Model Checking of Safety Properties

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

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

Abstract. Of special interest in formal verification are safety properties, which assert that the system always stays within some allowed region. A computation that violates a general linear property reaches a bad cycle, which witnesses the violation of the property. Accordingly, current methods and tools for model checking of linear properties are based on a search for bad cycles. A symbolic implementation of such a search involves the calculation of a nested fixed-point expression over the system's state space, and is often very difficult. Every computation that violates a safety property has a finite prefix along which the property is violated. We use this fact in order to base model checking of safety properties on a search for finite bad prefixes. Such a search can be performed using a simple forward or backward symbolic reachability check. A naive methodology that is based on such a search involves a construction of an automaton (or a tableau) that is doubly exponential in the property. We present an analysis of safety properties that enables us to prevent the doubly-exponential blow up and to use the same automaton used for model checking of general properties, replacing the search for bad cycles by a search for bad prefixes.

1 Introduction

Today's rapid development of complex and safety-critical systems requires reliable verification methods. In formal verification, we verify that a system meets a desired property by checking that a mathematical model of the system meets a formal specification that describes the property. Of special interest are properties asserting that observed behavior of the system always stays within some allowed set of finite behaviors, in which nothing "bad" happens. For example, we may want to assert that every message received was previously sent. Such properties of systems are called *safety properties*. Intuitively, a property ψ is a safety property if every violation of ψ occurs after a finite execution of the system. In our example, if in a computation of the system a message is received without previously being sent, this occurs after some finite execution of the system.

In order to define safety properties formally, we refer to computations of a nonterminating system as infinite words over an alphabet Σ. Typically, $\Sigma = 2^{AP}$, where AP is the set of the system's atomic propositions. Consider a language L of infinite words over Σ. A finite word x over Σ is a *bad prefix* for L iff for all infinite words y over Σ, the concatenation $x \cdot y$ of x and y is not in L. Thus, a bad prefix for L is a finite word that cannot be

⋆ Part of this work was done when this author was visiting Cadence Berkeley Laboratories.
⋆⋆ Supported in part by the NSF grants CCR-9628400 and CCR-9700061, and by a grant from the Intel Corporation. Part of this work was done when this author was a Varon Visiting Professor at the Weizmann Institute of Science.

N. Halbwachs and D. Peled (Eds.): CAV'99, LNCS 1633, pp. 172–183, 1999.

extended to an infinite word in L. A language L is a *safety language* if every word not in L has a finite bad prefix. For example, $L = \{0^\omega, 1^\omega\} \subseteq \{0,1\}^\omega$ is a safety language: every word not in L contains 01 or 10, and a prefix that ends in one of these sequences cannot be extended to a word in L. The definition of safety we consider here is given in [AS85], it coincides with the definition of limit closure defined in [Eme83], and is different from the definition in [Lam85], which also refers to the property being closed under stuttering.

Linear properties of nonterminating systems are often specified using Büchi automata on infinite words or linear temporal logic (LTL) formulas. We say that an automaton $\mathcal{A}$ is a safety automaton if it recognizes a safety language. Similarly, an LTL formula is a safety formula if the set of computations that satisfy it form a safety language. Sistla shows that the problem of determining whether a nondeterministic Büchi automaton or an LTL formula are safety is PSPACE-complete [Sis94] (see also [AS87]). From the results in [KV97], it follows that the problem is in PSPACE even when the Büchi automaton is alternating. On the other hand, when the Büchi automaton is deterministic, the problem can be solved in linear time [MP92]. Sistla also describes sufficient syntactic requirements for safe LTL formulas. For example, a formula (in positive normal form) whose only temporal operators are G (always) and X (next), is a safety formula [Sis94]. Suppose that we want to verify the correctness of a system with respect to a safety property. Can we use the fact that the property is known to be a safety property in order to improve general verification methods? The positive answer to this question is the subject of this paper.

Much previous work on verification of safety properties follow the *proof-based* approach to verification [Fra92]. In the proof-based approach, the system is annotated with assertions and proof rules are used to verify the assertions. In particular, Manna and Pnueli consider verification of reactive systems with respect to safety properties in [MP92, MP95]. The definition of safety formulas considered in [MP92,MP95] is syntactic: a safety formula is a formula of the form $G\varphi$ where φ is a past formula. The syntactic definition is equivalent to the definition discussed here [MP92]. While proof-rules approaches are less sensitive to the size of the state space of the system, they require a heavy user support. Our work here considers the *state-exploration* approach to verification, where automatic *model checking* [CE81,QS81] is performed in order to verify the correctness of a system with respect to a specification. Previous work in this subject considers special cases of safety properties such as invariance checking [GW91,McM92,Val93,MR97], or assume that a general safety property is given by the set of its bad prefixes [GW91].

General methods for model checking of linear properties are based on a construction of a tableau or an automaton $\mathcal{A}_{\neg\psi}$ that accepts exactly all the infinite computations that violate the property ψ [LP85,VW94]. Given a system M and a property ψ, verification of M with respect to ψ is reduced to checking the emptiness of the product of M and $\mathcal{A}_{\neg\psi}$ [VW86]. This check can be performed on-the-fly and symbolically [CVWY92,GPVW95, TBK95]. When ψ is an LTL formula, the size of $\mathcal{A}_\psi$ is exponential in the length of ψ, and the complexity of verification that follows is PSPACE, with a matching lower bound [SC85].

