Concurrent Program Repair Via SAT Solving

Ali Cherri, Kinan Dak El Bab
Department of Computer Science
American University of Beirut
ahc11@mail.aub.edu, kmd14@mail.aub.edu

May 19, 2015

Abstract

We consider the *model repair problem*: given a finite Kripke structure M and a CTL formula η, determine if M contains a substructure M' that satisfies η. Thus, M can be "repaired" to satisfy η by deleting some transitions. We map an instance (M, η) of model repair to a boolean formula $\mathit{repair}(M, \eta)$ such that (M, η) has a solution iff $\mathit{repair}(M, \eta)$ is satisfiable. Furthermore, a satisfying assignment determines which states and transitions must be removed from M to generate a model M' of η. Thus, we can use any SAT solver to repair Kripke structures. Using a complete SAT solver yields a complete algorithm: it always finds a repair if one exists. We also show that CTL model repair is NP-complete. We also show that several Kripke structures can model interaction more efficiently than one, using pairwise representation.

1 Preliminaries:

1.1 What is CTL

Computation Tree Logic is a system of rules and symbolism for representing, and reasoning about, propositions qualified in terms of time. It extends propositional logic with temporal modalities.

Quantifiers over paths:

- $\mathsf{A}\varphi$: All: φ has to hold on all paths from the current state.
- $\mathsf{E}\varphi$: Exists: there exists at least one path starting from the current state where φ holds.

Path-specific quantifiers:

- $\mathsf{X}\varphi$: Next: φ has to hold at the next state.
- $\mathsf{G}\varphi$: Globally: φ has to hold on the entire subsequent path.
- $\mathsf{F}\varphi$: Finally: φ eventually has to hold.
- $\varphi\mathsf{U}\psi$: Until: φ has to hold until at some position ψ holds. This implies that ψ will be verified in the future.

- $\varphi \mathsf{W} \psi$: Weak until: φ has to hold until ψ holds. The difference with U is that there is no guarantee that ψ will ever be verified.
- $\varphi \mathsf{V} \psi$: Release: as long as ψ does not hold φ holds. ψ is required to hold at some point.

CTL modalities can all be reduced to a minimal set {EX, AV, EV } combined with the basic boolean operators { $\wedge$, $\vee$, }

1.2 Kripke Structures

A Kripke structure M is defined as a 4-tuple M = (S_0, S, R, L, AP):

- a set of initial states $S_0 \subseteq$ S
- a finite set of states S.
- a transition relation R $\subseteq$ S x S.
- a labeling function L: S $\rightarrow 2^{AP}$
- a set of atomic propositions(labels) AP.

1.3 Model Checking

Model checking is a method for formally verifying finite-state concurrent systems. Specifications about the system are expressed as temporal logic formulas, and efficient symbolic algorithms are used to traverse the model defined by the system and check if the specification holds or not. In order to solve such a problem algorithmically, both the model of the system and the specification are formulated in some precise mathematical language: To this end, it is formulated as a task in logic, namely to check whether a given structure satisfies a given logical formula. The concept is general and applies to all kinds of logics and suitable structures. A simple model-checking problem is verifying whether a given formula in the propositional logic is satisfied by a given structure.

2 Model Repair

2.1 Subtractive Repair Method

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 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, denotedv$M' \subseteq M$, iff $S'_0 \subseteq S_0$, $S' \subseteq S$, $R' \subseteq R$, and $L' = L \restriction S'$.*

Definition 2 (Repairable)**.** *Given Kripke structure $M = (S_0, S, R, L, AP)$ and CTL formula η. M is* repairable *with respect to η if there exists a Kripke structure $M' = (S'_0, S', R', L', AP)$ such that M' is total, $M' \subseteq M$, and $M', S'_0 \models \eta$.*

Definition 3 (Model Repair Problem)**.** *For Kripke structure M and CTL formula η, we use $\langle M, \eta \rangle$ for the corresponding repair problem. The decision version of repair problem $\langle M, \eta \rangle$ is to decide if M is repairable w.r.t. η. The functional version of repair problem $\langle M, \eta \rangle$ is to return an M' that satisfies Def. 2, in the case that M is repairable w.r.t. η.*

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

Figure 1: The model repair algorithm.

