Model Repair via SAT Solving

Paul Attie and Mohamad Sakr
Department of Computer Science
American University of Beirut
paul.attie@aub.edu.lb, mis14@mail.aub.edu

February 24, 2015

Abstract

We consider the *model repair problem*: given a finite Kripke structure M and a CTL formula η, determine if M contains a substructure M' that satisfies η. Thus, M can be "repaired" to satisfy η by deleting some transitions.

We map an instance (M, η) of model repair to a boolean formula $\mathit{repair}(M, \eta)$ such that (M, η) has a solution iff $\mathit{repair}(M, \eta)$ is satisfiable. Furthermore, a satisfying assignment determines which states and transitions must be removed from M to generate a model M' of η. Thus, we can use any SAT solver to repair Kripke structures. Using a complete SAT solver yields a complete algorithm: it always finds a repair if one exists. We also show that CTL model repair is NP-complete.

We also give some optimizations, for special cases, and extend the basic repair method in two directions: (1) the use of abstraction mappings, i.e., repair a structure abstracted from M and then concretize the resulting repair to obtain a repair of M, and (2) the repair of hierarchical Kripke structures, by using a CTL formula to summarize the behavior of each "box", and CTL deduction to relate the box formula with the overall specification.

1 Introduction and Motivation

Counterexample generation in model checking produces an example behavior that violates the formula being checked, and so facilitates debugging the model. However, there could be many counterexamples, and they may have to be dealt with by making different fixes manually, thus increasing debugging effort. In this paper we deal with all counterexamples at once, by "repairing" the model: we present a method for automatically fixing Kripke structures with respect to CTL [15] specifications.

Our contribution. We present a "subtractive" repair algorithm: fix a Kripke structure only by removing transitions and states (roughly speaking, those transitions and states that "cause" violation of the specification). If the initial state is not deleted, then the resulting structure satisfies the specification. We show that this algorithm is sound and relatively complete. An advantage of subtractive repair is that it does not introduce new behaviors, and thus any missing (i.e., not part of the formula being repaired against) conjuncts of the specification that are expressible in ACTL* [16], i.e., CTL* without the existential path quantifier, are still satisfied, if they originally were. Hence we can fix w.r.t. incomplete specifications.

We extend the basic repair method in several directions: (1) the addition of states and transitions, and the modification of the atomic propositions that hold in a state (i.e., the propositional labeling function of the Kripke structure), and (2) the use of abstraction mappings, i.e., repair a structure abstracted from M and then concretize the resulting repair to obtain a repair of M, and (3) the repair of hierarchical Kripke structures [1], by using a CTL formula to summarize the behavior of each "box", and CTL deduction to relate the box formula with the overall specification, and (4) the repair of concurrent Kripke structures, i.e., several Kripke structures that are composed in parallel and that synchronize on common operations [2].

Finally, we show that the model repair problem is NP-complete. We have implemented the repair method as a GUI-based tool, Eshmun, available at `http://eshmuntool.blogspot.com/`, which also provides a user manual. All of the examples in this paper were produced using Eshmun.

Formally, we consider the *model repair problem*: given a Kripke structure M and a CTL or ATL formula η, does there exist a substructure M' of M (obtained by removing transitions and states from M) such that M' satisfies η? In this case, we say that M is *repairable* with respect to η, or that a repair exists.

Our algorithm computes (in deterministic time polynomial in the sizes of M and η) a propositional formula $repair(M, \eta)$ such that $repair(M.\eta)$ is satisfiable iff M contains a substructure M' that satisfies η. Furthermore, a satisfying assignment for $repair(M, \eta)$ determines which transitions must be removed from M to produce M'. Thus, a single run of a complete SAT solver is sufficient to find a repair, if one exists.

Soundness of our repair algorithm means that the resulting M' (if it exists) satisfies η. Completeness means that if the initial structure M contains a substructure that satisfies η, then our algorithm will find such a substructure, provided that a complete SAT solver is used to check satisfaction of $repair(M, \eta)$.

While our method has a worst case running time exponential in the number of global states, this occurs only if the underlying SAT solver uses exponential time. SAT-solvers have proved to be efficient in practice, as demonstrated by the success of SAT-solver based tools such as Alloy, NuSMV, and Isabelle/HOL. The success of SAT solvers in practice indicates that our method will be applicable to reasonable size models, just as, for example, Alloy [19] is. Furthermmore, our method is intended for the construction of models that serve as specifications, and which can be coinsrtructed interactively by a user. By definition, these cannot be flat Kripke structures of thousands of states or more, since a user cannot handle this. We do however, deal with state-explosion issues by providing for hierarchical Kripke structures, and concurrently composed Kripke structures. These correspond to arbitrarily large flat structures, but require only a linear increase in the size of the formulae that the SAT solver must handle. Furthermore, they can be handled interactively by a user, since each component Kripke structure is small.

The rest of the paper is as follows. Section 2 provides brief technical preliminaries. Section 3 states formally the CTL model repair problem and shows that it is NP-complete. Section 4 presents our model repair method and shows that it is a sound and complete solution to the model repair problem. Section 5 presents two example applications of our method: mutual exclusion and barrier synchronization. Section 6 extends the methjod to deal with the addition of states and transitions, and the modification of the atomic propositions that hold in a state. It also gives some optimizaitons that improve the running time in some cases, shows how to increase the expressiveness of repairs by using generalized boolean constraints, and discusses state-explosion. Section 7 shows how to reapir abstract strucutres and then concretize the

resulting repair to obtain a repair of the original structure. Section 8 shows how to repair hierarchical Kripke structures. Section 9 discusses the repair of concurrent Kripke structures. Section 10 discusses our implementation, the Eshmun tool. Section 11 discusses related work. Section 12 concludes.

2 Preliminaries: Computation Tree Logic and Kripke Structures

Let AP be a set of atomic propositions, including the constants true and false. We use true, false as "constant" propositions whose interpretation is always the semantic truth values tt, ff, respectively. The logic CTL [14,15] is given by the following grammar:

$$\varphi ::= \mathsf{true} \mid \mathsf{false} \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathsf{AX}\varphi \mid \mathsf{EX}\varphi \mid \mathsf{A}[\varphi \mathsf{V} \varphi] \mid \mathsf{E}[\varphi \mathsf{V} \varphi]$$

where $p \in AP$, and $\mathsf{true}, \mathsf{false}$ are constant propositions with interpretation tt, ff respectively (i.e., "syntactic" true, false respectively).

The semantics of CTL formulae are defined with respect to a Kripke structure.

Definition 1. *A Kripke structure is a tuple $M = (S_0, S, R, L, AP)$ where S is a finite state of states, $S_0 \subseteq S$ is a set of initial states, $R \subseteq S \times S$ is a transition relation, and $L : S \mapsto 2^{AP}$ is a labeling function that associates each state $s \in S$ with a subset of atomic propositions, namely those that hold in the state. State t is a* sucessor *of state s in M iff $s, t \in R$.*

We assume that a Kripke structure $M = (S_0, S, R, L, AP)$ is total, i.e., $\forall s \in S, \exists s' \in S : (s, s') \in R$. A path in M is a (finite or infinite) sequence of states, $\pi = s_0, s_1, \ldots$ such that $\forall i \geq 0 : (s_i, s_{i+1}) \in R$. A fullpath is an infinite path. A state is reachable iff it lies on a path that starts in an initial state. Without loss of generality, we assume in the sequel that the Kripke structure M that is to be repaired does not contain any unreachable states, i.e., every $s \in S$ is reachable.

Definition 2. *$M, s \models \varphi$ means that formula φ is true in state s of structure M and $M, s \not\models \varphi$ means that formula φ is false in state s of structure M. We define $\models$ inductively as usual:*

1. $M, s \models \mathsf{true}$
2. $M, s \not\models \mathsf{false}$
3. $M, s \models p$ *iff* $p \in L(s)$ *where atomic proposition* $p \in AP$
4. $M, s \models \neg\varphi$ *iff* $M, s \not\models \varphi$
5. $M, s \models \varphi \wedge \psi$ *iff* $M, s \models \varphi$ *and* $M, s \models \psi$
6. $M, s \models \varphi \vee \psi$ *iff* $M, s \models \varphi$ *or* $M, s \models \psi$
7. $M, s \models \mathsf{AX}\varphi$ *iff for all t such that $(s, t) \in R : (M, t) \models \varphi$*
8. $M, s \models \mathsf{EX}\varphi$ *iff there exists t such that $(s, t) \in R$ and $(M, t) \models \varphi$*
9. $M, s \models \mathsf{A}[\varphi \mathsf{V} \psi]$ *iff for all fullpaths $\pi = s_0, s_1, \ldots$ starting from $s = s_0$:*
   $\forall k \geq 0 : (\forall j < k : (M, s_j \not\models \varphi)$ *implies* $M, s_k \models \psi$

10. $M, s \models \mathsf{E}[\varphi \mathsf{V} \psi]$ *iff for some fullpath* $\pi = s_0, s_1, \ldots$ *starting from* $s = s_0$*:*
$\forall k \geq 0 : (\forall j < k : (M, s_j \not\models \varphi)$ *implies* $M, s_k \models \psi$

We use $M \models \varphi$ to abbreviate $M, S_0 \models \varphi$. We introduce the abbreviations $\mathsf{A}[\phi \mathsf{U} \psi]$ for $\neg\mathsf{E}[\neg\varphi \mathsf{V} \neg\psi]$, $\mathsf{E}[\phi \mathsf{U} \psi]$ for $\neg\mathsf{A}[\neg\varphi \mathsf{V} \neg\psi]$, $\mathsf{AF}\varphi$ for $\mathsf{A}[\mathsf{true}\mathsf{U}\varphi]$, $\mathsf{EF}\varphi$ for $\mathsf{E}[\mathsf{true}\mathsf{U}\varphi]$, $\mathsf{AG}\varphi$ for $\mathsf{A}[\mathsf{false}\mathsf{V}\varphi]$, $\mathsf{EG}\varphi$ for $\mathsf{E}[\mathsf{false}\mathsf{V}\varphi]$.

Definition 3 (Formula expansion)**.** *Given a CTL formula* φ*, its set of subformulae* $sub(\varphi)$ *is defined as follows:*

- $sub(p) = p$ *where* p *is* true*,* false*, or an atomic proposition*
- $sub(\neg\varphi) = \{\neg\varphi\} \cup sub(\varphi)$
- $sub(\varphi \wedge \psi) = \{\varphi \wedge \psi\} \cup sub(\varphi) \cup sub(\psi)$
- $sub(\varphi \vee \psi) = \{\varphi \vee \psi\} \cup sub(\varphi) \cup sub(\psi)$
- $sub(\mathsf{AX}\varphi) = \{\mathsf{AX}\varphi\} \cup sub(\varphi)$
- $sub(\mathsf{EX}\varphi) = \{\mathsf{EX}\varphi\} \cup sub(\varphi)$
- $sub(\mathsf{A}[\varphi \mathsf{V} \psi]) = \{\mathsf{A}[\varphi \mathsf{V} \psi], \varphi, \psi\} \cup sub(\varphi) \cup sub(\psi)$
- $sub(\mathsf{E}[\varphi \mathsf{V} \psi]) = \{\mathsf{E}[\varphi \mathsf{V} \psi], \varphi, \psi\} \cup sub(\varphi) \cup sub(\psi)$

3 The Subtractive Model Repair Problem

Given Kripke structure M and a CTL formula η, we consider the problem of removing parts of M, resulting in a substructure M' such that $M' \models \eta$.

Definition 4 (Substructure)**.** *Given Kripke structures* $M = (S_0, S, R, L, AP)$ *and* $M' = (S'_0, S', R', L', AP)$ *we say that* M' *is a* substructure *of* M*, denotedv*$M' \subseteq M$*, iff* $S'_0 \subseteq S_0$*,* $S' \subseteq S$*,* $R' \subseteq R$*, and* $L' = L \restriction S'$*.*

Definition 5 (Repairable)**.** *Given Kripke structure* $M = (S_0, S, R, L, AP)$ *and CTL formula* η*.* M *is* repairable *with respect to* η *if there exists a Kripke structure* $M' = (S'_0, S', R', L', AP)$ *such that* M' *is total,* $M' \subseteq M$*, and* $M', S'_0 \models \eta$*.*

Definition 6 (Model Repair Problem)**.** *For Kripke structure* M *and CTL formula* η*, we use* $\langle M, \eta \rangle$ *for the corresponding repair problem. The decision version of repair problem* $\langle M, \eta \rangle$ *is to decide if* M *is repairable w.r.t.* η*. The functional version of repair problem* $\langle M, \eta \rangle$ *is to return an* M' *that satisfies Def. 5, in the case that* M *is repairable w.r.t.* η*.*

3.1 Complexity of the Model Repair Problem

Theorem 1. *The decision version of the model repair problem is NP-complete.*

Proof. Let (M, η) be an arbitrary instance of the CTL model repair problem.

NP-membership: Given a candidate solution $M' = (S'_0, S', R', L', AP)$, the condition $M' \subseteq M$ is easily verified in polynomial time. $M', S'_0 \models \eta$ is verified in linear time using the CTL model checking algorithm of [11].

NP-hardness: We reduce 3SAT to CTL model repair.

Given a Boolean formula $f = \bigwedge_{1 \leq i \leq n}(a_i \vee b_i \vee c_i)$ in 3cnf, where a_i, b_i, c_i are literals over the a set $x_1, \ldots, x_m$ of propositions, i.e each of a_i, b_i, c_i is x_j or $\neg x_j$, for some $j \in 1 \ldots m$. We reduce f to (M, η) where $M = (\{s_0\}, S, R, L, AP)$, $S = \{s_0, s_1, \ldots, s_m, t_1, \ldots, t_m\}$, and $R = \{(s_0, s_1), \ldots, (s_0, s_m), (s_1, t_1), \ldots, (s_m, t_m)\}$, i.e., transitions from s_0 to each of $s_1, \ldots, s_m$, and a transition from each s_i to t_i for $i = 1, \ldots, m$. The underlying set AP of atomic propositions is $\{p_1, \ldots p_m, q_1, \ldots, q_m\}$. These propositions are distinct from the $x_1, \ldots, x_m$ used in the 3cnf formula f. L is given by:

- $L(s_0) = \emptyset$
- $L(s_j) = p_j$ where $1 \leq j \leq m$
- $L(t_j) = q_j$ where $1 \leq j \leq m$

η is given by:

$$\eta = \bigwedge_{1 \leq i \leq n} (\varphi_i^1 \vee \varphi_i^2 \vee \varphi_i^3)$$

where:

- if $a_i = x_j$ then $\varphi_i^1 = \mathsf{AG}(p_j \Rightarrow \mathsf{EX}q_j)$
- if $a_i = \neg x_j$ then $\varphi_i^1 = \mathsf{AG}(p_j \Rightarrow \mathsf{AX}\neg q_j)$
- if $b_i = x_j$ then $\varphi_i^2 = \mathsf{AG}(p_j \Rightarrow \mathsf{EX}q_j)$
- if $b_i = \neg x_j$ then $\varphi_i^2 = \mathsf{AG}(p_j \Rightarrow \mathsf{AX}\neg q_j)$
- if $c_i = x_j$ then $\varphi_i^3 = \mathsf{AG}(p_j \Rightarrow \mathsf{EX}q_j)$
- if $c_i = \neg x_j$ then $\varphi_i^3 = \mathsf{AG}(p_j \Rightarrow \mathsf{AX}\neg q_j)$

Thus, if $a_i^1 = x_i$, then the transition from s_i to t_i (which we write as $s_i \rightarrow t_i$ must be retained in M', and if $a_i = \neg x_i$, then the transition $s_i \rightarrow t_i$ must not appear in M'. It is obvious that the reduction can be computed in polynomial time.
It remains to show that:

f is satisfiable iff (M, η) can be repaired. The proof is by double implication.

f is satisfiable implies that (M, η) can be repaired: Let $\mathcal{V} : \{x_1, \ldots, x_m\} \mapsto \{tt, ff\}$ be a satisfying truth assignment for f. Define R' as follows. $R' = \{(s_0, s_i), (s_i, s_i), (t_i, t_i) \mid 1 \leq i \leq m\} \cup \{(s_i, t_i) \mid \mathcal{V}(x_i) = tt\}$, i.e., the transition $s_i \rightarrow t_i$ is present in M' if $\mathcal{V}(x_i) = tt$ and $s_i \rightarrow t_i$ is deleted in $M^{'}$ if $\mathcal{V}(x_i) = F$. We show that $M', s_0 \models \eta$. Since $\mathcal{V}$ is satisfying assignment, we have $\bigwedge_{1 \leq i \leq n}(\mathcal{V}(a_i) \vee \mathcal{V}(b_i) \vee \mathcal{V}(c_i))$. Without loss of generality, assume that $\mathcal{V}(a_i) = tt$ (similar argument for $\mathcal{V}(b_i) = tt$ and $\mathcal{V}(c_i) = tt$). We have two cases. Case 1 is $a_i = x_j$. Then $\mathcal{V}(x_j) = tt$, so $(s_j, t_j) \in R'$. Also since $a_i = x_j$, $\varphi_i^1 = \mathsf{AG}(p_j \Rightarrow \mathsf{EX}q_j)$. Since $(s_j, t_j) \in R'$, $M', s_0 \models \varphi_i^1$. Hence $M', s_0 \models \eta$. Case 2 is $a_i = \neg x_j$. Then $\mathcal{V}(x_j) = ff$, so $(s_j, t_j) \notin R'$. Also since $a_i = \neg x_j$, $\varphi_i^1 = \mathsf{AG}(p_j \Rightarrow \mathsf{AX}\neg q_j)$. Since $(s_j, t_j) \notin R^{'}$, $M', s_0 \models \varphi_i^1$. Hence $M', s_0 \models \eta$.