Consider a safety property ψ. Let *pref*(ψ) denote the set of all bad prefixes for ψ. Recall that every computation that violates ψ has a prefix in *pref*(ψ). We say that an automaton on finite words is *tight* for a safety property ψ if it recognizes *pref*(ψ). Since every system that violates ψ has a computation with a prefix in *pref*(ψ), an automaton tight for ψ is practically more helpful than $\mathcal{A}_{\neg\psi}$. Indeed, reasoning about automata on finite words is easier than reasoning about automata on infinite words (cf. [HKSV97]). In particular,

when the words are finite, we can use backward or forward symbolic reachability analysis [BCM+92,IN97]. In addition, using an automaton for bad prefixes, we can return to the user a finite error trace, which is a bad prefix, and which is often more helpful than an infinite error trace.

Given a safety property ψ, we construct an automaton tight for ψ. We show that the construction involves an exponential blow-up in the case ψ is given as a nondeterministic Büchi automaton, and involves a doubly-exponential blow-up in the case ψ is given in LTL. These results are surprising, as they indicate that detection of bad prefixes with a nondeterministic automaton has the flavor of determinization. The tight automata we construct are indeed deterministic. Nevertheless, our construction avoids the difficult determinization of the Büchi automaton for ψ (cf. [Saf88]) and just uses a subset construction.

Our construction of tight automata reduces the problem of verification of safety properties to the problem of *invariance checking* [Fra92,MP92], Indeed, once we take the product of a tight automaton with the system, we only have to check that we never reach an accepting state of the tight automaton. Invariance checking is amenable to both model checking techniques [BCM+92,IN97] and deductive verification techniques [BM83, SOR93,MAB+94]. In practice, the verified systems are often very large, and even clever symbolic methods cannot cope with the state-explosion problem that model checking faces. The way we construct tight automata also enables, in case the BDDs constructed during the symbolic reachability test get too large, an analysis of the intermediate data that has been collected. The analysis can lead to a conclusion that the system does not satisfy the property without further traversal of the system.

In view of the discouraging blow-ups described above, we release the requirement on tight automata and seek, instead, an automaton that need not accept all the bad prefixes, yet must accept at least one bad prefix of every computation that does not satisfy ψ. We say that such an automaton is *fine* for ψ. For example, an automaton that recognizes $p^* \cdot (\neg p) \cdot (p \vee \neg p)$ does not accept all the words in *pref*(Gp), yet is fine for Gp. In practice, almost all the benefit that one obtain from a tight automaton can also be obtained from a fine automaton. We show that for natural safety formulas ψ, the construction of an automaton fine for ψ is as easy as the construction of $\mathcal{A}_\psi$. In order to formalize the notion of "natural safety formulas", we partition safety properties into *intentionally*, *accidentally*, and *pathologically* safe properties. While most safety properties are intentionally safe, accidentally safe and especially pathologically safe properties contain some redundancy, and we do not expect to see them often in practice. We show that the automaton $\mathcal{A}_{\neg\psi}$, which accepts exactly all infinite computations that violate ψ, can easily (and with no blow-up) be modified to an automaton $\mathcal{A}^{true}_{\neg\psi}$ on finite words, which is tight for ψ that is intentionally safe, and is fine for ψ that is accidentally safe. We present a methodology for model checking of safety properties that is based on the above classification, uses $\mathcal{A}^{true}_{\neg\psi}$ instead of $\mathcal{A}_{\neg\psi}$, and thus replaces the search for bad cycles by a search for bad prefixes.

2 Preliminaries

2.1 Safety Languages and Formulas

< Consider a language $L \subseteq \Sigma^\omega$ of infinite words over the alphabet Σ. A finite word $x \in \Sigma^*$ is a *bad prefix* for L iff for all $y \in \Sigma^\omega$, we have $x \cdot y \notin L$. A language L is a *safety* language iff every $w \notin L$ has a finite bad prefix. For a safety language L, we denote by *pref*(L) the set of all bad prefixes for L. We say that a set $X \subseteq$ *pref*(L) is a *trap* for a

safety language L iff every word $w \notin L$ has at least one prefix in X. We denote all the traps for L by *trap*(L).

For a language $L \subseteq \Sigma^\omega$, we use *comp*$(L)$ to denote the complement of L; i.e., *comp*$(L) = \Sigma^\omega \setminus L$. We say that a language $L \subseteq \Sigma^\omega$ is a *co-safety* language iff *comp*(L) is a safety language. (The term used in [MP92] is *guarantee* language.) Equivalently, L is co-safety iff every $w \in L$ has a *good prefix* $x \in \Sigma^*$ such that for all $y \in \Sigma^\omega$, we have $x \cdot y \in L$. For a co-safety language L, we denote by *co-pref*(L) the set of good prefixes for L. Note that *co-pref*(L) = *pref*(*comp*(L)).

For an LTL formula ψ over a set AP of atomic propositions, let $\|\psi\|$ denote the set of computations in $(2^{AP})^\omega$ that satisfy ψ. We say that ψ is a safety formula iff $\|\psi\|$ is a safety language. Also, ψ is a co-safety formula iff $\|\psi\|$ is a co-safety language or, equivalently, $\|\neg\psi\|$ is a safety language.

