Model Repair via SAT Solving

Paul Attie, Kinan Dak Al Bab, Mohamad Sakr, Jad Saklawi, Ali Cherri
Department of Computer Science
American University of Beirut
paul.attie@aub.edu.lb, kmd14@aub.edu.lb
mis14@mail.aub.edu, jad.saklawi@gmail.com, ahc14@mail.aub.edu

June 22, 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 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) repair concurrent Kripke structures and concurrent programs: we use the pairwise method of Attie and Emerson to represent and repair the behavior of a concurrent program, as a set of "concurrent Kripke structures", with only a quadratic increase in the size of the repair formula. (3) repair hierarchical Kripke structures: we use 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 [16] 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* [18], 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 [21] 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 9 shows how to repair hierarchical Kripke structures. Section 8 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 [15, 16] 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 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 $\mathcal{V}(x_j) = \mathit{ff}$, 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 $\mathcal{V}(x_j) = \mathit{ff}$. Therefore $\mathcal{V}(a_i) = tt$. Hence $\mathcal{V}(a_i) \vee \mathcal{V}(b_i) \vee \mathcal{V}(c_i)$. □

4 The 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, η) such that *repair*(M, η) is satisfiable iff (M, η) has a solution. *repair*(M, η) is defined over the following propositions:

1. $E_{s,t} : (s, t) \in R$
2. $X_s : s \in S$
3. $X_{s,\psi} : s \in S, \psi \in \mathit{sub}(\eta)$
4. $X^n_{s,\psi} : s \in S, 0 \leq n \leq |S|$, and $\psi \in \mathit{sub}(\eta)$ 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, η). 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, η), 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, η))**.** *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, η) *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, η) *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, $repair(M, \eta)$ 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 $model(M, \mathcal{V})$ 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 $repair(M, \eta)$ 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
▷case condition
$\mathcal{V}(X_{s,\neg\varphi}) = tt$ iff
▷Clause 6 (propositional consistency) of Def. 8
$\mathcal{V}(X_{s,\varphi}) = ff$ iff
▷induction hypothesis
$not(M', s \models \varphi)$ iff
▷CTL semantics
$M', s \models \neg\varphi$ iff
▷case condition
$M', s \models \xi$

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

Case $\xi = \varphi \wedge \psi$:
$\mathcal{V}(X_{s,\xi}) = tt$ iff
▷case condition
$\mathcal{V}(X_{s,\varphi\wedge\psi}) = tt$ iff
▷Clause 6 (propositional consistency) of Def. 8
$\mathcal{V}(X_{s,\varphi}) = tt$ and $\mathcal{V}(X_{s,\psi}) = tt$ iff
▷induction hypothesis
$(M', s \models \varphi)$ and $(M', s \models \psi)$ iff ▷CTL semantics
$M', s \models \varphi \wedge \psi$ iff ▷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 $M'', S''_0 \models \eta$. □

```
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.

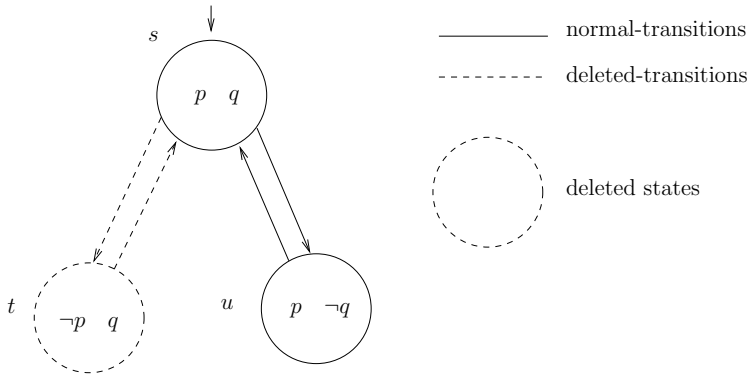

Figure 2: Simple Kripke structure

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 $X_{s,\mathsf{AG}p}$.

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

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

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

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

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

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

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

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 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, $\mathbf{Ni}, \mathbf{Ti}, \mathbf{Ci}$. In the neutral state, **Ni** is true and $\mathbf{Ti}, \mathbf{Ci}$ are false. In the trying state, **Ti** is true and $\mathbf{Ni}, \mathbf{Ci}$ are false. In the critical state, **Ci** is true and $\mathbf{Ni}, \mathbf{Ti}$ are false. The CTL specification η is $\mathsf{AG}(\neg(\mathbf{C1} \wedge \mathbf{C2}))$, 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. $\mathsf{AG}(\neg(\mathbf{C1} \wedge \mathbf{C2}))$. 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. In Fig. 4 the violating state **C1 C2** is now unreachable, and so $\mathsf{AG}(\neg(\mathbf{C1} \wedge \mathbf{C2}))$ is now satisfied. This repair is however, overly restrictive, as follows. If P_1 remains forever in its neutral state, it is not possible for P_2 to cycle indefinitely from neutral to trying to critical, and vice-versa. This should be possible in a good solution to mutual exclusion. Moreover, whenever P_2 enters the critical section, it must wait for P_1 to subsequently enter before it can enter again. We show in the sequel how to improve the quality of the repairs.

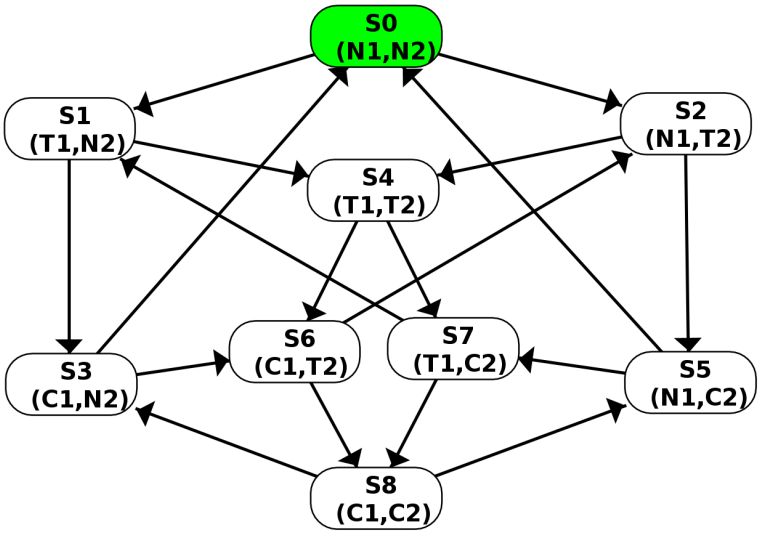

Figure 3: Mutex: initial structure

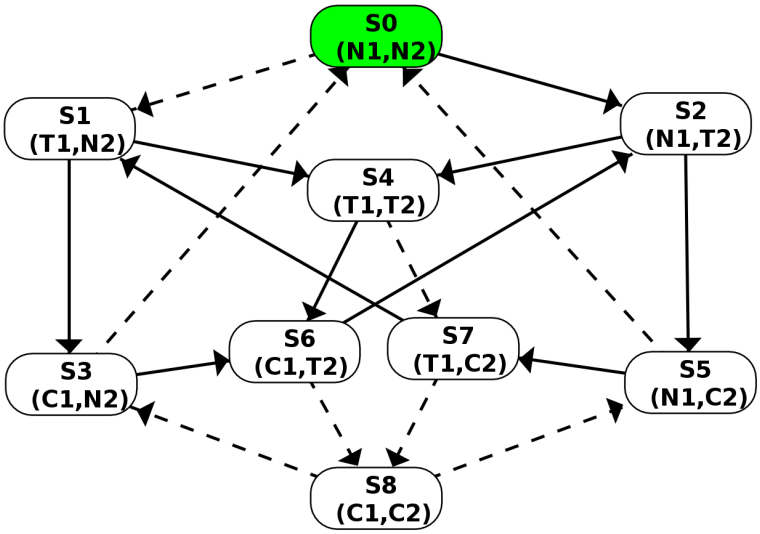

Figure 4: Mutex: repaired structure

5.2 Mutual exclusion: interactive design using semantic feedback

We add liveness to the mutex example by repairing w.r.t. $\mathsf{AG}(\neg(\mathbf{C1} \wedge \mathbf{C2})) \wedge \mathsf{AG}(\mathbf{T1} \Rightarrow \mathsf{AF}\mathbf{C1}) \wedge \mathsf{AG}(\mathbf{T2} \Rightarrow \mathsf{AF}\mathbf{C2})$. We start with the structure shown in Fig. 3. This yields the structure of Fig. 7, in which P_1 cycles repreatedly through neutral, trying, andcritical, while P_2 has no transitions at all! Obviously, this repair is much too restrictive. We notice that some transitions where a process requests the critical section, by moving from neutral to trying (e.g., **N1 N2** to **N1 T2**) 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 $\mathsf{AG}(\mathbf{T2} \Rightarrow \mathsf{AF}\mathbf{C2})$, we repair w.r.t. $\mathsf{AG}(\neg(\mathbf{C1} \wedge \mathbf{C2})) \wedge \mathsf{AG}(\mathbf{T1} \Rightarrow \mathsf{AF}\mathbf{C1})$ and obtain a repair, given in Fig. 8. We observe that $\mathsf{AG}(\mathbf{T1} \Rightarrow \mathsf{AF}\mathbf{C1})$ was previously violated by the cycle $\mathbf{T1\,N2} \rightarrow \mathbf{T1\,T2} \rightarrow \mathbf{T1\,C2} \rightarrow \mathbf{T1\,N2}$, which is now broken. By symmetric reasoning, $\mathsf{AG}(\mathbf{T2} \Rightarrow \mathsf{AF}\mathbf{C2})$ was previously violated by the cycle $\mathbf{N1,T2} \rightarrow \mathbf{T1,T2} \rightarrow \mathbf{C1,T2} \rightarrow \mathbf{N1,T2}$. Inspection shows that we cannot break both cycles at once: $\mathbf{T1,N2} \rightarrow \mathbf{T1,T2}$ and $\mathbf{N1,T2} \rightarrow \mathbf{T1,T2}$ are now non-deletable. Removing either of $\mathbf{T1,C2} \rightarrow \mathbf{T1,N2}$ or $\mathbf{C1,T2} \rightarrow \mathbf{N1,T2}$, leaves $\mathbf{T1,C2}$, $\mathbf{C1,T2}$ respectively without an outgoing transition, so M' is no longer total. Hence we have to remove both $\mathbf{T1,T2} \rightarrow \mathbf{T1,C2}$ and $\mathbf{T1,T2} \rightarrow \mathbf{C1,T2}$, which leaves $\mathbf{T1,T2}$ without an outgoing transition. The problem is that in $\mathbf{T1,T2}$ 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* $\mathbf{T1,T2}$ 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, 29]. 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.