f is satisfiable follows from (M, η) *can be repaired*: Let $M' = (S'_0, S', R', L', AP)$ be such that $M' \subseteq M$, $M', s_0 \models \eta$. We define a truth assignment $\mathcal{V}$ as follows: $\mathcal{V}(x_j) = tt$ iff $(s_j, t_j) \in R'$. We show that $\mathcal{V}(f) = tt$, i.e., $\mathcal{V}(a_i) \vee \mathcal{V}(b_i) \vee \mathcal{V}(c_i)$ for all $i = 1 \ldots n$. Since $M, s_0 \models \eta$ we have $M, s_0 \models \varphi_i^1 \vee \varphi_i^2 \vee \varphi_i^3$ for all $i = 1 \ldots n$. Without loss of generality, suppose that $M, s_0 \models \varphi_i^1$ (similar argument for $M, s_0 \models \varphi_i^2$ and $M, s_0 \models \varphi_i^3$). We have two cases. Case 1 is $a_i = x_j$. Then $\varphi_i^1 = \mathsf{AG}(p_j \Rightarrow \mathsf{EX}q_j)$. Since $M', s_0 \models \varphi_i^1$, we must have $(s_j, t_j) \in R'$. Hence $\mathcal{V}(x_j) = tt$ by definition of $\mathcal{V}$. Therefore $\mathcal{V}(a_i) = tt$. Hence $\mathcal{V}(a_i) \vee \mathcal{V}(b_i) \vee \mathcal{V}(c_i)$. Case 2 is $a_i = \neg x_j$. Then $\varphi_i^1 = \mathsf{AG}(p_j \Rightarrow \mathsf{AX}\neg q_j)$. Since $M', s_0 \models \neg\varphi_i^1$, we must have $(s_j, t_j) \notin R'$. Hence $\mathcal{V}(x_j) = ff$. Therefore $\mathcal{V}(a_i) = tt$. Hence $\mathcal{V}(a_i) \vee \mathcal{V}(b_i) \vee \mathcal{V}(c_i)$. □

4 The Subtractive Model Repair Algorithm

Given an instance of model repair (M, η), where $M = (S_0, S, R, L, AP)$ and η is a CTL formula, we define a propositional formula $repair(M, \eta)$ such that $repair(M, \eta)$ is satisfiable iff (M, η) has a solution. $repair(M, \eta)$ is defined over the following propositions:

1. $E_{s,t} : (s, t) \in R$
2. $X_s : s \in S$
3. $X_{s,\psi} : s \in S, \psi \in sub(\eta)$
4. $X^n_{s,\psi} : s \in S, 0 \leq n \leq |S|$, and $\psi \in sub(\eta)$ has the form $\mathsf{A}[\varphi \mathsf{V} \varphi']$ or $\mathsf{E}[\varphi \mathsf{V} \varphi']$

The meaning of $E_{s,t}$ is that the transition (s, t) is retained in the fixed model M' iff $E_{s,t}$ is assigned tt ("true") by the satisfying valuation $\mathcal{V}$ for $repair(M, \eta)$. The meaning of X_s is that state s is retained in the repaired model M'. The meaning of $X_{s,\psi}$ is that ψ holds in state s. $X^n_{s,\psi}$ is used to propagate release formulae (AV or EV) for as long as necessary to determine their truth, i.e., $|S|$ in the worst case.

A satisfying assignment $\mathcal{V}$ of $repair(M, \eta)$, e.g., as given by a SAT solver, gives directly a solution to model repair. Denote this solution by $model(M, \mathcal{V})$, defined as follows.

Definition 7 ($model(M, \mathcal{V})$)**.** *$model(M, \mathcal{V}) = (S'_0, S', R', L', AP)$, where $S'_0 = \{s \mid s \in S_0 \wedge \mathcal{V}(X_s) = tt\}$, $S' = \{s \mid s \in S \wedge \mathcal{V}(X_s) = tt\}$, $R' = \{(s, t) \mid (s, t) \in R \wedge \mathcal{V}(E_{s,t}) = tt\}$, $L' = L \restriction S'$, and $AP' = AP$.*

Note that $model(M, \mathcal{V})$ does not depend directly on η.

Definition 8 ($repair(M, \eta)$)**.** *Let $M = (S_0, S, R, L, AP)$ be a Kripke structure and η a CTL formula. Let $s \to t$ abbreviate $(s, t) \in R$. $repair(M, \eta)$ is the conjunction of all the propositional formulae listed below. These are grouped into sections, where each section deals with one issue, e.g., propositional consistency. s, t implicitly range over S. Other ranges are explicitly given.*

Essentially, $repair(M, \eta)$ encodes all of the usual local constraints, e.g., $\mathsf{AX}\varphi$ holds in s iff φ holds in all successors of s, i.e., all t such that $(s, t) \in R$. We modify these however, to take transition deletion into account. So, the local constraint for AX becomes $\mathsf{AX}\varphi$ holds in s iff φ holds in all successors of s after *transitions have been deleted (to effect the repair). More precisely, instead of $X_{s,\mathsf{AX}\varphi} \equiv \bigwedge_{t|s\to t} X_{t,\varphi}$, we have $X_{s,\mathsf{AX}\varphi} \equiv \bigwedge_{t|s\to t}(E_{s,t} \Rightarrow X_{t,\varphi})$. Here $s \to t$ abbreviates $(s, t) \in R$. The other modalities ($\mathsf{EX}, \mathsf{AV}, \mathsf{EV}$) are treated similarly. We deal with AU, EU by reducing them to EV, AV using duality. We also require that the repaired structure M' be total by requiring that every state has at least one outgoing transition.*

1. *some initial state s_0 is not deleted*

   $\bigvee_{s_0 \in S_0} X_{s_0}$

2. *M' satisfies η, i.e., all undeleted initial states satisfy η*

   for all $s_0 \in S_0 : X_{s_0} \Rightarrow X_{s_0,\eta}$

3. *M' is total, i.e., each retained state has at least one outgoing transition, to some other retained state*

   for all $s \in S : X_s \equiv \bigvee_{t|s\to t}(E_{s,t} \wedge X_t)$

4. *If an edge is retained then both its source and target states are retained*

   for all $(s,t) \in R : E_{s,t} \Rightarrow (X_s \wedge X_t)$

5. *Propositional labeling, for all $s \in S$:*

   for all $p \in AP \cap L(s)$: $X_{s,p}$
   for all $p \in AP - L(s)$: $\neg X_{s,p}$

6. *Propositional consistency, for all $s \in S$:*

   for all $\neg\varphi \in sub(\eta)$: $X_{s,\neg\varphi} \equiv \neg X_{s,\varphi}$
   for all $\varphi \vee \psi \in sub(\eta)$: $X_{s,\varphi\vee\psi} \equiv X_{s,\varphi} \vee X_{s,\psi}$
   for all $\varphi \wedge \psi \in sub(\eta)$: $X_{s,\varphi\wedge\psi} \equiv X_{s,\varphi} \wedge X_{s,\psi}$

7. *Nexttime formulae, for all $s \in S$:*

   for all $\mathsf{AX}\varphi \in sub(\eta)$: $X_{s,\mathsf{AX}\varphi} \equiv \bigwedge_{t|s\to t}(E_{s,t} \Rightarrow X_{t,\varphi})$
   for all $\mathsf{EX}\varphi \in sub(\eta)$: $X_{s,\mathsf{EX}\varphi} \equiv \bigvee_{t|s\to t}(E_{s,t} \wedge X_{t,\varphi})$

8. *Release formulae. Let $n = |S|$, i.e., the number of states in M. Then, for all $s \in S$:*

   for all $\mathsf{A}[\varphi\mathsf{V}\psi] \in sub(\eta)$, $m \in \{1,\ldots,n\}$:
   $X_{s,\mathsf{A}[\varphi\mathsf{V}\psi]} \equiv X^n_{s,\mathsf{A}[\varphi\mathsf{V}\psi]}$
   $X^m_{s,\mathsf{A}[\varphi\mathsf{V}\psi]} \equiv X_{s,\psi} \wedge (X_{s,\varphi} \vee \bigwedge_{t|s\to t}(E_{s,t} \Rightarrow X^{m-1}_{t,\mathsf{A}[\varphi\mathsf{V}\psi]}))$
   $X^0_{s,\mathsf{A}[\varphi\mathsf{V}\psi]} \equiv X_{s,\psi}$
   for all $\mathsf{E}[\varphi\mathsf{V}\psi] \in sub(\eta)$, $m \in \{1,\ldots,n\}$:
   $X_{s,\mathsf{E}[\varphi\mathsf{V}\psi]} \equiv X^n_{s,\mathsf{E}[\varphi\mathsf{V}\psi]}$
   $X^m_{s,\mathsf{E}[\varphi\mathsf{V}\psi]} \equiv X_{s,\psi} \wedge (X_{s,\varphi} \vee \bigvee_{t|s\to t}(E_{s,t} \wedge X^{m-1}_{t,\mathsf{E}[\varphi\mathsf{V}\psi]}))$
   for all $\mathsf{E}[\varphi\mathsf{V}\psi] \in sub(\eta)$: $X^0_{s,\mathsf{E}[\varphi\mathsf{V}\psi]} \equiv X_{s,\psi}$

The various clauses of Def. 8 handle corresponding clauses of Def. 2. Clause 5 handles Clause 3 of Def. 2. Clause 6 handles Clauses 4, 5, and 6 of Def. 2. Clause 7 handles Clauses 7 and 8 of Def. 2. Note here the use of the edge booleans $E_{s,t}$ to remove from consideration (in the determination of whether s satisfies $\mathsf{AX}\varphi$ or $\mathsf{EX}\varphi$) an edge that is to be deleted as part of the repair, so that $E_{s,t}$ is false. Clause 8 handles Clauses 9 and 10 of Def. 2, as follows. Along each path, either (1) a state is reached where $[\varphi\mathsf{V}\psi]$ is discharged ($\varphi \wedge \psi$), or (2) $[\varphi\mathsf{V}\psi]$ is shown to be false ($\neg\varphi \wedge \neg\psi$), or (3) some state eventually repeats. In case (3), we know that release also holds along this path. Thus, by expanding the release modality up to n times, where n is the number of states in the original structure M, we ensure that the third case holds if the first two have not yet resolved the truth of $(\varphi\mathsf{V}\psi)$ along the path in question. To carry out the expansion correctly, we use a version of $X_{s,\mathsf{A}[\varphi\mathsf{V}\psi]}$ that is superscripted with an integer between 0 and n. This imposes a "well foundedness" on the $X^m_{s,\mathsf{A}[\varphi\mathsf{V}\psi]}$ propositions, and prevents for

example, a cycle along which ψ holds in all states and yet the $X_{s,\mathsf{A}[\varphi \mathsf{V} \psi]}$ are assigned false in all states s along the cycle. Finally, Clauses 1 and 2 of Def. 2 can be dealt with by viewing true as an abbreviation of $p \vee \neg p$ (for some $p \in AP$) and false as an abbreviation of $p \wedge \neg p$ (for some $p \in AP$), so they reduce to other clasues.

Note that the above requires all states, even those rendered unreachable by transition deletion, to have some outgoing transition. This "extra" requirement on the unreachable states does not affect the method however, since there will actually remain a satisfying assignment which allows unreachable state to retain all their outgoing transitions, if some $M' \subseteq M$ exists that satisfies η. For s unreachable from s_0 in M', assign the value to $X_{s,\varphi}$ that results from model checking $M', s \models \varphi$. This gives a consistent assignment that satsifies $\mathit{repair}(M, \eta)$. Clearly, $X_{s,\varphi}$ does not affect $X_{s_0,\eta}$ since s is unreachable from s_0.

Proposition 1. *The length of $\mathit{repair}(M, \eta)$ is $O(|S|^2 \times |\eta| \times d + |S| \times |AP| + |R|)$. where $|S|$ is the number of states in S, $|R|$ is the number of transitions in R, $|\eta|$ is the length of η, $|AP|$ is the numebr of atomic propositions in AP, and d is the outdegree of M, i.e., the maximum of the number of successors that any state in M has.*

Proof. $\mathit{repair}(M, \eta)$ is the conjunction of Clauses 1–8 of Def. 8. We analyze the length of each clause in turn:

1. Clause 1: $O(|S|)$, since we have the term X_{s_0} for each $s_0 \in S_0$, and $S_0 \subseteq S$.
2. Clause 2: $O(|S|)$, since we have the term $X_{s_0} \Rightarrow X_{s_0,\eta}$ for each $s \in S$.
3. Clause 3: $O(|S| \times d)$, since we have the term $X_s \equiv \bigvee_{t|s \to t}(E_{s,t} \wedge X_t)$ for each $s \in S$.
4. Clause 4: $O(|R|)$, since we have the term $E_{s,t} \Rightarrow (X_s \wedge X_t)$ for each $(s,t) \in R$.
5. Clause 5: $O(|S| \times |AP|)$, since we have either $X_{s,p}$ or $\neg X_{s,p}$ for each $s \in S$ and $p \in AP$
6. Clause 6: $O(|S| \times |\eta|)$, since we have one of $X_{s,\neg\varphi} \equiv \neg X_{s,\varphi}$, $X_{s,\varphi\vee\psi} \equiv X_{s,\varphi} \vee X_{s,\psi}$, $X_{s,\varphi\wedge\psi} \equiv X_{s,\varphi} \wedge X_{s,\psi}$ for each $s \in S$ and each formula in $\mathit{sub}(\eta)$ whose main operator is propositional, and $|\mathit{sub}(\eta)|$ is in $O(|\eta|)$.
7. Clause 7 $O(|S| \times |\eta| \times d)$, since we have a term of size $O(d)$, namely either $X_{s,\mathsf{AX}\varphi} \equiv \bigwedge_{t|s \to t}(E_{s,t} \Rightarrow X_{t,\varphi})$ or $X_{s,\mathsf{EX}\varphi} \equiv \bigvee_{t|s \to t}(E_{s,t} \wedge X_{t,\varphi})$, for each formula in $\mathit{sub}(\eta)$ whose main operator is either AX or EX.
8. Clause 8 $O(|S|^2 \times |\eta| \times d)$, since the second $|S|$ term is due to the superscripts $m \in \{1, \ldots, |S|\}$. Each of these formulae has length $O(d)$. The sum of lengths of all these formulae is $O(|\eta| \times |S|^2 \times d)$.

Summing all of the above, we obtain that the overall length of $\mathit{repair}(M, \varphi)$ is $O(|S|^2 \times |\eta| \times d + |S| \times |AP| + |R|)$. □

Clearly, $\mathit{repair}(M, \eta)$ can be constructed in polynomial time. Figure 1 presents our model repair algorithm, $\mathsf{Repair}(M, \varphi)$, which we show is sound, and complete provided that a complete SAT-solver is used. Recall that we use $\mathit{model}(M, \mathcal{V})$ to denote the structure M' derived from the repair of M w.r.t. η, as per Def. 7.

Theorem 2 (Soundness)**.** *Let $M = (S_0, S, R, L, AP)$ be a Kripke structure, η a CTL formula, and $n = |S|$. Suppose that repair(M, η) is satisfiable and that $\mathcal{V}$ is a satisfying truth assignment for it. Let $M' = (S'_0, S', R', L', AP) = model(M, \mathcal{V})$, Then for all reachable states $s \in S'$ and CTL formulae $\xi \in sub(\eta)$: $\mathcal{V}(X_{s,\xi}) = tt$ iff $M', s \models \xi$*

Proof. We proceed by induction on the structure of ξ. We sometimes write $\mathcal{V}(X_{s,\xi})$ instead of $\mathcal{V}(X_{s,\xi}) = tt$ and $\neg\mathcal{V}(X_{s,\xi})$ instead of $\mathcal{V}(X_{s,\xi}) = ff$.

Case $\xi = \neg\varphi$:
$\mathcal{V}(X_{s,\xi}) = tt$ iff
$\triangleright$case condition
$\mathcal{V}(X_{s,\neg\varphi}) = tt$ iff
$\triangleright$Clause 6 (propositional consistency) of Def. 8
$\mathcal{V}(X_{s,\varphi}) = ff$ iff
$\triangleright$induction hypothesis
$not(M', s \models \varphi)$ iff
$\triangleright$CTL semantics
$M', s \models \neg\varphi$ iff
$\triangleright$case condition
$M', s \models \xi$

Case $\xi = \varphi \vee \psi$:
$\mathcal{V}(X_{s,\xi}) = tt$ iff
$\triangleright$case condition
$\mathcal{V}(X_{s,\varphi\vee\psi}) = tt$ iff
$\triangleright$Clause 6 (propositional consistency) of Def. 8
$\mathcal{V}(X_{s,\varphi}) = tt$ or $\mathcal{V}(X_{s,\psi}) = tt$ iff
$\triangleright$induction hypothesis
$(M', s \models \varphi)$ or $(M', s \models \psi)$ iff
$\triangleright$CTL semantics
$M', s \models \varphi \vee \psi$ iff $\triangleright$case condition
$M', s \models \xi$

Case $\xi = \varphi \wedge \psi$:
$\mathcal{V}(X_{s,\xi}) = tt$ iff
$\triangleright$case condition
$\mathcal{V}(X_{s,\varphi\wedge\psi}) = tt$ iff
$\triangleright$Clause 6 (propositional consistency) of Def. 8
$\mathcal{V}(X_{s,\varphi}) = tt$ and $\mathcal{V}(X_{s,\psi}) = tt$ iff
$\triangleright$induction hypothesis
$(M', s \models \varphi)$ and $(M', s \models \psi)$ iff $\triangleright$CTL semantics
$M', s \models \varphi \wedge \psi$ iff $\triangleright$case condition
$M', s \models \xi$

Case $\xi = \mathsf{AX}\varphi$:
$\mathcal{V}(X_{s,\xi}) = tt$ iff
$\triangleright$case assumption

$\mathcal{V}(X_{s,\mathsf{AX}\varphi}) = tt$ iff
$\triangleright$Def. 8
$\bigwedge_{t|s\to t} \mathcal{V}(E_{s,t} \Rightarrow X_{t,\varphi}) = tt$ iff
$\triangleright$boolean semantics of $\Rightarrow$
$\bigwedge_{t|s\to t} \mathcal{V}(E_{s,t}) = tt \Rightarrow \mathcal{V}(X_{t,\varphi}) = tt$ iff
$\triangleright s$ reachable by assumption, $E_{s,t}$ implies t isreachable, and apply ind. hyp.
$\bigwedge_{t|s\to t}(s,t) \in R' \Rightarrow M', t \models \varphi$ iff
$\triangleright R' \subseteq R$, so restriction of t to $t \mid s \to t$, i.e., $t \mid (s,t) \in R$, does not matter
$M', s \models \mathsf{AX}\varphi$ iff
$\triangleright$case assumption
$M', s \models \xi$

Case $\xi = \mathsf{EX}\varphi$:
$\mathcal{V}(X_{s,\xi}) = tt$ iff
$\triangleright$case assumption
$\mathcal{V}(X_{s,\mathsf{EX}\varphi}) = tt$ iff
$\triangleright$Def. 8
$\bigvee_{t|s\to t} \mathcal{V}(E_{s,t} \wedge X_{t,\varphi}) = tt$ iff
$\triangleright$boolean semantics of $\wedge$
$\bigvee_{t|s\to t} \mathcal{V}(E_{s,t}) = tt \wedge \mathcal{V}(X_{t,\varphi}) = tt$ iff $\triangleright t$ is reachable from s by assumption, and apply ind. hyp.
$\bigvee_{t|s\to t}(s,t) \in R' \wedge M', t \models \varphi$ iff
$\triangleright R' \subseteq R$, so restriction of t to $t \mid s \to t$, i.e., $t \mid (s,t) \in R$, does not matter
$M', s \models \mathsf{EX}\varphi$ iff
$\triangleright$case assumption
$M', s \models \xi$

Case $\xi = \mathsf{A}[\varphi \mathsf{V} \psi]$: We do the proof for each direction separately.