2.2 Word Automata

Given an alphabet Σ, an *infinite word over* Σ is an infinite sequence $w = \sigma_1 \cdot \sigma_2 \cdots$ of letters in Σ. We denote by w^l the suffix $\sigma_l \cdot \sigma_{l+1} \cdot \sigma_{l+2} \cdots$ of w. An automaton on infinite words is $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, F \rangle$, where Σ is the input alphabet, Q is a finite set of states, δ is a transition function, $Q_0 \subseteq Q$ is a set of initial states, and $F \subseteq Q$ is an acceptance condition. When $\mathcal{A}$ is *deterministic*, the size of Q_0 is 1, and $\delta : Q \times \Sigma \to Q$ maps each state and letter to a single successor state. When $\mathcal{A}$ is *nondeterministic*, $\delta : Q \times \Sigma \to 2^Q$ maps each state and letter to a possible set of successor states. Since the choice of a successor state is existential, we can regard a transition $\rho(q, \sigma) = \{q_1, q_2, q_3\}$ as a disjunction $q_1 \vee q_2 \vee q_3$. Transitions of *alternating* automata can be arbitrary positive formulas over Q. We can have, for instance, a transition $\delta(q, \sigma) = (q_1 \wedge q_2) \vee (q_3 \wedge q_4)$, meaning that the automaton accepts from state q a suffix w^l, starting by σ, of w, if it accepts w^{l+1} from both q_1 and q_2 or from both q_3 and q_4. Such a transition combines existential and universal choices. Runs of an alternating automaton are infinite trees, where branches corresponds to universal choices of the automaton. For example, if $\mathcal{A}$ is an automaton with an initial state q_0 and $\delta(q_{in}, \sigma_0) = (q_1 \vee q_2) \wedge (q_3 \vee q_4)$, then possible runs of $\mathcal{A}$ on w have a root labeled q_{in}, have one node in level 1 labeled q_1 or q_2, and have another node in level 1 labeled q_3 or q_4. When $\mathcal{A}$ is a *Büchi* automaton on infinite words, a run is *accepting* iff it visits infinitely many states from F along each of its branches. The automaton $\mathcal{A}$ can also run on finite words in Σ^*. Then, a run over a word in Σ^n is accepting if it visits states in F in it all its nodes of level n. A word (either finite or infinite) is accepted by $\mathcal{A}$ iff there exists an accepting run on it. The language of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts. Deterministic and nondeterministic automata can be viewed as special cases of alternating automata. Formally, an alternating automaton is deterministic if for all q and σ, we have $\delta(q, \sigma) \in Q \cup \{\mathit{false}\}$, and it is nondeterministic if $\delta(q, \sigma)$ is always a disjunction. For a detailed definition of alternating automata see [Var96].

We define the *size* of an alternating automaton $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, F \rangle$ as the sum of $|Q|$ and $|\delta|$, where $|\delta|$ is the sum of the lengths of the formulas in δ. We say that the automaton $\mathcal{A}$ over infinite words is a safety (co-safety) automaton iff $\mathcal{L}(\mathcal{A})$ is a safety (co-safety) language. We use *pref*$(\mathcal{A})$, *co-pref*$(\mathcal{A})$, *trap*$(\mathcal{A})$, and *comp*$(\mathcal{A})$ to abbreviate *pref*$(\mathcal{L}(\mathcal{A}))$, *co-pref*$(\mathcal{L}(\mathcal{A}))$, *trap*$(\mathcal{L}(\mathcal{A}))$, and *comp*$(\mathcal{L}(\mathcal{A}))$, respectively. For an automaton $\mathcal{A}$ and a set of states S, we denote by $\mathcal{A}^S$ the automaton obtained from $\mathcal{A}$ by defining the set of initial states to be S. We say that an automaton $\mathcal{A}$ over infinite words is *universal* iff $\mathcal{L}(\mathcal{A}) = \Sigma^\omega$.

When $\mathcal{A}$ runs on finite words, it is universal iff $\mathcal{L}(\mathcal{A}) = \Sigma^*$. An automaton is *empty* iff $\mathcal{L}(\mathcal{A}) = \emptyset$. A set S of states is *universal* (resp., *rejecting*), when $\mathcal{A}^S$ is universal (resp., empty). Note that the universality problem for nondeterministic automata is known to be PSPACE-complete [MS72,Wol82].

3 Detecting Bad Prefixes

Linear properties of nonterminating systems are often specified using automata on infinite words or linear temporal logic (LTL) formulas. Given an LTL formula ψ, one can build a nondeterministic Büchi automaton $\mathcal{A}_\psi$ that recognizes $\|\psi\|$. The size of $\mathcal{A}_\psi$ is, in the worst case, exponential in ψ [GPVW95,VW94]. In practice, when given a property that happens to be safe, what we want is a nondeterministic automaton on finite words that detects bad prefixes. As we discuss in the introduction, such an automaton is easier to reason about. In this section we construct, from a given safety property, an automaton for its bad prefixes.

