A Practical Approach to Coverage in Model Checking

Hana Chockler[1] Orna Kupferman[1]* Robert P. Kurshan[2] Moshe Y. Vardi[3]**

[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] Bell Laboratories, 700 Mountain Avenue, Murray Hill, NJ 07974, U.S.A.
Email: k@research.bell-labs.com
[3] 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. When verification succeeds and 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 from a practical point of view. 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 suggest several definitions of coverage, suitable for specifications given in linear temporal logic or by automata on infinite words. We describe two algorithms for computing the parts of the system that are not covered by the specification. The first algorithm is built on top of automata-based model-checking algorithms. The second algorithm reduces the coverage problem to the model-checking problem. Both algorithms can be implemented on top of existing model checking tools.

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 Kripke structure that models the system satisfies a specification of this behavior, expressed in terms of a temporal logic formula or a finite automaton [CGP99]. 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 to the importance of suspecting the system of containing an error also in the case model checking

* Supported in part by BSF grant 9800096.
** Supported in part by NSF grant CCR-9700061, NSF grant CCR-9988322, BSF grant 9800096, 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 way 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 $\varphi = G(req \rightarrow F\,grant)$ is vacuously satisfied in a system in which *req* is always **false**. A similar way 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 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 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 that is not captured by the specification also escapes the attention of the designer, who is often the one to provide the specification.

In simulation-based verification techniques, coverage metrics are used in order to reveal states that were not visited during the testing procedure (i.e, not "covered" by this procedure) [HMA95,HYHD95,DGK96,HH96,KN96,FDK98,MAH98,BH99,FAD99]. These metrics are a useful way of measuring progress of the verification process. However, the same intuition cannot be applied to model checking because the process of model checking visits all states. We can say that in testing, a state is "uncovered" if it is not essential to the success of the testing procedure. The similar idea can be applied to model checking, where the state is defined as "uncovered" if its labeling is not essential to the success of the model checking process. This approach was first suggested by Hoskote et al. [HKHZ99]. 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. Then, the output of a coverage check is helpful in completing the specification. Another possibility is that the system contains redundancies. Then, the output of the coverage check is helpful in simplifying the system.

There are two different approaches to coverage in model checking. One approach, introduced by Katz et al. [KGG99], states that a well-covered system should closely resemble the tableau of its specification, thus the coverage criteria of [KGG99] are based on the analysis of the differences between the system and the tableau of its specification. We find the approach of [KGG99] too strict – we want specifications to be much more abstract than their implementations. In addition, the approach is restricted to universal safety specifications, whose tableaux have no fairness constraints, and it is computationally hard to compute the coverage criteria. Another approach, introduced in [HKHZ99], is to check the influence of small changes in the system on the satisfaction of the specification. Intuitively, if a part of the system can be changed without violating the specification, this part is *uncovered* by the specification. Formally, for a Kripke structure K, a state w in K, and an *observable signal* q, the *dual structure*

$\tilde{K}_{w,q}$ is obtained from K by *flipping* the value of q in w (the signal q corresponds to a Boolean variable that is **true** if w is labeled with q and is **false** otherwise. When we say that we flip the value of q, we mean that we switch the value of this variable). For a specification φ, Hoskote et al. define the set q-*cover*(K, φ) as a set of states w such that $\tilde{K}_{w,q}$ does not satisfy φ. A state is covered if it belongs to q-*cover*(K, φ) for some observable signal q. Indeed, this indicates that the value of q in w is crucial for the satisfaction of φ in K. It is easy to see that for each observable signal, the set of covered states can be computed by a naive algorithm that performs model checking of φ in $\tilde{K}_{w,q}$ for each state w of K. The naive algorithm, however, is very expensive, and is useless for practical applications [1]. In [CKV01], we suggested two alternatives to the naive algorithm for specifications in the branching time temporal logic CTL. The first algorithm is symbolic and it computes the set of pairs $\langle w, w' \rangle$ such that flipping the value of q in w' falsifies φ in w. The second algorithm improves the naive algorithm by exploiting overlaps in the many dual structures that we need to check. The two algorithms are still not attractive: the symbolic algorithm doubles the number of BDD's variables, and the second algorithm requires the development of new procedures. Also, these algorithms cannot be extended to specifications in LTL, as they heavily use the fixed-point characterization of CTL, which is not applicable to LTL.

In this paper we study coverage metrics for model checking from a practical point of view. First, we consider specifications given as formulas in the linear temporal logic LTL or by automata on infinite words. These formalisms are used in many model-checking tools (e.g., [HHK96,Kur98]), and we suggest alternative definitions of coverage, which suit better the linear case. Second, we describe two algorithms for LTL specifications. Both algorithms can be relatively easily implemented on top of existing model checking tools.

