Explaining Verifier Traces with Explanation Based Learning

Daniel Bryce
SIFT, LLC.
dbryce@sift.net

Abstract

Model checking is an important tool for verifying that system designs will satisfy their requirements. Model checkers excel at reasoning about probabilistic systems, but can be improved when explaining counter-examples. Probabilistic counter-examples are sets of verifier traces that demonstrate how and when the probability of satisfying a system property is insufficient. Understanding a single verifier trace can be challenging; reconciling multiple verifier traces is even more difficult.

We present a new technique for explaining a set of verifier traces that applies explanation based learning. Our approach extracts causal proofs from verifier traces and then generalizes over the proofs. Generalization is achieved through encoding and solving a weighted MaxSAT problem that maps trace explanations onto a generalized explanation. We present results on our model of an infantry fighting vehicle and show that the reduction in verifier traces is typically two orders of magnitude.

Introduction

Model checkers are powerful tools for the model based design of complex systems. Checking whether system models will fulfill design requirements can provide considerable savings and prevent operational disasters. Probabilistic model checkers reason about models capturing stochastic behavior, such as race conditions, failures, and interactions with an unpredictable environment. In such models, it is often only possible to meet requirements probabilistically, and as such, properties bound the acceptable probability that requirements are met. When a property is not met, there can be multiple traces of the system's behavior whose collective probability witnesses that the acceptable probability threshold is not met. This set of traces is a probabilistic counter-example, and includes useful information about how the system model (mis)behaves. Creating causal explanations of alternative system behaviors is the primary contribution of this work.

We use a running example of a ramp subsystem of an infantry fighting vehicle (IFV), illustrated in Figure 1. In the IFV ramp model, one of the primary requirements is that the ramp will never become jammed and inoperable. Unfortunately, designing the system to satisfy this requirement with probability 1.0 is prohibitively costly. Instead, we accept a lower, yet still high, probability that the ramp will not become stuck. Existing tools are useful for verifying this property, but not for explaining why the property is violated. Prior works that explain probabilistic counter-examples, include the use of raw state-transition sequences (Aljazzar and Leue 2010), regular expressions over state transition sequences (Han, Katoen, and Berteun 2009), fault trees (Kuntz, Leitner-Fischer, and Leue 2011), or fault modes of system components (Musliner and Engstrom 2011). These approaches either under-simplify the verifier traces so that they are not easy to understand or over-simplify. We seek explanations that can both simplify verifier traces considerably and retain important causal interactions.

We describe an approach to generating explanations of verifier traces in the counter-examples with explanation based learning (EBL) (Mitchell, Keller, and Kedar-Cabelli 1986; DeJong and Mooney 1986). We extract a causal proof (Kambhampati and Kedar 1994) of the system behavior required to violate a property. With causal proofs extracted from multiple verifier traces, we construct a generalization. Unlike EBL applied to theorem proving, the generalization is more succinct than a simple disjunction of the causal proofs. Our generalization approach identifies transitions occurring in each of explanations and attempt the minimize the number of transitions required to summarize the set of explanations. We solve this generalization problem as a weighted MaxSAT instance (Borchers and Furman 1998).

Developing generalized explanations involves three main steps. We extract a causal proof from each verifier trace, reduce the proof, and then generalize over multiple proofs to create a generalized explanation. The causal proofs represent the minimal causal structure required to explain how a verifier trace violates the property. In our IFV model, the causal proofs represent less than 5% of the verifier traces. We also reduce the causal proofs to a smaller set of transitions that maintain the integrity of the proof. This reduction can sometimes be dramatic, a greater than 75% reduction over the already small proof. Finally, generalizing over reduced proofs combines many of the steps that are common to multiple causal proofs. The generalization provides a more succinct explanation that also highlights the simi-

