Coverage Metrics for Temporal Logic Model Checking

Hana Chockler[1] and Orna Kupferman[1] and Moshe Y. Vardi[2*]

[1] Hebrew University, School of Engineering and Computer Science, Jerusalem 91904, Israel
Email: {hanac,orna}@cs.huji.ac.il, URL: http://www.cs.huji.ac.il/{~hanac,~orna}
[2] Rice University, Department of Computer Science, Houston, TX 77251-1892, U.S.A.
Email:vardi@cs.rice.edu, URL: http://www.cs.rice.edu/~vardi

Abstract. In formal verification, we verify that a system is correct with respect to a specification. Even when the system is proven to be correct, there is still a question of how complete the specification is, and whether it really covers all the behaviors of the system. In this paper we study coverage metrics for model checking. Coverage metrics are based on modifications we apply to the system in order to check which parts of it were actually relevant for the verification process to succeed. We introduce two principles that we believe should be part of any coverage metric for model checking: a distinction between state-based and logic-based coverage, and a distinction between the system and its environment. We suggest several coverage metrics that apply these principles, and we describe two algorithms for finding the uncovered parts of the system under these definitions. The first algorithm is a symbolic implementation of a naive algorithm that model checks many variants of the original system. The second algorithm improves the naive algorithm by exploiting overlaps in the variants. We also suggest a few helpful outputs to the user, once the uncovered parts are found.

1 Introduction

In *model checking* [CE81,QS81,LP85], 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, expressed in terms of a temporal logic formula or a finite automaton. Beyond being fully-automatic, an additional attraction of model-checking tools is their ability to accompany a negative answer to the correctness query by a counterexample to the satisfaction of the specification in the system. Thus, together with a negative answer, the model checker returns some erroneous execution of the system. These counterexamples are very important and they can be essential in detecting subtle errors in complex designs [CGMZ95]. 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 at first seems like a reasonable policy. In the last few years, however, there has been growing awareness of the importance of suspecting the system of containing an error also in the case model checking

* Supported in part by NSF grant CCR-9700061, NSF grant CCR-9988322, and by a grant from the Intel Corporation.

succeeds. The main justification of such suspects are possible errors in the modeling of the system or of the behavior, and possible incompleteness in the specification.

There are various ways to look for possible errors in the modeling of the system or the behavior. One direction is to detect *vacuous satisfaction* of the specification [BBER97,KV99], where cases like antecedent failure [BB94] make parts of the specification irrelevant to its satisfaction. For example, the specification $\varphi = AG(req \rightarrow AF grant)$ is vacuously satisfied in a system in which req is always **false**. A similar direction is to check the validity of the specification. Clearly, a valid specification is satisfied trivially, and suggests some problem. A related approach is taken in the process of constraint validation in the verification tool FormalCheck [Kur98], where sanity checks for constraint validation include a search for enabling conditions that are never enabled, and a replacement of all or some of the constraints by **false**. FormalCheck also keeps track of variables and values of variables that were never used in the process of model checking.

It is less clear how to check completeness of the specification. Indeed, specifications are written manually, and their completeness depends entirely on the competence of the person who writes them. The motivation for such a check is clear: an erroneous behavior of the system can escape the verification efforts if this behavior is not captured by the specification. In fact, it is likely that a behavior not captured by the specification also escapes the attention of the designer, who is often the one to provide the specification.

This direction, of checking whether the specification describes the system exhaustively, has roots in simulation-based verification techniques, where *coverage metrics* are used to improve the quality of test vectors. For example, *code coverage* [CK93] measures the fraction of HDL statements executed during simulation, *transition coverage* [HYHD95,HMA95] measures the fraction of transitions executed, and *tag coverage* [DGK96] attributes variables with tags that are used to detect whether assigning a forbidden value to a variable leads to an erroneous behavior of the system. In [FDK98,FAD99], Fallah et al. compute the tag coverage achieved by simulation and generated simulation sequences that cover a given tagged variable. Of a similar nature is the *tour-generation algorithm* in [HYHD95], which generates test vectors that traverse all states of the system. Ho and Horowitz [HH96] define test coverage in terms of *control events*. Each control event identifies an interesting subset of the control variables, and the test vectors have to cover all the events. They also define *design coverage* by means of the states and edges covered by the test vectors (see also [MAH98]). In order to circumvent the state-explosion problem in these methods, Bergmann and Horowitz develop the technique of *projection directed state exploration*, which allows to compute the above coverage metrics for small portions of the design [BH99]. Coverage metrics are helpful as an indicator whether the simulation process has been exhaustive. Still, simulation-based verification techniques lack of a uniform definition of coverage.

Following the same considerations, analyzing coverage in model checking can discover parts of the system that are not relevant for the verification process to succeed. Low coverage can point to several problems. One possibility is that the specification is not complete enough to fully describe all the possible behaviors of the system. In this case, the output of a coverage check is helpful in completing the specification. Another possibility is that the system contains redundancies. In this case, the output of the coverage check is helpful in simplifying the system.

Two approaches for defining and developing algorithms for coverage metrics in temporal logic model checking are studied in the literature. The first approach, of Katz et al. [KGG99], is based on a comparison of the system with a tableau of the specification. Essentially, a tableau of a universal specification φ is a system that satisfies φ and subsumes all the behaviors allowed by φ. By comparing a system with the tableau of φ, Katz et al. are able to detect parts of the systems that are irrelevant to the satisfaction of the specification, to detect different behaviors of the system that are indistinguishable by the specification, and to detect behaviors that are allowed by the specification but not generated by the system. Such cases imply that the specification is incomplete or not sufficiently restrictive. The tableau used in [KGG99] is reduced: a state of the tableau is associated with subformulas that have be true in it, and it induces no obligations on the other, possibly propositional, subformulas. This leads to smaller and less restrictive tableaux. Still, we found the approach in [KGG99] too strict. Indeed, a system passes the criteria in [KGG99] iff it is bisimilar to the tableau of the specification, but we want specifications to be much more abstract than their implementations[1].

