Optimal state-space lumping in markov chains. *Inf. Process. Lett.*, 87(6):309–315, sep 2003.

[47] 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.

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

[49] 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.

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

[51] Antonio Filieri, Carlo Ghezzi, and Giordano Tamburrelli. Run-time efficient probabilistic model checking. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 341–350, New York, NY, USA, 2011. ACM.

[52] 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.

[53] 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.

[54] 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.

[55] Susanne Graf and Hassen Saidi. Construction of abstract state graphs with pvs. In Orna Grumberg, editor, *Computer Aided Verification*, vol-

ume 1254 of *Lecture Notes in Computer Science*, pages 72–83. Springer Berlin / Heidelberg, 1997.

[56] 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.

[57] 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.

[58] 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.

[59] 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.

[60] Holger Hermanns, Björn Wachter, and Lijun Zhang. Probabilistic cegar. In *International Conference on Computer Aided Verification*, pages 162–175. Springer, 2008.

[61] 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.

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

[63] 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.

[64] Joost-Pieter Katoen, Daniel Klink, Martin Leucker, and Verena Wolf. Three-valued abstraction for probabilistic systems. *The Journal of Logic and Algebraic Programming*, 81(4):356–389, 2012.

[65] Mark Kattenbelt, Marta Kwiatkowska, Gethin Norman, and David Parker. Abstraction refinement for probabilistic software. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 182–197. Springer, 2009.

[66] Mark Kattenbelt, Marta Kwiatkowska, Gethin Norman, and David Parker. A game-based abstraction-refinement framework for markov

decision processes. *Formal Methods in System Design*, 36(3):246–280, 2010.

[67] 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.

[68] Anvesh Komuravelli, Corina S Păsăreanu, and Edmund M Clarke. Assume-guarantee abstraction refinement for probabilistic systems. In *International Conference on Computer Aided Verification*, pages 310–326. Springer, 2012.

[69] M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. Performance analysis of probabilistic timed automata using digital clocks. *Formal Methods in System Design*, 29:33–78, 2006.

[70] Marta Kwiatkowska, Gethin Norman, and David Parker. Stochastic model checking. In *SFM*, volume 7, pages 220–270. Springer, 2007.

[71] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In *Proceedings of the 23rd International Conference on Computer Aided Verification*, CAV'11, pages 585–591, Berlin, Heidelberg, 2011. Springer-Verlag.

[72] Craig T Lawrence and André L Tits. A computationally efficient feasible sequential quadratic programming algorithm. *Siam Journal on optimization*, 11(4):1092–1118, 2001.

[73] 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.

[74] 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.

[75] Shashank Pathak, Erika Abraham, 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.

[76] Mohammad Roohitavaf, Yiyan Lin, and Sandeep S Kulkarni. Lazy repair for addition of fault-tolerance to distributed programs. In *Parallel and Distributed Processing Symposium, 2016 IEEE International*, pages 1071–1080. IEEE, 2016.

[77] 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.

[78] 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.

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

[80] 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.

[81] Jana Tumova, Luis I Reyes Castro, Sertac Karaman, Emilio Frazzoli, and Daniela Rus. Minimum-violation ltl planning with conflicting specifications. In *American Control Conference (ACC), 2013*, pages 200–205. IEEE, 2013.

[82] 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.

[83] Stephen A Vavasis. Quadratic programming is in np. *Information Processing Letters*, 36(2):73–77, 1990.

[84] 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.

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

[86] 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.

[87] 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.

[88] 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.

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

[90] 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.