What Causes a System to Satisfy a Specification?

Hana Chockler IBM Research[*]
Joseph Y. Halpern Cornell University[†]
Orna Kupferman Hebrew University[‡]

Abstract

Even when a system is proven to be correct with respect to a specification, there is still a question of how complete the specification is, and whether it really covers all the behaviors of the system. *Coverage metrics* attempt to check which parts of a system are actually relevant for the verification process to succeed. Recent work on coverage in model checking suggests several coverage metrics and algorithms for finding parts of the system that are not covered by the specification. The work has already proven to be effective in practice, detecting design errors that escape early verification efforts in industrial settings. In this paper, we relate a formal definition of causality given by Halpern and Pearl [2005] to coverage. We show that it gives significant insight into unresolved issues regarding the definition of coverage and leads to potentially useful extensions of coverage. In particular, we introduce the notion of *responsibility*, which assigns to components of a system a quantitative measure of their relevance to the satisfaction of the specification.

1 Introduction

In *model checking*, we verify the correctness of a finite-state system with respect to a desired behavior by checking whether a labeled state-transition graph that models the system satisfies a specification of this behavior [Clarke, Grumberg, and Peled 1999]. An important feature of model-checking tools is their ability to provide, along with a negative answer to the correctness query, a counterexample to the satisfaction of the specification in the system. These counterexamples can be essential in detecting subtle errors in complex designs [Clarke, Grumberg, McMillan, and Zhao 1995]. On the other hand, when the answer to the correctness query is positive, most model-checking tools terminate with no further information to the user. Since a positive answer means that the system is correct with respect to the specification, this may seem to be reasonable at first glance.

In the last few years, however, there has been growing awareness that further analysis may be necessary even if a model checker reports that a specification is satisfied by a given system. The concern is that the satisfiability may be due to an error in the specification of the desired behavior or the modelling of the system, rather than being due to the correctness of the system. Two main lines of research have focused on techniques for checking such errors. One approach involves *vacuity detection*, that is, checking whether the specification is satisfied for vacuous reasons in the model [Beatty and Bryant 1994; Beer, Ben-David, Eisner, and Rodeh 1997; Kurshan 1998; Kupferman and Vardi 1999; Purandare and Somenzi 2002]. One particularly trivial reason for vacuity is that the specification is valid; perhaps more interesting are cases of antecedent failure or valid/unsatisfiable constraints in the system. For example, the branching-time

[*]Address: Haifa University, Carmel Mountains, Haifa, 31905, Israel. Email: hanac@il.ibm.com.
[†]Address: Department of Computer Science, Ithaca, NY 14853, U.S.A. Email: halpern@cs.cornell.edu.
[‡]Address: School of Engineering and Computer Science, Jerusalem 91904, Israel. Email: orna@cs.huji.ac.il

specification $AG(req \rightarrow AFgrant)$ (every request is eventually followed by a grant on every path) is satisfied vacuously in a system where requests are never sent. A specification that is satisfied vacuously is likely to point to some problems in the modelling of the system or its desired behavior.

A second approach, which is more the focus of this paper, uses what is called *coverage estimation*. Initially, coverage estimation was used in simulation-based verification techniques, where coverage metrics are used in order to reveal states that were not visited during the testing procedure (i.e, not "covered" by this procedure); see [Dill 1998; Peled 2001] for surveys. In the context of model checking, this intuition has to be modified, as the process of model checking may visit all the states of the system regardless of their relevance to the satisfaction of the specification. Intuitively, a component or a state is *covered* by a specification ψ if changing this component falsifies ψ (see [Hoskote, Kam, Ho, and Zhao 1999; Chockler, Kupferman, and Vardi 2001]). For example, if a specification requires that $AG(req \rightarrow AFgrant)$ holds at an initial state, and there is a path in which *req* holds only in one state, followed by two states both satisfying *grant*, then neither of these two states is covered by the specification (changing the truth of *grant* in either one does not render the specification untrue). On the other hand, if there is only one state on the path in which *grant* holds, then that state is covered by the specification. The intuition is that the presence of many uncovered states suggests that either the specification the user really desires has more requirements than those explicitly written (for example, perhaps the specification should really require a correspondence between the number of requests and grants), or that the system contains redundancies, and can perhaps be simplified (for example, perhaps there should be only a single grant on the path). This approach has already proven to be effective in practice, detecting design errors that escape early verification efforts in industrial settings [Hoskote, Kam, Ho, and Zhao 1999].

Intuitively, coverage considers the question of *what causes the system to satisfy the specification*. Here, we make this intuition precise by relating a formal definition of causality given by Halpern and Pearl [2005] (HP from now on) to coverage. We show that it gives significant insight into unresolved issues regarding the definition of coverage, and leads to potentially useful extensions of coverage.

The definition of causality used by HP, like other definitions of causality in the philosophy literature going back to Hume [1739], is based on *counterfactual dependence*. Essentially, event A is a cause of event B if, had A not happened (this is the counterfactual condition, since A did in fact happen) then B would not have happened. Unfortunately, this definition does not capture all the subtleties involved with causality. (If it did, there would be far fewer papers in the philosophy literature!) For example, suppose that Suzy and Billy both pick up rocks and throw them at a bottle. Suzy's rock gets there first, shattering the bottle. Since both throws are perfectly accurate, Billy's would have shattered the bottle had it not been preempted by Suzy's throw. (This story is taken due to Hall [2004].) Thus, according to the counterfactual condition, Suzy's throw is not a cause for shaterring the bottle. This problem is dealt with by HP by, roughly speaking, taking A to be a cause of B if B counterfactually depends on A under some contingency. For example, Suzy's throw is a cause of the bottle shattering because the bottle shattering counterfactually depends on Suzy's throw, under the contingency that Billy doesn't throw. It may seem that this solves one problem only to create another. While this allows Suzy's throw to be a cause of the bottle shattering, it also seems to allow Billy's throw to be a cause too.

Why do most people think that Suzy's throw is a cause and Billy's is not? Clearly, it is because Suzy's throw hit first. As is shown by HP, in a naive model that does not take into account who hit first, both Suzy's throw and Billy's throw are in fact causes. But in a more sophisticated model that can talk about the fact that Suzy's throw came first, Suzy's throw is a cause, but Billy's is not. One moral of this example is that, according to the HP definitions, whether or not A is a cause of B depends in part on the model used. Event A can be the cause of event B in one model and not in another.

Like the definitions of causality, the main definitions of coverage in the literature are inspired by counterfactual dependence. A state s is p-covered by the specification ψ if, had the value of the atomic proposition p been different in state s, then ψ would not have been true. The initial definition of coverage [Hoskote, Kam, Ho, and Zhao 1999] and its generalization [Chockler, Kupferman, and Vardi 2001] can be understood in terms of causality. The variant definition of coverage used in the algorithm proposed by Hoskote et al. [1999], which the authors say is "less formal but meets our intuitions better", can also be described as an instance of causality. In fact, the variant definition can be captured using ideas similar to those needed to deal with the Suzy-Billy story. For example, the distinction made by Hoskote et al. between the first position in which an eventuality is satisfied and later positions in which the eventuality is satisfied is similar to the distinction between Suzy, whose rock gets to the bottle first, and Billy, whose rock gets there later.

Coverage, like causality, is an all-or-nothing notion. Either A is a cause or B or it is not; similarly, either a state s is p-covered by a specification ψ or it is not. In a companion paper, Chockler and Halpern [Chockler and Halpern 2004] (CH from now on) introduced an extension of causality called *degree of responsibility* that allows us to do a more fine-grained analysis of causality. Here, we show how degree of responsibility leads to useful insights in the applied in the context of coverage as well.

To understand the notion of degree of responsibility, let us return to Suzy and Billy, and consider a scenario in which their rocks get to the bottle at exactly the same time. According to the HP definition, both Suzy and Billy are causes of the bottle shattering (for example, the bottle shattering depends counterfactually on Suzy's throw if Billy does not throw). However, there seems to be a difference between this scenario and one where Suzy is the only one who throws a rock at the bottle. We would like to say that Suzy and Billy each have some *responsibility* for the bottle being shattered if they both throw, while Suzy bears more responsibility if she is the only one that throws a rock. And if, instead of just Suzy and Billy, there are 100 children all throwing rocks at the bottle, hitting it simultaneously, we would like to say that each child is less responsible for the bottle being shattered than in the case of Suzy and Billy and their two rocks. The CH definition of responsibility captures this intution.

The notion of responsibility is quite relevant in the context of coverage. Consider for example the specification *EXp*. There seems to be a qualitative difference between a system where the initial state has 100 successors satisfying p and one where there are only two successors satisfying p. Although, in both cases, no state is p-covered by the specification, intuitively, the states that satisfy p play a more important role in the case where there are only two of them than in the case where there are 100 of them. That is, each of the two successors is more responsible for the satisfaction of EXp than each of the 100 successors.

According to the CH definition, the degree of responsibility of a state s for a specification ψ is a number between 0 and 1. A state s is covered by specification ψ iff its degree of responsibility for ψ is 1; the value of s is a cause of ψ being true if the degree of responsibility of s for ψ is positive. A degree 0 of responsibility says intuitively that s plays no role in making ψ true; a degree of responsibility strictly between 0 and 1 says that s plays some role in making ψ true, even if s by itself failing will not make ψ false. For example, if the specification is *EXp* and the initial state has two successors where p is true, then the degree of responsibility of each one for *EXp* is $1/2$; if there are one hundred successors where p is true, then the degree of responsibility of each one is $1/100$.

The issue of responsibility becomes particularly significant when one considers that an important reason that a state might be uncovered is due to fault tolerance. Here, one checks the ability of the system to cope with unexpected hardware or software faults, such as a power failure, a link failure, or a Byzantine failure [Lynch 1996]. It is often the case that fault tolerance is achieved by duplication, so that if one component fails, another can take over. Accordingly, in this analysis, redundancies in the system are welcome: a state that is covered represents a single point of failure; if there is some physical problem or software problem that involves this state, then the specification will not be satisfied. To increase fault tolerance, we want states to be uncovered. On the other hand, we still want states to somehow "carry their weight". Thus, from the point of view of fault tolerance, while having a degree of responsibility of 1 is not good, since it means a single point of failure, a degree of responsibility of $1/100$ implies perhaps unnecessary redundancy.

Another area in formal verification where the study of causality has been applied is explaining counterexamples. Groce et al. introduce a notion of *error explanation*, based on counterfactual causality [Groce, Chaki, Kroening, and Strichman]. In their framework, a counterexample represented as a trace of a program is compared to a good trace that is closest to this error trace according to some distance metric. The difference between the error trace and a good trace is considered a cause of the error. Since there can be many closest good traces, it seems that computing the degree of responsibility of each change in causing a trace to be an error can contribute to our understanding of a given error trace.