The second approach, of Hoskote et al. [HKHZ99], is to define coverage by examining the effect of modifications in the system on the satisfaction of the specification. Given a system modeled by a Kripke structure K, a formula φ satisfied in K, and a signal (atomic proposition) q, a state w of K is q-covered by φ if the Kripke structure obtained from K by flipping the value of q in w no longer satisfies φ (the signal q corresponds to a boolean variable that is **true** if w is labeled with q and is **false** otherwise, so when we say that we flip the value of q, we mean that we switch the value of this variable). Indeed, this indicates that the value of q in w is crucial for the satisfaction of φ in K. The signal q is referred to as the *observable signal*. Let us denote by $\tilde{K}_{w,q}$ the Kripke structure obtained from K by flipping the value of q in w, and denote by q-*cover*(K, φ) the set of states q-covered by φ in K. It is easy to see that q-*cover*(K, φ) can be computed by a naive algorithm that performs model checking of φ in $\tilde{K}_{w,q}$ for each state w of K. By [HKHZ99], a state is covered if it belongs to q-*cover*(K, φ) for some observable signal q.

Hoskote et al. describe an algorithm for computing the set of states that are q-covered by a formula φ in the logic *acceptable ACTL*. Acceptable ACTL is a restriction of the universal fragment ACTL of CTL in which no disjunctions are allowed and all the implications $\alpha \to \beta$ are such that α is propositional. The algorithm in [HKHZ99] is applied to φ after an *observability transformation*. The restricted syntax of acceptable ACTL together with the observability transformation lead to a symbolic algorithm that, like CTL model-checking, requires linear time [2]. On the other hand, the set of states designated as q-covered by the algorithm is not q-*cover*(K, φ). It is claimed in

[1] The approach in [KGG99] also has some technical and computational drawbacks: the specification considered is the (big) conjunction of all the properties the system should satisfy, the complexity of the algorithm is exponential in the specification (for φ in ACTL), and it is restricted to universal safety specifications whose tableaux have no fairness constraints.

[2] The restricted syntax of acceptable ACTL and the observability transformation force $\tilde{K}_{w,q}$, for all states w, to satisfy φ in exactly the same way K does. For example, if a path in K satisfies

[HKHZ99] that the set found by the algorithm meets better the intuition of coverage. One can argue whether this is indeed the case; we actually found several performances of the algorithm in [HKHZ99] quite counter-intuitive (for example, the algorithm is syntax-dependent, thus, equivalent formulas may induce different coverage sets; in particular, the set of states q-covered by the tautology $q \rightarrow q$ is the set of states that satisfy q, rather than the empty set, which meets our intuition of coverage). Anyway, this is not the point we want to make here — there are many possible ways of defining coverage sets, each way has its advantages, and there need not be a best way. The point we want to make in this paper is that there are two important principles that should be part of any coverage metric for temporal logic model checking: a distinction between state-based and logic-based approaches, and a distinction between the system and its environment. These principles, which we explain below, are not applied in [HKHZ99] and in other work on coverage hitherto.

The first principle, namely a distinction between state-based and logic-based approaches, is based on the observation that there are several ways to model a system, and the different ways should induce different references to the observability signal and its modification. Recall [HKHZ99]'s definition of coverage. Hoskote et al. model systems by Kripke structures and the observable signal q is one of the atomic propositions that label the states of the structure and encode the system's variables. For every state w, the truth value of q is flipped in $\tilde{K}_{w,q}$. This approach is *state based*, as it modifies q in each of the states. When the system is modeled as a circuit and its state space is 2^V, for the set V of signals, transitions are given as relations between current and next values to the signals in V [MP92]. Then, flipping the value of a signal in a state changes not only the "label" of the state but also the transitions to and from the state. So, in the state-based approach, we consider modifications that refer to a single state of the system and to the adjacent transitions. When the system is modeled as a circuit, we can also take the *logic-based* approach, where we do not flip the value of a signal in a particular state, but rather, fix the signal to 0, 1, or "don't care" everywhere in the circuit, and check the effect of these fixes on the satisfaction of the specification.

In order to explain the second principle, namely a distinction between a system and its environment, assume a definition of coverage in which a state is covered iff its removal violates the specification. Since universal properties are preserved under state removal, no state would be covered by a universal specification in such a definition. So, is this a silly definition? The definition makes perfect sense in the context of closed systems. There, universal properties can be satisfied by the empty system, and if a designer wants the system to do something (more than just being correct), this something should be specified by an existential specification. On the other hand, in an open system, which interacts with its environment, the above definition makes sense only if we restrict it to states whose removal leaves the system responsive to all the behaviors of the environment and does not deadlock the interaction between the system and its environment. Indeed, we cannot remove a state if the environment can force a visit to it. Likewise, it makes no sense to talk about q-coverage for a signal q that corresponds to an input variable. Indeed, it is the environment that determines the value of q, we cannot flip its

$\alpha U \beta$ by fulfilling β in the present, this path is expected to satisfy β in the present also in $\tilde{K}_{w,q}$. This restriction is what makes the algorithm so efficient.

value, and anyway we cannot talk about states being q-covered or not: all the values of q should be around simply since the environment can force them all. Hence, in the definition of coverage metrics, in both the design and implementation levels, there should be a distinction between input and output variables, and coverage should be examined only with respect to elements the system has control on.

The contribution of our paper is as follows. We introduce the above two principles in the definition of coverage, and we give several coverage metrics that apply them. Our definitions are similar to the one in [HKHZ99] in the sense that they consider the influence of local modifications of the system on the satisfaction of the specification (in fact, [HKHZ99] can be viewed as a special case of our state-based approach, for a closed system, with $C \cap O = \emptyset$, and with only output variables being observable). Hence, the naive algorithm, which finds the set of covered states (in the state-based approach) or signals (in the logic-based approach) by model checking each of the modified systems is applicable. We describe two alternatives to this naive algorithm. The first alternative is a symbolic approach to finding the uncovered parts of the system. The second alternative is an algorithm that makes use of overlaps among the modified systems — since each modification involves a small change in the original system, there is a great deal of work that can be shared when we model check all the modified systems. Both algorithms work for full CTL, and the ideas in them can be adopted to various definitions of coverage. Once the set of covered states is found, we suggest a few helpful outputs to the user (more helpful than just the percentage of covered states).