Let us describe informally our alternative definitions. For a Kripke structure K, let $\mathcal{K}$ be the unwinding of K to an infinite tree. Recall that a dual structure $\tilde{K}_{w,q}$ is obtained in [HKHZ99,CKV01] by flipping the value of the signal q in the state w of K. A state w of K may correspond to many w-*nodes* in $\mathcal{K}$. The definition of coverage that refers to $\tilde{K}_{w,q}$ flips the value of q in all the w-nodes in $\mathcal{K}$. We call this *structure coverage*. Alternatively, we can examine also *node coverage*, where we flip the value of q in a single w-node in $\mathcal{K}$, and *tree coverage*, where we flip the value of q in some w-nodes. Each approach measures a different sensitivity of the satisfaction of the specification to changes in the system. Intuitively, in structure coverage we check whether the value of q in all the occurrences of w has been crucial for the satisfaction of the specification. On the other hand, in node coverage we check whether the value of q in some occurrence of w has been crucial for the satisfaction of the specification[2].

[1] Hoskote et al. describe an alternative algorithm that is symbolic and runs in linear time, but their algorithm handles specifications in a very restricted syntax (a fragment of the universal fragment ∀CTL of CTL) and it does not return the set q-*cover*(K, φ), but a set that corresponds to a different definition of coverage, which is sometimes 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 $q \to q$ is the set of states that satisfy q, rather than the empty set, which meets our intuition of coverage.

[2] As we show in Section 2.2, this intuition is not quite precise, and node coverage does not imply structure coverage, which is why tree coverage is required.

The first algorithm we describe computes the set of node-covered states and is built on top of automata-based model-checking algorithms. In automata-based model checking, we translate an LTL specification φ to a nondeterministic Büchi automaton $\mathcal{A}_{\neg\varphi}$ that accepts all words that do not satisfy φ [VW94]. Model checking of K with respect to φ can then be reduced to checking the emptiness of the product $K \times \mathcal{A}_{\neg\varphi}$. When K satisfies φ, the product is empty. A state w is covered iff flipping the value of q in w makes the product nonempty. This observation enables us to compute the set of node covered states by a simple manipulation of the set of reachable states in the product $K \times \mathcal{A}_{\neg\varphi}$, and the set of states in this product from which a fair path exists. Fortunately, these sets have already been calculated in the process of model checking. We describe an implementation of this algorithm in the tool COSPAN, which is the engine of FormalCheck [HHK96,Kur98]. We also describe the changes in the implementation that are required in order to adapt the algorithm to handle structure and tree coverage.

In the second algorithm we reduce the coverage problem to model checking. Given an LTL specification φ and an observable signal q, we construct an *indicator* formula $\mathit{Ind}_q(\varphi)$, such that for every structure K and state w in K, the state w is node q-covered by φ iff w satisfies $\mathit{Ind}_q(\varphi)$. The indicator formulas we construct are in μ-calculus with both past and future modalities, their length is, in the worst case, exponential in the size of the specification φ, they are of alternation depth two for general LTL specifications, and are alternation free for safety LTL specifications. We note that the exponential blow-up may not appear in practice. Also, tools that support symbolic model checking of μ-calculus with future modalities can be extended to handle past modalities with no additional cost [KP95]. In the full version of the paper we show that bisimilar states may not agree on their coverage, which is why the indicators we construct require both past and future modalities.

The two algorithms that we present in this paper are derived from the two possible approaches to linear-time model checking. The first approach is to analyze the product of the system with the automaton of the negation of the property. The second approach is to translate the property to a μ-calculus formula and then check the system with respect to this formula. Both approaches may involve exponential blow-up. In the first approach, the size of the automaton can be exponential in the size of the property, and in the second approach the size of the μ-calculus formula can be exponential in the size of the property.

2 Preliminaries

2.1 Structures and trees

We model systems by 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 $L : W \to 2^{AP}$. If $R(w, w')$, we say that w' is a successor of w. For a state $w \in W$, a w-*path* $\pi = w_0, w_1, \ldots$ in K is a sequence of states in K such that $w_0 = w$ and for all $i \geq 0$, we have $R(w_i, w_{i+1})$. If $w_0 = w_{in}$, the path π is called an *initialized path*. The labeling function L can be extended to paths in a straightforward way, thus $L(\pi) = L(w_0) \cdot L(w_1) \cdots$ is an infinite word over the alphabet 2^{AP}. A *fair Kripke structure* is a Kripke structure augmented with a fairness constraint. We consider here the Büchi fairness condition. There, $K = \langle AP, W, R, w_{in}, L, \alpha \rangle$, where $\alpha \subseteq W$ is a set of fair states. A path of K is *fair* if it visits states in α infinitely often. Formally, let $inf(\pi)$ denote the set of states repeated in π infinitely often. Thus, $w \in inf(\pi)$ iff $w_i = w$ for infinitely many i's. Then, π is fair iff $inf(\pi) \cap \alpha \neq \emptyset$. The *language* of K, denoted $\mathcal{L}(K)$ is the set of words $L(\pi)$ for the initialized fair paths π of K. Often, it is convenient to have several initial states in K. Our results hold also for this model.