We first study the case where the property is given by a nondeterministic Büchi automaton. When the given automaton $\mathcal{A}$ is deterministic, the construction of an automaton $\mathcal{A}'$ for *pref*$(\mathcal{A})$ is straightforward. Indeed, we can obtain $\mathcal{A}'$ from $\mathcal{A}$ by defining the set of accepting states to be the set of states s for which $\mathcal{A}^s$ is empty. Theorem 1 below shows that when $\mathcal{A}$ is a nondeterministic automaton, things are not that simple. While we can avoid a difficult determinization of $\mathcal{A}$ [Saf88], we cannot avoid an exponential blow-up.

Theorem 1. *Given a safety nondeterministic Büchi automaton $\mathcal{A}$ of size n, the size of an automaton that recognizes pref$(\mathcal{A})$ is $2^{\Theta(n)}$.*

Proof. We start with the upper bound. Let $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, F \rangle$. Recall that *pref*$(\mathcal{L}(\mathcal{A}))$ contains exactly all prefixes $x \in \Sigma^*$ such that for all $y \in \Sigma^\omega$, we have $x \cdot y \notin \mathcal{L}(\mathcal{A})$. Accordingly, the automaton for *pref*$(\mathcal{A})$ accepts a prefix x iff the set of states that $\mathcal{A}$ could be in after reading x is rejecting. Formally, we define the (deterministic) automaton $\mathcal{A}' = \langle \Sigma, 2^Q, \delta', \{Q_0\}, F' \rangle$, where F' contains all the rejecting sets of $\mathcal{A}$, and δ' follows the subset construction induced by δ; that is, for every $S \in 2^Q$ and $\sigma \in \Sigma$, we have $\delta'(S, \sigma) = \bigvee_{s \in S} \delta(s, \sigma)$.

We now turn to the lower bound. Essentially, it follows from the fact that *pref*$(\mathcal{A})$ refers to words that are not accepted by $\mathcal{A}$, and hence, it has the flavor of complementation. Complementing a nondeterministic automaton on finite words involves an exponential blow-up [MF71]. In fact, one can construct a nondeterministic automaton $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, Q \rangle$, in which all states are accepting, such that the smallest nondeterministic automaton that recognizes *comp*$(\mathcal{A})$ has $2^{\Theta(|Q|)}$ states. (To see this, consider the language L_n consisting all all words w such that either $|w| < 2n$ or $w = uvz$, where $|u| = |v| = n$ and $u \neq v$.) Given $\mathcal{A}$ as above, let $\mathcal{A}'$ be $\mathcal{A}$ when regarded as a Büchi automaton on infinite words. It is not hard to see that *pref*$(\mathcal{A}')$ = *comp*$(\mathcal{A})$.

The lower bound in Theorem 1 is not surprising, as complementation of nondeterministic automata involves an exponential blow-up, and, as we demonstrate in the lower-bound proof, there is a tight relation between *pref*$(\mathcal{A})$ and *comp*$(\mathcal{A})$. We could hope, therefore, that when properties are specified in a negative form (that is, they describe the forbidden behaviors of the system) or are given in LTL, whose formulas can be negated, detection of bad prefixes would not be harder than detection of bad computations. In Theorems 2 and 3 we refute this hope.

Theorem 2. *Given a co-safety nondeterministic Büchi automaton $\mathcal{A}$ of size n, the size of an automaton that recognizes co-pref$(\mathcal{L}(\mathcal{A}))$ is $2^{\Theta(n)}$.*

Proof. The upper bound is similar to the one in Theorem 1, only that now we define the set of accepting states in $\mathcal{A}'$ as the set of all the universal sets of $\mathcal{A}$. We prove a matching lower bound. For $n \geq 1$, let $\Sigma_n = \{1, \ldots, n, \&\}$. We define L_n as the language of all words $w \in \Sigma_n^\omega$ such that w contains at least one $\&$ and the letter after the first $\&$ is either $\&$ or it has already appeared somewhere before the first $\&$. The language L_n is a co-safety language. Indeed, each word in L_n has a good prefix (e.g., the one that contains the first $\&$ and its successor). We can recognize L_n with a nondeterministic Büchi automaton with $O(n)$ states (the automaton guesses the letter that appears after the first $\&$). Obvious good prefixes for L_n are $12\&\&$, $123\&2$, etc. We can recognize these prefixes with a nondeterministic automaton with $O(n)$ states. But L_n also has some less obvious good prefixes, like $1234 \cdots n\&$ (a permutation of $1 \ldots n$ followed by $\&$). These prefixes are indeed good, as every suffix we concatenate to them would start in either $\&$ or a letter in $\{1, \ldots, n\}$ that has appeared before the $\&$. To recognize these prefixes, a nondeterministic automaton needs to keep track of subsets of $\{1, \ldots, n\}$, for which it needs 2^n states. Consequently, a nondeterministic automaton for *co-pref*(L_n) must have at least 2^n states.

We now extend the proof of Theorem 2 to get a doubly-exponential lower bound for going from a safety LTL formula to a nondeterministic automaton for its bad prefixes. The idea is similar: while the proof in Theorem 2 uses the exponential lower bound for going from nondeterministic to deterministic Büchi automata, the proof for this case is a variant of the doubly exponential lower bound for going from LTL formulas to deterministic Büchi automata [KV98].

