Vacuity Detection in Temporal Model Checking

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

[1] Hebrew University, The institute of Computer Science
Jerusalem 91904, Israel
orna@cs.huji.ac.il
http://www.cs.huji.ac.il/~orna

[2] Rice University, Department of Computer Science
Houston, TX 77251-1892, U.S.A.
vardi@cs.rice.edu
http://www.cs.rice.edu/~vardi

Abstract. One of the advantages of temporal-logic 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. On the other hand, when the answer to the correctness query is positive, most model-checking tools provide no witness for the satisfaction of the specification. In the last few years there has been growing awareness to the importance of suspecting the system or the specification of containing an error also in the case model checking succeeds. The main justification of such suspects are possible errors in the modeling of the system or of the specification. Many such errors can be detected by further automatic reasoning about the system and the environment. In particular, Beer et al. described a method for the detection of *vacuous satisfaction* of temporal logic specifications and the generation of *interesting witnesses* for the satisfaction of specifications.
For example, verifying a system with respect to the specification $\varphi = AG(req \rightarrow AF grant)$ ("every request is eventually followed by a grant"), we say that φ is satisfied vacuously in systems in which requests are never sent. An interesting witness for the satisfaction of φ is then a computation that satisfies φ and contains a request. Beer et al. considered only specifications of a limited fragment of ACTL, and with a restricted interpretation of vacuity. In this paper we present a general method for detection of vacuity and generation of interesting witnesses for specifications in CTL*. Our definition of vacuity is stronger, in the sense that we check whether all the subformulas of the specification affect its truth value in the system. In addition, we study the advantages and disadvantages of alternative definitions of vacuity, study the problem of generating linear witnesses and counterexamples for branching temporal logic specifications, and analyze the complexity of the problem.

* Supported in part by the NSF grants CCR-9628400 and CCR-9700061, and by a grant from the Intel Corporation.

1 Introduction

Temporal logics, which are modal logics geared towards the description of the temporal ordering of events, have been adopted as a powerful tool for specifying and verifying concurrent systems [Pnu81]. One of the most significant developments in this area is the discovery of algorithmic methods for verifying temporal-logic properties of *finite-state* systems [CE81,CES86,LP85,QS81,VW86a]. This derives its significance both from the fact that many synchronization and communication protocols can be modeled as finite-state systems, as well as from the great ease of use of fully algorithmic methods. In temporal-logic *model checking*, we verify the correctness of a finite-state system with respect to a desired behavior by checking whether a labeled state-transition graph that models the system satisfies a temporal logic formula that specifies this behavior (for a survey, see [CGL93]).

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 provide no witness for the satisfaction of the specification in the system. 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 (or the specification) of containing an error also in the case model checking succeeds. The main justification of such suspects are possible errors in the modeling of the system or of the behavior.

Early work on "suspecting a positive answer" concerns the fact that temporal logic formulas can suffer from antecedent failure [BB94]. For example, verifying a system with respect to the specification $\varphi = AG(req \rightarrow AF grant)$ ("every request is eventually followed by a grant"), one should distinguish between *vacuous satisfaction* of φ, which is immediate in systems in which requests are never sent, and non-vacuous satisfaction, in which φ's precondition is sometimes satisfied. Evidently, vacuous satisfaction suggests some unexpected properties of the system, namely the absence of behaviors in which the precondition was expected to be satisfied.

Several years of experience in practical formal verification have convinced the verification group in IBM Haifa Research Laboratory that vacuity is a serious problem [BBER97]. To quote from [BBER97]: "Our experience has shown that typically 20% of specifications pass vacuously during the first formal-verification runs of a new hardware design, and that vacuous passes always point to a real problem in either the design or its specification or environment." Often, it is possible to detect vacuity easily, by checking the system with respect to hand-written formulas that ensure the satisfaction of the preconditions in the specification [BB94,PP95]. Formally, we say that a formula φ' is a *witness formula* for the specification φ if a system M satisfies φ non-vacuously iff M satisfies both φ and φ'.[1]. In the example above, it is not hard to see that a system satisfies φ non-vacuously iff it also satisfies $EFreq$. Sometimes, however, the generation of witness formulas is not trivial, especially when we are interested in other types of vacuity passes, more involved than antecedent failure.

These observations led Beer et al. to develop a method for automatic generation of witness formulas [BBER97]. Witness formulas are then used for two tasks. First, for the original task of detecting vacuity, and second, for the generation of an *interesting witness* for the satisfaction of the specification in the system. A witness for the satisfaction of a specification in a system is a sub-system, usually a computation, that satisfies the specification. A witness is interesting if it satisfies the specification non-vacuously. For example, a computation in which both *req* and *grant* hold is an interesting witness for the satisfaction of φ above. An interesting witness gives the user a confirmation that his specification models correctly the desired behavior, and enables the user to study some nontrivial executions of the system. In order to generate an interesting witness for the specification φ, one only has to find a (not necessarily interesting) witness for the conjunction $\varphi \wedge \varphi'$ of the specification and its witness formula. This can be done using the counterexample mechanism of model-checking tools. Indeed, a computation is a witness for $\varphi \wedge \varphi'$ iff it is a counterexample to $\neg(\varphi \wedge \varphi')$.

While [BBER97] nicely set the basis for a methodology for detecting vacuity in temporal-logic specifications, the particular method described in [BBER97] is quite limited. The type of vacuity passes handled is indeed richer than antecedent failure, yet it is still very restricted. Beer et al. consider the subset w-ACTL of the universal fragment ACTL of CTL. The logic w-ACTL consists of all ACTL formulas in which all the (Boolean or temporal) binary operators have at least one operand that is a propositional formula. Many natural specifications cannot be expressed in w-ACTL. Beyond specifications that contain existential requirements, like $AGEFgrant$ (and thus cannot be expressed in ACTL), this includes also universal specifications like $AG(AXgrant \vee AX\neg grant)$, which ensures that the granting event do not distinguish between “brothers” (different successors of the same state) in the system, as we expect in systems with delayed updates (that is, when the reaction of the system to events occurs only in the successors of the position in which the event has occurred). The syntax of w-ACTL enables [BBER97] to associate with each specification, a single subformula (called *important subformula*) and the vacuity of passes of the specifications is then checked only with respect to this subformula. For example, in formulas like $AG(req \rightarrow AFgrant)$, the algorithm in [BBER97] checks that *req* eventually holds in some path, yet it ignores the cases where $AFgrant$ always holds. While, as claimed in [BBER97], the latter case is less interesting, it can still help in many scenarios. The restricted syntax of w-ACTL and the restriction to important subformulas led to efficient algorithms for detection of vacuity and generation of interesting witnesses.