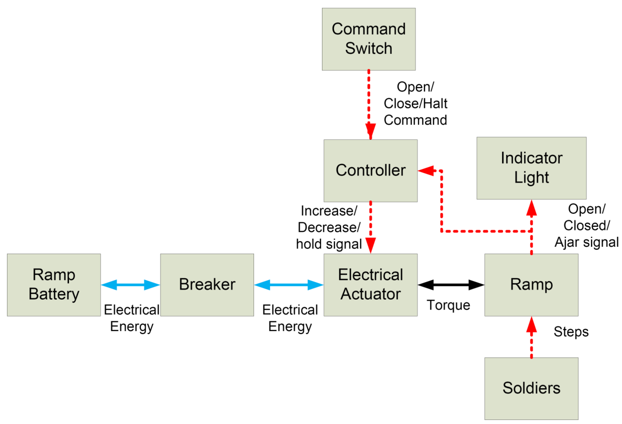

Figure 1: Infantry fighting vehicle ramp model overview.

larities between causal proofs and underscores their differences. On average, generalized explanations compress the set of reduced causal proofs by 50%. The most difficult IFV model counter-example to explain averages 34 steps per causal proof. Reducing the causal proofs results in an average of 8 steps per proof. The generalized explanation includes 14 steps (out of 32 steps across four reduced causal proofs). An explanation with fourteen steps is complex, but significantly more succinct than the ten thousand steps (1000 steps in each of 1000 verifier traces) simulated to verify the property. On average, our approach can reduce the number of transition steps among all verifier traces to a generalized explanation with two orders of magnitude fewer transition steps.

PRISM

PRISM (Hinton et al. 2006) is a model checker for probabilistic systems including discrete and continuous time Markov chains and probabilistic timed automata. In this work, we focus on discrete time Markov chains (DTMCs). The PRISM modeling language allows synchronized, component-based models (such as those needed to specify each component of the IFV model depicted in Figure 1). PRISM computes the cross product of the component models to create a single product model (e.g., a DTMC), and we discuss our algorithms in terms of this product model. In the following, we detail the DTMC product model and the probabilistic computation tree logic (PCTL) (Hansson and Jonsson 1994) properties checked by PRISM.

PRISM DTMCs: A PRISM DTMC model defines the triple (A, T, s_0) where A is a set of propositions, $S = 2^A$ is a set of states, $T : S \times S \rightarrow [0, 1]$ is a probabilistic transition relation, and $s_0 \subseteq A$ is an initial state. The transition relation is specified by a set of transition rules and the states, by a set of propositions. Each transition rule t:

$$g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$$

states that if the guard g is satisfied (i.e., $g \subseteq s$) by the current system state, then with probability λ_i a state update u_i will occur. Each guard g is a set of propositions and each update u_i is a pair of sets of propositions (u_i^+, u_i^-), added or deleted from the state. We assume that among all transitions, the guards are mutually exclusive and exhaustive.

For example, the IFV model includes a transition rule:

$$\begin{aligned}&\{mode(B1, nominal), currentLeq(B1, 8)\} \rightarrow\\&\quad 0.00013 : (\{mode(B1, failed), voltageOut(B1, 0)\},\\&\qquad\qquad \{mode(B1, nominal)\})+\\&\quad 0.99987 : (\{\}, \{\})\end{aligned}$$

which states that when battery B1 is in its nominal mode and the current drawn is no more than 8 units, then with probability 0.00013, the battery will enter its failed mode, set its output voltage to zero, and no longer be in its nominal mode. With probability 0.99987 there will be no change.

A verifier trace π is a sequence of $m + 1$ transitions $((s_0, t, u), \cdots, (s_m, t', u'))$ ending in state s_{m+1}. Each transition (s_i, t, u) results in a state $s_{i+1} = s_i \backslash u^- \cup u^+$.

PRISM Properties: PRISM checks properties specified in PCTL (Hinton et al. 2006). In this work, we discuss model checking properties specified in a limited, but highly useful, fragment of the PCTL language. We check bounded "safety" properties, of the form

$$P_{\geq P}[G^{\leq k}\psi]$$

where the $P_{\geq P}[\Phi]$ operator is a Boolean test on whether the system model will satisfy Φ with probability no less than P. The modal operator $G^{\leq k}\psi$ states that ψ must "globally" be satisfied by every state in a verifier trace with up to k transitions. For the sake of exposition, ψ is a disjunction of negated literals (i.e., $\psi = \neg a_0 \vee \ldots \vee \neg a_n$) and is satisfied by a state s if $\exists a_i . a_i \notin s$. Explaining a counter-example to ψ involves explaining how $\neg\psi = a_0 \wedge \ldots \wedge a_n$ (a conjunction of positive literals) is satisfied by each verifier trace.

