Model and Program Repair via SAT Solving

PAUL C. ATTIE, KINAN DAK AL BAB, and MOUHAMMAD SAKR,
American University of Beirut

We consider the *subtractive 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 and states. We map an instance $\langle M, \eta \rangle$ of model repair to a Boolean formula $repair(M, \eta)$ such that $\langle M, \eta \rangle$ has a solution iff $repair(M, \eta)$ is satisfiable. Furthermore, a satisfying assignment determines which states and transitions must be removed from M to yield 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 three directions: (1) the use of abstraction mappings, that is, repair a structure abstracted from M and then concretize the resulting repair to obtain a repair of M, (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, and (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.

CCS Concepts: • **Software and its engineering → Model checking**; *State systems*; *Model-driven software engineering*; *Correctness*;

Additional Key Words and Phrases: Model repair, program repair, model checking, temporal logic

ACM Reference format:
Paul C. Attie, Kinan Dak Al Bab, and Mouhammad Sakr. 2017. Model and Program Repair via SAT Solving. *ACM Trans. Embed. Comput. Syst.* 17, 2, Article 32 (December 2017), 25 pages.
https://doi.org/10.1145/3147426

32

1 INTRODUCTION AND MOTIVATION

Temporal logic model checking [17] is an automatic technique for verifying the functional correctness of hardware and software: given (1) a finite state-transition diagram M ("Kripke structure") describing the behavior of the hardware/software artifact in question, and (2) a formula η, written in temporal logic, which specifies a required behavioral property, a model checking algorithm determines whether M satisfies η, that is, whether the artifact behaves as required. For example, M may describe the behavior of a resource controller that allocates resources among a competing set of processes, while η requires that (1) no resource is allocated to two processes at once, and (2)

Authors' addresses: P. C. Attie, Department of Computer Science, American University of Beirut, Beirut, Lebanon; email: paul.attie@aub.edu.lb; K. D. Al Bab (current address), Department of Computer Science, Boston University, Boston MA, USA; email: babman@bu.edu; M. Sakr (current address), Department of Computer Science, University of Saarland, Saarbrücken, Germany; email: sakr@react.uni-saarland.de.

https://doi.org/10.1145/3147426

every request is eventually granted. Model checking has achieved significant industrial application in both hardware and software verification [14]. An important issue in practice is what to do when M does not satisfy η? An initial answer was to produce a *counterexample* [16], which shows a behavior violating η. The designer then uses this to manually modify the software/hardware to eliminate the counterexample as a possible behavior and then tries model checking again. This is repeated until the model check succeeds. A natural question is to attempt to automate this as much as possible, so as to speed up the verify-diagnose-repair cycle used in software and hardware design. This leads to the problem of *model repair*, introduced in Buccafurri et al. [11], as a natural extension of model checking: given a Kripke structure M and a formula η such that M does not satisfy η, is there a way of modifying M to produce some M' that does satisfy η?

This article considers *subtractive model repair*: given M and η, is there a substructure M' of M such that M' satisfies η? We use the propositional branching-time temporal logic CTL [21] to express the formula η, as do many other approaches to model repair [11, 12, 13, 32]. Our algorithm computes (in deterministic polynomial time 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. We submit $repair(M, \eta)$ to a Boolean satisfiability solver ("SAT solver"), and a satisfying assignment for $repair(M, \eta)$, if it exists, determines which transitions and states 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. SAT solvers are an active research area, and so our approach leverages current progress in SAT solving. Considering the resource allocation example given above, our algorithm would delete any transition that grants a resource when that resource is already allocated, thereby preventing the resource from being allocated to two processes at once.

We extend our method to use abstraction mappings (i.e., repair an abstract structure and then concretize to obtain a repair of the original structure), to repair hierarchical Kripke structures [2], and to repair concurrent Kripke structures and concurrent programs. This last extension uses the pairwise approach of Attie and Emerson [8] and Attie [4, 7], thereby avoiding state-explosion. We also show that the subtractive model repair problem is NP-complete.

We have implemented the repair method as a tool, Eshmun,[1] which we used to produce the examples in this article. Eshmun is written in Java, uses the `javax.swing` library for GUI functionality, and the `SAT4jSolver` [28] to check satisfiability. Eshmun is an interactive GUI tool; it allows users to create a Kripke structure M by entering states and transitions. Users then enter a CTL formula η and proceed to repair M w.r.t. η. Eshmun shows a repair by presenting the states/transitions to be deleted using dashed lines. Users can mark a transition/state as non-deletable by checking the **retain** button. The transition/state is then displayed in bold. Eshmun can also display transitions and the local state of different processes in different colors, and has many other features, including comprehensive documentation. 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 repair of concurrent Kripke structures and concurrent programs, and also the CTL decision procedure [21], used for checking validity of CTL formulae, which is useful in hierarchical repair. Eshmun can be downloaded from http://eshmuntool.blogspot.com/.

2 PRELIMINARIES: COMPUTATION TREE LOGIC AND KRIPKE STRUCTURES

Let AP be a set of atomic propositions. The propositional branching-time temporal logic CTL [20, 21] is given by the following grammar:

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

[1] Eshmun is the name of the Phoenician god of healing.

where $p \in AP$, and true, false are constant propositions whose interpretation is the semantic truth values *tt*, *ff*, respectively. The semantics of CTL formulae are defined with respect to a Kripke structure.

Definition 2.1. 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 *successor* of state s in M iff $(s, t) \in R$.

We require that a Kripke structure $M = (S_0, S, R, L, AP)$ be total, that is, $\forall s \in S, \exists s' \in S : (s, s') \in R$. 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. State t is reachable from state s iff there is a path from s to t. State t is reachable iff there is a path from an initial state to t. Without loss of generality, we assume that the Kripke structure M to be repaired does not contain unreachable states, that is, every $s \in S$ is reachable.

A CTL formula φ is evaluated (i.e., is true or false) in a state s of a Kripke structure M. Its value in s depends on the value of its subformulae in states that are reachable from s, and so we define $\models$ by induction on the structure of CTL formulae:

Definition 2.2. $M, s \models \varphi$ means that formula φ is true in state s of structure M, and $M, s \not\models \varphi$ means that φ is false in s.

(1) $M, s \models \mathsf{true}$ and $M, s \not\models \mathsf{false}$
(2) $M, s \models p$ iff $p \in L(s)$ where atomic proposition $p \in AP$
(3) $M, s \models \neg\varphi$ iff $M, s \not\models \varphi$
(4) $M, s \models \varphi \land \psi$ iff $M, s \models \varphi$ and $M, s \models \psi$
(5) $M, s \models \varphi \lor \psi$ iff $M, s \models \varphi$ or $M, s \models \psi$
(6) $M, s \models \mathsf{AX}\varphi$ iff for all t such that $(s, t) \in R : (M, t) \models \varphi$
(7) $M, s \models \mathsf{EX}\varphi$ iff there exists t such that $(s, t) \in R$ and $(M, t) \models \varphi$
(8) $M, s \models \mathsf{A}[\varphi \,\mathsf{R}\, \psi]$ 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 $M, s_k \models \psi$
(9) $M, s \models \mathsf{E}[\varphi \,\mathsf{R}\, \psi]$ 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 $M, s_k \models \psi$

$[\varphi \,\mathsf{R}\, \psi]$ (φ "releases" ψ) means that ψ must hold up to and including the first state where φ holds, and forever if φ never holds. We use $M \models \varphi$ to abbreviate $\forall s_0 \in S_0 : M, s_0 \models \varphi$, that is, φ holds in all initial states of M. We introduce the abbreviations $\mathsf{A}[\phi \,\mathsf{U}\, \psi]$ for $\neg\mathsf{E}[\neg\varphi \,\mathsf{R}\, \neg\psi]$, $\mathsf{E}[\phi \,\mathsf{U}\, \psi]$ for $\neg\mathsf{A}[\neg\varphi \,\mathsf{R}\, \neg\psi]$, $\mathsf{AF}\varphi$ for $\mathsf{A}[\mathsf{true} \,\mathsf{U}\, \varphi]$, $\mathsf{EF}\varphi$ for $\mathsf{E}[\mathsf{true} \,\mathsf{U}\, \varphi]$, $\mathsf{AG}\varphi$ for $\mathsf{A}[\mathsf{false} \,\mathsf{R}\, \varphi]$, $\mathsf{EG}\varphi$ for $\mathsf{E}[\mathsf{false} \,\mathsf{R}\, \varphi]$. We use *tt*, *ff* for the (semantic) truth values of true, false, respectively. Whereas true, false are atomic propositions whose interpretation is always *tt*, *ff*, respectively.

Definition 2.3 (Formula expansion). Given a CTL formula φ, its set of subformulae $sub(\varphi)$ is defined by induction on the structure of CTL formulae, as follows:

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

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 3.1 (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, denoted $M' \subseteq M$, iff $S'_0 \subseteq S_0$, $S' \subseteq S$, $R' \subseteq R$, $AP' = AP$, and $L' = L \upharpoonright S'$ (where $\upharpoonright$ denotes domain restriction).

Definition 3.2 (Repairable). Given Kripke structure M and CTL formula η, we say that M is *repairable* with respect to η iff there exists a Kripke structure M' such that M' is total, $M' \subseteq M$, and $M' \models \eta$.

Definition 3.3 (Model Repair Problem). We use $\langle M, \eta \rangle$ to denote the repair problem for Kripke structure M and CTL formula η. The decision version of $\langle M, \eta \rangle$ is to decide if M is repairable w.r.t. η. The functional version of $\langle M, \eta \rangle$ is to return an M' that satisfies Definition 3.2, in the case that M is repairable w.r.t. η.

3.1 Complexity of the Model Repair Problem

Theorem 3.4. *The decision version of the model repair problem is NP-complete.*

Proof. Let $\langle M, \eta \rangle$ 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. $M' \models \eta$ is verified in quadratic time using $|S'_0|$ invocations of the CTL model checking algorithm of Clarke et al. [17].