[1] The notion of a witness formula that we use here is dual to the one used in [BBER97]. There, a system M satisfies φ vacuously iff M satisfies both φ and its witness φ'.

In this paper we present a general method for detection of vacuity and generation of interesting witnesses for specifications in CTL* (and hence also LTL). Beyond the extension of the method in [BBER97] to highly expressive specification languages, our definition of vacuity is stronger, in the sense that we check whether all the subformulas of the specification affect its truth value in the system. In addition, we study the advantages and disadvantages of alternative definitions to vacuity, study the problem of generating linear witnesses and counterexamples for branching temporal logic specifications, and analyze the complexity of the problem.

From a computational point of view, we show that deciding whether a formula φ passes vacuously in a system M can be checked in time $O(C_M(|\varphi|) \cdot |\varphi|)$, where $C_M(|\varphi|)$ is the complexity of the model-checking problem for M and φ. Then, for φ in both LTL and CTL*, the problem of generating an interesting witness for φ in M (or deciding that no such witness exists) is PSPACE-complete. Both algorithms can be implemented symbolically on top of model checking tools like SMV and VIS. As explained in Section 4, part of the difficulty in generating an interesting witness comes from the fact that we insist on linear witnesses. When we consider worst-case complexity, the algorithm for generating interesting witnesses in [BBER97] is more efficient than ours (even when applied to w-ACTL formulas). Nevertheless, as explained in Section 4, for natural formulas, the performance of the algorithms coincides.

2 Temporal Logic

The logic *LTL* is a linear temporal logic. Formulas of LTL are built from a set AP of atomic proposition using the usual Boolean operators and the temporal operators X ("next time"), U ("until"), and $\tilde{U}$ ("duality of until"). We present here a positive normal form in which negation may be applied only to atomic propositions. Given a set AP, an LTL formula is defined as follows:

- **true**, **false**, p, or $\neg p$, for $p \in AP$.
- $\psi \vee \varphi$, $\psi \wedge \varphi$, $X\psi$, $\psi U \varphi$, or $\psi \tilde{U} \varphi$, where ψ and φ are LTL formulas.

We define the semantics of LTL with respect to a *computation* $\pi = \sigma_0, \sigma_1, \sigma_2, \ldots$, where for every $j \geq 0$, we have that σ_j is a subset of AP, denoting the set of atomic propositions that hold in the j'th position of π. We denote the suffix $\sigma_j, \sigma_{j+1}, \ldots$ of π by π^j. We use $\pi \models \psi$ to indicate that an LTL formula ψ holds in the path π. The relation $\models$ is inductively defined as follows:

- For all π, we have that $\pi \models$ **true** and $\pi \not\models$ **false**.
- For an atomic proposition $p \in AP$, we have $\pi \models p$ iff $p \in \sigma_0$ and $\pi \models \neg p$ iff $p \notin \sigma_0$.
- $\pi \models \psi \vee \varphi$ iff $\pi \models \psi$ or $\pi \models \varphi$.
- $\pi \models \psi \wedge \varphi$ iff $\pi \models \psi$ and $\pi \models \varphi$.
- $\pi \models X\psi$ iff $\pi^1 \models \psi$.
- $\pi \models \psi U \varphi$ iff there exists $k \geq 0$ such that $\pi^k \models \varphi$ and $\pi^i \models \psi$ for all $0 \leq i < k$.

– $\pi \models \psi \tilde{U} \varphi$ iff for every $k \geq 0$ for which $\pi^k \not\models \varphi$, there exists $0 \leq i < k$ such that $\pi^i \models \psi$.

We use the following abbreviations in writing formulas:

– $F\varphi = \mathbf{true} U \varphi$ ("eventually").
– $G\varphi = \mathbf{false} \tilde{U} \varphi$ ("always").

The logic CTL* is a branching temporal logic. A path quantifier, E ("for some path") or A ("for all paths"), can prefix an assertion composed of an arbitrary combination of linear time operators. There are two types of formulas in CTL*: *state formulas*, whose satisfaction is related to a specific state, and *path formulas*, whose satisfaction is related to a specific path. Formally, let AP be a set of atomic proposition names. A CTL* state formula (again, in a positive normal form) is either:

– **true**, **false**, p or $\neg p$, for $p \in AP$.
– $\psi \vee \varphi$ or $\psi \wedge \varphi$ where ψ and φ are CTL* state formulas.
– $E\psi$ or $A\psi$, where ψ is a CTL* path formula.

A CTL* path formula is either:

– A CTL* state formula.
– $\psi \vee \varphi$, $\psi \wedge \varphi$, $X\psi$, $\psi U \varphi$, or $\psi \tilde{U} \varphi$, where ψ and φ are CTL* path formulas.

The logic CTL* consists of the set of state formulas generated by the above rules. The logic *CTL* is a restricted subset of CTL*. In CTL, the temporal operators X, U, and $\tilde{U}$ must be immediately preceded by a path quantifier. Formally, it is the subset of CTL* obtained by restricting the path formulas to be $X\psi$, $\psi U \varphi$, or $\psi \tilde{U} \varphi$, where ψ and φ are CTL state formulas.

