Proof-like Counter-Examples

Arie Gurfinkel and Marsha Chechik

Department of Computer Science, University of Toronto,
Toronto, ON M5S 3G4, Canada.
Email: {arie,chechik}@cs.toronto.edu

Abstract. Counter-examples explain why a desired temporal logic property fails to hold, and as such considered to be the most useful form of output from model-checkers. Reported explanations are typically short and described in terms of states and transitions of the model; as a result, they can be effectively used for debugging. However, counter-examples are not available for every CTL property and are often inadequate for explaining exactly what the answer means [CLJV02]. In this paper we present the approach of annotating counter-examples with additional proof steps. This approach does not sacrifice any of the advantages of traditional counter-examples, yet allows the user to understand and navigate through the counter-example better. We describe our proof system, discuss how to connect it with counter-example generators, and present KEGVis – a tool for visualizing and browsing the annotated counter-examples.

1 Introduction

A model-checker can tell the user not only whether a desired temporal property holds, but also generate a counter-example, explaining the reasons why this property failed. Typically, counter-examples are fairly small and are given in terms of states and transitions of the model; thus, they are readily understood by engineers and can be effectively used for debugging the model. The counter-example generation ability has been one of the major advantages of model-checking in comparison with other verification methods.

Counter-examples are a form of mathematical proof: to disprove that some property φ holds on all elements of some set S, it is sufficient to produce a single element $s \in S$ such that $\neg\varphi$ holds on s. For model-checking, this has two ramifications. First, counter-examples are restricted to universally-quantified formulas, and second, counter-examples have been viewed as infinite or finite *paths*, starting from the initial state, that illustrate failure of a given property. This notion of path-like counter-examples has been implemented in SMV [McM93,CGMZ95]. Yet only a subset of universally-quantified CTL (ACTL), namely ACTL $\cap$ LTL, has linear counter-examples [CLJV02]. Recent work by Clarke et. al. [CLJV02] has extended this notion to *tree-like* counter-examples. This method generates trees instead of paths as counter-examples and is complete for ACTL. For example, a counter-example for $AF(\neg y \wedge AX\neg x)$, where shaded areas indicate which subformula is being disproved, is shown in Figure 1. This example is taken from [CLJV02]. Note that tree-like counter-examples are fairly hard to understand: different parts of the counter-example correspond to different parts of the property; thus,

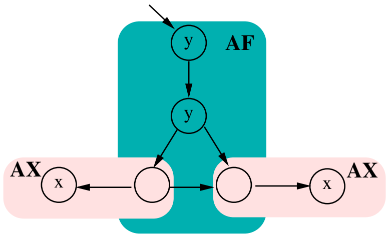

Fig. 1. Counter-example for $AF(\neg y \ \wedge \ AX\neg x)$ (from [CLJV02]).

local information is insufficient to understand what each branch is attempting to disprove. This makes it difficult to navigate to "interesting" parts of the counter-example, so the user has to understand the counter-example in its entirety.

The typical approach of existing counter-example generators is to give a complete explanation, if it is available. Yet the model-checker cannot explain why an existential property is false or why a universal one is true, and gives no feedback to the user in these cases. Such an explanation would include the entire model! Thus, in general, we say that a counter-example is not available if it is too large to be practical. Certainly, the problem gets even more difficult when temporal quantifiers are nested, e.g., $EFAX\varphi$ (is there a reachable state from which φ holds in all successors?). Counter-example generators, conventional or tree-like, do not give the user any feedback for such properties, even though counter-examples are available for parts of the formula.

The goals of the work reported in this paper are as follows: (1) to preserve the desired usability aspects of counter-examples, i.e., their short length and the close correspondence to the model; (2) to provide some feedback even if in general the counter-example is not available; (3) to help the user in *understanding* complex counter-examples.

Our results follow from the primary observation that counter-examples are simply proofs by example; yet they are the *coarsest* type of proofs which skips all steps except those that result in the transition to the next state. Additional proof steps that explain *why* the model-checker chose this sequence of states for presentation can be added as means of annotating the counter-examples. With this approach, short linear counter-examples remain intuitive, but long and bushy counter-examples become significantly easier to understand because the user can refer to the proof for the explanation. Further, when counter-examples are not available, they can be replaced by proof obligations which are either discharged by a theorem-prover or taken on faith. For example, if the model-checker determines that a property $AX\varphi$ holds, we (1) show which states are successors of the current state; (2) indicate that φ holds in these states; and (3) generate a proof obligation that the current state has no additional successors. The user might check the last claim by using a theorem-prover or simply by eye-balling the model.

In this paper we concentrate on the problem of computing witnesses to existential properties. This problem is dual to the one of computing counter-examples. We choose witnesses for our presentation because we find it more natural to talk about why a property is true as opposed to why a negation of a property is false [CGMZ95]. The rest of this paper is organized as follows: Section 2 introduces our notation and gives some background on CTL model-checking. Our approach for generating proofs for fair CTL is introduced in Section 3. The main contribution of this paper, the generation and browsing of proof-like counter-examples, is discussed and illustrated in Section 4.

Our primary goal is not to produce proofs for all possible temporal properties. In that, we differ from the work that uses model-checking for proof generation [PZ01,PPZ01, Nam01,TC02].Instead, we concentrate on annotating counter-examples with proof steps. We compare our results with related work in Section 5. Section 6 concludes the paper with the summary of our approach and venues for future work.

2 Background

We assume that the reader is familiar with the basics of model checking and CTL; this information is available in [CGP99]. Below, we recall some specific concepts and fix the notation.

We use $\top$ and $\bot$ to represent *True* and *False*, respectively. $\mathbb{B}$ and $\mathbb{N}$ are the set of boolean values $\{\top, \bot\}$ and all natural numbers, respectively. $\mu Z \cdot f(Z)$ and $\nu Z \cdot f(Z)$ are the least and the greatest fixpoints of a function f, respectively.