For a finite set Υ, an Υ-tree T is a set $T \subseteq \Upsilon^*$ such that if $x \cdot \upsilon \in T$ where $x \in \Upsilon^*$ and $\upsilon \in \Upsilon$, then also $x \in T$. The elements of T are called *nodes* and the empty word ε is the *root* of T. For every $x \in T$, the nodes $x \cdot \upsilon \in T$ where $\upsilon \in \Upsilon$ are the *children* of x. Each node x of T has a *direction* in Υ. The direction of the root is some designated member of Υ, denoted by υ_0. The direction of a node $x \cdot \upsilon$ is υ. We denote by $dir(x)$ the direction of node x. A node x such that $dir(x) = \upsilon$ is called υ-*node*. A *path* ρ of a tree T is a set $\rho \subseteq T$ such that $\varepsilon \in \rho$ and for every $x \in \rho$ there exists a unique $\upsilon \in \Upsilon$ such that $x \cdot \upsilon \in \rho$. For an alphabet Σ, a Σ-*labeled* Υ-tree is a pair $\langle T, V \rangle$, where $V : T \to \Sigma$ labels each node of T with a letter from Σ.

A Kripke structure K can be unwound into an infinite computation tree in a straightforward way. Formally, the tree that is obtained by unwinding K is denoted by $\mathcal{K}$ and is the 2^{AP}-labeled W-tree $\langle T^K, V^K \rangle$, where $\varepsilon \in T^K$ and $dir(\varepsilon) = w_{in}$, for all $x \in T^K$ and $v \in W$ with $R(dir(x), v)$, we have $x \cdot v \in T^K$, and for all $x \in T^K$, we have $V^K(x) = L(dir(x))$. That is, V^K maps a node that was reached by taking the direction w to $L(w)$.

2.2 Coverage

Given a system and a formula that is satisfied in this system, we check the influence of modifications in the system on the satisfaction of the formula. Intuitively, a state is *covered* if a modification in this state falsifies the formula in the initial state of the structure. We limit ourselves to modifications that flip the value of one atomic proposition (an observable signal) in one state of the structure[3]. Flipping can be performed in different ways. Through the execution of the system we can visit a state several times, each time in a different context. This gives rise to a distinction between "flipping always", "flipping once", and "flipping sometimes", which we formalize in the definitions of *structure coverage*, *node coverage*, and *tree coverage* below. We first need some notations.

For a domain Y, a function $V : Y \to 2^{AP}$, an observable signal $q \in AP$, and a set $X \subseteq Y$, the *dual function* $\tilde{V}_{X,q} : Y \to 2^{AP}$ is such that $\tilde{V}_{X,q}(x) = V(x)$ for all $x \notin X$, $\tilde{V}_{X,q}(x) = V(x) \setminus \{q\}$ if $x \in X$ and $q \in V(x)$, and $\tilde{V}_{X,q}(x) = V(x) \cup \{q\}$ if $x \in X$ and $q \notin V(x)$. When $X = \{x\}$ is a singleton, we write $\tilde{V}_{x,q}$. For a Kripke structure $K = \langle AP, W, R, w_{in}, L \rangle$, an observable signal $q \in AP$, and a state $w \in W$, we denote by $\tilde{K}_{w,q}$ the structure obtained from K by flipping the value of q in w. Thus, $\tilde{K}_{w,q} = \langle AP, W, R, W_{in}, \tilde{L}_{w,q} \rangle$, where $\tilde{L}_{w,q}(v) = L(v)$ for $v \neq w$, $\tilde{L}_{w,q}(w) = L(w) \cup \{q\}$, in

[3] In [CKV01], we consider richer modifications (e.g., modifications that change both the labeling and the transitions), and show how the algorithms described there for the limited case can be extended to handle richer modifications. Extending the algorithms described in this paper to richer modifications is nontrivial.

case $q \notin L(w)$, and $\tilde{L}_{w,q}(w) = L(w) \setminus \{q\}$, in case $q \in L(w)$. For $X \subseteq T^K$ we denote by $\tilde{\mathcal{K}}_{X,q}$ the tree that is obtained by flipping the value of q in all the nodes in X. Thus, $\tilde{\mathcal{K}}_{X,q} = \langle T^K, \tilde{V}^K_{X,q}\rangle$. When $X = \{x\}$ is a singleton, we write $\tilde{\mathcal{K}}_{x,q}$.

Definition 1. *Consider a Kripke structure K, a formula φ satisfied in K, and an observable signal $q \in AP$.*

- *A state w of K is* structure q-covered *by φ iff the structure $\tilde{K}_{w,q}$ does not satisfy φ.*
- *A state w of K is* node q-covered *by φ iff there is a w-node x in T^K such that $\tilde{\mathcal{K}}_{x,q}$ does not satisfy φ.*
- *A state w of K is* tree q-covered *by φ iff there is a set X of w-nodes in T^K such that $\tilde{\mathcal{K}}_{X,q}$ does not satisfy φ.*

Note that, alternatively, a state is structure q-covered iff $\tilde{\mathcal{K}}_{X,q}$ does not satisfy φ for the set X of all w-nodes in $\mathcal{K}$. In other words, a state w is structure q-covered if flipping the value of q in all the instances of w in $\mathcal{K}$ falsifies φ, it is node q-covered if a single flip of the value of q falsifies φ, and it is tree q-covered if some flips of the value of q falsifies φ.

For a Kripke structure $K = \langle AP, W, R, w_{in}, L\rangle$, an LTL formula φ, and an observable signal $q \in AP$, we use $SC(K, \varphi, q)$, $NC(K, \varphi, q)$, and $TC(K, \varphi, q)$, to denote the sets of states that are structure q-covered, node q-covered, and tree q-covered, respectively in K.