We denote the length of a formula φ by $|\varphi|$. When we consider subformulas of an LTL formula ψ, we refer to the syntactic subformulas of ψ, thus to path formulas. On the other hand, when we consider subformulas of a CTL* formula ψ, we refer to the state subformulas of ψ. Then, the *closure* $cl(\psi)$ of an LTL or a CTL* formula ψ is the set of all subformulas of ψ (including ψ but excluding **true** and **false**). For example, $cl(pU(Xq)) = \{pU(Xq), p, Xq, q\}$, and $cl(E(pU(AXq))) = \{E(pU(AXq)), p, AXq, q\}$. It is easy to see that the size of $cl(\psi)$ is linear in the size of ψ. We use $\varphi[\psi \leftarrow \xi]$ to denote the formula obtained from φ by replacing its subformula ψ by the formula ξ.

We define the semantics of CTL* (and its sublanguage CTL) with respect to *systems*. A system $M = \langle AP, W, R, w_0, L \rangle$ 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 $w_0 \in W$, and a labeling function $L : W \rightarrow 2^{AP}$. A *computation* of a system is a sequence of states, $\pi = w_0, w_1, \ldots$ such that for every $i \geq 0$, we have that $\langle w_i, w_{i+1} \rangle \in R$. We define the *size* $|M|$, of a system M as above as $|W| + |R|$. We use $w \models \varphi$ to indicate that a state formula φ holds at state w (assuming an agreed fair module M). The relation $\models$ is inductively defined as follows (the relation $\pi \models \psi$ for a path formula ψ is the same as for ψ in LTL).

- For all w, we have that $w \models \mathbf{true}$ and $w \not\models \mathbf{false}$.
- For an atomic proposition $p \in AP$, we have $w \models p$ iff $p \in L(w)$ and $w \models \neg p$ iff $p \notin L(w)$.
- $w \models \psi \vee \varphi$ iff $w \models \psi$ or $w \models \varphi$.
- $w \models \psi \wedge \varphi$ iff $w \models \psi$ and $w \models \varphi$.
- $w \models E\psi$ iff there exists a computation $\pi = w_0, w_1, \ldots$ such that $w_0 = w$ and $\pi \models \psi$.
- $w \models A\psi$ iff for all computations $\pi = w_0, w_1, \ldots$ such that $w_0 = w$, we have $\pi \models \psi$.
- $\pi \models \varphi$ for a computation $\pi = w_0, w_1, \ldots$ and a state formula φ iff $w_0 \models \varphi$.

A system M satisfies a formula φ iff φ holds in the initial state of M. The problem of determining whether M satisfies φ is the *model-checking* problem. For a particular temporal logic, a system M, and an integer n, we use $C_M(n)$ to denote the complexity of checking whether a formula of size n in the logic is satisfied in M.

3 Satisfying a Formula Vacuously

Intuitively, a system M satisfies a formula φ vacuously if M satisfies φ yet it does so in a non-interesting way, which is likely to point on some trouble with either M or φ. For example, a system in which *req* never occurs satisfies $AG(\mathit{req} \rightarrow AF\mathit{grant})$ vacuously. In order to formalize this intuition, it is suggested in [BBER97] to use the following definition of when a subformula of φ affects its truth value in M.

Definition 1. [BBER97] *The subformula ψ of φ* does not affect the truth value of φ in M *(ψ does not affect φ in M, for short) if for every formula ξ, the system M satisfies $\varphi[\psi \leftarrow \xi]$ iff M satisfies φ.*

Note that one can talk about a subformula ψ affecting φ in M or about an *occurrence* of ψ affecting φ in M. As we shall see in Section 3.2, dealing with occurrences is much easier than dealing with subformulas. In the sequel, we assume for simplicity that all the subformulas of φ have single occurrences. (Equivalently, we could change the definition to talk about when a particular occurrence of ψ does not affect φ. All the results in the paper hold also for this alternative.)

As stated, Definition 1 is not effective, since it requires evaluation $\varphi[\psi \leftarrow \xi]$ for all formulas ξ. To deal with this difficulty, [BBER97] considers only a small class, called w-ACTL, of branching temporal logic formulas. In Theorem 1 bellow, we show that instead of checking the replacement of ψ by all formulas ξ, one can check only the replacement of ψ by the formulas **true** and **false**. For that, we first partition the subformulas of φ according to their *polarity* as follows. Every subformula ψ of φ may be either *positive in* φ, in the case it is in the scope of an even number of negations, or *negative in* φ, in the case it is in the scope of an odd number of negations (note that an antecedent of an implication is considered to be under negation)[2]. For a formula φ and a subformula ψ of φ, let $\varphi[\psi \leftarrow \bot]$ denote the formula obtained from φ by replacing ψ by **false**, in case ψ is positive in φ, and replacing ψ by **true**, in case ψ is negative in φ. Dually, $\varphi[\psi \leftarrow \top]$ replaces a positive ψ by **true** and replaces a negative ψ by **false**. We say that a Boolean or temporal operator $f(\xi_1)$ or $g(\xi_1, \xi_2)$ of temporal logic is *positively monotonic* if for every ξ_1 and ξ_2 we have that $f(\xi_1) \rightarrow f(\mathbf{true})$ and $g(\xi_1, \xi_2) \rightarrow g(\mathbf{true}, \xi_2) \wedge g(\xi_1, \mathbf{true})$. Dually, the operator is *negatively monotonic* if for every ξ_1 and ξ_2 we have that $f(\xi_1) \rightarrow f(\mathbf{false})$ and $g(\xi_1, \xi_2) \rightarrow g(\mathbf{false}, \xi_2) \wedge g(\xi_1, \mathbf{false})$. Since all the operators in CTL* are positively monotonic, except for $\neg$, which is negatively monotonic, the following properties of positive and negative subformulas of φ can be easily proved by an induction on the structure of φ.