The rest of this paper is organized as follows. We review the HP definition of causality and the CH definition of responsibility in Section 2, as well as the temporal logic CTL used in model checking. The formal definitions of responsibility and causality are a little complicated. Thus, we provide somewhat simpler definitions that are appropriate for Boolean circuits. This setting arises naturally in the automata-theoretic approach to branching-time model checking, as shown in [Kupferman, Vardi, and Wolper 2000]. In Section 3 we formally relate the definitions of causality and responsibility to coverage estimation and show that various definitions of causality from the literature can be related to various definitions of coverage used in the literature. We consider complexity-theoretic issues in Section 4. For a complexity class A, $\mathrm{FP}^{\mathrm{A}[\log n]}$ consists of all functions that can be computed by a polynomial-time Turing machine with an oracle for a problem in A, which on input x asks a total of $O(\log |x|)$ queries (cf. [Papadimitriou 1984]). Eiter and Lukasiewicz [2002a] show that testing causality is Σ_2^P-complete; CH show that the problem of computing responsibility is $\mathrm{FP}^{\Sigma_2^P[\log n]}$-complete. We focus here on simpler versions of these problems that are more relevant to coverage. We show that computing the degree of responsibility for Boolean circuits is $\mathrm{FP}^{\mathrm{NP}[\log n]}$-complete. (It follows from results of Eiter and Lukasiewicz that the problem of computing causality in Boolean circuits is NP-complete.) We then consider special cases of the problem that are more tractable, and arise naturally in the context of coverage. Proofs of theorems are given in the appendix.

2 Definitions and Notation

In this section, we review the HP definition of causality and the CH definition of (degree of) responsibility. We start with an overview of the general framework of causality. Then, we argue that since models in formal verification are binary, it is sufficient to study the significantly simpler versions of causality and responsibility for binary models (see [Eiter and Lukasiewicz 2002b] for the simplification of the definition of causality for the binary case). However, we stress that the restriction to binary models is being made for expository purposes only; all the definitions make perfect sense in non-binary models. We also present the definitions of causality and responsibility for Boolean circuits and argue that binary recursive causal models are equivalent to Boolean circuits. We use Boolean circuits in our algorithms for computing responsibility in model checking and we justify this choice in Section 3.3. We define some basic concepts from model checking in Section 2.3.

2.1 The general framework of causality

In this section, we review the details of the HP definition of causality and the CH definition of responsibility.

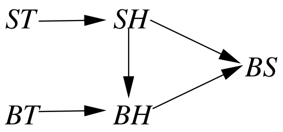

Figure 1: The rock-throwing example.

A *signature* is a tuple $\mathcal{S} = \langle \mathcal{U}, \mathcal{V}, \mathcal{R} \rangle$, where $\mathcal{U}$ is a finite set of *exogenous* variables, $\mathcal{V}$ is a set of *endogenous* variables, and the function $\mathcal{R} : \mathcal{U} \cup \mathcal{V} \rightarrow 2^{\mathcal{D}}$ associates with every variable $Y \in \mathcal{U} \cup \mathcal{V}$ a nonempty set $\mathcal{R}(Y)$ of possible values for Y from the range $\mathcal{D}$. Intuitively, the exogenous variables are ones whose values are determined by factors outside the model, while the endogenous variables are ones whose values are ultimately determined by the exogenous variables. A *causal model* over signature $\mathcal{S}$ is a tuple $M = \langle \mathcal{S}, \mathcal{F} \rangle$, where $\mathcal{F}$ is a function that associates with every endogenous variable $X \in \mathcal{V}$ a function F_X such that $F_X : (\times_{U \in \mathcal{U}} \mathcal{R}(U)) \times (\times_{Y \in \mathcal{V} \setminus \{X\}} \mathcal{R}(Y)) \rightarrow \mathcal{R}(X)$. That is, F_X describes how the value of the endogenous variable X is determined by the values of all other variables in $\mathcal{U} \cup \mathcal{V}$. If the range $\mathcal{D}$ contains only two values, we say that M is a *binary causal model*.

We can describe (some salient features of) a causal model M using a *causal network*. This is a graph with nodes corresponding to the random variables in $\mathcal{V}$ and an edge from a node labeled X to one labeled Y if the function F_Y depends on the value of X. Intuitively, variables can have a causal effect only on their descendants in the causal network; if Y is not a descendant of X, then a change in the value of X has no affect on the value of Y. For ease of exposition, we restrict attention to what are called *recursive* models. These are ones whose associated causal network is a directed acyclic graph (that is, a graph that has no cycle of edges). For a causal model M, a *context* $\vec{u}$ assigns to each exogenous variable (i.e., variable in $\mathcal{U}$) a value in the domain. If M is a recursive causal model, then given a context $\vec{u}$, the values of endogenous variables are uniquely determined by equations in $\mathcal{F}$.

The equations determined by $\{F_X : X \in \mathcal{V}\}$ can be thought of as representing processes (or mechanisms) by which values are assigned to variables. For example, if $F_X(Y, Z, U) = Y + U$ (which we usually write as $X = Y + U$), then if $Y = 3$ and $U = 2$, then $X = 5$, regardless of how Z is set. This equation also gives counterfactual information. It says that, in the context $U = 4$, if Y were 4, then X would be $u + 4$, regardless of what value X, Y, and Z actually take in the real world.

While the equations for a given problem are typically obvious, the choice of variables may not be. For example, consider the rock-throwing example from the introduction. In this case, a naive model might have an exogenous variable U that encapsulates whatever background factors cause Suzy and Billy to decide to throw the rock (the details of U do not matter, since we are interested only in the context where U's value is such that both Suzy and Billy throw), a variable *ST* for Suzy throws (*ST* $= 1$ if Suzy throws, and *ST* $= 0$ if she doesn't), a variable *BT* for Billy throws, and a variable BS for bottle shatters. In the naive model, BS is 1 if one of *ST* and *BT* is 1.

This causal model does not distinguish between Suzy and Billy's rocks hitting the bottle simultaneously and Suzy's rock hitting first. A more sophisticated model is the one that takes into account the fact that Suzy throws first. It might also include variables *SH* and *BH*, for Suzy's rock hits the bottle and Billy's rock hits the bottle. Clearly BS is 1 iff one of *BH* and *BT* is 1. However, now, *SH* is 1 if *ST* is 1, and *BH* $= 1$ if *BT* $= 1$ and *SH* $= 0$. Thus, Billy's throw hits if Billy throws *and* Suzy's rock doesn't hit. This model is described by the following graph, where there is an arrow from variable X to variable Y if the value of Y depends on the value of X. (The graph ignores the exogenous variable U, since it plays no role.)

Given a causal model $M = (\mathcal{S}, \mathcal{F})$, a (possibly empty) vector $\vec{X}$ of variables in $\mathcal{V}$, and vectors $\vec{x}$

and $\vec{u}$ of values for the variables in $\vec{X}$ and $\mathcal{U}$, respectively, we can define a new causal model denoted $M_{\vec{X}\leftarrow\vec{x}}$ over the signature $\mathcal{S}_{\vec{X}} = (\mathcal{U}, \mathcal{V} - \vec{X}, \mathcal{R}|_{\mathcal{V}-\vec{X}})$. Formally, $M_{\vec{X}\leftarrow\vec{x}} = (\mathcal{S}_{\vec{X}}, \mathcal{F}^{\vec{X}\leftarrow\vec{x}})$, where $F_Y^{\vec{X}\leftarrow\vec{x}}$ is obtained from F_Y by setting the values of the variables in $\vec{X}$ to $\vec{x}$. Intuitively, this is the causal model that results when the variables in $\vec{X}$ are set to $\vec{x}$ by some external action that affects only the variables in $\vec{X}$; we do not model the action or its causes explicitly. For example, if M is the more sophisticated model for the rock-throwing example, then $M_{ST\leftarrow 0}$ is the model where Suzy doesn't throw.

Given a signature $\mathcal{S} = (\mathcal{U}, \mathcal{V}, \mathcal{R})$, a formula of the form $X = x$, for $X \in V$ and $x \in \mathcal{R}(X)$, is called a *primitive event*. A *basic causal formula* is one of the form $[Y_1 \leftarrow y_1, \ldots, Y_k \leftarrow y_k]\varphi$, where

- φ is a Boolean combination of primitive events;
- $Y_1, \ldots, Y_k$ are distinct variables in $\mathcal{V}$; and
- $y_i \in \mathcal{R}(Y_i)$.

Such a formula is abbreviated as $[\vec{Y} \leftarrow \vec{y}]\varphi$. The special case where $k = 0$ is abbreviated as φ. Intuitively, $[Y_1 \leftarrow y_1, \ldots, Y_k \leftarrow y_k]\varphi$ says that φ holds in the counterfactual world that would arise if Y_i is set to y_i, $i = 1, \ldots, k$. A *causal formula* is a Boolean combination of basic causal formulas.

A causal formula φ is true or false in a causal model, given a context. We write $(M, \vec{u}) \models \varphi$ if φ is true in causal model M given context $\vec{u}$. $(M, \vec{u}) \models [\vec{Y} \leftarrow \vec{y}](X = x)$ if the variable X has value x in the unique (since we are dealing with recursive models) solution to the equations in $M_{\vec{Y}\leftarrow\vec{y}}$ in context $\vec{u}$ (that is, the unique vector of values for the exogenous variables that simultaneously satisfies all equations $F_Z^{\vec{Y}\leftarrow\vec{y}}$, $Z \in \mathcal{V} - \vec{Y}$, with the variables in $\mathcal{U}$ set to $\vec{u}$). We extend the definition to arbitrary causal formulas in the obvious way.

With these definitions in hand, we can give the HP definition of causality.

Definition 2.1 (Cause) *We say that $\vec{X} = \vec{x}$ is a* cause *of φ in $(M, \vec{u})$ if the following three conditions hold:*

AC1. $(M, \vec{u}) \models (\vec{X} = \vec{x}) \wedge \varphi$.

AC2. *There exist a partition $(\vec{Z}, \vec{W})$ of $\mathcal{V}$ with $\vec{X} \subseteq \vec{Z}$ and some setting $(\vec{x}', \vec{w}')$ of the variables in $(\vec{X}, \vec{W})$ such that if $(M, \vec{u}) \models Z = z^*$ for $Z \in \vec{Z}$, then*

(a) $(M, \vec{u}) \models [\vec{X} \leftarrow \vec{x}', \vec{W} \leftarrow \vec{w}']\neg\varphi$. That is, changing $(\vec{X}, \vec{W})$ from $(\vec{x}, \vec{w})$ to $(\vec{x}', \vec{w}')$ changes φ from **true** *to* **false***.*

(b) $(M, \vec{u}) \models [\vec{X} \leftarrow \vec{x}, \vec{W} \leftarrow \vec{w}', \vec{Z}' \leftarrow \vec{z}^*]\varphi$ for all subsets $\vec{Z}'$ of $\vec{Z}$. That is, setting $\vec{W}$ to $\vec{w}'$ should have no effect on φ as long as $\vec{X}$ has the value $\vec{x}$, even if all the variables in an arbitrary subset of $\vec{Z}$ are set to their original values in the context $\vec{u}$.