For example, a desirable IFV model property ensures that the ramp will remain functional with high probability:

$$P_{\geq 0.99}[G^{\leq 1000}\neg stuck(ramp)]$$

This property asserts that (with probability no less than 0.99 over the first 1000 time steps) the ramp will not become stuck. Counter-examples illustrate how the probability that a trace will satisfy $stuck(ramp)$ is greater than 0.01.

PRISM Model Checking: PRISM uses several algorithms for model checking that are either based on statistical or analytic model checking. Both types of algorithms compute possible verifier traces to assess the probability that the safety property is satisfied. Statistical approaches typically use Monte Carlo sampling to generate verifier traces that begin in the initial state by sampling a sequence of probabilistic transitions.

The PCTL property $P_{\geq P}[\Phi]$ is satisfied if the probability that a verifier trace will satisfy Φ is no less than P. If there exists a set of verifier traces with probability greater than $1 - P$ that do not satisfy Φ, then this set of verifier traces is a probabilistic counter-example. The focus of this work is to explain these probabilistic counter-examples, as we discuss in the following section.

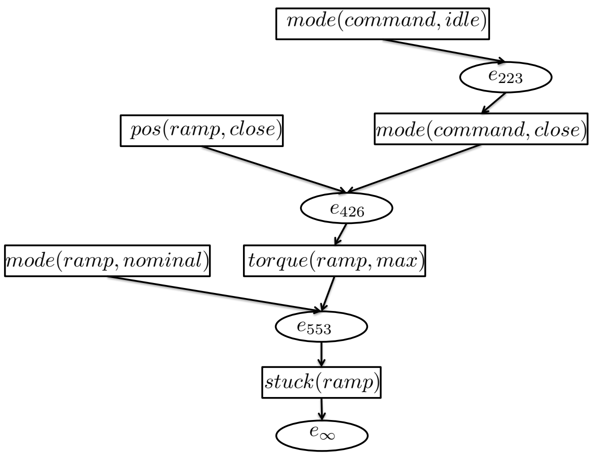

Figure 2: This causal proof is extracted from a verifier trace and explains how the ramp can become stuck. Transition steps are denoted by ovals, and state conditions by rectangles.

Explaining Counter-Examples

Counter-examples are critical to understanding and improving system models. Unfortunately, in probabilistic systems it is even more difficult to understand counter-examples because they include many verifier traces. Explaining a set of verifier traces can lead to better understanding by removing irrelevant information and generalizing over common features. Our EBL-based approach to generating probabilistic counter-example explanations accomplishes this by three steps that are detailed in the following subsections. The first step extracts a partially-ordered causal proof from each verifier trace that includes only the transitions required to violate the safety property. The second step reduces each causal proof by removing causally connected, but unnecessary, transitions. The third step generalizes over a set of causal proofs to highlight both the common and distinct transitions required by the counter-example.

Extracting Causal Proofs

We extract causal proof with a technique based upon goal regression where the negation of the safety property (our goal) is regressed. For example, our safety property $P_{\geq P}[G^{\leq 1000}\neg stuck(ramp)]$ from the previous section would result in the goal $stuck(ramp)$, and the causal proof would illustrate how $stuck(ramp)$ becomes true.

Regression involves maintaining a set of outstanding subgoals, and stepping backward through the verifier trace to identify transitions that cause subgoals to become satisfied. A transition satisfying a subgoal is added to the proof and its guard is added to the set of subgoals. In this manner the causal proof identifies which transitions and state conditions must occur in a partially ordered sequence to satisfy a property.

Causal Proofs: The causal proof c of the counter-example trace π is a triple (E, O, L), where E is a set of transition steps, O is a set of ordering relations, and L is a set of causal links. Each step $e \in E$ maps to pair (t, u) denoting a transition and update. Each $o \in O$ is a pair (e, e') indicating that e must precede e'. Each causal link $(e, a, e') \in L$ indicates that the transition and update denoted by step $e = (t, (u^+, u^-))$ cause a to become true (i.e., $a \in u^+$), a is an element of the guard of the transition denoted by $e' = (t', u')$ (i.e., $a \in g(t')$), and no other transition can occur between e and e' that will make a false. Each causal proof includes unique steps $e^0, e^\infty \in E$ which denote steps for dummy transitions whose respective update and guard denote the initial state and condition Φ. We define the set of ordering relations so that:

$$\begin{aligned} O = &\{(e^0, e) | e \in E \backslash \{e^0\}\} \cup \\ &\{(e, e^\infty) | e \in E \backslash \{e^\infty\}\} \cup \\ &\{(e, e') | (e, a, e') \in L\} \end{aligned}$$