Lemma 1. *For a subformula ψ of φ and for every system M, if $M \models \varphi[\psi \leftarrow \bot]$, then for every formula ξ, we have $M \models \varphi[\psi \leftarrow \xi]$. Also, if $M \not\models \varphi[\psi \leftarrow \top]$, then for every formula ξ, we have $M \not\models \varphi[\psi \leftarrow \xi]$.*

Lemma 1 implies that **true** and **false** are two "extreme" replacements for ψ in φ; thus instead of checking agreement on the satisfaction of φ with all replacements ξ, one may only consider these two extreme replacements. Theorem 1 below formalizes this intuition.

Theorem 1. *For every formula φ, a subformula ψ of φ, and a system M, the following are equivalent:*

(1) *ψ does not affect φ in M.*
(2) *M satisfies $\varphi[\psi \leftarrow \mathbf{true}]$ iff M satisfies $\varphi[\psi \leftarrow \mathbf{false}]$.*

Proof. Assume first that ψ does not affect φ in M. Then, in particular, $M \models \varphi[\psi \leftarrow \mathbf{true}]$ iff $M \models \varphi$, and $M \models \varphi[\psi \leftarrow \mathbf{false}]$ iff $M \models \varphi$. It follows that $M \models \varphi[\psi \leftarrow \mathbf{true}]$ iff $M \models \varphi[\psi \leftarrow \mathbf{false}]$. For the other direction, assume that M satisfies $\varphi[\psi \leftarrow \mathbf{true}]$ iff M satisfies $\varphi[\psi \leftarrow \mathbf{false}]$. Consider first the case ψ is positive in φ. We distinguish between two cases. First, if M satisfies $\varphi[\psi \leftarrow \mathbf{false}]$, then, as ψ is positive in φ, it follows from Lemma 1 that for every formula ξ, we have $M \models \varphi[\psi \leftarrow \xi]$, and in particular $M \models \varphi$. Thus, ψ does not affect φ in M. Now, if M does not satisfy $\varphi[\psi \leftarrow \mathbf{false}]$, we have that M does not satisfy $\varphi[\psi \leftarrow \mathbf{true}]$ either. Then, as ψ is positive in φ, it follows from Lemma 1 that for every formula ξ, we have $M \not\models \varphi[\psi \leftarrow \xi]$, and in particular $M \not\models \varphi$. Thus, ψ does not affect φ in M. The case ψ is negative in φ is dual.

We can now define formally the notion of vacuous satisfaction:

[2] If we assume that the formula φ is given in a positive normal form, all the subformulas of φ, except maybe some propositions, are positive in φ. Since an assertion $\neg p$, for a proposition p, does not affect φ in M iff p does not affect φ in M, we can regard such assertions as atomic propositions, thus assume that all subformulas are positive in φ. In this section, however, we consider also formulas that are not in positive normal form, thus we refer to both positive and negative subformulas.

Definition 2. [BBER97] *A system M satisfies a formula φ* vacuously *iff $M \models \varphi$ and there is some subformula ψ of φ such that ψ does not affect φ in M.*

Theorem 1 reduces the problem of vacuity detection to the problem of model checking of M with respect to the formulas $\varphi[\psi \leftarrow \textbf{true}]$ and $\varphi[\psi \leftarrow \textbf{false}]$ for all subformulas ψ of φ. In fact, by Lemma 1, whenever M satisfies φ, it also satisfies $\varphi[\psi \leftarrow \top]$ for all subformulas ψ of φ. Accordingly, M satisfies φ vacuously if $M \models \varphi$ and there is some subformula ψ of φ such that M satisfies $\varphi[\psi \leftarrow \bot]$. Since the number of subformulas of φ is bounded by $|\varphi|$, it follows that vacuity detection involves model checking of M with respect to at most $|\varphi|$ formulas, all smaller than φ. Hence the following theorem.

Theorem 2. *The problem of checking whether a system M satisfies a formula φ vacuously can be solved in time $O(C_M(|\varphi|) \cdot |\varphi|)$.*

3.1 Alternative Definitions

In Definition 1, we require that for every ξ, the replacement of ψ by ξ does not affect the value of φ in M. One can also think about an alternative definition in which ψ does not affect φ in M if M satisfies φ iff for every formula ξ, we have that M satisfies $\varphi[\psi \leftarrow \xi]$. This alternative definition seems equivalent to Definition 1. Nevertheless, as we show below, the definitions are not equivalent and only Definition 1 agrees with our intuitive understanding of affecting a truth value. To see this, consider a system M that has a single state with a self loop, in which p does not hold. Let $\varphi = \psi = p$. By the definition above, the formula ψ does not affect φ in M. Indeed, both sides of the iff condition in that definition do not hold: M does not satisfy p, and there is a formula ξ ($\xi = \textbf{false}$) such that M does not satisfy $p[p \leftarrow \xi]$. Nevertheless, our intuition is that p does affect the truth value of p in M. This agrees with Definition 1. Indeed, there is a formula ξ ($\xi = \textbf{true}$) such that $M \models p[p \leftarrow \xi]$ yet $M \not\models p$.

Note that the definition of when a system satisfies a formula vacuously is insensitive to the difference between the two definitions. Indeed, when $M \models \varphi$, both definitions require M to satisfy $\varphi[\psi \leftarrow \xi]$ for all ξ.

While Definition 1 cares about the satisfaction of φ in the initial state of M, and thus corresponds to *local* model checking, *global* model-checking algorithms calculate the set of all states that satisfy φ. Accordingly, if we use $M(\psi)$ to denote the set of states in M that satisfy ψ, one could also consider the alternative definition where ψ does not affect φ in M if for every formula ξ, $M(\varphi[\psi \leftarrow \xi]) = M(\varphi)$. The problem of this definition is that the replacement of ψ in ξ may change the set of states that satisfy φ in some non-interesting way, say with respect to non-reachable states. For example, consider a system M with one reachable state s with a self-loop, labeled $\{q\}$, and one non-reachable state s' with a self-loop, labeled $\emptyset$. The state s satisfies both $\varphi = AG(p \rightarrow q)$ and AGq. Thus, p does not affect φ in M according to Definition 1. On the other hand, while $M(\varphi) = \{s, s'\}$, we have that $M(AGq) = \{s\}$. Thus, p affects φ in M according to the global definition above. Since s' is not reachable, the fact that s' satisfies φ vacuously is not of real interest, thus we believe that the fact Definition 1 ignores such vacuous satisfaction meets our goals. It is easy, however, to extend all the results in the paper to handle also global vacuity. In particular, the corresponding variant of Lemma 1, namely $M(\varphi[\psi \leftarrow \mathbf{false}]) \subseteq M(\varphi) \subseteq M(\varphi[\psi \leftarrow \mathbf{true}])$ is valid for all ψ, thus global vacuous satisfaction of φ in M (and there are in fact several possible definitions here as well), can be detected in time $O(C_M(|\varphi|)\cdot|\varphi|)$.