AC3. *$(\vec{X} = \vec{x})$ is minimal, that is, no subset of $\vec{X}$ satisfies AC2.*

AC1 just says that A cannot be a cause of B unless both A and B are true, while AC3 is a minimality condition to prevent, for example, Suzy throwing the rock and sneezing from being a cause of the bottle shattering. Eiter and Lukasiewicz [2002b] showed that one consequence of AC3 is that causes can always be taken to be single conjuncts. The core of this definition lies in AC2. Informally, the variables in $\vec{Z}$ should be thought of as describing the "active causal process" from $\vec{X}$ to φ. These are the variables that mediate between $\vec{X}$ and φ. AC2(a) is reminiscent of the traditional counterfactual criterion. However, AC2(a) is more permissive than the traditional criterion; it allows the dependence of φ on $\vec{X}$ to be tested under special *structural contingencies*, in which the variables $\vec{W}$ are held constant at some setting $\vec{w}'$. AC2(b) is an attempt to counteract the "permissiveness" of AC2(a) with regard to structural contingencies. Essentially, it ensures that $\vec{X}$ alone suffices to bring about the change from φ to $\neg\varphi$; setting $\vec{W}$ to $\vec{w}'$ merely eliminates spurious side effects that tend to mask the action of $\vec{X}$.

To understand the role of AC2(b), consider the rock-throwing example again. Looking at the simple model, it is easy to see that both Suzy and Billy are causes of the bottle shattering. Taking $\vec{Z} = \{\mathit{ST}, BS\}$, consider the structural contingency where Billy doesn't throw ($\mathit{BT} = 0$). Clearly $[\mathit{ST} \leftarrow 0, \mathit{BT} \leftarrow 0]BS = 0$ and $[\mathit{ST} \leftarrow 1, \mathit{BT} \leftarrow 0]BS = 1$ both hold, so Suzy is a cause of the bottle shattering. A symmetric argument shows that Billy is also the cause.

But now consider the model described in Figure 1. It is still the case that Suzy is a cause in this model. We can take $\vec{Z} = \{\mathit{ST}, \mathit{SH}, BS\}$ and again consider the contingency where Billy doesn't throw. However, Billy is *not* a cause of the bottle shattering. For suppose that we now take $\vec{Z} = \{\mathit{BT}, \mathit{BH}, BS\}$ and consider the contingency where Suzy doesn't throw. Clearly AC2(a) holds, since if Billy doesn't throw (under this contingency), then the bottle doesn't shatter. However, AC2(b) does not hold. Since $\mathit{BH} \in \vec{Z}$, if we set BH to 0 (it's original value), then AC2(b) requires that $[\mathit{BT} \leftarrow 1, \mathit{ST} \leftarrow 0, \mathit{BH} \leftarrow 0](BS = 1)$ hold, but it does not. Similar arguments show that no other choice of $(\vec{Z}, \vec{W})$ makes Billy's throw a cause.

The definition of responsibility refines the "all-or-nothing" concept of causality by measuring the degree of responsibility of $X = x$ for the truth value of φ in $(M, \vec{u})$. We give here only the definition of responsibility for binary models.

Definition 2.2 (Responsibility) *The* degree of responsibility *of $X = x$ for the value of φ in $(M, \vec{u})$, denoted $dr((M, \vec{u}), X = x, \varphi)$, is 0 if $X = x$ is not a cause of φ and otherwise is $1/(|\vec{W}| + 1)$, where $\vec{W} \subseteq \mathcal{V}$ is the smallest set of variables that satisfies the condition* **AC2** *in Definition 2.1.*

Thus, the degree of responsibility measures the minimal number of changes that have to be made in $\vec{u}$ in order to make φ counterfactually depend on X. If $X = x$ is not a cause of φ in $(M, \vec{u})$, then the minimal set $\vec{W}$ in Definition 2.2 is taken to have cardinality ∞, and thus the degree of responsibility of $X = x$ is 0. If φ counterfactually depends on $X = x$, then its degree of responsibility is 1. In other cases the degree of responsibility is strictly between 0 and 1. Note that $X = x$ is a cause of φ iff the degree of responsibility of $X = x$ for the value of φ is greater than 0.

2.2 Causality and responsibility in Boolean circuits

In this section, we consider an important setting in which to consider causality and responsibility: Boolean circuits. A Boolean circuit is just a representation of a propositional formula, where the leaves represent atomic propositions and the interior nodes represent the Boolean operations $\neg$, $\wedge$, and $\vee$. Given an assignment of values to the leaves, the value of the root is the value of the formula. Without loss of generality, we assume that propositional formulas are in *positive normal form*, so that negation is applied only to atomic propositions. (Converting a formula to an equivalent formula in positive normal form at most doubles the length of the formula.) Thus, in the Boolean circuit, negations occur only at the level above the leaves. We also assume without loss of generality that all $\wedge$ and $\vee$ gates in a Boolean circuit are binary.

Let $g : \{0,1\}^n \rightarrow \{0,1\}$ be a Boolean function on n variables, and let $\mathcal{C}$ be a Boolean circuit that computes g. As usual, we say that a circuit $\mathcal{C}$ is monotone if it has no negation gates. We denote by $\vec{X}$ the set of variables of $\mathcal{C}$. A truth assignment f to the set $\vec{X}$ is a function $f : \vec{X} \rightarrow \{1, 0\}$. The value of a gate w

of $\mathcal{C}$ under an assignment f is defined as the value of the function of this gate under the same assignment. Thus, we can extend the domain of f to all gates of the circuit. For an assignment f and a variable X, we denote by $\tilde{f}_X$ the truth assignment that differs from f in the value of X. Formally, $\tilde{f}_X(Y) = f(Y)$ for all $Y \neq X$, and $\tilde{f}_X(X) = \neg f(X)$. Similarly, for a set $\vec{Z} \subseteq \vec{X}$, $\tilde{f}_{\vec{Z}}$ is the truth assignment that differs from f in the values of variables in $\vec{Z}$.

It is easy to see that Boolean circuits are a special case of binary causal models, where each gate of the circuit is a variable of the model, and values of inner gates are computed based on the values of the inputs to the circuit and the Boolean functions of the gates. A context $\vec{u}$ is a setting to the input variables of the circuit. For the ease of presentation, we explicitly define the notion of *criticality* in Boolean circuits, which captures the notion of counter-factual causal dependence.

Definition 2.3 *Consider a Boolean circuit $\mathcal{C}$ over the set $\vec{X}$ of variables, an assignment f, a variable $X \in \vec{X}$, and a gate w of $\mathcal{C}$. We say that X is* critical *for w under f if $\tilde{f}_X(w) = \neg f(w)$.*

If a variable X is critical for the output gate of a circuit $\mathcal{C}$, changing the value X alone causes a change in the value of $\mathcal{C}$. If X is not critical, changing its value alone does not affect the value of $\mathcal{C}$. However, it might be the case that changing the value of X together with several other variables causes a change in the value of $\mathcal{C}$. Fortunately, the definitions of cause and responsibility can be easily re-written for Boolean circuits, where the only causal formulas we consider are the formulas of the gates.

Definition 2.4 *Consider a Boolean circuit $\mathcal{C}$ over the set $\vec{X}$ of variables, an assignment f, a variable $X \in \vec{X}$, and a gate w of $\mathcal{C}$. A (possibly empty) set $\vec{Z} \subseteq \vec{X} \setminus \{X\}$* makes X critical for w *if $\tilde{f}_{\vec{Z}}(w) = f(w)$ and X is critical for w under $\tilde{f}_{\vec{Z}}$. (The value of) X is a* cause *of (the value of) w if there is some $\vec{Z}$ that makes X critical for w.*

Similarly, we can rewrite the definition of responsibility for Boolean circuits in the following way.

Definition 2.5 (Degree of Responsibility) *Consider a Boolean circuit $\mathcal{C}$ over the set $\vec{X}$ of variables, an assignment f, a variable $X \in \vec{X}$, and a gate w of $\mathcal{C}$. The* degree of responsibility *of (the value of) X for (the value of) w under f, denoted $dr(\mathcal{C}, X, w, f)$, is $1/(1 + |\vec{Z}|)$, where $\vec{Z} \subseteq \vec{X} \setminus \{X\}$ is a set of variables of minimal size that makes X critical for w under f.*

Thus, $dr(\mathcal{C}, X, w, f)$ measures the minimal number of changes that have to be made in f in order to make X critical for w. If no subset $\vec{Z} \subseteq \vec{X} \setminus \{X\}$ makes X critical for w under f, then the minimal set $\vec{Z}$ in Definition 2.5 is taken to have cardinality ∞, and thus the degree of responsibility of X is 0. If X is critical for w under f, then its degree of responsibility is 1. In other cases the degree of responsibility is strictly between 0 and 1. We denote by $dr(\mathcal{C}, X, f)$ the degree of responsibility of X for the value of the output gate of $\mathcal{C}$. For example, if f is the assignment that gives all variables the value 1, then $dr(X_1 \vee X_2, X_1, f) = 1/2$, while $dr(\bigvee_{i=1}^{100} X_i, X_1, f) = 1/100$. For another example, consider a circuit $\mathcal{C} = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z) \vee (X \wedge U)$. That is, either two out of three variables X, Y, and Z should be assigned 1, or X and U should be assigned 1 in order for $\mathcal{C}$ to have the value 1. Consider an assignment f_1 that assigns all variables the value 1. Then, $dr(\mathcal{C}, X, f_1) = 1/3$, since changing the value of two out of three variables Y, Z, and U does not affect the value of $\mathcal{C}$, but changing the value of two out of three variables Y, Z, and U together with X falsifies $\mathcal{C}$. Now consider an assignment f_2 that assigns Y, Z, and U the value 1, and X the value 0. Clearly, changing the value of X from 0 to 1 cannot falsify $\mathcal{C}$, thus $dr(\mathcal{C}, X, f_2) = 0$. Finally, consider an assignment f_3 that assigns X and Y the value 1, and Z and U the value 0. In this case, changing the value of X alone falsifies $\mathcal{C}$, so $dr(\mathcal{C}, X, f_3) = 1$.