Theorem 3. *Given a safety LTL formula, the size of a nondeterministic Büchi automaton for pref(ψ) is doubly exponential in the length of ψ.*

In order to get the upper bound in Theorem 3, we apply the exponential construction in Theorem 1 to the exponential Büchi automaton $\mathcal{A}_\psi$ for $\|\psi\|$. The construction in Theorem 1 is based on a subset construction for $\mathcal{A}_\psi$, and it requires a check for the universality of sets of states Q of $\mathcal{A}_\psi$. Such a check corresponds to a validity check for a DNF formula in which each disjunct corresponds to a state in Q. While the size of the formula can be exponential in $|\psi|$, the number of distinct literals in the formula is at most linear in $|\psi|$, implying that the the universality of Q can be checked using space polynomial in $|\psi|$.

Given a safety formula ψ, we say that a nondeterministic automaton $\mathcal{A}$ over finite words is *tight* for ψ iff $\mathcal{L}(\mathcal{A}) = \mathit{pref}(\|\psi\|)$. In view of the lower bounds proven above, a construction of tight automata may be too expensive. We say that a nondeterministic automaton $\mathcal{A}$ over finite words is *fine* for ψ iff there exists $X \in \mathit{trap}(\|\psi\|)$ such that $\mathcal{L}(\mathcal{A}) = X$. Thus, a fine automaton need not accept all the bad prefixes, yet it must accept at least one bad prefix of every computation that does not satisfy ψ. In practice, almost all the benefit that one obtain from a tight automaton can also be obtained from a fine automaton (we will get back to this point in Section 6). It is an open question whether there are feasible constructions of fine automata for general safety formulas. In Section 5 we show that for natural safety formulas ψ, the construction of an automaton fine for ψ is as easy as the construction of an automaton for ψ.

4 Symbolic Verification of Safety Properties

Our construction of tight automata reduces the problem of verification of safety properties to the problem of invariance checking, which is amenable to a large variety of techniques. In particular, backward and forward symbolic reachability analysis have proven to be effective techniques for checking invariant properties on systems with large state spaces [BCM+92,IN97]. In practice, however, the verified systems are often very large, and even clever symbolic methods cannot cope with the state-explosion problem that model checking faces. In this section we describe how the the way we construct tight automata enables, in case the BDDs constructed during the symbolic reachability test get too big, an analysis of the intermediate data that has been collected. The analysis solves the model-checking problem without further traversal of the system.

Consider a system $M = \langle AP, W, R, W_0, L\rangle$, where W is the set of states, $R \subseteq W \times W$ is a transition relation, W_0 is a set of initial states, and $L : W \to 2^{AP}$ maps each state to the sets of atomic propositions that hold in it. Let $\mathit{fin}(M)$ be an automaton that accepts all finite computations of M. Given ψ, let $\mathcal{A}_{\neg\psi}$ be the nondeterministic co-safety automaton for $\neg\psi$, thus $\mathcal{L}(\mathcal{A}_{\neg\psi}) = \|\neg\psi\|$. In the proof of Theorem 2, we construct an automaton $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') = \mathit{pref}(\psi)$ by following the subset construction of $\mathcal{A}_{\neg\psi}$ and defining the set of accepting states to be the set of universal sets in $\mathcal{A}_{\neg\psi}$. Then, one needs to verify the invariance that the product $\mathit{fin}(M) \times \mathcal{A}'$ never reaches an accepting state of $\mathcal{A}'$. In addition to forward and backward symbolic reachability analysis, one could use a variety of recent techniques for doing semi-exhaustive reachability analysis [RS95,YSAA97], including standard simulation techniques [LWA98]. Note, however, that if $\mathcal{A}'$ is doubly exponential in $|\psi|$, the BDD representation of $\mathcal{A}'$ will use exponentially (in $|\psi|$) many Boolean variables.

Another approach is to apply forward reachability analysis to the product $M \times \mathcal{A}_{\neg\psi}$ of the system M and the automaton $\mathcal{A}_{\neg\psi}$. Formally, let $\mathcal{A}_{\neg\psi} = \langle 2^{AP}, Q, \delta, Q_0, F\rangle$, and let M be as above. The product $M \times \mathcal{A}_{\neg\psi}$ has state space $W \times Q$, and the successors of a state $\langle w, q\rangle$ are all pairs $\langle w', q'\rangle$ such that $R(w, w')$ and $q' \in \delta(q, L(w))$. Forward symbolic methods use the predicate $\mathit{post}(S)$, which, given a set of S of states (represented symbolically) returns the successor set of S, that is, the set of all states t such that there is a transition from some state in S to t. Starting from the initial set $S_0 = W_0 \times Q_0$, forward symbolic methods iteratively construct, for $i \geq 0$, the set $S_{i+1} = \mathit{post}(S_i)$. The calculation the S_i's proceeds symbolically, and they are represented by BDDs. Doing so, forward symbolic methods actually follow the subset construction of $M \times \mathcal{A}_{\neg\psi}$. Indeed, for each $w \in W$ the set $Q_i^w = \{q : \langle w, q\rangle \in S_i\}$ is the set of states that $\mathcal{A}_{\neg\psi}$ that can be reached via a path of length i in M from a state in W_0 to the state w. Note that this set can be exponentially (in $|\psi|$) large resulting possibly in a large BDD; on the other hand, the number of Boolean variables used to represent $\mathcal{A}_{\neg\psi}$ is linear in $|\psi|$.

