Model and Program Repair via SAT Solving[*]

Paul Attie[†§] and Jad Saklawi[†]

[†]Department of Computer Science
American University of Beirut
[§]Center for Advanced Mathematical Sciences
American University of Beirut
{paul.attie, jad.saklawi}@aub.edu.lb

October 30, 2018

Abstract

We consider the following *model repair problem*: given a finite Kripke structure M and a specification formula η in some modal or temporal logic, determine if M contains a substructure M' (with the same initial state) that satisfies η. Thus, M can be "repaired" to satisfy the specification η by deleting some transitions.

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

We extend our method to repair finite-state shared memory concurrent programs, to solve the discrete event supervisory control problem [18, 19], to check for the existence of symmettric solutions [12], and to accomodate any boolean constraint on the existence of states and transitions in the repaired model.

Finally, we show that model repair is NP-complete for CTL, and logics with polynomial model checking algorithms to which CTL can be reduced in polynomial time. A notable example of such a logic is Alternating-Time Temporal Logic (ATL).

1

1 Introduction and Motivation

Counterexample generation in model checking produces an example behavior that violates the formula being checked, and so facilitates debugging the model. However, there could be many counterexamples, and they may have to be dealt with by making different fixes manually, thus increasing debugging effort. In this paper we deal with all counterexamples at once, by "repairing" the model: we present a method for automatically fixing Kripke structures and shared memory concurrent programs with respect to CTL [11] and ATL [1] specifications.

Our contribution. We first present a "subtractive" repair algorithm: fix a Kripke structure only by removing transitions and states (roughly speaking, those transitions and states that "cause" violation of the specification). If the initial state is not deleted, then the resulting structure (or program) satisfies the specification. We show that this algorithm is sound and relatively complete. An advantage of subtractive repair is that it does not introduce new behaviors, and thus any missing (i.e., not part of the formula being repaired against) conjuncts of the specification that are expressible in a universal temporal logic (no existential path quantifier) are still satisfied (if they originally were). Hence we can fix w.r.t. incomplete specifications.

We also extend the subtractive repair method in several directions: to accommodate the addition of states and transitions, to solve the discrete event supervisory control problem [18, 19], to accommodate arbitrary boolean constraints on the existence of states and transitions in the repaired model, and to repair atomic read/write shared memory concurrent programs. Finally, we show that the model repair problem is NP-complete.

Formally, we consider the *model repair problem*: given a Kripke structure M and a CTL or ATL formula η, does there exist a substructure M' of M (obtained by removing transitions and states from M) such that M' satisfies η? In this case, we say that M is *repairable* w.r.t, η, or that a repair exists.

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

Soundness of our repair algorithm means that the resulting M' (if it ex-

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

While our method has a worst case running time exponential in the number of global states, this occurs only if the underlying SAT solver uses exponential time. SAT-solvers have proved to be efficient in practice, as demonstrated by the success of SAT-solver based tools such as Alloy, NuSMV, and Isabelle/HOL. The success of SAT solvers in practice indicates that our method will be applicable to reasonable size models, just as, for example, Alloy [15] is.

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

The use of model checking to generate counterexamples was suggested by Clarke et. al. [9] and Hojati et. al. [14]. [9] presents an algorithm for generating counterexamples for symbolic model checking. [14] presents BDD-based algorithms for generating counterexamples ("error traces") for both language containment and fair CTL model checking. Game-based model checking [23, 20] provides a method for extracting counterexamples from a model checking run. The core idea is a *coloring algorithm* that colors nodes in the model-checking game graph that contribute to violation of the formula being checked.

The idea of generating a propositional formula from a model checking problem was presented in [6]. That paper considers LTL specifications and bounded model checking: given an LTL formula f, a propositional formula is generated that is satisfiable iff f can be verified within a fixed number k of transitions along some path ($\mathsf{E}f$). By setting f to the negation of the required property, counterexamples can be generated. Repair is not discussed.

Some authors [16, 22, 21] have considered algorithms for solving the repair problem: given a program (or circuit), and a specification, how to automatically modify the program (or circuit), so that the specification is satisfied. There appears to be no automatic repair method that is (1) complete (i.e., if a repair exists, then find a repair) for a full temporal logic (e.g., CTL, LTL), and (2) repairs all faults in a single run, i.e., deals implicitly with all counterexamples "at once." For example, Jobstmann et. al. [16]

considers only one repair at a time, and their method is complete only for invariants. In [21], the approach of [16] is extended so that multiple faults are considered at once, but at the price of exponential complexity in the number of faults.

In [7] the repair problem for CTL is considered and solved using adductive reasoning. The method generates repair suggestions that must then be verified by model checking, one at a time. In contrast, we fix all faults at once.

Antoniotti [2] has shown that the related problem of discrete event supervisory control is also NP-complete.

The rest of the paper is as follows. Section 2 provides brief technical preliminaries. Section 3 is the core of the paper: it presents our model repair method for CTL in detail, discuses how the method is modified to handle ATL. Section 4 presents the various extensions discussed above. Section 5 presents several example applications of the method. Section 6 discusses our implementation, including experimental performance data. Section 7 discusses future work and concludes. Appendix A presents a manual simplification of an example repair formula, Appendix B provides proofs for all theorems, and Appendix C provides full technical preliminaries.

2 Preliminaries

We assume basic of knowledge of CTL [10, 11] and ATL [1]. The logic CTL is given by the following grammar:

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

where $p \in \mathcal{AP}$, a set of atomic propositions. The semantics of a CTL formula are given with respect to a Kripke structure $M = (s_0, S, R, L)$ where s_0 is the start state, S is the set of states, $R \subseteq S \times S$ is the transition relation and $L : S \mapsto 2^{\mathcal{AP}}$ is the labeling function. We use $M \models \varphi$ to abbreviate $M, s_0 \models \varphi$. We use the abbreviations $\mathsf{A}[\phi\mathsf{U}\psi]$ for $\neg\mathsf{E}[\neg\varphi\mathsf{V}\neg\psi]$, $\mathsf{E}[\phi\mathsf{U}\psi]$ for $\neg\mathsf{A}[\neg\varphi\mathsf{V}\neg\psi]$, $\mathsf{AF}\varphi$ for $\mathsf{A}[\mathsf{true}\mathsf{U}\varphi]$, $\mathsf{EF}\varphi$ for $\mathsf{E}[\mathsf{true}\mathsf{U}\varphi]$, $\mathsf{AG}\varphi$ for $\mathsf{A}[\mathsf{false}\mathsf{V}\varphi]$, $\mathsf{EG}\varphi$ for $\mathsf{E}[\mathsf{false}\mathsf{V}\varphi]$.