Remark 2.6 We note that while we define the degree of responsibility for a specific circuit, in fact its value depends solely on the Boolean function that is computed by the circuit and is insensitive to the circuit structure. Thus, degree of responsibility is a semantic notion, not a syntactic one. $\square$

2.3 The temporal logic CTL: a review

We briefly review the temporal logic CTL here; see [Emerson and Clarke 1982]. Given a set AP of atomic propositions, the set of CTL formulas over AP is the least set that includes $\mathtt{true}$, $\mathbf{false}$, and AP, and is closed under propositional connectives, the unary modal operators EX, and the binary modal operators AU and EU. We typically write AU and EU using a modified infix notation; thus, if φ and ψ are CTL formulas, then so are $\neg\varphi$, $\varphi \wedge \psi$, $EX\varphi$, $A(\varphi U \psi)$ and $E(\varphi U \psi)$. We take $AX\varphi$, $AG\varphi$, and $EG\varphi$ as abbreviations for $\neg EX\neg\varphi$, $\neg E(\mathbf{true}U\neg\varphi)$, and $\neg A(\mathbf{true}U\neg\varphi)$, respectively.

We define the semantics of CTL with respect to *Kripke structures*. A Kripke structure $K = \langle AP, W, R, w_{in}, L\rangle$ consists of a set AP of atomic propositions, a set W of states, a total transition relation $R \subseteq W \times W$, an initial state $w_{in} \in W$, and a labeling function $L : W \to 2^{AP}$. For $\langle w, w'\rangle \in R$, we say that w' is a successor of w. A *computation* of a Kripke structure is an infinite sequence $\pi = w_0, w_1, \ldots$ of states, such that for all $i \geq 0$, the state w_{i+1} is a successor of w_i. We define the *size* $|K|$ of a Kripke structure K as $|W| + |R|$. We use $K, w \models \varphi$ to indicate that a CTL formula φ holds at state w of K. When K is known from the context we omit it and write $w \models \varphi$. The relation $\models$ is inductively defined as follows.

- For all w, we have that $w \models \mathtt{true}$ and $w \not\models \mathbf{false}$.
- For an atomic proposition $p \in AP$, we have $w \models p$ iff $p \in L(w)$ and $w \models \neg p$ iff $p \notin L(w)$.
- $w \models \neg\psi$ iff $w \not\models \psi$.
- $w \models \psi \vee \varphi$ iff $w \models \psi$ or $w \models \varphi$.
- $w \models EX\psi$ iff there exists a successor w' of w such that $w' \models \psi$.
- $w \models E\psi U\varphi$ iff there exists a computation $\pi = w_0, w_1, \ldots$ such that $w_0 = w$ and there is $i \geq 0$ such that $w_i \models \varphi$ and for all $0 \leq j < i$, we have $w_j \models \psi$.
- $w \models A\psi U\varphi$ iff for all computations $\pi = w_0, w_1, \ldots$ such that $w_0 = w$, there exists $i \geq 0$ such that $w_i \models \varphi$ and for all $0 \leq j < i$, we have $w_j \models \psi$.

Let $\|\varphi\|_K = \{s \in K : (M, s)\|\varphi\|\}$ denote the set of states that satisfy φ in Kripke structure K. A Kripke structure K satisfies a formula φ, denoted $K \models \varphi$ iff φ holds in the initial state of K. The problem of determining whether K satisfies φ is the *model-checking* problem.

3 Coverage, Causality, and Responsibility in Model Checking

In this section we show how thinking in terms of causality and responsibility is useful in the study of coverage. In Section 3.1 we show that the most common definition of coverage in model checking conforms to the definition of counter-factual causality and demonstrate how the coverage information can be enhanced by the degrees of responsibility of uncovered states. In Section 3.2 we discuss other definitions of coverage that arise in the literature and in practice and describe how they fit into the framework of causality.

3.1 Coverage in the framework of causality

The following definition of coverage is perhaps the most natural one. It arises from the study of *mutant coverage* in simulation-based verification [Millo, Lipton, and Sayward 1978; Millo and Offutt 1991; Ammann and Black 2001], and is a common approach both in simulation-based verification in industry [DeMillo, Lipton, and Sayward 1978; Budd 1981; Budd and Angluin 1982; Bieman, Dreilinger, and Lin 1996; Zhu, Hall, and May 1997] and in formal verification [Hoskote, Kam, Ho, and Zhao 1999; Chockler, Kupferman, and Vardi 2001; Chockler, Kupferman, Kurshan, and Vardi 2001; Chockler and Kupferman 2002; Jayakumar, Purandare, and Somenzi 2003]. For a Kripke structure K, an atomic proposition q, and a state w, we denote by $\tilde{K}_{w,q}$ the Kripke structure obtained from K by flipping the value of q in w. Similarly, for a set of states Z, $\tilde{K}_{\vec{Z},q}$ is the Kripke structure obtained from K by flipping the value of q in all states in Z.

Definition 3.1 (Coverage) *Consider a Kripke structure K, a specification φ that is satisfied in K, and an atomic proposition $q \in AP$. A state w of K is q-*covered by *φ if $\tilde{K}_{w,q}$ does not satisfy φ.*

It is easy to see that coverage corresponds to the simple counterfactual-dependence approach to causality, where q is the only variable in $\mathcal{V}$. Indeed, a state w of K is *q-covered by φ* if φ holds in K and if q had other value in w, then φ would not have been true in K. The following example illustrates the notion of coverage and shows that the counter-factual approach to coverage misses some important insights in how the system satisfies the specification. Let K be a Kripke structure presented in Figure 2 and let $\varphi = AG(req \rightarrow AFgrant)$. It is easy to see that K satisfies φ. State w_7 is $grant$-covered by φ. On the other hand, states w_2, w_3, w_4, and w_5 are not $grant$-covered, as flipping the value of $grant$ in one of them does not falsify φ in K. Note that while the value of $grant$ in states w_2, w_3, and w_4 plays a role in the satisfaction of φ in K, the value of $grant$ in w_5 does not.

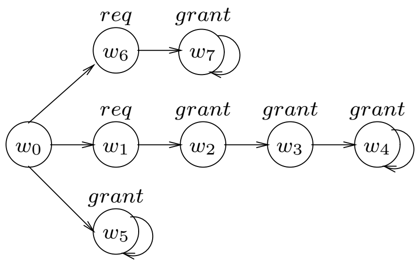

Figure 2: States w_2, w_3, and w_4 are not covered by $AG(req \rightarrow AFgrant)$, but have degree of responsibility $1/3$ for its satisfaction.

One way to capture this distinction is by using causality rather than coverage. The following definition is obtained by translating Definition 2.1 to the setting of model checking.

Definition 3.2 *Consider a Kripke structure K, a specification φ that is satisfied in K, and an atomic proposition $q \in AP$. A state w is a* cause of φ in K with respect to q *if there exists a (possibly empty) subset of states $\vec{Y}$ of K such that flipping the value of q in $\vec{Y}$ does not falsify φ in K, and flipping the value of q in both w and $\vec{Y}$ falsifies φ in K.*

In Figure 2, we describe a Kripke structure K in which the states w_2, w_3, w_4, and w_7 are causes of $AG(req \rightarrow AFgrant)$ in K with respect to grant, while w_5 is not a cause. This reflects the fact that while the value of grant is critical for the satisfaction of φ only in the state w_7, in states w_2, w_3, and w_4 the value of grant also has some effect on the value of φ in K. It does not, however, give us a quantative measure of this effect. Such a quantative measure is provided using the analogue of responsibility in the context of model checking.

Definition 3.3 *Consider a Kripke structure K, a specification φ that is satisfied in K, and an atomic proposition $q \in AP$. The* degree of q-responsibility of a state w for φ *is $1/(|\vec{Z}|+1)$, where $\vec{Z}$ is a subset of states of K of minimal size such that $\tilde{K}_{\vec{Z},q}$ satisfies φ and w is q-covered by φ in $\tilde{K}_{\vec{Z},q}$.*

In the Kripke structure described in Figure 2, states w_2, w_3, and w_4 have degree of responsibility $1/3$ for the satisfaction of $AG(req \rightarrow AFgrant)$, state w_5 has degree of responsibility 0, and state w_7 has degree of responsibility 1, all with respect to the atomic proposition $grant$.

Assigning to each state its degree of responsibility gives much more information than the yes/no answer of coverage. Coverage does not distinguish between states that are quite important for the satisfaction of the specification, even though not essential for it, and those that have very little influence on the satisfaction of the specification; responsibility can do this well. This is particularly relevant for specifications that implicitly involve disjunctions, such as formulas of the form *EX*ψ or *EF*ψ. Such specifications typically result in many uncovered states. Using responsibility gives a sense of how redundant some of these states really are. Moreover, as we observed in the introduction, any degree of redundancy in the system automatically leads to low coverage. On the other hand, for fault tolerance, we may actually want to require that no state has degree of state higher than, say, $1/3$, that is, every state should be backed up at least twice. Thus, the degree of q-responsibility of a state w for φ refines the binary definition of coverage: the smaller the degree of responsibility, the less covered the state. In the context of fault tolerance, the degree of responsibility indicates the resistance of the system to faults: the smaller the degree of responsibility of an element, the more tolerant the system is to its failure.

Remark 3.4 Definition 3.1 has been extended to capture reacher definition of coverage, like code, branch, and assertion coverage [Chockler, Kupferman, and Vardi 2003]. The extension involves the notion of *vacuity coverage*, where a state w is q-covered if $\tilde{K}_{w,q}$ does satisfy φ, but it does so in a vacuous way [Beer, Ben-David, Eisner, and Rodeh 2001; Kupferman and Vardi 2003]. The refinement of vacuity to responsibility in Definition 3.3 applies also to vacuity coverage. □

In the context of the error explanation, counterfactual causality is used to help in explaining an error [Groce, Chaki, Kroening, and Strichman]. In this context, responsibility can be used to find primary (or "root") causes of errors. For a given error trace π, the degree of responsibility of a line l in the code in causing π to falsify the specification indicates how important l is for creating an error. Lines with a low degree of responsibility are more likely to be secondary causes, that is, causes that disappear when the root causes for the error are taken care of.

3.2 Other definitions of coverage

In the previous section we showed that the definition of coverage used most often in the literature can be captured in the framework of causality. There is another definition for coverage given in [Hoskote, Kam, Ho, and Zhao 1999] that, while based on mutations, is sensitive to syntax. Thus, according to this definition, w may q-cover φ but not q-cover φ', although φ and φ' are semantically equivalent formulas. The justification for such syntactic dependencies is that the way a user chooses to write a specification carries some information. (Recall that the same issue arose in the case of Boolean circuits, although there we offered a different justification for it.) The variant definition given by Hoskote et al. [1999] has two significant advantages: it leads to an easier computational problem, and it deals to some extent with the fact that very few states are covered by eventuality formulas, which implicitly involve disjunction. Moreover, according to Hoskote et al., the definition "meets our intuitions better".