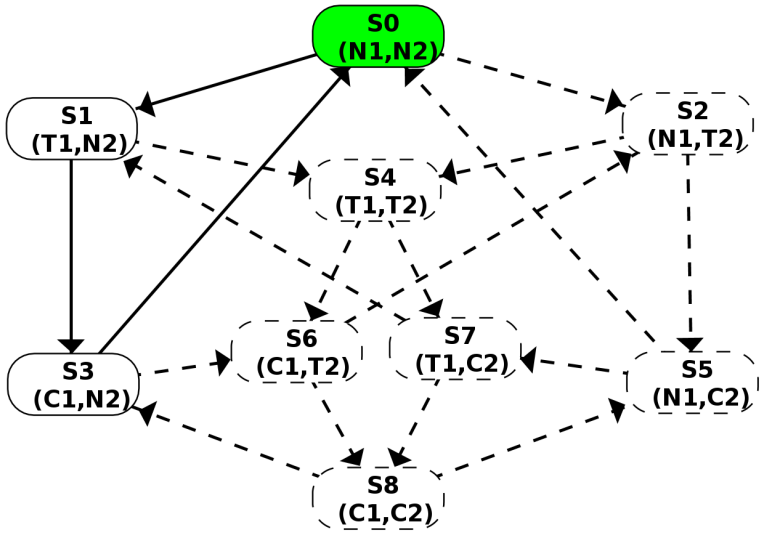

Figure 5: Mutex: overconstrained repair

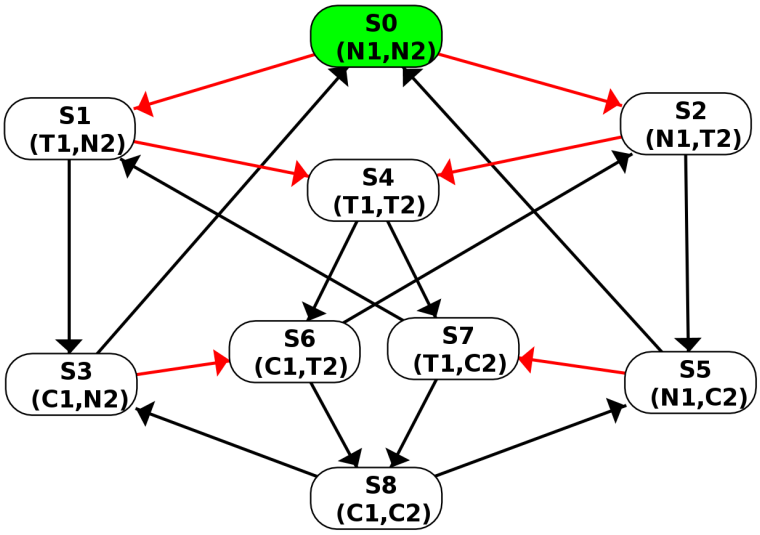

Figure 6: Mutex: unrepairable due to retained request transitions

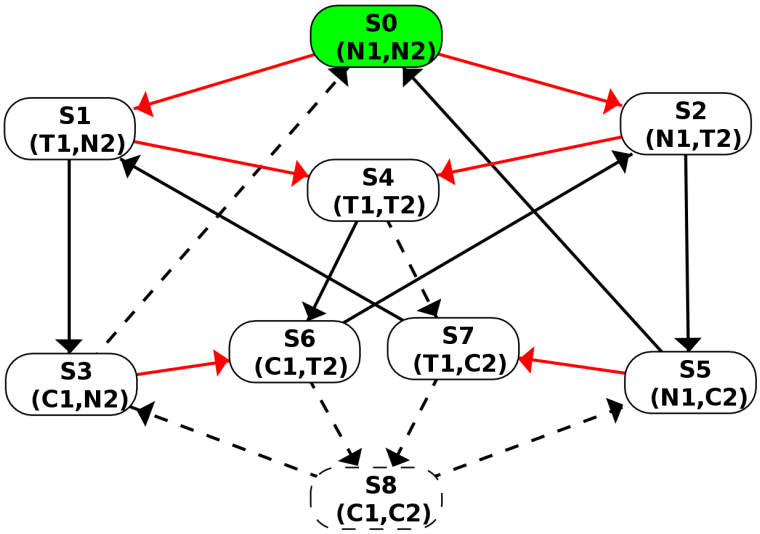

Figure 7: Mutex: only P_1 is live

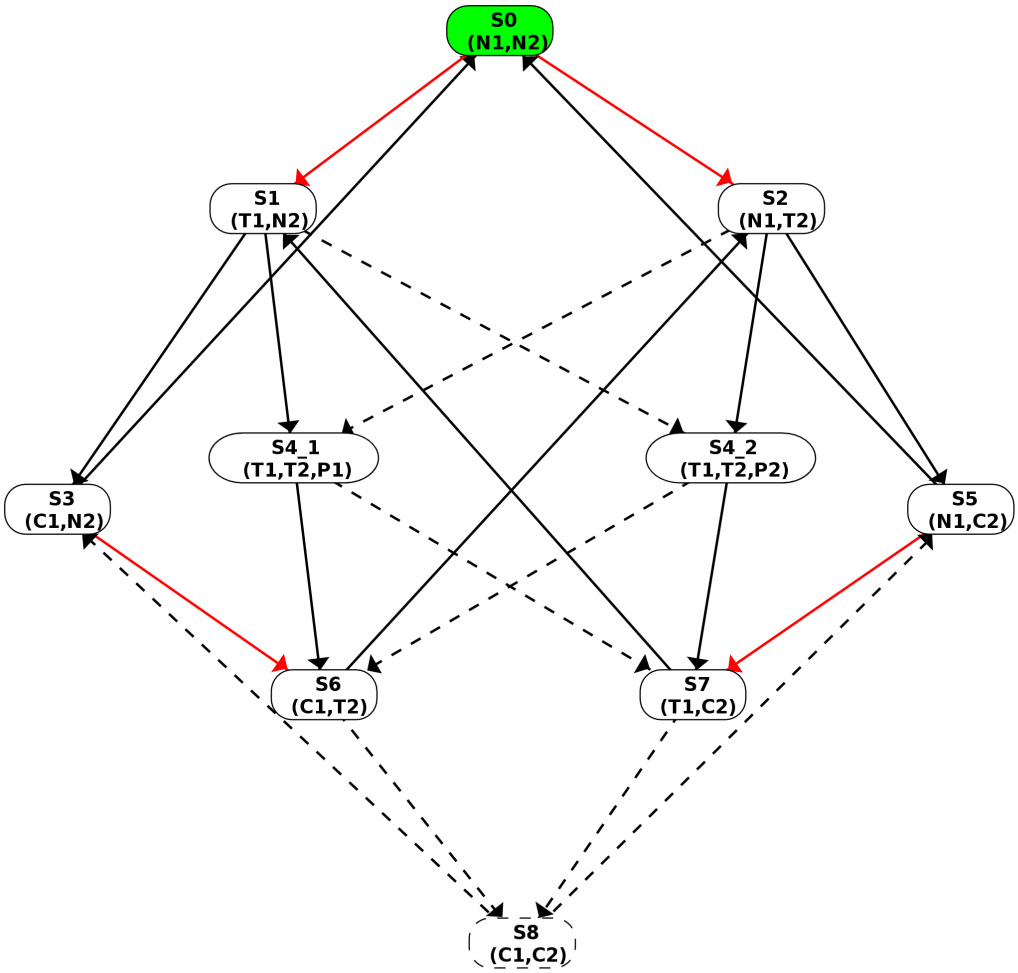

Figure 8: Mutex: live solution

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, **SAi**, **EAi**, **SBi**, **EBi**, 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 **SAi**. The CTL specification is the conjunction of the following:

1. Initially both processes are at the start of phase A: $\mathbf{SA1} \wedge \mathbf{SA2}$
2. P_1 and P_2 are never simultaneously at the start of different phases:
   $\mathsf{AG}(\neg(\mathbf{SA1} \wedge \mathbf{SB2})) \wedge \mathsf{AG}(\neg(\mathbf{SA2} \wedge \mathbf{SB1}))$
3. P_1 and P_2 are never simultaneously at the end of different phases:
   $\mathsf{AG}(\neg(\mathbf{EA1} \wedge \mathbf{EB2})) \wedge \mathsf{AG}(\neg(\mathbf{EA2} \wedge \mathbf{EB1}))$

(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. 9 violates the synchronization rules (2) and (3). Our implementation produced the repair shown in Fig. 10.