A Kripke structure is a tuple $M = (S, R, s_0, A, I)$ where S is a set of states; $R : S \times S \to \mathbb{B}$ is a (total) transition function; $s_0 \in S$ is an initial state, A is a set of atomic propositions; and $I : S \to 2^A$ is a (total) labeling function. CTL formulas are evaluated over infinite trees of computations produced by M . We write $[\![\varphi]\!]^M(s)$ to indicate the value of φ in state s of M. If M is clear from the context, we omit it from our notation. If a formula φ holds in the initial state, i.e. $[\![\varphi]\!](s_0) = \top$, it is considered to hold in the model.

We use EG, EX and EU as our adequate set for CTL [CGP99]. These operators are defined as follows:

$$\begin{array}{rll} EX\varphi & \triangleq \lambda s \cdot \exists t \in S \cdot R(s,t) \wedge [\![\varphi]\!](t) & \text{def. of } EX \\ EG\varphi & \triangleq \nu Z \cdot \varphi \ \wedge \ EXZ & \text{def. of } EG \\ E[\varphi\ U\ \psi] & \triangleq \mu Z \cdot \psi \vee (\varphi \ \wedge \ EXZ) & \text{def. of } EU \end{array}$$

We also explicitely define the bounded versions of EU:

$$\begin{array}{rll} E[\varphi\ U_0\ \psi] & \triangleq \psi & \text{def. of } EU_0 \\ E[\varphi\ U_i\ \psi] & \triangleq \psi \vee (\varphi \ \wedge \ EXE[\varphi\ U_{i-1}\ \psi]) & \text{def. of } EU_i \end{array}$$

Note that

$$E[\varphi\ U\ \psi] = E[\varphi\ U_\infty\ \psi]$$

The remaining operators can be computed from these, as shown in [CGP99].

A computation of M on which all of the given fairness conditions $C = \{c_1, ..., c_k\}$ occur infinitely often is called a *fair computation*. Model-checking restricted to fair computations of M is called *fair*, and the resulting language is called FCTL. This language can be obtained by providing a fair version of EG (E_CG) as follows:

$$E_CG\varphi \triangleq \nu Z \cdot \varphi \wedge \bigwedge_{i=1}^{k} EXE[\varphi U \varphi \wedge c_i \wedge Z] \quad \text{def. of } E_CG$$

Thus, $[\![E_CG\top]\!](s) = \top$ if and only if s is a starting state of some fair computation of M. Computing $E_CX\varphi$ in s amounts to restricting successors of s to the ones at the start

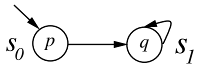

Fig. 2. A simple Kripke structure.

$$\frac{a}{a \vee b}\ \vee\text{-intro} \qquad\qquad \frac{a \quad b}{a \wedge b}\ \wedge\text{-intro}$$

$$\frac{b}{a \vee b}\ \vee\text{-intro} \qquad\qquad \frac{f(d)}{\exists x \in D \cdot f(x)}\ \text{one-point rule}$$

$$\frac{f(d_1) \qquad f(d_2) \qquad \ldots \qquad f(d_n)}{\forall d \in D \cdot f(d)}\ \text{finite quantification, with } \bigcup_{i=1}^{n}\{d_i\} = D$$

$$\frac{\forall x \in D_1 \cdot f(x) \wedge g(x) \qquad \forall x \in D \setminus D_1 \cdot \neg f(x)}{\forall x \in D \cdot f(x) \rightarrow g(x)}\ \text{universal case splitting}$$

Fig. 3. Some axioms of propositional and boolean logic.

of some fair computation, and evaluating $EX\varphi$ using only these successors. Computing $E_C U$ is similar:

$$\begin{array}{lll} E_C X\varphi & \triangleq EX(\varphi \ \wedge \ E_C G\top) & \text{def. of } E_C X \\ E_C[\varphi \ U \ \psi] & \triangleq E[\varphi \ U \ (\psi \ \wedge \ E_C G\top)] & \text{def. of } E_C U \end{array}$$

ECTL is the universal fragment of CTL where every path is existentially quantified and negation is restricted to atomic propositions. ACTL is the dual universal fragment of CTL, i.e., only universal quantification is allowed. Their fair counterparts are referred to as FECTL and FACTL, respectively.

In this paper, we use p and q to stand for arbitrary atomic propositions, s and t to represent states, and φ and ψ to represent CTL formulas. We also use the notation $\{s\}$ to express a formula that evaluates to $\top$ at state s and $\perp$ otherwise, i.e. $[\![\{s\}]\!](t) \triangleq (s = t)$. We use $\overline{\{s\}}$ for the negation of $\{s\}$.

3 Generating Proofs

In this section we develop a proof system that allows us to generate proofs for fair CTL. We start from ECTL and then extend our framework to deal with fairness, universal quantification, and negation.

3.1 Proof rules for ECTL

Our initial goal is to develop a sound and complete proof system that allows us to prove validity of sentences of the form $[\![\varphi]\!](s)$, where φ is an ECTL formula, and s is a state of a given Kripke structure M.