Roughly speaking, Hoskote et al.'s definition distinguishes between the first state where an eventuality is fulfilled and other states on the path. That is, if an eventuality φ is first fulfilled in a state w in the original system and is no longer fulfilled in w in the mutant system obtained by flipping the value of q in some state v, then v is said to be q-covered by φ, even if φ is still satisfied in the mutant system.

This special definition of coverage leads to an algorithm in which a specification φ is transformed to a new specification $trans_q(\varphi)$ that may include a fresh atomic proposition q'. Then, coverage is computed with respect to q' by $trans_q(\varphi)$ in the Kripke structure K' that extends K by defining q' to be true at exactly the same states as q. We do not give the full definition of $trans_q$ here (see [Hoskote, Kam, Ho, and Zhao 1999]); however, to give the intuition, we show how it works for universal until formulas. Assuming that $trans_q$ has been recursively defined for φ and ψ, let

$$trans_q(A(\varphi U \psi)) = A[trans_q(\varphi) U \psi] \wedge A[(\varphi \wedge \neg\psi) U\, trans_q(\psi)],$$

where $trans_q(q) = q'$, for some fresh atomic proposition q', and $trans_q(p) = p$ if $p \neq q$. Thus, for example, $trans_q(A(pUq)) = A(pUq) \wedge (A(p \wedge \neg q)Uq')$. It is not hard to see that if K satisfies $A(pUq)$, then Hoskote et al. say that w is q-covered by $A(pUq)$ iff w is the first state where q is true in some path in K. For example, let K be a Kripke structure that consists of a single path $\pi = w_0, w_1, w_2, \ldots$, and assume that w_0 and w_1 are the only states where p is true and that w_1 and w_2 are the only states where q is true. Then the specification $\varphi = A(pUq)$ is satisfied in K and neither w_1 nor w_2 is q-covered by φ. Note that φ is fulfilled for the first time in w_1 and that if we flip q in w_1, w_1 no longer fulfils the eventuality. Thus, w_1 is said by Hoskote et al. [1999] to be q-covered by φ.

While the intuitiveness of this interpretation of coverage is debatable, it is interesting to see that this requirement can be represented in the framework of causality. Intuitively, the eventuality being fulfilled first in w_1 is much like Suzy's rock hitting the bottle first. And just as in that example, the key to capturing the intuition is to add extra variables that describe where the eventuality is first fulfilled. Thus, we introduce two additional variables called $F1$ ("eventuality is first fulfilled in w_1") and $F2$ ("eventuality is first fulfilled in w_2"). This gives us the causal model described in Figure 3.

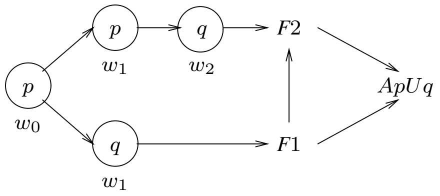

Figure 3: The cause of $A(pUq)$ being **true** in K is taken to be the first place where the eventuality is fulfilled.

Hoskote et al.'s definition of coverage for eventuality formulas can be viewed as checking whether an eventuality formula is satisfied "in the same way" in the original model and the mutant model. Only a fragment of the universal subset of CTL is dealt with by Hoskote et al., but this approach can be generalized to deal with other formulas that can be satisfied in several ways. For example, a specification $\psi = EXp$

is satisfied in a Kripke structure K if there exists at least one successor of the initial state w_0 labeled with p. If we want to check whether ψ is satisfied in a mutant structure K' in the same way it is satisfied in the original system K, we introduce a new variable X_w for each successor w of w_0 and we assign 1 to X_w iff w is labeled with p. Then we replace model checking of ψ in mutant systems by model checking of $\psi' = \bigwedge_{w \in succ(w_0)} l_w$, where l_w is X_w if $X_w = 1$ and is $\neg X_w$ otherwise. Clearly, a mutant system satisfies ψ' iff the mutation does not affect the values of p in successors of the initial state. More generally, this idea of adding extra variables to check that certain features are preserved can be used to give a more fine-grained control over what coverage is checking for.

3.3 Boolean circuits in model checking

To motivate Boolean circuits in the context of model checking, we review the automata-theoretic approach to branching-time model checking, introduced by Kupferman, Vardi, and Wolper [2000]. We focus on the branching-time logic CTL. Suppose that we want to check whether a specification φ written in branching-time temporal logic holds for a system described by a Kripke structure K. We assume that K has a special initial state denoted w_{in}. Checking if K satisfies φ amounts to checking if the model with root w_{in} obtained by "unwinding" K satisfies φ.

In the automata-theoretic approach, we transform φ to an alternating tree automaton $\mathcal{A}_\varphi$ that accepts exactly the models of φ. Checking if K satisfies φ is then reduced to checking the nonemptiness of the product $\mathcal{A}_{K,\varphi}$ of K and $\mathcal{A}_\varphi$ (where we identify K with the automaton that accepts just K). When φ is a CTL formula, the automaton $\mathcal{A}_\varphi$ is linear in the length of φ; thus, the product automaton is of size $O(|K| \cdot |\varphi|)$.

Let W be the set of states in K and let AP be the set of atomic propositions appearing in φ. The product automaton $\mathcal{A}_{K,\varphi}$ can be viewed as a graph $G_{K,\varphi}$. The interior nodes of $G_{K,\varphi}$ are pairs $\langle w, \psi \rangle$, where $w \in W$ and ψ is a subformula of φ that is not an atomic proposition. The root of $G_{K,\varphi}$ is the vertex $\langle w_{in}, \varphi \rangle$. The leaves of $G_{K,\varphi}$ are pairs $\langle w, p \rangle$ or $\langle w, \neg p \rangle$, where $w \in W$ and $p \in AP$. As shown by Chockler, Kupferman, and Vardi [2001] (CKV from now on), we can assume that each interior node $\langle w, \psi \rangle$ has two successors, and is classified according to the type of ψ as an OR-node or an AND-node. Each leaf $\langle w, p \rangle$ or $\langle w, \neg p \rangle$ has a value, 1 or 0, depending on whether p is in the label of state w in the model K. The graph has at most $2 \cdot |AP| \cdot |W|$ leaves. We present a simple example of such a graph in Figure 4.

We would like to view the graph $G_{K,\varphi}$ as a Boolean circuit. To do this, we first replace each node labeled $\langle w, \neg p \rangle$ by a NOT-node, and add an edge from the leaf $\langle w, p \rangle$ to the NOT-node. Clearly this does not increase the size of the graph. The only thing that now prevents $G_{K,\varphi}$ from being a Boolean circuit is that it may have cycles. However, as shown by Kupferman, Vardi, and Wolper [2000], the cycles in $G_{K,\varphi}$ contain only vertices of one type—either OR or AND. Thus, each cycle can be "collapsed" into one node with many successors; this node can then be replaced by a tree, where each node has two successors. The size of the resulting graph is still $O(|K| \cdot |\varphi|)$. Model checking is equivalent to finding the value of the root of $G_{K,\varphi}$ given the values of the leaves. That is, model checking reduces to evaluating a Boolean circuit. The following result is straightforward, given the definitions.

Proposition 3.5 *Consider a Kripke structure K, a specification φ, and an atomic proposition q. The following are equivalent:*

(a) the degree of q-responsibility of w for φ is $1/k$;

(b) the node (w, q) has degree of responsibility $1/k$ for (w_{in}, φ) in the Boolean circuit corresponding to K and φ;

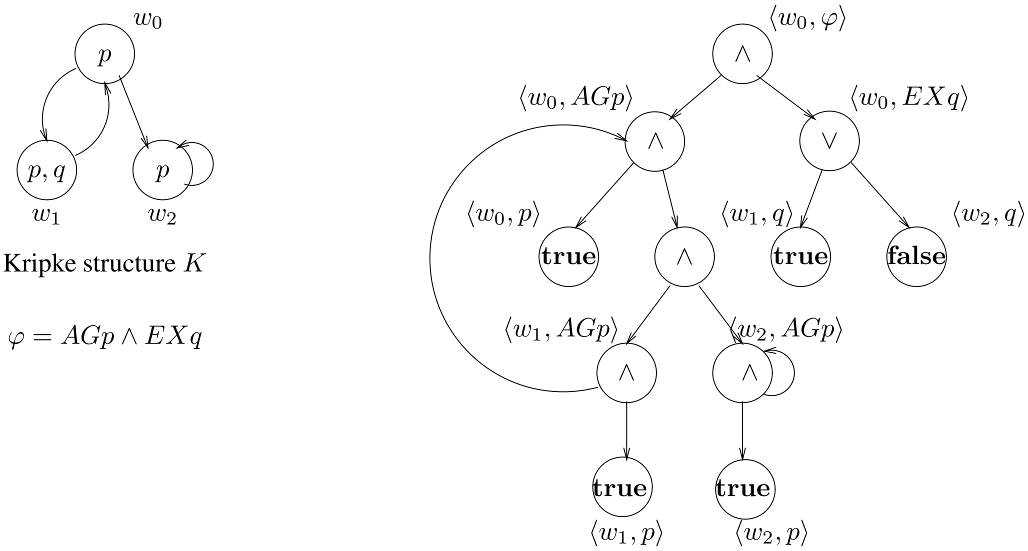

Figure 4: Construction of the product automaton.

module example(o_1, o_2, o_3);
    reg o_1, o_2, o_3;
    initial begin
        $o_1 = o_2 = o_3 = 0$;
    end
    always @(posedge clk) begin
        assign $o1 = o1$;
        assign $o2 = o2 \mid o3$;
        assign $o3 = \neg o3$;
    end
endmodule

Figure 5: The Verilog description of the design.

(c) $X_{w,q}$ has degree of responsibility $1/k$ for the output in the causal model corresponding to K and φ.

It is almost immediate from Proposition 3.5 that w is q-covered by φ in the Kripke structure K iff (w, q) is critical (i.e., has degree of responsibility 1) for the value of (w_{in}, φ) in the Boolean circuit iff $X_{w,q}$ has degree of responsibility 1 for the value of the output in the causal model.

3.4 A Detailed Example

In this section we demonstrate how responsibility can be used in model checking. The design is described as a part of a Verilog program, a finite state machine (FSM), and a circuit. The design has three registers, which encode the state space. The design is described by means of a Verilog program in Figure 5. Figure 6 shows the FSM (each state is labeled by the truth values of o_1, o_2, and o_3) and the circuit for this design.