Membership of a given state w in each of the sets above can be decided by running an LTL model checking algorithm on modified structures. For $SC(K, \varphi, q)$, we have to model check $\tilde{K}_{w,q}$. For $NC(K, \varphi, q)$ and $TC(K, \varphi, q)$, things are a bit more complicated, as we have to model check several (possibly infinitely many) trees. Since, however, the set of computations in these trees is a modification of the language of K, it is possible to obtain these computations by modifying K as follows. For tree coverage, we model check the formula φ in the structure obtained from K by adding a copy w' of the state w in which q is flipped. Node coverage is similar, only that we have to ensure that the state w' is visited only once, which can be done by adding a copy of K to which we move after a visit in w'. It follows that the sets $SC(K, \varphi, q)$, $NC(K, \varphi, q)$, and $TC(K, \varphi, q)$ can be computed by a naive algorithm that runs the above checks $|W|$ times, one time for each state w. In Sections 3 and 4 we describe two alternatives to this naive algorithm.

We now study the relation between the three definitions. It is easy to see that structure and node coverage are special cases of tree coverage, thus $SC(K, \varphi, q) \subseteq TC(K, \varphi, q)$ and $NC(K, \varphi, q) \subseteq TC(K, \varphi, q)$ for all K, φ, and q. The relation between structure coverage and node coverage, however, is not so obvious. Intuitively, in structure coverage we check whether the value of q in all the occurrences of w has been crucial for the satisfaction of the specification. On the other hand, in node coverage we check whether the value of q in some occurrence of w has been crucial for the satisfaction of the specification. It may therefore seem that node coverage induces bigger covered sets. The following example shows that in that general case neither one of the covered sets $SC(K, \varphi, q)$ and $NC(K, \varphi, q)$ is a subset of the other. Let K be a Kripke structure with one state w, labeled q, with a self-loop. Let $\varphi_1 = Fq$. It is easy to see that

$\tilde{K}_{w,q}$ does not satisfy φ_1. On the other hand, $\mathcal{K}$ is an infinite tree that is labeled with q everywhere, thus $\tilde{\mathcal{K}}_{x,q}$ satisfies φ_1 for every node x. So, w is structure q-covered, but not node q-covered. Now, let $\varphi_2 = Gq \vee G\neg q$. It is easy to see that $\tilde{K}_{w,q}$ satisfies φ_2. On the other hand, $\tilde{\mathcal{K}}_{x,q}$ is a tree that is labeled with q in all nodes $y \neq x$, thus $\tilde{\mathcal{K}}_{x,q}$ does not satisfy φ_2. So, w is tree q-covered, but it is not structure q-covered. As a corollary, we get the following theorem.

Theorem 1. *There is a Kripke structure K, LTL formulas φ_1 and φ_2, and an observable signal q such that $SC(K, \varphi_1, q) \not\subseteq NC(K, \varphi_1, q)$ and $NC(K, \varphi_2, q) \not\subseteq SC(K, \varphi_2, q)$.*

2.3 Automata

A *nondeterministic Büchi automaton* over infinite words is $\mathcal{A} = \langle \Sigma, S, \delta, S_0, \alpha \rangle$, where Σ is an alphabet, S is a set of states, $\delta : S \times \Sigma \to 2^S$ is a transition relation, $S_0 \subseteq S$ is a set of initial states, and $\alpha \subseteq S$ is the set of accepting states. Given an infinite word $\tau = \sigma_0 \cdot \sigma_1 \cdots$ in Σ^ω, a run r of $\mathcal{A}$ on τ is an infinite sequence of states $s_0, s_1, s_2 \ldots$ such that $s_0 \in S_0$ and for all $i \geq 0$, we have $s_{i+1} \in \delta(s_i, \sigma_i)$. The set $inf(r)$ is the set of states that appear in r infinitely often. Thus, $s \in inf(r)$ iff $s_i = s$ for infinitely many i's. The run r is *accepting* iff $inf(r) \cap \alpha \neq \emptyset$ [Büc62]. That is, a run is accepting iff it visits some accepting state infinitely often. The language of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of infinite words $\tau \in \Sigma^\omega$ such that there is an accepting run of $\mathcal{A}$ on τ. Finally, for $s \in S$, we define $\mathcal{A}^s = \langle \Sigma, S, \delta, \{s\}, \alpha \rangle$ as $\mathcal{A}$ with initial set $\{s\}$.

We assume that specifications are given either by LTL formulas or by nondeterministic Büchi automata. It is shown in [VW94] that given an LTL formula φ, we can construct a nondeterministic Büchi automaton $\mathcal{A}_\varphi$ over the alphabet 2^{AP} such that $\mathcal{A}_\varphi$ accepts exactly all the words that satisfy φ. Formally, $\mathcal{L}(\mathcal{A}_\varphi) = \{\tau \in (2^{AP})^\omega : \tau \models \varphi\}$.