We assume that our proof system includes all axioms of boolean and propositional logic. Several of such axioms are shown in Figure 3. For example, in the one-point rule, also known as the $\exists$ introduction rule, f is a predicate and d is some element of D. Intuitively, the one-point rule states that to justify an existential statement $\exists x \in D \cdot f(x)$,

$$\frac{\top}{[\![\top]\!](s)} \text{ value-rule} \qquad \frac{[\![\varphi]\!](s) \quad [\![\psi]\!](s)}{[\![\varphi \wedge \psi]\!](s)} \wedge\text{-rule}$$

$$\frac{I(s,p)}{[\![p]\!](s)} \text{ atomic-rule} \qquad \frac{[\![\varphi]\!](s) \vee [\![\psi]\!](s)}{[\![\varphi \vee \psi]\!](s)} \vee\text{-rule}$$

$$\frac{\neg I(s,p)}{[\![\neg p]\!](s)} \text{ neg-atomic-rule} \qquad \frac{\exists t \in S \cdot R(s,t) \wedge [\![\varphi]\!](t)}{[\![EX\varphi]\!](s)} \; EX$$

Fig. 4. Proof rules for non-temporal operators and EX.

one simply needs to exhibit an element $d \in D$ for which $f(d)$ holds. Note also we only consider quantification over finite domains.

In addition, we assume that all axioms of the theory of Kripke structures and the axiomatization of a particular Kripke structure M are available. The latter includes statements about M's transition relation R and its labeling function I. For example, some of the axioms describing the Kripke structure in Figure 2 are:

$$\begin{array}{ll} R(s_0, s_1) = \top & I(s_0, p) = \top \\ R(s_0, s_0) = \bot & I(s_0, q) = \bot \end{array}$$

The proof rules for non-temporal operators and EX are shown in Figure 4. They follow directly from the definition of the corresponding operators. The EX-rule introduces an existential quantifier, which is typically eliminated by the one-point rule shown in Figure 3.

The proof rules for the bounded EU are given in Figure 5 and follow directly from the definition of this operator. To derive the rule for the unbounded EU, we start by noting the monotonicity of EU_i:

Proposition 1 *Let φ, ψ be ECTL formulas and $i, j \in \mathbb{N}$. Then,*

$$i \geq j \Rightarrow \forall s \in S \cdot ([\![E[\varphi \; U_i \; \psi]]\!](s) \Leftarrow [\![E[\varphi \; U_j \; \psi]]\!](s))$$

Since we assume that the state space is finite, the rule is actually bi-directional. That is, for a given Kripke structure M, there always exists a natural number n, which depends on the diameter of the directed graph induced by M, such that $E[\varphi \; U \; \psi] = E[\varphi \; U_n \; \psi]$. The proof rule for the unbounded EU is given in Figure 5.

To complete our proof system, we need to find a proof rule for EG. Unfortunately, we cannot proceed in the same manner as before and use the ECTL equivalence $EG\varphi = \varphi \wedge EXEG\varphi$. Doing so would result in a proof system which is not complete, since this proof rule can potentially be applied an infinite number of times.

Instead, note that $[\![EG\varphi]\!](s)$ is the result of evaluating $G\varphi$ on all infinite paths emanating from the state s. Moreover, since we are dealing with finite state systems, every infinite path can be decomposed into a finite (possibly empty) prefix and a finite repeating suffix. Thus, we can decompose $[\![EG\varphi]\!](s)$ into EG restricted to all non-trivial cycles around s, and EG restricted to all infinite paths that do not contain s in the future.

$$\frac{[\![\psi]\!](s)}{[\![E[\varphi\, U_0\, \psi]]\!](s)}\; EU_0 \qquad\qquad \frac{\exists n \in \mathbb{N} \cdot [\![E[\varphi\, U_n\, \psi]]\!](s)}{[\![E[\varphi\, U\, \psi]]\!](s)}\; EU$$

$$\frac{[\![\psi \vee (\varphi \wedge EXE[\varphi\, U_{i-1}\, \psi])]\!](s)}{[\![E[\varphi\, U_i\, \psi]]\!](s)}\; EU_i$$

$$\frac{[\![(\varphi \wedge EXE[\varphi\, U\, \varphi \wedge \{s\}]) \vee (\varphi \wedge EXEG(\varphi \wedge \overline{\{s\}}))]\!](s)}{[\![EG\varphi]\!](s)}\; EG$$

Fig. 5. Proof rules for EU and EG.

First, we consider the restriction of $[\![EG\varphi]\!](s)$ to all non-trivial cycles around s. Essentially, this is simply a fair-EG, where the fairness condition is given by a single formula $\{s\}$. That is, the set of non-trivial cycles around s is exactly the set of paths along which s occurs infinitely often. Furthermore, since our starting state is s, any infinite path along which s occurs infinitely often is equivalent to a finite path from s to itself. Thus, to evaluate $[\![EG\varphi]\!](s)$ restricted to cycles around s, it is sufficient to consider only finite paths from s to s. This intuition is formalized in the following theorem, the proof of which is available in [Gur02]:

Theorem 1 *Let φ be an ECTL formula and s be a state of a Kripke structure. Then,*

$$[\![EG\varphi]\!](s) = [\![(\varphi \wedge EXE[\varphi\, U\, \varphi \wedge \{s\}]) \vee (\varphi \wedge EXEG(\varphi \wedge \overline{\{s\}}))]\!](s)$$

A proof rule for EG is given in Figure 5.

Theorem 2 *The proof system for ECTL is sound and complete.*

Proof:
The proof of soundness comes from the fact that our proof rules have been derived using definitions and equivalences between ECTL operators.

To prove completeness, we show that any valid statement of the form $[\![\varphi]\!](s)$ can be proven by a finite number of applications of our proof rules. The proof proceeds on the structure of the formula φ. A proof sketch follows:

(1) Let φ be a propositional temporal formula, that is, φ does not contain EU and EG. Each rule given in Figure 4 reduces φ to its subformulas. Thus, the rest of the proof for this case proceeds by induction on the number of subformulas of φ.

(2) If in addition to (1), φ can also contain EU_i, EU_i can be removed by expanding it using rules for EU_0 and EU_i given in Figure 5.