NP-hardness: We reduce 3SAT to model repair. Let $f = \bigwedge_{1 \le i \le n}(a_i \vee b_i \vee c_i)$ be a Boolean formula in 3CNF, where a_i, b_i, c_i are literals over some set $x_1, \ldots, x_m$ of propositions, that is, 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 $\langle M, \eta \rangle$ where $M = (\{s_0\}, S, R, L, AP)$, $S = \{s_0, s_1, \ldots, s_m, t_1, \ldots, t_m\}$, and $R = \{(s_0, s_1), \ldots, (s_0, s_m), (s_1, t_1), \ldots, (s_m, t_m), (t_1, t_1), \ldots, (t_m, t_m)\}$, that is, transitions from s_0 to each of $s_1, \ldots, s_m$, a transition from s_i to t_i, and a self-loop on each t_i, for $1 \le i \le m$. Also $AP = \{p_1, \ldots p_m, q_1, \ldots, q_m\}$, and the propositions in AP are distinct from the $x_1, \ldots, x_m$ used in f. L is given by

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

Also $\eta = \bigwedge_{1 \le i \le n}(\varphi_i^1 \vee \varphi_i^2 \vee \varphi_i^3)$ where:

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

Thus, if $a_i = x_i$, then the transition from s_i to t_i (written as $s_i \rightarrow t_i$) must be retained in M', and if $a_i = \neg x_i$, then 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 $\langle M, \eta \rangle$ has a solution: Let $\mathcal{V} : \{x_1, \ldots, x_m\} \mapsto \{tt, ff\}$ be a satisfying truth assignment for f. Define R' as follows. $R' = \{(s_0, s_i), (s_i, s_i), (t_i, t_i) \mid 1 \le i \le m\} \cup \{(s_i, t_i) \mid \mathcal{V}(x_i) = tt\}$, that is, 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 $\mathcal{V}(x_i) = ff$. We show that $M', s_0 \models \eta$. Since $\mathcal{V}$ is a satisfying assignment, we have $\bigwedge_{1 \le i \le n}(\mathcal{V}(a_i) \vee \mathcal{V}(b_i) \vee \mathcal{V}(c_i))$. Without loss of generality, assume that $\mathcal{V}(a_i) = tt$ (similar argument for $\mathcal{V}(b_i) = tt$ and $\mathcal{V}(c_i) = tt$). We have two cases. Case 1 is $a_i = x_j$. Then $\mathcal{V}(x_j) = tt$,

so $(s_j, t_j) \in R'$. Also, since $a_i = x_j$, $\varphi_i^1 = \mathsf{AG}(p_j \Rightarrow \mathsf{EX}q_j)$. Since $(s_j, t_j) \in R'$, $M', s_0 \models \varphi_i^1$. Hence, $M', s_0 \models \eta$. Case 2 is $a_i = \neg x_j$. Then $\mathcal{V}(x_j) = ff$, so $(s_j, t_j) \notin R'$. Also, since $a_i = \neg x_j$, $\varphi_i^1 = \mathsf{AG}(p_j \Rightarrow \mathsf{AX}\neg q_j)$. Since $(s_j, t_j) \notin R'$, $M', s_0 \models \varphi_i^1$. Hence, $M', s_0 \models \eta$.

f is satisfiable follows from $\langle M, \eta \rangle$ has a solution: 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 $\mathcal{V}(f) = tt$, that is, $\mathcal{V}(a_i) \vee \mathcal{V}(b_i) \vee \mathcal{V}(c_i)$ 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, $\mathcal{V}(a_i) \vee \mathcal{V}(b_i) \vee \mathcal{V}(c_i)$, and so $\mathcal{V}(f) = tt$. Case 2 is $a_i = \neg x_j$. Then $\varphi_i^1 = \mathsf{AG}(p_j \Rightarrow \mathsf{AX}\neg q_j)$. Since $M', s_0 \models \varphi_i^1$, we must have $(s_j, t_j) \notin R'$. Hence, $\mathcal{V}(x_j) = ff$. Therefore, $\mathcal{V}(a_i) = tt$. Hence, $\mathcal{V}(a_i) \vee \mathcal{V}(b_i) \vee \mathcal{V}(c_i)$, and so $\mathcal{V}(f) = tt$. □

4 THE MODEL REPAIR ALGORITHM

Given an instance $\langle M, \eta \rangle$ of model repair, where $M = (S_0, S, R, L, AP)$ and η is a CTL formula, we define a Boolean formula $repair(M, \eta)$ such that $repair(M, \eta)$ is satisfiable iff $\langle M, \eta \rangle$ has a solution. $repair(M, \eta)$ contains the following propositions, which have the indicated meanings, where $\mathcal{V}$ is a satisfying Boolean assignment for $repair(M, \eta)$, and M' is the repaired structure. The subscripts indicate dependence, for example, one $E_{s,t}$ variable per transition (s, t), one X_s per state s, and so on:

(1) $E_{s,t}$, $(s, t) \in R$: transition (s, t) is retained in M' iff $\mathcal{V}(E_{s,t}) = tt$
(2) $X_s, s \in S$: state s is retained in M' iff $\mathcal{V}(X_s) = tt$
(3) $X_{s,\psi}, s \in S, \psi \in sub(\eta)$: $M', s \models \psi$, that is, ψ holds in state s of M' iff $\mathcal{V}(X_{s,\psi}) = tt$
(4) $X^n_{s,\psi} : s \in S, 0 \le n \le |S|$, and $\psi \in sub(\eta)$ has the form $\mathsf{A}[\varphi \mathsf{R} \varphi']$ or $\mathsf{E}[\varphi \mathsf{R} \varphi']$: $X^n_{s,\psi}$ is used to propagate release formulae (AR or ER) for as long as necessary to determine their truth, that is, $|S|$ times.

The repair formula $repair(M, \eta)$ encodes the usual local constraints, for example, $\mathsf{AX}\varphi$ holds in s iff φ holds in all successors of s, that is, all t such that $(s, t) \in R$. We modify these, however, to account for transition deletion. So, the local constraint for AX becomes $\mathsf{AX}\varphi$ holds in s iff φ holds in all successors of s *after* transition deletion, that is, instead of $X_{s,\mathsf{AX}\varphi} \equiv \bigwedge_{t|s\to t} X_{t,\varphi}$, we have $X_{s,\mathsf{AX}\varphi} \equiv \bigwedge_{t|s\to t}(E_{s,t} \Rightarrow X_{t,\varphi})$, where $s \to t$ abbreviates $(s, t) \in R$. EX is treated similarly. AR, ER are dealt with using a "counting" technique, discussed in the following. There are also clauses for propositional consistency, propositional labeling, initial states, and to require that the repaired structure M' is total.

Definition 4.1 (repair(M, η)). Let η be a CTL formula and $M = (S_0, S, R, L, AP)$ be a Kripke structure. $repair(M, \eta)$ is the conjunction of all the propositional formulae listed below. These are grouped into sections, where each section deals with one issue, for example, propositional consistency. s, t implicitly range over S. Other ranges are explicitly given.

(1) some initial state s_0 is not deleted
$\bigvee_{s_0 \in S_0} X_{s_0}$
(2) M' satisfies η, that is, undeleted initial states satisfy η
for all $s_0 \in S_0 : X_{s_0} \Rightarrow X_{s_0,\eta}$
(3) M' is total, that is, 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 $p \in AP \cap L(s)$: $X_{s,p}$
for all $p \in AP - L(s)$: $\neg X_{s,p}$
(6) Propositional consistency
for all $\neg\varphi \in sub(\eta)$: $X_{s,\neg\varphi} \equiv \neg X_{s,\varphi}$
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 $\mathsf{AX}\varphi \in sub(\eta)$: $X_{s,\mathsf{AX}\varphi} \equiv \bigwedge_{t|s\to t}(E_{s,t} \Rightarrow X_{t,\varphi})$
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 $n = |S|$, that is, the number of states in M.
for all $\mathsf{A}[\varphi \mathsf{R} \psi] \in sub(\eta)$, $m \in \{1 \ldots n\}$:
$$X_{s,\mathsf{A}[\varphi \mathsf{R} \psi]} \equiv X^n_{s,\mathsf{A}[\varphi \mathsf{R} \psi]}$$
$$X^m_{s,\mathsf{A}[\varphi \mathsf{R} \psi]} \equiv X_{s,\psi} \wedge (X_{s,\varphi} \vee \textstyle\bigwedge_{t|s\to t}(E_{s,t} \Rightarrow X^{m-1}_{t,\mathsf{A}[\varphi \mathsf{R} \psi]}))$$
$$X^0_{s,\mathsf{A}[\varphi \mathsf{R} \psi]} \equiv X_{s,\psi}$$
for all $\mathsf{E}[\varphi \mathsf{R} \psi] \in sub(\eta)$, $m \in \{1 \ldots n\}$:
$$X_{s,\mathsf{E}[\varphi \mathsf{R} \psi]} \equiv X^n_{s,\mathsf{E}[\varphi \mathsf{R} \psi]}$$
$$X^m_{s,\mathsf{E}[\varphi \mathsf{R} \psi]} \equiv X_{s,\psi} \wedge (X_{s,\varphi} \vee \textstyle\bigvee_{t|s\to t}(E_{s,t} \wedge X^{m-1}_{t,\mathsf{E}[\varphi \mathsf{R} \psi]}))$$
$$X^0_{s,\mathsf{E}[\varphi \mathsf{R} \psi]} \equiv X_{s,\psi}$$

We handle the $[\varphi \mathsf{R} \psi]$ modality by "counting down", as follows. Along each path, either (1) a state is reached where $[\varphi \mathsf{R} \psi]$ is discharged ($\varphi \wedge \psi$), or (2) $[\varphi \mathsf{R} \psi]$ is shown to be false ($\neg\varphi \wedge \neg\psi$), or (3) some state eventually repeats. In case (3), we know that $[\varphi \mathsf{R} \psi]$ also holds along this path. Thus, by expanding the release modality up to $|S|$ times, we ensure that the third case holds if the first two have not yet resolved the truth of $[\varphi \mathsf{R} \psi]$ along the path in question. We therefore use a version of $X_{s,\mathsf{A}[\varphi \mathsf{R} \psi]}$ that is superscripted with an integer between 0 and $|S|$. This imposes a "well foundedness" on the $X^m_{s,\mathsf{A}[\varphi \mathsf{R} \psi]}$ propositions, and prevents, for example, a cycle along which ψ holds in all states and yet the $X_{s,\mathsf{A}[\varphi \mathsf{R} \psi]}$ are assigned false in all states.

States rendered unreachable are still required to have some outgoing transition, but this does not affect the final result, since unreachable states do not affect reachable ones. Hence, an unreachable state can retain its successors (recall that the initial structure M is total) without impacting the repair.

A satisfying assignment $\mathcal{V}$ of $repair(M, \eta)$ defines a single solution to model repair. Denote this solution by $model(M, \mathcal{V})$. It is as follows.