Due to lack of space, many details are omitted from this version. A full version of the paper can be found in the authors' URLs.

2 Coverage Metrics

In this section we suggest several coverage metrics for temporal logic model checking. As describe in Section 1, we distinguish between a state-based and a logic-based approach to coverage, and we distinguish between a system and its environment. Our definitions are independent of the temporal logic being used. We assume the reader is familiar with temporal logic. In particular, the algorithms we are going to present are for the branching time logic CTL. Formulas of CTL are built from a set AP of atomic propositions using the boolean operators $\vee$ and $\neg$, the temporal operators X ("next") and U ("until"), and the path quantifiers E ("exists a path") and A ("for all paths"). Every temporal operator must be immediately preceded by a path quantifier. The semantics of temporal logic formulas is defined with respect to Kripke structures. For a full definition of Kripke structures, and the syntax and semantics of CTL, see [Eme90] and full version of this paper. For a formula φ (and an agreed Kripke structure K), we denote by $||\varphi||$ the set of states that satisfy φ in K, and use $cl(\varphi)$ to denote the set of φ's subformulas. 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.

We distinguish between two types of systems: *closed* and *open* [HP85]. A closed system is a system whose behavior is completely determined by the state of the system. An open system is a system that interacts with its environment and whose behavior depends on external nondeterministic choices made by the environment [Hoa85]. In a Kripke structure, all the atomic propositions describe internal signals, thus Kripke structures model closed systems. We study here open systems, and we model them by *sequential circuits*.

A sequential circuit (*circuit*, for short) is a tuple $\mathcal{S} = \langle I, O, C, \theta, \rho, \delta \rangle$, where I is a set of input signals, O is a set of output signals, and C is a set of control signals that induce the state space 2^C. Accordingly, $\theta \in 2^C$ is an initial state, $\rho : 2^C \times 2^I \to 2^C$ is a deterministic transition function, and $\delta : 2^C \to 2^O$ is an output function. Possibly $O \cap C \neq \emptyset$, in which case for all $x \in O \cap C$ and $s \in 2^C$, we have $x \in s$ iff $x \in \delta(s)$. Thus, $\delta(s)$ agrees with s on signals in C. We partition the signals in $O \cup C$ into three classes as follows. A signal $x \in O \setminus C$ is a *pure-output* signal. A signal $x \in C \setminus O$ is a *pure-control* signal. A signal $x \in C \cap O$ is a *visible-control* signal. While pure output signals have no control on the transitions of the system, a specification of the system can refer only to the values of the pure-output or the visible-control signals.

We define the semantics of CTL with respect to circuits by means of the Kripke structure they induce. A circuit $\mathcal{S} = \langle I, O, C, \theta, \rho, \delta \rangle$ can be converted to a Kripke structure $K_{\mathcal{S}} = \langle I \cup C \cup O, 2^C \times 2^I, R, \langle \theta, \emptyset \rangle, L \rangle$, where for all s and s' in 2^C, and i and i' in 2^I, we have $R(\langle s, i \rangle, \langle s', i' \rangle)$ iff $\rho(s, i) = s'$. Also, $L(\langle s, i \rangle) = \delta(s) \cup i \cup s$. Note that each state in $K_{\mathcal{S}}$ has $2^{|I|}$ successors, reflecting external nondeterminism induced by the environment of $\mathcal{S}$. We assume that the interaction between the circuit and its environment is initiated by the circuit, hence the single initial state. The other possibility, where the interaction between the circuit and its environment is initiated by the environment, corresponds to a Kripke structure with a set $\theta \times 2^I$ of initial states. Our definitions and algorithms assume a single initial state, yet they can be easily modified to handle multiple initial states.

We now define what it means for a specification to cover a circuit. Let $\mathcal{S}$ be a circuit that satisfies a specification φ. We want to check whether the specification describes $\mathcal{S}$ exhaustively. Intuitively, the uncovered part of $\mathcal{S}$ is the part that can be modified without falsifying φ in $\mathcal{S}$. Formally, we suggest several definitions of coverage, reflecting the various possible ways in which a part of $\mathcal{S}$ can be modified.

We start with the state-based definition. Here, we check whether the satisfaction of φ is sensitive to local changes in the values of output and control signals; i.e., changes in one state.

For a circuit $\mathcal{S} = \langle I, O, C, \theta, \rho, \delta \rangle$, a state $s \in 2^C$, and a signal $x \in C$, we define the *x-twin of s*, denoted $twin_x(s)$, as the state s' obtained from s by dualizing the value of x. Thus, $x \in s'$ iff $x \notin s$. Now, given $\mathcal{S}$, s, and a signal $x \in O \cup C$, we define the *dual circuit* $\tilde{\mathcal{S}}_{s,x} = \langle I, O, C, \tilde{\theta}, \tilde{\rho}, \tilde{\delta} \rangle$ as follows.

- If x is a pure-output signal, then $\tilde{\theta} = \theta$, $\tilde{\rho} = \rho$, and $\tilde{\delta}$ is obtained from δ by dualizing the value of x in s, thus $x \in \tilde{\delta}(s)$ if $x \notin \delta(s)$.
- If x is a pure-control signal, then $\tilde{\delta} = \delta$, and $\tilde{\theta}$ and $\tilde{\rho}$ are obtained by replacing all the occurrences of s in θ and in the range of ρ by $twin_x(s)$. Thus, if $\theta = s$, then $\tilde{\theta} = twin_x(s)$ (otherwise, $\tilde{\theta} = \theta$), and for all $s' \in 2^C$ and $i \in 2^I$, if $\rho(s', i) = s$, then $\tilde{\rho}(s', i) = twin_x(s)$ (otherwise, $\tilde{\rho}(s', i) = \rho(s', i)$).