3 Concurrent Model Repair

3.1 Definition

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

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

A *pair-structure* $M_{ij} = (S_0^{ij}, S_{ij}, R_{ij}, L_{ij}, AP_{ij})$ is a multiprocess Kripke structure over two process indices, e.g., i, j. Its set of atomic propositions is $AP_i \cup AP_j$. M_{ij} defines the direct interaction between processes P_i and P_j. If P_i interacts directly with a third process P_k, then a second pair structure, $M_{ik} = (S_0^{ik}, S_{ik}, R_{ik}, L_{ik}, AP_{ik})$, over indices i, k, defines this interaction. Note that M_{ij} and M_{ik} have the atomic propositions AP_i in common, and so their parallel composition must obey the following consistency condition: in any reachable global state s, the corresponding (i.e., projected) local states s_{ij} of M_{ij} and s_{ik} of M_{ik} must agree on all the atomic poropositions in AP_i. Formally, in the parallel composition of M_{iJ} and M_{ik}, a global

state can be viewed as having the form $\langle s_{ij}, s_{ik} \rangle$, where s_{ij} is a state of M_{ij}, and s_{ik} is a state of M_{ik}. Then, we require, $\forall p \in AP_i : p \in L_{ij}(s_{ij})$ iff $p \in L_{ik}(s_{ik})$.

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

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

to the repair formula, where $E(tr)$ is the repair proposition corresponding to transition tr.

4 Tool Review

The tool consists of five packages: Data Structures, Logical Expressions, Model Checker, Model Repairer and the GUI.

- Data Structures consist of a structure, states and transitions. An abstraction is done on structures implementing basic structure components. A Kripke inherits from the abstract structure implementation.
- Logical Expressions consist of an abstract expression class which all logical expressions inherit from it. Furthermore, abstract subclasses are implemented to differentiate between CTL modalities predict logic, boolean literals and boolean variables.
- Model Checker takes as input a Kripke structure $M = (S_0, S, R, L, AP)$, and a CTL formula ϕ and verifies if M satisfies ϕ.
- Mode Repairer takes as input a Kripke structure M and a CTL formulae ϕ and return a repaired model with respect to ϕ.
- SAT Solver: takes as input a CNF file and return a flag that specifies whether the CNF formulae is satisfiable or not. In case it is satisfiable it also returns the satisfying valuation.
- GUI implements a graphical interface between user and the other modules.

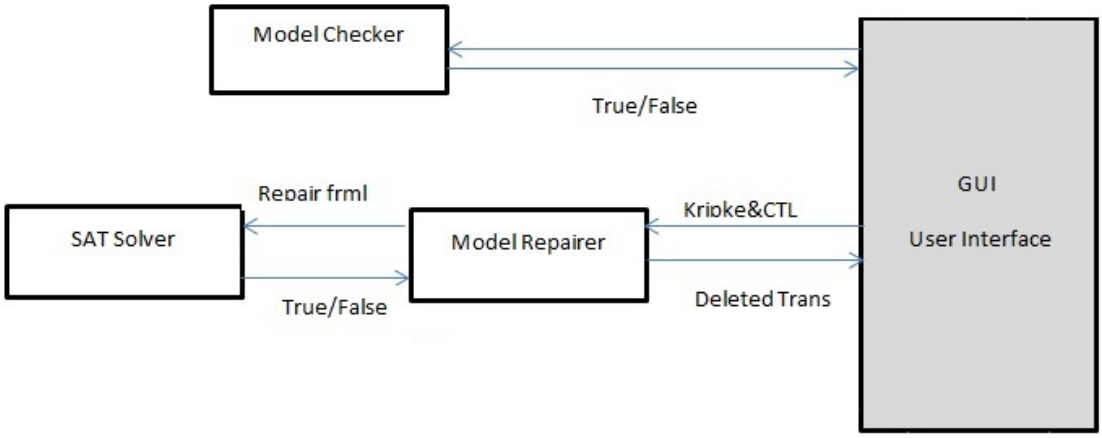

Figure 2: Tool main modules

5 Experimental Results

# Processes	Old Tool	Our Tool
2	93	107
3	1,210	104
4	Out of memory	248

Table 1: Times (in Miliseconds) taken to repair mutual exclusion w.r.t. safety, single kripke strcuture

# Processes	Old Tool	Our Tool
2	67	63
3	460	107
4	1,105	164

Table 2: Times (in Miliseconds) taken to repair barrier w.r.t. synchronisation, single kripke structure

6 Technical Details

6.1 Algorithm for Generating the Repair Formula

The Algorithm is provided in pseudo code in the appendix on the last page.