(3) If φ can contain bounded or unbounded EU, we note that for a given Kripke structure, there exists an equivalent formula in which all unbounded EU operators are replaced by their bounded versions, reducing the resulting formula to case (2) considered above.

(4) If φ can also contain EG, i.e., $\varphi \in$ ECTL, the EG-rule reduces a formula with EG to two formulas: (a) the one with EU, handled by the above case, and (b) a new formula containing EG, where the EG operator is restricted to a subset of the state space which does not contain the current state s. Therefore, this rule can only be applied up to $|S|$ times, ensuring that a valid statement $[\![\varphi]\!](s)$ can be proven by a finite number of applications of our proof rules. □

(a)

proc `orRule`(φ, ψ, s)
  $k_\varphi :=$ `modelCheck`(φ, s)
  $k_\psi :=$ `modelCheck`(ψ, s)
  if k_φ **then**
    apply $\vee$-introduction with φ
  else if k_ψ **then**
    apply $\vee$-introduction with ψ
  else
    terminate with **invalid**
  end if
end proc

(c)

proc `exOnePoint`(φ, s, ℓ)
  $k :=$ `modelCheck`$(EX\varphi, s)$
  if not k **then**
    terminate with **invalid**
  end if
  $\hat{t} :=$ `exWitness`(φ, s)
  apply the one-point rule substituting $\hat{t}$ for t
end proc

(b)

proc `euOnePoint`(φ, ψ, s, ℓ)
  $i := 0$
  $eu :=$ `modelCheck`$(E[\varphi\ U\ \psi], s)$
  $eui = \perp$
  while $eui \neq eu$ **do**
    $eui{:=}$`modelCheck`$(E[\varphi\ U_i\ \psi], s)$
    $i := i + 1$
  end while
  if eui **then**
    apply the one-point rule substituting i for n
  else
    terminate with **invalid**
  end if
end proc

Fig. 6. Algorithms for generating proof-like witnesses.

3.2 Automatic proof generation

Given a statement $[\![\varphi]\!](s)$, we are interested in an automated proof of its validity. We can achieve this by embedding the proof system of Section 3.1 into an automated theorem prover, such as PVS [OSR93], and use its facilities for generating the proof. Yet we can do so more efficiently if we use the model-checker as a decision procedure for (a) deciding the validity of a given subformula (so that our proof generator avoids exploring irrelevant proof branches) and for (b) applying the one-point rule. We call this decision procedure `modelCheck` and assume that `modelCheck`(φ, s) computes $[\![\varphi]\!](s)$.

We start with the boolean connective $\vee$. Given a statement of the form $[\![\varphi \vee \psi]\!](s)$, we apply the $\vee$-rule:

$$\frac{[\![\varphi]\!](s) \vee [\![\psi]\!](s)}{[\![\varphi \vee \psi]\!](s)}\ \vee\text{-rule}$$

Next, we must apply the $\vee$-introduction rule, using either φ or ψ. Since the choice depends on the validity of $[\![\varphi]\!](s)$ and $[\![\psi]\!](s)$, this suggests a simple proof strategy: if $[\![\varphi]\!](s)$ is valid, apply the $\vee$-introduction rule with φ; otherwise, if $[\![\psi]\!](s)$ is valid, apply the $\vee$-introduction rule with ψ; otherwise, terminate declaring that the statement is invalid. This proof strategy is implemented by an algorithm shown in Figure 6(a).

We now examine the case of the unbounded until (EU) operator. Given the statement $[\![E[\varphi\ U\ \psi]]\!](s)$, we first apply the EU-rule:

$$\frac{\exists n \in \mathbb{N} \cdot [\![E[\varphi\ U_n\ \psi]]\!](s)}{[\![E[\varphi\ U\ \psi]]\!](s)}\ EU$$

The next step is to find an instantiation of n for the one-point rule. Recall that the bounded EU_i is monotone when viewed as a function of i (by Proposition 1). Moreover, it is bounded above by the unbounded EU. Therefore, we can find the instantiation of n by a linear search, starting from $n = 0$. The algorithm for the application of the one-point rule is given in Figure 6(b). The intermediate computations performed by this algorithm are exactly the same as the ones done by a symbolic model-checking algorithm. Thus, if the results of the intermediate computations performed by $\texttt{modelCheck}(E[\varphi\ U\ \psi], s)$ are available, a more efficient binary search can be used instead of the linear one.

For example, consider the Kripke structure in the Figure 2 and assume that we want to prove that $[\![E[p\ U\ q]]\!](s_0)$. After the application of the EU-rule, we get

$$\exists n \in \mathbb{N} \cdot [\![E[p\ U_n\ q]]\!](s_0)$$

To apply the one-point rule, we first try $[\![E[p\ U_0\ q]]\!](s_0) = [\![q]\!](s_0)$. This yields $\bot$. Increasing the bound, we get $[\![E[p\ U_1\ q]]\!](s_0) = \top$ and therefore can apply the one-point rule by instantiating n to 1.

Finally, given the statement $[\![EX\varphi]\!](s)$, we start by applying the EX rule:

$$\frac{\exists t \in S \cdot R(s,t) \wedge [\![\varphi]\!](t)}{[\![EX\varphi]\!](s)}\ EX$$

Then, we need to eliminate the existential quantifier by applying the one-point rule. First, let us define a function $\mathit{img} : S \to \mathbb{B}$ as

$$\mathit{img}(x) \triangleq R(s,x) \wedge [\![\varphi]\!](x)$$

Clearly,

$$\exists t \in S \cdot R(s,t) \wedge [\![\varphi]\!](t) \Leftrightarrow \exists t \in S \cdot \mathit{img}(t) = \top$$

Thus, to apply the one-point rule, we need to find an element $\hat{t} \in \mathit{img}^{-1}(\top)$. Unfortunately, unlike the previous cases, we cannot obtain this element by only using the model-checker as a black box. Note, however, that if a path $s, \hat{t}$ is a witness to $[\![EX\varphi]\!](s)$, then the element $\hat{t} \in \mathit{img}^{-1}(\top)$. Thus, we assume the availability of a function $\texttt{exWitness}(\varphi, s)$ which computes a witness to $[\![EX\varphi]\!](s)$. A version of this function can be easily obtained from any symbolic model-checker. The algorithm for the application of the one-point rule in the case of EX is given in Figure 6(c).