First, it is easy to see that all states in which $o_1 = 1$ are unreachable, and thus have responsibility 0 for

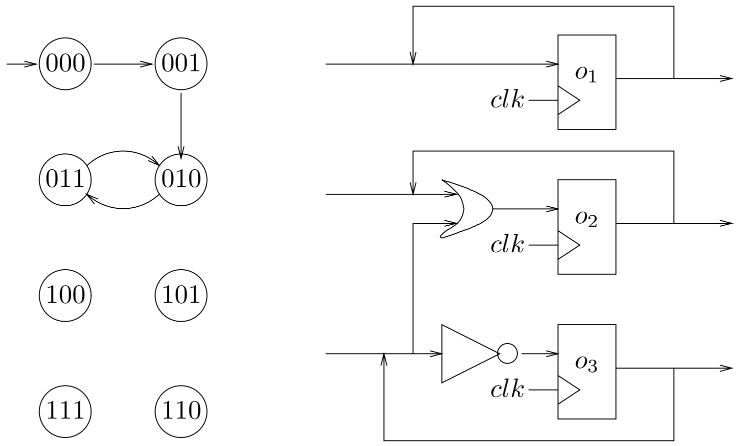

Figure 6: The FSM with three latches.

the satisfaction of all specifications that are satisfied in the design. We consider coverage and responsibility with respect to several specifications. We start with the specification $\varphi = AG\neg o_1$. That is, φ requires that the value of o_1 is 0 in all reachable states. It is easy to see that all reachable states of the design are covered with respect to φ and mutations of o_1. Accordingly, the value of o_1 in all reachable states has responsibility 1 for the satisfaction of φ in the design. Also, no reachable state is covered with respect to φ and mutations of o_2 or o_3. Accordingly, the values of o_2 and o_3 in all states have responsibility 0 for the satisfaction of φ in the design.

Next consider the specification $\varphi' = AFo_3$. That is, φ' requires that the value of o_3 is 1 in at least one state at each path of the design that starts in state 000. Note that the value of o_3 is 1 in states 001 and 011. Thus, since more than one state can satisfy the eventuality, neither 001 nor 011 is covered with respect to φ' and mutations of o_3 (in fact, there are no covered states for this specification). Responsibility provides a finer measure, showing that the value of o_3 in the states 001 and 011 are responsible for the satisfaction of φ', while the values of other registers and the value of o_3 in other states do not have any responsibility. The responsibility of the value of o_3 in each one of the states 001 and 011 for the satisfaction of φ' is $1/2$, because we need to change the value of only one register in one state in order to create a counterfactual dependence. For example, changing the value of o_3 in 001 from 1 to 0 creates a counterfactual dependence of the satisfaction of φ' on the value of o_2 in the state 011.

The third specification is $\varphi'' = AXAG(o_2 \vee o_3)$. That is, all states reachable from a successor of 000 are such that either o_2 or o_3 have the value 1. While the states 001 and 010 are covered with respect to φ'' and mutations of o_3 and o_2, respectively, the state 011 is not covered. Indeed, both o_2 and o_3 can change their value (separately) without falsifying the specification. Again, computing the degree of responsibility enables us to distinguish between the values of o_2 and o_3 in state 011, which have responsibility $1/2$ for the satisfaction of φ'', and the values of all registers in state 000, which have responsibility 0.

4 Computing the Degree of Responsibility in Binary Causal Models

In this section we examine the complexity of computing the degree of responsibility. We start with the complexity result for the general case of binary causal models. Then we discuss several special cases for which the complexity of computing responsibility is much lower and is feasible for practical applications.

4.1 The general case

For a complexity class A, FP$^{\mathrm{A}[\log n]}$ consists of all functions that can be computed by a polynomial-time Turing machine with an oracle for a problem in A, which on input x asks a total of $O(\log |x|)$ queries (cf. [Papadimitriou 1984]). Eiter and Lukasiewicz [2002a] show that testing causality is Σ_2^P-complete; CH show that the problem of computing responsibility is FP$^{\Sigma_2^P[\log n]}$-complete for general causal models. Eiter and Lukasiewicz showed that in binary causal models, computing causality is NP-complete. Since the causal model corresponding to a Boolean circuit is binary, computing causality is NP-complete in Boolean circuits. We show that computing the degree of responsibility is FP$^{\mathrm{NP}[\log n]}$-complete in binary causal models. We actually prove the FP$^{\mathrm{NP}[\log n]}$-completeness first for Boolean circuits. Then we show that a slight extension of our argument can be used to prove the same complexity result for all binary causal models.

Formally, the problem RESP-CIRCUIT is defined as follows: given a circuit $\mathcal{C}$ over the set of variables $\vec{X}$, a variable $X \in \vec{X}$, and a truth assignment f, compute $dr(\mathcal{C}, X, f)$. We prove the following theorem.

Theorem 4.1 *RESP-CIRCUIT is FP*$^{\mathrm{NP}[\log n]}$*-complete.*

The proofs of Theorem 4.1 and its easy extension below can be found in Appendix A.

Theorem 4.2 *Computing the degree of responsibility is FP*$^{\mathrm{NP}[\log n]}$*-complete in binary causal models.*

By Proposition 3.5, the upper bound in Theorem 4.1 applies immediately to computing the degree of responsibilty of a state w for a formula φ. The lower bound also applies to model checking, since it is not hard to show that for every Boolean function f over the set of variables $\vec{X}$ and assignment $\vec{x}$ there exists a pair $\langle K, \varphi \rangle$ such that K is a Kripke structure, φ is a specification, and model checking of φ in K amounts to evaluating a circuit $\mathcal{C}$ that computes f under the assignment $\vec{x}$. Indeed, let K be a single-state structure with a self-loop over the set $\vec{X}$ of atomic propositions, where the single state of K is labeled with $X \in \vec{X}$ iff X is 1 under the assignment $\vec{x}$. Let φ be a propositional formula over the set of variables $\vec{X}$ that computes the function f. Then the graph $G_{K,\varphi}$ is a circuit that computes f and evaluating $G_{K,\varphi}$ is equivalent to evaluating f under the assignment $\vec{x}$.

4.2 Tractable special cases

Theorem 4.1 shows that there is little hope of finding a polynomial-time algorithm for computing the degree of responsibility for general circuits. The situation may not be so hopeless in practice. For one thing, we are typically not interested in the exact degree of responsibility of a node, but rather want a report of all the nodes that have low degree of responsibility. This is the analogue of getting a report of the nodes that are not covered, which is the goal of algorithms for coverage. As in the case of coverage, the existence of nodes that have a low degree of responsibility suggests either a problem with the specification or unnecessary redundancies in the system.

Clearly, for any fixed k, the problem of deciding whether $dr(\mathcal{C}, X, w, f) \geq 1/k$ can be solved in time $O(|\vec{X}|^k)$ by the naive algorithm that simply checks whether X is critical for $\mathcal{C}$ under the assignment $\tilde{f}_{\vec{Z}}$ for all possible sets $\vec{Z} \subseteq \vec{X}$ of size at most $k-1$. The test itself can clearly be done in linear time. We believe that, as in the case of coverage, where the naive algorithm can be improved by an algorithm that exploits the fact that we check many small variants of the same Kripke structure [Chockler, Kupferman, and Vardi 2001], there are algorithms that are even more efficient. In any case, this shows that for values of k like 2 or 3, which are perhaps of most interest in practice, computing responsibility is quite feasible.

There is also a natural restriction on circuits that allows a linear-time algorithm for responsibility. We say that a Boolean formula φ is *read-once* if each variable appears in φ only once. Clearly, a Boolean circuit for a read-once formula is a tree. While only a small fraction of specifications are read-once, every formula can be converted to a read-once formula simply by replacing every occurrence of an atomic proposition by a new atomic proposition. For example, $\psi = (p \wedge q) \vee (p \wedge r)$ can be converted to $\psi' = (p_0 \wedge q) \vee (p_1 \wedge r)$. Given an assignment for the original formula, there is a corresponding assignment for the converted formula that gives each instance of an atomic proposition the same truth value. While this does not change the truth value of the formula, it does change responsibility and causality. For example, under the assignment that gives every atomic proposition the value 1, p is critical for ψ and thus has responsibility 1 for the value of ψ, while under the corresponding assignment, p_0 has responsibility only $1/2$ for ψ'. Similarly, p is not a cause of the value of $p \vee \neg p$ under the assignment that gives value 1 to p, but p_0 is cause of the value of $p_0 \vee \neg p_1$ under the corresponding assignment.

If we think of each occurrence of an atomic proposition as being "handled" by a different process, then as far as fault tolerance goes, the converted formula is actually a more reasonable model of the situation. The conversion models the fact that each occurrence of p in ψ can then fail "independently". This observation shows exactly why different models may be appropriate to capture causality. Interestingly, this type of conversion is also used in vacuity detection in [Beer, Ben-David, Eisner, and Rodeh 1997; Kupferman and Vardi 1999; Purandare and Somenzi 2002], where each atomic proposition is assumed to have a single occurrence in the formula.

In model checking, we can convert a Boolean circuit obtained from the product of a system K with a specification φ to a read-once tree by unwinding the circuit into a tree. This results in a degree of responsibility assigned to each occurrence of a pair $\langle w, \psi \rangle$, and indeed each pair may occur several times. The way one should interpret the result is then different than the interpretation for the Boolean-circuit case and has the flavor of *node coverage* introduced by Chockler et al. [2001]. Essentially, in node coverage, one measures the effect of flipping the value of an atomic proposition in a single occurrence of a state in the infinite tree obtained by unwinding the system.

The general problem of vacuity detection for branching-time specifications is co-NP-complete; the problem is polynomial for read-once formulas [Kupferman and Vardi 1999]. Considering read-once formulas also greatly simplifies computing the degree of responsibility. To prove this, we first need the following property of monotone Boolean circuits.

Lemma 4.3 *Given a monotone Boolean circuit $\mathcal{C}$ over the set $\vec{X}$ of variables, a variable $X \in \vec{X}$, a gate $w \in \mathcal{C}$, and an assignment f, if $f(w) \neq f(X)$, then $dr(\mathcal{C}, X, w, f) = 0$.*

Proof: Both functions $\wedge$ and $\vee$ are monotone non-decreasing in both their variables, and thus also their composition is monotone non-decreasing in each one of the variables. Each gate of $\mathcal{C}$ is a composition of functions $\wedge, \vee$ over the set $\vec{X}$ of variables, thus all gates of $\mathcal{C}$ are monotone non-decreasing in each one of the variables of $\mathcal{C}$. A gate w represents a function over the basis $\{\wedge, \vee\}$. The assignment f assigns the variable X a value in $\{0, 1\}$, and $f(w)$ is computed from the values assigned by f to all variables of $\mathcal{C}$. We assume that $f(X) \neq f(w)$. Without loss of generality, let $f(X) = 1$ and $f(w) = 0$. Assume by way of contradiction that $dr(\mathcal{C}, X, w, f) \neq 0$. Then there exists a set $\vec{Z} \subseteq \vec{X} \setminus \{X\}$ such that $\tilde{f}_{\vec{Z}}(w) = f(w) = 0$ and X is critical for w under $\tilde{f}_{\vec{Z}}$. Thus, changing the value of X from 1 to 0 changes the value of w from 0 to 1. However, this contradicts the fact that w is monotone nondecreasing in X.