6.2 Constructing The Synchronisation Formula

```
/**
 * Construct the structure formula, the formula that synchronizes transitions
 * for each processes over all the structures.
 *
 * <p>The formula necessitates that for a specific process, if all the transitions
 * taking that process from one state to another were deleted in one Structure, they
 * must be also deleted in all of the structures.</p>
 *
 * @return the Structure Formula for this MultiKripkeStructure.
 */
protected AbstractExpression getSynchFormula() {
   HashMap<String, EquivalenceOperator> equiMap = new HashMap<String,
       EquivalenceOperator>();
   Integer[] allProcesses = getParticipatingProcesses();
   AndOperator andOp = new AndOperator();
   for(int p : allProcesses) {
      ArrayList<KripkeStructure> structures = new ArrayList<KripkeStructure>();
      ArrayList<Integer> indices = new ArrayList<Integer>();
      for(int i = 0; i < processes.size(); i++) {
         int[] pairs = processes.get(i);
         if(pairs[0] == p || pairs[1] == p) {
            structures.add(kripkeList.get(i));
            indices.add(pairs[0] == p ? 0 : 1);
         }
      }

      for(int i = 0; i < structures.size(); i++) {
         KripkeStructure s = structures.get(i);
         HashMap<String, ArrayList<KripkeTransition>> map =
             s.mapLabelToTransition(indices.get(i));

         for(String key : map.keySet()) {
            if(equiMap.get(key) == null)
               equiMap.put(key, new EquivalenceOperator());

            OrOperator orOp = new OrOperator();
            for(KripkeTransition t : map.get(key)) {
               orOp.or(new BooleanVariable(t.getVarName()));
            }

            equiMap.get(key).equate(orOp);
         }
      }
   }

   for(String key : equiMap.keySet()) {
      andOp.and(equiMap.get(key));
   }

   return andOp;
}
```