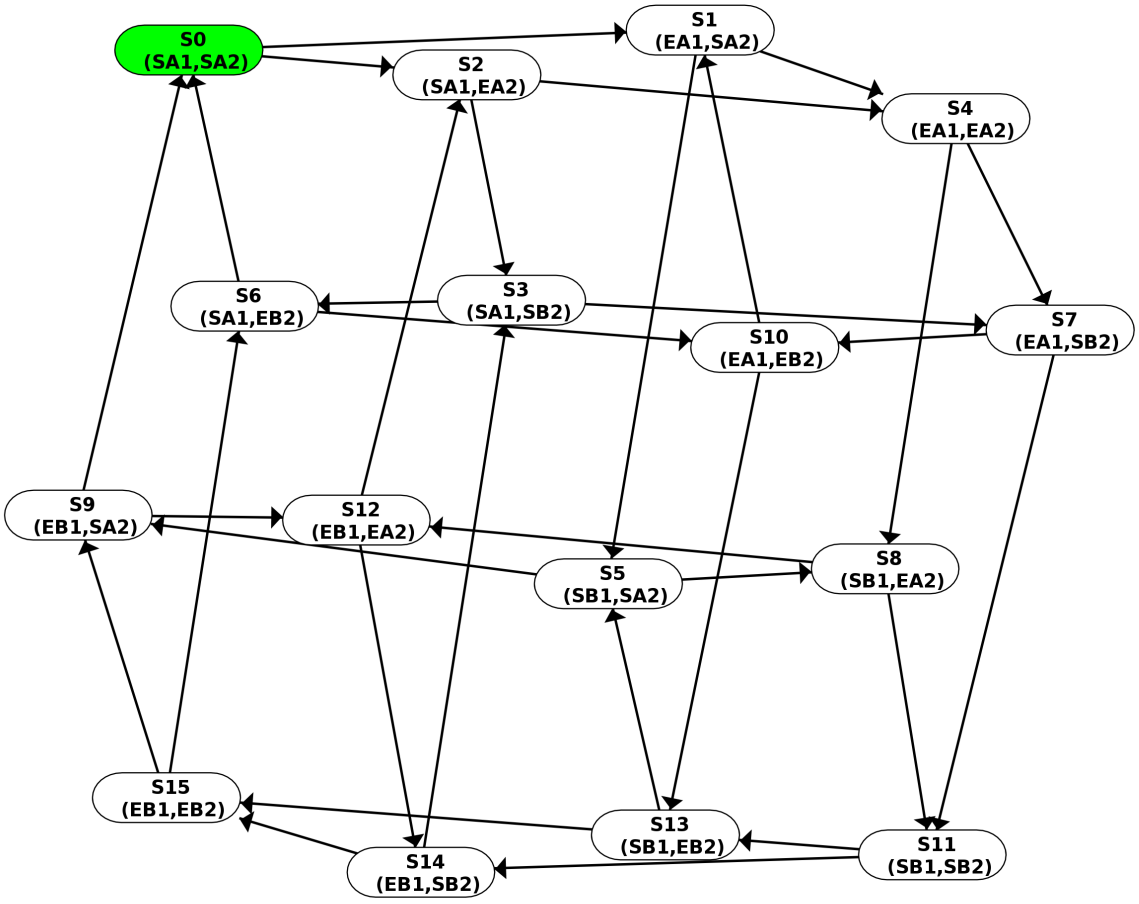

Figure 9: Barrier syncrhonization: initial structure

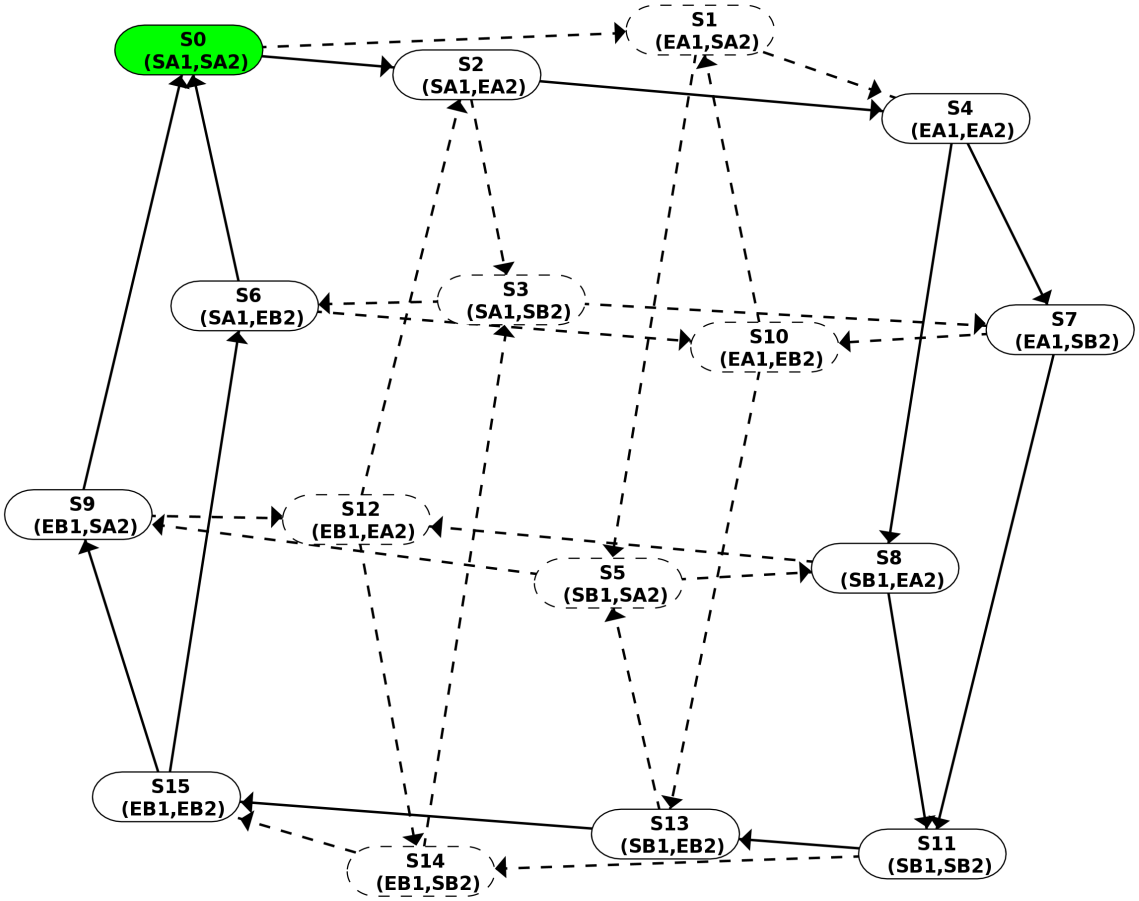

Figure 10: Barrier syncrhonization: repaired structure

6 Extensions of the Repair Algorithm

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

6.1 Addition of States and Transitions

The subtractive repair algorithm performs repair by deleting states and transitions. In some cases, it may be useful to add states and transitions, e.g., as shown in the mutual exclusion example in Sect. 5.2. Currently we can do this manually in Eshmun, leading to an experimental/interactive method for adding states and transitions. A useful boundary case, illustrated in the sequel, is to add all possible transitions and then repair. When the structure we are repairing is a *Kripke multiprocess structure* (see Sect. 8), the number of transitions tends to be linear in the number of states, rather than quadratic, and so this is not as burdensome as it seems.

6.2 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.

6.3 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.

6.4 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 *concurrent* Kripke structures, as used for example in Statecharts [19]
3. repair of *hierarchical* Kripke structures [1]

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 $\mathit{repair}(M, \eta)$.* $\mathit{repair}(M, \eta)$ 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* $\overline{L}([s]) = L(s) \cap AP_\eta$

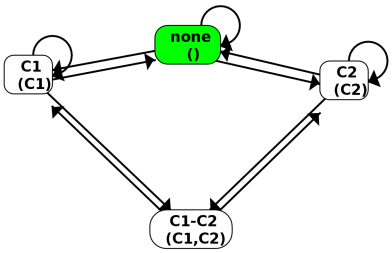

Figure 11: Mutex: abstraction by label (i.e., by atomic propositions)

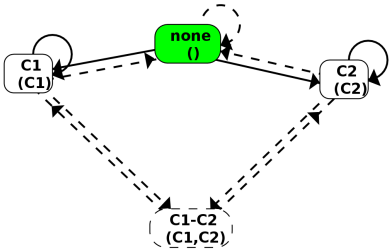

Figure 12: Mutex: bad repair of abstract-by-label structure

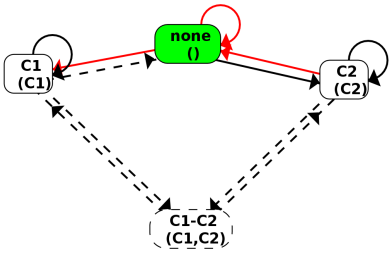

Figure 13: Mutex: good repair of abstract-by-label model

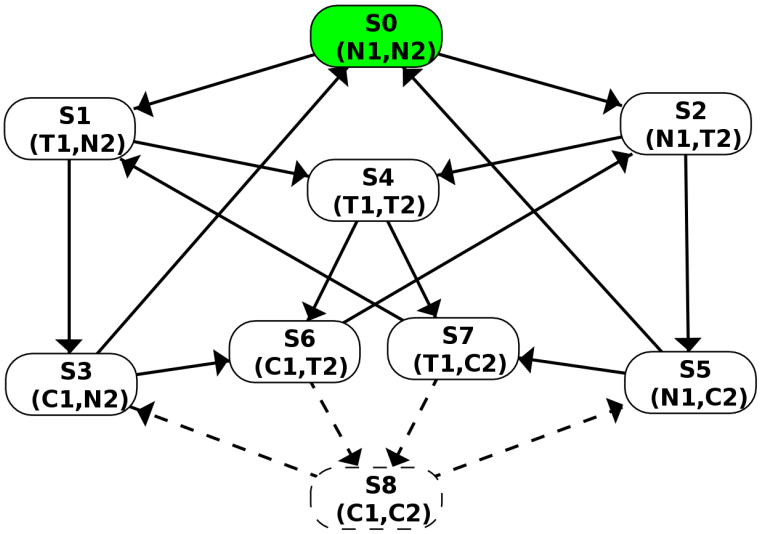

Figure 14: Mutex: concretization of repaired abstract-by-label structure

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: **none_1** $= \{\mathbf{N1N2}, \mathbf{N1T2}, \mathbf{T1N2}, \mathbf{T1T2}\}$, **C1** $= \{\mathbf{C1N2}, \mathbf{C1T2}\}$, **C2** $= \{\mathbf{N1C2}, \mathbf{T1C2}\}$, **C1_C2** $=$ **C1C2**. Fig. 11 shows the resulting abstract structure $\overline{M}$, which has four states, corresponding to these equivalence classes. Figure 12 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 **none_1** $\rightarrow$ **C1**, **none_1** $\rightarrow$ **C2**, since the critical sections should always be reachable. Figure 13 shows the resulting repair, and Fig. 14 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* $[s]$.

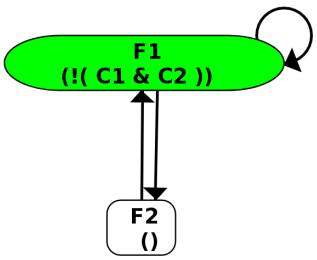

Figure 15: Mutex: abstraction by subformulae

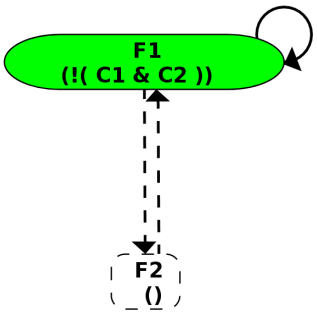

Figure 16: Mutex: repair of abstract-by-subformulae structure

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 $\mathbf{none\_1} = \{\mathbf{N1N2}, \mathbf{N1T2}, \mathbf{T1N2}, \mathbf{T1T2}, \mathbf{C1N2}, \mathbf{C1T2}, \mathbf{N1C2}, \mathbf{T1C2}\}$ and $\mathbf{C1\_C2} = \mathbf{C1C2}$. Fig. 15 shows the resulting abstract structure $\overline{M}$, which has two states, corresponding to these equivalence classes.

Figure 16 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. 14, 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

Fig. 17 shows the result of abstraction by subformulae applied to the barrier synchronization Kripke structure in Fig. 9. Fig. 18 show a repair of this structure, and Fig. 19 shows the concretization of this repair to the structure of Fig. 9. Again this produced a minimal repair, in which all frame properties are preserved.