The case where $f(X) = 0$ and $f(w) = 1$ follows by a dual argument. $\square$

Theorem 4.4 *The problem of computing the degree of responsibility in read-once Boolean formulas can be solved in linear time.*

Proof: We describe a linear-time algorithm for computing the degree of responsibility for read-once Boolean formulas. Since we have assumed that formulas are given in positive normal form, we can assume that the trees that represent the formulas do not contain negation gates. (The leaves may be labeled with negations of atomic propositions instead.) This means that the circuits corresponding to read-once formulas can be viewed as monotone Boolean treess, to which Lemma 4.3 can be applied.

Consider the following algorithm, which gets as input a monotone Boolean tree T, an assignment f, and a variable X whose degree of responsibility for the value of T under the assignment f we want to compute. The algorithm starts from the variables and goes up the tree to the root. For each node w in the tree, the algorithm computes two values:

- $size(T, X, w, f)$, which is the size of the minimal $\vec{Z}$ such that X is critical for w under $\tilde{f}_{\vec{Z}}$ (we take $size(T, X, w, f) = \infty$ if there is no $\vec{Z}$ such that X is critical for w under $\tilde{f}_{\vec{Z}}$; i.e., if X is not a cause of the formula associated with w under assignment f); and
- $c(w, f)$, which the size of the minimal $\vec{Z}$ such that $\vec{Z} \subseteq \vec{X}$ and $\tilde{f}_{\vec{Z}}(w) \neq f(w)$.

Note that $size(T, X, w, f) = \frac{1}{dr(T,X,w,f)} - 1$ if $dr(T, X, w, f) > 0$, and $size(T, X, w, f) = \infty$ if $dr(T, X, w, f) = 0$.

For a leaf l_X labeled with X, we have $c(l_X, f) = 1$ and $size(T, X, l_X, f) = 0$, by Definition 2.5. For a leaf l_Y labeled with $Y \neq X$ we have $c(l_Y, f) = 1$ and $size(T, X, l_Y, f) = \infty$. Suppose that w is a gate with children u and v, and that we have already computed $size(T, X, y, f)$ and $c(y, f)$, for $y \in \{u, v\}$. Then $size(T, X, w, f)$ and $c(w, f)$ are computed as follows.

1. If w is an $\wedge$-gate and $f(w) = f(u) = f(v) = 0$, or if w is $\vee$-gate and $f(w) = f(u) = f(v) = 1$, then
   (a) $c(w, f) = c(u, f) + c(v, f)$ (because we have to change the values of both u and v in order to change the value of w);
   (b) $size(T, X, w, f) = \min(size(T, X, u, f), size(T, X, v, f)) + c(v, f)$.
2. If w is an $\wedge$-gate, $f(w) = f(u) = 0$ and $f(v) = 1$, or if w is an $\vee$-gate, $f(w) = f(u) = 1$, and $f(v) = 0$, then $c(w, f) = c(u, f)$, and $size(T, X, w, f) = \infty$, by Lemma 4.3.
3. If w is an $\wedge$-gate and $f(w) = f(u) = f(v) = 1$, or if w is an $\vee$-gate and $f(w) = f(u) = f(v) = 0$, then $c(w, f) = \min(c(u, f), c(v, f))$, and $size(T, X, w, f) = \min(size(T, X, u, f), size(T, X, v, f))$.

Clearly we can compute $size(T, X, w, f)$ and $c(w, f)$ in constant time given $size(T, X, u, f)$, $size(T, X, v, f)$, $c(u, f)$, and $c(v, f)$. Moreover, because T is a tree, it is easy to check that $size(T, X, w, f)$ really is the size of the minimal $\vec{Z}$ such that X is critical for w under $\tilde{f}_{\vec{Z}}$; the argument uses the fact that we cannot have both $size(T, X, u, f) < \infty$ and $size(T, X, v, f) < \infty$, since this would mean that X is a descendant of both u and v, contradicting the read-once assumption. As we observed earlier, the degree of responsibility of X for the value of node w under f is $1/(1 + size(T, X, w, f))$, so we are done. □

5 Conclusion

We have shown that it is useful to think of coverage estimation in terms of causality. This way of thinking about coverage estimation not only shows that a number of different definitions of coverage can be thought of as being defined by different models of causality, but also suggests how the notion of coverage might be extended, to take into account which features of satisfaction are important. The notion of responsibility also provides a useful generalization of coverage, that gives a more fine-grained analysis of the importance of a state for satisfying a specification. Our complexity results suggest that these notions can be usefully incorporated into current model-checking techniques.

Acknowledgment

We thank Thomas Henzinger and Shmuel Katz for bringing to our attention the duality between coverage and fault tolerance and Michael Ben-Or for helpful discussions.

References

Ammann, P. and P. Black (2001, December). A specification-based coverage metric to evaluate test sets. *International Journal of Quality, Reliability and Safety Engineering 8*(4), 275–300.

Beatty, D. and R. Bryant (1994). Formally verifying a microprocessor using a simulation methodology. In *Proc. 31st Design Automation Conference*, pp. 596–602. IEEE Computer Society.

Beer, I., S. Ben-David, C. Eisner, and Y. Rodeh (1997). Efficient detection of vacuity in ACTL formulas. In *Proc. 9th Conference on Computer Aided Verification*, Volume 1254 of *Lecture Notes in Computer Science*, pp. 279–290.

Beer, I., S. Ben-David, C. Eisner, and Y. Rodeh (2001). Efficient detection of vacuity in ACTL formulas. *Formal Methods in System Design 18*(2), 141–162.

Bieman, J., D. Dreilinger, and L. Lin (1996). Using fault injection to increase software test coverage. In *Proc. Int. Symp. Software Reliability Engineering (ISSRE'96)*, pp. 166–174.

Budd, T. (1981). Mutation analysis: Ideas, examples, problems, and prospects. In B. Chandrasekaran and S. Radichi (Eds.), *Computer Program Testing*, pp. 129–148. North Holland.

Budd, T. and D. Angluin (1982). Two notions of correctness and their relation to testing. *Acta Informatica 18*, 31–45.

Chockler, H. and J. Halpern (2004). Responsibility and blame: a structural-model approach. *Journal of Artificial Intelligence Research (JAIR) 22*, 93–115.

Chockler, H. and O. Kupferman (2002, August). Coverage of implementations by simulating specifications. In R. Baeza-Yates, U. Montanari, and N. Santoro (Eds.), *Proceedings of 2nd IFIP International Conference on Theoretical Computer Science*, Volume 223 of *IFIP Conference Proceedings*, Montreal, Canada, pp. 409–421. Kluwer Academic Publishers.

Chockler, H., O. Kupferman, R. Kurshan, and M. Vardi (2001). A practical approach to coverage in model checking. In *Computer Aided Verification, Proc. 13th International Conference*, Volume 2102 of *Lecture Notes in Computer Science*, pp. 66–78. Springer-Verlag.

Chockler, H., O. Kupferman, and M. Vardi (2001). Coverage metrics for temporal logic model checking. In *Tools and Algorithms for the Construction and Analysis of Systems*, Number 2031 in Lecture Notes in Computer Science, pp. 528 – 542. Springer-Verlag.

Chockler, H., O. Kupferman, and M. Vardi (2003). Coverage metrics for formal verification. In *Correct Hardware Design and Verification Methods (CHARME)*, Number 2860 in Lecture Notes in Computer Science, pp. 111–125. Springer-Verlag.

Clarke, E., O. Grumberg, K. McMillan, and X. Zhao (1995). Efficient generation of counterexamples and witnesses in symbolic model checking. In *Proc. 32nd Design Automation Conference*, pp. 427–432. IEEE Computer Society.

Clarke, E. M., O. Grumberg, and D. A. Peled (1999). *Model Checking*. Cambridge, Mass.: MIT Press.

DeMillo, R. A., R. J. Lipton, and F. G. Sayward (1978). Hints on test data selection: Help for the practicing programmer. *IEEE Computer 11*(4), 34–43.

Dill, D. (1998). What's between simulation and formal verification? In *Proc. 35st Design Automation Conference*, pp. 328–329. IEEE Computer Society.

Eiter, T. and T. Lukasiewicz (2002a). Causes and explanations in the structural-model approach: tractable cases. In *Proc. Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI 2002)*, pp. 146–153.

Eiter, T. and T. Lukasiewicz (2002b). Complexity results for structure-based causality. *Artificial Intelligence 142*(1), 53–89.

Emerson, E. and E. Clarke (1982). Using branching time logic to synthesize synchronization skeletons. *Science of Computer Programming 2*, 241–266.

Groce, A., S. Chaki, D. Kroening, and O. Strichman. Error explanation with distance metrics. *International Journal on Software Tools for Technology Transfer (STTT)*. accepted for publication.

Hall, N. (2004). Two concepts of causation. In J. Collins, N. Hall, and L. A. Paul (Eds.), *Causation and Counterfactuals*. Cambridge, Mass.: MIT Press.

Halpern, J. Y. and J. Pearl (2005). Causes and explanations: A structural-model approach. Part I: Causes. *British Journal for Philosophy of Science*, 843–887.

Hoskote, Y., T. Kam, P.-H. Ho, and X. Zhao (1999). Coverage estimation for symbolic model checking. In *Proc. 36th Design Automation Conference*, pp. 300–305.

Hume, D. (1739). *A Treatise of Human Nature*. John Noon, London.

Jayakumar, N., M. Purandare, and F. Somenzi (2003). Dos and don'ts of ctl state coverage estimation. In *Proc. 40th Design Automation Conference (DAC)*. IEEE Computer Society.

Krentel, M. (1988). The complexity of optimization problems. *Journal of Computer and System Sciences 36*, 490–509.

Kupferman, O. and M. Vardi (1999). Vacuity detection in temporal model checking. In *10th Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, Volume 1703 of *Lecture Notes in Computer Science*, pp. 82–96. Springer-Verlag.

Kupferman, O. and M. Vardi (2003, February). Vacuity detection in temporal model checking. *Journal on Software Tools For Technology Transfer 4*(2), 224–233.

Kupferman, O., M. Vardi, and P. Wolper (2000, March). An automata-theoretic approach to branching-time model checking. *Journal of the ACM 47*(2), 312–360.

Kurshan, R. (1998). *FormalCheck User's Manual*. Cadence Design, Inc.

Lynch, N. (1996). *Distributed Algorithms*. San Francisco: Morgan Kaufmann.

Millo, R. D., R. Lipton, and F. Sayward (1978, April). Hints on test data selection: Help for the practicing programmer. *IEEE Computer 11*(4), 34–41.

Millo, R. D. and A. Offutt (1991, September). Constraint-based automatic test data generation. *IEEE Transactions on Software Engineering 17*(9), 900–910.