Regression: From a counter-example trace π and property $P_{\geq P}[G^{\leq k}\psi]$, we construct a causal proof by regressing the set of propositions appearing in $\neg\psi$. In regression, we record the transitions and updates that are required to satisfy $\neg\psi$ directly or indirectly. We begin with a causal proof $(\{e^0, e^\infty\}, \{(e^0, e^\infty)\}, \{\})$ containing only the steps for the initial state and property, such that e^∞ maps to a dummy transition with the guard $g = \{a_0, \ldots, a_n\}$ where $\neg\psi = a_0 \wedge \ldots \wedge a_n$ and e^0 maps to a dummy transition and update where $u^+ = s_0$. For each step (s_i, t, u) in the verifier trace, we regress the subgoals over the transition-update pair and create a step in the causal proof if it contributes subgoals.

We initially define our goal $D_{m+1} = \{a_0, \ldots, a_n\}$, and a regression operation $reg(D_{i+1}, (s_i, t, u))$ so that:

$$reg(D_{i+1}, (s_i, t, u)) = \begin{cases} D_{i+1} \backslash u^+ \cup g(t) \colon u^+ \cap D_{i+1} \neq \emptyset \\ D_{i+1} \qquad\qquad\quad \colon otherwise \end{cases}$$

where $D_i = reg(D_{i+1}, (s_i, t, u))$ is the set of subgoals required to execute transition t (i.e., satisfy the guard $g(t)$ of t), and any subgoals not satisfied by the update u^+. We check each step in the verifier trace from the end to the beginning to update the set of goals D_i to be satisfied at each time i.

From our example (Figure 2), if $D_{1001} = \{stuck(ramp)\}$, it is not until time 554 that $D_{554} = \{stuck(ramp)\}$ is satisfied by a transition. The transition t and update u at time 553 define $g(t) = \{mode(actuator, fail), mode(command, open)\}$, and $u^+ = \{stuck(ramp)\}$. Regressing the goal over this transition leads to the subgoals $D_{553} = \{mode(actuator, fail), mode(command, open)\}$.

If at any step i, $u^+ \cap D_{i+1} \neq \emptyset$, then the step i contributes to the causal proof. Contributing steps are added to the proof as follows. We create a new step $e \in E$ that maps to the transition-update pair (t, u). For each proposition $a \in u^+ \cap D_{i+1}$, we create a causal link $(e, a, e') \in L$ such that e' is the causal proof step mapping to the earliest verifier trace step (s_j, t', u') such that $j > i$ and $a \in g(t')$.

After regressing D_{554} above, the causal proof c adds a step e_{553} for the corresponding transition-update pair, a

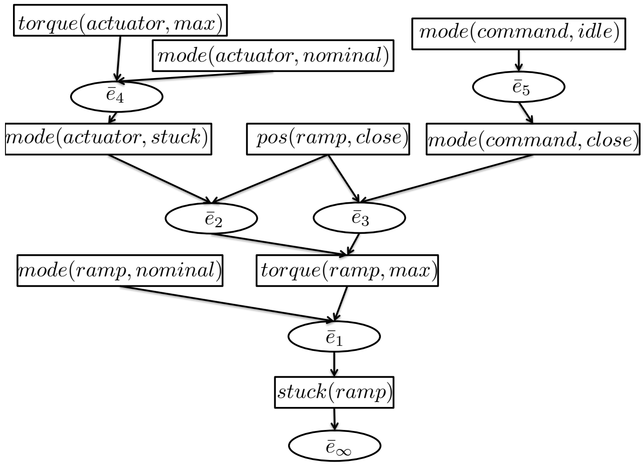

Figure 3: The generalized explanation created from two causal proofs (including the proof illustrated in Figure 2).