3 An automata-based algorithm for computing coverage

In this section we extend automata-based model-checking algorithms to find the set of covered states. In automata-based model checking, we translate an LTL specification φ to a nondeterministic Büchi automaton $\mathcal{A}_{\neg\varphi}$ that accepts all words that do not satisfy φ [VW94]. Model checking of K with respect to φ can then be reduced to checking the emptiness of the product $K \times \mathcal{A}_{\neg\varphi}$. Let $K = \langle AP, W, R, w_{in}, L \rangle$ be a Kripke structure that satisfies φ, and let $\mathcal{A}_{\neg\varphi} = \langle 2^{AP}, S, \delta, S_0, \alpha \rangle$ be the nondeterministic Büchi automaton for $\neg\varphi$. The product of K with $\mathcal{A}_{\neg\varphi}$ is the fair Kripke structure $K \times \mathcal{A}_{\neg\varphi} = \langle AP, W \times S, M, \{w_{in}\} \times S_0, L', W \times \alpha \rangle$, where $M(\langle w, s \rangle, \langle w', s' \rangle)$ iff $R(w, w')$ and $s' \in \delta(s, L(w))$, and $L'(\langle w, s \rangle) = L(w)$. Note that an infinite path π in $K \times \mathcal{A}_{\neg\varphi}$ is fair iff the projection of π on S satisfies the acceptance condition of $\mathcal{A}_{\neg\varphi}$. Since K satisfies φ, we know that no initialized path of K is accepted by $\mathcal{A}_{\neg\varphi}$. Hence, $\mathcal{L}(K \times \mathcal{A}_{\neg\varphi})$ is empty.

Let $P \subseteq W \times S$ be the set of pairs $\langle w, s \rangle$ such that $\mathcal{A}_{\neg\varphi}$ can reach the state s as it reads the state w. That is, there exists a sequence $\langle w_0, s_0 \rangle, \ldots, \langle w_k, s_k \rangle$ such that $w_0 = w_{in}$, $s_0 \in S_0$, $w_k = w$, $s_k = s$, and for all $i \geq 0$ we have $R(w_i, w_{i+1})$

and $s_{i+1} \in \delta(s_i, L(w_i))$. Note that $\langle w, s\rangle \in P$ iff $\langle w, s\rangle$ is reachable in $K \times \mathcal{A}_{\neg\varphi}$. For an observable signal $q \in AP$ and $w \in W$, we define the set $P_{w,q} \subseteq W \times S$ as the set of pairs $\langle w', s'\rangle$ such that w' is a successor of w and $\mathcal{A}_{\neg\varphi}$ can reach the state s' as it reads the state w' in a run in which the last occurrence of w has q flipped. Formally, if we denote by $\tilde{L}_q : W \rightarrow 2^{AP}$ the labeling function with q flipped (that is, $\tilde{L}_q(w) = L(w) \cup \{q\}$ if $q \notin L(w)$, and $\tilde{L}_q(w) = L(w) \setminus \{q\}$ if $q \in L(w)$), then

$$P_{w,q} = \{\langle w', s'\rangle : \text{ there is } s \in S \text{ such that } \langle w, s\rangle \in P, R(w, w'), \text{ and } s' \in \delta(s, \tilde{L}_q(w))\}.$$

Recall that a state w is node q-covered in K iff there exists a a w-node x in T^K such that $\tilde{\mathcal{K}}_{x,q}$ does not satisfy φ. We can characterize node q-covered states also as follows (see the full version for the proof).

Theorem 2. *Consider a Kripke structure K, an LTL formula φ, and an observable signal q. A state w is node q-covered in K by φ iff there is a successor w' of w and a state s' such that $\langle w', s'\rangle \in P_{w,q}$ and there is a fair $\langle w', s'\rangle$-path in $K \times \mathcal{A}_{\neg\varphi}$.*

Theorem 2 reduces the problem of checking whether a state w is node q-covered to computing the relation $P_{w,q}$ and checking for the existence of a fair path from a state in the product $K \times \mathcal{A}_{\neg\varphi}$. Model-checking tools compute the relation P and compute the set of states from which we have fair paths. Therefore, Theorem 2 suggests an easy implementation for the problem of computing the set of node-covered states. We describe a possible implementation in the tool COSPAN, which is the engine of FormalCheck [HHK96,Kur98]. We also show that the implementation can be modified in order to handle structure and tree coverage.

In COSPAN, the system is modeled by a set of modules, and the desired behavior is specified by an additional module $\mathcal{A}$. The language $\mathcal{L}(\mathcal{A})$ is exactly the set of wrong behaviors, thus the module $\mathcal{A}$ stands for the automaton $\mathcal{A}_{\neg\varphi}$ in cases the specification is given an LTL formula φ. In order to compute the set of node q-covered states, the system has to nondeterministically choose a step in the synchronous composition of the modules, in which the value of q is flipped in all modules that refer to q. Note that this is the same as to choose a step in which the module $\mathcal{A}$ behaves as if it reads the dual value of q. This can be done by introducing two new Boolean variables *flip* and *flag*, local to $\mathcal{A}$. The variable *flip* is nondeterministically assigned **true** or **false** in each step. The variable *flag* is initialized to **true** and is set to **false** one step after *flip* becomes **true**. Instead of reading q, the module $\mathcal{A}$ reads $q \oplus (\mathit{flip} \wedge \mathit{flag})$. Thus, when both *flip* and *flag* hold, which happens exactly once, the value of q is flipped ($\oplus$ stands for exclusive or). So, the synchronous composition of the modules is not empty iff the state that was visited when *flip* becomes **true** for the first time is node q-covered. The complexity of model checking is linear in the size of the state space of the model, which is bounded by $O(2^n)$, where n is the number of state variables. We increase the number of state variables by 2, thus the complexity of coverage computation is still $O(2^n)$.