Definition 1 (Formula expansion)**.** *Given a CTL formula φ, its set of subformulae $sub(\varphi)$ is defined as follows:*

- *$sub(p) = p$ where p is* true, false, *or an atomic proposition*

- *$sub(\neg\varphi) = \{\neg\varphi\} \cup sub(\varphi)$*

4

- $sub(\varphi \wedge \psi) = \{\varphi \wedge \psi\} \cup sub(\varphi) \cup sub(\psi)$

- $sub(\varphi \vee \psi) = \{\varphi \vee \psi\} \cup sub(\varphi) \cup sub(\psi)$

- $sub(\mathsf{AX}\varphi) = \{\mathsf{AX}\varphi\} \cup sub(\varphi)$

- $sub(\mathsf{EX}\varphi) = \{\mathsf{EX}\varphi\} \cup sub(\varphi)$

- $sub(\mathsf{A}[\varphi\mathsf{V}\psi]) = \{\mathsf{A}[\varphi\mathsf{V}\psi], \mathsf{AXA}[\varphi\mathsf{V}\psi], \varphi\vee\mathsf{AXA}[\varphi\mathsf{V}\psi], \psi\wedge(\varphi\vee\mathsf{AXA}[\varphi\mathsf{V}\psi])\}\cup$ $sub(\varphi) \cup sub(\psi)$

- $sub(\mathsf{E}[\varphi\mathsf{V}\psi]) = \{\mathsf{E}[\varphi\mathsf{V}\psi], \mathsf{EXE}[\varphi\mathsf{V}\psi], \varphi\vee\mathsf{EXE}[\varphi\mathsf{V}\psi], \psi\wedge(\varphi\vee\mathsf{EXE}[\varphi\mathsf{V}\psi])\}\cup$ $sub(\varphi) \cup sub(\psi)$

The logic ATL is given by the following grammar:

$$\varphi ::= \mathsf{true} \mid \mathsf{false} \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \ll\!A\!\gg X\varphi \mid \ll\!A\!\gg [\varphi\mathsf{V}\varphi]$$

where $p \in \mathcal{AP}$, $A \subseteq \Sigma$. Σ denotes the set of players. $\ll\!A\!\gg \varphi$ holds iff the players in A have a collective strategy to enforce the truth of φ.

3 The Model Repair Problem

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

Definition 2 (Substructure). *Given a Kripke structure $M = (s_0, S, R, L)$ and a structure $M' = (s_0', S', R', L')$ we say that $M \subseteq M'$ iff $S \subseteq S'$, $s_0 = s_0'$, $R' \subseteq R$, and $L' = L \upharpoonright S'$.*

Definition 3 (Repairable). *Given Kripke structure $M = (s_0, S, R, L)$ and a formula η. M is repairable with respect to η if there exists a Kripke structure $M' = (s_0', S', R', L')$ such that M' is total, $M' \subseteq M$, and $M', s_0 \models \eta$.*

Recall that a Kripke strucutre is total iff every state has at least one outgoing transition.

Definition 4 (Model Repair Problem). *Given a Kripke structure $M = (s_0, S, R, L)$, and a formula η, the repair problem is to decide if M is repairable with respect to η.*

The model repair problem is defined for any temporal or modal logic for which the $\models$ relation is defined, e.g μ-calculus , CTL*, CTL, etc. So, for example, we speak of the model repair problem for CTL (CTL model repair for short). An instance of model repair is then the pair (M, φ).

3.1 Complexity of the Model Repair Problem

Theorem 1. *The model repair problem for CTL is NP-complete.*

Corollary 1. *Let L be any temporal logic interpreted in Kripke structures such that (1) model checking for L is in polynomial time, and (2) there exists a polynomial time reduction from CTL model checking to L model checking. Then the model repair problem for L is NP-complete.*

An immediate consequence is that model repair for alternating-time temporal logic (ATL) is NP-complete.

3.2 CTL Model Repair using SAT solvers

Given an instance of model repair (M, η), where $M = (s_0, S, R, L)$ and η is a CTL formula, we define a propositional formula $repair(M, \eta)$ such that $repair(M, \eta)$ is satisfiable iff (M, η) has a solution. $repair(M, \eta)$ is defined over the following propositions:

1. $E_{s,t} : (s, t) \in R$

2. $X_{s,\psi} : s \in S, \psi \in sub(\eta)$

3. $X_{s,\psi}^n : s \in S, 0 \leq n \leq |S|$, and $\psi \in sub(\eta)$ has the form $\mathsf{A}[\varphi \mathsf{V} \varphi']$ or $\mathsf{E}[\varphi \mathsf{V} \varphi']$

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

A solution for satisfiability of $repair(M, \eta)$, e.g., as given by a SAT solver, gives directly a solution to model repair. Denote this solution by $model(M, \mathcal{V})$. Then $model(M, \mathcal{V}) = (s_0', S', R', L')$, where $R' = \{(s, t) | \mathcal{V}(E_{s,t}) = tt\}$, S' consists of all states reachable from s_0 via paths of transitions in R', and $L' = L \upharpoonright S'$. Note that $model(M, \mathcal{V})$ does not depend directly on η.

Essentially, $repair(M, \eta)$ encodes all of the usual local constraints, e.g., $\mathsf{AX}\varphi$ holds in s iff φ holds in all successors of s. We modify these however, to take transition deletion into account. So, the local constraint for AX becomes $\mathsf{AX}\varphi$ holds in s iff φ holds in all successors of s *after* transitions have been deleted (to effect the repair). More precisely, instead of $X_{s,\mathsf{AX}\varphi} \equiv \bigwedge_{t|s \to t} X_{t,\varphi}$, we have $X_{s,\mathsf{AX}\varphi} \equiv \bigwedge_{t|s \to t}(E_{s,t} \Rightarrow X_{t,\varphi})$. Here $s \to t$ abbreviates $(s, t) \in R$. The other modalities ($\mathsf{EX}, \mathsf{AV}, \mathsf{EV}$) are treated similarly. We deal

with AU, EU by reducing them to EV, AV using duality. We require that the repaired structure M' be total by requiring that every state has at least one outgoing transition.

Definition 5 (*repair(M, η)*). *Let $M = (s_0, S, R, L)$ be a Kripke structure and η a CTL formula. Let $s \to t$ abbreviate $(s, t) \in R$. repair(M, η) is the conjunction of all the propositional formulae listed below. These are grouped into sections, where each section deals with one issue, e.g., propositional consistency. s, t implicitly range over S. Other ranges are explicitly given.*

M' satisfies η: $X_{s_0, \eta}$

M' is total, i.e., each state has an outgoing transition
$$\text{for all } s \in S: \bigvee\nolimits_{t | s \to t} E_{s,t}$$

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

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

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

Release formulae. Let $n = |S|$, i.e., the number of states in M.
for all $\mathsf{A}[\varphi\mathsf{V}\psi] \in sub(\eta)$: $X_{s,\mathsf{A}[\varphi\mathsf{V}\psi]} \equiv X^n_{s,\mathsf{A}[\varphi\mathsf{V}\psi]}$
for all $\mathsf{A}[\varphi\mathsf{V}\psi] \in sub(\eta)$, $m \in \{1...n\}$:
$$X^m_{s,\mathsf{A}[\varphi\mathsf{V}\psi]} \equiv X_{s,\psi} \wedge (X_{s,\varphi} \vee \bigwedge\nolimits_{t | s \to t}(E_{s,t} \Rightarrow X^{m-1}_{t,\mathsf{A}[\varphi\mathsf{V}\psi]}))$$
for all $\mathsf{A}[\varphi\mathsf{V}\psi] \in sub(\eta)$: $X^0_{s,\mathsf{A}[\varphi\mathsf{V}\psi]} \equiv X_{s,\psi}$
for all $\mathsf{E}[\varphi\mathsf{V}\psi] \in sub(\eta)$: $X_{s,\mathsf{E}[\varphi\mathsf{V}\psi]} \equiv X^n_{s,\mathsf{E}[\varphi\mathsf{V}\psi]}$
for all $\mathsf{E}[\varphi\mathsf{V}\psi] \in sub(\eta)$, $m \in \{1...n\}$:
$$X^m_{s,\mathsf{E}[\varphi\mathsf{V}\psi]} \equiv X_{s,\psi} \wedge (X_{s,\varphi} \vee \bigvee\nolimits_{t | s \to t}(E_{s,t} \wedge X^{m-1}_{t,\mathsf{E}[\varphi\mathsf{V}\psi]}))$$
for all $\mathsf{E}[\varphi\mathsf{V}\psi] \in sub(\eta)$: $X^0_{s,\mathsf{E}[\varphi\mathsf{V}\psi]} \equiv X_{s,\psi}$

We handle the "φ releases ψ" modality $[\varphi\mathsf{V}\psi]$ as follows. Along each path, either (1) a state is reached where $[\varphi\mathsf{V}\psi]$ is discharged ($\varphi \wedge \psi$), or (2) $[\varphi\mathsf{V}\psi]$ is shown to be false ($\neg\varphi \wedge \neg\psi$), or (3) some state eventually repeats. In case (3), we know that release also holds along this path. Thus, by expanding the release modality up to n times, where n is the number of states in the original structure M, we ensure that the third case holds

if the first two have not yet resolved the truth of $(\varphi \mathsf{V} \psi)$ along the path in question. To carry out the expansion correctly, we use a version of $X_{s,\mathsf{A}[\varphi \mathsf{V} \psi]}$ that is superscripted with an integer between 0 and n. This imposes a "well foundedness" on the $X^m_{s,\mathsf{A}[\varphi \mathsf{V} \psi]}$ propositions, and prevents for example, a cycle along which ψ holds in all states and yet the $X_{s,\mathsf{A}[\varphi \mathsf{V} \psi]}$ are assigned false in all states s along the cycle.