– If x is a visible-control signal, then we do both changes. Thus, $\tilde{\delta}$ is obtained from δ by dualizing the value of x in s, and $\tilde{\theta}$ and $\tilde{\rho}$ are obtained by replacing all the occurrences of s in θ and in the range of ρ by $twin_x(s)$.

Intuitively, dualizing a control signal x in a state s in $\mathcal{S}$ means that all the transitions leading to s are now directed to its x-twin. In particular, the state s is no longer reachable in $\tilde{\mathcal{S}}_{s,x}$ (which is why we do not have to redefine $\tilde{\rho}(s,i)$). For a specification φ such that $\mathcal{S} \models \varphi$, a state s is x-covered by φ if $\tilde{\mathcal{S}}_{s,x}$ does not satisfy φ.

Note that it makes no sense to define coverage with respect to observable input signals. This is because an open system has no control on the values of the input signals, which just resolve the external nondeterminism of the system. In a closed system, the set of input signals is empty, and thus the system has the control on all its variables. Therefore in closed systems we can define coverage with respect to all signals.

Our second approach to coverage, which we call *logic-based* coverage, does not refer to a particular state of $\mathcal{S}$, and it examines the possibility of fixing some control signals to 0 or 1. For a circuit $\mathcal{S} = \langle I, O, C, \theta, \rho, \delta \rangle$ and a control signal $x \in C$, the *x-fixed-to-1 circuit* $\mathcal{S}_{x,1} = \langle I, O, C, \theta', \rho', \delta \rangle$ is obtained from $\mathcal{S}$ by replacing all the occurrences of x in θ and in the range of ρ by 1; i.e., $\theta' = \theta \cup \{x\}$, and for all $s \in 2^C$ and $i \in 2^I$, we have $\rho'(s,i) = \rho(s,i) \cup \{x\}$. Similarly, the *$x$-fixed-to-$0$ circuit* $\mathcal{S}_{x,0}$ is defined by replacing all the occurrences of x in θ and in the range of ρ by 0. A control signal x is *1-covered* if $\mathcal{S}_{x,1}$ does not satisfy φ. Similarly, x is *0-covered* if $\mathcal{S}_{x,0}$ does not satisfy φ. [3]

3 Algorithms for Computing Coverage

In this section we describe algorithms for solving the following problems. Let φ be a specification in CTL.

– Given a circuit $\mathcal{S}$ that satisfies φ and an observable output or control signal x, return the set of states not x-covered by φ in $\mathcal{S}$.
– Given a circuit $\mathcal{S}$ that satisfies φ, return the set of control signals that can be fixed to 0 or fixed to 1.

All these problems have a naive algorithm that model checks each of the corresponding dual circuits. For example, in order to find the set of states not x-covered by φ in a circuit $\mathcal{S} = \langle I, O, C, \theta, \rho, \delta \rangle$, with the observable signal x being a pure-output signal, the naive algorithm executes the model-checking procedure $|2^C|$ times, once for each dual circuit, where each dual circuit is obtained from $\mathcal{S}$ by dualizing the value of x in one state. A similar thing happens in the naive algorithm with the observable signal being a control signal, only that here the dual circuits also differ in some of

[3] The logic-based definition of coverage is closely related to the notion of *observability don't care conditions* as presented in [HS96]. There, there is a set $C' \subseteq C$ of control signals and an assignment for the signals in C', denoted by the set $\alpha \subseteq C'$ of signals that are assigned **true** in this assignment, such that the behavior of the output signals in all the states $\alpha \cup \beta$, for all $\beta \subseteq C \setminus C'$, is the same. Thus, whenever the system is in a state in which the control signals in C' has value α, the value of the other control signals is "don't care".

their transitions. Finally, the naive algorithm for the logic-based coverage executes the model-checking procedure $|C|$ times, once for each control signal.

We present two alternatives to the naive algorithm. The first is a symbolic algorithm that manipulates pairs of sets of states, and the second is an algorithm that makes use of overlaps among the various dual circuits. We are going to describe our coverage algorithms in terms of Kripke structures. For that, we first need to adjust the definitions in Section 2 to Kripke structures. For a Kripke structure K, an observable signal q, a set $Y \subseteq W$ of states, and sets Z^+ and Z^- of transitions in $W \times W$, the *dual Kripke structure* $\tilde{K}(q, Y, Z^+, Z^-)$ is obtained from K by dualizing the value of q in states in Y, adding to R transitions in Z^+, and removing from R transitions in Z^-. It is easy to see that all the dualizations of circuits mentioned in Section 2 can be described in terms of dualizations of the Kripke structures they induce.

For simplicity, we describe in detail the algorithms for computing the set of x-covered states for a pure-output signal x and the case $I = \emptyset$. The same ideas work for the more general cases. Some generalizations are very simple (e.g., releasing the requirement for I being empty) and some result in a significantly more complicated algorithm (e.g., when the modification involve control signals, where transitions are modified). For the detailed explanation of the required modifications in both algorithms presented below see the full version of this paper. For our special case, we define, for a specification φ, a Kripke structure K that satisfies φ, and an observable atom q, the set of states q-covered by φ in K as the set of states w such that the dual Kripke structure $\tilde{K}_{w,q} = \tilde{K}(q, \{w\}, \emptyset, \emptyset)$ does not satisfy φ. Thus, a state of K is not q-covered if we can flip the value of q in it and still satisfy φ. This definition, which is studied in [HKHZ99], corresponds to the special case of the observable signal being a pure-output signal and I being empty. So, in the algorithms below, we are given a Kripke structure $K = \langle AP, W, R, w_0, L\rangle$ that satisfies a CTL formula φ and an observable atom q, and we look for the set of states w such that $\tilde{K}_{w,q}$ does not satisfy φ.

The naive algorithm for computing the set of q-covered states performs model checking of φ in $\tilde{K}_{w,q}$ for each $w \in W$. Since CTL model-checking complexity is linear in the Kripke structure and the specification, the naive algorithm requires time $O(|K| \cdot |\varphi| \cdot |W|)$, which is quadratic in the size of structure[4]. Our symbolic algorithm returns an OBDD of the q-covered states. Our improved algorithm has an average running time of $O(|K| \cdot |\varphi| \cdot \log |W|)$.