With a small change in the implementation we can also check tree coverage. Since in tree coverage we can flip the value of q several times, the variable *flag* is no longer needed. Instead, we need $\log |W|$ variables $x_1, \ldots, x_{\log |W|}$ for encoding the state w that is now being checked for tree q-coverage. The state w is not known in advance and the variables $x_1, \ldots, x_{\log |W|}$ are initialized non-deterministically and then kept unchanged to maintain the encoding of some state of the system. The variable *flip* is nondeterministically assigned **true** or **false** in each step. Instead of reading q, the module $\mathcal{A}$ reads $q \oplus (\mathit{flip} \wedge at\_w)$, where at_w holds iff the encoding of the current state coincides with $x_1, \ldots, x_{\log |W|}$. Thus, when both *flip* and at_w hold, which may happen several times, yet only when the current state is w, the value of q is flipped. So, the synchronous composition of the modules is not empty iff the state that was visited when *flip* becomes **true** for the first time is tree q-covered. Finally, by nondeterministically choosing the values of $x_1, \ldots, x_{\log |W|}$ at the first step of the run and fixing *flip* to **true**, we can also check structure coverage.

The complexity of coverage computation for tree and structure coverage is a function of the size of the state space, which is at most exponential in the number of state variables. For both tree and structure coverage, we double the number of variables by introducing n new variables that encode the flipped state. Thus, the state-space size is $O(2^{2n})$ instead of $O(2^n)$. While symbolic algorithms may have the same worst-case complexity as enumerative algorithms, in practice they are typically superior for many classes of applications. We believe that there is an ordering of the BDD variables that would circumvent the worst-case complexity. On the other hand, the naive approach always require 2^n model-checking iterations. Thus, our algorithm is likely to perform better than the naive approach.

In our definitions of coverage we assumed that a change in the labeling of states does not affect the transitions of the system. This is why the transitions of the modules that model the behavior of the system remain unchanged when flipping happens. A different definition, which involves changes in the transition relation is required when we assume that the states are encoded by atomic propositions in AP and the transition relation is given as a relation between values of the atomic propositions in AP. Then, flipping q in a state w causes changes in the transitions to and from w [CKV01]. Thus, in this case it is not enough to change the module $\mathcal{A}$ in order to compute the covered sets and we also have to change the modules of the system. This can be achieved by defining the variables *flip* and *flag* globally, and referring to their value in all modules of the system. This involves a broader change in the source code of the model.

Note that our algorithm is independent of the fairness condition being Büchi, and it can handle any fairness condition for which the model-checking procedure supports the check for fair paths. Also, it is easy to see that the same algorithm can handle systems with multiple initial states.

4 Indicators for LTL formulas

In this section we reduce the computation of node q-covered sets to model checking. Given an LTL formula φ and an observable signal q, we want to find an *indicator* formula for φ that distinguishes between the covered and uncovered states in all Kripke structures. Formally, we have the following.

Definition 2. *Given an LTL formula φ and an observable signal q, an indicator for φ and q is a formula $Ind_q(\varphi)$ such that for all Kripke structures K that satisfy φ, we have*

$$\{w \in W : w \models Ind_q(\varphi)\} = NC(K, \varphi, q).$$

The motivation of indicators is clear. Once $Ind_q(\varphi)$ is found, global model-checking procedures can return the set of node q-covered states.

We show that for LTL formulas, we can construct indicators in the full μ-calculus, where we allow both future and past modalities (see [Koz83] for a definition of μ-calculus with future modalities). Formally, the full μ-calculus for a set AP of atomic propositions and the set *Var* of variables includes the following formulas:

- **true**, **false**, p, for $p \in AP$, and y, for $y \in Var$.
- $\neg\varphi_1$, $\varphi_1 \vee \varphi_2$, and $\varphi_1 \wedge \varphi_2$ for full μ-calculus formulas φ_1 and φ_2.
- $AX\varphi$, $EX\varphi$, $AY\varphi$, and $EY\varphi$ for full μ-calculus formula φ.
- $\mu y.\varphi(y)$ and $\nu y.\varphi(y)$, where $y \in Var$ and φ is a full μ-calculus formula monotone in y.

A *sentence* is a formula that contains no free atomic proposition variables. The semantics of full μ-calculus sentences is defined with respect to Kripke structures. The semantics of the path quantifiers A ("for all paths") and E ("there exists a path"), and the temporal operators X ("next"), and Y ("yesterday") assumes that both future and past are branching [KP95]. That is, for a state w, we have $w \models AX\varphi$ iff for all v such that $R(w, v)$, we have $v \models \varphi$, and $w \models AY\psi$ iff for all u such that $R(u, w)$, we have $u \models \psi$. We assume that the initial states of the Kripke structure are labeled with a special atomic proposition $init$ ($w_0 \models AY$**false** and $init \not\models AY$**true**).