Left to right, i.e., $\mathcal{V}(X_{s,\mathsf{A}[\varphi\mathsf{V}\psi]})$ implies $M', s \models \mathsf{A}[\varphi\mathsf{V}\psi]$:
$\mathcal{V}(X_{s,\mathsf{A}[\varphi\mathsf{V}\psi]})$ iff
$\triangleright$Def. 8
$\mathcal{V}(X^n_{s,\mathsf{A}[\varphi\mathsf{V}\psi]})$ iff
$\triangleright$Def. 8
$\mathcal{V}(X_{s,\psi} \wedge (X_{s,\varphi} \vee \bigwedge_{t|s\to t}(E_{s,t} \Rightarrow X^{n-1}_{t,\mathsf{A}[\varphi\mathsf{V}\psi]})))$ iff
$\triangleright \mathcal{V}$ is a boolean valuation function, and so distributes over boolean connectives
$\mathcal{V}(X_{s,\psi}) \wedge (\mathcal{V}(X_{s,\varphi_1}) \vee (\bigwedge_{t|s\to t} \mathcal{V}(E_{s,t}) \Rightarrow \mathcal{V}(X^{n-1}_{t,\mathsf{A}[\varphi_1\mathsf{V}\psi]})))$ iff
$\triangleright$induction hypothesis
$M', s \models \psi \wedge (M', s \models \varphi \vee \bigwedge_{t|s\to t}((s,t) \in R' \Rightarrow \mathcal{V}(X^{n-1}_{t,\mathsf{A}[\varphi\mathsf{V}\psi]}))$.

We now have two cases

1. $M', s \models \varphi$. In this case, $M', s \models \mathsf{A}[\varphi\mathsf{V}\psi]$, and so $M', s \models \xi$.
2. $\bigwedge_{t|s\to t}(s,t) \in R' \Rightarrow \mathcal{V}(X^{n-1}_{t,\mathsf{A}[\varphi\mathsf{V}\psi]})$.

For case 2, we proceed as follows. Let t be an arbitrary state such that $(s,t) \in R'$. Then $\mathcal{V}(X^{n-1}_{t,\mathsf{A}[\varphi\mathsf{V}\psi]})$. If we show that $\mathcal{V}(X^{n-1}_{t,\mathsf{A}[\varphi\mathsf{V}\psi]})$ implies $M', s \models \mathsf{A}[\varphi\mathsf{V}\psi]$ then we are done, by CTL semantics. The argument is essentially a repetition of the above argument for $\mathcal{V}(X_{s,\mathsf{A}[\varphi\mathsf{V}\psi]})$ implies $M', s \models$

$\mathsf{A}[\varphi \mathsf{V} \psi]$.
Proceeding as above, we conclude $M', t \models \psi$ and one of the same two cases as above:

- $M', t \models \varphi$
- $\bigwedge_{u|t \to u}(t,u) \in R' \Rightarrow \mathcal{V}(X^{n-2}_{u,\mathsf{A}[\varphi \mathsf{V} \psi]})$

However note that, in case 2, we are "counting down." Since we count down for $n = |S|$, then along every path starting from s, either case (1) occurs, which "terminates" that path, as far as valuation of $[\varphi \mathsf{V} \psi]$ is concerned, or we will repeat a state before (or when) the counter reaches 0. Along such a path (from s to the repeated state), ψ holds at all states, and so $[\varphi \mathsf{V} \psi]$ holds along this path. We conclude that $[\varphi \mathsf{V} \psi]$ holds along all paths starting in s, and so $M', s \models \mathsf{A}[\varphi \mathsf{V} \psi]$.

Right to left, i.e., $\mathcal{V}(X_{s,\mathsf{A}[\varphi \mathsf{V} \psi]})$ follows from $M', s \models \mathsf{A}[\varphi \mathsf{V} \psi]$:
Assume that $M', s \models \mathsf{A}[\varphi \mathsf{V} \psi]$ holds. Hence $M', s \models \psi \wedge (M', s \models \varphi \vee \bigwedge_{t|t \to s}((s,t) \in R' \Rightarrow M', t \models \mathsf{A}[\varphi \mathsf{V} \psi]))$. By the induction hypothesis, $\mathcal{V}(X_{s,\psi}) \wedge (\mathcal{V}(X_{s,\varphi}) \vee \bigwedge_{t|t \to s}((s,t) \in R' \Rightarrow M', t \models \mathsf{A}[\varphi \mathsf{V} \psi]))$. We now have two cases

1. $\mathcal{V}(X_{s,\varphi})$. Since we have $\mathcal{V}(X_{s,\psi}) \wedge \mathcal{V}(X_{s,\varphi})$ we conclude $\mathcal{V}(X_{s,\mathsf{A}[\varphi \mathsf{V} \psi]})$, and so we are done.
2. $\bigwedge_{t|s \to t}(s,t) \in R' \Rightarrow M', t \models \mathsf{A}[\varphi \mathsf{V} \psi]$

For case 2, we proceed as follows. Let t be an arbitrary state such that $(s,t) \in R'$. Then $M', t \models \mathsf{A}[\varphi \mathsf{V} \psi]$. If we show that $\mathcal{V}(X^{n-1}_{t,\mathsf{A}[\varphi \mathsf{V} \psi]})$ follows from $M', t \models \mathsf{A}[\varphi \mathsf{V} \psi]$ then we can conclude $\mathcal{V}(X_{s,\mathsf{A}[\varphi \mathsf{V} \psi]})$ by Definition 8. Proceeding as above, we conclude $\mathcal{V}(X_{t,\psi})$ and one of the same two cases as above:

- $\mathcal{V}(X_{t,\varphi})$, so by Definition 8, $\mathcal{V}(X^{n-1}_{t,\mathsf{A}[\varphi \mathsf{V} \psi]})$ holds.
- $\bigwedge_{u|t \to u}(t,u) \in R' \Rightarrow \mathcal{V}(X^{n-2}_{u,\mathsf{A}[\varphi \mathsf{V} \psi]})$

As before, in case 2 we are "counting down." Since we count down for $n = |S|$, then along every path starting from s, either case (1) occurs, which "terminates" that path, as far as establishment of $\mathcal{V}(X_{t,\varphi})$ is concerned, or we will repeat a state before (or when) the counter reaches 0. Along such a path (from s to the repeated state, call it v), ψ holds at all states. By Definition 8, $X^0_{v,\mathsf{A}[\varphi \mathsf{V} \psi]} \equiv X_{v,\psi}$. From $M', v \models \psi$ and the induction hypothesis, $\mathcal{V}(X_{v,\psi})$ holds. Hence $X^0_{v,\mathsf{A}[\varphi \mathsf{V} \psi]}$ holds. Thus, along every path starting from s, we reach a state w such that $\mathcal{V}(X^m_{w,\mathsf{A}[\varphi \mathsf{V} \psi]})$ holds for some $m \in \{0, \ldots, n\}$. Hence by Definition 8, $\mathcal{V}(X_{s,\mathsf{A}[\varphi \mathsf{V} \psi]})$ holds.

Case $\xi = \mathsf{E}[\varphi \mathsf{V} \psi]$: this is argued in the same way as the above case for $\xi = \mathsf{A}[\varphi \mathsf{V} \psi]$, except that we expand along one path starting in s, rather than all paths. The differences with the AV case are straightforward, and we omit the details. □

Corollary 1 (Soundness)**.** *If* $\mathsf{Repair}(M, \eta)$ *returns a structure* $M' = (S'_0, S', R', L', AP)$*, then (1)* M' *is total, (2)* $M' \subseteq M$*, (3)* $M', S'_0 \models \eta$*, and (4)* M *is repairable.*

Proof. Let $\mathcal{V}$ be the truth assignment for $repair(M, \eta)$ that was returned by the SAT-solver in the execution of $\mathsf{Repair}(M, \eta)$. Since the SAT-solver is assumed to be sound, $\mathcal{V}$ is actually a

satisfying assignment for $repair(M, \eta)$. (1) follows from Clause 3 (M' is total) of Def. 8. (2) holds by construction of M', which is derived from M by deleting transitions and (subsequently) unreachable states. For (3), let s_0 be an arbitrary state in S'_0. Since s_0 was not deleted, we have $\mathcal{V}(X_{s_0}) = tt$. Hence, by Clause 2 of Def. 8, $\mathcal{V}(X_{s_0,\eta}) = tt$. Hence, by Th. 2, $M', s_0 \models \eta$. Finally, (4) follows from (1)–(3) and Def. 5. □

Theorem 3 (Completness)**.** *If M is repairable with respect to η then* $\mathsf{Repair}(M, \eta)$ *returns a Kripke structure $M'' = (S''_0, S'', R'', L'', AP)$ such that M'' is total, $M'' \subseteq M$, and $M'', S''_0 \models \eta$.*

Proof. Assume that M is repairable with respect to η. By Definition 5, there exists a total substructure $M' = (S'_0, S', R', L', AP)$ of M such that $M', S'_0 \models \eta$. We define a satisfying valuation $\mathcal{V}$ for $repair(M, \eta)$ as follows.

Assign tt to $E_{s,t}$ for every edge $(s, t) \in R'$ and ff to every $E_{s,t}$ for every edge $(s, t) \notin R'$. Since M' is total, the "M' is total" section is satisfied by this assignment.

Assign tt to $X_{s_0,\eta}$ for all $s_0 \in S'_0$. Consider an execution of the CTL model checking algorithm of Clarke, Emerson, and Sistla [11] for checking $M', s_0 \models \eta$. This algorithm will assign a value to every formula φ in $sub(\eta)$ in every reachable state s of M'. Set $\mathcal{V}(X_{s,\varphi})$ to this value. By construction of the model checking algorithm [11], these valuations will satisfy all of the constraints given in the "propositional labeling," "propositional consistency," "nexttime formulae," and "release formulae" sections of Definition 8. Hence all conjuncts of $repair(M, \eta)$ are assigned tt by $\mathcal{V}$. Hence $\mathcal{V}(repair(M, \eta)) = tt$, and so $repair(M, \eta)$ is satisfiable.

Now the SAT-solver used is assumed to be complete, and so will return some satisfying assignment for $repair(M, \eta)$ (not necessarily $\mathcal{V}$, since there may be more than one satisfying assignment). Thus, $\mathsf{Repair}(M, \eta)$ returns a structure $M'' = (S''_0, S'', R'', L'', AP)$, rather than "failure." By Cor. 1, M'' is total, $M'' \subseteq M$, and $M'', S''_0 \models \eta$. □

Let ACTL* [16] denote the universal fragment (no existential path quantifier) of CTL*, and [16].

Proposition 2. *If* $\mathsf{Repair}(M, \eta)$ *returns a structure M', then (1) there is a simulation relation from M' to M, and (2) for all* ACTL* *formulae f, $M \models f$ implies $M' \models f$.*

Proof. Clause (1) follows by observing that the relation mapping each state in M' to "itself" in M is a simulation relation [16] from M' to M. This is becasue M' results by removing transitions and unreachable states from M. Clause (2) follows from immediately from Clause (1) and Corollary 1 of [16]. □

Hence the repair method preserves satisfaction of all ACTL* formulae, while adding the satisfaction of η. Note that we cannot repair w.r.t. ACTL*, since our method does not deal with path formulae (e.g., FG, GF), but we can preseve ACTL*properties that are originally satisfied by M, and have been verified by other means.

Other model correction methods to date [9, 28] do not guarantee the preservation of properties that are not explicitly mentioned. Instead, they aim for a "minimal" repair, i.e., the repair should perturb the structure as little as possible, w.r.t. some distance metric. Our method, in contrast, guarantees preservation of all ACTL*properties, regardless of whether they are given by η or not. In addition, we can enforce similarity between M and $M' = \mathsf{Repair}(M, \eta)$ by requiring that specific transitions not be deleted. A transition from state s to state t can be made non-deletable by conjoining $E_{s,t}$ to $repair(M, \eta)$; see the discussion in Sect. 6.4.

```
Repair(M, η):
model check M, S_0 ⊨ η;
if successful, then return M
else
    compute repair(M, η) as given in Section 3;
    submit repair(M, η) to a sound and complete SAT-solver;
    if the SAT-solver returns "not satisfiable" then
        return "failure"
    else
        the solver returns a satisfying assignment V;
        return M' = model(M, V)
```

Figure 1: The model repair algorithm.

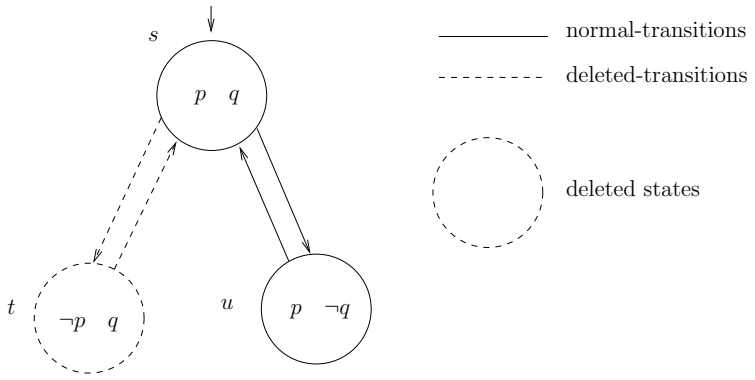

Figure 2: Simple Kripke structure

4.1 Example repair formula

We show how $repair(M, \eta)$ is calculated (manually) for the simple Kripke structure in Fig. 2 and the CTL formula $\eta = (\mathsf{AG}p \vee \mathsf{AG}q) \wedge \mathsf{EX}p$. We omit the clauses dealing with totality of the model. We use $\mathsf{AG}p$ rather than $\mathsf{A}[\mathsf{false}\mathsf{V}p]$, but with the same index superscript for "couting down".

$X_{s,\eta} \equiv X_{s,(\mathsf{AG}p \vee \mathsf{AG}q) \wedge \mathsf{EX}p}$

$X_{s,(\mathsf{AG}p \vee \mathsf{AG}q) \wedge \mathsf{EX}p} \equiv X_{s,\mathsf{AG}p \vee \mathsf{AG}q} \wedge \mathsf{X}_{s,\mathsf{EX}p}$

$X_{s,\mathsf{AG}p \vee \mathsf{AG}q} \equiv X_{s,\mathsf{AG}p} \vee X_{s,\mathsf{AG}q}$

We start by solving for $X_{s,\mathsf{AG}p}$.

$X_{s,\mathsf{AG}p} \equiv X^3_{s,\mathsf{AG}p}$

$X^3_{s,\mathsf{AG}p} \equiv X_{s,p} \wedge (E_{s,t} \Rightarrow X^2_{t,\mathsf{AG}p}) \wedge (E_{s,u} \Rightarrow X^2_{u,\mathsf{AG}p})$

$X^2_{t,\mathsf{AG}p} \equiv X_{t,p} \wedge (E_{t,s} \Rightarrow X^1_{s,\mathsf{AG}p})$

$X^2_{u,\mathsf{AG}p} \equiv X_{u,p} \wedge (E_{u,s} \Rightarrow X^1_{s,\mathsf{AG}p})$

$X^1_{s,\mathsf{AG}p} \equiv X_{s,p} \wedge (E_{s,t} \Rightarrow X^0_{t,\mathsf{AG}p}) \wedge (E_{s,u} \Rightarrow X^0_{u,\mathsf{AG}p})$

$X^0_{t,\mathsf{AG}p} \equiv X_{t,p} \equiv \mathit{ff}$

$X^0_{u,\mathsf{AG}p} \equiv X_{u,p} \equiv \mathit{tt}$

By replacing $X_{s,p}$ etc. by their truth values, we can simplify the above as follows. It is more intuitive to work "bottom up"

$X^1_{s,\mathsf{AG}p} \equiv \neg E_{s,t}$

$X^2_{u,\mathsf{AG}p} \equiv (E_{u,s} \Rightarrow X^1_{s,\mathsf{AG}p})$

$X^2_{u,\mathsf{AG}p} \equiv E_{u,s} \Rightarrow \neg E_{s,t}$

$X^3_{s,\mathsf{AG}p} \equiv \neg E_{s,t} \wedge (E_{s,u} \Rightarrow X^2_{u,\mathsf{AG}p})$

$X^3_{s,\mathsf{AG}p} \equiv \neg E_{s,t} \wedge (E_{s,u} \Rightarrow (E_{u,s} \Rightarrow \neg E_{s,t})) \equiv \neg E_{s,t}$

$X_{s,\mathsf{AG}p} \equiv \neg E_{s,t}$

Symmetrically, we have:

$X_{s,\mathsf{AG}q} \equiv \neg E_{s,u}$

It remains to solve for $X_{s,\mathsf{EX}p}$.