6.3 UML Diagram for CTL Logic API

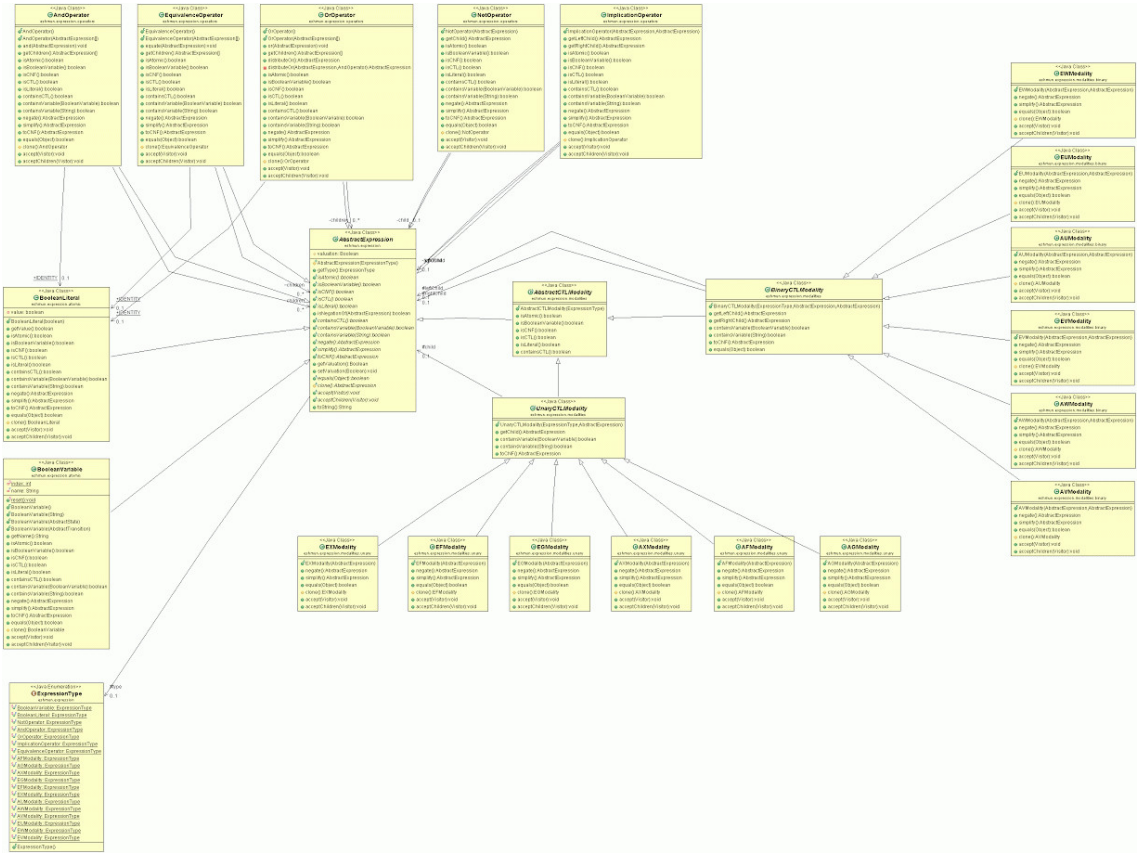

Figure 3: Tool main modules

6.4 UML Diagram for the Models Data Structures

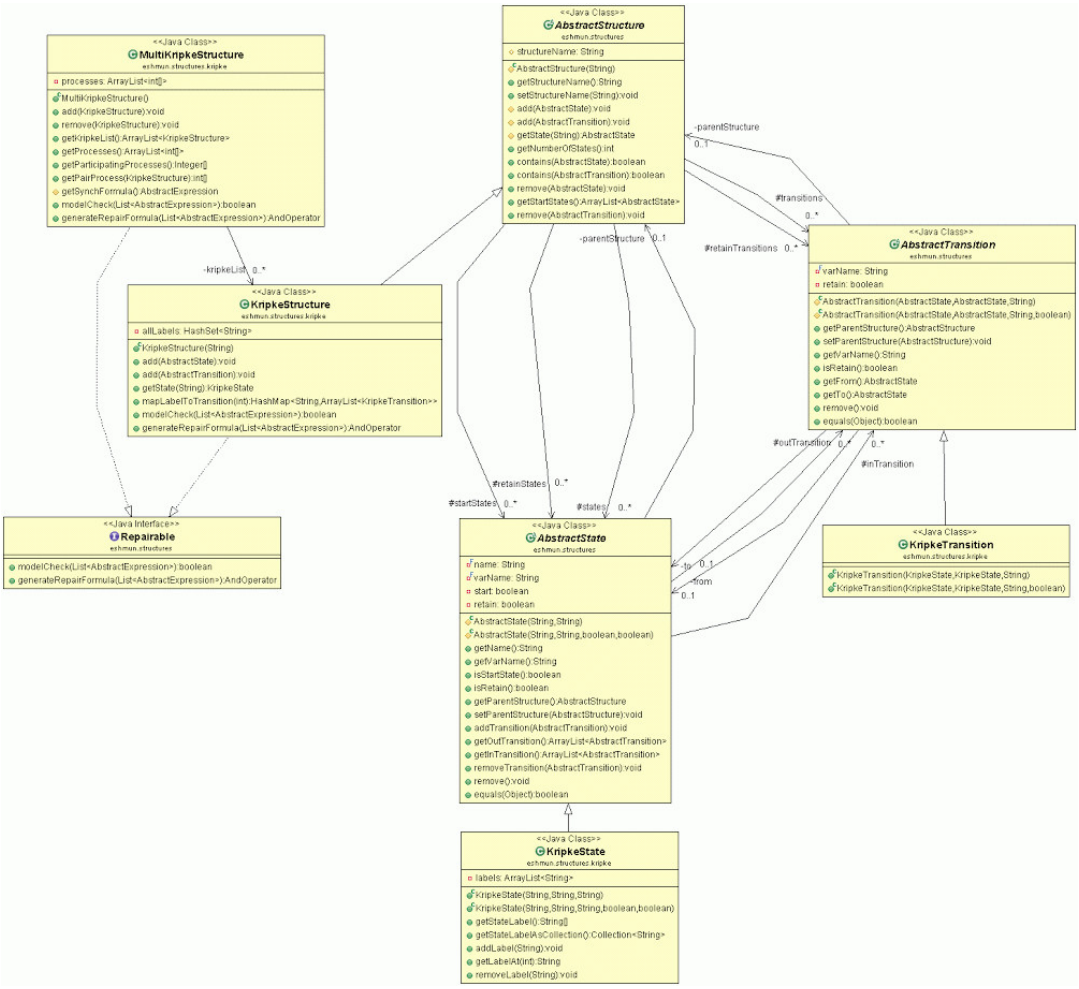

Figure 4: Tool main modules

7 Future Work

- Support more formal models for concurrent programs: I/O Automata, BIP.
- Extend logic to include ATL(Alternating Temporal Logic), which is more expressive.
- Develop an additive model repair algorithm that allows repair to be carried out by adding new states or transitions.

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

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

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

Figure 5: Formula propagation.