The construction of $Ind_q(\varphi)$ proceeds as follows. We first construct a formula, denoted Ψ, that describes $\mathcal{A}_{\neg\varphi}$. The formula Ψ is a disjunction of formulas ψ_s, for states s of $\mathcal{A}_{\neg\varphi}$, and it describes states of $\mathcal{A}_{\neg\varphi}$ that participate in an accepting run of $\mathcal{A}_{\neg\varphi}$. For each state s, the formula ψ_s is the conjunction of two formulas, $Reach_s$ and Acc_s, defined as follows.

- The formula $Reach_s$ is satisfied in a state w of a Kripke structure K iff there exists a run of $\mathcal{A}_{\neg\varphi}$ on an initialized path of K that visits the state s as it reads w.
- The formula Acc_s is satisfied in a state w of a Kripke structure K iff there exists an accepting run of $\mathcal{A}^s_{\neg\varphi}$ on a w-path of K (recall that $\mathcal{A}^s_{\neg\varphi}$ is defined as $\mathcal{A}_{\neg\varphi}$ with initial set $\{s\}$).

Then, $\Psi = \bigvee_{s\in S} Reach_s \wedge Acc_s$. So, for every Kripke structure K, a state w in K satisfies Ψ iff there exists a state $s \in S$ such that there exists an accepting run of $\mathcal{A}_{\neg\varphi}$ on an initialized path of K that visits the state s as it reads w. The formulas $Reach_s$ refer to the past and are constructed as in [HKQ98] using past modalities. The formulas Acc_s refer to the future and are constructed as in [EL86,BC96], using future modalities [4].

Note that $K \not\models \varphi$ iff there exists $w \in W$ such that $w \models \Psi$. Since K satisfies φ, there is no state $w \in W$ that satisfies Ψ. Our goal is to find the node q-covered states of K. These are the states that satisfy Ψ after a flip of the value of q in them [5]. In order

[4] The algorithms in [HKQ98,EL86,BC96] construct μ-systems of equational blocks, and not μ-calculus formulas. The translation from μ-formulas to μ-systems may involve an exponential blow-up. While the our algorithm is described here in terms of μ-calculus formulas, we can work with μ-systems directly. The operators $\wedge$ and $\vee$ on μ-formulas are defined on the systems of equational blocks as well.

[5] This semantics naturally translates to node coverage. For structure and tree coverage other definitions are needed.

to simulate such a flip, we have to separate between the part that describes present behavior, and the parts that describe past or future behavior in the formulas $Reach_s$ and Acc_s, respectively. For that, we first replace all μ-calculus formulas by equivalent *guarded* formulas. A μ-calculus formula is *guarded* if for all $y \in Var$, all the occurrences of y are in the scope of X or Y [BB87]. It is shown in [KVW00] that given a μ-calculus formula, we can construct an equivalent guarded formula in linear time. Then, in order to separate the part that describes present behavior, we replace each formula $\mu y.f(y)$ by the equivalent formula $f(\mu y.f(y))$. For example, the formula $\mu y.p \vee AXy$ is replaced by $p \vee AX\mu y.p \vee AXy$. In fact, when constructed as in [HKQ98], the formulas $Reach_s$ are already pure-past formulas, they do not refer to the present, and the above separation is required only for the formulas Acc_s.

We can now complete the construction of the indicators. We distinguish between two cases. In the first case, w is labeled q and we check whether changing the label to $\neg q$ creates an accepting run of $\mathcal{A}_{\neg\varphi}$. In the second case, w is labeled $\neg q$ and we check whether changing the label to q creates an accepting run of $\mathcal{A}_{\neg\varphi}$. Let $NC^+(K,\varphi,q)$ be the set of node q-covered states of K for φ and q that are labeled with q, that is, $NC^+(K,\varphi,q) = NC(K,\varphi,q) \cap \{w \in W : q \in L(w)\}$. Let Ψ_q^+ be the formula obtained from Ψ by replacing with q each occurrence of $\neg q$ that is not in the scope of a temporal operator. A state $w \in W$ satisfies Ψ_q^+ iff there exists a state $s \in S$ and an accepting run of $\mathcal{A}_{\neg\varphi}$ on an initialized path of K that visits the state s as it reads w with the value of q flipped. Thus, the set $NC^+(K,\varphi,q)$ is exactly the set $\{w \in W : w \models \Psi^+\}$. In the similar way we can define $NC^-(K,\varphi,q)$ as the set $NC(K,\varphi,q) \cap \{w \in W : q \notin L(w)\}$, and the formula Ψ_q^- that is obtained from Ψ by replacing with $\neg q$ each positive occurrence of q that is not in the scope of a temporal operator. The set $NC^-(K,\varphi,q)$ is exactly the set $\{w \in W : w \models \Psi^-\}$. Now, the indicator formula for φ is $Ind_q(\varphi) = \Psi_q^+ \vee \Psi_q^-$.