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

In each state $s \in S$, there are $O(|\eta| \times |S|)$ formulae to check, each of which has length $O(d)$, where d is the maximum number of succesors that any state in S has. The sum of lengths of all these formulae is $O(|\eta| \times |S|^2 \times d)$. The propositional labelling formulae add $O(|S| \times |\mathcal{AP}|)$ length, and so the size of $repair(M, \varphi)$ is $O(|\eta| \times |S|^2 \times d + |S| \times |\mathcal{AP}|)$, and so is polynomial in the size of (M, η). Clearly, $repair(M, \eta)$ can be constructed in polynomial time. Figure 1 presents our model repair algorithm, REPAIR(M, φ), which we show is sound, and complete provided that a complete SAT-solver is used. Recall that we use $model(M, \mathcal{V})$ to denote the structure M' derived from the repair of M w.r.t. η, i.e., $M' = (s'_0, S', R', L')$, where $R' = \{(s, t) | \mathcal{V}(E_{s,t}) = tt\}$, S' consists of all states reachable from s_0 via paths of transitions in R', and $L' = L \upharpoonright S'$.

Theorem 2 (Soundness). *Let $M = (s_0, S, R, L)$ 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 $M' = model(M, \mathcal{V})$, Then for all reachable states $s \in S'$ and CTL formulae $\xi \in sub(\eta)$:*
$$\mathcal{V}(X_{s,\xi}) = tt \text{ iff } M', s \models \xi \text{ and}$$
$$\text{for } m \in \{1...n\} : \mathcal{V}(X^m_{s,\xi}) = tt \text{ iff } M', s \models \xi.$$

Corollary 2 (Soundness). *If REPAIR(M, η) returns a structure $M' = (s'_0, S', R', L')$, then (1) M' is total, (2) $M' \subseteq M$, (3) $M', s_0 \models \eta$, and (4) M is repairable.*

Theorem 3 (Completness). *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 $M'', s_0 \models \eta$.*

Since M' results by removing transitions and unreachable states from M, the relation mapping each state in M' to "itself" in M is a simulation relation [13] from M' to M. Hence the following, where ACTL* [13] is the universal fragment (no existential path quantifier) of CTL*, and clause (2) follows from [13].

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

REPAIR(M, η):

model check $M, s_0 \models \eta$;
if successful, **then return** M
else
      compute $repair(M, \eta)$ as given in Section 3;
      submit $repair(M, \eta)$ to a sound and complete SAT-solver;
      if the SAT-solver returns "not satisfiable" **then**
            return "failure"
      else
            the solver returns a satisfying assignment $\mathcal{V}$;
            return $M' = model(M, \mathcal{V})$

Figure 1: The model repair algorithm.

3.3 ATL Model Repair using SAT solvers

We adapt Definition 5 for ATL as follows.

We omit the conjuncts for $\mathsf{AX}\varphi$, $\mathsf{EX}\varphi$, $\mathsf{A}[\varphi\mathsf{V}\psi]$, $\mathsf{E}[\varphi\mathsf{V}\psi]$, and add the following conjuncts. Here $\sigma(s)$ is the player whose turn it is to move in state s.

Nexttime formulae

$$\text{for all } \mathsf{X}\varphi \in sub(\eta) : X_{s, \ll A \gg \mathsf{X}\varphi} \equiv \begin{cases} \bigvee_{t|s \to t}(E_{s,t} \wedge X_{t, \ll A \gg \varphi}) \text{ if } \sigma(s) \in A \\ \bigwedge_{t|s \to t}(E_{s,t} \Rightarrow X_{t, \ll A \gg \varphi}) \text{ if } \sigma(s) \notin A \end{cases}$$

Release formulae. Let $n = |S|$ and $m \in \{1...n\}$. Then, for all $\ll A \gg [\varphi\mathsf{V}\psi] \in sub(\eta)$ we have the following conjuncts:
$X_{s, \ll A \gg [\varphi\mathsf{V}\psi]} \equiv X^n_{s, \ll A \gg [\varphi\mathsf{V}\psi]}$

$X^m_{s,\ll A\gg[\varphi\mathsf{V}\psi]} \equiv X_{s,\ll A\gg\psi}\wedge(X_{s,\ll A\gg\varphi}\vee\bigvee_{t|s\to t}(E_{s,t}\wedge X^{m-1}_{t,\ll A\gg[\varphi\mathsf{V}\psi]}))$ if $\sigma(s)\in A$

$X^m_{s,\ll A\gg[\varphi\mathsf{V}\psi]} \equiv X_{s,\ll A\gg\psi}\wedge(X_{s,\ll A\gg\varphi}\vee\bigwedge_{t|s\to t}(E_{s,t}\Rightarrow X^{m-1}_{t,\ll A\gg[\varphi\mathsf{V}\psi]}))$ if $\sigma(s)\notin A$

$X^0_{s,\ll A\gg[\varphi\mathsf{V}\psi]} \equiv X_{s,\ll A\gg\psi}$

As in Definition 5, the above formula encodes the possibilities for valuation of η and all its subformulae on M, and the possible substructures resulting from deleting transitions from M. We can still reduce until to release, since $\ll A\gg[\varphi\mathsf{U}\psi]\equiv\ll\Sigma-A\gg[\neg\varphi\mathsf{V}\neg\psi]$ in turn-based synchronous games [17].

4 Extensions of the Subtractive Repair Algorithm

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

4.1 Addition of States and Transitions

The subtractive repair algorithm performs repair by deleting transitions, with states being implicitly deleted if they become unreachable. Let $M = (s_0, S, R, L)$ be a Kripke structure with underlying set of atomic propositions $\mathcal{AP}$. By adding some states and transitions to M before performing repair, we can end up with a substructure M' that includes some of the added states. Thus, we have *addiditve repair*: repair performed by adding states and transitions,

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

In practice, the added states and transitions would be determined either manually, by the user of the repair tool, or mechanically using heuristics. While it seems possible to modify the repair formula directly to accommodate state/transition addition (e.g., by introducing new propositions for the added states and transitions), doing so does not seem to be any better than adding to the structure M and then regenerating the repair formula using the existing Definition 5. Note that proposition 1 no longer holds when we add states and transitions.