3.3 Extending to FCTL

Here we extend the results presented earlier in this section: to fair ECTL (FECTL), to ACTL, and finally to full FCTL.

Extending to ECTL with fairness Let a set of fairness conditions $C = \{c_1, ..., c_k\}$ be given. $E_C G$, a fair version of EG, is defined as a restriction of EG to paths where each fairness condition occurs infinitely often (see Section 2). This intuition is formalized below:

Theorem 3 *Let $C = \{c_1, \ldots, c_k\}$ be a set of fairness conditions, and $F_i(Z) = EX\ E[\varphi\ U\ \varphi \wedge c_i \wedge Z]$. Then,*

$$E_C G\varphi = \nu Z \cdot \varphi \wedge (F_1 \circ \ldots \circ F_k)(Z)$$

For example, if $C = \{c_1, c_2\}$, then

$$E_C G\varphi = \nu Z \cdot \varphi \wedge EXE[\varphi\ U\ \varphi \wedge c_1 \wedge EXE[\varphi\ U\ \varphi \wedge c_2 \wedge Z]]$$

The proofs of this and the remaining theorems in this paper are available in [Gur02].

The proof rule for $E_C G$ is obtained similarly to the EG-rule. We show how $[\![E_C G\varphi]\!](s)$ can be decomposed into (1) $E_C G$ restricted to fair cycles around s and (2) fair paths that do not contain s, and then use this result to define the proof rule.

Theorem 4 *Let φ be a FECTL formula, s be a state of a Kripke structure, $C = \{c_1, \ldots, c_k\}$ be a set of fairness conditions, and F_i be defined as in Theorem 3. Then*

$$[\![E_C G\varphi]\!](s) = [\![(\varphi \wedge (F_1 \circ \cdots \circ F_k)(EXE[\varphi\ U\ \varphi \wedge \{s\}])) \vee (\varphi \wedge EXE_C G(\varphi \wedge \overline{\{s\}}))]\!](s)$$

This theorem gives rise to the proof rule for $E_C G$, shown in Figure 7.

Recall from Section 2 that other FECTL operators are defined through $E_C G$, so no additional proof rules are required.

Theorem 5 *The proof system for FECTL is sound and complete.*

The proof is similar to the one for ECTL.

Extension to ACTL The ACTL subset of CTL is very similar to ECTL. The essential difference is that the EX operator in the fixpoint definition of the ECTL temporal operators is replaced by its dual AX. The proof rules are derived similarly to the ones for ECTL, and are summarized in Figure 8.

The AX-rule introduces universal quantification over the state space. Although the state-space is finite, the result is too large to present to the user explicitly and thus forms the basis of a proof obligation. Yet sometimes it is possible to reduce the complexity of such proof obligations. The application of the universal case splitting rule

$$\frac{\forall t \in S_1 \cdot R(s,t) \wedge [\![\varphi]\!](t) \qquad \forall t \in S \setminus S_1 \cdot \neg R(s,t)}{\forall t \in S \cdot R(s,t) \rightarrow [\![\varphi]\!](t)} \text{ universal case splitting}$$

tells us that we can show that φ holds for all successors of s, and no other states are successors of s. If the set S_1 is relatively small, it is demonstrated to the user via the finite quantification rule:

$$\frac{R(s,t_1) \wedge [\![\varphi]\!](t_1) \qquad \ldots \qquad R(s,t_n) \wedge [\![\varphi]\!](t_n)}{\forall t \in S_1 \cdot R(s,t) \wedge [\![\varphi]\!](t)} \text{ finite quantification, with } \bigcup_{i=1}^{n} \{t_i\} = S_1$$

and the fact that the elements of $S \setminus S_1$ are not successors of s is retained as a proof obligation.

ACTL proofs that do not involve the use of AX, such as proving $[\![A[\varphi\ U\ \psi]]\!](s)$ when $[\![\psi]\!](s)$ holds, are generated automatically.

$$\frac{[\![(\varphi \wedge (F_1 \circ \cdots \circ F_k)(EXE[\varphi\, U\, \varphi \wedge \{s\}])) \vee (\varphi \wedge EXE_C G(\varphi \wedge \overline{\{s\}}))]\!](s)}{[\![E_C G\varphi]\!](s)}\; E_C G$$

Fig. 7. A proof rule for $E_C G$.

$$\frac{\forall t \in S \cdot R(s,t) \rightarrow [\![\varphi]\!](t)}{[\![AX\varphi]\!](s)}\; AX \qquad \frac{[\![\psi]\!](s)}{[\![A[\varphi\, U_0\, \psi]]\!](s)}\; AU_0$$

$$\frac{[\![\varphi \wedge AXAG(\{s\} \vee \varphi)]\!](s)}{[\![AG\varphi]\!](s)}\; AG \qquad \frac{[\![\psi \vee (\varphi \wedge AXA[\varphi\, U_{i-1}\, \psi])]\!](s)}{[\![A[\varphi\, U_i\, \psi]]\!](s)}\; AU_i$$

$$\frac{\exists n \in \mathbb{N} \cdot [\![A[\varphi\, U_n\, \psi]]\!](s)}{[\![A[\varphi\, U\, \psi]]\!](s)}\; AU$$

Fig. 8. Proof rules for ACTL.