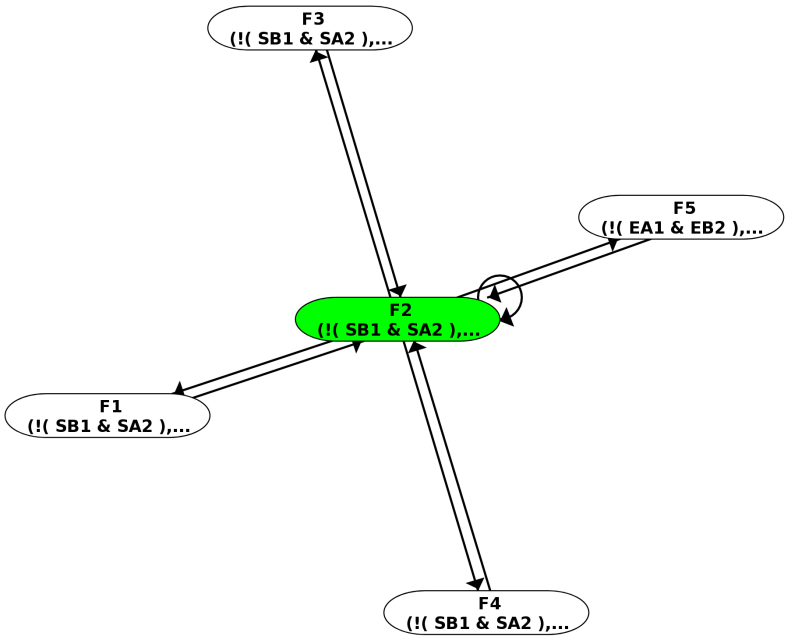

Figure 17: Barrier synchronization: abstraction by subformulae

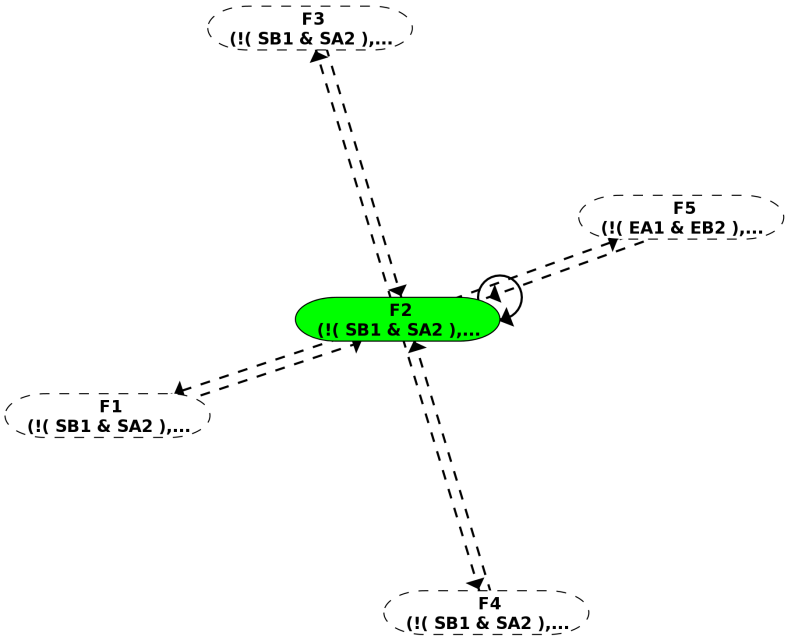

Figure 18: Barrier synchronization: repair of abstract-by-subformulae structure

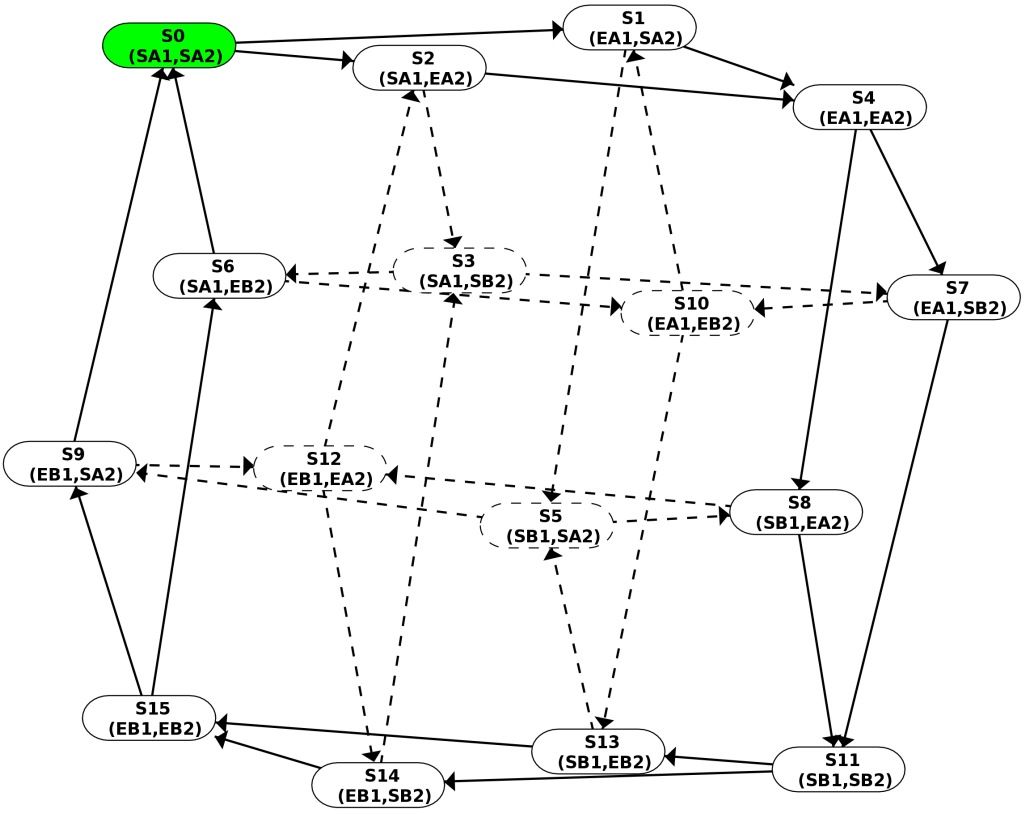

Figure 19: Barrier synchronization: concretization of repaired abstract-by-subformulae structure

8 Repair of Concurrent Kripke Structures

We consider finite-state shared-memory concurrent programs $P = (St_P, P_1 \| \cdots \| P_K)$ consisting of K sequential processes $P_1, \ldots, P_K$ running in parallel, together with a set St_P of starting global states. For each P_i, there is a finite set AP_i of *atomic propositions* that are *local to* P_i: only P_i can change the value of atomic propositions in AP_i. Other processes can read, but not change, these values. Local atomic propositions are not shared: $AP_i \cap AP_j = \emptyset$ when $i \neq j$. We also admit a set $\mathcal{SH} = \{x_1, \ldots, x_m\}$ of shared variables. These can be read/written by all checks that processes, and have values from finite domains[1]. We define the set of all atomic propositions $AP = AP_1 \cup \cdots \cup AP_K$.

Each P_i is a *synchronization skeleton* [16], that is, a directed multigraph where each node is a *local state* of P_i, which is labeled by a unique name s_i, and where each arc is labeled with a *guarded command* [13] $B_i \to A_i$ consisting of a guard B_i and corresponding action A_i. We write such an arc as the tuple $(s_i, B_i \to A_i, s_i')$, where s_i is the source node and s_i' is the target node. Each node must have at least one outgoing arc, i.e., a synchronization skeleton contains no "dead ends."

The read/write restrictions on atomic propositions are reflected in the syntax of processes: for an arc $(s_i, B_i \to A_i, s_i')$ of P_i, the guard B_i is a boolean formula over $AP - AP_i$, and the action

[1] In Eshmun all shared variables are boolean, i.e., atomic propositions that are not local to any process.

A_i is any piece of terminating pseudocode that updates only the shared variables $\mathcal{SH}$.

Let S_i denote the set of local states of P_i. There is a mapping $V_i : S_i \to (AP_i \to \{\mathsf{true}, \mathsf{false}\})$ from local states of P_i to boolean valuations over AP_i: for $p_i \in AP_i$, $V_i(s_i)(p_i)$ is the value of atomic proposition p_i in s_i. Hence, as P_i executes transitions and changes its local state, the atomic propositions in AP_i are updated, since $V_i(s_i) \neq V_i(s_i')$ in general.

A *global state* is a tuple $(s_1, \ldots, s_K, v_1, \ldots, v_m)$ where s_i is the current local state of P_i and $v_1, \ldots, v_m$ is a list giving the current values of the shared variables in $\mathcal{SH}$.

Let $s = (s_1, \ldots, s_i, \ldots, s_K, v_1, \ldots, v_m)$ be the current global state, and let P_i contain an arc from node s_i to node s_i' labeled with $B_i \to A_i$. If B_i holds in s, then a possible next state is $s' = (s_1, \ldots, s_i', \ldots, s_K, v_1', \ldots, v_m')$ where $v_1', \ldots, v_m'$ are the new values, respectively, for the shared variables $x_1, \ldots, x_m$ resulting from the execution of action A_i. The set of all (and only) such triples (s, i, s') constitutes the *next-state relation* of program P. As stated above, local atomic propositions AP_i are implicitly updated, since P_i changed its local state from s_i to s_i'.

The appropriate semantic model for a concurrent program is a *multiprocess Kripke structure*, which is a Kripke structure that has its set AP of atomic propositions partitioned into $AP_1 \cup \cdots \cup AP_K$, and 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. Shared variables may also be present. The structure of Fig. 3 is a multiprocess Kripke structure.

The semantics of a concurrent program $P = (St_P, P_1 \,\|\, \cdots \,\|\, P_K)$ is given its global state transition digram (GSTD): the smallest multiprocess Kripke structure M such that (1) the start states of M are St_P, and (2) M is closed under the next state relation of P. Effectively, M is obtained by "simulating" all possible executions of P from its start states St_P. A program satisfies a CTL formula η iff its GSTD does.

Conversely, given a multiprocess Kripke structure M, we can extract a concurrent program by projecting onto the individual process indices [16]. If M contains a transition from $s = (s_1, \ldots, s_i, \ldots, s_K, v_1, \ldots, v_m)$ to $s' = (s_1, \ldots, s_i', \ldots, s_K, v_1', \ldots, v_m')$, then we can project this onto P_i as the arc $(s_i, B_i \to A_i, s_i')$, where B_i checks that the current global state is $(s_1, \ldots, s_i, \ldots, s_K, v_1, \ldots, v_m)$, and A_i is the multiple assignment $x_1, \ldots, x_m := v_1', \ldots, v_m'$, i.e., it assigns v_ℓ' to x_ℓ, $\ell = 1, \ldots, m$. For example, the concurrent program given in Fig. 20 is extracted from the multiprocess Kripke of Fig. 14. Each local state is shown labeled with the atomic propositions that it evaluates to true. Take for example the transition in Fig. 14 from $s2$ to $s4$: this is a transition by P_1 from **N1** to **T1** which can be taken only when **T2** holds. It contributes an arc $(\mathbf{N1}, \mathbf{T2} \to \epsilon, \mathbf{T1})$ to P_1, where ϵ denotes the empty action, which changes nothing. Likewise the transition from $s0$ to $s1$ contributes $(\mathbf{N1}, \mathbf{N2} \to \epsilon, \mathbf{T1})$, and the transition from $s5$ to $s7$ contributes $(\mathbf{N1}, \mathbf{C2} \to \epsilon, \mathbf{T1})$. We group all these arcs into a single arc, whose label is $(\mathbf{N2} \to \epsilon) \oplus (\mathbf{T2} \to \epsilon) \oplus (\mathbf{C2} \to \epsilon)$. The $\oplus$ operator is a "disjunction" of guarded commands: $(B_i \to A_i) \oplus (B_i' \to A_i')$ means nondeterministically select one of the two guarded commands whose guard holds, and execute the corresponding action. Using $\oplus$ means we have at most one arc, in each direction, between any pair of local states. To avoid clutter in the figures, we replace $B \to \epsilon$ by just B, i.e., we omit empty actions.