The discussion above suggests the following technique for the case we encounter space problems. Suppose that at some point the BDD for S_i gets too big. We then check whether there is a state w such that the set Q_i^w is universal. As discussed in Section 3, we can check the universality of Q_i^w in space polynomial in $|\psi|$. Note that we do not need to enumerate all states w and then check Q_i^w. We can enumerate directly the sets Q_i^w, whose number is at most doubly exponential in $|\psi|$. It can be shown that $M \times \mathcal{A}_{\neg\psi}$ is nonempty iff Q_i^w is universal for some $w \in W$ and $i > 0$, thus this check solves the model-checking problem without further traversal of the system.

Note that it is possible to use semi-exhaustive reachability techniques also when analyzing $M \times \mathcal{A}_{\neg\psi}$. That is, instead of taking S_{i+1} to be *post*(S_i) we can take it to be a subset S'_{i+1} of *post*(S_i) [RS95,YSAA97]. We have to ensure, however, that S'_{i+1} is *saturated* with respect to states of $\mathcal{A}_{\neg\psi}$ [LWA98]. Informally, we are allowed to drop states of M from S_{i+1}, but we are not allowed to drop states of $\mathcal{A}_{\neg\psi}$. Formally, if $\langle w, q\rangle \in S'_{i+1}$ and $\langle w, q'\rangle \in S_{i+1}$, then $\langle w, q'\rangle \in S'_{i+1}$. This ensures that if the semi-exhaustive analysis follows a bad prefix of length i in M, then $Q'^{w}_i = \{q : \langle w, q\rangle \in S'_i\}$ will be universal. In the extreme case, we follow only one trace of M, i.e., we simulate M. In that case, we have that $S'_{i+1} = \{w\} \times Q'^{w}_i$. For a related approach see [CES97].

5 Classification of Safety Properties

Consider the safety LTL formula Gp. A bad prefix x for Gp must contain a state in which p does not hold. If the user gets x as an error trace, he can immediately understand why Gp is violated. Consider now the LTL formula $\psi = G(p \vee (Xq \wedge X\neg q))$. The formula ψ is equivalent to Gp and is therefore a safety formula. Moreover, the set of bad prefixes for ψ and Gp coincide. Nevertheless, a minimal bad prefix for ψ (e.g., a single state in which p does not hold) does not tell the whole story about the violation of ψ. Indeed, the latter depends on the fact that $Xq \wedge X\neg q$ is unsatisfiable, which (especially in more complicated examples), may not be trivially noticed by the user. This intuition, of a prefix that "tells the whole story", is the base for a classification of safety properties into three distinct safety levels. We first formalize this intuition in terms of *informative prefixes*. We assume that LTL formulas are given in positive normal form, where negation is applied only to propositions (when we write $\neg\psi$, we refer to its positive normal form). In the positive normal form, we use the operator V as dual to the operator U, and use $cl(\psi)$ to denote the closure of ψ, namely, the set of ψ's subformulas.

For an LTL formula ψ and a finite computation $\pi = \sigma_1 \cdot \sigma_2 \cdots \sigma_n$, with $\sigma_i \in 2^{AP}$, we say that π is *informative for* ψ iff there exists a mapping $L : \{1, \ldots, n+1\} \to 2^{cl(\psi)}$ such that the following hold: (1) $\neg\psi \in L(1)$. (2) $L(n+1)$ is empty. (3) For all $1 \leq i \leq n$ and $\varphi \in L(i)$, the following hold.

- If φ is a propositional assertion, it is satisfied by σ_i.
- If $\varphi = \varphi_1 \vee \varphi_2$ then $\varphi_1 \in L(i)$ or $\varphi_2 \in L(i)$.
- If $\varphi = \varphi_1 \wedge \varphi_2$ then $\varphi_1 \in L(i)$ and $\varphi_2 \in L(i)$.
- If $\varphi = X\varphi_1$, then $\varphi_1 \in L(i+1)$.
- If $\varphi = \varphi_1 U \varphi_2$, then $\varphi_2 \in L(i)$ or $[\varphi_1 \in L(i)$ and $\varphi_1 U \varphi_2 \in L(i+1)]$.
- If $\varphi = \varphi_1 V \varphi_2$, then $\varphi_2 \in L(i)$ and $[\varphi_1 \in L(i)$ or $\varphi_1 V \varphi_2 \in L(i+1)]$.

Note that the emptiness of $L(n+1)$ guarantees that all the requirements imposed by $\neg\psi$ are fulfilled along π. For example, while the finite computation $\{p\} \cdot \emptyset$ is informative for Gp (with $L(1) = \{F\neg p\}$,$L(2) = \{F\neg p, \neg p\}$, and $L(3) = \emptyset$), it is not informative for $\psi = G(p \vee (Xq \wedge X\neg q))$. Indeed, as $\neg\psi = F(\neg p \wedge (X\neg q \vee Xq))$, an informative prefix for ψ must contain at least one state after the first state in which $\neg p$ holds.

We distinguish between three types of safety formulas.