Extension to CTL To extend the proof rules to full CTL, we need to extend the neg-atomic-rule to temporal operators. Using the well-known dualities between ECTL and ACTL operators [CGP99], we obtain the negation rules summarized in Figure 9. Note that the negation applied to the EU operator (neg-EU rule) is handled differently. Since any adequate set for CTL must contain the EU operator [Lar95], it is not possible to express its negation as a combination of only ACTL operators.

Finally, to yield the proof system for FCTL, we combine the above proof rules with the one for $E_C G$.

4 Generating Proof-Like Counter-Examples

In this section we describe how to use the proof system introduced in Section 3 to generate annotated witnesses and counter-examples. We also discuss and illustrate the tool support for this approach.

4.1 From Proofs to Counter-Examples

There is a clear one-to-one correspondence between witnesses and proofs for CTL. Note, however, that proofs are finite, whereas witnesses can be infinite and are typically represented using a finite prefix and a finite repeating suffix. We return to this subject below.

Consider the proof for $[\![EX\varphi]\!](s)$ shown in Figure 10(a). Here, $\hat{t}$ is some state in S obtained by the model-checking procedure `exWitness`(φ, s) as part of the one-point rule for EX (see line 6 of the algorithm in Figure 6(c)). The witness can be visually obtained from the proof of EX via the following steps:

1. remove all nodes from the proof tree except for: (a) the root node, and (b) nodes that are the result of the application of the one-point rule to EX and that do not correspond to the axioms of the transition relation (see Figure 10(b));
2. replace horizontal bars by directed edges (see Figure 10(c));
3. relabel the root node by s and each node of the form $[\![\varphi]\!](\hat{t})$ by $\hat{t}$ (see Figure 10(d)).

$$\frac{\llbracket AX\neg\varphi\rrbracket(s)}{\llbracket \neg EX\varphi\rrbracket(s)}\ \text{neg-}EX \qquad \frac{\llbracket A[\top\ U\ \neg\varphi]\rrbracket(s)}{\llbracket \neg EG\varphi\rrbracket(s)}\ \text{neg-}EG$$

$$\frac{\llbracket EX\neg\varphi\rrbracket(s)}{\llbracket \neg AX\varphi\rrbracket(s)}\ \text{neg-}AX \qquad \frac{\llbracket E[\top\ U\ \neg\varphi]\rrbracket(s)}{\llbracket \neg AG\varphi\rrbracket(s)}\ \text{neg-}AG$$

$$\frac{\llbracket(\neg\psi\wedge\neg\varphi)\vee(\neg\psi\wedge\neg EXE[\varphi\wedge\overline{\{s\}}\ U\ \psi\wedge\neg\overline{\{s\}}])\rrbracket(s)}{\llbracket\neg E[\varphi\ U\ \psi]\rrbracket(s)}\ \text{neg-}EU$$

$$\frac{\llbracket EG\neg\psi\vee E[\neg\varphi\ U\ \neg\varphi\wedge\neg\psi]\rrbracket(s)}{\llbracket\neg A[\varphi\ U\ \psi]\rrbracket(s)}\ \text{neg-}AU$$

Fig. 9. Proof rules for negation.

(a)
$$\cfrac{\cfrac{R(s,\hat{t}) \qquad \llbracket\varphi\rrbracket(\hat{t})}{\exists t\in S\cdot R(s,t)\wedge\llbracket\varphi\rrbracket(t)}\ \text{one-point rule}}{\llbracket EX\varphi\rrbracket(s)}\ EX$$

(c)
$$\begin{array}{c}\llbracket\varphi\rrbracket(\hat{t})\\ \uparrow\\ \llbracket EX\varphi\rrbracket(s)\end{array}$$

(b)
$$\frac{\llbracket\varphi\rrbracket(\hat{t})}{\llbracket EX\varphi\rrbracket(s)}$$

(d)
$$\begin{array}{c}\hat{t}\\ \uparrow\\ s\end{array}$$

Fig. 10. From proofs to witnesses.

Clearly, the result of the application of the above procedure to the proof of $\llbracket EX\varphi\rrbracket(s)$ is a path through the Kripke structure that is a witness to $\llbracket EX\varphi\rrbracket(s)$. A similar procedure that uses finite quantification instead of the one-point rule also exists for AX.

Note that a sequence of steps in a Kripke structure is the only information given by the witness generators. We referred to it as the *coarsest* type of proof in Section 1. All of such steps are results of the application of the one-point rule to EX or the finite quantification rule to AX. Other proof steps do not result in state transitions and are not shown by conventional witness generators. When extracting the witness from our proofs, we label each state in a witness with these missing proof steps. Branches in the witness result from applications of $\wedge$-intro and finite quantification rules. Loops can be identified by merging states with the same label, allowing the representation of nested loops. The resulting structures are called *proof-like witnesses* or *counter-examples*.

For example, a proof-like witness for $\llbracket E[\top\ U\ q]\rrbracket(s_0)$ on the Kripke structure in Figure 2 is shown in Figure 11. In this figure, the circled nodes represent states and solid arrows between them represent state transitions. Thus, the witness for $\llbracket E[\top\ U\ q]\rrbracket(s_0)$ consists of two states: s_0 followed by s_1. Each state is labeled with the part of the proof that directly depends on it. This is indicated by the dashed *proof* arrows. Finally, each step in the proof that adds new states to the witness is labeled with the states it introduces, indicated by the dotted *one-point rule* (finite quantification rule) arrows.