Theorem 3. *Given an LTL formula φ and an observable signal q, there exists a full μ-calculus formula $Ind_q(\varphi)$ of size exponential in φ such that for every Kripke structure K, the set of node-uncovered states of K with respect to φ and q is exactly the set of states of K that satisfy $Ind_q(\varphi)$.*

As discussed in [HKQ98,EL86,BC96], Ψ has alternation depth 2 (alternation is required in order to specify Büchi acceptance) and is alternation free if φ is a safety formula (then, $\mathcal{A}_{\neg\varphi}$ can be made an automaton with a looping acceptance condition [Sis94]). The size of the automaton $\mathcal{A}_{\neg\varphi}$ is exponential in the size of the formula φ [VW94], and the size of the formulas $Reach_s$ and Acc_s is linear in the size of $\mathcal{A}_{\neg\varphi}$. Hence, the size of indicator formula $Ind_q(\varphi)$ is exponential in the size of φ. We note that the exponential blow-up may not appear in practice [KV98,BRS99]. Since the semantics of μ-calculus with past modalities refers to structure, rather than trees (that is, the past is branching), model checking algorithms for μ-calculus with only future modalities can be modified to handle past without increasing complexity [KP95]. Model-checking complexity $K \models \psi$ for a μ-calculus formula ψ with alternation depth 2 is quadratic in $|K| \cdot |\psi|$ [EL86]. For alternation-free μ-calculus, the complexity is linear [CS91]. So, the complexity of finding the covered set using our reduction is $(|K| \cdot 2^{O(|\varphi|)})^2$ for general LTL properties and is $O(|K| \cdot 2^{O(|\varphi|)})$ for safety properties.

Remark 1. Two-way bisimulation extends bisimulation by examining both successors and predecessors of a state [HKQ98]. Two states w and w' are two-way bisimilar iff they satisfy the same full μ-calculus formulas. Since indicators are full μ-calculus formula, it follows that if w and w' are two-way bisimilar, they agree on the value of the indicator formula, thus w is node q-covered iff w' is node q-covered. In other words, the *distinguishing power* of node coverage is not greater than that of two-way bisimulation. In the full version we show that node coverage can distinguish between one-way bisimilar states. Thus, the use of full μ-calculus is essential for the construction of indicators.

References

[BB87] B. Banieqbal and H. Barringer. Temporal logic with fixed points. In *Temporal Logic in Specification*, *LNCS* 398, pp. 62–74, 1987.

[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.

[BC96] G. Bhat and R. Cleaveland. Efficient local model-checking for fragments of the modal μ-calculus. In *Proc. TACAS*, *LNCS* 1055, 1996.

[BGS00] R. Bloem, H.N. Gabow, and F. Somenzi. An algorithm for strongly connected component analysis in $n \log n$ symbolic steps. In *FMCAD*, *LNCS*,2000.

[BH99] J.P. Bergmann and M.A. Horowitz. Improving coverage analysis and test generation for large designs. In *Proc 11th CAD*, pp. 580–584, November 1999.

[BRS99] R. Bloem, K. Ravi, and F. Somenzi. Efficient decision procedures for model checking of linear time logic properties. In *Proc. 11th CAV*, *LNCS* 1633, pp. 222–235, 1999.

[Büc62] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Internat. Congr. Logic, Method. and Philos. Sci. 1960*, pp. 1–12, Stanford, 1962. Stanford University Press.

[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, 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.

[CKV01] H. Chockler, O. Kupferman, and M.Y. Vardi. Coverage metrics for temporal logic model checking. In *TACAS*, LNCS 2031, pp. 528 – 542, 2001.

[CS91] R. Cleaveland and B. Steffen. A linear-time model-checking algorithm for the alternation-free modal μ-calculus. In *Proc. 3rd CAD*, *LNCS* 575, pp. 48–58, 1991.

[DGK96] S. Devadas, A. Ghosh, and K. Keutzer. An observability-based code coverage metric for functional simulation. In *Proc. 8th CAD*, pp. 418–425, 1996.

[EL86] E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional μ-calculus. In *Proc. 1st LICS*, pp. 267–278, Cambridge, June 1986.

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

[FDK98] F. Fallah, S. Devadas, and K. Keutzer. OCCOM: efficient computation of observability-based code coverage metrics for functional simulation. In *Proc. of the 35th DAC*, pp. 152–157, June 1998.

[HH96] R.C. Ho and M.A. Horowitz. Validation coverage analysis for complex digital designs. In *Proc 8th CAD*, pp. 146–151, November 1996.

[HHK96] R.H. Hardin, Z. Har'el, and R.P. Kurshan. COSPAN. In *Proc. 8th CAV LNCS* 1102, pp. 423–427, 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.

[HKQ98] T.A. Henzinger, O. Kupferman, and S. Qadeer. From pre-historic to post-modern symbolic model checking. In *Proc 10th CAV*, *LNCS* 1427, 1998.

[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. of ICDD*, pp. 532–537, October 1995.

[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, June 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, 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. 33th DAC*, pp. 325–330, June 1996.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[KP95] O. Kupferman and A. Pnueli. Once and for all. In *Proc. 10th IEEE Symp. on Logic in Comp. Sci.*, pp. 25–35, San Diego, June 1995.

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

[KV98] O. Kupferman and M.Y. Vardi. Relating linear and branching model checking. In *IFIP Work. Conf. on Programming Concepts and Methods*, pp. 304 – 326, New York, June 1998. Chapman & Hall.

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

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

[Sis94] A.P. Sistla. Satefy, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6:495–511, 1994.

[VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.