Definition 4.2 (model$(M, \mathcal{V})$). $model(M, \mathcal{V})$ is the Kripke structure (S'_0, S', R', L', AP'), where $S'_0 = \{s_0 \mid s_0 \in S_0 \wedge \mathcal{V}(X_{s_0}) = tt\}$, $S' = \{s \mid s \in S \wedge \mathcal{V}(X_s) = tt\}$, $R' = \{(s,t) \mid (s,t) \in R \wedge \mathcal{V}(E_{s,t}) = tt\}$, $L' = L \upharpoonright S'$, and $AP' = AP$.

Figure 1 presents our model repair algorithm, Repair(M, η), which we show is sound, and complete provided that a complete SAT-solver is used. We also show that $repair(M, \eta)$ has length polynomial in the sizes of M and η, and so can be constructed in polynomial time. Note that we use different fonts to indicate the repair algorithm Repair(M, η), as opposed to the repair formula $repair(M, \eta)$.

Proposition 4.3. *The length of repair(M, η) 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 number*

> Repair(M, η):
>
> **if** $M, S_0 \models \eta$ (by model checking) **return** M and terminate;
> submit $repair(M, \eta)$ to a (complete) SAT-solver;
> **if** the SAT-solver returns satisfying assignment $\mathcal{V}$ **then**
>   **return** $M' = model(M, \mathcal{V})$
> **else return** "the model cannot be repaired"

Fig. 1. The model repair algorithm.

of atomic propositions in AP, and d is the outdegree of M, that is, the maximum of the number of successors that any state in M has.

Proof. $repair(M, \eta)$ is the conjunction of Clauses 1–8 of Definition 4.1. 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 $sub(\eta)$ whose main operator is propositional, and $|sub(\eta)|$ 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 $sub(\eta)$ 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 $m \in \{1, \dots, |S|\}$. Each of these formulae has length $O(d)$. The sum of lengths of all these formulae is $O(|S|^2 \times |\eta| \times d)$.

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

Theorem 4.4 (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 $\mathcal{V}(X_{s,\xi}) = f\!f$.

Case $\xi = p$ where $p \in AP$: $\mathcal{V}(X_{s,\xi}) = tt$ iff (case condition) $\mathcal{V}(X_{s,p}) = tt$ iff (Definition 4.1 Clause 5) $p \in L(s)$ iff (CTL semantics) $M', s \models p$ iff (case condition) $M', s \models \xi$.

Case $\xi = \neg\varphi$: $\mathcal{V}(X_{s,\xi}) = tt$ iff (case condition) $\mathcal{V}(X_{s,\neg\varphi}) = tt$ iff (Definition 4.1 Clause 6) $\mathcal{V}(X_{s,\varphi}) = f\!f$ 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 (Definition 4.1 Clause 6) $\mathcal{V}(X_{s,\varphi}) = tt$ 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 (Definition 4.1 Clause 6) $\mathcal{V}(X_{s,\varphi}) = tt$ 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 $\xi = \mathrm{AX}\,\varphi$: $\mathcal{V}(X_{s,\xi}) = tt$ iff (case condition) $\mathcal{V}(X_{s,\mathrm{AX}\varphi}) = tt$ iff (Definition 4.1 Clause 7) $\bigwedge_{t|s\to t} \mathcal{V}(E_{s,t} \Rightarrow X_{t,\varphi}) = tt$ iff (Boolean semantics of $\Rightarrow$) $\bigwedge_{t|s\to t} \mathcal{V}(E_{s,t}) = tt \Rightarrow \mathcal{V}(X_{t,\varphi}) = tt$ iff (s reachable by assumption, and $E_{s,t}$ implies that t is reachable. Now apply the induction hypothesis) $\bigwedge_{t|s\to t}(s,t) \in R' \Rightarrow M', t \models \varphi$ iff ($R' \subseteq R$, so restriction of t to $t \mid s \to t$, that is, $t \mid (s,t) \in R$, does not matter) $M', s \models \mathrm{AX}\varphi$ iff (case condition) $M', s \models \xi$.

Case $\xi = \mathrm{EX}\,\varphi$: $\mathcal{V}(X_{s,\xi}) = tt$ iff (case condition) $\mathcal{V}(X_{s,\mathrm{EX}\varphi}) = tt$ iff (Definition 4.1 Clause 7) $\bigvee_{t|s\to t} \mathcal{V}(E_{s,t} \wedge X_{t,\varphi}) = tt$ iff (Boolean semantics of $\wedge$) $\bigvee_{t|s\to t} \mathcal{V}(E_{s,t}) = tt \wedge \mathcal{V}(X_{t,\varphi}) = tt$ iff (s reachable by assumption, and $E_{s,t}$ implies that t is reachable. Now apply the induction hypothesis) $\bigvee_{t|s\to t}(s,t) \in R' \wedge M', t \models \varphi$ iff ($R' \subseteq R$, so restriction of t to $t \mid s \to t$, that is, $t \mid (s,t) \in R$, does not matter) $M', s \models \mathrm{EX}\varphi$ iff (case condition) $M', s \models \xi$.

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

Left to right, that is, $\mathcal{V}(X_{s,\mathrm{A}[\varphi \mathrm{R} \psi]})$ *implies* $M', s \models \mathrm{A}[\varphi \,\mathrm{R}\, \psi]$: $\mathcal{V}(X_{s,\mathrm{A}[\varphi \mathrm{R} \psi]})$ iff (Definition 4.1, Clause 8) $\mathcal{V}(X^n_{s,\mathrm{A}[\varphi \mathrm{R} \psi]})$ iff (Definition 4.1, Clause 8) $\mathcal{V}(X_{s,\psi} \wedge (X_{s,\varphi} \vee \bigwedge_{t|s\to t}(E_{s,t} \Rightarrow X^{n-1}_{t,\mathrm{A}[\varphi \mathrm{R} \psi]})))$ iff ($\mathcal{V}$ is a Boolean valuation function, and so distributes over Boolean connectives) $\mathcal{V}(X_{s,\psi}) \wedge (\mathcal{V}(X_{s,\varphi}) \vee (\bigwedge_{t|s\to t} \mathcal{V}(E_{s,t}) \Rightarrow \mathcal{V}(X^{n-1}_{t,\mathrm{A}[\varphi \mathrm{R} \psi]})))$ iff (induction hypothesis) $(M', s \models \psi)$ and $((M', s \models \varphi)$ or $(\bigwedge_{t|s\to t}(s,t) \in R' \Rightarrow \mathcal{V}(X^{n-1}_{t,\mathrm{A}[\varphi \mathrm{R} \psi]})))$. We now have two cases:

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

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

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

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 $[\varphi \,\mathrm{R}\, \psi]$ 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 $[\varphi \,\mathrm{R}\, \psi]$ holds along this path. We conclude that $[\varphi \,\mathrm{R}\, \psi]$ holds along all paths starting in s, and so $M', s \models \mathrm{A}[\varphi \,\mathrm{R}\, \psi]$.

Right to left, that is, $\mathcal{V}(X_{s,\mathrm{A}[\varphi \mathrm{R} \psi]})$ *follows from* $M', s \models A[\varphi \,R\, \psi]$: Assume that $M', s \models \mathrm{A}[\varphi \,\mathrm{R}\, \psi]$ holds. By CTL semantics, $M', s \models \psi \wedge (M', s \models \varphi \vee \bigwedge_{t|t\to s}((s,t) \in R' \Rightarrow M', t \models \mathrm{A}[\varphi \,\mathrm{R}\, \psi]))$. By the induction hypothesis, $\mathcal{V}(X_{s,\psi}) \wedge (\mathcal{V}(X_{s,\varphi}) \vee \bigwedge_{t|t\to s}((s,t) \in R' \Rightarrow M', t \models \mathrm{A}[\varphi \,\mathrm{R}\, \psi]))$. 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 $\mathcal{V}(X_{s,\mathrm{A}[\varphi \mathrm{R} \psi]})$ by Definition 4.1, Clause 8, and so we are done.
(2) $\bigwedge_{t|s\to t}(s,t) \in R' \Rightarrow M', t \models \mathrm{A}[\varphi \,\mathrm{R}\, \psi]$

For Case (2), we proceed as follows. Let t be an arbitrary state such that $(s,t) \in R'$. Then $M', t \models \mathsf{A}[\varphi \mathsf{R} \psi]$. If we show that $\mathcal{V}(X^{n-1}_{t,\mathsf{A}[\varphi \mathsf{R} \psi]})$ follows from $M', t \models \mathsf{A}[\varphi \mathsf{R} \psi]$, then we can conclude $\mathcal{V}(X_{s,\mathsf{A}[\varphi \mathsf{R} \psi]})$ by Definition 4.1. Proceeding as for s previously, we conclude $\mathcal{V}(X_{t,\psi})$ and one of the same two cases as previously:

(1) $\mathcal{V}(X_{t,\varphi})$, so by Definition 4.1, $\mathcal{V}(X^{n-1}_{t,\mathsf{A}[\varphi \mathsf{R} \psi]})$ holds.
(2) $\bigwedge_{u|t \to u}(t,u) \in R' \Rightarrow \mathcal{V}(X^{n-2}_{u,\mathsf{A}[\varphi \mathsf{R} \psi]})$

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 $\mathcal{V}(X_{s,\mathsf{A}[\varphi \mathsf{R} \psi]})$ is concerned, or we keep going until the counter reaches 0, say at state v. Let π be the path from s to v, and let w any state along π. Then, $M', w \models \psi$, and so $\mathcal{V}(X_{w,\psi})$ by the induction hypothesis. By Definition 4.1, Clause 5, $X^0_{v,\mathsf{A}[\varphi \mathsf{R} \psi]} \equiv X_{v,\psi}$, and so $\mathcal{V}(X^0_{v,\mathsf{A}[\varphi \mathsf{R} \psi]})$. Applying Definition 4.1, Clause 5 along π, and noting that π is an arbitrary path starting in s, we have that $\mathcal{V}(X_{s,\mathsf{A}[\varphi \mathsf{R} \psi]})$ holds.