- A safety formula ψ is *intentionally safe* iff all the bad prefixes for ψ are informative. For example, the formula Gp is intentionally safe.

- A safety formula ψ is *accidentally safe* iff not all the bad prefixes for ψ are informative, but every computation that violates ψ has an informative bad prefix. For example, the formulas $G(q \vee XGp) \wedge G(r \vee XG\neg p)$ and $G(p \vee (Xq \wedge X\neg q))$ are accidentally safe.
- A safety formula ψ is *pathologically safe* if there is a computation that violates ψ and has no informative bad prefix. For example, the formula $[G(q \vee GFp) \wedge G(r \vee GF\neg p)] \vee Gq \vee Gr$ is pathologically safe.

Sistla has shown that all temporal formulas in positive normal form constructed with the temporal connectives X and V are safety formulas [Sis94]. We call such formulas *syntactically safe*. The following strengthens Sistla's result.

Theorem 4. *If ψ is syntactically safe, then ψ is intentionally or accidentally safe.*

Given an LTL formula ψ in positive normal form, one can build an alternating Büchi automaton $\mathcal{A}_\psi = \langle 2^{AP}, Q, \delta, Q_0, F\rangle$ such that $\mathcal{L}(\mathcal{A}_\psi) = \|\psi\|$. Essentially, each state of $\mathcal{L}(\mathcal{A}_\psi)$ corresponds to a subformula of ψ, and its transitions follow the semantics of LTL [Var96]. We define the alternating Büchi automaton $\mathcal{A}^{true}_\psi = \langle 2^{AP}, Q, \delta, Q_0, \emptyset\rangle$ by redefining the set of accepting states to be the empty set. So, while in $\mathcal{A}_\psi$ a copy of the automaton may accept by either reaching a state from which it proceed to *true* or visiting states of the form $\varphi_1 V \varphi_2$ infinitely often, in $\mathcal{A}^{true}_\psi$ all copies must reach a state from which they proceed to *true*. Accordingly, $\mathcal{A}^{true}_\psi$ accepts exactly these computations that have a finite prefix that is informative for ψ. To see this, note that such computations can be accepted by a run of $\mathcal{A}_\psi$ in which all the copies eventually reach a state that is associated with propositional assertions that are satisfied. Now, let *fin*$(\mathcal{A}^{true}_\psi)$ be $\mathcal{A}^{true}_\psi$ when regarded as an automaton on finite words.

Theorem 5. *For every safety formula ψ, the automaton fin$(\mathcal{A}^{true}_{\neg\psi})$ accepts exactly all the prefixes that are informative for ψ.*

Corollary 1. *Consider a safety formula ψ. If ψ is intentionally safe, then fin$(\mathcal{A}^{true}_{\neg\psi})$ is tight for ψ. Also, if ψ is accidentally safe, then fin$(\mathcal{A}^{true}_{\neg\psi})$ is fine for ψ.*

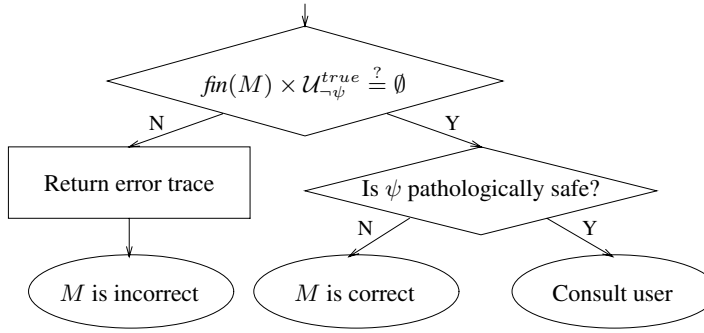
Theorem 6. *Deciding whether a given formula is pathologically safe is PSPACE-complete.*

Proof. Consider a formula ψ. Recall that the automaton $\mathcal{A}^{true}_\psi$ accepts exactly these computations that have a finite prefix that is informative for ψ. Hence, ψ is not pathologically safe iff every computation that does not satisfy ψ is accepted by $\mathcal{A}^{true}_{\neg\psi}$. Accordingly, checking whether ψ is pathologically safe can be reduced to checking the containment of $\mathcal{L}(\mathcal{A}_{\neg\psi})$ in $\mathcal{L}(\mathcal{A}^{true}_{\neg\psi})$. Since the size of $\mathcal{A}_\psi$ is linear in the length of ψ and containment for alternating Büchi automata can be checked in polynomial space [KV97], we are done. For the lower bound, we do a reduction from the problem of deciding whether a given formula is a safety formula. Consider a formula ψ, and let p, q, and r be atomic propositions not in ψ. The formula $\varphi = [G(q \vee GFp) \wedge G(r \vee GF\neg p)] \vee Gq \vee Gr$ is pathologically safe. It can be shown that ψ is a safety formula iff $\psi \wedge \varphi$ is pathologically safe.

Note that the lower bound in Theorem 6 implies that the reverse direction of Theorem 4 does not hold.