4.2 Discrete Event Supervisory Control

In the well-know discrete event supervisory control problem (DESC) [18, 19], a Kripke structure is given in which the transitions are labelled as "controllable" and "not controllable". The problem is to delete (disable) only controllable transitions so that the resulting structure satisfies a property, e.g., expressed in CTL. We easily subsume DESC when the property is expressed in CTL as follows. Conjoin to the repair formula $\text{REPAIR}(M, \eta)$ the transition propositions $E_{s,t}$ for all uncontrollable transitions $s \rightarrow t$. Thus, we submit the following formula to the SAT solver: $\text{REPAIR}(M, \eta) \wedge (\bigwedge_{(s \rightarrow t) \text{is uncontrollable}} E_{s,t})$. The resulting assignment produced by the SAT solver must then assign tt to all $E_{s,t}$ for all uncontrollable transitions $s \rightarrow t$, and so none of these transitions are deleted. By Theorem 3 (completeness), our repair method will then find a solution that involves deleting only controllable transitions, if such a solution exists. Thus, we subsume the discrete event supervisory control problem.

4.3 Generalized Boolean Constraints on Transition and State Deletion

The reduction given above used only simple conjunctions of $E_{s,t}$ propositions. We can conjoin arbitrary boolean formulae over the $E_{s,t}$ to $\text{REPAIR}(M, \eta)$, e.g., $E_{s,t} \equiv E_{s',t'} \wedge E_{s',t'} \equiv E_{s'',t''}$ adds the constraint that either all three transitions $s \rightarrow t$, $s' \rightarrow t'$, $s'' \rightarrow t''$ are deleted, or none are. This is useful in enforcing atomic read/write semantics in shared memory, as discussed below.

We can also add constraints on deleting states as follows. We can introduce a proposition N_s (N for "node") for each each state s with meaning that s is retained in the final model iff N_s is assigned tt. We now modify the clause for M' being total to: for all $s \in S : N_s \implies \bigvee_{t|s \rightarrow t} E_{s,t}$, and we add as conjunct: for all $s \in S : \neg N_s \implies (\bigwedge_{t|s \rightarrow t} \neg E_{s,t}) \wedge (\bigwedge_{t|t \rightarrow s} \neg E_{t,s})$, that is, a nondeleted state must have some outgoing transition, and a deleted state has no transitions, either incoming or outgoing.

Suppose we have a Kripke structure for two processes P_1 and P_2 executing some protocol, e.g., mutual exclusion. We can both fix the protocol and require the result to be symmetric in P_1 and P_2 (i.e., the code for P_2 results from interchanging the process indices 1 and 2 in the code for P_1 [3]) by adding the conjunct $N_s \equiv N_t$ for every pair of symmetric state s, t, i.e., such that t results from s by interchanging the process indices 1 and 2, and likewise for symmetric transitions (start and end states are symmetric).

Thus, we can check for the existence of symmetric concurrent algorithms. Note that these more general constraints cannot be dealt with by discrete event supervisory control, which only allows to specify individual transitions as controllable or not, and does not allow relating the deletion of one transition to the deletion of another.

4.4 Concurrent Program Repair

We now extend our approach to the repair of shared memory concurrent programs $P = P_1 \parallel \cdots \parallel P_K$, where processes atomically read, write one shared variable at a time. We provide repair w.r.t, CTL specifications. We partition $\mathcal{AP}$ into $\mathcal{AP}_1, \ldots, \mathcal{AP}_K$, where $\mathcal{AP}_i$ consists of the atomic propositions that can only be written by P_i (but can be read by other processes). There are also shared variables $x_1, \ldots, x_m$ (with finite domains) that can be read and written by all processes.

We use the atomic read/write notation introduced in [4, 5] for atomic read/write programs. Each process P_i is a synchronization skeleton [11], i.e., a directed graph where the nodes are *local states* that determine a truth assignment for the propositions in $\mathcal{AP}_i$, and the arcs between nodes are labeled with guarded commands; the guard reads atomic propositions of other processes and shared variables, the body is a parallel assignment that updates shared variables.

The atomic propositions in $\mathcal{AP}_i$ are consolidated into a single variable L_i (the "externally visible location counter") owned by P_i (i.e., written by P_i and read by other processes), so that the value of L_i in s_i is the set of all propositions in $\mathcal{AP}_i$ that hold in s_i. L_i provides incomplete information to other processes about the current local state of P_i: when P_i writes to L_i, its change of local state is visible to other processes. When P_i writes to a shared variable x, or reads, then its change of local state is not visible to other processes. Since L_i encodes location information, a single machine word is usually sufficient to store L_i.

$(s_i, B \to A, t_i)$ denotes an arc in P_i from local state s_i to local state t_i that is labeled with guarded command $B \to A$. The restrictions to atomic read/write syntax (cf. Definition 3.1.4 in [5]) are that each arc $(s_i, B \to A, t_i)$ of P_i is either:

– *unguarded and single-writing*: there is no guard (i.e., B is "true") and A either writes to L_i (i.e., L_i has different values in s_i and t_i, so its value in t_i must be written into it by A) or it writes to a single shared variable x (i.e., has the form $x := c$, where c is a value from the domain of x, in which case L_i must have the same value in s_i and t_i,).

– *single-reading and nonwriting*: there is no assignment (i.e., A is "skip"), L_i has the same value in s_i and t_i, and B has the form $Q_j \in L_j$ where $Q_j \in \mathcal{AP}_j$, $j \neq i$ or the form $x = c$. We call such a form for B a *simple term*.

A *global state* s is a tuple $\langle s_1, \ldots, s_K, v_1, \ldots, v_m \rangle$ where s_i is the current local state of P_i, and v_j is the current value of shared variable x_j. We write $s \upharpoonright i$ for the component of s that gives the local state of P_i.

An arc $arc = (s_i, B \rightarrow A, t_i)$ of P_i is *enabled* in global state s iff $s \upharpoonright i = s_i$ and $s(B) = \mathsf{true}$. Execution of $(s_i, B \rightarrow A, t_i)$ in a global state s where it is enabled generates a transition $s \overset{arc}{\rightarrow} t$, where t results from s by changing the local state of P_i from s_i to t_i, and changing the value of x to c if A has the form $x := c$. In general, an arc can be enabled in several global states. In the global state transition diagram M generated by execution of P, the set of all transitions generated by a single arc is called a *family*. We label every transition by the name of the family that it belongs to. Two different families do not intersect, since their transitions have different labels, even if the transitions have the same "effect" on the global state. This makes the technical development more convenient and does not cause loss of generality. Thus, the set of transitions in M is partitioned into families.

Let P be a shared memory atomic read/write concurrent program, and η a CTL specification for P. We generate the global state transition diagram $M = (s_0, S, R, L)$ of P. Suppose that $repair(M, \eta)$ has a satisfying assignment $\mathcal{V}$, and that $\mathcal{V}(E_{s,t}) = f\!f$ for some transition (s, t) in M. Let $\mathcal{F}$ be the family that (s, t) belongs to, and P_i be the process in which the arc arc generating $\mathcal{F}$ occurs. To preclude executing arc in global state s, the repaired P_i must detect that s is actually the current global state (and then not execute arc). This requires that P_i read enough externally visible location counters $L_j, j \neq i$, and shared variables, so that it can determine a pattern of assignment of values to these that is unique to s. In general, this may require that P_i read several location counters and shared variables atomically.

We now have two cases, depending on arc. First, suppose that arc is unguarded and single-writing, Then we cannot modify arc to read any information without violating the atomic read/write syntax restriction (effectively, arc becomes a test-and-set operation). We are thus left with two options: either make s unreachable, by deleting other transitions, or delete all the transitions in $\mathcal{F}$. This can be expressed as $(\bigwedge_{(s,t) \in \mathcal{F}} (\neg E_{s,t} \Rightarrow \neg r_s)) \vee (\bigwedge_{(s,t) \in \mathcal{F}} \neg E_{s,t})$, where r_s is the "reachability" proposition given in Definition 5. The first disjunct states that deletion of (s, t) requires that s be unreachable. The second disjunct states that all transitions in $\mathcal{F}$ are

deleted. We add the above as a conjunct to $repair(M, \eta)$.