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

Corollary 4.5 (Soundness). *If* Repair(M, η) *returns a Kripke structure M', then (1) M' is total, (2) $M' \subseteq M$, (3) $M' \models \eta$, and (4) M is repairable.*

Proof. Since Repair(M, η) returns a Kripke structure, it follows from Figure 1 that the invoked SAT-solver returned a satisfying truth assignment $\mathcal{V}$ for $repair(M, \eta)$, and also that $M' = model(M, \mathcal{V})$. (1) follows from Definition 4.2 and Clauses 3 and 4 of Definition 4.1. (2) holds by construction of M' (Definition 4.2), which is derived from M by deleting transitions and states. For (3), let $M' = (S'_0, S', R', L', AP')$ and s_0 be an arbitrary state in S'_0. Since s_0 was not deleted, we have, by Definition 4.2, $\mathcal{V}(X_{s_0}) = tt$. Hence, by Clause 2 of Definition 4.1, $\mathcal{V}(X_{s_0,\eta}) = tt$. Hence, by Theorem 4.4, $M', s_0 \models \eta$, and so (3) holds. Finally, (4) follows from (1)–(3) and Definition 3.2. □

Theorem 4.6 (Completeness). *If M is repairable with respect to η, then* Repair(M, η) *returns a Kripke structure M'' such that M'' is total, $M'' \subseteq M$, and $M'' \models \eta$.*

Proof. Assume that M is repairable with respect to η. By Definition 3.2, there exists a total substructure $M' = (S'_0, S', R', L', AP')$ of M such that $M' \models \eta$. We define a valuation $\mathcal{V}$ for $repair(M, \eta)$ as follows, and show that it satisfies all clauses of Definition 4.1.

Assign tt to $E_{s,t}$ for every edge $(s,t) \in R'$ and ff to every $E_{s,t}$ for every edge $(s,t) \notin R'$. Assign tt to X_s for every state $s \in S'$, and ff to every $s \notin S'$. Since M' is total, Clauses 3 and 4 are satisfied by this assignment. Since $S'_0 \neq \emptyset$, Clause 1 is satisfied.

Select an arbitrary $s_0 \in S'_0$ and consider an execution of the CTL model checking algorithm of Clarke et al. [17] for checking $M', s_0 \models \eta$. This algorithm will assign a value to every formula φ in $sub(\eta)$ in every state s of M' that is reachable from s_0. Set $\mathcal{V}(X_{s,\varphi})$ to this value. To cover all states in M', repeat the previous, starting from another state $s'_0 \in S'_0$, but *retaining* all the valuations to the formulae in $sub(\eta)$ made by the first execution. (Recall that the model checking algorithm checks subformulae of η before checking η). Repeat for all states in S'_0.

By construction and correctness of the model checking algorithm [17], the resulting valuations satisfy Clauses 2, 5, 6, 7, and 8.

Since all clauses of Definition 4.1 are satisfied, 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.

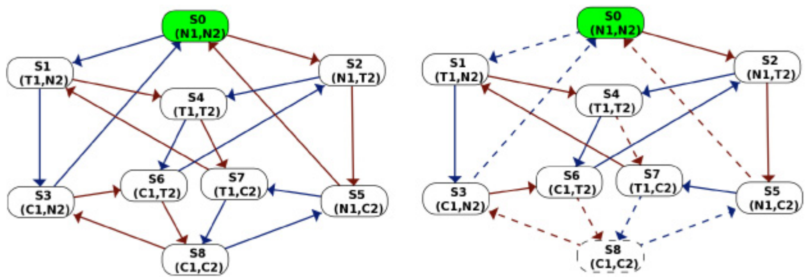

Fig. 2. Mutex: initial structure and basic repair.

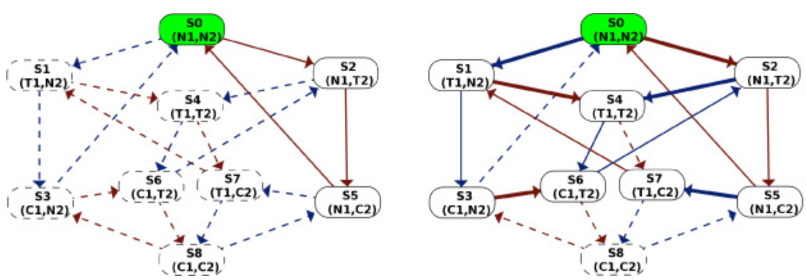

Fig. 3. Mutex: overconstrained repair (left), and only P_1 is live (right).

Now the SAT-solver used is assumed to be complete, and so returns some satisfying assignment for $repair(M, \eta)$, not necessarily $\mathcal{V}$, since there may be more than one satisfying assignment. Thus, Repair(M, η) returns a structure M'', rather than "failure." By Corollary 4.5, M'' is total, $M'' \subseteq M$, and $M'' \models \eta$. □

4.1 Example: Mutual Exclusion with Safety

Consider two-process mutual exclusion, in which P_i $(i = 1, 2)$ cycles through three states: neutral (performing local computation), trying (requested the critical section), and critical (inside the critical section). P_i has three atomic propositions, **Ni**, **Ti**, **Ci**. In the neutral state, **Ni** is true and **Ti**, **Ci** are false. In the trying state, **Ti** is true and **Ni**, **Ci** are false. In the critical state, **Ci** is true and **Ni**, **Ti** are false. The specification η is $\mathrm{AG}(\neg(\mathbf{C1} \wedge \mathbf{C2}))$, that is, P_1 and P_2 are never simultaneously in their critical sections.

Figure 2 (left) shows an initial Kripke structure M, which violates mutual exclusion, and Figure 2 (right) shows a repair of M w.r.t. $\mathrm{AG}(\neg(\mathbf{C1} \wedge \mathbf{C2}))$. Eshmun displays the transitions to be deleted (to effect the repair) as dashed. Initial states are colored green, and the text attached 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 the repaired structure the violating state **S8** is now unreachable, and so $\mathrm{AG}(\neg(\mathbf{C1} \wedge \mathbf{C2}))$ is satisfied. This repair is, however, overly restrictive: 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.

4.2 Example: Interactive Design Using Semantic Feedback

We add liveness to the mutex example by repairing Figure 2 (left) w.r.t. $\mathrm{AG}(\neg(\mathbf{C1} \wedge \mathbf{C2})) \wedge \mathrm{AG}(\mathbf{T1} \Rightarrow \mathrm{AF}\mathbf{C1}) \wedge \mathrm{AG}(\mathbf{T2} \Rightarrow \mathrm{AF}\mathbf{C2})$. This yields Figure 3 (left), in which P_2 cycles repeatedly through neutral, trying, and critical, while P_1 has no transitions at all! Obviously, this repair is

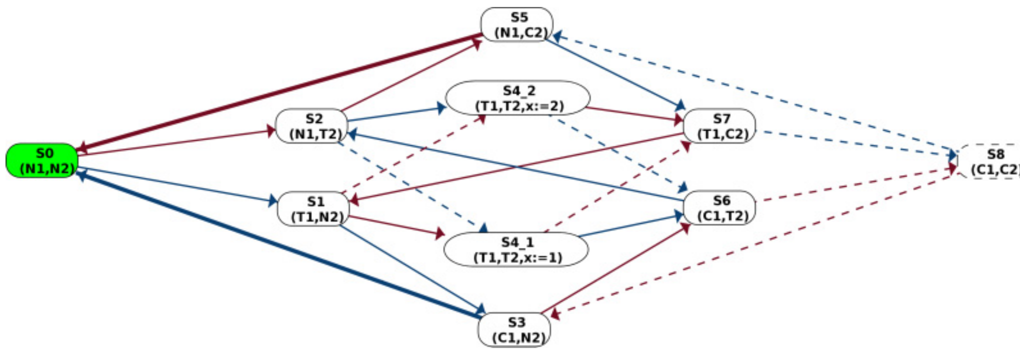

Fig. 4. Mutex: safe and live solution.

much too restrictive. We notice that some transitions where a process requests the critical section, by moving from neutral to trying (e.g., **S0** to **S1**) 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 $repair(M, \eta)$, 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!

We observe that AG(**T1** ⇒ AF**C1**) was previously violated by the cycle **S1** → **S4** → **S7** → **S1**, which is now broken. By symmetric reasoning, AG(**T2** ⇒ AF**C2**) was previously violated by the cycle **S2** → **S4** → **S6** → **S2**. Inspection shows that we cannot break both cycles at once: **S1** → **S4** and **S2** → **S4** are now non-deletable. Removing **S7** → **S1**, **S6** → **S2**, leaves **S7**, **S6**, respectively, without an outgoing transition, so M' is no longer total. Hence, we have to remove both **S4** → **S7** and **S4** → **S6**, which leaves **S4** without an outgoing transition. Hence, no repair is possible. This manual analysis is confirmed by an unsat-core analysis of this example, which gives an unsat-core consisting of the states **S0**, **S4**, **S6**, and **S7**, together with their incident transitions, and two other transitions (**S1** → **S3**) and (**S2** → **S5**).

We can weaken the specification by dropping, say AG(**T2** ⇒ AF**C2**), so we repair w.r.t. AG(¬(**C1** ∧ **C2**)) ∧ AG(**T1** ⇒ AF**C1**) and obtain Figure 3 (right) with only P_1 live. A better fix is to *add* a shared variable x (the usual "turn" variable) to record priority among P_1 and P_2, effectively *splitting* **S4** into two states. After doing so and repairing w.r.t. AG(¬(**C1** ∧ **C2**)) ∧ AG(**T1** ⇒ AF**C1**) ∧ AG(**T2** ⇒ AF**C2**), we obtain Figure 4.

Adding a variable entails *enlarging the domain of the state space*, which differs from adding a state over the *existing* domain, which some repair methods currently do [13, 32]. Extending our method and tool to automate such repair is a topic for future work. We will also investigate heuristics for weakening a specification when no repair is possible. Eliminating a conjunct, as previously, is one possible heuristic. Adding a disjunct is another but requires that the disjunct be formulated.

5 REPAIR USING ABSTRACTIONS

We now extend the repair method to use abstractions, that is, repair a structure abstracted from M and then concretize the resulting repair to obtain a repair of M. The purpose of abstractions is twofold. First, they reduce the size of M and so reduce the length of $repair(M, \eta)$, which is quadratic in $|M|$, and hence repairing an abstract structure significantly increases the size of structures that can be handled. Second, they "focus" the attention of the repair algorithm, which in practice produces "better" repairs. Repair(M, η) nondeterministically chooses a repair from those available, according to the valuation returned by the SAT-solver. This repair may be undesirable, as it may

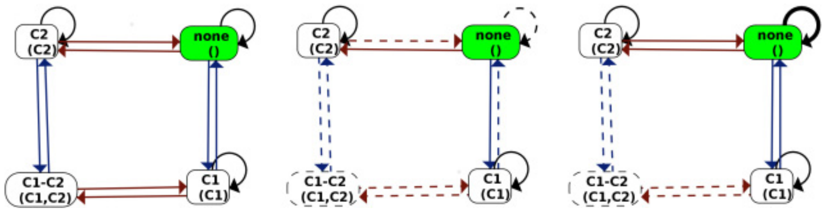

Fig. 5. Abstraction and repair by label.

result in restrictive behavior, for example, Figure 2 (right). By repairing an abstract structure that tracks only the values of **C1**, **C2**, (see Figure 2), we obtain a better repair, which removes only the transitions that enter state **S8**.