In principle then, we can repair a concurrent program by (1) generating its GSTD M, (2) repairing M w.r.t. η to produce M', and (3) extracting a repaired program from M'. In practice, however, this quickly runs up against the *state explosion problem*: the size of M is exponential in the number of processes K. We avoid state explosion as follows. In [2–4] an approach for the synthesis and verification of concurrent programs based on *pairwise composition*

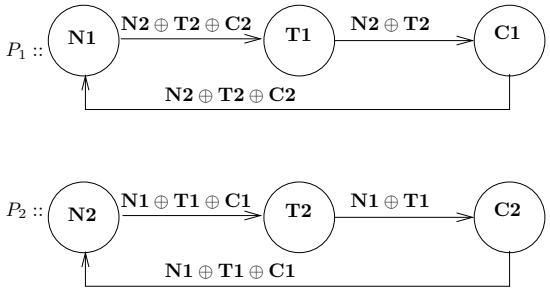

Figure 20: Concurrent program extracted from Fig. 14.

is presented: for each pair of processes P_i, P_j that interact directly, we provide a *pair-structure* $M_{ij} = (S_0^{ij}, S_{ij}, R_{ij}, L_{ij}, AP_{ij})$, which is a multiprocess Kripke structure over P_i and P_j. M_{ij} defines the direct interaction between processes P_i and P_j. Hence, 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 P_i, P_k, defines this interaction. So, M_{ij} and M_{ik} have the atomic propositions AP_i in common. Their shared atomic propositions are disjoint. The pairs of directly interacting processes are given by an *interaction relation* I, a symmetric relation over the set $\{1, \ldots, K\}$ of process indices.

We extract from each pair-structure M_{ij} a corresponding *pair-program* $P_i^j \parallel P_j^i$, which consists of two *pair-processes* P_i^j and P_j^i. We then compose all of the pair-programs to produce the final concurrent program P. Composition is syntactic: the process P_i in P is the result of composing all the pair-processes $P_i^j, P_i^k, \ldots$. For example, 3 process mutual exclusion can be synthesized by having 3 pair-programs $P_1^2 \parallel P_2^1$, $P_1^3 \parallel P_3^1$, $P_2^3 \parallel P_3^2$, which implement mutual exclusion between each respective pair of processes. These are all isomorphic to Fig. 20, modulo index substitution. Then, the 3 process solution $P = P_1 \parallel P_2 \parallel P_3$ is given in Fig. 21, where only P_1 is shown, P_2 and P_3 being isomorphic to P_1 modulo index substitution. P_1 results from composing P_1^2 and P_1^3. In P_1, the arc from **T1** to **C1** is the composition (using the $\otimes$ operator) of "corresponding" arcs in P_1^2 and P_1^3. For example, the arc from **T1** to **C1** in P_1^2, namely $(\mathbf{T1}, (\mathbf{N2} \to \epsilon) \oplus (\mathbf{T2} \to \epsilon), \mathbf{C1})$, and the arc from **T1** to **C1** in P_1^3, namely $(\mathbf{T1}, (\mathbf{N3} \to \epsilon) \oplus (\mathbf{T3} \to \epsilon), \mathbf{C1})$ are corresponding arcs, since they have the same source and target local states, namely **T1** and **C1**. Their composition given the arc $(\mathbf{T1}, ((\mathbf{N2} \to \epsilon) \oplus (\mathbf{T2} \to \epsilon)) \otimes ((\mathbf{N3} \to \epsilon) \oplus (\mathbf{T3} \to \epsilon)), \mathbf{C1})$ The meaning of $(B_i \to A_i) \otimes (B_i' \to A_i')$ is that both B_i and B_i' must hold, and then A_i and A_i' must be executed concurrently. It is a "conjunction" of guarded commands. Hence, the meaning of $(\mathbf{T1}, ((\mathbf{N2} \to \epsilon) \oplus (\mathbf{T2} \to \epsilon)) \otimes ((\mathbf{N3} \to \epsilon) \oplus (\mathbf{T3} \to \epsilon)), \mathbf{C1})$ is that P_1 can enter its critical state **C1** iff P_2 is in either its neutral state **N2** or it trying state **T2**, and also P_3 is in either its neutral state **N3** or its trying state **T3**. The other two arcs of P_1 are similarly constructed. We use $\oplus$ only when the actions of the guarded commands update disjoint sets of variables, so that the result of execution is always well-defined.

Let $I(i) = \{j \mid (i,j) \in I\}$, so that $I(i)$ is the set of proceses that P_i interacts directly with. For each $j \in I(i)$, we create and repair pair structure $M_{i,j}$ w.r.t. a "pair-specification" $\eta_{i,j}$. From $M_{i,j}$ we extract pair-program $P_i^j \parallel P_j^i$. Process P_i in the overall large program is obtained by composing the pair-processes P_i^j for all $j \in I(i)$, as follows, [3, Def. 15, pairwise synthesis]:

P_i contains an arc from s_i to t_i with label $\bigotimes_{j=\in I(i)} \bigoplus_{\ell \in [1:n_j]} B_{i,\ell}^j \to A_{i,\ell}^j$

iff

for $j \in I(i)$: P_i^j contains an arc from s_i to t_i with label $\bigoplus_{\ell \in [1:n_j]} B_{i,\ell}^j \to A_{i,\ell}^j$.

Recall that two arcs in P_i^j, P_i^k correspond iff they have the same source and target nodes. An

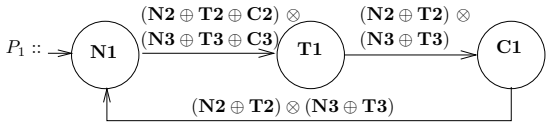

Figure 21: Concurrent program extracted from Fig. **??**

arc in P_i is then a composition of corresponding arcs in all the $P_i^j, j \in I(i)$. For this composition to be possible, it must be that the corresponding arcs all exist. We must therefore check that, for all $j, k \in I(i)$, that if M_{ij} contains some transition by P_i^j in which it changes its local state from s_i to t_i, then M_{ik} also contains some transition by P_i^k in which it changes its local state from s_i to t_i. This ensures that every set of corresponding arcs contains a representative from each pair-program P_i^j for all $j \in I(i)$, and so our definition of pairwise synthesis produces a well-defined result. We call this the *process graph consistency constraint*, since it states, in effect, that P_i^j, P_i^k have the same graph: the result of removing all arc labels. Consider again the three process mutual exclusion example, and suppose that M_{12} contains a transition in which P_1^2 moves from **T1** to **C1**, but that M_{13} does not contain a transition in which P_1^3 moves from **T1** to **C1**. Then, it would not be possible to compose arcs from P_1^2 and P_1^3 to produce an arc in which P_1 moves from **T1** to **C1**.

To enforce this constraint in our repair, we define a boolean formula over transition variables $E_{s,t}$ whose satisfaction implies it. Consider just two structures M_{ij}, M_{ik}. For M_{ij} and local states s_i, t_i of P_i^j, let $(s_{ij}^1, i, t_{ij}^1), \ldots, (s_{ij}^n, i, t_{ij}^n)$ be all the transitions in M_{ij} that (1) are executed by P_i^j, (2) start with P_i^j in s_i, and (3) end with P_i^j in t_i. Likewise, for pair-structure M_{ik} and the same local states s_i, t_i, let $(s_{ik}^1, i, t_{ik}^1), \ldots, (s_{ik}^m, i, t_{ik}^m)$ be all the transitions in M_{ik} that (1) are executed by P_i^k, (2) start with P_i^k in s_i, and (3) end with P_i^k in t_i. Then, define $grCon(i, j, k, s_i, t_i,) =$

$$E[s_{ij}^1, t_{ij}^1] \vee \cdots \vee E[s_{ij}^n, t_{ij}^n] \equiv E[s_{ik}^1, t_{ik}^1] \vee \cdots \vee E[s_{ik}^m, t_{ik}^m]$$

For clarity of sub/superscripts, we use $E[s, t]$ rather than $E_{s,t}$ here. Now define *grCon* to be the conjunction of the $grCon(i, j, k, s_i, t_i,)$, taken over all $i \in \{1, \ldots, K\}$, $j, k \in I(i), j \neq k$, and all pairs s_i, t_i of local states of P_i. Then, our overall repair formula is

$$grCon \wedge (\bigwedge_{(i,j) \in I} repair(M_{ij}, \eta_{ij}))$$

Eshmun generates this repair formula from the pair-structures M_{ij} and their respective specifications η_{ij}. In particular, it computes *grCon* automatically and conjoins it into the repair formula. A satisfying assignment of this formula gives a repair for each M_{ij} w.r.t. η_{ij} and also ensures that the repaired structures satisfy the process graph consistency constraint. A concurrent program $P = (St_P, P_1 \parallel \cdots \parallel P_K)$ can then be extracted as outlined above. By the large model theorem of [2–4], P satisfies $\bigwedge_{(i,j) \in I} \eta_{ij}$. Let $n_i = |I(i)|$ i.e., n_i is the number of pairs that P_i is involved in. The length of *grCon* is then linear in $\Sigma_{i \in \{1,\ldots,K\}} n_i^2$, since we take overlapping pair-structures (those with a common process P_i) two at a time $(j, k \in I(i), j \neq k)$, and we repeat this for every P_i. It is also linear in the size of the pair-structures. By Prop. 1, the length of each $repair(M_{ij}, \eta_{ij})$ is quadratic in the size of M_{ij} and linear in the length of η_{ij}. So overall the length of the repair formula is quadratic in the n_i, quadratic in the size of the M_{ij}, and linear in the length of the η_{ij}.

Proposition 2. *The length of* $grCon \wedge (\bigwedge_{(i,j) \in I} repair(\eta_{ij},))$ *is* $O(|I| \times |M|^2 \times |\eta| + |M|^2 \times \Sigma_{i \in \{1,\ldots,K\}} n_i^2)$ *where* I *is the number of pairs,* $|M|$ *is the size (states + transitions) of the largest pair-structure* M_{ij}*, and* $|\eta|$ *is the length of the largest pair-specification* η_{ij}*.*

Hence, provided that the SAT solver remains efficient, we can repair a concurrent program $P = (St_P, P_1 \| \cdots \| P_K)$ without incurring state explosion—complexity exponential in K.

8.1 Example: eventually serializable data service

The eventually-serializable data service (ESDS) of Fekete et. al. [17] and Ladin et. al. [23] is a replicated, distributed data service that trades off immediate consistency for improved efficiency. A shared data object is replicated, and the response to an operation at a particular replica may be out of date, i.e., not reflecting the effects of other operations which have not yet been received by that replica. Operations may be reordered *after* the response is issued. Replicas communicate to each other the operations they receive, so that eventually every operation "stabilizes," i.e., its ordering is fixed w.r.t. all other operations. Clients may require an operation to be *strict*, i.e., stable at the time of response (and so it cannot be reordered after the response is issued). Clients may also specify, in an operation x, a set $x.prev$ of operations that must precede x (client-specified constraints, CSC). We let $\mathcal{O}$ be the (countable) set of all operations, $\mathcal{R}$ the set of all replicas, $client(x)$ be the client issuing operation x, $replica(x)$ be the replica that handles

operation x. We use x to index over operations, c to index over clients, and r, r' to index over replicas. For each operation x, we define a client process C_c^x and a replica process R_r^x, where $c = client(x)$, $r = replica(x)$. Thus, a client consists of many processes, one for each operation

that it issues. As the client issues operations, these processes are created dynamically. Likewise a replica consists of many processes, one for each operation that it handles. Thus, we can use dynamic process creation and finite-state processes to model an infinite-state system, such as the one here, which in general handles an unbounded number of operations with time.