The second case is that $arc = (s_i, B \to A, t_i)$ is single-reading and non-writing. Since B holds in s, the repair cannot allow B to continue being used as the guard for arc, unless s is made unreachable (in which case B is never evaluated in s), or the entire family $\mathcal{F}$ is deleted, in which case the arc arc is removed from P_i. However, it is possible that a simple term other than B could be used to effect a repair, namely one that holds in the initial states of all transitions in $\mathcal{F}$ except for the transition (s, t), i.e., in all states s' such that $s' \neq s \wedge \exists t' : (s', t') \in \mathcal{F}$, and also does not hold in any other global state. In [5], an algorithm for finding a suitable simple term, if it exists, is presented. Essentially, the algorithm checks all possible simple terms (their number is $O(|M|)$). While our approach is not able to replace the guard B by another guard, it is capable of deleting unsuitable simple terms, by removing the family corresponding to the arc in which the simple term is used as a guard. This encourages an experimental style, where we add extra arcs to the synchronization skeletons in the initial program, if we think they may contain suitable guards. Since this does not increase the number of local states of any process, nor the number of shared variables, the number of global states is unaffected. Thus, we could even add arcs for every possible simple term. Note however that Proposition 1 could be violated, as the additional arcs may induce additional transitions in M' that are not simulated by M. The conclusion of the preceding discussion is that we use the same idea for repair as we did for the unguarded and single-writing case, namely add $(\bigwedge_{(s,t) \in \mathcal{F}} (\neg E_{s,t} \Rightarrow \neg r_s)) \vee (\bigwedge_{(s,t) \in \mathcal{F}} \neg E_{s,t})$, a conjunct to $repair(M, \eta)$, with the possibility that we add "extra arcs" before repairing, to increase the possibilities for the repair.

5   Examples

5.1   Simple Example for CTL Model Repair

Consider the model in Figure 2 and the formula $\eta = (\mathsf{AG}p \vee \mathsf{AG}q) \wedge \mathsf{EX}p$. Manual simplification of $repair(M, \eta)$ yields $\mathsf{X}_{s,\eta} \equiv \neg E_{s,t} \wedge E_{s,u}$, so $\textsc{Repair}(M, \eta)$ will remove the edge (s, t) as shown. Our implementation produces the following truth assignment:

```
A_A_A & s_u & ~s_t & u_s & t_s & ~u_5_0 & ~t_1_0 & s_10_0 & s_7_0 &
s_9_0 & s_0_0 & ~t_4_0 & ~s_5_0 & ~u_1_0 & ~s_6_0 & t_2_0 & ~u_2_0 &
t_0_0 & ~s_1_0 & ~t_3_0 & s_2_0 & s_8_0 & u_4_0 & ~s_3_0 & t_5_0 &
u_3_0 & s_4_0 & u_0_0
```
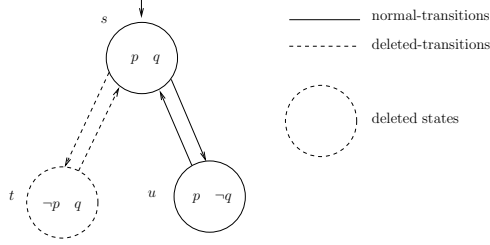
Figure 2: Input Kripke structure.

The variable s_t represents the edge from s to t, etc. Note that s_t is negated, indicating an assignment of ff, i.e., the edge should be deleted, as required.

We also ran our implementation with repair formula $\mathsf{AX}p \wedge \mathsf{AX}\neg p$. As expected, it returned "unsatisfiable," indicating that no repair exists.

5.2 Barrier Synchronization Problem Repair

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

1. Initially both processes are at the start of phase A: $SA_1 \wedge SA_2$

2. P_1 and P_2 are never simultaneously at the start of different phases:
$$\mathsf{AG}(\neg(SA_1 \wedge SB_2)) \wedge \mathsf{AG}(\neg(SA_2 \wedge SB_1))$$

3. P_1 and P_2 are never simultaneously at the end of different phases:
$$\mathsf{AG}(\neg(EA_1 \wedge EB_2)) \wedge \mathsf{AG}(\neg(EA_2 \wedge EB_1))$$

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

The structure in Figure 3 is repaired by removing edges and states that cause the violation of the synchronization rules (2) and (3)[1] Our implementation produced exactly this repair. The repair formula in CNF contained 236 propositions and 401 clauses.

---

[1]Note that the bottom $[SA_1 SA_2]$ state is the same as the top $[SA_1 SA_2]$ state, and is repeated only for clarity of the figure.
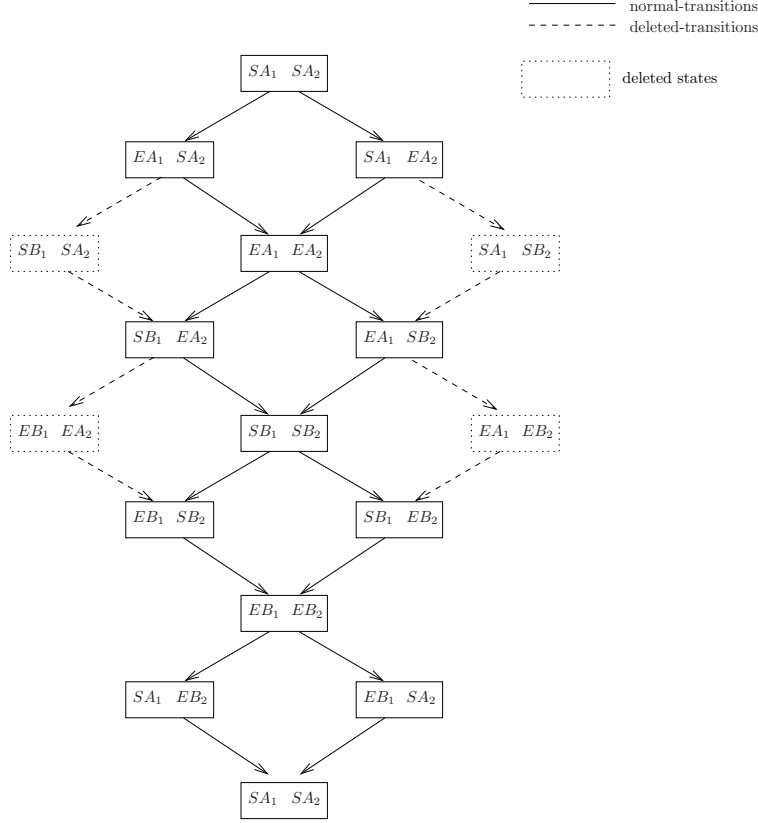
Figure 3: Barrier synchronization repair.

6   Implementation of the Repair Method

We implemented the method in Python. Our implementation takes a Kripke structure M and CTL formula η as input, generates $repair(M, \eta)$ as given by Definition 5, converts it to CNF, and then invokes the SAT solver zChaff. The implementation is available at `http://www.cs.aub.edu.lb/pa07/pca/Eshmun.html`.

Table 1 gives performance figures for our implementation, running on a PC with Pentium 4 CPU at 3.00GHz, and 512MB RAM. For M, we generated transitions graphs randomly, specifying the number of nodes N and the probability P that there is a transition from some given node to some other given node. We used a constant probability $P = 0.1$. In M, we used $\mathcal{AP} = \{p, q\}$, and the propositional labels were generated randomly for each state.

We show the number of propositions and clauses in the CNF form of $repair(M, \eta)$, and the total time our implementation takes to produce a satisfying assignment. This shows a typically expected increase with the number of nodes in the graph.