Eshmun implements two abstractions: *abstraction by label* preserves the values of all $p \in AP_\eta$, where AP_η denotes the set of atomic propositions that occur in η, and *abstraction by formula* preserves the values of all $\varphi \in AS_\eta$, where AS_η is the set of propositional subformulae of η, that is, it is basically predicate abstraction [23]. These abstractions are defined by equivalence relations [15] over the set of states S, where equivalence is according to the valuation of atomic propositions (abstraction by label) and subformulae (abstraction by formula), respectively:

(1) abstraction by label: $s \equiv_p t$ iff $L(s) \cap AP_\eta = L(t) \cap AP_\eta$,
(2) abstraction by formula: $s \equiv_f t$ iff $(\forall \varphi \in AS_\eta : s \models \varphi \text{ iff } t \models \varphi)$.

Let $\equiv$ be an equivalence relation over S, and let $[s]$ be the equivalence class of s in $\equiv$. For $M = (S_0, S, R, L, AP)$, the corresponding abstract structure $\overline{M} = (\overline{S}_0, \overline{S}, \overline{R}, \overline{L}, AP_\eta)$ is obtained by taking the quotient of M by $\equiv$, as usual, that is, $\overline{S} = \{[s] \mid s \in S\}$, $\overline{S_0} = \{[s_0] \mid s_0 \in S_0\}$, and $\overline{R} = \{([s], [t]) \mid (s, t) \in R\}$. For abstraction by label, $\overline{L} : \overline{S} \to 2^{AP_\eta}$ is given by $\overline{L}([s]) = L(s) \cap AP_\eta$. For abstraction by formula, $\overline{L}$ assigns truth values to the formulae being abstracted, that is, $\overline{L} : \overline{S} \longrightarrow 2^{AS_\eta}$ is given by $\overline{L}([s]) = \{\varphi \mid \varphi \in AS_\eta \wedge s \models \varphi\}$. That is, abstract states are labeled by the set of formulae in AS_η that hold in the corresponding concrete states. We do not need values for atomic propositions, since the values of the formulae in AS_η determine the values of all temporal formulae. Hence, Definition 4.1 is modified so that formulae in AS_η play the role of atomic propositions (Clauses 5, 6 of Definition 4.1).

Abstract repair does not guarantee concrete repair. When we concretize the repair of $\overline{M}$, we obtain a *possible* repair of M, which we verify by model checking. We concretize as follows. The abstraction algorithm maintains a 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.

Consider structure M in Figure 2, with $\eta = \mathrm{AG}(\neg(\mathbf{C1} \wedge \mathbf{C2}))$, and abstraction by label, that is, equivalent states agree on both **C1** and **C2**. The equivalence classes of $\equiv_p$ are: **none** = $\{\mathbf{S0}, \mathbf{S1}, \mathbf{S2}, \mathbf{S4}\}$, **C1** = $\{\mathbf{S3}, \mathbf{S6}\}$, **C2** = $\{\mathbf{S5}, \mathbf{S7}\}$, **C1_C2** = $\{\mathbf{S8}\}$. Figure 5 (left) shows the resulting abstract structure $\overline{M}$, which has four states, corresponding to these equivalence classes. Figure 5 (middle) shows the first repair of $\overline{M}$ (recall, we delete the dashed transitions), which does not allow return to state **none**. Since a process must be able to exit its critical section, we checked **retain** for transitions **C1** $\to$ **none**, **C2** $\to$ **none**, thereby obtaining the repair in Figure 5 (right). Now consider abstraction by the subformula $\mathbf{C1} \wedge \mathbf{C2}$. The equivalence classes of $\equiv_f$ are **none** = $\{\mathbf{S0}–\mathbf{S7}\}$ and **C1_C2** = $\{\mathbf{S8}\}$. Figure 6 (left) shows the resulting abstract structure $\overline{M}$, which has two states, corresponding to these equivalence classes, and Figure 6 (right) shows the repair. Figure 7 shows

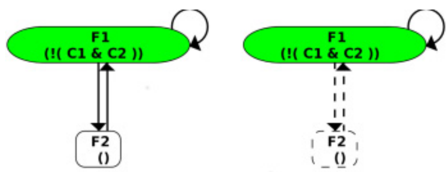

Fig. 6. Abstraction and repair by subformulae.

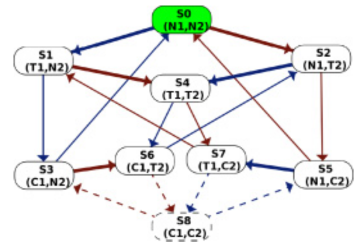

Fig. 7. Concretized repair.

the concretized repair of the original structure. In this case, both abstractions give the same concrete repair, which is good in that it does not prevent either process from making a request (i.e., moving from neutral to trying) at any time.

6 REPAIR OF CONCURRENT KRIPKE STRUCTURES AND CONCURRENT PROGRAMS

We consider finite-state shared-memory concurrent programs $P = (St_P, P_1 \parallel \cdots \parallel P_K)$ 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 $SH = \{x_1, \ldots, x_m\}$ of shared variables. These can be read/written by all processes, and have values from finite domains (in Eshmun all shared variables are Boolean, that is, atomic propositions that are not local to any process). We define the set of all atomic propositions $AP = AP_1 \cup \cdots \cup AP_K$.

Each process P_i is a *synchronization skeleton* [21], 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* [19] $B_i \rightarrow A_i$ consisting of a guard B_i and corresponding action A_i. We write such an arc as the tuple $(s_i, B_i \rightarrow A_i, s'_i)$, where s_i is the source node and s'_i is the target node. Each node must have at least one outgoing arc, that is, a synchronization skeleton contains no "dead ends." The read/write restrictions on atomic propositions are enforced by the syntax of processes: in an arc $(s_i, B_i \rightarrow A_i, s'_i)$ of P_i, guard B_i is a Boolean formula over $AP - AP_i$ and SH, and action A_i is a piece of terminating pseudocode that updates the shared variables in SH. Let S_i denote the set of local states of P_i. There is a mapping $V_i : S_i \rightarrow (AP_i \rightarrow \{tt, \mathit{ff}\})$ 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 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 $B_i \rightarrow A_i$. 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 previously, 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

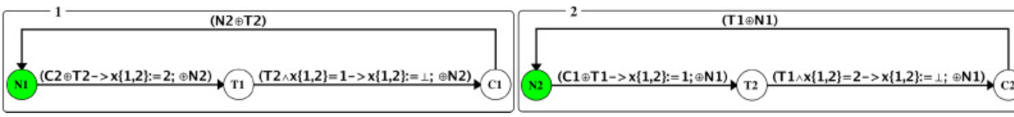

Fig. 8. Concurrent program extracted from Figure 4.

changed by a transition. Shared variables may also be present. The structures of Figure 2 are multiprocess Kripke structures, since the atomic propositions are partitioned into $\{\mathbf{N1}, \mathbf{T1}, \mathbf{C1}\}$ (which are modified by P_1) and $\{\mathbf{N2}, \mathbf{T2}, \mathbf{C2}\}$ (which are modified by P_2). The semantics of a concurrent program $P = (St_P, P_1 \parallel \cdots \parallel P_K)$ is given by 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 [21]. 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 $x_1, \ldots, x_m := v'_1, \ldots, v'_m$, that is, it assigns v'_ℓ to x_ℓ, $\ell = 1, \ldots, m$. For example, the concurrent program given in Figure 8 is extracted from the multiprocess Kripke structure of Figure 4. Each local state is shown labeled with the atomic propositions that it evaluates to true. Take for example the transition in Figure 4 from **S2** to **S4_2**: this is a transition by P_1 from **N1** to **T1**, which can be taken only when **T2** holds. It contributes an arc $(\mathbf{N1}, \mathbf{T2} \to x := 2, \mathbf{T1})$ to P_1. Likewise, the transition from **S0** to **S1** contributes $(\mathbf{N1}, \mathbf{N2} \to \epsilon, \mathbf{T1})$, where ϵ is the empty action, which changes nothing, 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 $(\mathbf{N1}, (\mathbf{N2} \to \epsilon) \oplus (\mathbf{T2} \to x := 2) \oplus (\mathbf{C2} \to \epsilon), \mathbf{T1})$. The $\oplus$ operator [8] 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, Eshmun replaces $B \to \epsilon$ by just B, that is, it omits empty actions, so the previous arc appears as $(\mathbf{N1}, \mathbf{N2} \oplus (\mathbf{T2} \to x := 2) \oplus \mathbf{C2}, \mathbf{T1})$.

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 by using the *pairwise composition* approach of Reference [8], Attie [1999, 2016b], which the next three paragraphs summarize.

For each pair of processes P_i, P_j that interact directly, we provide as inputs 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 variables 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 three 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

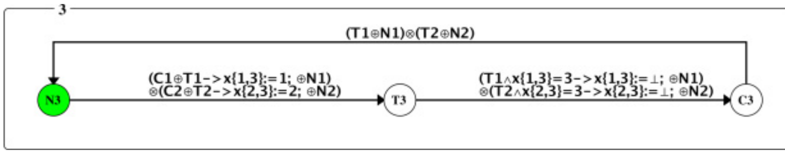

Fig. 9. Concurrent program for three process live mutual exclusion (P_3 only shown).

to Figure 8, modulo index substitution. Then, the 3 process solution $P = P_1 \,\|\, P_2 \,\|\, P_3$ is given in Figure 9, where only P_3 is shown, P_1 and P_2 being isomorphic to P_3 modulo index substitution. P_3 results from composing P_3^1 and P_3^2. For example, the arc from **N3** to **T3** is the composition (using the $\otimes$ operator [8]) of "corresponding" arcs in P_3^1 and P_3^2. The arc from **N3** to **T3** in P_3^1, namely $(\mathbf{N3}, \mathbf{N1} \oplus \mathbf{C1} \oplus (\mathbf{T1} \rightarrow x_{13} := 1), \mathbf{T3})$, and the arc from **N3** to **T3** in P_3^2, namely $(\mathbf{N3}, \mathbf{N2} \oplus \mathbf{C2} \oplus (\mathbf{T2} \rightarrow x_{12} := 1), \mathbf{T3})$ are corresponding arcs, since they have the same source and target local states, namely **N3** and **T3**. Their composition is given by the arc $(\mathbf{N3}, (\mathbf{N1} \oplus \mathbf{C1} \oplus (\mathbf{T1} \rightarrow x_{13} := 1)) \otimes (\mathbf{N2} \oplus \mathbf{C2} \oplus (\mathbf{T2} \rightarrow x_{12} := 1)), \mathbf{T3})$. The meaning of $(B_i \rightarrow A_i) \otimes (B_i' \rightarrow A_i')$ 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 $(\mathbf{N3}, (\mathbf{N1} \oplus \mathbf{C1} \oplus (\mathbf{T1} \rightarrow x_{13} := 1)) \otimes (\mathbf{N2} \oplus \mathbf{C2} \oplus (\mathbf{T2} \rightarrow x_{12} := 1)), \mathbf{T3})$, is that (1) in moving from **N3** to **T3**, P_3 must assign 1 to x_{13} if P_1 is in trying, otherwise (P_1 is is neutral or critical) P_3 need make no such assignment, and (2) likewise w.r.t. P_2.