We use Eshmun to repair a simple instance of ESDS with one strict operation x, one client **Cx**, a replica **R1x** that processes x, another replica **R2x** that receives gossip of x, and another replica **R2y** that processes an operation $y \in x.prev$. There are three pairs, **Cx** $\|$ **R1x**, **R1x** $\|$ **R2x**, and **R2x** $\|$ **R2y**.**Cx** moves through three local states in sequence: initial state **IN_Cx**, then state **WT_Cx** after **Cx** submits x, and then state **DN_Cx** after **Cx** receives the result of x from **R1x**. **R1x** moves through five local states: initial state **IN_R1x**, then state **WT_R1x** after it receives x from **Cx**, then state **DN_R1x** after it performs x, then state **ST_R1x** when it stabilizes x , and finally state **SNT_R1x** when it sends the result of x to **Cx**. **R2x** moves through four local states: initial state **IN_R2x**, then state **WT_R2x** after it receives x from **R1x**, then state **DN_R2x** after it performs x, and finally state **ST_R2x** when it stabilizes x . **R2y** moves through four local states: initial state **IN_R2y**, then state **WT_R2y** after it receives y from its client (which is not shown), then state **DN_R2y** after it performs y, then state **ST_R2y** when it stabilizes y For each pair, we start with a "naive" pair-structure in which all possible transitions are present, modulo the above sequences. We then repair w.r.t. these pair-specifications:

Client-replica interaction, pair-specification for ***Cx*** $\|$ ***R1x*** where $x \in \mathcal{O}$, **Cx** $= client(x)$, **R1x** $=$ $replica(x)$

- $\mathsf{AG}(\mathbf{WT\_R1x} \Rightarrow \mathbf{WT\_Cx})$: x is not received by **R1x** before it is submitted by **Cx**
- $\mathsf{A}[\mathbf{WT\_R1x} \mathsf{V} (\neg(\mathbf{WT\_Cx} \wedge \mathsf{EG}(\neg \mathbf{WT\_R1x})))]$ if x is submitted by **Cx** then it is received by **R1x**

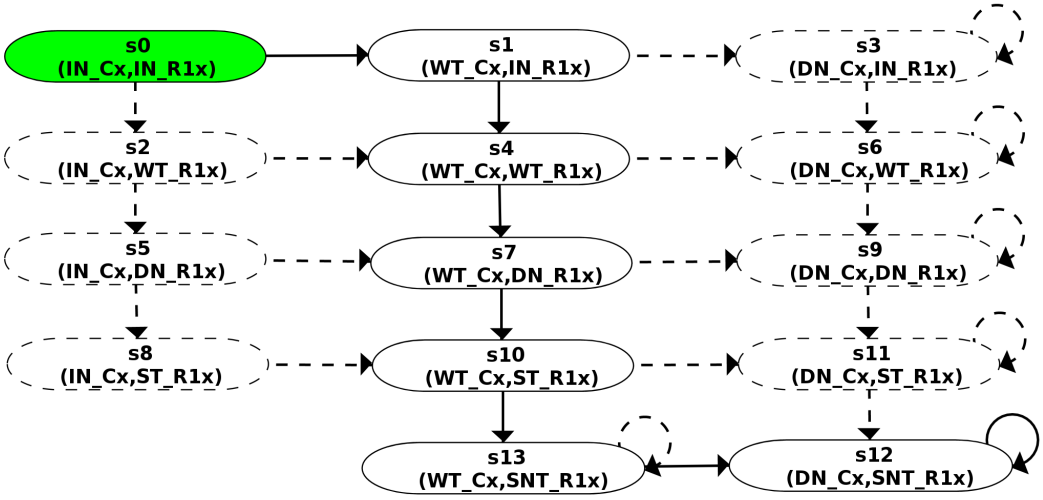

Figure 22: Repaired pair-structure for $\mathbf{Cx} \,\|\, \mathbf{R1x}$

- $\mathsf{AG}(\mathbf{WT\_Cx} \Rightarrow \mathsf{AF}\,\mathbf{DN\_Cx})$: if **Cx** submits x then it eventually receives a result
- $\mathsf{AG}(\mathbf{DN\_Cx} \Rightarrow \mathbf{SNT\_R1x})$: **Cx** does not receive a result before it is sent by **R1x**

The repaired pair-structure is given in Fig. 22.

Pair-specification for $\boldsymbol{R1x} \| \boldsymbol{R2x}$, where $x \in \mathcal{O}$, $x.strict$, $\mathbf{R1x} = replica(x)$

- $\mathsf{AG}(\mathbf{SNT\_R1x} \Rightarrow \mathbf{ST\_R2x})$: the result of a strict operation is not sent to the client until it is stable at all replicas

The repaired pair-structure is given in Fig. 23.

CSC constraints, pair-specification for $\boldsymbol{R1x} \| \boldsymbol{R2y}$, where $x \in \mathcal{O}$, $y \in x.prev$, $\mathbf{R1x} = replica(x)$, $\mathbf{R2y} = replica(y)$

- $\mathsf{AG}(\mathbf{DN\_R1x} \Rightarrow \mathbf{DN\_R2y})$: operation y in $x.prev$ is performed before x is

The repaired pair-structure is given in Fig. 24.

Fig. 25 gives a concurrent program P that is extracted from the repaired pair-structures as discussed above. By the large model theorem [2–4], P satisfies all the above pair-specifications.