For $N = 20, 30$ we used $\eta = \mathsf{AXA}[p\mathsf{V}q] \wedge \mathsf{EX}q$. For $N = 40$ to 80, we used $\eta = \mathsf{A}[p\mathsf{V}q]$.

Table 1: Model Repair Results

N	Propositions	Clauses	Time
30	309	3506	2.437s
40	449	3986	3.563s
50	608	13909	9.228s
60	781	47665	31.223s
70	993	106136	1m52.231s
80	1183	174107	3m17.140s

7 Conclusions

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

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

References

[1] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49:672–713, 2002.

[2] M. Antoniotti and B. Mishra. Np-completeness of the supervisor synthesis problem for unrestricted CTL specifications. In *Workshop on Discrete Event Systems*, 1996.

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

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

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

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

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

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

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

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

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

[12] E. A. Emerson and A. P. Sistla. Symmetry and model checking. In *Computer-Aided Verification*, 1993.

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

[14] R. Hojati, R. K. Brayton, and R. P. Kurshan. Bdd-based debugging of design using language containment and fair CTL. In *Computer Aided Verification*, 1993.

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

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

[17] F. Laroussinie, N. Markey, and G.Oreiby. On the expressiveness and complexity of ATL. In *FoSSaCS*, 2007.

[18] P. J. G. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM*, 25(25), Jan. 1987.

[19] P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

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

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

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

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

A Manual Simplification of the Repair Formula of the First Example

We show how $repair(M, \eta)$. for our first example in Section 5 is simplified manually. We omit the clauses dealing with reachability.

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

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

$$X_{s,\mathsf{AG}p} \equiv X^3_{s,\mathsf{AG}p}$$
$$X^3_{s,\mathsf{AG}p} \equiv X_{s,p} \wedge (E_{s,t} \Rightarrow X^2_{t,\mathsf{AG}p}) \wedge (E_{s,u} \Rightarrow X^2_{u,\mathsf{AG}p})$$
$$X^2_{t,\mathsf{AG}p} \equiv X_{t,p} \wedge (E_{t,s} \Rightarrow X^1_{s,\mathsf{AG}p})$$
$$X^2_{u,\mathsf{AG}p} \equiv X_{u,p} \wedge (E_{u,s} \Rightarrow X^1_{s,\mathsf{AG}p})$$
$$X^1_{s,\mathsf{AG}p} \equiv X_{s,p} \wedge (E_{s,t} \Rightarrow X^0_{t,\mathsf{AG}p}) \wedge (E_{s,u} \Rightarrow X^0_{u,\mathsf{AG}p})$$
$$X^0_{t,\mathsf{AG}p} \equiv X_{t,p} \equiv \mathit{ff}$$
$$X^0_{u,\mathsf{AG}p} \equiv X_{u,p} \equiv \mathit{tt}$$

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

$$X^1_{s,\mathsf{AG}p} \equiv \neg E_{s,t}$$
$$X^2_{u,\mathsf{AG}p} \equiv (E_{u,s} \Rightarrow X^1_{s,\mathsf{AG}p})$$
$$X^2_{u,\mathsf{AG}p} \equiv E_{u,s} \Rightarrow \neg E_{s,t}$$
$$X^3_{s,\mathsf{AG}p} \equiv \neg E_{s,t} \wedge (E_{s,u} \Rightarrow X^2_{u,\mathsf{AG}p})$$
$$X^3_{s,\mathsf{AG}p} \equiv \neg E_{s,t} \wedge (E_{s,u} \Rightarrow (E_{u,s} \Rightarrow \neg E_{s,t})) \equiv \neg E_{s,t}$$
$$X_{s,\mathsf{AG}p} \equiv \neg E_{s,t}$$

Symmetrically, we have:

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

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

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

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

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

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

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

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

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

B Proofs

B.1 Proof of Theorem 1.

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

NP-membership: Given a candidate solution M', the condition $M' \subseteq M$ is easily verified in polynomial time. $M', s_0 \models \eta$ is verified in linear time using the CTL model checking algorithm of [8].

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

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

- $L(s_0) = \emptyset$

- $L(s_j) = p_j$ where $1 \leq j \leq m$

- $L(t_j) = q_j$ where $1 \leq j \leq m$

η is given by:
$$\eta = \bigwedge_{1 \leq i \leq n} (\varphi_i^1 \vee \varphi_i^2 \vee \varphi_i^3)$$

where:

- if $a_i = x_j$ then $\varphi_i^1 = \mathsf{AG}(p_j \Rightarrow \mathsf{EX} q_j)$

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

- if $b_i = x_j$ then $\varphi_i^2 = \mathsf{AG}(p_j \Rightarrow \mathsf{EX} q_j)$

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

- if $c_i = x_j$ then $\varphi_i^3 = \mathsf{AG}(p_j \Rightarrow \mathsf{EX} q_j)$

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

Thus, if $a_i^1 = x_i$, then the transition from s_i to t_i (which we write as $s_i \to t_i$ must be retained in M', and if $a_i = \neg x_i$, then the transition $s_i \to t_i$ 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 fixed. The proof is by double implication.

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

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

B.2 Proof of Corollary 1.

Proof. NP-membership: guess the substructure M' of M and then check $M \models \eta$ in polynomial time using a polynomial time model checking algorithm.

NP-hardness: use the reduction from 3SAT to CTL model checking given in the proof of Theorem 1, and then use the assumed reduction to L model checking. □

B.3 Proof of Theorem 2.

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

Case $\xi = \neg\varphi$:
$\mathcal{V}(X_{s,\xi}) = tt$ iff
$\mathcal{V}(X_{s,\neg\varphi}) = tt$ iff (by propositional consistency clause of Definition 5)
$\mathcal{V}(X_{s,\varphi}) = ff$ iff (by the induction hypothesis)
$not(M', s \models \varphi)$ iff
$M', s \models \neg\varphi$ iff
$M', s \models \xi$

Case $\xi = \varphi \vee \psi$:
$\mathcal{V}(X_{s,\xi}) = tt$ iff
$\mathcal{V}(X_{s,\varphi\vee\psi}) = tt$ iff (by propositional consistency clause of Definition 5)
$\mathcal{V}(X_{s,\varphi}) = tt$ or $\mathcal{V}(X_{s,\psi}) = tt$ iff (by the induction hypothesis)
$(M', s \models \varphi)$ or $(M', s \models \psi)$ iff $M', s \models \varphi \vee \psi$ iff $M', s \models \xi$

Case $\xi = \varphi \wedge \psi$:
$\mathcal{V}(X_{s,\xi}) = tt$ iff
$\mathcal{V}(X_{s,\varphi\wedge\psi}) = tt$ iff (by propositional consistency clause of Definition 5)
$\mathcal{V}(X_{s,\varphi}) = tt$ and $\mathcal{V}(X_{s,\psi}) = tt$ iff (by the induction hypothesis)
$(M', s \models \varphi)$ and $(M', s \models \psi)$ iff $M', s \models \varphi \wedge \psi$ iff $M', s \models \xi$

Case $\xi = \mathsf{AX}\varphi$:
$\mathcal{V}(X_{s,\xi}) = tt$ iff
$\mathcal{V}(X_{s,\mathsf{AX}\varphi}) = tt$ iff
$\bigwedge_{t|s\rightarrow t} \mathcal{V}(E_{s,t} \Rightarrow X_{t,\varphi}) = tt$ iff
$\bigwedge_{t|s\rightarrow t} \mathcal{V}(E_{s,t}) = tt \Rightarrow \mathcal{V}(X_{t,\varphi}) = tt$ iff (since s is reachable by assumption,
$E_{s,t}$ implies that t also reachable, and apply the induction hypothesis)
$\bigwedge_{t|s\rightarrow t}(s,t) \in R' \Rightarrow M', t \models \varphi$ iff
$M', s \models \mathsf{AX}\varphi$ iff
$M', s \models \xi$