$X_{s,\mathsf{EX}p} \equiv (E_{s,t} \wedge X_{t,p}) \vee (E_{s,u} \wedge X_{u,p})$

By replacing $X_{t,p}$ and $X_{u,p}$ by their values we get:

$X_{s,\mathsf{EX}p} \equiv (E_{s,t} \wedge \mathit{ff}) \vee (E_{s,u} \wedge \mathit{tt}) \equiv E_{s,u}$

Therefore, we now can solve for $X_{s,\eta}$ producing:

$\mathsf{X}_{s,\eta} \equiv (\neg E_{s,t} \vee \neg E_{s,u}) \wedge E_{s,u} \equiv \neg E_{s,t} \wedge E_{s,u}$

The above solution implies that $\mathsf{Repair}(M, \eta)$ will remove the edge (s, t) and all the resulting unreachable states as shown in Fig. 2.

Note that for $\eta = (\mathsf{AG}p \vee \mathsf{AG}q)$, we obtain $\mathsf{X}_{s,\eta} \equiv (\neg E_{s,t} \vee \neg E_{s,u})$, which admits two satisfying valuations, i.e., removing either (s, t) or (s, u) produces the needed repair.

5 Examples of Subtractive Repair

5.1 Mutual exclusion: safety

We treat the standard two-process mutual exclusion example, in which P_i $(i = 1, 2)$ cycles through three states: neutral (preforming local computation), tryng (requested the critical section), and critical (inside the critical section). P_i has three atomic propositions, N_i, T_i, C_i. In the neutral state, N_i is true and T_i, C_i are false. In the trying state, T_i is true and N_i, C_i are false. In the critical state, C_i is true and N_i, T_i are false. The CTL specification η is $\mathsf{AG}(\neg(C_1 \wedge C_2))$, i.e., P_1 and P_2 are never simultaneously in their critical sections.

Figure 3 shows an initial Kripke strucure M which violates mutual exclusion, and Fig. 4 shows a repair of M w.r.t. $\mathsf{AG}(\neg(C_1 \wedge C_2))$. Our tool shows the transitions to be deleted (to effect the repair) as dashed. Initial states are colored green, and the text atached to each state has the form "name$(p_1, \ldots, p_n)$" where "name" is a symbolic name for the state, and $(p_1, \ldots, p_n)$ is the list of atomic propositions that are true in the state. By default, our tool concatenates

the proposition names to obtain the state name. In Fig. 4 the violating state `C1C2` is now unreachable, and so $\mathsf{AG}(\neg(C_1 \wedge C_2))$ is now satisfied. This repair is however, overly restrictive, as it leaves only a single path through the structure, and moreover it does not permit P_1 to enter the critical section. We show in the sequel how to improve the quality of the repairs.

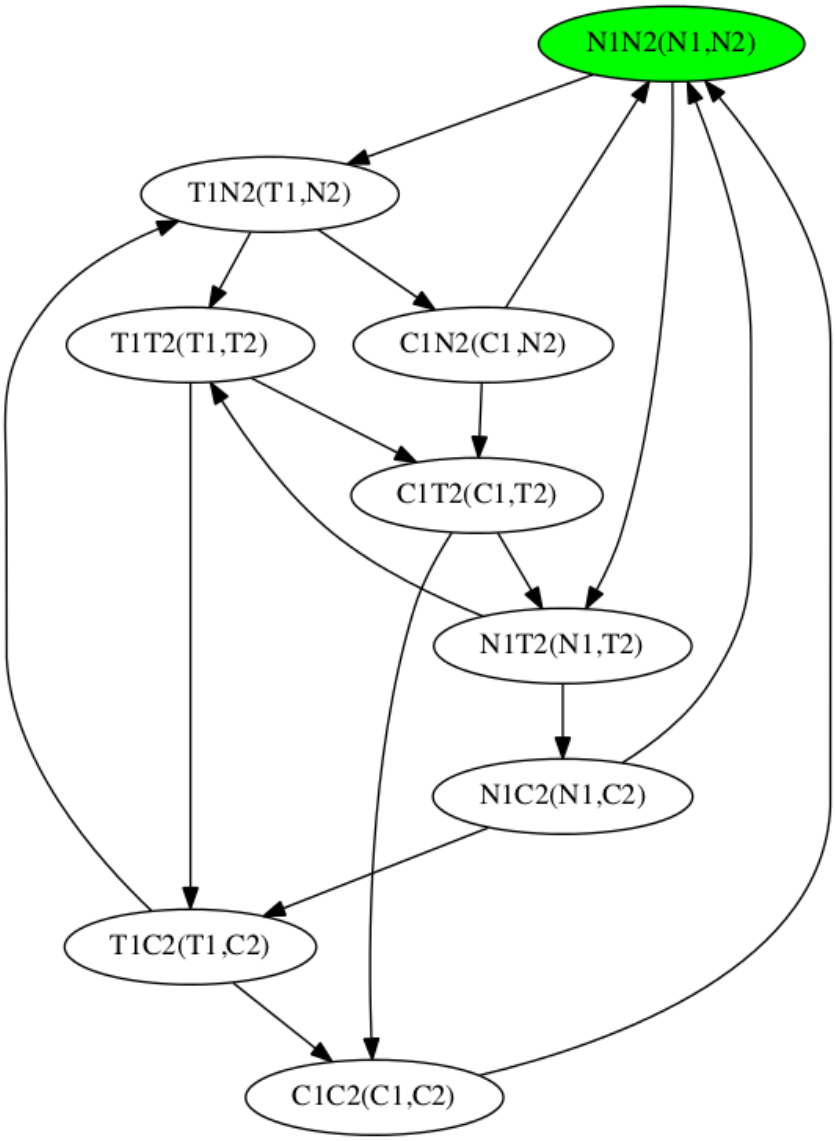

Figure 3: Mutex: initial structure

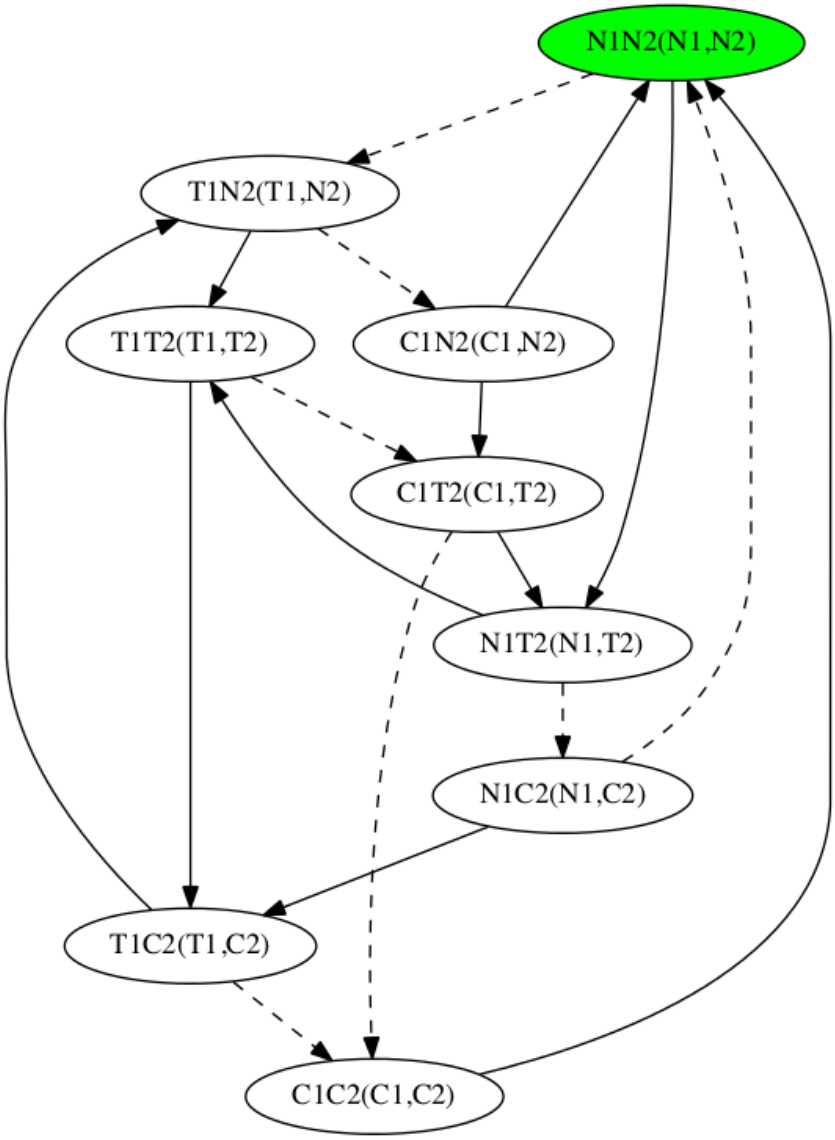

Figure 4: Mutex: repaired structure

5.2 Mutual exclusion: interactive design using semantic feedback

We add liveness to the mutex example by repairing w.r.t. $\mathsf{AG}(\neg(C_1 \wedge C_2)) \wedge \mathsf{AG}(T_1 \Rightarrow \mathsf{AF}C_1) \wedge \mathsf{AG}(T_2 \Rightarrow \mathsf{AF}C_2)$. We start with the structure shown in Fig. 5. This yields the structure of Fig. 6, in which P_1 and P_2 alternately enter the critical section, which is too restrictive. We notice that some transitions where a process requests the critical section, by moving from neutral to trying (e.g., `N1N2` to `N1T2`) are deleted. Since a process should always be able to *request* the critical section, we mark as "retain" all such transitions. The retain button in Eshmun renders a transition (s, t) not deletable, by conjoining $E_{s,t}$ to $\mathit{repair}(M, \eta)$, thereby requiring $E_{s,t}$ to be assigned true. After marking all such request transitions (there are six) and re-attempting repair, we obtain that the structure is not repairable! By dropping, say $\mathsf{AG}(T_2 \Rightarrow \mathsf{AF}C_2)$, we repair w.r.t. $\mathsf{AG}(\neg(C_1 \wedge C_2)) \wedge \mathsf{AG}(T_1 \Rightarrow \mathsf{AF}C_1)$ and obtain a repair, given in Fig. 7.

We observe that $\mathsf{AG}(T_1 \Rightarrow \mathsf{AF}C_1)$ was previously violated by the cycle `T1N2` $\rightarrow$ `T1T2` $\rightarrow$ `T1C2` $\rightarrow$

T1N2, which is now broken. By symmetric reasoning, $\mathsf{AG}(T_2 \Rightarrow \mathsf{AF}C_2)$ was previously violated by the cycle N1T2 → T1T2 → C1T2 → N1T2. Inspection shows that we cannot break both cycles at once: T1N2 → T1T2 and N1T2 → T1T2 are now non-deletable. Removing either of T1C2 → T1N2 or C1T2 → N1T2, leaves T1C2, C1T2 respectively without an outgoing transition, so M' is no longer total. Hence we have to remove both T1T2 → T1C2 and T1T2 → C1T2, which leaves T1T2 without an outgoing transition.

The problem is that in T1T2 there is no notion of priority: which process should enter first. We can add this by *adding* a new proposition (the usual "turn" variable) to record priority among P_1 and P_2, effectvely *splitting* T1T2 into two states. This is a kind of repair that involves *enlarging the domain of the state space*, which is different from adding a state over the existing domain, which some repair methods currently do [10, 28]. Extending our method and tool to automate such repair is a topic for future work, but we note that manual interaction and semantic feedback are helpful here, and an initial step is to provide information about the reasons for the failure of a repair.

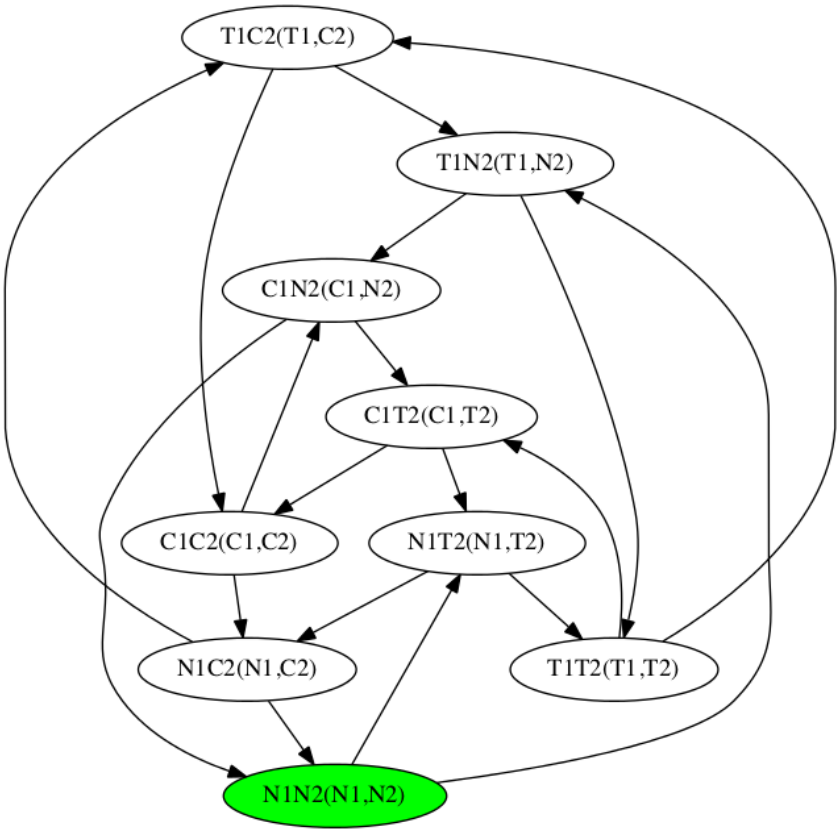

Figure 5: Mutex: initial structure

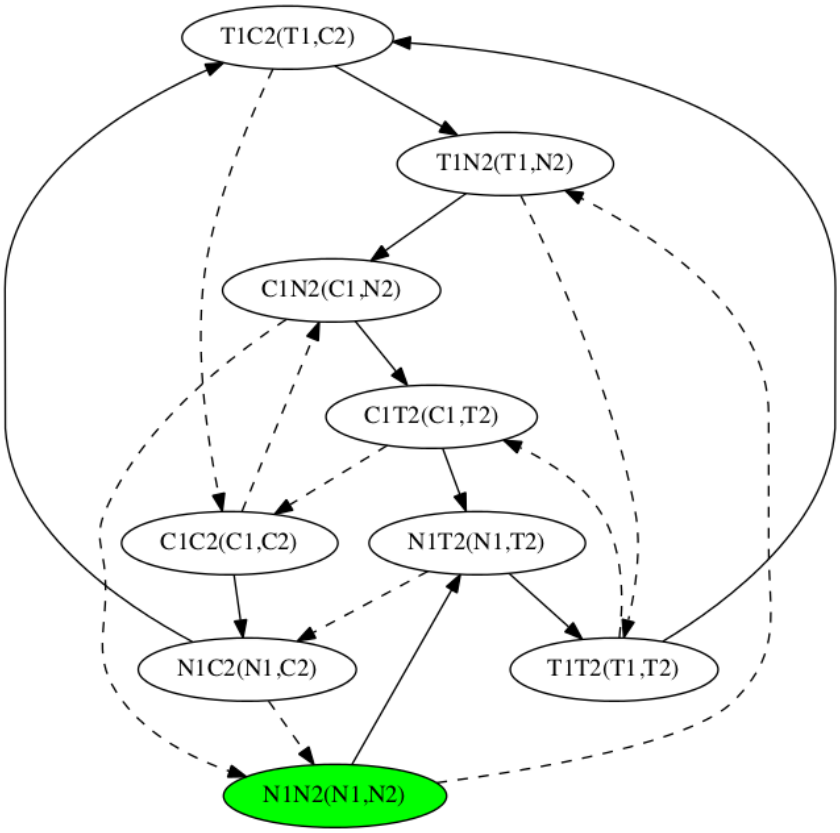

Figure 6: Mutex: alternating entry

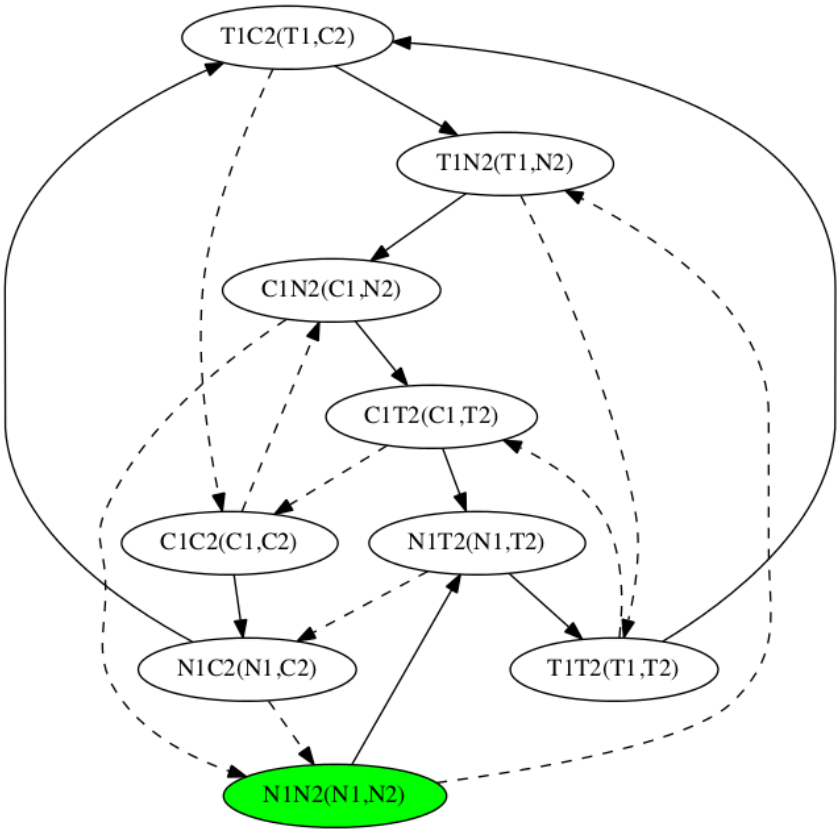

Figure 7: Mutex: P_1 is live

5.3 Barrier synchronization

In this problem, each process P_i is a cyclic sequence of two terminating phases, phase A and phase B. P_i, $(i \in \{1,2\})$, is in exactly one of four local states, sa_i, ea_i, sb_i, eb_i, corresponding to the start of phase A, then the end of phase A, then the start of phase B, and then the end of phase B, afterwards cycling back to sa_i. The CTL specification is the conjunction of the following:

1. Initially both processes are at the start of phase A: $sa_1 \wedge sa_2$
2. P_1 and P_2 are never simultaneously at the start of different phases:
   $\mathsf{AG}(\neg(sa_1 \wedge sb_2)) \wedge \mathsf{AG}(\neg(sa_2 \wedge sb_1))$
3. P_1 and P_2 are never simultaneously at the end of different phases:
   $\mathsf{AG}(\neg(ea_1 \wedge eb_2)) \wedge \mathsf{AG}(\neg(ea_2 \wedge eb_1))$

(2) and (3) together specify the synchronization aspect of the problem: P_1 can never get one whole phase ahead of P_2 and vice-versa.