Let $I(i) = \{j \mid (i,j) \in I\}$, so that $I(i)$ is the set of processes that P_i interacts directly with. For each $j \in I(i)$, we create and repair pair-structure M_{ij} w.r.t. a *pair-specification* η_{ij}. From M_{ij}, we extract pair-program $P_i^j \,\|\, P_j^i$. 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, [4, Definition 15, pairwise synthesis]:

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

iff

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

Hence, the inputs to our method are the pair-specifications η_{ij}, the pair-structures M_{ij}, and the interaction relation I.

Recall that two arcs in P_i^j, P_i^k correspond iff they have the same source and target nodes. An 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*, that is, the result of removing all arc labels is the same in both cases. 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, we define a Boolean formula over transition variables $E_{s,t}$. 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 $grCon(i, j, k, s_i, t_i,) =$

$$\left(E[s^1_{ij}, t^1_{ij}] \vee \cdots \vee E[s^n_{ij}, t^n_{ij}]\right) \equiv \left(E[s^1_{ik}, t^1_{ik}] \vee \cdots \vee E[s^m_{ik}, t^m_{ik}]\right).$$

For clarity of sub/superscripts, we use $E[s, t]$ rather than $E_{s,t}$. Let $I(i) = \{j_1, \ldots, j_n\}$, and let $grCon(i, s_i, t_i,) = grCon(i, j_1, j_2, s_i, t_i,) \wedge \cdots \wedge grCon(i, j_{n-1}, j_n, s_i, t_i,)$, so that we enforce consistency among all pair-machines that involve P_i, since $\equiv$ is transitive. Now define $grCon(i) = \bigwedge_{s_i, t_i} grCon(i, s_i, t_i,)$, where s_i, t_i range over all pairs of local states of P_i such that some P_i-transition in some pair-machine moves P_i from s_i to t_i. Finally, let $grCon = \bigwedge_{i \in \{1,\ldots,K\}} grCon(i)$. Then, our overall repair formula is

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

Eshmun computes this repair formula from the pair-structures M_{ij} and their respective specifications η_{ij}. 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 previous. By the large model theorems of Attie and Emerson [8] and Attie [4, 7], P satisfies $\bigwedge_{(i,j) \in I} \eta_{ij}$. We use $|\ldots|$ for the size of formulae (length) and Kripke structures (number of states + number of transitions). Let $n_i = |I(i)|$, that is, the number of pairs that P_i is involved in. Then $|grCon|$ is linear in $\Sigma_{i \in \{1,\ldots,K\}} n_i$, and linear in $|M_{pair}|$, the size of the largest pair-structure M_{ij}, since each transition Boolean $E[s^n_{ij}, t^n_{ij}]$ is mentioned at most twice. Now $\Sigma_{i \in \{1,\ldots,K\}} n_i = 2|I|$, and so $|grCon| = O(|I| \times |M_{pair}|)$. By Proposition 4.3, $|repair(M_{ij}, \eta_{ij})|$ is quadratic in $|M_{ij}|$, and linear in $|\eta_{ij}|$. There are $|I|$ such formulae. Let $|\eta_{pair}|$ be the length of the largest pair-specification η_{ij}, so that $|(\bigwedge_{(i,j) \in I} repair(M_{ij}, \eta_{ij}))| = O(|I| \times |M_{pair}|^2 \times |\eta_{pair}|)$. Hence:

Proposition 6.1. *$grCon \wedge (\bigwedge_{(i,j) \in I} repair(M_{ij}, \eta_{ij}))$ has length in $O(|I| \times |M_{pair}|^2 \times |\eta_{pair}|)$.*

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

To summarize, we repair a concurrent program as follows: (1) for each $(i, j) \in I$, input either a pair-structure M_{ij} or a pair-program $P^j_i \parallel P^i_j$, (2) generate the pair-structure (if input is a pair-program) and repair the pair-structures, and (3) extract a correct concurrent program from the pair structures.

6.1 Example: Eventually Serializable Data Service

The eventually serializable data service (ESDS) of [22] and [27] 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, that is, not reflecting the effects of other operations that 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," that is, its ordering is fixed w.r.t. all other operations. Clients may require an operation to be *strict*, that is, 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 [4].

We use Eshmun to repair a simple instance of ESDS with one strict operation x, one client **Cx** that issues x, 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 **R1x** $\|$ **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 previous sequences. We then repair w.r.t. these pair-specifications:

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

- AG(**WT_R1x** ⇒ **WT_Cx**): x is not received by **R1x** before it is submitted by **Cx**
- A[**WT_R1x** R (¬(**WT_Cx** ∧ EG(¬**WT_R1x**)))] if x is submitted by **Cx** then it is received by **R1x**
- AG(**WT_Cx** ⇒ AF **DN_Cx**): if **Cx** submits x then it eventually receives a result
- AG(**DN_Cx** ⇒ **SNT_R1x**): **Cx** does not receive a result before it is sent by **R1x**

Pair-specification for **R1x** $\|$ **R2x**, where $x \in O$, $x.strict$, **R1x** $= replica(x)$

- AG(**SNT_R1x** ⇒ **ST_R2x**): the result of a strict operation is not sent to the client until it is stable at all replicas

CSC constraints, pair-specification for **R1x** $\|$ **R2y**, where $x \in O$, $y \in x.prev$, **R1x** $= replica(x)$, **R2y** $= replica(y)$

- AG(**DN_R1x** ⇒ **DN_R2y**): operation y in $x.prev$ is performed before x is

The repaired pair-structures are given in Figures 10, 11, and 12. Figure 13 gives a concurrent program P that is extracted from the repaired pair-structures as discussed previously. By the large model theorem [4, 7, 8], P satisfies all the previous pair-specifications. Repairing this example took 0.61s on our Intel Xeon 3.3GHz PC.

7 REPAIR OF HIERARCHICAL KRIPKE STRUCTURES

We now extend our repair method to hierarchical Kripke structures. As given by Alur and Yannakakis [2], a hierarchical Kripke structure K over a set AP of atomic propositions is a tuple $K_1, \ldots, K_n$ of structures, where each K_i has the following components:

(1) A finite set N_i of nodes.
(2) A finite set B_i of boxes (or supernodes). The sets N_i and B_i are all pairwise disjoint.

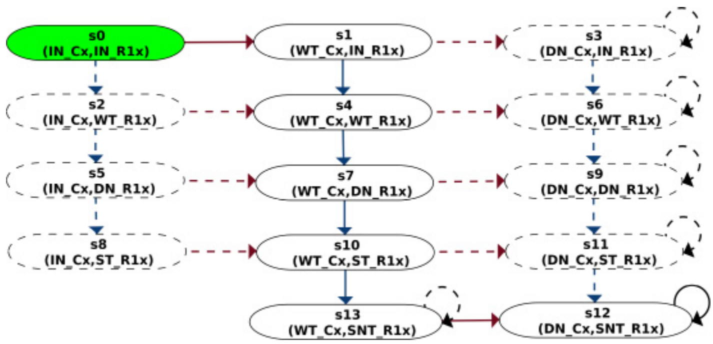

Fig. 10. Repaired pair-structure for **Cx** || **R1x**.

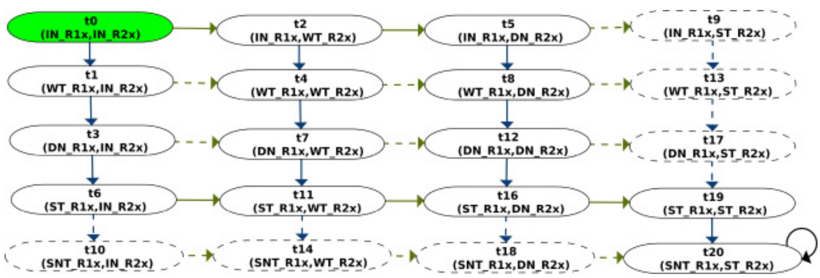

Fig. 11. Repaired pair-structure for **R1x** || **R2x**.

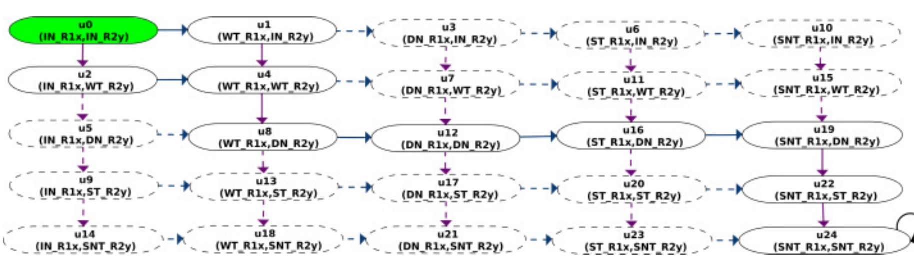

Fig. 12. Repaired pair-structure for **R1x** || **R2y**.