Case $\xi = \mathsf{EX}\varphi$:
$\mathcal{V}(X_{s,\xi}) = tt$ iff
$\mathcal{V}(X_{s,\mathsf{EX}\varphi}) = tt$ iff
$\bigvee_{t|s\rightarrow t} \mathcal{V}(E_{s,t} \wedge X_{t,\varphi}) = tt$ iff
$\bigvee_{t|s\rightarrow t} \mathcal{V}(E_{s,t}) = tt \wedge \mathcal{V}(X_{t,\varphi}) = tt$ iff (since t is reachable from s by assumption, and apply the induction hypothesis)
$\bigvee_{t|s\rightarrow t}(s,t) \in R' \wedge M', t \models \varphi$ iff
$M', s \models \mathsf{EX}\varphi$ iff

$M', s \models \xi$

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

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

We now have two cases

1. $M', s \models \varphi$. In this case, $M', s \models \mathsf{A}[\varphi \mathsf{V} \psi]$, and so $M', s \models \xi$.

2. $\bigwedge_{t|s \to t}(s,t) \in R' \Rightarrow \mathcal{V}(X_{t,\mathsf{A}[\varphi \mathsf{V} \psi]}^{n-1})$.

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

- $M', t \models \varphi$

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

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

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

1. $\mathcal{V}(X_{s,\varphi})$. Since we have $\mathcal{V}(X_{s,\psi}) \wedge \mathcal{V}(X_{s,\varphi})$ we conclude $\mathcal{V}(X_{s,\mathsf{A}[\varphi\mathsf{V}\psi]})$, and so we are done.

2. $\bigwedge_{t|s\to t}(s,t) \in R' \Rightarrow M',t \models \mathsf{A}[\varphi\mathsf{V}\psi]$

For case 2, we proceed as follows. Let t be an arbitrary state such that $(s,t) \in R'$. Then $M',t \models \mathsf{A}[\varphi\mathsf{V}\psi]$. If we show that $\mathcal{V}(X^{n-1}_{t,\mathsf{A}[\varphi\mathsf{V}\psi]})$ follows from $M',t \models \mathsf{A}[\varphi\mathsf{V}\psi]$ then we can conclude $\mathcal{V}(X_{s,\mathsf{A}[\varphi\mathsf{V}\psi]})$ by Definition 5. Proceeding as above, we conclude $\mathcal{V}(X_{t,\psi})$ and one of the same two cases as above:

- $\mathcal{V}(X_{t,\varphi})$, so by Definition 5, $\mathcal{V}(X^{n-1}_{t,\mathsf{A}[\varphi\mathsf{V}\psi]})$ holds.

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

As before, in case 2 we are "counting down." Since we count down for $n = |S|$, then along every path starting from s, either case (1) occurs, which "terminates" that path, as far as establishment of $\mathcal{V}(X_{t,\varphi})$ is concerned, or we will repeat a state before (or when) the counter reaches 0. Along such a path (from s to the repeated state, call it v), ψ holds at all states. By Definition 5, $X^0_{v,\mathsf{A}[\varphi\mathsf{V}\psi]} \equiv X_{v,\psi}$. From $M',v \models \psi$ and the induction hypothesis, $\mathcal{V}(X_{v,\psi})$ holds. Hence $X^0_{v,\mathsf{A}[\varphi\mathsf{V}\psi]}$ holds. Thus, along every path starting from s, we reach a state w such that $\mathcal{V}(X^m_{w,\mathsf{A}[\varphi\mathsf{V}\psi]})$ holds for some $m \in \{0,\ldots,n\}$. Hence by Definition 5, $\mathcal{V}(X_{s,\mathsf{A}[\varphi\mathsf{V}\psi]})$ holds.

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

B.4 Proof of Corollary 2.

Proof. Let $\mathcal{V}$ be the truth assignment for $repair(M,\eta)$ that was returned by the SAT-solver in the execution of $\textsc{Repair}(M,\eta)$. Since the SAT-solver is assumed sound, $\mathcal{V}$ is actually a satisfying assignment for $repair(M,\eta)$. For (1), let u be an arbitrary reachable state in M'. Consider a path from s_0 to u. By definition of $repair(M,\eta)$, we have $\mathcal{V}(E_{s,t}) = tt$ for every transition (s,t) along this path. Hence $\mathcal{V}(\bigvee_{v|u\to v} E_{u,v}) = tt$. Hence u has some outgoing transition in M'. (2) holds by construction of M', which is derived from M by deleting transitions and (subsequently) unreachable states. For (3), note that $X_{s_0,\eta}$ is a conjunct of $repair(M,\eta)$ by definition of $repair(M,\eta)$. Hence $\mathcal{V}(X_{s_0,\eta}) = tt$. Hence, by Theorem 2, $M',s_0 \models \eta$. Finally, (4) follows from (1)–(3) and Definition 3. □

B.5 Proof of Theorem 3.

Proof. Assume that M is repairable with respect to η. By Definition 3, there exists a total substructure M' of M such that $M', s_0 \models \eta$. We define a satisfying valuation $\mathcal{V}$ for $repair(M, \eta)$ as follows.

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

Assign tt to $X_{s_0, \eta}$. Consider an execution of the CTL model checking algorithm of [8] for checking $M', s_0 \models \eta$. This algorithm will assign a value to every formula φ in $sub(\eta)$ in every reachable state s of M'. Set $\mathcal{V}(X_{s,\varphi}$ to this value. By construction of the [8] model checking algorithm, these valuations will satisfy all of the constraints given in the "propositional labeling," "propositional consistency," "nexttime formulae," and "release formulae" sections of Definition 5. Hence all conjuncts of $repair(M, \eta)$ are assigned tt by $\mathcal{V}$. Hence $\mathcal{V}(repair(M, \eta)) = tt$, and so $repair(M, \eta)$ is satisfiable.

Now the SAT-solver used is assumed to be complete, and so will return some satisfying assignment for $repair(M, \eta)$ (not necessarily $\mathcal{V}$, since there may be more than one satisfying assignment). Thus, REPAIR(M, η) returns a structure M', rather than "failure." By corollary 2, M'' is total, $M'' \subseteq M$, and $M'', s_0 \models \eta$. □

C   Technical Background

C.1   Computation Tree Logic

Let $\mathcal{AP}$ be a set of atomic propositions. including the constants true and false. We use true, false as "constant" propositions whose interpretation is always the truth values tt, ff, respectively.

The logic CTL [11] is given by the following grammar:

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

where $p \in \mathcal{AP}$.

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

Definition 6. *A Kripke structure is a tuple $M = (s_0, S, R, L)$ where S is a finite state of states, $s_0 \in S$ is a single initial state, $R \subseteq S \times S$ is a transition relation, and $L : S \mapsto 2^{\mathcal{AP}}$ is a labeling function that associates each state $s \in S$ with a subset of atomic propositions, namely those that hold in the state.*

We assume that a Kripke structure $M = (s_0, S, R, L)$ is total, i.e., $\forall s \in S, \exists s' \in S : (s, s') \in R$. A path in M is a (finite or infinite) sequence of states, $\pi = s_0, s_1, \ldots$ such that $\forall i \geq 0 : (s_i, s_{i+1}) \in R$. A fullpath is an infinite path.