3.1 Algorithm 1: a symbolic approach

Consider a Kripke structure $K = \langle AP, W, R, w_0, L\rangle$ and an atomic proposition $q \in AP$. For a CTL formula φ, we define

$$P(\varphi) = \{\langle w, v\rangle : \tilde{K}_{v,q}, w \models \varphi\}.$$

Thus, $P(\varphi) \subseteq W \times W$ contains exactly all pairs $\langle w, v\rangle$ such that w satisfies φ in the structure where we dualize the value of q in v. The definition of $P(\varphi)$ may not appear

[4] The algorithm for computing the set of q-covered states in [HKHZ99] runs in time $O(|K| \cdot |\varphi|)$. As we discuss in Section 1, however, the algorithm calculates the set of q-covered states according to a different definition of coverage, which we found less intuitive, and it handles only a restricted subset of ACTL.

helpful, as we are interested in the set of states in which dualizing the value of q falsifies φ. Nevertheless, the q-covered set in K for φ can be derived easily from $P(\varphi)$ as it is the set $\{w : \langle w_0, w\rangle \notin P(\varphi)\}$.

Our symbolic algorithm computes the OBDDs $P(\psi)$ for all $\psi \in cl(\varphi)$. The algorithm works bottom-up, and is based on the symbolic CTL model-checking algorithm. We assume that we have already performed symbolic model checking of φ in K; thus the sets $||\psi||$ have already been computed for all $\psi \in cl(\varphi)$. Let $S(\psi)$ denote the set of pairs $||\psi|| \times W$. In each step of the algorithm, it calculates $P(\psi)$ for some $\psi \in cl(\varphi)$, based on the sets $S(\psi')$ for $\psi' \in cl(\psi)$, and on the sets $P(\psi'')$ for $\psi'' \in cl(\psi) \setminus \{\psi\}$.

The algorithm uses the procedures $PairEX$ and $PairAX$ that take an OBDD $P(\psi)$ of pairs of states as an argument, and output an OBDD of pairs as follows: $PairEX(P(\psi)) = \{\langle w, v\rangle$: there exists a successor u of w such that $\langle u, v\rangle \in P(\psi)\}$. The procedure $PairAX(P(\psi))$ is defined dually. Thus, the procedures $PairEX$ and $PairAX$ work as in symbolic CTL model checking, only that here we compute sets of pairs instead of sets of singletons. The procedures apply the modalities to the first element of the pair, assuming the second is fixed.

We can now describe the algorithm. Given a CTL formula ψ, we define $P(\psi)$ according to the structure of ψ as follows (we describe here only the existential modalities. The universal ones are defined dually using $PairAX$, and are described in the full version).

- $\psi = p$ for an atomic proposition $p \neq q$. Obviously, changing the value of q in some state does not affect the value of p in any state. Therefore, $P(p) = \{\langle w, v\rangle : p \in L(w)\}$.
- $\psi = q$. Since the satisfaction of q in w depends only on the labeling of w, changing the value of q in some state v affects it iff $v = w$. Therefore, $P(q) = \{\langle w, v\rangle : w \neq v, q \in L(w)\} \cup \{\langle w, w\rangle : q \notin L(w)\}$.
- $\psi = \psi_1 \vee \psi_2$. A state w satisfies ψ in the dual structure $\tilde{K}_{v,q}$ iff it satisfies either ψ_1 or ψ_2 in $\tilde{K}_{v,q}$. Therefore, $P(\psi) = P(\psi_1) \cup P(\psi_2)$.
- $\psi = \neg\psi_1$. A state w satisfies ψ in the dual structure $\tilde{K}_{v,q}$ iff it does not satisfy $\neg\psi$ in $\tilde{K}_{v,q}$. Therefore, $P(\psi) = (W \times W) \setminus P(\psi_1)$.
- $\psi = EX\psi_1$. A state w satisfies ψ in the dual structure $\tilde{K}_{v,q}$ iff there exists a successor of w that satisfies ψ_1 in $\tilde{K}_{v,q}$. Therefore, $P(\psi) = PairEX(P(\psi_1))$.
- $\psi = E\psi_1 U \psi_2$. A state w satisfies ψ in the dual structure $\tilde{K}_{v,q}$ iff w satisfies ψ_2 in $\tilde{K}_{v,q}$, or w satisfies ψ_1 in $\tilde{K}_{v,q}$ and there exists a successor of w that satisfies ψ in $\tilde{K}_{v,q}$. Therefore, $P(\psi)$ is computed using the least fixed-point expression $\mu y.P(\psi_2) \vee (P(\psi_1) \wedge PairEX(y))$.

The symbolic algorithm for CTL model-checking uses a linear number of OBDD variables. The algorithm we present here doubles the number of OBDD variables, as it works with sets of pairs of states instead of sets of states. By the nature of the algorithm, it performs model-checking for all $\tilde{K}_{w,q}$ globally, and thus the OBDDs it computes contain information about the satisfaction of the specification in all the states of all the dual Kripke structures, and not only in their initial states.

The algorithm is described for the case that the Kripke structure K has one initial state, and can be easily extended to handle a set W_0 of initial states. Indeed, given the set $P(\varphi)$ of the set computed by the algorithm, the q-covered set in K for φ is the set of all states w such that there is an initial state $w_0 \in W_0$ such that $\langle w_0, w\rangle \notin P(\varphi)$.