causal link, and ordering, so that

$$\begin{aligned} c = (&\{e^0, e^\infty, e_{553}\}, \\ &\{(e^0, e^\infty), (e_{553}, e^\infty), (e^0, e_{553})\}, \\ &\{(e_{553}, stuck(ramp), e^\infty)\}) \end{aligned}$$

Causal Proof Reduction

We can sometimes extract causal proofs that are large. Due to the nature of probabilistic systems, some proofs are large because they include repeated transitions. For example, it is possible to satisfy the subgoal $mode(command, close)$ (indicating a button was pressed to close the ramp) with a single transition from a state where $mode(command, idle)$ (c.f., step e_{223} in Figure 2). It is also possible to press the button twice, changing the command mode from idle to close, close to idle, and idle to close. Clearly, this repetition does not add to the causal proof. In such cases, a single button press transition has the same effect as two or more. Eliminating such unnecessary repetition leads to more succinct causal proofs.

In the two button press example above, we can preserve the meaning of the proof by removing the last two transitions; that is, the button press and release. We accomplish this reduction in the general case using a simple greedy algorithm (Nakhost and Müller 2010). We try removing each causal link from the proof in turn and then check whether the conditions previously satisfied by the causal link can be satisfied by another step. If so, we remove the causal link and adjust the proof to reflect the new support for the condition (a causal link from a different step); if not, we do not remove the causal link and try another. If at any time a step does not produce any causal links, then it is removed. We continue until no more causal links can be removed. In this fashion,we can often reduce the size of causal proofs considerably. Having smaller causal proofs can help to scale-up generalization and achieve more succinct generalized explanations that are more easily interpreted.

Generalized Explanations

Traditional applications of EBL generalize over multiple examples by taking the disjunction of the explanations derived from each example. The causal proofs extracted from the verifier traces are graphs. A disjunction of the graphs can be naively represented by an and-or graph with a single "or" node. We recognize that the causal proofs ("and" graphs) can share common steps, and there is an opportunity to share nodes in the graphs. We decide which nodes to join (and thus minimize the size of the generalized explanation) by encoding and solving a MaxSAT instance. We start with an example to illustrate the benefits of generalization.

Example Figures 2 and 3 illustrate a causal proof explaining a single verifier trace and a generalized explanation of two causal proofs. Both figures illustrate subgoals (corresponding to causal links) as boxes, and steps as ovals. The boxes that are not supported by steps in the figures are supported by the dummy initial step (omitted for clarity).

The generalized explanation illustrates two alternative ways to support the guard of the step $\bar{e}_1$ causing the ramp to become stuck. The first, corresponding to the causal proof in Figure 2, supports the guard condition $torque(ramp, max)$ with a step e_{426} that maps to generalized step $\bar{e}_3$. This step is supported by two conditions $pos(ramp, close)$ and $mode(command, close)$, meaning that the ramp is commanded to close when already closed. The second causal proof (not shown) supports the guard condition $torque(ramp, max)$ with a step corresponding to generalized step $\bar{e}_2$. This step is supported by two conditions in its guard $mode(actuator, stuck)$ and $pos(ramp, close)$, meaning the ramp actuator fails. As illustrated by the generalized explanation, both proofs rely on common steps and conditions, including the step $\bar{e}_1$ (where the ramp becomes stuck because of high torque) and condition $pos(ramp, close)$. Generalizing over additional causal proofs will illustrate new ways to support the existing conditions as well as introduce new steps and conditions.

Generalization as Weighted MaxSAT The generalized explanation is created by constructing a MaxSAT instance, whose optimal solution is the best mapping of the steps in causal proofs to steps in the generalized explanation. The definition of a best mapping has two components, maximizing the probability mass associated with the generalized steps and minimizing the number of generalized steps (to make the explanation more succinct). In the following, we describe our MaxSAT formulation.