3.2 Occurrences vs. Subformulas

Recall that one can talk about a subformula ψ affecting φ in M or about an occurrence of ψ affecting φ in M. As we now show, the latter choice is computationally easier. Caring about whether a particular occurrence of ψ affects the value of φ in M, we assumed, for technical convenience, that all subformulas occur only once. Given ψ, φ, and M, Theorem 1 then suggests a simple solution for the problem of deciding whether ψ affects φ in M. Formally, the problem can be solved in time $O(C_M(|\varphi|))$. In particular, when φ is in CTL, the problem can be solved in time linear in M and φ [CES86]. When ψ has several occurrences, Theorem 1 is no longer valid. This is because different occurrences of ψ may have different polarities. We now show that in this case the problem of deciding whether ψ affects φ in M is most likely harder.

We say that ψ *affects* φ *in* M iff it is not the case that ψ does not affect φ in M. Thus, ψ affects φ in M iff there is a formula ξ such that either $M \models \varphi[\psi \leftarrow \xi]$ and $M \not\models \varphi$, or $M \not\models \varphi[\psi \leftarrow \xi]$ and $M \models \varphi$.

Theorem 3. *For φ in CTL, a subformula ψ of φ with multiple occurrences, and a system M, the problem of deciding whether ψ does not affect φ in M is co-NP-complete.*

Proof. We show that the complementary problem, of deciding whether ψ affects φ in M is NP-complete. To prove membership in NP, we first claim that if there is a formula ξ such that M does not agree on the satisfaction of φ and of $\varphi[\psi \leftarrow \xi]$, then there also exists such a formula ξ' of length $|M|$. Membership in NP then follows from fact that we can guess the formula ξ' above and check whether $M \models \varphi$ iff $M \models \varphi[\psi \leftarrow \xi']$. To prove hardness in NP, we do a reduction from SAT. Given $n \geq 0$, we define the Kripke structure $K_n = \langle \{q, r\}, \{0, \ldots, n+1\}, R, 0, L\rangle$, where $R = \{\langle 0, 1\rangle, \langle 1, 2\rangle, \ldots, \langle n, n+1\rangle, \langle n+1, n+1\rangle\}$, and L maps all states $i \in \{0, \ldots, n-1\} \cup \{n+1\}$ to $\emptyset$ and maps the state n to $\{r\}$. Thus, K_n is a chain of $n+2$ states none of which satisfies q, and only the state before the last one satisfies r. Giving a propositional formula θ over $p_0, \ldots, p_{n-1}$, let ψ be the CTL formula obtained from θ by replacing each occurrence of p_i by $(EX)^i q$. Then, let $\varphi = \psi \wedge (EX)^n q$. For example, if $\theta = (p_0 \vee p_1) \wedge (\neg p_1 \vee p_2)$, then $\varphi = (q \vee EXq) \wedge (\neg EXq \vee EXEXq) \wedge EXEXEXq$. Since no state of K_n satisfies q, the structure K_n does not satisfy φ. On the other hand, since, no matter what θ is, the only requirement that φ induces on the state n is to satisfy q, it is easy to see that there is a formula ξ such that $K_n \models \varphi[q \leftarrow \xi]$ iff θ is satisfiable: the formula ξ is induced by a satisfying assignment for θ and it holds at state i iff $i = n$ or p_i is assigned **true** in the satisfying assignment. Using the fact that n is the only state in which r holds, we can indeed "translate" each assignment to a corresponding ξ. In the example above, an assignment that assigns **true** to p_0 and p_2 induces the formula $\xi = r \vee EX(r \vee EXEXr)$. It follows that q affects φ in K_n iff θ is satisfiable.

4 Interesting Witnesses

When a good model-checking tool decides that a system M does not satisfy a required property φ, it returns a counterexample to the satisfaction of φ, namely, some erroneous execution of to detect problems in M or in φ. Most model-checking tools, however, provide no witness for the satisfaction of φ in M. Such a witness may be very helpful too, in particular when it describes an execution in which the formula is satisfied in an interesting way. In this section we discuss the generation of *interesting witnesses* to the satisfaction of LTL and CTL* formulas.

Definition 3. [BBER97] *Consider a system M and a formula φ such that $M \models \varphi$. A path π of M is an* interesting witness *for φ in M if π satisfies φ non-vacuously.*

The generation of an interesting witness involves two difficulties. The first is present in the case φ is a branching temporal logic formula and it involves the generation of a linear (rather than a branching) witness. This difficulty is analogous to the difficulty of constructing a linear counterexample in a system that violates a branching temporal logic formula. The second difficulty is present also when φ is a linear temporal logic formula and it involves the fact that all the subformulas of φ should affect the satisfaction of φ in the witness. Note that even when M satisfies φ non-vacuously, it may be that some paths of M satisfy φ vacuously. For example, a structure that satisfies $AG(req \rightarrow AF grant)$ non-vacuously may contain a path in which req never holds. Moreover, it may be that M satisfies φ non-vacuously, all the paths of M satisfy φ as well, yet no path of M is an interesting witness for φ. As an example, consider the formula above and a structure with two paths, one path that never satisfies req and a second path that always satisfies $grant$. To see another weakness of the definition of an interesting witness, consider the LTL formula $\varphi = G(req_1 \rightarrow F grant_1) \wedge G(req_2 \rightarrow grant_2)$. While a system M may satisfy φ non-vacuously and contain interesting witnesses for both $G(req_1 \rightarrow F grant_1)$ and $G(req_2 \rightarrow grant_2)$, the system M may not contain an interesting witness for φ, as both req_1 and req_2 are required to hold in such a witness. This difficulty arises since φ is a conjunction of two specifications, and it can be avoided by separating conjunctions to their conjuncts.