3.2 Algorithm 2: improving average complexity

Consider a Kripke structure $K = \langle AP, W, R, w_0, L\rangle$, a formula φ, and an atomic proposition q. Recall that the naive CTL coverage algorithm, which performs model checking for all dual Kripke structures, has running time of $O(|K| \cdot |\varphi| \cdot |W|)$. While for some dual Kripke structures model-checking may require less than $O(|K| \cdot |\varphi|)$, the naive algorithm always performs $|W|$ iterations of model checking; thus, its average complexity cannot be substantially better than its worst-case complexity. This unfortunate situation arises even when model checking of two dual Kripke structures is practically the same, and even when some of the states of K obviously do not affect the satisfaction of φ in K. In this section we present an algorithm that makes use of such overlaps and redundancies. The expectant running time of our algorithm is $O(|K| \cdot |\varphi| \cdot \log |W|)$. Formally, we have the following:

Theorem 1. *The set q-cover(K, φ) can be computed in average running time of $O(|K| \cdot |\varphi| \cdot \log |W|)$, where the average is taken with respect to all possible assignments of q in K.*

All possible assignments of q in K are all possible labelings of the structure K with the observable signal q, where the value of q is chosen in each state of K to be **true** or **false** with equal probability.

Our algorithm is based on the fact that for each w, the dual Kripke structure $\tilde{K}_{w,q}$ differs from K only slightly. Therefore, there should be a large amount of work that we can share when we model check all the dual structures. In order to explain the algorithm, we introduce the notion of *incomplete model checking*. Informally, incomplete model checking of K is model checking of K with its labeling function L partially defined. The solution to the incomplete model checking problem can rely only on the truth values of the atomic propositions in states for which the corresponding L is defined. Obviously, in the general case we are not guaranteed to solve the model-checking problem without knowing the values of all atoms in all states. We can, however, perform some work in this direction, which is not needed to be performed again when missing parts of L are revealed.

Consider a partition of W into two equal sets, W_1 and W_2. Our algorithm essentially works as follows. For all the dual Kripke structures $\tilde{K}_{w,q}$ such that $w \in W_1$, the states in W_2 maintain their original labeling. Therefore, we start by performing incomplete model checking of φ in K with L that does not rely on the values of q in states in W_1. We end up in one of the following two situations. It may be that the values of q in states in W_2 (and the values of all the other atomic propositions in all the states) are sufficient to imply the satisfaction of φ in K. Then, we can infer that all the states in W_1 are not q-covered. It may also be that the values of q in states in W_2 are not sufficient to imply the satisfaction of φ in K. Then, we continue and partition the set W_1 into two equal sets, W_{11} and W_{12}, and perform incomplete model checking that does not rely on the values of q in states in W_{11}. The important observation is that incomplete model checking is now performed in a Kripke structure to which we have already applied incomplete model checking in the previous iteration. Thus, we only have to propagate information that involves the values of q in W_{12}. Thus, as we go deeper in the recursion described above, we perform less work. The depth of the recursion is bounded by $\log|W|$. As we shall analyze exactly below, the work in depth i amounts in average to model checking of φ in a Kripke structure of size $\frac{|K|}{2^i}$. Hence the $O(|K|\cdot|\varphi|\cdot\log|W|)$ complexity.

It is easier to understand and analyze incomplete model checking by means of the automata-theoretic approach to branching time model checking [KVW00]. In this approach, we transform the specification φ to an alternating automaton $\mathcal{A}_\varphi$ that accepts exactly all the models of φ. Model checking of K with respect to φ is then reduced to checking the nonemptiness of the product $\mathcal{A}_{K,\varphi}$ of K and $\mathcal{A}_\varphi$. When φ is in CTL, the automaton $\mathcal{A}_\varphi$ is linear in the length of φ, thus the product is of size $O(K\cdot|\varphi|)$.

The product $\mathcal{A}_{K,\varphi}$ can be viewed as a boolean circuit $G_{K,\varphi}$ (unlike boolean circuits, the product $\mathcal{A}_{K,\varphi}$ may contain cycles. In the full version we show how to handle these cycles) . 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 $\langle w,\neg p\rangle$. The formula φ is assumed to be in a positive normal form, thus negation applies only to the leaves of $G_{K,\varphi}$. The inner vertices of $G_{K,\varphi}$ are labeled with **true** or **false**, and the leaves are labeled with literals (variables or their negations). Initially, each leaf $\langle w,p\rangle$ or $\langle w,\neg p\rangle$ has a value, **true** or **false**, depending on the membership of p in $L(w)$. The graph has at most $2\cdot|AP|\cdot|W|$ leaves. Intuitively, incomplete model checking corresponds to *shrinking* a boolean circuit part of whose inputs are known to an equivalent minimal circuit. The shrinking procedure (described in detail in the full version of this paper) replaces for example, an OR-gate one of whose successors is a leaf with value **true**, with a leaf with value **true**. In each iteration of the algorithm we assign values to half of the unassigned leaves of $G_{K,\varphi}$ and leave the other half unassigned. Recall that our average complexity is with respect to all assignments of q in K. Therefore, though we work with a specific Kripke structure, and the values assigned to half of the leaves are these induced by the Kripke structure, the average case corresponds to one in which we randomly assign to each unassigned leaf the value **true** with probability $\frac{1}{4}$, the value **false** with probability $\frac{1}{4}$, and leave it unassigned with probability $\frac{1}{2}$. The complexity described in Theorem 1 then follows from the following result from circuit complexity.

Theorem 2. *Boolean circuits shrink by at least the factor of ϵ under a random assignment that leaves the fraction of ϵ variables unassigned and assigns* **true** *or* **false** *to other variables with equal probability.*

Theorem 2 follows from the fact that boolean circuits for parity and co-parity, which are the most “shrink-resistant” circuits (shrinking a parity or co-parity circuit according to a random partial assignment to the variables results in a parity or a co-parity circuit for the remaining variables), are linear in the number of their variables (see [Weg87,Nig90]). Let m be the size of the graph $G_{K,\varphi}$. By Theorem 2, incomplete model checking in depth i of the algorithm handles graphs of size $\frac{m}{2^i}$, and there are 2^i such graphs. Hence, the overall average running time is

$$O(m)+2\cdot O(m/2)+4\cdot O(m/4)+\ldots+2^{\log|W|}\cdot O(1),$$

which is equal to $O(m \cdot \log |W|)$. Since m is $O(|K| \cdot |\varphi|)$, the $O(|K| \cdot |\varphi| \cdot \log |W|)$ complexity follows.