Each causal proof $c_i \in C$ is a triple $c_i = (E_i, O_i, L_i)$. The generalized explanation $\bar{c}$ of a set of causal proofs C is defined similarly, $\bar{c} = (\bar{E}, \bar{O}, \bar{L})$. The generalized explanation maps each causal proof step $e_i \in E_i$ into a generalized step $\bar{e} \in \bar{E}$, such that i) if two generalized steps are ordered, all corresponding causal proof steps are ordered the same, and ii) each causal link between generalized steps exists between all corresponding causal proof steps. Both the generalized explanation and casual proofs must be internally well formed: i) there are no cycles in the ordering relations, and

$$o_{e_i,e'_i} : \forall (e_i, e'_i) \in O_i \tag{1}$$
$$(o_{e''_i,e_i} \vee o_{e'_i,e''_i}) : \forall (e_i, x, e'_i) \in L, \forall e''_i \in E_i, \text{where } e''_i \text{ clobbers } x \tag{2}$$
$$\neg o_{e_i,e_i} : \forall e_i \in E_i \tag{3}$$
$$(o_{e_i,e'_i} \wedge o_{e'_i,e''_i}) \rightarrow o_{e_i,e''_i} : \forall e_i, e'_i, e''_i \in E_i \tag{4}$$

Table 1: Clauses encoding causal proofs.

$$b_{\bar{e},e_i} \wedge b_{\bar{e}',e'_i} \rightarrow (o_{\bar{e},\bar{e}'} \leftrightarrow o_{e,e'}) : \forall \bar{e}, \bar{e}' \in \bar{E}, \forall e_i, e'_i \in E_i \tag{5}$$
$$\bigvee_{\bar{e},\bar{e}'} (b_{\bar{e},e_i} \wedge b_{\bar{e}',e'_i}) : (e_i, x, e'_i) \in L_x, L_x = \{(e_i, x, e'_i) \in L_i | c_i \in C\} \tag{6}$$
$$\neg b_{\bar{e},e_i} \vee \neg b_{\bar{e}',e_i} : \forall \bar{e}, \bar{e}' \in \bar{E}, \forall e_i \in E_i, \forall c_i \in C \tag{7}$$
$$\neg b_{\bar{e},e_i} \vee \neg b_{\bar{e},e'_i} : \forall \bar{e} \in \bar{E}, \forall e_i, e'_i \in E_i, \forall c_i \in C \tag{8}$$
$$\neg b_{\bar{e},e_i} \vee \neg b_{\bar{e},e_j} : \forall \bar{e} \in \bar{E}, e_i \in E_i, e_j \in E_j, u(e_i) \neq u(e_j) \tag{9}$$

Table 2: Clauses encoding causal proof to generalized explanation mappings.

$$w_s : \neg \bigvee_{e_i \in E_i, c \in C} b_{\bar{e},e_i} : \forall \bar{e} \in \bar{E} \tag{10}$$
$$w_t : \bigvee_{\bar{e} \in \bar{E}} b_{\bar{e},e_i} : \forall e_i \in E_i, c_i \in C \tag{11}$$

Table 3: Clauses stating optimization criteria of the MaxSAT.

ii) no step that clobbers a causal link can be ordered between steps producing and consuming the causal link.

We illustrate the formalization of these requirements through the MaxSAT formulation of the constraints. A MaxSAT instance defines a set of Boolean propositions and a set of weighted clauses (disjunctions of Boolean literals). An optimal solution is a truth assignment to propositions that maximizes the sum of the weights of satisfied clauses.

The clauses in our formulation can be grouped into three sets that: establish the consistency of causal proofs, map causal proofs to the generalized explanation, or enforce optimization criteria. Each of the clauses in the first two sets are hard, meaning that their weight is infinity and must be satisfied. The optimization clauses are soft, meaning that they should be satisfied but may not be satisfied.

The clauses that establish the consistency of a causal proof $c_i \in C$ are summarized by Table 1, and are as follows. Equation 1 states that for each ordering constraint in O_i, the ordering proposition o_{e_i,e'_i} must be true. Equation 2 states that for each causal link in a causal proof and step e''_i that assigns a different value to the variable set by x (i.e., it clobbers x), e''_i must be ordered before or after the steps in causal link. Equation 3 enforces that no step is ordered before or after itself. Equation 4 captures step order transitivity.

The clauses in Table 2 ensure that the mapping from causal proof steps to generalized steps is consistent. Equation 5 ensures that orderings between generalized steps respect the orderings between causal proof steps to which they map ($b_{\bar{e},e_i}$ denotes the mapping). Equation 6 states that two generalized steps must map to respective causal proof steps sharing a causal link. Equation 7 enforces that no causal proof step maps to more than one generalized step. Equation 8 enforces that no more than one causal proof step from the same causal proof maps to the same generalized step. Equation 9 ensures that no two dissimilar updates are represented by the same generalized step.