(3) An initial node $start_i \in N_i$.
(4) A subset O_i of N_i, called exit nodes.
(5) A labeling function $X_i : N_i \longrightarrow 2^{AP}$ that labels each node with a subset of AP.
(6) An indexing function $Y_i : B_i \longrightarrow \{i+1 \dots n\}$ 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 repeat application of the result for one box/one occurrence to obtain results for several kinds

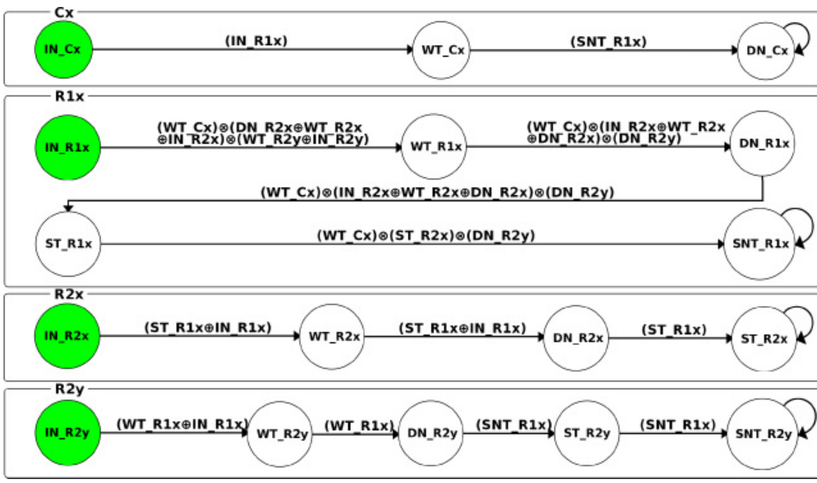

Fig. 13. ESDS Program.

and occurrences. Let $start_M$, $start_B$ denote the initial states of M, B, respectively, and O_M, O_B the sets of exit states of M, B, respectively. Let $M - B$ denote the part of M that is not B. Wlog, we assume that all states of M are reachable from $start_M$, and that, from every state of M, some exit state of M is reachable. We also assume that M is total. Likewise, for B. The reachability problem for hierarchical Kripke structures is solvable by a polynomial time depth-first search [2], so we can check this efficiently, and remove unreachable states. To avoid state-explosion, we repair level-by-level. We (inductively) repair B w.r.t. a specification η_B that describes the behavior of B. We then repair M w.r.t. a "coupling specification" φ, which describes how M uses B. Finally, we show that $\eta_B \wedge \varphi \Rightarrow \eta$ is a CTL validity, from which we infer $M \models \eta$.

To show soundness, we use weak (i.e., stuttering) forward simulations [10, 24], and so our results are restricted to ACTL-X, the universal fragment of CTL without nexttime [24].

Definition 7.1 (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'' iff, for all s, s', $H(s, s')$ implies: (1) $L(s) \cap AP'' = L(s') \cap AP''$, and (2) for every fullpath π from s in M, there exists a fullpath π' from s' in M' and partitions $B_1 B_2 \ldots$ of π, $B'_1 B'_2 \ldots$ 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.

Proposition 7.2. *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 [24] 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 implies 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.

7.1 Verification of the Box Specification

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

Definition 7.3 (B_M, the M-situated version of B). Construct B_M from B as follows. Include all the states and transitions of B. Add two "interface" states pre_B and $post_B$. Let pre_B be the single initial state of B_M. Add a transition from pre_B to $start_B$, the initial state of B, and a transition from every $s \in O_B$ (i.e., every exit state of B) to $post_B$. Add a self loop on $post_B$. If, in M, some fullpath from $start_M$ does not enter B, then add a transition from pre_B to $post_B$. If, in M, there is a path in $M - B$ (i.e., the states and transitions of M that are not in B) from some state in O_B to $start_B$, then add a transition from $post_B$ to pre_B.

Let AP_B be the set of atomic propositions corresponding to states in B, including $start_B$ and all states in O_B. Definition 7.3 ensures that B_M contains paths that simulate (w.r.t. AP_B) any path in M.

Proposition 7.4. *$M \leq_{AP_B} B_M$*

Proof. Define a weak forward simulation f from M to B_M as follows. f relates $start_M$ to pre_B. A state of M that is in B is related by f to "itself". A state s of M that is not in B is mapped as follows: If $start_B$ is reachable from s, then f relates s to pre_B. If $start_B$ is not reachable from s, then f relates s to $post_B$.

We verify that f satisfies Definition 7.1. Let π be an arbitrary fullpath in M that starts in $start_M$, and let π' be the simulating fullpath in B_M. If π does not enter B, then $\pi' = pre_B \rightarrow post_B^{\omega}$. If π enters B some number $n > 0$ times, and eventually leaves B forever, then $\pi' = (pre_B \rightarrow B \rightarrow post_B)^{n-1} \rightarrow pre_B \rightarrow B \rightarrow post_B^{\omega}$. If π enters B some number $n > 0$ times, and eventually remains in B forever, then $\pi' = (pre_B \rightarrow B \rightarrow post_B)^{n-1} \rightarrow pre_B \rightarrow B$. If π enters B an infinite number of times, then $\pi' = (pre_B \rightarrow B \rightarrow post_B)^{\omega}$. It is straightforward to check, from Definition 7.3, that π' exists in all cases, and that the block-correspondence required by Definition 7.1 also exists. The occurrence of B in "path expression" $pre_B \rightarrow B \rightarrow post_B$ means that we "take a path through B." □

Corollary 7.5. *For any ACTL-X formula f over AP_B, if $B_M \models f$ then $M \models f$.*

Proof. Follows immediately from Proposition 7.2 and Proposition 7.4. □

7.2 Verification of the Coupling Specification

Definition 7.6 (Abstract box). Construct the abstraction B_A of B as follows. The states of B_A are $\{start_B, int_B\} \cup O_B$, that is, the start state, all exit states, and a new state int_B, which represents the interior of B. int_B has the empty propositional labeling. Add the set of transitions $\{(start_B, int_B)\} \cup \{(int_B, s) \mid s \in O_B\}$ to B_A, that is, 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 also add the transition (int_B, int_B), that is, a self-loop on int_B, which models the possibility of remaining inside B forever.

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 7.7. *$M \leq_{AP_C} M_A$.*

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

Proof. Follows immediately from Proposition 7.2 and Proposition 7.7. □

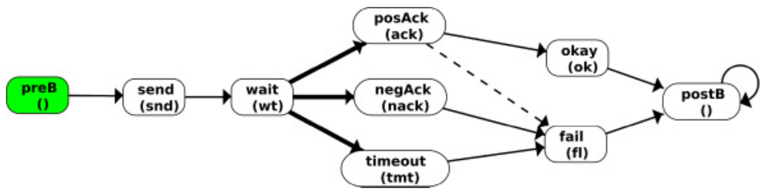

Fig. 14. Repaired box B_M for a single phone-call.

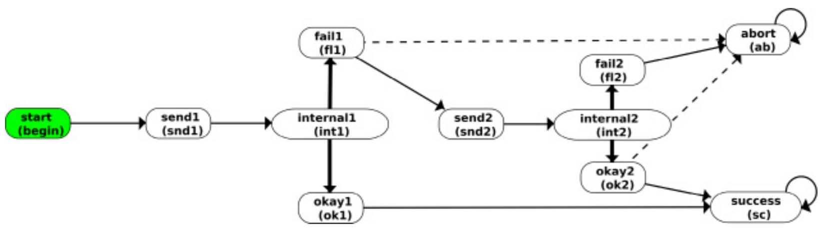

Fig. 15. Repaired structure M for phone-call example.

7.3 Hierarchical Repair

We summarize our hierarchical repair method for M w.r.t. η: (1) repair B_M w.r.t. η_B, and conclude by Corollary 7.5 that $M \models \eta_B$, (2) repair M_A w.r.t. φ, and conclude by Corollary 7.8 that $M \models \varphi$, and (3) show $\models \eta_B \wedge \varphi \Rightarrow \eta$, and so conclude $M \models \eta$.

THEOREM 7.9. *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 Corollary 7.5, we have $M \models \eta_B$. From $M_A \models \varphi$ and Corollary 7.8, we have $M \models \varphi$. From $M \models \eta_B$, $M \models \varphi$, and $\models \eta_B \wedge \varphi \Rightarrow \eta$, we have $M \models \eta$. □

7.4 Example: Phone System

Consider the phone call example from Alur and Yannakakis [2]. Figure 14 shows B_M, 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, that is, state **fail**, and **posAck** leads to placement of the call, that is, state **ok**. Figure 15 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, then the system should proceed to the success final state. If the first attempt fails, then the system should proceed to the start state of the second attempt. If the second attempt succeeds, then the system should proceed to the success final state, while if the second attempt fails, the system should proceed to the abort final state. The relevant formulae are:

(1) specification formula η for M: $\mathsf{AG}((ack_1 \vee ack_2) \Rightarrow \mathsf{AF}(suc))$, that is, if either attempt receives a positive ack, then eventually enter success state.
(2) specification formula η_B for B: $\mathsf{AG}(ack \Rightarrow \mathsf{AF}(ok))$, that is, 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 Figure 14.
(3) "coupling" formula φ: $\mathsf{AG}((ok_1 \vee ok_2) \Rightarrow \mathsf{AF}sc) \wedge \mathsf{AG}(fl1 \Rightarrow \mathsf{AF}(snd2))$, that is, 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

Name	N	Basic repair	Abst. by label	Abst. by sub frm.
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

Fig. 16. Times (in seconds) taken to repair the given structures.

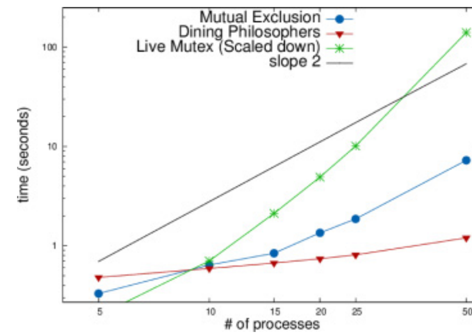

Fig. 17. Repair time for given programs.

#	5	10	15	20	25	50
Mutex	0.33	0.64	0.84	1.35	1.86	7.25
Mx. Lb.[2]	0.051	0.24	0.82	1.67	2.04	3.85
Mx. Frm.[3]	0.037	0.078	0.16	0.30	0.48	2.16
Live Mx.	1.98	7.03	21.17	49.2	101	1404
D. Ph.	0.48	0.59	0.67	0.74	0.81	1.20

2: abstracted by label, 3: abstracted by sub formula.

Fig. 18. Times (in seconds) to repair the given pairwise concurrent structures.

of B_A: **send** $\rightarrow$ **int**, **int** $\rightarrow$ **ok**, **int** $\rightarrow$ **fl**. The transitions from **fail1** to **abort** and **okay2** to **abort** are deleted, as shown in Figure 15.

Eshmun checks the validity of $\eta_B \wedge \varphi \Rightarrow \eta$ by using the CTL decision procedure of Reference [21]: we check satisfiability of $\neg(\eta_B \wedge \varphi \Rightarrow \eta)$. By Theorem 7.9, we conclude that $M \models \eta$.