Definition 7. *$M, s \models \varphi$ means that formula φ is true in state s of structure M and $M, s \not\models \varphi$ means that formula φ is false in state s of structure M. We define $\models$ inductively as usual:*

- $M, s \models \mathsf{true}$

- $M, s \not\models \mathsf{false}$

- *$M, s \models p$ iff $p \in L(s)$ where atomic proposition $p \in \mathcal{AP}$*

- *$M, s \models \neg\varphi$ iff $M, s \not\models \varphi$*

- *$M, s \models \varphi \wedge \psi$ iff $M, s \models \varphi$ and $M, s \models \psi$*

- *$M, s \models \varphi \vee \psi$ iff $M, s \models \varphi$ or $M, s \models \psi$*

- *$M, s \models \mathsf{AX}\varphi$ iff for all t such that $(s, t) \in R : (M, t) \models \varphi$*

- *$M, s \models \mathsf{EX}\varphi$ iff there exists t such that $(s, t) \in R$ and $(M, t) \models \varphi$*

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

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

We use $M \models \varphi$ to abbreviate $M, s_0 \models \varphi$. We introduce the abbreviations $\mathsf{A}[\phi \mathsf{U} \psi]$ for $\neg\mathsf{E}[\neg\varphi \mathsf{V} \neg\psi]$, $\mathsf{E}[\phi \mathsf{U} \psi]$ for $\neg\mathsf{A}[\neg\varphi \mathsf{V} \neg\psi]$, $\mathsf{AF}\varphi$ for $\mathsf{A}[\mathsf{true} \mathsf{U} \varphi]$, $\mathsf{EF}\varphi$ for $\mathsf{E}[\mathsf{true} \mathsf{U} \varphi]$, $\mathsf{AG}\varphi$ for $\mathsf{A}[\mathsf{false} \mathsf{V} \varphi]$, $\mathsf{EG}\varphi$ for $\mathsf{E}[\mathsf{false} \mathsf{V} \varphi]$.

C.2   Alternating-Time Temporal Logic

We review *Alternating-Time Temporal Logic* (ATL)[1]. ATL extends the existential and universal quantification over paths of CTL by offering selective path quantification by a set of *players*, i.e., paths along which the set of players can "enforce" the satisfaction of a formula. In general, ATL is interpreted over *concurrent game structures* where every state transition results from each player choosing it's move, and then all players moving "at the same time." There are also several kinds of restricted structures in

which ATL can be interpreted. *Turn-based synchronous* games are games where in each step only one player makes a move, and the current player is determined by the current state. *Moore synchronous* games are games where the state space is partitioned according to the players, and in each step, every player updates its own components of the state independently of other players. *Turn-based asynchronous* are games in which in each step only one player has a choice of moves and that player is determined by a fair scheduler. In this paper we restrict ourselves to turn-based synchronous games. The results obtained still apply to other types of games since they can be reduced to turn-based synchronous games in polynomial time [1].

Let $\mathcal{AP}$ be a set of atomic propositions including the constants true and false. Let Σ denote the set of players. The logic ATL is given by the following grammar:

$$\varphi ::= \mathsf{true} \mid \mathsf{false} \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \ll A \gg X\varphi \mid \ll A \gg [\varphi \mathsf{V} \varphi]$$

where $p \in \mathcal{AP}$, $A \subseteq \Sigma$

We use $M \models \varphi$ to abbreviate $M, s_0 \models \varphi$. We introduce the abbreviations $\ll A \gg [\phi \mathsf{U} \psi]$ for $\neg \ll \Sigma - A \gg [\neg\varphi \mathsf{V} \neg\psi]$, $\ll A \gg F\varphi$ for $\ll A \gg [\mathsf{true} \mathsf{U} \varphi]$, $\ll A \gg G\varphi$ for $\ll A \gg [\mathsf{false} \mathsf{V} \varphi]$.

Definition 8 (ATL formula subformulae). *Given an ATL formula φ, its subformulae $sub(\varphi)$ is defined as follows:*

- $sub(p) := p$ *where p is true, false, or an atomic proposition*

- $sub(\varphi \wedge \psi) := \{\varphi \wedge \psi\} \cup sub(\varphi) \cup sub(\psi)$

- $sub(\varphi \vee \psi) := \{\varphi \vee \psi\} \cup sub(\varphi) \cup sub(\psi)$

- $sub(\ll A \gg X\varphi) := \{\ll A \gg X\varphi\} \cup sub(\varphi)$

- $sub(\ll A \gg (\varphi \mathsf{V} \psi)) := exp(\ll A \gg (\varphi \mathsf{V} \psi)) \cup sub(\varphi) \cup sub(\psi)$

In [1], the semantics of ATL is defined with respect to concurrent game structures. Since we consider only turn-based synchronous game structures, we provide a (simpler) definition for ATL semantics with respect to turn-based synchronous game structures.

Definition 9. *A turn-based synchronous game structure is a tuple $M = (s_0, S, R, L, \sigma)$ where S is a finite state of states, s_0 is the single initial state, $R \subseteq S \times S$ is a transition relation and $L : S \to 2^{\mathcal{AP}}$ is a labeling function that associates each state $s \in S$ with a subset of atomic propositions, namely those that hold in the state. σ is the turn function $\sigma : S \mapsto \Sigma$ that maps each state to a player (whose turn it is to make a move).*

We assume that each structure $M = (s_0, S, R, L, \sigma)$ is total, i.e., $\forall s \in S, \exists s' \in S : (s, s') \in R$. A path in M is a (finite or infinite) sequence of states, $\pi = s_0, s_1, \dots$ such that $\forall i \geq 0 : (s_i, s_{i+1}) \in R$. A fullpath is an infinite path.

For a path π and a position $i \geq 0$, we use $\pi[i]$ to denote the ith state of π. A strategy for a player $a \in \Sigma$ is a mapping $f_a : S^* \mapsto S$ that assigns to every finite path π a successor state $s \in S$. Given a state $s \in S$ and a set $A \subseteq \Sigma$ of players, an A-strategy $F_A = \{f_a \mid a \in A\}$ is a set of strategies, one for each player in A. We define the outcomes of F_A from s to be the set $out(s, F_A)$ of all fullpaths that the players in A can enforce when they follow the strategies in F_A, i.e., a fullpath $\pi = s_0, s_1, \dots$ is in $out(s, F_A)$ if $s_0 = s$ and for all $i \geq 0$, if $a = \sigma(\pi[i])$ then $s_{i+1} = f_a(\pi[0, i])$.

Definition 10 (ATL semantics). *$M, s \models \varphi$ means that φ is true in state s of game structure $M = (s_0, S, R, L, \sigma)$. $M, s \not\models \varphi$ means that formula φ is false in state s of game structure M. We define $\models$ inductively as usual:*

- $M, s \models \mathsf{true}$

- $M, s \not\models \mathsf{false}$

- *$M, s \models p$ iff $p \in L(s)$ where atomic proposition $p \in \mathcal{AP}$*

- *$M, s \models \neg\varphi$ iff $M, s \not\models \varphi$*

- *$M, s \models \varphi \wedge \psi$ iff $M, s \models \varphi$ and $M, s \models \psi$*

- *$M, s \models \varphi \vee \psi$ iff $M, s \models \varphi$ or $M, s \models \psi$*

- *$M, s \models \ll A \gg \mathsf{X}\varphi$ iff there exists a set F_A of strategies, one for each player in A, such that for all fullpaths $\pi \in out(s, F_A)$, we have $M, \pi[1] \models \varphi$*

- *$M, s \models \ll A \gg [\varphi \mathsf{V} \psi]$ iff there exists a set F_A of strategies, one for each player in A, such that for all fullpaths $\pi \in out(s, F_A)$:*
  $\forall k \geq 0 : (\forall j < k : (M, \pi[j] \not\models \varphi)$ implies $M, \pi[k] \models \psi$