The structure in Figure Fig. 8 violates the synchronization rules (2) and (3).[1] Our implementation produced the repair shown in Fig. 9.

[1]Note that the bottom $[sa_1 sa_2]$ state is the same as the top $[sa_1 sa_2]$ state, and is repeated only for clarity of the figure.

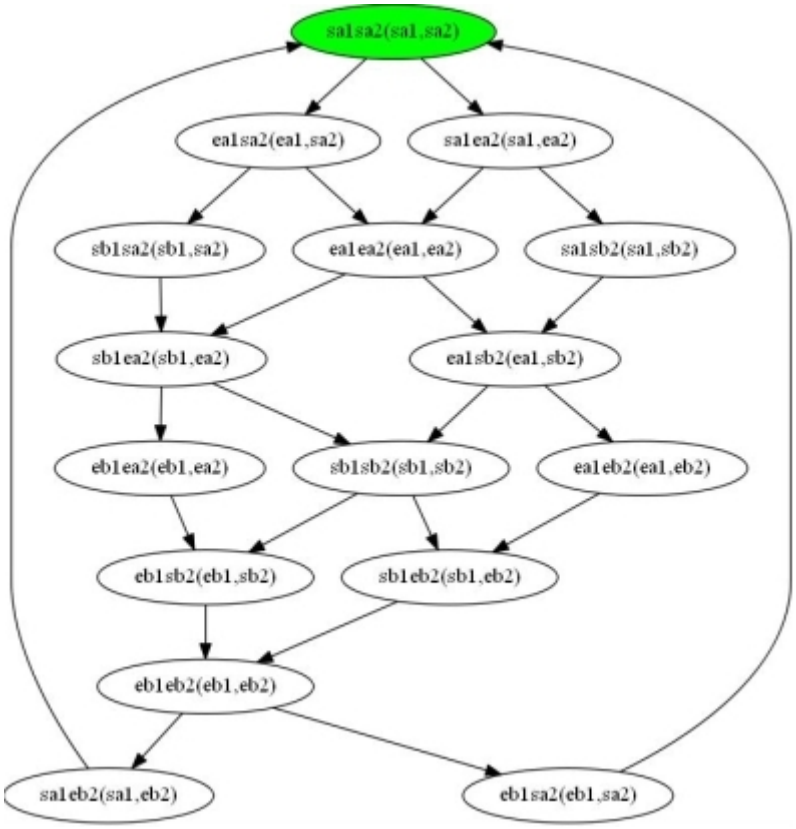

Figure 8: Barrier syncrhonization: initial structure

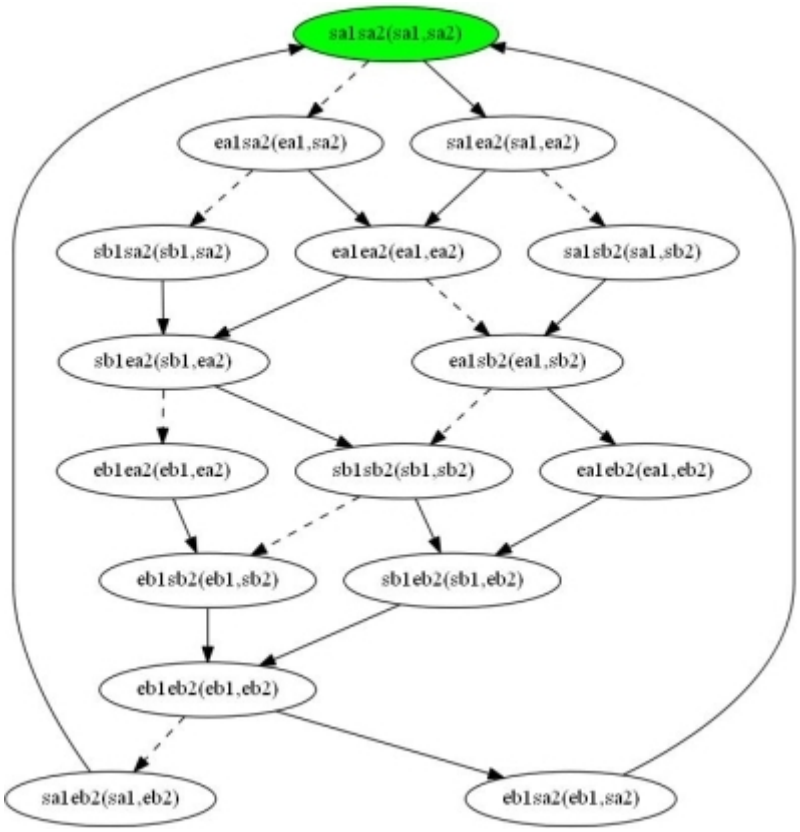

Figure 9: Barrier syncrhonization: repaired structure

6 Extensions of the Subtractive Repair Algorithm

We now present several extensions to the subtractive repair algorithm given in the previous section.

6.1 Optimization

In checking $[\varphi \mathsf{V} \psi]$, we use index n where n is the number of states. This is so that after going down a path for n states, we guarantee that a state has repeated, and so if ψ holds in all states, then we can certify $[\varphi \mathsf{V} \psi]$ along this path. So we actually need n to only be large enough to guarantee that a state is always repeated after at most n transitions. So, n can be the length of the longest simple path in M. The longest path problem is NP-hard, and so we can compute this by again using the SAT-solver. This may have the benefit of reducing the size of the repair formula.

6.2 Addition of States and Transitions

The subtractive repair algorithm performs repair by deleting transitions, with states being implicitly deleted if they become unreachable. Let $M = (s_0, S, R, L)$ be a Kripke structure with underlying set of atomic propositions AP. By adding some states and transitions to M before

performing repair, we can end up with a substructure M' that includes some of the added states. Thus, we have *additive repair*: repair performed by adding states and transitions,

Let S^+ be a finite set of states such that $S \cap S^+ = \emptyset$, let L^+ be an extension of L to $S \cup S^+$, and let R^+ be a subset of $(S \cup S^+) \times (S \cup S^+) - R$. Let $M^+ = (s_0, S \cup S^+, R \cup R^+, L^+)$. So, S^+ represents the states that are added to M, and R^+ represents the transitions that are added. Note that added transitions can involve only the original states (S), only the added states (S^+), or one state from each of S, S^+. We now execute the subtractive repair algorithm of Figure 1, i.e., $\mathsf{Repair}(M^+, \eta)$.

In practice, the added states and transitions would be determined either manually, by the user of the repair tool, or mechanically using heuristics. While it seems possible to modify the repair formula directly to accommodate state/transition addition (e.g., by introducing new propositions for the added states and transitions), doing so does not seem to be any better than adding to the structure M and then regenerating the repair formula using the existing Definition 8. Note that Proposition 2 no longer holds when we add states and transitions.

A useful boundary case is when S^+ is empty, and $R^+ = (S \times S) - R$. In this case, transitions may be both added to and deleted from M to effect a repair, but states can only be deleted. If a model of η that has the same states as M exists, then this approach is guaranteed to find such a model.

6.3 Repair of propositional assignments

By omitting Clause 5 from Def. 8, we remove the constraint that propositional labellings must be preseved. This allows these labelings to be updated as part of the repair.

6.4 Generalized Boolean Constraints on Transition and State Deletion

We can prevent the deletion of a particular transition, say from state s to state t, by conjoining $E_{s,t}$ to $\mathit{repair}(M, \eta)$. This is useful for tailoring the repair, and preventing undesirable repairs, e.g., repairs that prevent a process from requesting a resource.

More generally, we can conjoin arbitrary boolean formulae over the $E_{s,t}$ and X_s to $\mathit{repair}(M, \eta)$, e.g., $E_{s,t} \equiv E_{s',t'} \wedge E_{s',t'} \equiv E_{s",t"}$ adds the constraint that either all three transitions $s \to t$, $s' \to t'$, $s'' \to t''$ are deleted, or none are. Likewise we can add constraints for the removal of states. Such constraints are helpful for the interactive design of finite Kripke structures.

6.5 Dealing with state explosion

A key drawback of our method so far is that Kripke structures, even when used as specifications, may be quite large. To address state-explosion, we extend, in the following three sections of the paper, the basic repair method in three directions:

1. to use *abstraction*: repair an abstract model, and then concretize the repair to obtain a repair of the original model
2. repair of *hierarchical* Kripke structures [1]
3. repair of *concurrent* Kripke structures, as used for example in Statecharts [17]

7 Repair using Abstractions

We now extend the basic repair method to use abstraction mappings, i.e., repair a structure abstracted from M and then concretize the resulting repair to obtain a repair of M. The purpose of abstractions is two fold:

1. *Reduce the size of M, and so reduce the length of repair(M, η).* repair(M, η) has length quadratic in $|M|$, so repairing an abstract structure significantly increases the size of structures that can be handled.
2. *"focus" the attention of the repair algorithm, which in practice produces "better" repairs.* Our repair method nondeterministically chooses a repair from those available, according to the valuation returned by the SAT solver. This repair may be undesirable, as it may result in restrictive behavior, e.g., alternate entry into the critical section. By constructing an abstract structure which, for example, tracks only the values of C_1, C_2, we obtain a better repair, which removes only the transitions that enter `C1C2`.

We introduce four types of abstraction:

1. *abstraction by label* preserves the values of all atomic propositions in η.
2. *abstraction by label with state-adjacency* preserves the values of all atomic propositions in η, and also preserves adjacency of states.
3. *abstraction by formula* preserves the values of a user-selected subset of the propositional subformulae of η.
4. *abstraction by formula with state-adjacency* preserves the values of a user-selected subset of the propositional subformulae of η, and also preserves adjacency of states.

Two states are adjacent iff one is the successor of the other. We define these abstractions as equivalence relations over the states of M. The abstract structure is obtained as the quotient of M by the equivalence relations. We provide a concretization algorithm (Sect. 7.3) which maps the repair of the abstract structure back to the original (concrete) structure, to produce a possible repair of the original structure. Since the resulting repair can always be model-checked at no significant algorithmic expense, we use abstractions that are not necessariy correctness-preserving, since we can, in many cases, obtain larger reductions in the size of the structure than if we used only correctness-preserving abstractions.

7.1 Abstraction with respect to atomic propositions

In order to abstract our model with respect to atomic propositions we start by defining two equivalence relations, $\equiv_{p,a}$ and $\equiv_p$. Let AP_η be the set of atomic propositions that occur in η.

The first equivalence relation represents an abstraction strategy which takes adjacency of states into consideration.

Definition 9 (Adjacency-respecting propositional abstraction, $\equiv_{p,a}$)**.** *Given a Kripke structure $M = (S_0, S, R, L, AP)$ and a CTL formula η, we define an equivalence relation $\equiv_{p,a}$ over S as follows:*

- $s \approx_{p,a} t \stackrel{\text{df}}{=\!=} (L(s) \cap AP_\eta = L(t) \cap AP_\eta) \wedge ((s,t) \in R \vee (t,s) \in R)$
- $s \equiv_{p,a} t \stackrel{\text{df}}{=\!=} \approx^*_{p,a}$

that is, $s \approx_{p,a} t$ iff s and t agree on all of the atomic propositions of η, and there is a transition from s to t or vice-versa, and $\equiv_{p,a}$ is the transitive closure of $\approx_{p,a}$

The next equivalence relation, $\equiv_p$, ignores adjancency of states. This can result in the removal of existing cycles, or the introduction of new ones.

Definition 10 (Adjacency-ignoring propositonal abstraction, $\equiv_p$)**.** *Given a Kripke structure $M = (S_0, S, R, L, AP)$ and a specification formula η, we define an equivalence relation $\equiv_p$ as follows:*

- $s \equiv_p t \stackrel{\text{df}}{=\!=} L(s) \cap AP_\eta = L(t) \cap AP_\eta$

that is, $s \equiv_p t$ iff s and t agree on all of the atomic propositions of η.

Both $\equiv_{p,a}$ and $\equiv_p$ are equivalence relations over S, and they also preserve the values of all the atomic propositions in η. Hence, we can define a quotient of M by these relations as usual.

Definition 11 (Abstract Model)**.** *Let $\equiv \in \{\equiv_{p,a}, \equiv_p\}$. Given a Kripke structure $M = (S_0, S, R, L, AP)$ and a specification formula η, we define the abstract model $\overline{M} = (\overline{S}_0, \overline{S}, \overline{R}, \overline{L}, AP_\eta) = M/\equiv$ as follows:*

1. $\overline{S} = \{[s] \mid s \in S\}$
2. $\overline{S_0} = \{[s_0] \mid s_0 \in S_0\}$
3. $\overline{R} = \{([s],[t]) \mid (s,t) \in R\}$
4. $\overline{L} : \overline{S} \to AP$ *is given by* $\overline{L}([s]) = L(s) \cap AP_\eta$

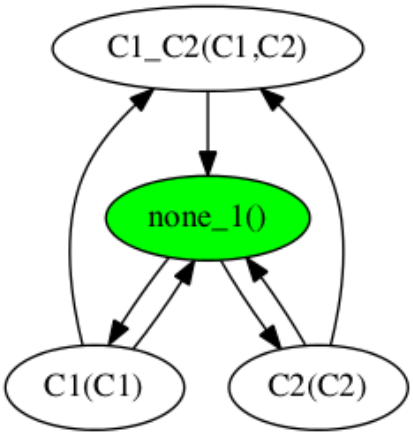

Figure 10: Mutex: abstraction by label (i.e., by atomic propositions)

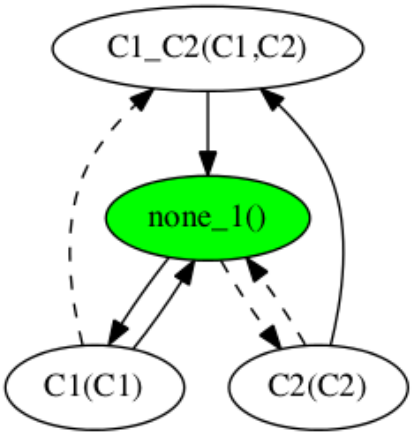

Figure 11: Mutex: bad repair of abstract-by-label structure

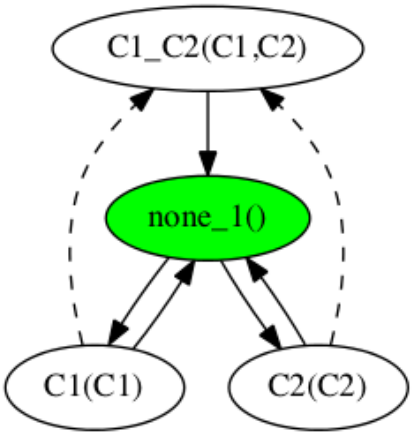

Figure 12: Mutex: good repair of abstract-by-label model

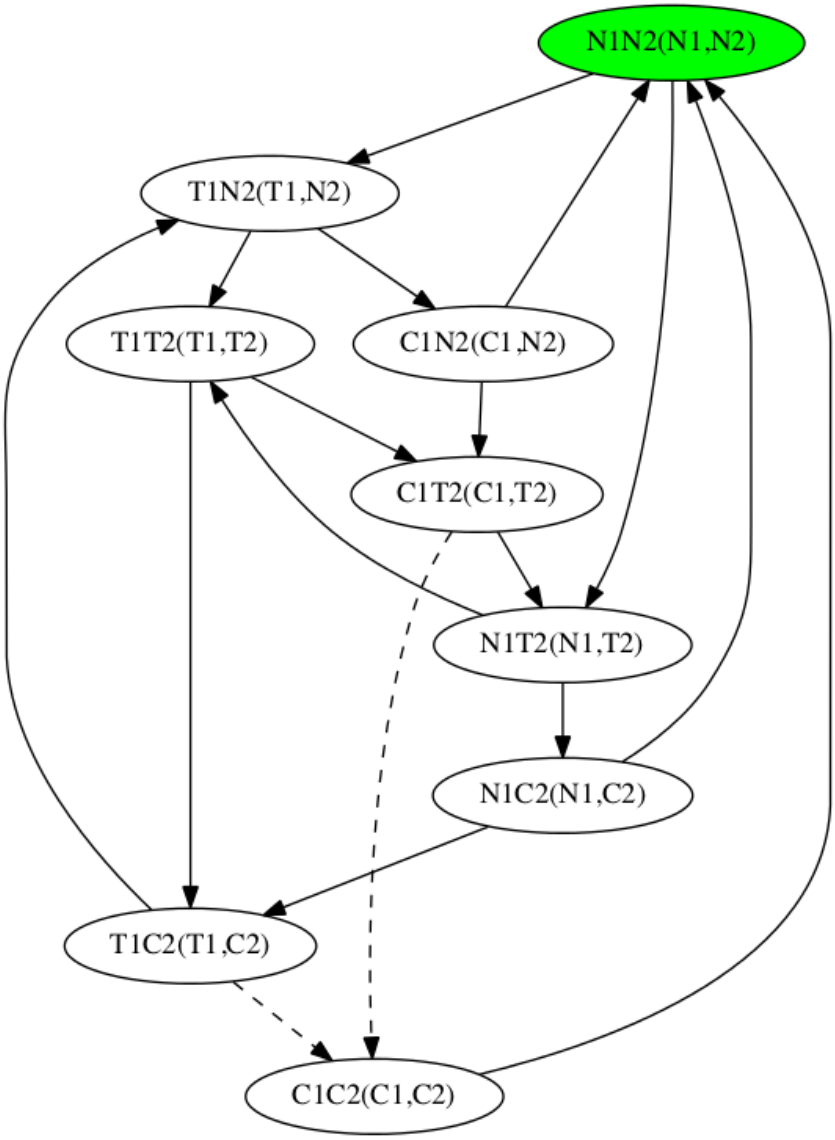

Figure 13: Mutex: concretization of repaired abstract-by-label structure

Consider the two-process mutual exclusion structure M in Fig. 3, with $\eta = \mathsf{AG}(\neg(C_1 \wedge C_2))$, and abstraction by label, i.e., states that agree on both C_1 and C_2 are considered equivalent. By Definition 10 the equivalence classes of $\equiv_p$ are: $\texttt{none\_1} = \{\texttt{N1N2}, \texttt{N1T2}, \texttt{T1N2}, \texttt{T1T2}\}$, $\texttt{C1} = \{\texttt{C1N2}, \texttt{C1T2}\}$, $\texttt{C2} = \{\texttt{N1C2}, \texttt{T1C2}\}$, $\texttt{C1\_C2} = \texttt{C1C2}$. Fig. 10 shows the resulting abstract structure $\overline{M}$, which has four states, corresponding to these equivalence classes. Figure 11 presents a repair of $\overline{M}$ (recall that transitions to be deleted are shown dashed). This is not a good repair, since the state C2 has been made unreachable, so that process 2 cannot now ever enter its critical section. To address this, we checked `retain` for transitions $\texttt{none\_1} \rightarrow \texttt{C1}$, $\texttt{none\_1} \rightarrow \texttt{C2}$, since the critical sections should always be reachable. Figure 12 shows the resulting repair, and Fig. 13 gives the concretization of this repair to the full model of Fig. 3. Notice that this is the "best" repair, in that the minimum number of transitions needed to effect the repair are deleted, and no state other than the "bad" state C1C2 is made unreachable. More importantly, the crucial frame property that "processes can always request access" is now satisfied.