We start with the first difficulty. We say that a branching temporal logic formula φ is *linearly witnessable* if for every system M, if $M \models \varphi$ then M has a path π such that $\pi \models \varphi$. The following lemma follows immediately from the definition.

Lemma 2. *All formulas of the universal fragment $ACTL^\star$ of* CTL* *are linearly witnessable, and so are all* CTL* *formulas with a single existential path quantifier.*

It follows from Lemma 2 that if a formula has no existential path quantifiers, or has a single path quantifier, then it is linearly witnessable. This syntactic condition is a sufficient but not a necessary one. For example, the CTL formula $EXEFp$ is linearly witnessable, and so is the less natural formula $EXp \vee EX\neg p$. The latter example suggests that testing a formula for being linearly witnessable is at least as hard as the validity problem.

Theorem 4. *Given a CTL formula φ, deciding whether φ is linearly witnessable is in 2EXPTIME and is EXPTIME-hard.*

Proof. We start with the upper bound. We first claim that if there is a system M such that $M \models \varphi$ yet M has no path π such that $\pi \models \varphi$, then there also exists such an M with branching degree bounded by $|\varphi|$. The proof of the claim is similar to the proof of the bounded-degree property for CTL [Eme90]. Give φ, let $\mathcal{A}_\varphi$ be a nondeterministic Büchi tree automaton that accepts exactly all trees of branching degree at most $|\varphi|$ that satisfy φ [VW86b], and let $\mathcal{A}'_\varphi$ be nondeterministic Büchi word automaton that accepts exactly all words (i.e., trees of branching degree 1) that satisfy φ [VW94]. We expand $\mathcal{A}'_\varphi$ to a Büchi tree automaton $\mathcal{A}''_\varphi$ that accepts a tree iff the tree has a path accepted by $\mathcal{A}'_\varphi$ (in each state, $\mathcal{A}''_\varphi$ guesses a direction in which it follows $\mathcal{A}'_\varphi$). We prove that φ is linearly witnessable iff $\mathcal{L}(\mathcal{A}_\varphi) \subseteq \mathcal{L}(\mathcal{A}''_\varphi)$. Since the containment problem $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$ for Büchi tree automata can be solved in time that is polynomial in the size of $\mathcal{A}$ and exponential in the size of A' [EJ88,MS95], the 2EXPTIME upper bound follows. Assume first that φ is linearly witnessable, and let T be a tree in $\mathcal{L}(\mathcal{A}_\varphi)$. Then, T contains a path π such that π satisfies φ, implying that π is accepted by $\mathcal{A}'_\varphi$. Then, however, by the definition of $\mathcal{A}''_\varphi$, we have that T is also in $\mathcal{L}(\mathcal{A}''_\varphi)$. Assume now that φ is not linearly witnessable. then, by the bounded-degree property above, there is a system, and therefore also a tree T of branching degree at most $|\varphi|$ such that $T \models \varphi$ yet no path of T satisfies φ. Hence, while the tree T is in $\mathcal{L}(\mathcal{A}_\varphi)$, it is not accepted by $\mathcal{A}''_\varphi$, implying that $\mathcal{L}(\mathcal{A}_\varphi)$ is not contained in $\mathcal{L}(\mathcal{A}''_\varphi)$.

For an EXPTIME lower bound, we do a reduction from the satisfiability problem for CTL. Consider a formula φ over a set of atomic propositions that does not contain p and q. We prove that φ is not satisfiable iff $\psi = \varphi \wedge EXp \wedge EXq$ is linearly witnessable. Clearly, if φ is not satisfiable, then so is ψ, which is therefore linearly witnessable. For the second direction, assume that φ is satisfiable, and consider a system M that satisfies φ. We define a system M' as follows. If the initial state of M has two or more successors, we label one of its successors by p and we label a different successor by q. If the initial state of M has only one successor, we duplicate it, and then proceed as above. It is easy to see that while M' satisfies ψ, no path of M' satisfies ψ, thus ψ is not linearly witnessable.

The gap between the upper and lower bounds in Theorem 4 is similar to gaps in related problems such as the complexity of determining whether a CTL*

formula has an equivalent LTL formula (a 2EXPTIME upper bound and an EXPTIME lower bound [KV98b]), the complexity of determining whether an LTL formula has an equivalent alternation-free μ-calculus formula (an EXPSPACE upper bound and a PSPACE lower bound [KV98a]), and several more problems. Essentially, in all the problems above we check the equivalence between a set of trees that satisfy $A\varphi$, for an LTL formula φ, and a set of trees that is defined directly by some branching-time formalism. The best known translation of $A\varphi$ to a tree automaton involves a doubly-exponential blow up. This is because the nondeterministic automaton for φ, whose size is exponential in $|\varphi|$, needs to be determinized before its expansion into a tree automaton, or, alternatively (as in the proof above), the nondeterministic tree automaton for $E\neg\varphi$ needs to be complemented. The doubly-exponential size of the tree automaton then leads to EXPSPACE and 2EXPTIME upper bounds. On the other hand, typical EXPSPACE and 2EXPTIME lower-bound proofs for temporal logic [VS85,KV95] require the use of temporal logic formulas that do not fit into the restricted syntax that is present in the problems above (e.g., formulas of the form $A\varphi^d \rightarrow \varphi$ for some CTL* formula φ).