Using our 3.3GHz Intel Xeon PC, we achieved linear increase in repair time with the number of attempted phone calls:

#Calls	1	2	3	4	5
Time	0.1	0.17	0.25	0.34	0.4

8 EXPERIMENTS AND BENCHMARKS

Figure 16 gives experimental results for repairing mutual exclusion (w.r.t. safety) and also barrier synchronization (two processes pass through a sequence of "barriers" and cannot be out of sync by more than one barrier). The structures used were generated by a Python program. N is the number of processes in mutual exclusion and the number of barriers in barrier synchronization. Figure 18 gives experimental results for various (pairwise) concurrent Kripke structures for safe mutual exclusion (Figure 7), safe and live mutual exclusion (Figure 4), and dining philosophers. We used a linux PC with openJDK 7, a 3.3GHz Intel Xeon CPU, and 8GB of RAM. For safe mutex and dining philosophers, Figure 18 agrees closely with Proposition 6.1. For mutex, 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. For live mutex, Eshmun started exhausting RAM and swapping frequently, which accounts for the greater than quadratic increase in run time (still less than cubic, though). The increased RAM usage is because the CTL specification η is longer when liveness is added, which makes the repair formula larger. For safe mutex, the un-repaired global structure (product of N processes) contains 3^N states and $3^N N$ transitions. Pairwise representation reduces this to $3^2 \frac{N(N-1)}{2}$ states and $3^2 N(N-1)$ transitions. With 50 Processes, we had 11,025 and 22,050 states and transitions, with 3,675 and 13,475 deleted. The formula had 917,550 and 2,173,825 clauses and literals.

9 RELATED WORK

Attie and Emerson [5, 6] used state and transition deletion to repair Kripke structures in the context of atomicity refinement: a concurrent program is refined naively (e.g., by replacing a test and set by the test, followed non-atomically by the set). In general, this introduces new computations (due to new interleavings) that violate the specification. These are removed by deleting states and transitions.

Several articles [18, 25, 29, 31] generate counterexamples from failed model checking runs, but none give a method for repairing the counterexamples. Biere et al. [9] proposed the idea of generating a propositional formula from a model checking problem, such that the formula is satisfiable iff the specification 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.

Buccafurri et al. [11] posed the repair problem for CTL and solved it using abductive reasoning. They generate repair suggestions that are verified by model checking, one at a time. In contrast, we fix all faults at once. Jobstmann et al. [26] and Staber et al. [30] present game-based repair methods for programs and circuits, but their method is complete (i.e., if a repair exists, then find a repair) only for invariants, and not for a full temporal logic (e.g., CTL, LTL). Zhang and Ding [32] present a "model update" method based on five operations: add a transition, remove a transition, change the propositional labeling of a state, add a state, and remove an isolated state. They 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 $|M|$, but appears to be highly nondeterministic, with several choices of actions (e.g., "do one of (a), (b), and (c)"). They do not discuss how this nondeterminism is resolved. Carrillo and Rosenblueth [12] presents a syntax-directed repair method that uses "protections" to deal with conjunction, and a representation of Kripke structures as "tables." Chatzieleftheriou et al. [13] repair abstract structures, using Kripke modal transition systems, three-valued CTL semantics, and these operations: add/remove a may/must transition, change the propositional label of a state, and add/remove a state. They aim to minimize the number of changes made to the concrete structure. Their repair algorithm is recursive CTL-syntax-directed.

10 CONCLUSIONS AND FUTURE WORK

We presented a method for repairing a Kripke structure with respect to a CTL formula η by deleting transitions and states, and we implemented it as an interactive graphical tool, Eshmun. This enables gradual design and construction of Kripke structures, aided by immediate semantic feedback. Our method handles concurrent and hierarchical Kripke structures, which are exponentially more succinct than normal Kripke structures, without incurring state-explosion. For (pairwise) concurrent structures the size of the repair formula is quadratic in the number of processes, so 50-process mutual exclusion, with about 3^{50} states, is repaired in under 8s. Moreover, the results of Attie [3] show that using pairwise form entails no loss of expressiveness: any finite state concurrent program can be written in pairwise form. Hierarchical structures are repaired "one level at a time," so the exponential blowup caused by replacing boxes by their definitions is avoided. We also count the number of deleted states and transitions, which allows a user of Eshmun to find the repair with the maximum or minimum number of deletions. Our experimental results validated avoidance of state-explosion. In particular, there were no "difficult" cases that defeated the SAT-solver.

Our repair algorithm cannot add states and transitions. However, given a multiprocess Kripke structure M, it is easy to add in all possible transitions: generate the synchronization skeletons, replace all guards by "true," and then generate a new structure M', which is then repaired. We will also support action-based models of concurrency and deal with alternating-time temporal logic [1]. Finally, we will attempt to repair infinite-state models using abstractions and SMT solvers.

REFERENCES

[1] R. Alur, T. A. Henzinger, and O. Kupferman. 2002. Alternating-time temporal logic. *J. ACM* 49 (2002), 672–713.

[2] R. Alur and M. Yannakakis. 2001. Model checking of hierarchical state machines. *ACM Trans. Program. Lang. Syst.* 23, 3 (2001), 273–303.

[3] P. C. Attie. 2016. Finite-state concurrent programs can be expressed in pairwise normal form. *Theor. Comput. Sci.* 619 (2016), 1–31. DOI: http://dx.doi.org/10.1016/j.tcs.2015.11.032

[4] P. C. Attie. 2016. Synthesis of large dynamic concurrent programs from dynamic specifications. *Formal Methods Syst. Design* 48, 1 (2016), 94–147.

[5] P. C. Attie and E. A. Emerson. 1996. Synthesis of concurrent systems for an atomic read/atomic write model of computation. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC'96).* ACM, New York, NY, 111–120.

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

[7] P. C. Attie. 1999. Synthesis of large concurrent programs via pairwise composition. In *Proceedings of the 10th International Conference on Concurrency Theory (CONCUR'99).* Springer-Verlag, Eindhoven, The Netherlands, 130–145.

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

[9] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. 1999. Symbolic model checking without BDDs. In *Proceedings of the Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99), LNCS No. 1579.* Springer-Verlag, Amsterdam, The Netherlands, 193–207.

[10] M. C. Browne, E. M. Clarke, and O. Grumberg. 1988. Characterizing finite kripke structures in propositional temporal logic. *Theoret. Comput. Sci.* 59 (1988), 115–131.

[11] F. Buccafurri, T. Eiter, G. Gottlob, and N. Leone. 1999. Enhancing model checking in verification by AI techniques. *Artif. Intell.* 112 (1999), 57–104.

[12] M. Carrillo and D. A. Rosenblueth. 2009. A method for CTL model update, representing kripke structures as table systems. *Int. J. Pure Appl. Math.* 52 (2009), 401–431.

[13] G. Chatzieleftheriou, B. Bonakdarpour, S. A. Smolka, and P. Katsaros. 2012. Abstract model repair. In *NASA Formal Methods*, Alwyn E. Goodloe and Suzette Person (Eds.). Lecture Notes in Computer Science, Vol. 7226. Springer, Berlin. 341–355.

[14] E. M. Clarke, E. A. Emerson, and J. Sifakis. 2009. Model checking: Algorithmic verification and debugging. *Commun. ACM* 52, 11 (Nov. 2009), 74–84. DOI: http://dx.doi.org/10.1145/1592761.1592781

[15] E. M. Clarke, O. Grumberg, and D. E. Long. 1994. Model checking and abstraction. *ACM Trans. Program. Lang. Syst.* 16, 5 (Sept. 1994), 1512–1542.

[16] E. M. Clarke and H. Veith. 2003. Counterexamples revisited: Principles, algorithms, applications. In *Verification: Theory and Practice.* Springer, 208–224.

[17] E. M. Clarke, E. A. Emerson, and P. Sistla. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. *Trans. Program. Lang. Syst.* 8, 2 (1986), 244–263.

[18] E. M. Clarke, O. Grumberg, K. L. McMillan, and X. Zhao. 1995. Efficient generation of counterexamples and witnesses in symbolic model checking. In *Proceedings of the Design Automation Conference.* ACM Press, New York, NY, 427–432.

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

[20] E. A. Emerson. 1990. Temporal and modal logic. *Handbook of Theoretical Computer Science B* (1990), 997–1072.

[21] E. A. Emerson and E. M. Clarke. 1982. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program.* 2, 3 (1982), 241–266.

[22] A. Fekete, D. Gupta, V. Luchango, N. Lynch, and A. Shvartsman. 1999. Eventually-serializable data services. *Theor. Comput. Sci.* 220 (1999), 113–156.

[23] S. Graf and H. Saidi. 1997. Construction of abstract state graphs with PVS. In *Proceedings of the Conference on Computer Aided Verification (CAV'97)*. LNCS, Vol. 1254. Springer, London, UK, 72–83.
[24] O. Grumberg and D. E. Long. 1994. Model checking and modular verification. *Trans. Program. Lang. Syst.* 16, 3 (1994), 843–871.
[25] R. Hojati, R. K. Brayton, and R. P. Kurshan. 1993. BDD-based debugging of designs using language containment and fair CTL. In *Proceedings of the Conference on Computer Aided Verification (CAV'93)*. Springer-Verlag, London, UK, 41–58. Springer LNCS No. 697.
[26] B. Jobstmann, A. Griesmayer, and R. Bloem. 2005. Program repair as a game. In *Proceedings of the Conference on Computer Aided Verification (CAV'05)*. Springer-Verlag, Berlin, Heidelberg, 226–238.
[27] R. Ladin, B. Liskov, L. Shrira, and S. Ghemawat. 1992. Providing high availability using lazy replication. *ACM Trans. Comput. Syst.* 10, 4 (Nov. 1992), 360–391.
[28] D. Le Berre, A. Parrain, et al. 2010. The sat4j library, release 2.2, system description. *J. Satis. Boolean Model. Comput.* 7 (2010), 59–64.
[29] S. Shoham and O. Grumberg. 2003. A game-based framework for CTL counterexamples and 3-valued abstraction-refinement. In *Proceedings of the Conference on Computer Aided Verification (CAV'03)*. ACM, New York, NY, 275–287.
[30] S. Staber, B. Jobstmann, and R. Bloem. 2005. Finding and fixing faults. In *Proceedings of the Conference on Correct Hardware Design and Verification Methods (CHARME'05)*. Springer-Verlag, Berlin, 35–49. Springer LNCS No. 3725.
[31] C. Stirling and D. Walker. 1991. Local model checking in the modal mu-calculus. *Theor. Comput. Sci.* 89, 1 (1991), 161–177.
[32] Y. Zhang and Y. Ding. 2008. CTL model update for system modifications. *J. Artif. Int. Res.* 31, 1 (Jan. 2008), 113–155.

Received January 2016; revised July 2016; accepted February 2017