7.2 Abstraction with respect to sub-formulae

We can obtain larger reductions in the size of the state space by dropping the requirement that we preserve the values of all the atomic propositions in η, cf. predicate abstraction. In some cases, it may be necessary only to preserve the values of some propositional subformulae in η. For example, in mutual exclusion, only the value of $C_1 \wedge C_2$ is of interest.

Let AS_η be a set of propositional sub-formulae of η that is specified by the user. Let $SUB(t) \subseteq AS_\eta$ be the set of sub-formulae in AS_η that are satisfied by the state t.

- $SUB(t) \stackrel{\text{df}}{=} \{f \mid f \in AS_\eta \text{ and } M, t \models f\}$

Definition 12 (Adjacency-respecting subformlua abstraction, $\equiv_{s,a}$)**.** *Given a Kripke structure $M = (S_0, S, R, L, AP)$ and a CTL formula η, we define an equivalence relation $\equiv_{s,a}$ over S as follows:*

- $s \approx_{s,a} t \stackrel{\text{df}}{=} (SUB(s) \cap AS_\eta = SUB(t) \cap AS_\eta) \wedge ((s,t) \in R \vee (t,s) \in R)$
- $\equiv_{s,a} \stackrel{\text{df}}{=} \approx^*_{s,a}$

that is, $s \equiv_{s,a} t$ iff s and t agree on all of the sub-formulae, and there is a transition from s to t or vice-versa and $\equiv_{s,a}$ is the transitive closure of $\approx_{s,a}$ We also define $[s] \stackrel{\text{df}}{=} \{t \mid s \equiv t\}$.

Definition 13 (Adjacency-ignoring subformula abstraction, $\equiv_{s,p}$)**.** *Given a Kripke structure $M = (S_0, S, R, L, AP)$ and a CTL formula η, we define an equivalence relation $\equiv_{s,p}$ over S as follows:*

- $s \equiv_{s,p} t \stackrel{\text{df}}{=} SUB(s) \cap AS_\eta = SUB(t) \cap AS_\eta$

that is, $s \equiv_{s,p} t$ iff s and t agree on all of the sub-formulae in AS_η.

Definition 14 (Abstract model)**.** *Let $\equiv \in \{\equiv_{s,a}, \equiv_{s,p}\}$. Given a Kripke structure $M = (S_0, S, R, L, AP)$ and a specification formula η, we define the reduced model $\overline{M} = M/\equiv$ as follows:*

1. $\overline{S} = \{[s] \mid s \in S\}$
2. $\overline{S_0} = \{[s_0] \mid s_0 \in S_0\}$
3. $\overline{R} = \{([s],[t]) \mid (s,t) \in R\}$
4. $\overline{L} : \overline{S} \to AP$ *is given by* $\overline{L}([s]) = \bigcap_{t \in [s]} L(t)$*. That is, the label consists of the atomic propositions, if any, that hold in all states of* $[s]$*.*

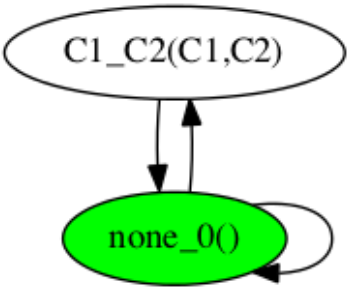

Figure 14: Mutex: abstraction by subformulae

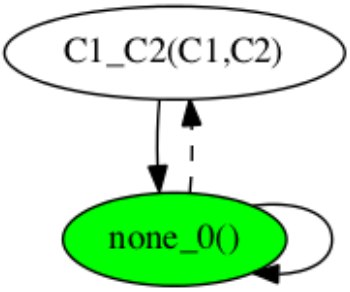

Figure 15: Mutex: repair of abstract-by-subformulae structure

Consider the two-process mutual exclusion structure M in Fig. 3, with $\eta = \mathsf{AG}(\neg(C_1 \wedge C_2))$, and abstraction with respect to sub-formula $(C1 \wedge C2)$, i.e., states that agree on the value of $(C1 \wedge C2)$ are considered equivalent. By Definition 13, the equivalence classes of $\equiv_{s,p}$ are $\texttt{none\_1} = \{\texttt{N1N2}, \texttt{N1T2}, \texttt{T1N2}, \texttt{T1T2}, \texttt{C1N2}, \texttt{C1T2}, \texttt{N1C2}, \texttt{T1C2}\}$ and $\texttt{C1\_C2} = \texttt{C1C2}$. Fig. 14 shows the resulting abstract structure $\overline{M}$, which has two states, corresponding to these equivalence classes.

Figure 15 gives the repair of Fig. 3 w.r.t. $\mathsf{AG}(\neg(C1 \wedge C2))$. The concretization of this repair to the full model of Fig. 3 gives Fig. 13, the same repair that we obtained from abstraction by label. Unlike abstraction by label however, we obtained this repair immediately, and did not have to use the `retain` button.

7.3 Concretizing an abstract repair to repair the original structure

Abstract repair does not guarantee concrete repair. When we concretize the repair of $\overline{M}$, we only to obtain a *possible* repair of M, which we then verify by model checking. We concretize as follows. The abstraction algorithm keeps a `Map` data structure that maps a transition in $\overline{M}$ to the set of corresponding transitions in M. If a transition in $\overline{M}$ is deleted by the repair of $\overline{M}$, then we delete all the corresponding transitions in M to construct the possible repair of M. The benefit of such abstractions is that we can, in many cases, obtain larger reductions in the state-space than if we used only repair-preserving abstractions.

7.4 Example: barrier synchronization

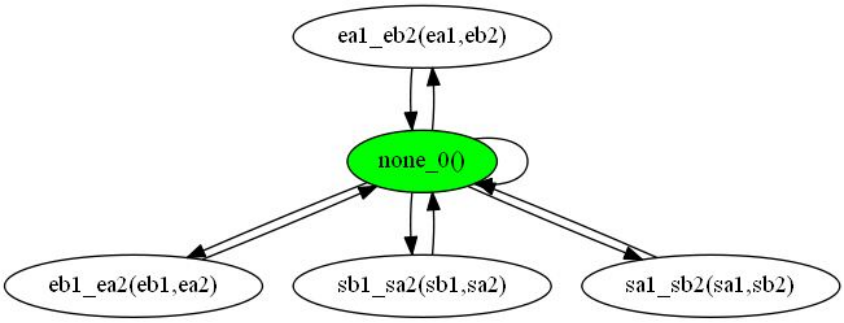

Figure 16: Barrier synchronization: abstraction by subformulae

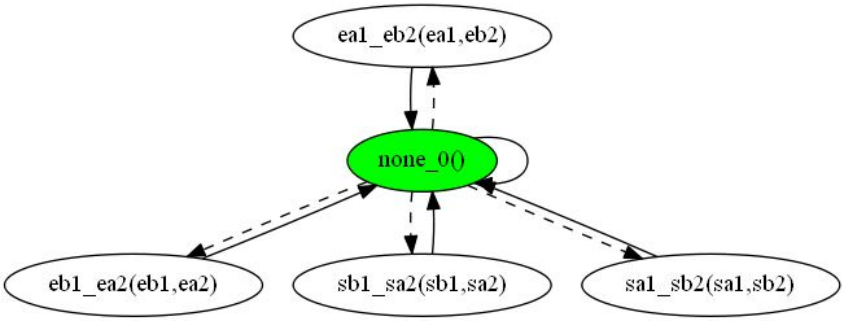

Figure 17: Barrier synchronization: repair of abstract-by-subformulae structure

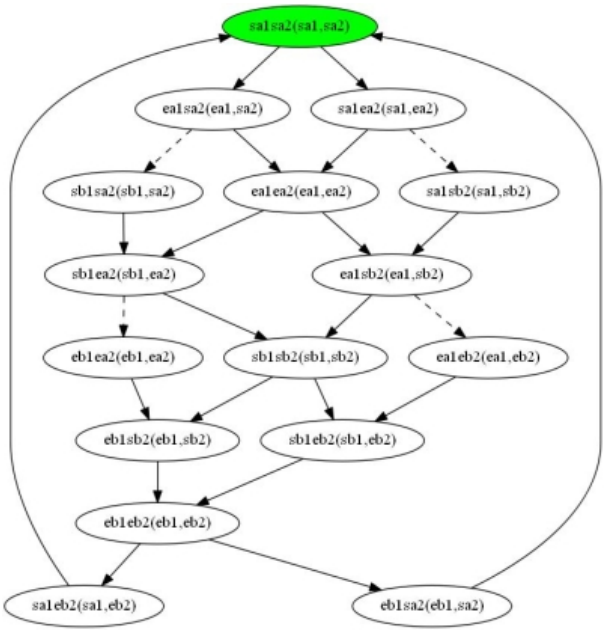

Figure 18: Barrier synchronization: concretization of repaired abstract-by-subformulae structure

8 Repair of Hierarchical Kripke Structures

We now extend our repair method to hierarchical Kripke structures. As given by Alur & Yannakakis [1], a hierarchical Kripke structure K over a set P of atomic propositions is a tuple $K_1, \ldots, K_n$ of structures, where each K_i has the following components:

1. A finite set N_i of nodes.
2. A finite set B_i of boxes (or supernodes). The sets N_i and B_i are all pairwise disjoint.
3. An initial node $start_i \in N_i$
4. A subset O_i of N_i, called exit nodes.
5. A labeling function $X_i : N_i \to 2^P$ that labels each node with a subset of P
6. An indexing function $Y_i : B_i \to \{i+1 \ldots n\}$ that maps each box of the i-th structure to an index greater than i. That is, if $Y_i(b) = j$, for a box b of structure K_i, then b can be viewed as a reference to the definition of the structure K_j.
7. An edge (transition) relation E_i. Each edge in E_i is a pair (u, v) with source u and sink v:
   - source u either is a node of K_i , or is a pair $(w1, w2)$, where $w1$ is a box of K_i with $Y_i(w1) = j$ and $w2$ is an exit-node of K_j;
   - sink v is either a node or a box of K_i.

For simplicity of notation and exposition, we restrict the discussion to two levels of hierarchy and one kind of box only, which we refer to as B, and we assume a single occurrence of the box B in M. We assume, without loss of generality, that the start and exit states of box B do not lie on a cycle wholly contained in B. We regard a box as performing some subsidiary computation, and signalling the result by means of selecting a particular exit state.

Our method for repairing M w.r.t. η is as follows:

1. Write a specification formula η_B for B; we assume (by induction on hierarchy level) that B has been repaired w.r.t. η_B. We infer, under suitable assumptions, that M satisfies η_B
2. Write a "coupling" formula φ which relates the behavior of B to that of M. We repair M w.r.t. φ.
3. Prove $\models \eta_B \wedge \varphi \Rightarrow \eta$, i.e., that $\eta_B \wedge \varphi \Rightarrow \eta$ is a CTL validity. We do this by implementing the CTL decision procedure [15] and checking satisfiability of the negation. Hence, from the previous two steps, infer $M \models \eta$.

We show below how to effect these repairs without incurring an generate an exponential blowup in size of the structure being repaired. To show that the method is sound, we use weak (i.e., stuttering) forward simulations [8, 16], which means that our results are restricted to ACTL-X, the universal fragment of CTL without nexttime [16].

Definition 15 (Weak forward simulation)**.** *Let $M = (S_0, S, R, L, AP)$ and $M' = (S'_0, S', R', L', AP)$ be Kripke structures such that $AP \supseteq AP'$. Let $AP'' \subseteq AP'$. A relation $H \subseteq S \times S'$ is a weak*

forward simulation relation w.r.t. AP'' and from t to t' iff (1) $H(t,t')$, and (2) for all s, s'; $H(s,s')$ implies: (2a) $L(s) \cap AP'' = L(s')$, and (2b) for every fullpath π from s in M, there exists a fullpath π' from s' in M' and partitions $B_1B_2\ldots$ of π, $B'_1B'_2\ldots$ of π', such that for all $i \geq 1$, B_i, B'_i are both nonempty and finite, and every state of B_i is H-related to every state of B'_i.

Write $M \leq_{AP''} M'$ iff there exists a weak forward simulation H w.r.t. AP'' and such that $\forall s \in S_0 \exists s' \in S'_0 : H(s,s')$. Note that we do not consider fairness here. Let A be a set of atomic propositions.

Proposition 3. *Suppose $M \leq_{AP''} M'$ for some set AP'' of atomic propositions. Then, for any ACTL-X formula f over AP'', if $M' \models f$ then $M \models f$.*

Proof. Adapt the proof of Theorem 3 in Grumberg and Long [16] to deal with stuttering. That is, remove the case for AX, and adapt the argument for AU to deal with the partition of fullpaths into blocks. The details are straightforward. □

We make the simplifying technical assumption that there is a bijection between states and propositions, so that each proposition holds in exactly its corresponding state, and does not hold in all other states. This amounts to an assumption of "alphabet disjointness" between M and B: M invokes B by entering $start_B$, and B returns a result by selecting a particular exit state. This assumption then amounts to an "information hiding" principle for hierarchical Kripke structures.

8.1 Verification of the box specification

To infer $M \models \eta_B$ from $B \models \eta_B$, we construct a version of B which reflects the impact on B of being placed inside M. We call this B_M, the "M-situated" version of B.

Definition 16 (M-situated version of B)**.** *The M-situated version of B, denoted B_M, is as follows. Include all the states and transitions of B. Add two "interface" states pre_B abd $post_B$. Add a transition from pre_B to $start_B$, the start state of B, and a transition from every $s \in O_B$ (i.e., every exit state of B) to $post_B$. If, in M, there is a path from some $s \in O_B$ back to $start_B$, then add a transition from $post_B$ to pre_B. If, in M, there is a path π from the start state of M to an exit state of M such that π does not enter B, then add a transition from pre_B to $post_B$.*

pre_B represents all states of M from which the start state of B is reachable. $post_B$ represents all states of M that are reachable from some exit state of B.

Let AP_B be the set of atomic propositions corresponding to states in B, including the start and all exit states.

Proposition 4. $M \leq_{AP_B} B_M$

Proof. Construct a forward simulation f from M to B_M as follows. A state of M that is in B is mapped to "itself". A state s of M that is not in B is mapped as follows: if $start_B$ is reachable from s, then relate s to pre_B, and if a state t is reachable from s along a path outside of B, such that $start_B$ is not reachable from t, then relate s to $post_B$. □

Corollary 2. *For any ACTL-X formula f over AP_B, if $B_M \models f$ then $M \models f$.*

Proof. Follows immediately from Prop. 3 and Prop. 4. □

8.2 Verification of the coupling specification

We define the abstraction B_A of B as follows.

Definition 17 (Abstract box)**.** *Let O_B^r be the subset of O_B consisting of all exit states that reachable from $start_B$. The states of B_A are $\{start_B, int_B\} \cup O_B^r$, i.e., the start state, all reachable exit states, and a new state int_B, which represents the interior of B. int_B has the empty propositional labelling.*

If B is acyclic, then the transitions of B_A are $\{(start_B, int_B)\} \cup \{(int_B, s) \mid s \in O_B^r\}$, i.e., there is a transition from the start state to the interior state, and from the interior state to every reachable exit state.

If B contains cycles, then we also add the transition (int_B, int_B), i.e., a self-loop on int_B, which models the possiblity of remaining inside B forever.

The reachability problem for hierarchical Kripke structures is solvable by a polynomial time depth-first search, see Theorem 1 of Alur & Yannakakis [1].