Papadimitriou, C. H. (1984). The complexity of unique solutions. *Journal of ACM 31*, 492–500.

Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley. 2nd edition.

Peled, D. (2001). *Software Reliability Methods*. Springer-Verlag.

Purandare, M. and F. Somenzi (2002, July). Vacuum cleaning ctl formulae. In *Proc. 14th Conference on Computer Aided Verification*, Lecture Notes in Computer Science, pp. 485–499. Springer-Verlag.

Zhu, H., P. Hall, and J. May (1997). Software unit test coverage and adequacy. *ACM Computing Surveys 29*(4), 366–427.

A Proofs

A.1 Proof of Theorem 4.1

First we prove membership in FP$^{\mathrm{NP}[\log n]}$ by describing an algorithm in FP$^{\mathrm{NP}[\log n]}$ for solving RESP-CIRCUIT. The algorithm queries an oracle O_{L_c} for membership in the language L_c, defined as follows:

$$L_c = \{\langle \mathcal{C}', X', f', i\rangle : dr(\mathcal{C}', X', f') \geq 1/i\}.$$

In other words, $\langle \mathcal{C}', X', f', i\rangle \in L_c$ if there exists a set $\vec{Z}$ of variables of size at most $i-1$ such that X' is critical for $\mathcal{C}'$ under the assignment $\tilde{f}'_{\vec{Z}}$. It is easy to see that $L_c \in$ NP. Indeed, given a set $\vec{Z}$ of size at most $i-1$, the check for whether X' is critical for $\mathcal{C}'$ under $\tilde{f}'_{\vec{Z}}$ can be performed in time linear in the size of $\mathcal{C}'$. Given input $(\mathcal{C}, X, f)$, the algorithm for solving RESP-CIRCUIT performs a binary search on the value of $dr(\mathcal{C}, X, f)$, each time dividing the range of possible values for $dr(\mathcal{C}, X, f)$ by 2 according to the answer of O_{L_c}. The number of possible candidates for $dr(\mathcal{C}, X, f)$ is the number of variables that appear in $\mathcal{C}$, and thus the number of queries to O_{L_c} is at most $\lceil \log n \rceil$, where n is the size of the input.

We now prove FP$^{\mathrm{NP}[\log n]}$-hardness by a reduction from the problem CLIQUE-SIZE, which is known to be FP$^{\mathrm{NP}[\log n]}$-complete [Papadimitriou 1984; Krentel 1988; Papadimitriou 1994]. CLIQUE-SIZE is the problem of determining the size of the largest clique of an input graph G. The reduction works as follows. Let $G = \langle \vec{V}, \vec{E}\rangle$ be a graph. We start by constructing a circuit $\mathcal{C}_G$, where the variables are the nodes in $\vec{V}$, and the output of the circuit is 1 iff the set of nodes assigned 0 forms a clique in G. The circuit $\mathcal{C}_G$ is $\mathcal{C}_G = \bigwedge_{(V,W)\notin E}(V \vee W)$. It is easy to see that the value of $\mathcal{C}_G$ under an assignment f is 1 iff there are edges between all pairs of nodes that are assigned 0 by f. In other words, the set of nodes assigned 0 by f forms a clique in G.

Now let X be a variable that does not appear in $\mathcal{C}_G$. Consider the circuit $\mathcal{C} = X \wedge \mathcal{C}_G$, and an assignment F that assigns 0 to all variables in V and to X. It is easy to see that the value of $\mathcal{C}$ under F is 0, and that for an assignment f that assigns X the value 1, $\mathcal{C}$ outputs the value of $\mathcal{C}_G$ under the assignment f restricted to V. We claim that $dr(\mathcal{C}, X, F) = 1/i > 0$ iff the size of the maximal clique in G is $|V| - i + 1$, and $dr(\mathcal{C}, X, F) = 0$ iff there is no clique in G.

We start with the "if" direction. Let $dr(\mathcal{C}, X, F) = 1/i > 0$. Then there exists a set $\vec{Z} \subseteq \vec{V}$ of size $i-1$ such that $\tilde{F}_{\vec{Z}}(\mathcal{C}) = \neg\tilde{F}_{\vec{Z}\cup\{X\}}(\mathcal{C})$. Since $\tilde{F}_{\vec{Z}}(X) = 0$, we also have $\tilde{F}_{\vec{Z}}(\mathcal{C}) = 0$, and thus $\tilde{F}_{\vec{Z}\cup\{X\}}(\mathcal{C}) = 1$. Therefore, the value of $\mathcal{C}_G$ under the assignment $\tilde{F}_{\vec{Z}}$ restricted to $\vec{V}$ is 1. Thus, the set of variables assigned 0 in $\tilde{F}_{\vec{Z}}$ forms a clique in G. The assignment $\tilde{F}_{\vec{Z}}$ differs from F precisely on the values it assigns to variables in $\vec{Z}$; thus, the set of variables assigned 0 by $\tilde{F}_{\vec{Z}}$ is $\vec{V} \setminus \vec{Z}$. We know that $|\vec{Z}| = i-1$, therefore $|\vec{V} \setminus \vec{Z}| = |\vec{V}| - i + 1$. On the other hand, by the definition of the degree of responsibility, for all sets $\vec{Z} \subseteq V$ of size $j < i-1$ we have $\tilde{F}_{\vec{Z}}(\mathcal{C}) = \neg\tilde{F}_{\vec{Z}\cup\{X\}}(\mathcal{C})$. Thus, the value of $\mathcal{C}_G$ under the assignment $\tilde{F}_{\vec{Z}}$ restricted to $\vec{V}$ is 0. Thus, for all sets $\vec{Z} \subseteq \vec{V}$ of size $j < i-1$, we have that $\vec{V} \setminus \vec{Z}$ is not a clique in G. Therefore, the maximal clique in G is of size $|\vec{V}| - i + 1$.

For the "only if" direction, let $\vec{Y} \subseteq \vec{V}$ of size $|\vec{V}| - i + 1$ be the maximal clique in G. Then the value of $\mathcal{C}_G$ is 1 under the assignment $\tilde{F}_{\vec{V}\setminus\vec{Y}}$. Therefore, $\tilde{F}_{(\vec{V}\setminus\vec{Y})\cup\{X\}}(\mathcal{C}) = 1$, while $\tilde{F}_{\vec{V}\setminus\vec{Y}}(\mathcal{C}) = F(\mathcal{C}) = 0$. Thus, X is critical for $\mathcal{C}$ under the assignment $\tilde{F}_{\vec{V}\setminus\vec{Y}}$, and therefore $dr(\mathcal{C}, X, f) \geq i$. On the other hand, since $\vec{Y}$ is maximal, for all sets $\vec{Z}$ of size $|\vec{V}| - j$ for $j < i-1$, we have that $\vec{Z}$ is not a clique in G, thus the value of $\mathcal{C}_G$ is 0 under the assignment $\tilde{F}_{\vec{V}\setminus\vec{Z}}$. Therefore, $\tilde{F}_{(\vec{V}\setminus\vec{Z})\cup\{X\}}(\mathcal{C}) = 0 = \tilde{F}_{\vec{V}\setminus\vec{Z}}(\mathcal{C})$, and thus X is not critical for $\mathcal{C}$ under the assignment $\tilde{F}_{\vec{V}\setminus\vec{Z}}$. It follows that $dr(\mathcal{C}, X, F) \leq i$. Since $dr(\mathcal{C}, X, f) \geq i$, we get that $dr(\mathcal{C}, X, F) = i$.

If $dr(\mathcal{C}, X, F) = 0$, then for all sets $\vec{Z} \subseteq \vec{V}$, we have $\tilde{F}_{\vec{Z}\cup\{X\}}(\mathcal{C}) = \tilde{F}_{\vec{Z}}(\mathcal{C}) = 0$, and thus $\tilde{F}_{\vec{Z}}(\mathcal{C}_G) = 0$. Thus, there is no clique in G. For the converse, assume that there is no clique in G. For the other direction, assume that there is no clique in G. Then for all $\vec{Y} \subseteq V$, we have $\tilde{F}_{\vec{V}\setminus\vec{Y}}(\mathcal{C}_G) = 0$, thus $\tilde{F}_{(\vec{V}\setminus\vec{Y})\cup\{X\}}(\mathcal{C}) = \tilde{F}_{\vec{V}\setminus\vec{Y}}(\mathcal{C}) = 0$. It follows that $dr(\mathcal{C}, X, F) = 0$.

A.2 Proof of Theorem 4.2

The lower bound follows from the lower bound in Theorem 4.1. For the upper bound, we use the following observation made by Eiter and Lukasiewicz: for binary causal models, the condition AC2 can be replaced by the following condition (to get an equivalent definition of causality):

AC2′. There exist a partition $(\vec{Z}, \vec{W})$ of $\mathcal{V}$ with $\vec{X} \subseteq \vec{Z}$ and some setting $(\vec{x}', \vec{w}')$ of the variables in $(\vec{X}, \vec{W})$ such that if $(M, \vec{u}) \models Z = z^*$ for $Z \in \vec{Z}$, then

1. $(M, \vec{u}) \models [\vec{X} \leftarrow \vec{x}', \vec{W} \leftarrow \vec{w}'] \neg\varphi$.
2. $(M, \vec{u}) \models [\vec{X} \leftarrow \vec{x}, \vec{W} \leftarrow \vec{w}', \vec{Z} \leftarrow \vec{z^*}] \varphi$.

That is, for binary causal models it is enough to check that changing the value of $\vec{W}$ does not falsify φ if all other variables keep their original values. Thus, given a partition $(\vec{Z}, \vec{W})$ and a setting $(\vec{x}', \vec{w}')$ we can *verify* that $(X = x)$ is an active cause in polynomial time: both conditions in AC2′ are verifiable by evaluating a Boolean formula under a given assignment to its variables. Thus checking causality in binary models is in NP. Therefore, the following language L'_c is also in NP.

$$L'_c = \{\langle (M, \vec{u}), \psi, (X = x), i \rangle : \text{ the degree of responsibility of } (X = x) \\ \text{for } \psi \text{ in the context } (M, \vec{u}) \text{ is at least } 1/i\}.$$

Indeed, membership of $\langle (M, \vec{u}), \psi, (X = x), i \rangle$ in L'_c is verifiable in polynomial time similarly to the causality check with the addition of measuring the size of witness $\vec{W}$, which has to be at most $i - 1$. The algorithm for computing the degree of responsibility of $(X = x)$ for the value of ψ in the context $(M, \vec{u})$ performs a binary search similarly to the same algorithm for Boolean circuits, each time dividing the range of possible values by 2 according to the answer of an oracle to the NP language L'_c. The number of queries is bounded by $\lceil \log n \rceil$, where n is the size of the input, thus the problem is in $\mathrm{FP}^{\mathrm{NP}[\log n]}$.