The clauses in Table 3 list the optimization criteria for the MaxSAT encoding. Equation 10 contributes weight w_s to the solution if there is no causal proof step mapped to a generalized step – encouraging the generalization to use as few generalized steps as possible. Equation 11 contributes weight w_t to the solution if there exists a generalized step to which a particular causal proof step is mapped – encouraging mapping causal proof steps to generalized steps. Clearly, these two criterion are conflicting; the first discourages introducing generalized steps and the second encourages mapping causal proof steps to generalized steps. We can obviate the first criterion by bounding the number of generalized steps that are included in the generalized explanation. In such a case, it may not be possible to bind all causal proof steps to a generalized explanation step and maintain explanation consistency. The weight w_t will ensure the most important steps are mapped. To capture the most important steps, we define the weight w_t of each clause by the probability that the step e_i will occur in the causal proof. In practice we use weights $w_t = \infty$ and $w_s = 1$, with a bound of fifty generalized step nodes.

The result of solving the MaxSAT is an assignment to the Boolean propositions that encodes a generalized explanation, as illustrated by the graph in Figure 3.

Property	Probability	Avg. Original Steps	Avg. Reduced Steps	Gen. Events	Extraction Time	Gen. Time	MaxSAT Vars	MaxSAT Clauses
P=? [G<=1000 "Ramp_Ramp1_FFL_nominal_p"]	0.993	34.0	8.0	14	231	18114	39353	1105111
P=? [F<=1000 Command_Command1_position_out=1]	1.000	6.0	2.5	4	6	80	973	10751
P=? [G<=1000 (Command_Command1_position_out=2)=>Ramp_Ramp1_moving=0]	0.502	26.0	2.0	3	79	33	521	4512
P=? [G<=1000 (Command_Command1_position_out=2)=>"RampBattery_Battery1_FFL_nominal_p"]	0.921	30.5	2.5	4	20	71	973	11480
P=? [G<=1000 (Command_Command1_position_out=2)=>"Ramp_Ramp1_FFL_nominal_p"]	0.996	16.8	5.0	10	38	2057	9123	221326
P=? [G<=1000 (Command_Command1_position_out=2)=>Ramp_Ramp1_position=0]	0.032	19.8	2.2	3	16	41	722	6834
P=? [F<=1000 (!IndicatorLight_Light1_ramp_open&Ramp_Ramp1_open)]	0.006	23.0	5.3	7	107	917	3927	85055
P=? [G<=1000 Ramp_Ramp1_open=>IndicatorLight_Light1_ramp_open]	0.991	27.0	4.2	9	82	1256	5470	14671
P=? [G<=1000 IndicatorLight_Light1_ramp_open=>Ramp_Ramp1_open]	1.000	-	-	-	-	-	-	-
P=? [G<=1000 IndicatorLight_Light1_ramp_open<=>Ramp_Ramp1_open]	0.988	30.2	5.0	10	62	1849	8719	197321
P=? [F<=1000 (!IndicatorLight_Light1_ramp_closed&Ramp_Ramp1_closed)]	0.013	21.0	4.8	7	48	1519	6569	150324
P=? [F<=1000 (IndicatorLight_Light1_ramp_closed&!Ramp_Ramp1_closed)]	0.000	-	-	-	-	-	-	-
P=? [G<=1000 Ramp_Ramp1_closed=>IndicatorLight_Light1_ramp_closed]	0.982	8.2	1.8	4	116	114	414	4196
P=? [G<=1000 IndicatorLight_Light1_ramp_closed=>Ramp_Ramp1_closed]	1.000	-	-	-	-	-	-	-
P=? [G<=1000 IndicatorLight_Light1_ramp_closed<=>Ramp_Ramp1_closed]	0.988	23.8	4.5	11	79	1765	7707	196905
P=? [F<=1000 (!IndicatorLight_Light1_ramp_ajar&Ramp_Ramp1_ajar)]	0.017	22.5	3.2	5	54	233	1672	29476
P=? [F<=1000 (IndicatorLight_Light1_ramp_ajar&!Ramp_Ramp1_ajar)]	0.000	-	-	-	-	-	-	-
P=? [G<=1000 Ramp_Ramp1_ajar=>IndicatorLight_Light1_ramp_ajar]	0.979	1.0	1.0	2	14	10	93	550
P=? [G<=1000 IndicatorLight_Light1_ramp_ajar<=>Ramp_Ramp1_ajar]	0.976	14.0	3.0	9	116	861	2225	63116