The generation of interesting witnesses in [BBER97] goes through a search for a counterexample for a "witnessing formula". This generation succeeds only for witnesses formulas for which a linear counterexample exists. It is claimed in [BBER97] that almost all interesting CTL formulas indeed have linear counterexamples. We say that a branching temporal logic formula is *linearly counterable* iff for every system M, if $M \not\models \varphi$ then M has a path π such that $\pi \not\models \varphi$. The following theorem, which characterizes linearly counterable formulas, follows immediately from the definitions of linearly witnessable and linearly counterable.

Theorem 5. *For a branching temporal logic formula φ, we have that φ is linearly counterable iff $\neg\varphi$ is linearly witnessable.*

Note that a formula φ may be both linearly witnessable and linearly counterable (in which case $\neg\varphi$ is both linearly witnessable and linearly counterable as well). The formulas AGp and EFq, for example, fall in this category. In fact, by Lemma 2, all formulas with at most one universal and one existential path quantifiers are both linearly witnessable and linearly counterable.

In the context of model checking, however, a particular system M is given, and while φ may not be linearly witnessable, it may still have a linear witness in M. We say that φ is *linearly witnessable in M* if $M \models \varphi$ implies that M has a path π such that $\pi \models \varphi$. In order to check whether φ is linearly witnessable in M, we first need the following notation. For a branching temporal logic formula φ in a positive normal form, let φ^d be the LTL formula obtained from φ by eliminating its path quantifiers. For example, $(AGEFp)^d = GFp$. By [CD88], φ has an equivalent LTL formula iff φ is equivalent to $A\varphi^d$.

Theorem 6. *For a branching temporal logic formula φ and a system M, we have that $M \not\models A\varphi^d$ iff M has a path π such that $\pi \not\models \varphi$.*

Proof. Assume first that $M \not\models A\varphi^d$. Then, M has a path π such that $\pi \not\models \varphi^d$. Since the branching degree of π is 1, the path π does not satisfy φ either. For the other direction, assume that M has a path π such that $\pi \not\models \varphi$. Since the branching degree of π is 1, the path π does not satisfy φ^d either. Hence, $M \not\models A\varphi^d$.

Theorem 7. *For a* CTL* *formula* φ *and a system* M*, deciding whether* φ *is linearly witnessable in* M *is PSPACE-complete.*

Proof. Replacing the formula φ in Theorem 6 by the formula $\neg\varphi$, we get that $M \not\models A(\neg\varphi)^d$ iff M has a path π such that $\pi \models \varphi$. It follows that φ is linearly witnessable in M iff $M \models \varphi \to E\varphi^d$. Membership in PSPACE then follows from CTL* model-checking complexity [EL87]. Given a system M and an ACTL formula φ, it is shown in [KV98b] that the model-checking problem $M \models A\varphi^d \to \varphi$ is PSPACE-complete. Equivalently, given a system M and an ECTL formula φ, the model-checking problem $M \models \varphi \to E\varphi^d$ is PSPACE-complete. Since φ is linearly witnessable in M iff $M \models \varphi \to E\varphi^d$, hardness in PSPACE follows (in fact, already for φ in ECTL).

In practice, we are interested in generating a linear witness (and thus in the question of linear witnessability) only in systems M that satisfy φ. Note that the proof of Theorem 7 shows that deciding whether φ is linearly witnessable in M is PSPACE-complete already for M as above.

We now study the second difficulty: finding an interesting linear witness. Recall that the generation of interesting witnesses in [BBER97] goes through a search for a counterexample for a witnessing formula. The definition of the witnessing formula in [BBER97] crucially depends on the restricted syntax of w-ACTL. Below we generate a witnessing formula for general branching or linear temporal logic formulas. Given a formula φ (in either LTL or CTL*), we define

$$\mathit{witness}(\varphi) = \varphi \wedge \bigwedge_{\psi \in cl(\varphi)} \neg\varphi[\psi \leftarrow \bot].$$

Note that the length of $\mathit{witness}(\varphi)$ is quadratic in $|\varphi|$. Intuitively, a path π satisfies $\mathit{witness}(\varphi)$ if π satisfies φ and in addition, π does not satisfy the formula $\varphi[\psi \leftarrow \bot]$ for all the subformulas ψ of φ. Thus, all subformulas of φ affect its value in π.

Theorem 8. *For a formula* φ *and a system* M*, a counter example for* $\neg\mathit{witness}(\varphi)$ *in* M *is an interesting witness for* φ *in* M*.*

Proof. Let π be a counterexample for $\neg\mathit{witness}(\varphi)$ in M. Then, π satisfies $\mathit{witness}(\varphi)$. As such, π satisfies φ, yet for all subformulas ψ of φ, the path π does not satisfy the formula $\varphi[\psi \leftarrow \bot]$. It follows that all subformulas ψ of φ affect φ in π, hence π satisfies φ non-vacuously.

Theorem 9. *For an LTL or a* CTL* *formula* φ *and a system* M*, an interesting witness for* φ *in* M *can be generated in polynomial space. Deciding whether such a witness exists is PSPACE-complete.*

Proof. By Theorem 8, one can generate an interesting witness π for φ in M by generating a counterexample for $\neg witness(\varphi)$ in M. When φ is an LTL formula, so is $\neg witness(\varphi)$, and the generation of π can be done by generating a path in the intersection of M and a Büchi word automaton for $\neg witness(\varphi)$. Membership in PSPACE then follows from the fact that the automaton for an LTL formula ξ is of size exponential in ξ [VW94], and the generation of a path in the intersection of the automaton with M can be done on-the-fly and nondeterministically in space that is logarithmic in the sizes of M and the automaton. When φ is a CTL* formula, we know, as discussed in the proof of Theorem 6, that a counterexample in M for $A(\neg witness(\varphi))^d$ is also a counterexample for $\neg witness(\varphi)$ in M. Thus, the generation can proceed as in the case of LTL formulas, replacing $\neg witness(\varphi)$ by $(\neg witness(\varphi))^d$. In both cases, the lower bound follows by a reduction from LTL model checking [SC85].