9 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.

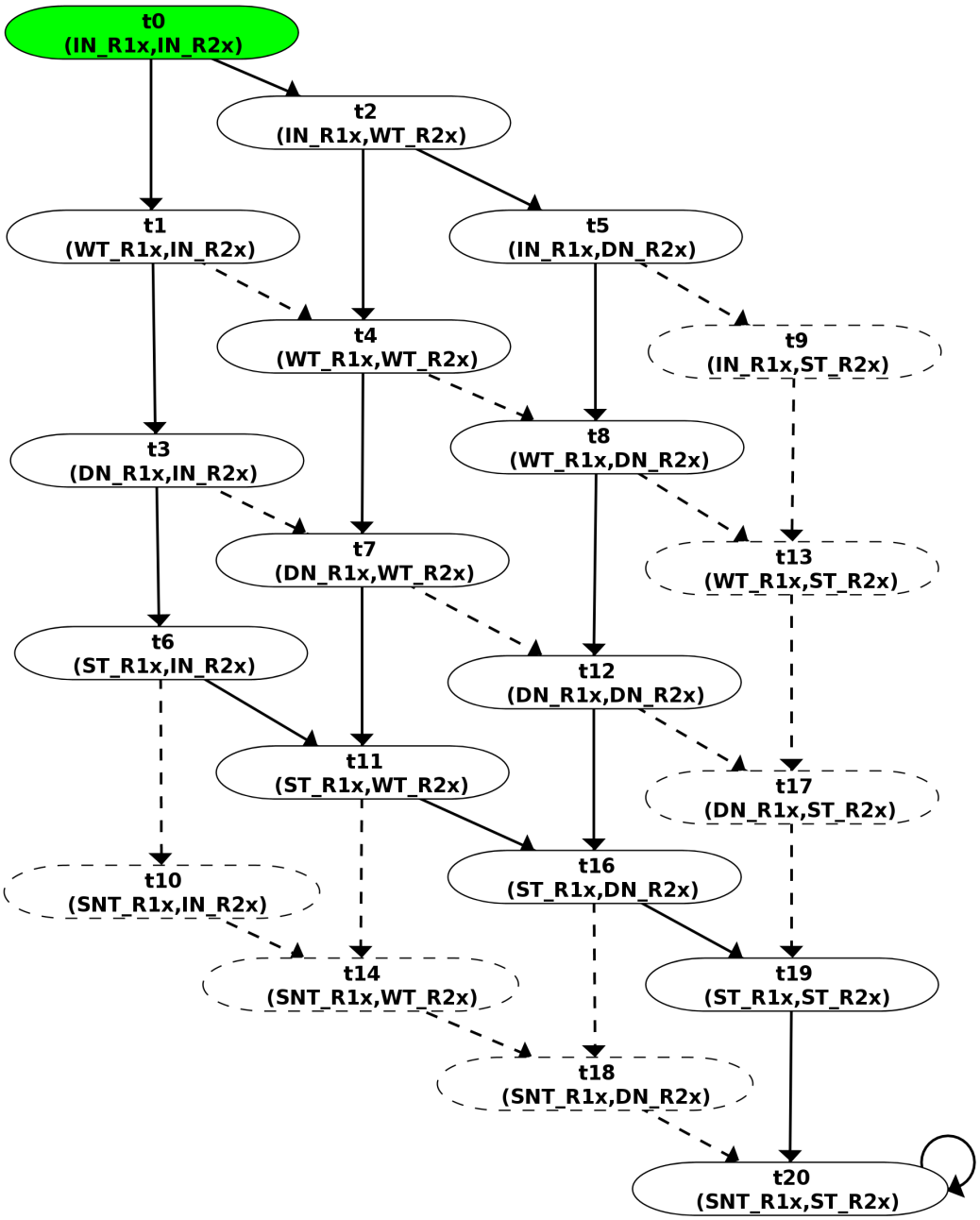

Figure 23: Repaired pair-structure for $\mathbf{R1x} \| \mathbf{R2x}$

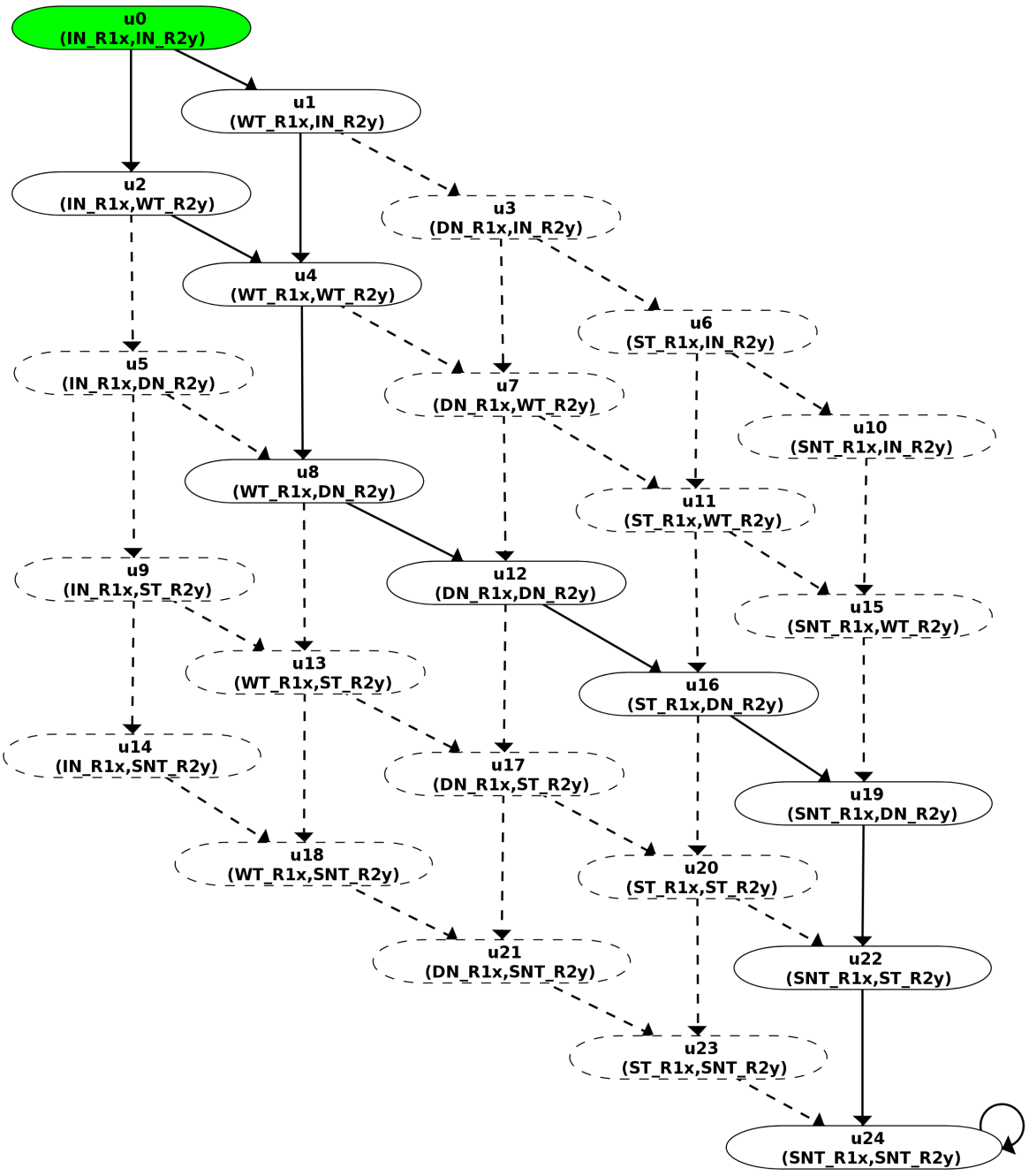

Figure 24: Repaired pair-structure for $\mathbf{R1x} \| \mathbf{R2y}$

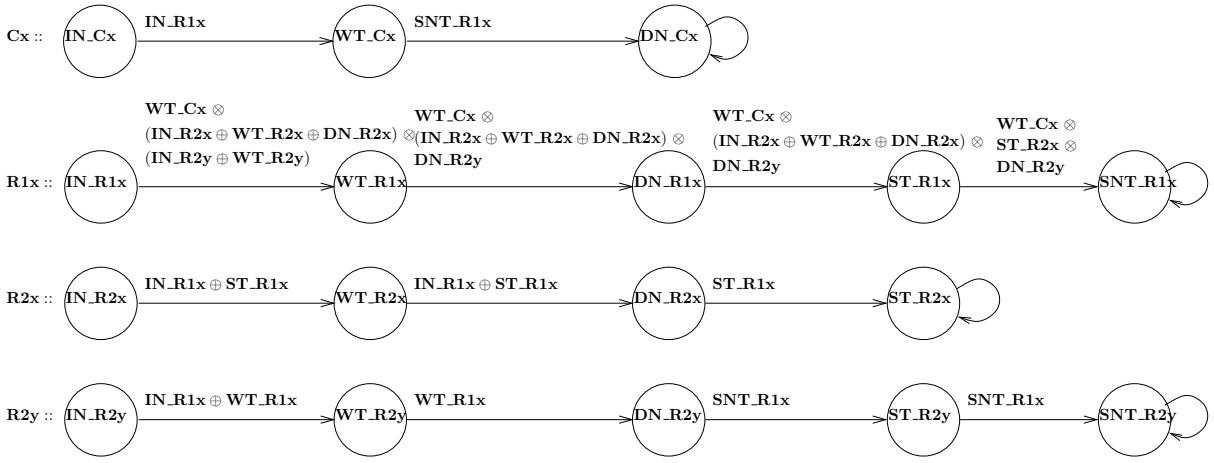

Figure 25: ESDS Program

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 [16] 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, 18], which means that our results are restricted to ACTL-X, the universal fragment of CTL without nexttime [18].

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 [18] 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.

9.1 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. □

9.2 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. □

9.3 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 $M \models \eta$. □

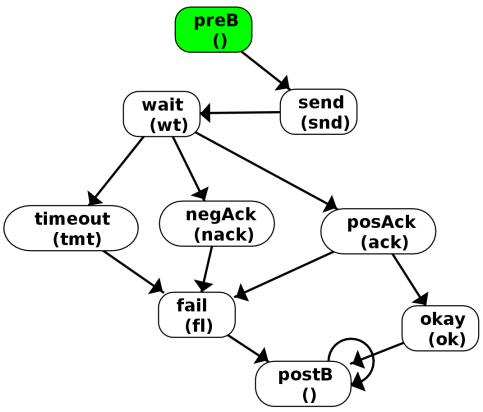

Figure 26: Box B_M for a single phone-call

9.4 Example: phone system

We illustrate hierarchical repair using the phone call example from [1]. Fig. 26 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. 28 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 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: $\mathsf{AG}((ack_1 \vee \mathbf{ack}_2) \Rightarrow \mathsf{AF}\mathbf{sc})$, i.e., if either attempt receives a positive ack, then eventually enter success state.

2. specification formula η_B for B: $\mathsf{AG}(\mathbf{ack} \Rightarrow \mathsf{AF}\mathbf{ok})$, 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. 27.

3. "coupling" formula φ: $\mathsf{AG}((ok_1 \vee \mathbf{ok}_2) \Rightarrow \mathsf{AF}\mathbf{sc}) \wedge \mathsf{AG}(fl1 \Rightarrow \mathsf{AF}(snd2))$, 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. 29.

Eshmun checks the validity of $\eta_B \wedge \varphi \Rightarrow \eta$ by using the CTL decision procedure of [16]: we check satisfiability of $\neg(\eta_B \wedge \varphi \Rightarrow \eta)$. By Th. 4, we conclude that $M \models \eta$.

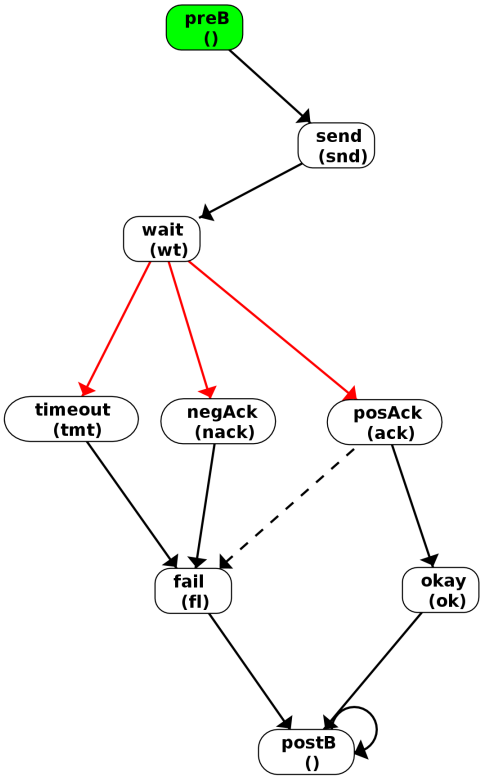

Figure 27: Repaired box B_M for a single phone-call

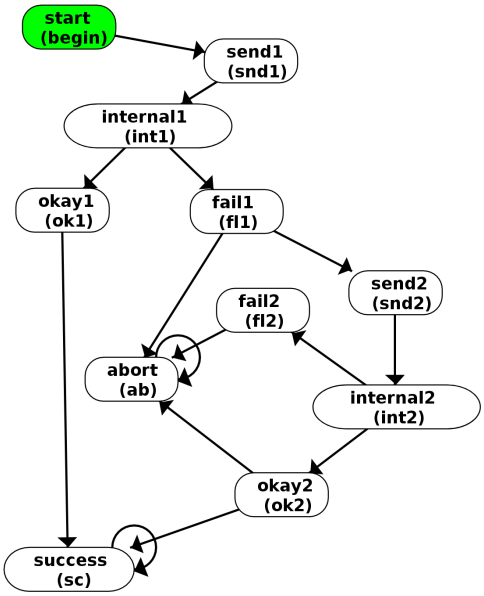

Figure 28: Initial structure M for phone-call example

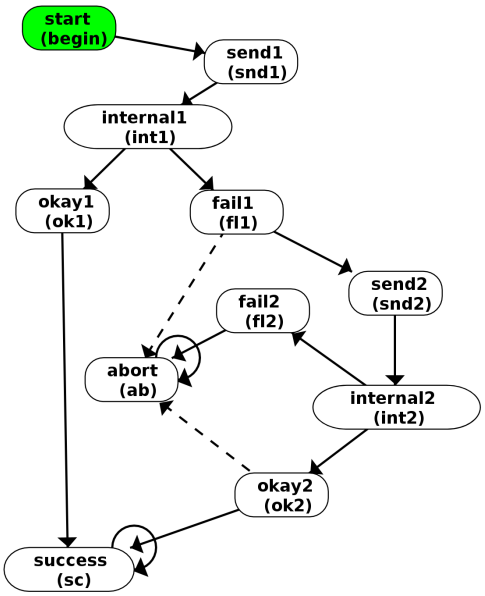

Figure 29: Repaired structure M for phone-call example

10 Overview of our Implementation: The Eshmun Tool

Eshmun is written in Java, and uses the `javax.swing` library for GUI functionality, `Graphviz` [14] for Kripke structure visualization, and `SAT4jSolver` [24] to check satisfiability. Eshmun is an interactive GUI tool; it allows users to create a Kripke structure M by adding states and transitions. Users then enter a CTL formula η and proceed to repair M w.r.t. η. Users can mark a transition as non-deletable by checking the **retain** button. This is essential, for example, in preventing the deletion of transitions of the abstract box B_A. Eshmun implements abstraction, and will show the abstract structure, the repair on the abstract structure, and the concretization of this repair back to the original structure. It also implements the CTL decision procedure [16], for checking validity of $\eta_B \wedge \varphi \Rightarrow \eta$ in hierarchical repair.