Let M_A be the result of replacing B by B_A in M, and let AP_C be the set of atomic propositions corresponding to the start state of B, the exit states of B, and all states of M that are not in B.

Proposition 5. $M \leq_{AP_C} M_A$.

Proof. Construct a forward simulation f from M to M_A as follows. A state of M that is not in B is mapped to "itself" in M_A. Likewise, the start state of B and every exit state of B are mapped to themselves (this is possible since B_A contains these states). Internal states of B are all mapped to the state int_B of B_A. □

Corollary 3. *For any ACTL-X formula f over AP_C, if $M_A \models f$ then $M \models f$.*

Proof. Follows immediately from Prop. 3 and Prop. 5. □

8.3 Heirarchical repair

Theorem 4. *If $M_A \models \varphi$, $B_M \models \eta_B$, and $\models \eta_B \wedge \varphi \Rightarrow \eta$, then $M \models \eta$.*

Proof. From $B_M \models \eta_B$ and Cor. 2, we have $M \models \eta_B$. From $M_A \models \varphi$ and Cor. 3, we have $M \models \varphi$. From $M \models \eta_B$, $M \models \varphi$, and $\models \eta_B \wedge \varphi \Rightarrow \eta$, we have $M \models \eta$. □

8.4 Example: phone system

We illustrate hierarchical repair using the phone call example from [1]. Fig. 19 shows (the M-situated version of) a box B that attempts to make a phone call; from the start state `send`, we enter a waiting state `wait`, after which there are three possible outcomes: `timeout`, `negAck` (negative acknowledgement), and `posAck` (positive acknowledgement). `timeout` and `negAck` lead to failure, i.e., state `fail`, and `posAck` leads to placement of the call, i.e., state `ok`. Fig. 21 shows M_A, the overall phone call system, with B replaced by B_A. M_A makes two attempts, and so contains two instances of B_A. If the first attempt succeeds, the system should proceed to the

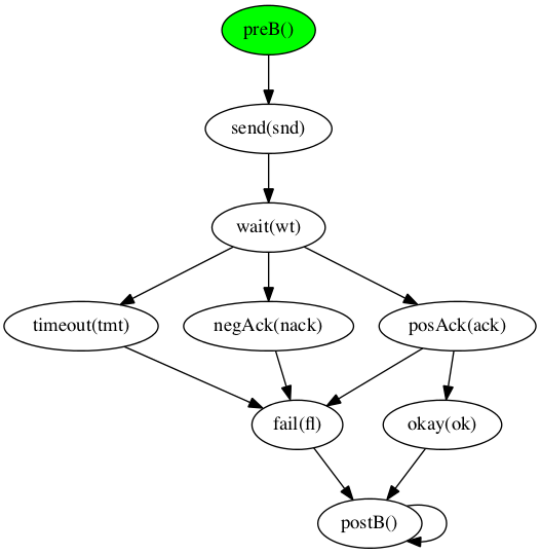

Figure 19: Box B_M for a single phone-call

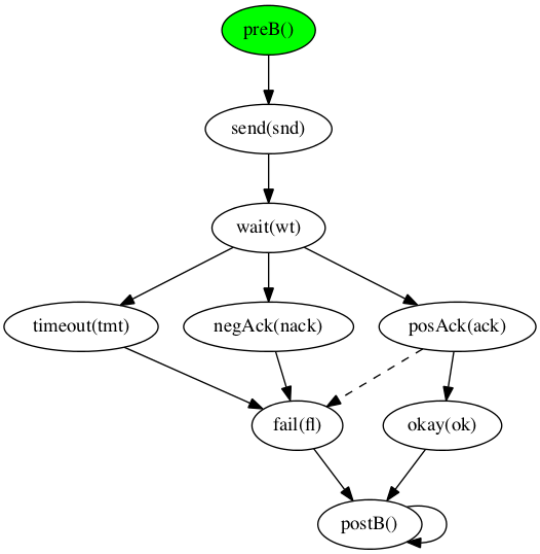

Figure 20: Repaired box B_M for a single phone-call

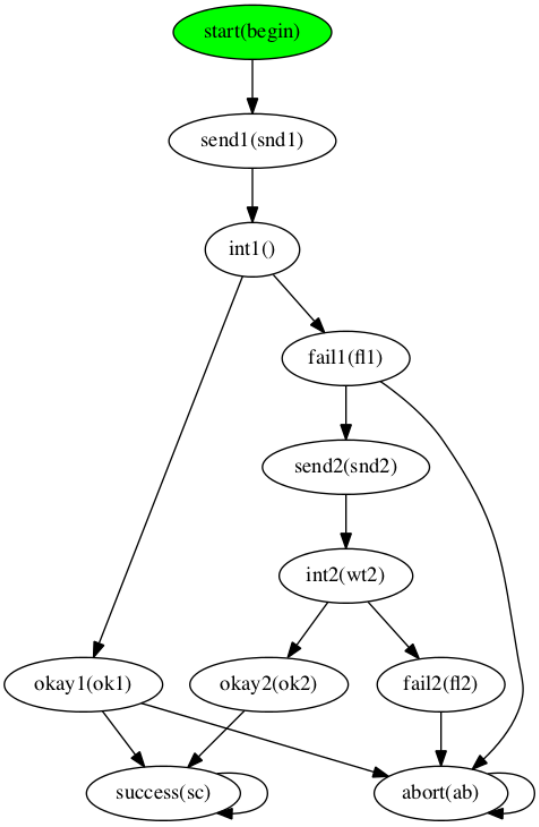

Figure 21: Initial structure M for phone-call example

success final state. If the first attempt fails, the system should proceed to the start state of the second attempt. If the second attempt succeeds, the system should proceed to the success final state, while if the second attempt fails, the system should proceed to the abort final state. The relevant formulae are:

1. specification formula η for M: $\mathsf{AG}((ack_1 \vee ack_2) \Rightarrow \mathsf{AF}sc)$, i.e., if either attempt receives a positive ack, then eventually enter success state.

2. specification formula η_B for B: $\mathsf{AG}(ack \Rightarrow \mathsf{AF}ok)$, i.e., a positive ack implies that the phone call will be placed. We repair B_M w.r.t. η_B using Eshmun: the transition from `posAck` to `fail` is deleted, as shown in Fig. 20.

3. "coupling" formula φ: $\mathsf{AG}((ok_1 \vee ok_2) \Rightarrow \mathsf{AF}sc) \wedge \mathsf{AG}(fl1 \Rightarrow \mathsf{AF}(snd2))$, i.e., if either call is placed, then eventually enter success state, and if first attempt fails, go to second attempt. We repair M_A w.r.t. φ using Eshmun, checking `retain` for all internal transitions of B_A: `send` $\rightarrow$ `int`, `int` $\rightarrow$ `ok`, `int` $\rightarrow$ `fl`. The transitions from `fail` to `abort` and `okay1` to `abort` are deleted, as shown in Fig. 22.

Eshmun checks the validity of $\eta_B \wedge \varphi \Rightarrow \eta$ by using the CTL decision procedure of [15]: we check satisfiability of $\neg(\eta_B \wedge \varphi \Rightarrow \eta)$. By Th. 4, we conclude that $M \models \eta$.

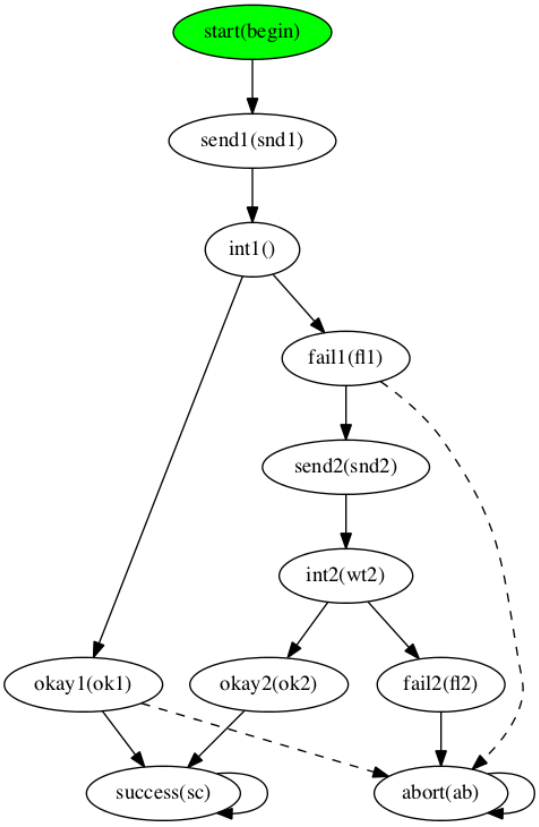

Figure 22: Repaired structure M for phone-call example

9 Repair of Concurrent Kripke Structures

We now consider several Kripke structures, which "execute" in parallel. As a boundary case, consider several Kripke structures $M_1, \ldots, M_n$ which do not interact, and which are to be repaired w.r.t. CTL formulae $\eta_1, \ldots, \eta_n$, respectively. This can be effected "in one shot" using the repair formula $\mathsf{Repair}(M_1, \eta_1) \wedge \cdots \wedge \mathsf{Repair}(M_n, \eta_n)$. This works since the repairs are independent, because the structures do not interact. So, what must be done to handle the case when structures do interact. The details of this depend of course on the precise interaction mechanism, e.g., shared variables (cf. synchronization skeletons of [15]) or shared events (cf. I/O automata [22] and BIP [6, 24]).

We consider interaction via shared events, where the events are transitions of a Kripke structure. Specifically, we consider the *pairwise composition* method presented in [2, 3], to which we refer the reader for full details. We summarize the method as follows. The Kripke structures are *multiprocess* Kripke structures. A multiprocess Kripke structure has its set AP of atomic porpositions partitioned into $AP_1 \cup \cdots \cup AP_K$, which are the atomic propositions of processes $P_1, \ldots, P_K$ respectively. Also, every transition is labeled with the index of a single process, which executes the transition. Only atomic propositions belonging to the executing process can be changed by a transition.

A *pair-structure* $M_{ij} = (S_0^{ij}, S_{ij}, R_{ij}, L_{ij}, AP_{ij})$ is a multiprocess Kripke structure over two process indices, e.g., i, j. Its set of atomic propositions is $AP_i \cup AP_j$. M_{ij} defines the direct interaction between processes P_i and P_j. If P_i interacts directly with a third process P_k, then a second pair structure, $M_{ik} = (S_0^{ik}, S_{ik}, R_{ik}, L_{ik}, AP_{ik})$, over indices i, k, defines this interaction.

Note that M_{ij} and M_{ik} have the atomic propositions AP_i in common, and so their parallel composition must obey the following consistency condition: in any reachable global state s, the corresponding (i.e., projected) local states s_{ij} of M_{ij} and s_{ik} of M_{ik} must agree on all the atomic poropositions in AP_i. Formally, in the parallel composition of M_{iJ} and M_{ik}, a global state can be viewed as having the form $\langle s_{ij}, s_{ik} \rangle$, where s_{ij} is a state of M_{ij}, and s_{ik} is a state of M_{ik}.[2] Then, we require, $\forall p \in AP_i : p \in L_{ij}(s_{ij})$ iff $p \in L_{ik}(s_{ik})$.

The consistency condition requires that a transition by some process P_i must be executed *synchronously* in every Kripke structure that P_i is represented in. A consequence of this is that P_i must have the same "local structure" in all of its pair-structures: if, in M_{ij}, there exists some transition by P_i that changes the local state of P_i from s_i to t_i, then there must also be a transition in M_{ik}, by P_i that changes the local state of P_i from s_i to t_i. In general, there may be several such transitions in each of M_{ij} and M_{ik}. Let these transitions in M_{ij} be $tr^j_{i1}, \ldots, tr^j_{in}$, and in M_{ik} be $tr^k_{i1}, \ldots, tr^k_{im}$. Then we conjoin

$$\bigvee_{1 \leq x \leq n} E(tr^j_{ix}) \equiv \bigvee_{1 \leq y \leq m} E(tr^j_{iy})$$

to the repair formula, where $E(tr)$ is the repair proposition corresponding to transition tr. Note that we are using the generalized boolean constraints of Section 6.4.

10 Overview of our Implementation: The Eshmun Tool

We have implemented the repair method as the Eshmun[3] too, available at `http://eshmuntool.blogspot.com/`. Eshmun is written in Java, and uses the `javax.swing` library for GUI functionality, `Graphviz` [13] for Kripke structure visualization, and `SAT4jSolver` [21] to check satisfiability.

10.1 Main modules

The following is a concise definition of our tool's main modules:

- CTL Parser: parses a CTL formula ϕ to generate a CTLParsedTree object which is a tree data structure representing ϕ.
- User Interface: implements GUI interface between user and the other modules.
- Model Checker: takes as input a Kripke structure $M = (S_0, S, R, L, AP)$, and a CTL formula ϕ and verifies if M satisfies ϕ.
- Model Repairer: takes as input a Kripke structure M and a CTL formulae ϕ and return a repaired model with respect to ϕ.
- Model Optimizer: reduces the state space of created Kripke structures. It implements the abstraction methods in section 7.

[2] The more traditional formulation as a tuple of local states and shared variable values is used in [2, 3], but the two views are equivalent.

[3] Eshmun is the name of the Phoenician god of healing

- SAT Solver: takes as input a CNF file and return a flag that specifies whether the CNF formulae is satisfiable or not. In case it is satisfiable it also returns the satisfying valuation.
- Decision Procedure: this module implements the CTL decision procedure given in [15]

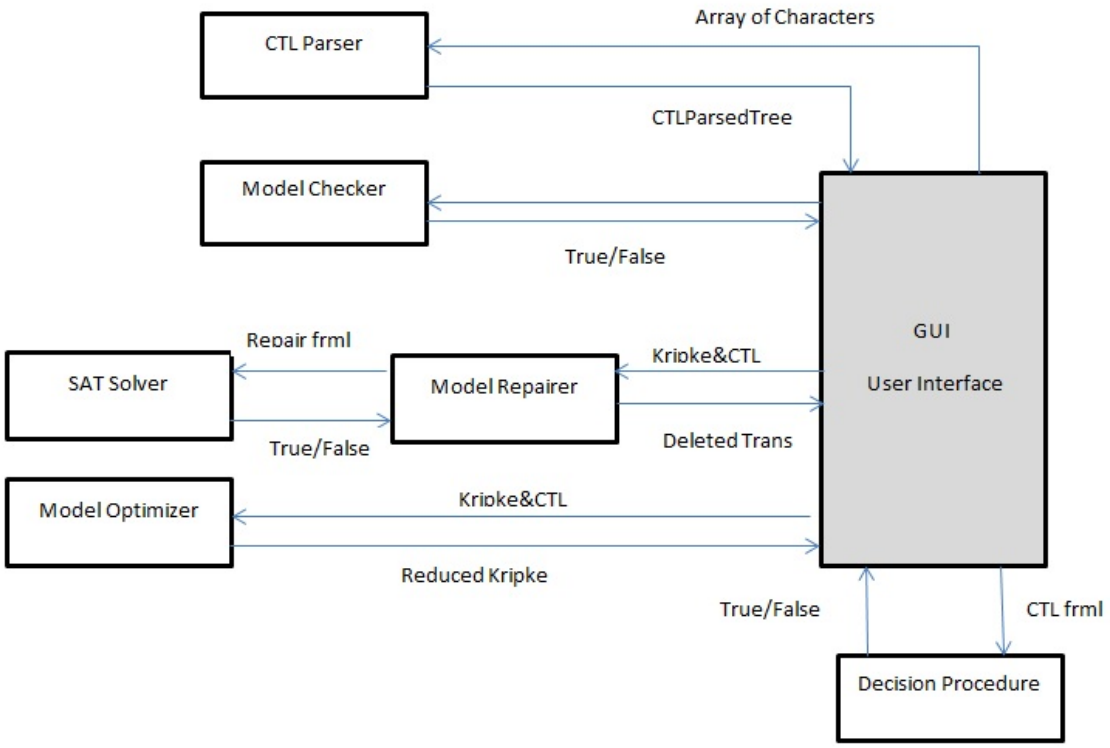

Figure 23: Tool main modules

10.2 Pseudocode for computing $repair(M, \eta)$

Figure 25 gives pseudocode for our algorithm to compute $repair(M, \eta)$ from $M = (S_0, S, R, L, AP)$ and η. The algorithm operates as follows. We introduce a label $\mathcal{L}(s)$ for each state s in M. $\mathcal{L}(s)$ is a subset of $sub(\eta)$. Initially, $\mathcal{L}(s_0) = \eta$, and $\mathcal{L}(s) = \emptyset$ for all $s \in S - \{s_0\}$. The algorithm propagates formulae from the label of some state s to the labels of all successor states t of s. This propagation is performed according to Definition 8, so that if, for some CTL formula φ, $\varphi \in \mathcal{L}(s)$, and Definition 8 requires that some other CTL formula ψ (related to φ) be evaluated in every successor t, then we add ψ to $\mathcal{L}(t)$. For example, suppose $\mathsf{A}[\varphi \mathsf{V} \psi] \in \mathcal{L}(s)$. Then, for every successor t of s, we must add φ, ψ, and $\mathsf{A}[\varphi \mathsf{V} \psi]$ to $\mathcal{L}(t)$. Note that for each $\xi \in \mathcal{L}(s)$, we propagate at most one formula to the successors of s. Once $\xi \in \mathcal{L}(s)$ has been processed in this manner, we "mark" it, so that we do not repeat the propagation. We introduce a boolean array $marked(s, \varphi)$ for this purpose. When a propagation is performed, the appropriate conjunct is added to $repair(M, \eta)$. For the release modality, we include the index with the propagated formulae, so that we can "count down" properly. We summarize the data structures used:

- $repair(M, \eta)$: a string, which accumulates the repair formula which is being computed
- $\mathcal{L}(s)$: a subset of $sub(\eta)$. Contains the formulae which have been propagated to s, and whose truth in s affects the truth of η in s_0.

- $marked(s, \xi)$: a boolean array, initially all false. An entry is set to true when formula $\xi \in \mathcal{L}(s)$ has been processed.