The lower bound in Theorem 9 implies that the generation of interesting witnesses may require, at the worst case, space that is polynomial in the length of the specification, which in practice means that it may require time that is exponential in the length of the specification. On the other hand, the method in [BBER97] requires only linear time. The comparison of the two approaches from a complexity-theoretic point of view is actually a special case of the traditional comparison between LTL and CTL model-checking complexity. Indeed, while the generation in [BBER97] goes through the counterexample mechanism for CTL formulas [CGMZ95], ours go through the counterexample mechanism for LTL formulas, which uses an automata-theoretic reduction (exponential in the worst case) to CTL counterexample generation [VW86a]. Our experience with this comparison teaches us that, in practice, standard LTL model checkers perform nicely on most formulas. In fact, for formulas that can be expressed in both LTL and CTL, LTL model-checking tools often proceeds essentially as CTL model-checking tools. Intuitively, both model checkers proceed according to the semantics of the formula and are insensitive to the syntax in which the formula is given (for a detailed analysis and comparison of the two verification paradigms see [KV98b]). Experimental results of LTL and CTL model checking of common specifications support our observation and show no advantage to the branching paradigm [Cla97,BRS99]. In addition, once LTL model checking (or generation of counterexamples) is reduced to detection of a fair computation in the product of the system and the automaton for the negation of the specification, such a detection can be performed using CTL model-checking tools, thus our method can be implemented symbolically on top of model checkers such as SMV or VIS.

5 Discussion

We presented a general method for detection of vacuity and generation of interesting witnesses for specifications in CTL*. The results in the paper can be easily extended to handle systems with *fairness* conditions. A typical fairness condition for a system $M = \langle AP, W, R, w_0, L\rangle$ is a tuple $\langle F_1, \ldots, F_k\rangle$ of subsets of W. Such a condition means that we restrict attention to computations that visit each F_i infinitely often [Fra86]. It is known that model-checking algorithms extend to systems with such fairness conditions [CES86,VW86a]. Since our method is based on the model-checking algorithm, it can therefore be easily extended to handle fairness. Also, being based on the model-checking algorithm, our method is fully automatic, and all the common heuristics for coping with the state-explosion problem are applicable to it. As with model checking, the discouraging complexity bounds for the problems discussed in the paper do rarely appear in practice. An interesting open question is how to find interesting witnesses of minimal length (cf. [CGMZ95]).

Vacuity check is only one approach to challenge the correctness of the verification process. We mention here two recent related approaches. An approach that is closely related to vacuity is taken in the process of constraint validation in the verification tool FormalCheck [Kur98]. In order to validate a set of constraints about the environment, the constraints are converted into specifications and are checked with respect to a model of the environment. Sometimes, however, there is no model of the environment, and instead, FormalCheck proceeds with some heuristic sanity checks for constraint validation. This includes a search for enabling conditions that are never enabled, and a replacement of all or some of the constraints by **false**. A different approach is described in [KGG99], where the authors extend the notion of *coverage* from testing to model checking. Given a specification and its implementation, bisimulation is used in order to check whether the specification covers the entire functionality performed by the implementation. If the answer is negative, the specification is suspected for not being sufficiently restrictive.

Acknowledgments

We thank Shoham Ben-David and Yaakov Crammer for helpful comments on a previous version of this work.

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. 83

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. 83, 84, 85, 87, 89, 91, 93, 94, 95

BRS99. R. Bloem, K. Ravi, and F. Somenzi. Efficient decision prcedures for model checking of linear time logic properties. In *Proc. 11th CAV*, LNCS, 1999. 95

CD88. E. M. Clarke and I. A. Draghicescu. Expressibility results for linear-time and branching-time logics. In *Proc. Workshop on Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, *LNCS* 354, pp. 428–437, 1988. 93

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. 83

CES86. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986. 83, 90, 96

CGL93. E. M. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. In *Decade of Concurrency – Reflections and Perspectives (Proceedings of REX School)*, *LNCS* 803, pp. 124–175, 1993. 83

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. 83, 95, 96

Cla97. E. Clarke. Private communication, 1997. 95

EJ88. E. A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *Proc. 29th FOCS*, pp. 368–377, White Plains, 1988. 92

EL87. E. A. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8:275–306, 1987. 94

Eme90. E. A. Emerson. Temporal and modal logic. *Handbook of Theoretical Computer Science*, pp. 997–1072, 1990. 92

Fra86. N. Francez. *Fairness.* Texts and Monographs in Computer Science. Springer-Verlag, 1986. 96

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, 1999. 96

Kur98. R. P. Kurshan. *FormalCheck User's Manual.* Cadence Design, Inc., 1998. 96

KV95. O. Kupferman and M. Y. Vardi. On the complexity of branching modular model checking. In *Proc. 6th CONCUR*, *LNCS* 962, pp. 408–422, 1995. 93

KV98a. O. Kupferman and M. Y. Vardi. Freedom, weakness, and determinism: from linear-time to branching-time. In *Proc. 13th LICS*, pp. 81–92, 1998. 93

KV98b. O. Kupferman and M. Y. Vardi. Relating linear and branching model checking. In *PROCOMET*, pp. 304 – 326, 1998. Chapman & Hall. 93, 94, 95

LP85. O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th POPL*, pp. 97–107, 1985. 83

MS95. D. E. Muller and P. E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141:69–107, 1995. 92

Pnu81. A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981. 83

PP95. B. Plessier and C. Pixley. Formal verification of a commercial serial bus interface. In *Proc. of 14th Annual IEEE International Phoenix Conf. on Computers and Communications*, pp. 378–382, March 1995. 83

QS81. J. P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 5th International Symp. on Programming*, pp. 337–351, LNCS 137, 1981. 83

SC85. A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logic. *Journal ACM*, 32:733–749, 1985. 95

VS85. M. Y. Vardi and L. Stockmeyer. Improved upper and lower bounds for modal logics of programs. In *Proc 17th STOC*, pp. 240–251, 1985. 93

VW86a. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st LICS*, pp. 322–331, 1986. 83, 95, 96

VW86b. M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Science*, 32(2):182–221, April 1986. 92

VW94. M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994. 92, 95