Remark 1. In fact, the algorithm usually performs better. First, since the graph $G_{K,\varphi}$ is induced by a top-down reasoning, it may not contain leaves for all $\langle w, q \rangle \in W \times \{q\}$. States w such that $\langle w, q \rangle$ does not exist in $G_{K,\varphi}$ are obviously not q-covered, and the algorithm indeed ignores them. In addition, note that in the first iteration of the algorithm (that is, the first shrinking of $G_{K,\varphi}$), the unassigned leaves are exactly all the leaves of $G_{K,\varphi}$ of the form $\langle w, q \rangle$, and all the other leaves have values. While we cannot apply Theorem 2 directly and get shrinkage with $\epsilon = \frac{1}{|AP|}$ for this case (this is since we defined the average performance of the algorithm with respect to random assignments to q only, and since $G_{K,\varphi}$ typically contains less than $|W| \cdot |AP|$ leaves), we can still expect significant shrinkage in this call.

The algorithm can be easily adjusted to handle multiple initial states, where we check whether the modification falsifies the specification in some initial state. A naive adjustment repeats the algorithm for each of the initial states. A better adjustment handles all the initial states together, say by adding a new single initial state with transitions to all the initial states, and replacing the specification φ by the specification $AX\varphi$.

4 Presentation of the Output

Once we found the parts of the system that are not covered, there are several ways to make use of this information. For a circuit $\mathcal{S}$ and a signal x, let x-*cover* $(\mathcal{S}, \varphi)$ denote the set of states x-covered by φ in $\mathcal{S}$. One can compute x-*cover*$(\mathcal{S}, \varphi)$ for several output signals $x \in C \cup O$ (possibly, each of the properties that together compose the specification has a different set of signals that are of potential relevance to it). In [HKHZ99], coverage is defined as the ratio between the number of states that are members of x-*cover* $(\mathcal{S}, \varphi)$ for at least one signal x and the number of states in the design. This ratio indeed gives a good estimation to the part of the system that has been relevant for the verification process to succeed. In their implementation, Hoskote et al. also generate computations that lead to uncovered states.

We believe that once we worked hard and calculated x-*cover*$(\mathcal{S}, \varphi)$ for signals x, there are more helpful things that can be done with these sets. First, before merging x-covered sets for different signals, it is good to check whether x-*cover* $(\mathcal{S}, \varphi)$ is empty for some of the x's in isolation. An empty set may indicate vacuity in the specification (see also Section 5). In addition, we can use the sets x-*cover*$(\mathcal{S}, \varphi)$ in order to generate uncovered computations. Consider a state s of $\mathcal{S}$ that is not x-covered. The fact that s is not x-covered means that the specification fails to distinguish between $\mathcal{S}$ and the dual circuit $\tilde{\mathcal{S}}_{s,x}$. Therefore, possible errors caused by an erroneous value of x in s are not captured by the specification. So, a computation that contains the state s may be an interesting output. Even more interesting are computations that contain no covered states or many states that are not covered. Indeed, such computations correspond to behaviors of the circuit that are not referred to by the specification. Recall that the circuit models an open system. Thus, these computations are induced by input sequences that are ignored by the specification.

It is not hard to generate computations that consist of uncovered states only (if such computations exist) — we just have to restrict the transition function of $\mathcal{S}$ to uncovered states, which can also be done symbolically. In addition, computations that have many uncovered states can be found symbolically using the following greedy heuristic: we start from the initial state. Whenever we are in state s, we apply the *post* operator (given a set T of states, *post*(T) returns the set of successors of states in T) until we encounter a set of states that contains an uncovered state, from which we continue. Alternatively, we can start with maximal computations consisting of uncovered states only, and connect them with transitions via covered states.

The above methodology can be applied also to the logic-based definition of coverage. Obviously, if we discover that a control signal x can be fixed to 1 or to 0 without violating the specification, this means that x is useless in the circuit according to the specification. This should not happen in a correct implementation of a complete specification. Therefore, a valuable output in this case is the list of uncovered control signals.

Recall that model-checking tools accompany a negative answer to the correctness query by a counterexample to the satisfaction of the specification. When model checking succeeds, we suggest to accompany the positive reply with two computations: one is the *interesting witness* of [BBER97], which describes a behavior of the system that cares about the satisfaction of all the subformulas of the specification. The second is a *non-interesting witness*: a computation that is not covered by the specification. We believe that both outputs are of great importance to the user. While the first describes some nontrivial correct behavior of the system, the second describes a possibly incorrect behavior of the system that escaped the verification efforts.

5 Discussion

In this section we briefly discuss some more aspects of coverage metrics for temporal logic model checking. An extended discussion can be found in the full version.

Definition of coverage metrics There are several interesting possible relaxations of the definitions given in Section 2. One of them is allowing nondeterministic circuits. In nondeterministic circuits we can examine more coverage criteria: check the circuit obtained by merging s and $twin_x(s)$, check the circuit obtained by fixing a control signal to "don't care" (i.e., replace a transition to $\alpha \subseteq C$, with two transitions, to $\alpha \cup \{x\}$ and to $\alpha \setminus \{x\}$), etc. Another possibility is to allow different types of modifications in the system, for example flipping the value of q simultaneously in several states. Our definitions and algorithms can be easily modified to handle simultaneous modifications of the system. The corresponding algorithms, however, need to examine exponentially many modified systems and their complexity is much higher. Finally, we can think of δ as a function that maps the output signals to **true**, **false**, or *don't care*, thus allowing the designer to specify in advance that the values of certain signals are not important in some states. In this case we say that a system satisfies a specification φ if φ is satisfied no matter how q is assigned in states where its value is don't care. Our definitions of coverage apply also to designs with incomplete δ as above. Our algorithms can be easily adjusted to such designs.