Figure 4: Experimental results explaining 20 properties. The results include the property, the probability that it holds, the average number of steps extracted from each trace explaining the property, the average number of reduced steps, the number of steps in the generalization, the total time (ms) to extract four causal proofs, the total time (ms) to generalize over four traces, and the number of MaxSAT variables and clauses encoding the generalization problem.

Empirical Analysis

We evaluated our approach on several properties in the IFV ramp model (summarized in Figure 4). Many of these include reachability properties (which use the operator F instead of G). The explanations in these cases are of how the property is satisfied, instead of how it is unsatisfied. To see how this is possible, we note that we can translate reachability properties to safety properties through negating them. In these cases, explaining satisfaction of a reachability property is equivalent to explaining the counter-example of a safety property. In addition, the properties are of the form $P_{=?}[\Phi]$, where we estimate the probability that they are satisfied, and thus explain the verifier traces not satisfying safety properties and those satisfying reachability properties.

In terms of scalability, the properties illustrate different scenarios where the number of verifier traces explaining a property and the number of steps per trace can vary. Figure 4 summarizes the average number of traces and steps per trace for each property. The average number of steps per casual proof can be quite large, and reducing the causal proofs can have a dramatic effect on their size. The number of generalized steps (when generalizing over proofs of four verifier traces) tends to be much smaller than the sum of the reduced number of causal proof steps, meaning that there are frequently steps in common among the causal proofs that are summarized. We also see that the extraction time and generalization time are linked to the number of causal proof steps. The size of the MaxSAT encoding also increases with the number of steps and influences the generalization time.

Conclusion

We have presented a technique for generalizing over causal proofs of multiple verifier traces to create succinct explanations of probabilistic counter-examples. The compression afforded by our techniques can make understanding verifier traces much easier.

Acknowledgements: This work was supported by US Army Research Laboratory Contract W911QX-13-C-0067.

References

Aljazzar, H., and Leue, S. 2010. Directed explicit state-space search in the generation of counterexamples for stochastic model checking. *Software Engineering, IEEE Transactions on* 36(1):37–60.

Borchers, B., and Furman, J. 1998. A two-phase exact algorithm for max-sat and weighted max-sat problems. *Journal of Combinatorial Optimization* 2(4):299–306.

DeJong, G., and Mooney, R. 1986. Explanation-based learning: An alternative view. *Machine learning* 1(2):145–176.

Han, T.; Katoen, J.-P.; and Berteun, D. 2009. Counterexample generation in probabilistic model checking. *Software Engineering, IEEE Transactions on* 35(2):241–257.

Hansson, H., and Jonsson, B. 1994. A logic for reasoning about time and reliability. *Formal aspects of computing* 6(5):512–535.

Hinton, A.; Kwiatkowska, M.; Norman, G.; and Parker, D. 2006. Prism: A tool for automatic verification of probabilistic systems. In *Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 441–444.

Kambhampati, S., and Kedar, S. 1994. A unified framework for explanation-based generalization of partially ordered and partially instantiated plans. *Artificial Intelligence* 67(1):29–70.

Kuntz, M.; Leitner-Fischer, F.; and Leue, S. 2011. From probabilistic counterexamples via causality to fault trees. *Computer Safety, Reliability, and Security* 71–84.

Mitchell, T. M.; Keller, R. M.; and Kedar-Cabelli, S. T. 1986. Explanation-based generalization: A unifying view. *Machine learning* 1(1):47–80.

Musliner, D. J., and Engstrom, E. 2011. Prismatic: Unified hierarchical probabilistic verification tool. Technical report, DTIC Document.

Nakhost, H., and Müller, M. 2010. Action elimination and plan neighborhood graph search: Two algorithms for plan improvement. In *ICAPS*, 121–128.