As can be seen from this example, a proof-like witness is essentially a composition of two trees: the witness state tree, and the proof tree. This allows the user to either

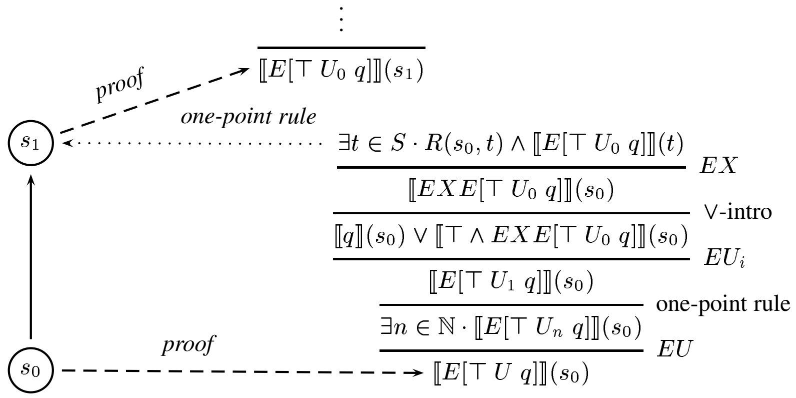

Fig. 11. Proof-like witness for $\llbracket E[\top\ U\ q]\rrbracket(s_0)$.

ignore the proof part and only explore the witness, or ignore the witness part and only explore the proof, or switch between the two representations at will.

In Section 1, we set out three goals for our work. The first goal, preserving usability aspects of witnesses, is achieved by our ability to generate conventional witnesses. The second goal, providing some feedback even if the complete counter-example is not available, is satisfied by generating proof obligations for universal properties and combining them with other proof steps so that users can understand the reasons behind these. The last goal, helping users in understanding complex counter-examples, is achieved by providing *complete* information necessary to justify the result of the model-checker. Yet proof-like counter-examples often end up being quite large. Automated support for *browsing* these counter-examples is a key for the effective use of this information for understanding and debugging the model. We address the tool support and browsing strategies below.

4.2 Tool Support for Browsing Counter-Examples

We have developed a prototype proof/witness browser tool called KEGVis (Kounter-example generator and visualizer). The tool uses the symbolic model-checker NuSMV [CCGR99] to generate proof-like witnesses for CTL formulas, and the daVinci [FW94] graph visualization package for their visual presentation. KEGVis can give the user a high-level overview of proof-like witnesses and allow him/her to skip certain steps, fast forward to "interesting" parts of the witness, determine whether the current state is part of a loop, etc.

Figure 12 lists several examples of strategies for efficient navigation through proof-like counter-examples. We distinguish between static and dynamic modes of navigation. In the former, the witness is generated prior to browsing, while in the latter, it is generated in response to user actions.

In the static mode, the exploration can proceed either in the forward direction, convenient for discovering why a particular path in a witness justifies a given subformula, or in the backward direction, used for identifying why a particular "suspicious" state is part of the witness. In both cases, the user can restrict the part of the witness of property

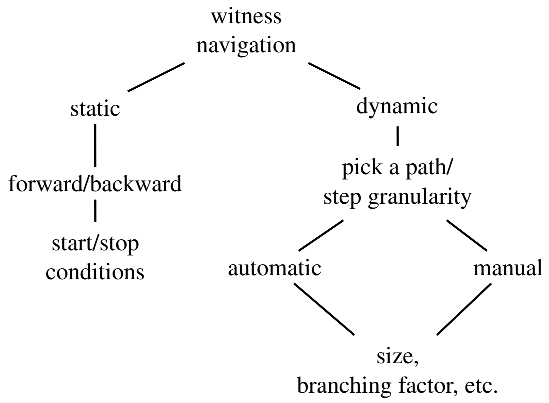

Fig. 12. Browsing strategies for proof-like witnesses/counter-examples.

φ to be displayed by selecting subformulas or operators of φ. This restriction is implemented in KEGVis by letting the user specify the initial condition, i.e. the one that should hold in the beginning of a path to explore, and the final condition, i.e. the one that tells KEGVis when to stop the exploration. For example, to restrict the witness to the EF operator of property $P = EGEF(x \wedge EXx)$, we set the initial and the final conditions to $EF(x \wedge EXx)$ and $x \wedge EXx$, respectively. Note that there are several paths satisfying the initial condition; all of them will be displayed. If a chosen subformula or operator φ occurs in multiple places in the temporal formula (e.g., x in property P above), then paths satisfying φ can be found by the following algorithm: (1) find the smallest unique subformula ψ that contains φ; (2) find all states that justify ψ; (3) for each state t found in step (2), follow the proof to a state labeled with φ. For example, the smallest unique subformula that contains the second occurrence of x in P is EXx. Thus, this x can be justified by successors of states labelled with EXx.

Witness generation is usually quite expensive. The dynamic mode of witness exploration allows the user to control which path of the witness is to be generated, or what should be the granularity of the logical step before further user input is solicited. These choices are based on the information available from the proof part of the proof-like witness, and can be taken manually or by a user-supplied heuristic function. Consider exploring the witness for the formula $[\![EFx \wedge EFy]\!](s)$. After a few steps of the exploration, the proof labeling state s may indicate that fixpoints for EFx and EFy have been achieved in 3 and 5 iterations, respectively:

$$\dfrac{\dfrac{[\![E[\top\ U_3\ x]]\!](s)}{[\![EFx]\!](s)} \qquad \dfrac{[\![E[\top\ U_5\ y]]\!](s)}{[\![EFy]\!](s)}}{[\![EFx \wedge EFy]\!](s)}\ \wedge\text{-intro}$$

The user can then use this information to guide the witness generation in the direction of the shorter path.

Current implementation of KEGVis supports static exploration only; the dynamic exploration mode is still under development.