Properties of coverage metrics The covered sets defined in Section 2 are *sensitive to abstraction*, in the sense that there is no correlation between the covered states in a system and its abstraction. From the other hand, the set of covered states is not sensitive to applying *cone of influence reduction*, where we abstract away parts of the systems that do not contain variables appearing in the specification or influence variables that appear in the specification [CGP99]. We also note that the notions of coverage and *vacuity* [BBER97,KV99] are closely related. Vacuity can be viewed as a coverage metric for the specification. Also, if there is a signal x that does not influence the satisfaction of φ in the system, then no state in the system is x-covered by φ. Our coverage metrics are *compositional*, in the sense that the intersection of the uncovered sets for the underlying conjuncts is equivalent to the uncovered set for their conjunction. Formally, if S_1 is the set of states not q-covered by φ_1 and S_2 is the set of states not q-covered by φ_2, then $S_1 \cap S_2$ is the set of states not q-covered by $\varphi_1 \wedge \varphi_2$. On the other hand, the coverage criteria defined in [KGG99], as well as the covered sets found by the algorithm in [HKHZ99] are not compositional.

There are still several open issues in the adoption of coverage metrics to temporal logic model checking. For example, it is not clear whether and how a coverage metric that aims at checking incompleteness of the specification should be different from a metric that aims at finding redundancies in the system. Another issue is the feasibility of coverage algorithms. While the algorithms that we presented have better complexity that the naive algorithm, their complexity is still larger than model checking. This may prevent a wide use of coverage metrics for temporal logic in formal verification. Clearly, there is room for improving the current algorithms, as well as searching for the new ones both for CTL and for other temporal logics, in particular LTL logic. Finally, while it is clear that outputs as these described in Section 4 are very helpful to the user, it is not clear whether high coverage is something one should seek for. Indeed, there is a trade-off between complete and abstract specifications, and neither completeness nor abstraction has clear priority.

Acknowledgment We thank Orna Grumberg, Yatin Hoskote, Amir Pnueli, and Uri Zwick for helpful discussions.

References

[BB94] D. Beaty and R. Bryant. Formally verifying a microprocessor using a simulation methodology. In *Proc. 31st DAC*, pp. 596–602. IEEE Computer Society, 1994.

[BBER97] I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. Efficient detection of vacuity in ACTL formulas. In *Proc. 9th CAV*, *LNCS* 1254, pp. 279–290, 1997.

[BCG88] M.C. Browne, E.M. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59:115–131, 1988.

[BH99] J.P. Bergmann and M.A. Horowitz. Improving coverage analysis and test generation for large designs. In *IEEE Int. Conf. for CAD*, pp. 580–584, 1999.

[CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic of Programs*, *LNCS* 131, pp. 52–71. Springer-Verlag, 1981.

[CGMZ95] E.M. Clarke, O. Grumberg, K.L. McMillan, and X. Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In *Proc. 32nd DAC*, pp. 427–432. IEEE Computer Society, 1995.

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

[CK93] K.-T. Cheng and A. Krishnakumar. Automatic functional test generation using the extended finite state machine model. In *Proc. 30th DAC*, pp. 86–91, 1993.

[DGK96] S. Devadas, A. Ghosh, and K. Keutzer. An observability-based code coverage metric for functional simulation. In *Proc. CAD*, pp. 418–425, 1996.

[Eme90] E.A. Emerson. Temporal and modal logic. *Handbook of Theoretical Computer Science*, pp. 997–1072, 1990.

[FAD99] F. Fallah, P. Ashar, and S. Devadas. Simulation vector generation from hdl descriptions for observability enhanced-statement coverage. In *Proc. 36th DAC*, pp. 666–671, 1999.

[FDK98] F. Fallah, S. Devadas, and K. Keutzer. OCCOM: Efficient Computation of Observability-Based Code Coverage Metrics for Functional Simulation. In *Proc. 35th DAC*, pp. 152–157, 1998.

[HH96] R.C. Ho and M.A. Horowitz. Validation coverage analysis for complex digital designs. In *Proc. CAD*, pp. 146–151, 1996.

[HKHZ99] Y. Hoskote, T. Kam, P.-H Ho, and X. Zhao. Coverage estimation for symbolic model checking. In *Proc. 36th DAC*, pp. 300–305, 1999.

[HMA95] Y. Hoskote, D. Moundanos, and J. Abraham. Automatic extraction of the control flow machine and application to evaluating coverage of verification vectors. In *Proc. ICDD*, pp. 532–537, 1995.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[HP85] D. Harel and A. Pnueli. On the development of reactive systems. *NATO Advanced Summer Institutes*, volume F-13, pp. 477–498. Springer-Verlag, 1985.

[HS96] G.D. Hachtel and F. Somenzi. *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publishers, MA, 1996.

[HYHD95] R. Ho, C. Yang, M. Horowitz, and D. Dill. Architecture validation for processors. In *Proc. of the 22nd Annual Symp. on Comp. Arch.*, pp. 404–413, 1995.

[KGG99] S. Katz, D. Geist, and O. Grumberg. “Have I written enough properties ?” a method of comparison between specification and implementation. In *10th CHARME*, *LNCS* 1703, pp. 280-297. Springer-Verlag, 1999.

[KN96] M. Kantrowitz and L. Noack. I'm done simulating: Now what? verification coverage analysis and correctness checking of the dec chip 21164 alpha microprocessor. In *Proc. of DAC*, pp. 325–330, 1996.

[Kur98] R.P. Kurshan. *FormalCheck User's Manual*. Cadence Design, Inc., 1998.

[KV99] O. Kupferman and M.Y. Vardi. Vacuity detection in temporal model checking. In *10th CHARME*, *LNCS* 1703. pp. 82-96, Springer-Verlag, 1999.

[KVW00] O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th POPL*, pp. 97–107, 1985.

[MAH98] D. Moumdanos, J.A. Abraham, and Y.V. Hoskote. Abstraction techniques for validation coverage analysis and test generation. *IEEE Trans. on Computers*, 1998.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.

[Nig90] R. G. Nigmatulin. *The Complexity of Boolean Functions*. Nauka, Main Editorial Board for Phys. and Math. Lit., Moscow, 1990.

[QS81] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 5th Int. Symp. on Progr.*, *LNCS* 137, pp. 337–351. Springer-Verlag, 1981.

[Weg87] I. Wegener. *The Complexity of Boolean Functions*. John Wiley & Sons, 1987.