Figure 24 gives the overall algorithm ComputeRepairFormula. We initialize $repair(M, \eta)$ by invoking InitializeRepairFormula(,) which sets $repair(M, \eta)$ to the conjunction of Clause 2–5 of Definition 8. These clauses do not depend on the transitions in M, and so can be computed without traversing M. Figure 25 gives the propagation step propagate, which propagates formulae from the label $\mathcal{L}(s)$ of s to the labels of the sucessor states $t \in R[s]$ of s. When a propagation is performed, propagateinvokes conjoin(given in Figure 27), which updates $repair(M, \eta)$, according to Definition 8, by conjoining the appropriate clause.

ComputeRepairFormula($repair(M, \eta)$)

InitializeRepairFormula(M, η);
forall $s_0 \in S_0 : new(s) := \{\eta\}$ **endfor** ; $\triangleright \eta$ must hold in all initial states
repeat until no change
    select some state s in M and some $\xi \in new(s)$;
    propagate(s, ξ)

Figure 24: The model repair algorithm.

11 Related Work

The use of transition deletion to repair Kripke structures was suggested in Attie and Emerson [4,5] in the context of atomicity refinement: a large grain concurrent program is refined naively (e.g., by replacing a test and set by the test, followed nonatomically by the set). In general, this may introduce new computations (corresponding to "bad interleavings") that violate the specification. These are removed by deleting some transitions.

The use of model checking to generate counterexamples was suggested by Clarke et. al. [12] and Hojati et. al. [18]. [12] presents an algorithm for generating counterexamples for symbolic model checking. [18] presents BDD-based algorithms for generating counterexamples ("error traces") for both language containment and fair CTL model checking. Game-based model checking [23, 27] provides a method for extracting counterexamples from a model checking run. The core idea is a *coloring algorithm* that colors the nodes in the model-checking game graph which contribute to violation of the formula being checked.

The idea of generating a propositional formula from a model checking problem was presented in Biere et. al. [7]. That paper considers LTL specifications and bounded model checking: given an LTL formula f, a propositional formula is generated that is satisfiable iff f can be verified within a fixed number k of transitions along some path ($\mathsf{E}f$). By setting f to the negation of the required property, counterexamples can be generated. Repair is not discussed.

Some authors [20, 25, 26] have considered algorithms for solving the repair problem: given a program (or circuit), and a specification, how to automatically modify the program (or circuit), so that the specification is satisfied. There appears to be no automatic repair method that is (1) complete (i.e., if a repair exists, then find a repair) for a full temporal logic (e.g., CTL, LTL), and (2) repairs all faults in a single run, i.e., deals implicitly with all counterexamples "at once." For example, Jobstmann et. al. [20] considers only one repair at a time, and their method is complete only for invariants. In Staber et. al. [25], the approach of Jobstmann et. al. [20] is extended so

that multiple faults are considered at once, but at the price of exponential complexity in the number of faults.

In Buccafurri et. al. [9] the repair problem for CTL is considered and solved using abductive reasoning. The method generates repair suggestions that must then be verified by model checking, one at a time. In contrast, we fix all faults at once. Zhang and Ding [28] present a model repair method (which they call "model update" based on a set of five primitive update operations: add a transition, remive a transition, change the propositional labeling of a state, add a state, and remove an isolated state (one that has no incident transitions). They also present a "minimum change principle" , which essentially states that the repaired model retains as much as possible of the information present in the original model. Their repair algorithm runs in time exponential in $|\eta|$ and quadratic in $|M|$. Their algorithm appears to be highly nondeterministic, with several choices of actions (e.g., "do one of (a), (b), and (c)"). The paper does not discuss how this nondeterminism is resolved. The main correctness result is given by Theorem 8, which encompasses soundenss, completeness, minimality of change (which the authors call admissibility), and complexity. The proof of soundness and completeness is five lines of informal prose. The proof of complexity does not address the nondeterminism of the repair algorithm. Chatzieleftheriou et. al. [10] presents an approach to repairing abstract structures, using Kripke modal transition systems and 3-valued semantics for CTL. They also aim to minimize the number of changes made to the concrete structure to effect a repair. They provide a set of basic repair operations: add/remove a may/must transition,m change the propositional label of a state, and add/remove a state. Their repair algorithm is recursive CTL-syntax-directed.

12 Conclusions

We presented a method for repairing Kripke structures so that they satisfy a CTL formula η, by deleting transitions that "cause" violation of η. Our method is sound, and is complete relative to our transition deletion strategy. We address the NP-completeness of our model repair problem by translating it (in polynomial time) into a propositional formula, such that a satisfying assignment determines a solution to model repair. Thus, we can bring SAT solvers to bear, which leads us to believe that our method will apply to nontrivial structures, despite the NP-completeness. Unlike other methods, ours both fixes all counterexamples at once, and is complete for temporal properties, specifically full CTL. We extended our method in various directions, to use abstraction, to repair hierarchical and concurrent Kripke structures, to allow addition of states and transitions, and to allow the modification of the propositional labeling of states. We also provided experimental results from our implementation.

Future work includes application of our implementation to larger examples and case studies. Our implementation is useful in model construction, where it provides a check that the constructed structure contains a model.

References

[1] Rajeev Alur and Mihalis Yannakakis. Model checking of hierarchical state machines. *ACM Trans. Program. Lang. Syst.*, 23(3):273–303, 2001.

[2] P. C. Attie. Synthesis of large concurrent programs via pairwise composition. In *CONCUR'99: 10th International Conference on Concurrency Theory*, number 1664 in LNCS. Springer-Verlag, Aug. 1999.

[3] P. C. Attie and E. A. Emerson. Synthesis of concurrent systems with many similar processes. *ACM Trans. Program. Lang. Syst.*, 20(1):51–115, Jan. 1998.

[4] P.C. Attie and E.A. Emerson. Synthesis of concurrent systems for an atomic read / atomic write model of computation (extended abstract). In *PODC*, 1996.

[5] P.C. Attie and E.A. Emerson. Synthesis of concurrent programs for an atomic read/write model of computation. *TOPLAS*, 23(2):187–242, 2001.

[6] Ananda Basu, Saddek Bensalem, Marius Bozga, Jacques Combaz, Mohamad Jaber, Thanh-Hung Nguyen, and Joseph Sifakis. Rigorous component-based system design using the BIP framework. *Software, IEEE*, 28(3):41–48, 2011.

[7] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In *TACAS'99, LNCS number 1579*, 1999.

[8] M.C. Browne, E. M. Clarke, and O. Grumberg. Characterizing finite kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59:115–131, 1988.

[9] F. Buccafurri, T. Eiter, G. Gottlob, and N. Leone. Enhancing model checking in verification by AI techniques. *Artif. Intell.*, 1999.

[10] George Chatzieleftheriou, Borzoo Bonakdarpour, ScottA. Smolka, and Panagiotis Katsaros. Abstract model repair. In AlwynE. Goodloe and Suzette Person, editors, *NASA Formal Methods*, volume 7226 of *Lecture Notes in Computer Science*, pages 341–355. Springer Berlin Heidelberg, 2012.

[11] E. M. Clarke, E. A. Emerson, and P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *TOPLAS*, 1986.

[12] E. M. Clarke, O. Grumberg, K. L. McMillan, and X. Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In *Design Automation Conference*. ACM Press, 1995.

[13] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C North, and Gordon Woodhull. Graphvizopen source graph drawing tools. In *Graph Drawing*, pages 483–484. Springer, 2002.

[14] E. A. Emerson. Temporal and modal logic. *Handbook of Theoretical Computer Science*, pages 997–1072, 1990.

[15] E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.

[16] O Grumberg and D.E. Long. Model checking and modular verification. *TOPLAS*, 16(3):843–871, 1994.

[17] David Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8(3):231 – 274, 1987.

[18] R. Hojati, R. K. Brayton, and R. P. Kurshan. Bdd-based debugging of design using language containment and fair ctl. In *CAV '93*, 1993. Springer LNCS no. 697.

[19] D. Jackson. Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology*, 11(2):256–290, 2002.

[20] B. Jobstmann, A. Griesmayer, and R. Bloem. Program repair as a game. In *CAV*, pages 226–238, 2005.

[21] Daniel Le Berre, Anne Parrain, et al. The sat4j library, release 2.2, system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.

[22] N.A. Lynch and M.R. Tuttle. An introduction to input/output automata. Technical Report CWI-Quarterly, 2(3):219–246, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands, Sept. 1989.

[23] S Shoham and O Grumberg. A game-based framework for ctl counterexamples and 3-valued abstraction-refinement. In *CAV*, pages 275–287, 2003.

[24] Joseph Sifakis. Rigorous System Design. *Foundations and Trends in Electronic Design Automation*, 6(4):293–362, 2012.

[25] S. Staber, B. Jobstmann, and R. Bloem. Diagnosis is repair. In *Intl. Workshop on Principles of Diagnosis*, June 2005.

[26] S. Staber, B. Jobstmann, and R. Bloem. Finding and fixing faults. In *CHARME '05*, 2005. Springer LNCS no. 3725.

[27] C. Stirling and D. Walker. Local model checking in the modal mu-calculus. *Theor. Comput. Sci.*, 89(1), 1991.

[28] Yan Zhang and Yulin Ding. Ctl model update for system modifications. *J. Artif. Int. Res.*, 31(1):113–155, Jan. 2008.

$\underline{\mathsf{propagate}(s,\xi)}$

if $\xi \in old(s)$ **then** $\triangleright\xi$ has already been processed
  $new(s) := new(s) - \xi$; **return**
$\triangleright$already checked for larger index
if $\xi = \mathsf{A}[\varphi\mathsf{V}\psi]^m$ and $\mathsf{A}[\varphi\mathsf{V}\psi]^{m'} \in old(s)$ for some $m' \geq m$ **then**
  $new(s) := new(s) - \xi$; **return**
$\triangleright$already checked for larger index
if $\xi = \mathsf{E}[\varphi\mathsf{V}\psi]^m$ and $\mathsf{E}[\varphi\mathsf{V}\psi]^{m'} \in old(s)$ for some $m' \geq m$ **then**
  $new(s) := new(s) - \xi$; **return**

case ξ: $\triangleright\xi$ has not been processed
  $\xi = \neg\varphi$:
    $new(s) := new(s) \cup \{\varphi\}$; $\mathsf{conjoin}(\text{“}X_{s,\neg\varphi} \equiv \neg X_{s,\varphi}\text{”})$;
  $\xi = \varphi \vee \psi$:
    $new(s) := new(s) \cup \{\varphi, \psi\}$; $\mathsf{conjoin}(\text{“}X_{s,\varphi\vee\psi} \equiv X_{s,\varphi} \vee X_{s,\psi}\text{”})$;
  $\xi = \varphi \wedge \psi$:
    $new(s) := new(s) \cup \{\varphi, \psi\}$; $\mathsf{conjoin}(\text{“}X_{s,\varphi\wedge\psi} \equiv X_{s,\varphi} \wedge X_{s,\psi}\text{”})$;
  $\xi = \mathsf{AX}\varphi$:
    forall $t \in R[s] : new(t) := new(t) \cup \{\varphi\}$ **endfor** ;
    $\mathsf{conjoin}(\text{“}\bigwedge_{t\in R[s]}(E_{s,t} \Rightarrow X_{t,\varphi})\text{”})$
  $\xi = \mathsf{EX}\varphi$:
    forall $t \in R[s] : new(t) := new(t) \cup \{\varphi\}$ **endfor** ;
    $\mathsf{conjoin}(\text{“}\bigvee_{t\in R[s]}(E_{s,t} \Rightarrow X_{t,\varphi})\text{”})$
  $\xi = \mathsf{A}[\varphi\mathsf{V}\psi]$:
    $new(s) := new(s) \cup \{\mathsf{A}[\varphi\mathsf{V}\psi]^n\}$;
    $\mathsf{conjoin}(\text{“}X_{s,\mathsf{A}[\varphi\mathsf{V}\psi]} \equiv X^n_{s,\mathsf{A}[\varphi\mathsf{V}\psi]}\text{”})$;
  $\xi = \mathsf{A}[\varphi\mathsf{V}\psi]^m$, $m \in \{1, \ldots, n\}$:
    $new(s) := new(s) \cup \{\varphi, \psi\}$;
    forall $t \in R[s] : new(t) := new(t) \cup \{\mathsf{A}[\varphi\mathsf{V}\psi]^{m-1}\}$;
    $\mathsf{conjoin}(\text{“}X^m_{s,\mathsf{A}[\varphi\mathsf{V}\psi]} \equiv X_{s,\psi} \wedge (X_{s,\varphi} \vee \bigwedge_{t\in R[s]}(E_{s,t} \Rightarrow X^{m-1}_{t,\mathsf{A}[\varphi\mathsf{V}\psi]}))\text{”})$
  $\xi = \mathsf{A}[\varphi\mathsf{V}\psi]^0$:
    $new(s) := new(s) \cup \{\psi\}$;
    $\mathsf{conjoin}(\text{“}X^0_{s,\mathsf{A}[\varphi\mathsf{V}\psi]} \equiv X_{s,\psi}\text{”})$
  $\xi = \mathsf{E}[\varphi\mathsf{V}\psi]$:
    $new(s) := new(s) \cup \{\mathsf{E}[\varphi\mathsf{V}\psi]^n\}$;
    $\mathsf{conjoin}(\text{“}X_{s,\mathsf{E}[\varphi\mathsf{V}\psi]} \equiv X^n_{s,\mathsf{E}[\varphi\mathsf{V}\psi]}\text{”})$;
  $\xi = \mathsf{E}[\varphi\mathsf{V}\psi]^m$, $m \in \{1, \ldots, n\}$:
    $new(s) := new(s) \cup \{\varphi, \psi\}$;
    forall $t \in R[s] : new(t) := new(t) \cup \{\mathsf{A}[\varphi\mathsf{V}\psi]^{m-1}\}$;
    $\mathsf{conjoin}(\text{“}X^m_{s,\mathsf{E}[\varphi\mathsf{V}\psi]} \equiv X_{s,\psi} \wedge (X_{s,\varphi} \vee \bigvee_{t\in R[s]}(E_{s,t} \Rightarrow X^{m-1}_{t,\mathsf{E}[\varphi\mathsf{V}\psi]}))\text{”})$
  $\xi = \mathsf{E}[\varphi\mathsf{V}\psi]^0$:
    $new(s) := new(s) \cup \{\psi\}$;
    $\mathsf{conjoin}(\text{“}X^0_{s,\mathsf{E}[\varphi\mathsf{V}\psi]} \equiv X_{s,\psi}\text{”})$
endcase ;
$new(s) := new(s) - \{\xi\}$; $\triangleright$remove ξ from new since it has been processed
$old(s) := old(s) \cup \{\xi\}$; $\triangleright$record that ξ has been processed

Figure 25: Formula propagation.

InitializeRepairFormula(M, η)

$repair(M, \eta)$:= true;
conjoin("$\bigvee_{s_0 \in S_0} X_{s_0}$"); ▷Clause 1
forall $s \in S_0$: conjoin("$X_{s_0} \Rightarrow X_{s_0,\eta}$"); ▷Clause 2
forall $s \in S$: conjoin("$X_s \equiv \bigvee_{t \in R[s]}(E_{s,t} \wedge X_t)$") ▷Clause 3
forall $(s,t) \in R$: conjoin("$E_{s,t} \Rightarrow (X_s \wedge X_t)$") ▷Clause 4
forall $s \in S, p \in AP \cap L(s)$: conjoin("$X_{s,p}$"); ▷Clause 5
forall $s \in S, p \in AP - L(s)$: conjoin("$\neg X_{s,p}$"); ▷Clause 5

Figure 26: Initializing $repair(M, \eta)$

conjoin(f)

g := "";
case f:
f does not contain either of $\bigwedge_{t \in R[s]}, \bigvee_{t \in R[s]}$
$g := f$;
$f =$ "$\bigwedge_{t \in R[s]}(E_{s,t} \Rightarrow X_{t,\varphi})$"
forall $t \in R[s] : g := g \frown$ "$\wedge (E_{s,t} \Rightarrow X_{t,\varphi})$"
$f =$ "$\bigvee_{t \in R[s]}(E_{s,t} \Rightarrow X_{t,\varphi})$"
forall $t \in R[s] : g := g \frown$ "$\vee (E_{s,t} \Rightarrow X_{t,\varphi})$"
$f =$ "$X^m_{s,\mathsf{A}[\varphi \mathsf{V} \psi]} \equiv X_{s,\psi} \wedge (X_{s,\varphi} \vee \bigwedge_{t \in R[s]}(E_{s,t} \Rightarrow X^{m-1}_{t,\mathsf{A}[\varphi \mathsf{V} \psi]}))$"
forall $t \in R[s] : g := g \frown$ "$\wedge (E_{s,t} \Rightarrow X^{m-1}_{t,\mathsf{A}[\varphi \mathsf{V} \psi]})$";
$g :=$ "$X^m_{s,\mathsf{A}[\varphi \mathsf{V} \psi]} \equiv X_{s,\psi} \wedge (X_{s,\varphi} \vee$" $\frown g \frown$ "$)$"
$f =$ "$X^m_{s,\mathsf{E}[\varphi \mathsf{V} \psi]} \equiv X_{s,\psi} \wedge (X_{s,\varphi} \vee \bigvee_{t \in R[s]}(E_{s,t} \Rightarrow X^{m-1}_{t,\mathsf{E}[\varphi \mathsf{V} \psi]}))$"
forall $t \in R[s] : g := g \frown$ "$\vee (E_{s,t} \Rightarrow X^{m-1}_{t,\mathsf{E}[\varphi \mathsf{V} \psi]})$";
$g :=$ "$X^m_{s,\mathsf{E}[\varphi \mathsf{V} \psi]} \equiv X_{s,\psi} \wedge (X_{s,\varphi} \vee$" $\frown g \frown$ "$)$"
endcase ;
$repair(M, \eta) := repair(M, \eta) \frown g$

Figure 27: Adding a conjunct to $repair(M, \eta)$