5 Related Work

The proof system developed in this paper is similar to the tableaux used for local model-checking [SW91]. In fact, the automated proof-generation technique can be seen as a simulation of a run of a local model-checker, where the information collected from the run of a global model-checker is used to guide the construction of the proof.

Several other researches have explored the idea of generating proofs from the model-checking runs [PZ01,PPZ01,Nam01,TC02]. Their common motivation is to use proofs to provide *complete* information justifying the result of the model-checker, in particular, in cases where a witness, or a counter-example, is not feasible.

The work of Namjoshi [Nam01] concentrates on generating a proof of validity of a run of a global μ-calculus model-checker, as opposed to a proof of satisfaction of a given formula by the model. This is accomplished by augmenting the model-checker to record the set of states satisfying each subformula and the convergence bound for each fixpoint operator. In cases where the formula has a finite witness, this information can also be used to produce proof-like witnesses, using techniques similar to the one described in Section 3.2. Tan and Cleaveland [TC02] extend Namjoshi's work to local model-checking.

An approach of Peled et. al. [PZ01,PPZ01] is similar to ours in spirit. The main goal is to generate proofs that are as easy to understand as possible, and use them to communicate the reasons behind the model-checking result to the user. The technique developed in [PZ01,PPZ01] generates proofs of satisfaction for LTL properties. Note that this is equivalent to providing witnesses for ACTL; thus, the approach is complimentary to ours, as it addresses the problem of discharging proof obligations that we generate.

In this paper we have only explored how proofs can be used to aid in understanding witnesses, yet other applications are also possible. For example, Namjoshi [Nam01] suggests that one can use proofs extracted from the model-checker to debug the model-checker itself, or use them as a basis for integration between model-checking and theorem proving. Since proof-like witnesses are effectively proofs, they can be used in any application that calls for a proof of satisfaction of a temporal property.

6 Conclusion

The main contribution of this paper is the concept of *proof-like witnesses* (and counter-examples) that bridges the gap between proofs and witnesses. We have shown that expressing witnesses as proofs, in combination with an intelligent browser tool, provides a significant support for understanding complex witnesses.

In this paper, we have also developed a sound and complete proof system for CTL that serves as the basis for our approach to witness generation. This makes our approach a logical, rather than an algorithmic, one. The main advantage here is that a proof of correctness of our witness generator is significantly simplified; furthermore, the technique can be easily extended to other, non-traditional, applications such as witness generation for temporal-logic queries [GDC02].

The major limitation of our approach is that it provides only limited information in the case of witnesses for ACTL and properties that use both universal and existential quantifiers. In the future, we plan to explore possible solutions to this problem along the lines of [PPZ01]. We also plan to enhance our tool, KEGVis, to support dynamic witness exploration.

References

[CCGR99] A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new Symbolic Model Verifier. In N. Halbwachs and D. Peled, editors, *Proceedings of 11th Conference on Computer-Aided Verification (CAV'99)*, number 1633 in Lecture Notes in Computer Science, pages 495–499, Trento, Italy, July 1999. Springer.

[CGMZ95] E.M. Clarke, O. Grumberg, K.L. McMillan, and X. Zhao. Efficient Generation of Counterexamples and Witnesses in Symbolic Model Checking. In *Proceedings of 32nd Design Automation Conference (DAC 95)*, pages 427–432, San Francisco, CA, USA, 1995.

[CGP99] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[CLJV02] E.M. Clarke, Y. Lu, S. Jha, and H. Veith. Tree-Like Counterexamples in Model Checking. In *Proceedings of the Seventeenth Annual IEEE Symposium on Logic in Computer Science (LICS'02)*, pages 19–29, Copenhagen, Denmark, July 2002. IEEE Computer Society.

[FW94] M. Fröhlich and M. Werner. The Graph Visualization System daVinci — A user interface for applications. Technical Report 5/94, Department of Computer Science, Bremen University, 1994.

[GDC02] A. Gurfinkel, B. Devereux, and M. Chechik. "Model Exploration with Temporal Logic Query Checking". In *Proceedings of SIGSOFT Conference on Foundations of Software Engineering (FSE'02)*, Charleston, South Carolina, November 2002. ACM Press. (to appear).

[Gur02] A. Gurfinkel. Multi-valued symbolic model-checking: Fairness, counter-examples, running time. Master's thesis, University of Toronto, Department of Computer Science, October 2002.

[Lar95] F. Laroussinie. "About the Expressive Power of CTL Combinators". *Information Processing Letters*, 54:343–345, 1995.

[McM93] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic, 1993.

[Nam01] K. Namjoshi. Certifying Model Checkers. In *Proceedings of 13th International Conference on Computer-Aided Verification (CAV'01)*, volume 2102 of *LNCS*. Springer-Verlag, 2001.

[OSR93] S. Owre, N. Shankar, and J. Rushby. "User Guide for the PVS Specification and Verification System (Draft)". Technical report, Computer Science Lab, SRI International, Menlo Park, CA, 1993.

[PPZ01] D. Peled, A. Pnueli, and L. Zuck. From falsification to verification. In *FST&TCS*, volume 2245 of *LNCS*. Springer-Verlag, 2001.

[PZ01] D. Peled and L. Zuck. From model checking to a temporal proof. In *Proceedings of the 8th International SPIN Workshop (SPIN'2001)*, volume 2057 of *LNCS*, pages 1–14, Toronto, Canada, May 2001. Springer.

[SW91] C. Stirling and D. Walker. Local model-checking in the modal mu-calculus. *Theoretical Computer Science*, 89, 1991.

[TC02] L. Tan and R. Cleaveland. Evidence-Based Model Checking. In *Proceedings of 14th Conference on Computer-Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 455–470, Copenhagen, Denmark, July 2002. Springer-Verlag.