10.1 Experimental results

Table 1 gives experimental results for repairing mutual exclusion (w.r.t. safety) and also barrier synchronization. The structures used were generated by a Python program. N gives the number of processes in mutual exclusion and the number of barriers in barrier synchronization. Table 2 gives experimental results for different concurrent programs expressed as concurrent Kripke structures. The experiments were conducted on a linux computer using openJDK 7 with 4GB of ram and 3.3GHz CPU. These agree closely with Prop. 2. For mutex quadratic growth is expected since the number of pairs is $N(N-1)/2$, and the number of pairs that P_i is involved in is $N-1$, and so we expect quadratic growth with N. For dining philosophers (in a ring), the number of pairs is N and the number of pairs that P_i is involved in is 2, and so we expect linear growth with N. The point of Eshmun is not to handle large state spaces, but to provide an interactive environment, with *semantic feedback* for the design and construction of small/medium sized structures. These are to be composed concurrently and/or hierarchically to yield a complex structure which would be exponentially large if "flattened out".

Program Name	N	Basic repair	Abst. by label	Abst. by subformulae
Mutex	2	0.083	0.026	0.036
Mutex	3	0.51	0.25	0.038
Mutex	4	2.34	0.84	0.17
Mutex	5	89.08	2.48	1.59
Barrier	2	0.31	0.11	0.017
Barrier	3	0.67	0.21	0.047
Barrier	4	1.03	0.76	0.11
Barrier	5	1.81	1.04	0.26

Table 1: Times (in seconds) taken to repair the given programs

Processes	5	10	15	20	25	50
Mutex	0.33	0.64	0.84	1.35	1.86	7.25
Dining Phil.	0.48	0.59	0.67	0.74	0.81	1.20

Table 2: Times (in seconds) to repair given concurrent programs

10.2 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.
- 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 [16]

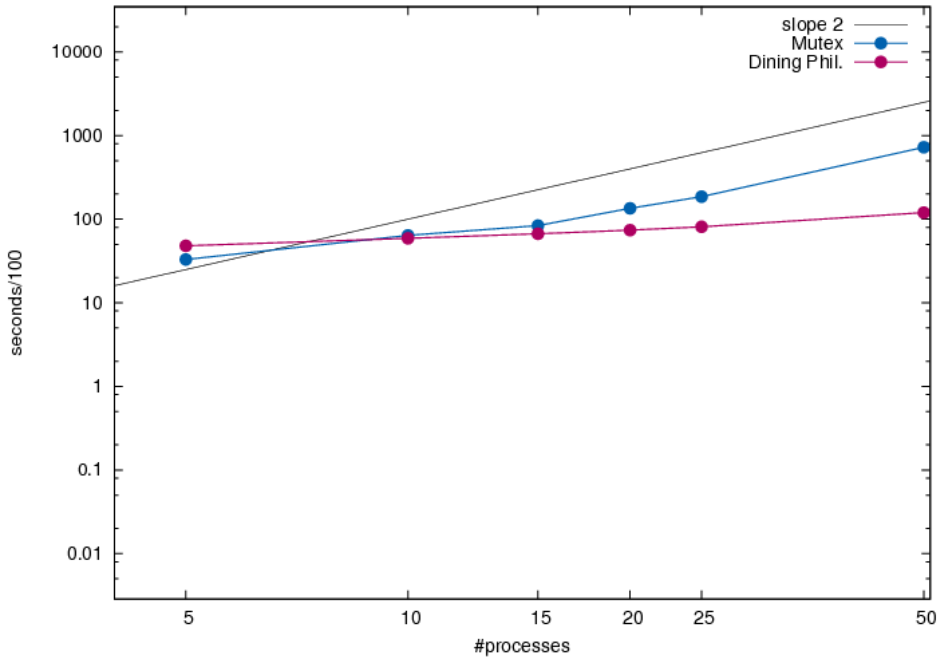

Figure 30: Repair time for given programs

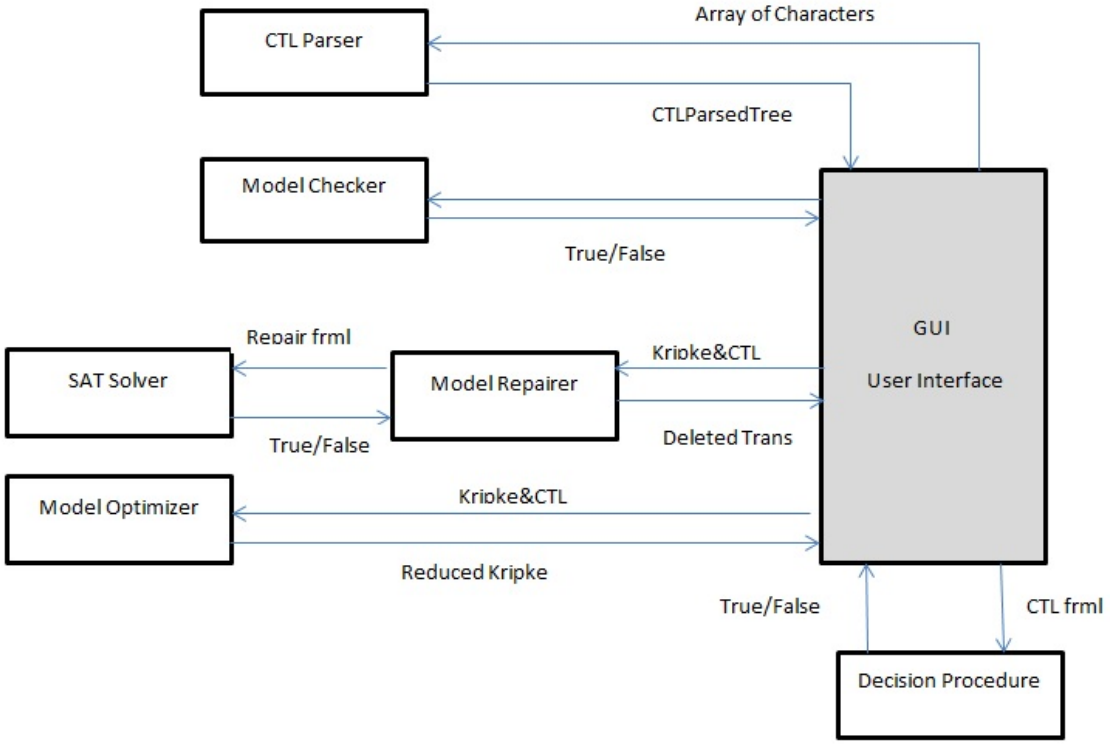

Figure 31: Tool main modules

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

Figure 33 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 32 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 33 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 35), which updates $repair(M, \eta)$, according to Definition 8, by conjoining the appropriate clause.

```
ComputeRepairFormula(repair(M, η))
InitializeRepairFormula(M, η);
forall s_0 ∈ S_0 : new(s) := {η} endfor ;        ▷η must hold in all initial states
repeat until no change
      select some state s in M and some ξ ∈ new(s);
      propagate(s, ξ)
```

Figure 32: The model repair algorithm.

11 Related Work

The use of transition deletion to repair Kripke structures was suggested in Attie and Emerson [5,6] 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. [20]. [12] presents an algorithm for generating counterexamples for symbolic model checking. [20] presents BDD-based algorithms for generating counterexamples ("error traces") for both language containment and fair CTL model checking. Game-based model checking [25, 28] 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 [22, 26, 27] 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. [22] considers only one repair at a time, and their method is complete only for invariants. In Staber et. al. [26], the approach of Jobstmann et. al. [22] 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 [29] 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 a Kripke structure w.r.t. a CTL formula η by deleting transitions and states, and implemented it as an interactive graphical tool, Eshmun. This allows the gradual design and construction of Kripke structures, assisted by immediate semantic

feedback. Our method can handle concurrent and hierarchical Kripke structures, which can be exponentially more succinct than the equivalent flat structure.

A crucial point is that we do not incur state-explosion in dealing with concurrent and hierarchical structures: for concurrent structures, the size of the repair formula grows quadratically with number of processes, in the worst case, and for hierarchical strctures, we repair "one level at a time", and so avoid the exponential blowup causes by replacing boxes by their definitions. We also provided experimental results from our implementation, which validated our avoidance of state-explosion.

Future Work
Future work includes using our implementation to repair larger examples and case studies. Next, we will work on supporting more formal models that express concurrent programs, namely, I/O Automata, and BIP. We want to expand the specifications logic to include ATL, which will help us repair and model multi-agent games and systems. There are many other directions we could expand our work in, including repairing infinite-state model using abstractions, and additive repair (by adding transitions and/or states).

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. Synthesis of large dynamic concurrent programs from dynamic specifications. Technical report, American University of Beirut, Beirut, Lebanon, 2015. To appear in Formal Methods in System Design.

[4] 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.

[5] 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.

[6] 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.

[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] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall Inc., Englewood Cliffs, N.J., 1976.

[14] 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.

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

[16] 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.

[17] A. Fekete, D. Gupta, V. Luchango, N. Lynch, and A. Shvartsman. Eventually-serializable data services. *Theoretical Computer Science*, 220:113–156, 1999.

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

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

[20] 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.

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

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

[23] R. Ladin, B. Liskov, L. Shrira, and S. Ghemawat. Providing high availability using lazy replication. *ACM Transactions on Computer Systems*, 10(4):360–391, Nov. 1992.

[24] 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.

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

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

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

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

[29] 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** ;
    $\mathsf{conjoin}(\text{“}\bigvee_{t\in R[s]}(E_{s,t} \wedge X_{t,\varphi})\text{”})$
  $\xi = \mathsf{A}[\varphi\mathsf{V}\psi]$:
    $new(s) := new(s) \cup \{\mathsf{A}[\varphi\mathsf{V}\psi]^n\}$;
    $\mathsf{conjoin}(\text{“}X_{s,\mathsf{A}[\varphi\mathsf{V}\psi]} \equiv X^n_{s,\mathsf{A}[\varphi\mathsf{V}\psi]}\text{”})$;
  $\xi = \mathsf{A}[\varphi\mathsf{V}\psi]^m$, $m \in \{1, \ldots, n\}$:
    $new(s) := new(s) \cup \{\varphi, \psi\}$;
    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 $t \in R[s] : new(t) := new(t) \cup \{\mathsf{A}[\varphi\mathsf{V}\psi]^{m-1}\}$;
    $\mathsf{conjoin}(\text{“}X^m_{s,\mathsf{E}[\varphi\mathsf{V}\psi]} \equiv X_{s,\psi} \wedge (X_{s,\varphi} \vee \bigvee_{t\in R[s]}(E_{s,t} \wedge X^{m-1}_{t,\mathsf{E}[\varphi\mathsf{V}\psi]}))\text{”})$
  $\xi = \mathsf{E}[\varphi\mathsf{V}\psi]^0$:
    $new(s) := new(s) \cup \{\psi\}$;
    $\mathsf{conjoin}(\text{“}X^0_{s,\mathsf{E}[\varphi\mathsf{V}\psi]} \equiv X_{s,\psi}\text{”})$
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 33: 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 34: 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 35: Adding a conjunct to $repair(M, \eta)$