6 A Methodology

In Section 5, we partitioned safety formulas into three safety levels and showed that for some formulas, we can circumvent the blow-up involved in constructing a tight automaton for the bad prefixes. In particular, we showed that the automaton $\mathit{fin}(\mathcal{A}^{true}_{\neg\psi})$, which is linear in the length of ψ, is tight for ψ that is intentionally safe and is fine for ψ that is accidentally safe. In this section we describe a methodology for efficient verification of safety properties that is based on these observations. Consider a system M and a safety LTL formula ψ. Let $\mathit{fin}(M)$ be a nondeterministic automaton on finite words that accepts the prefixes of computations of M, and let $\mathcal{U}^{true}_{\neg\psi}$ be the nondeterministic automaton on finite words equivalent to the alternating automaton $\mathit{fin}(\mathcal{A}^{true}_{\neg\psi})$ [CKS81]. The size of $\mathcal{U}^{true}_{\neg\psi}$ is exponential in the size of $\mathit{fin}(\mathcal{A}^{true}_{\neg\psi})$, hence it is exponential in the length of ψ. Given M and ψ, we suggest to proceed as follows (see the figure below).



Instead of checking the emptiness of $M \times \mathcal{A}_{\neg\psi}$, verification starts by checking $\mathit{fin}(M)$ with respect to $\mathcal{U}^{true}_{\neg\psi}$. Since both automata refer to finite words, this can be done using finite forward reachability analysis. If the product $\mathit{fin}(M) \times \mathcal{U}^{true}_{\neg\psi}$ is not empty, we return a word w in the intersection, namely, a bad prefix for ψ that is generated by M[1]. If the product $\mathit{fin}(M) \times \mathcal{U}^{true}_{\neg\psi}$ is empty, then, as $\mathcal{U}^{true}_{\neg\psi}$ is fine for intentionally and accidentally safe formulas, there may be two reasons for this. One, is that M satisfies ψ, and the second is that ψ is pathologically safe. Therefore, we next check whether ψ is pathologically safe. (Note that for syntactically safe formulas this check is unnecessary, by Theorem 4.) If it is not pathologically safe, we conclude that M satisfies ψ. Otherwise, we tell the user that his formula is pathologically safe, indicating that his specification is needlessly complicated (accidentally and pathologically safe formulas contain redundancy). At this point, the user would probably be surprised that his formula was a safety formula (if he had known it is safety, he would have simplified it to an intentionally safe formula – a feasible automatic simplification of such formulas is an open problem). If the user wishes to continue with this formula, we give up using the fact that ψ is safety and proceed with usual LTL model checking, thus we check the emptiness of $M \times \mathcal{A}_{\neg\psi}$. (Recall that the symbolic algorithm for emptiness of Büchi automata is in the worst case quadratic [HKSV97,TBK95].) Note that at this point, the error trace that the user gets if M does not satisfy ψ consists of a prefix and a cycle, yet since the user does not want to change his formula, he probably has no idea why it is a safety formula and a finite non-informative error trace would not

[1] Note that since ψ may not be intentionally safe, the automaton $\mathcal{U}^{true}_{\neg\psi}$ may not be tight for ψ, thus while w is a minimal informative bad prefix, it may not be a minimal bad prefix.

help him). If the user prefers, or if M is very large (making the discovery of bad cycles infeasible), we can build an automaton for *pref*(ψ), hoping that by learning it, the user would understand how to simplify his formula or that, in spite of the potential blow-up in ψ, finite forward reachability would work better.

Acknowledgement. The second author is grateful to Avner Landver for stimulating discussions.

References

[AS85] B. Alpern and F.B. Schneider. Defining liveness. *Information processing letters*, 21:181–185, 1985.

[AS87] B. Alpern and F.B. Schneider. Recognizing safety and liveness. *Distributed computing*, 2:117–126, 1987.

[BCM+92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[BM83] R.S. Boyer and J.S. Moore. Proof-checking, theorem-proving and program verification. Technical Report 35, Institute for Computing Science and Computer Applications, University of Texas at Austin, January 1983.

[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.

[CES97] W. Canfield, E.A. Emerson, and A. Saha. Checking formal specifications under simulation. In *Proc. International Conference on Computer Design*, pp. 455–460, 1997.

[CKS81] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, January 1981.

[CVWY92] C. Courcoubetis, M.Y. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1:275–288, 1992.

[Eme83] E.A. Emerson. Alternative semantics for temporal logics. *Theoretical Computer Science*, 26:121–130, 1983.

[Fra92] N. Francez. *Program verification*. International Computer Science. Addison-Weflay, 1992.

[GPVW95] R. Gerth, D. Peled, M.Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification, Testing, and Verification*, pp. 3–18. Chapman & Hall, August 1995.

[GW91] P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. In *Proc. 3rd CAV*, LNCS 575, pp. 332–342, 1991.

[HKSV97] R.H. Hardin, R.P. Kurshan, S.K. Shukla, and M.Y. Vardi. A new heuristic for bad cycle detection using BDDs. In *Proc. 9th CAV*, LNCS 1254, pp. 268–278, 1997.

[IN97] H. Iwashita and T. Nakata. Forward model checking techniques oriented to buggy designs. In *Proc. IEEE/ACM ICCAD*, pp. 400–404, 1997.

[KV97] O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. In *Proc. 5th ISTCS*, pp. 147–158. IEEE Computer Society Press, 1997.

[KV98] O. Kupferman and M.Y. Vardi. Freedom, weakness, and determinism: from linear-time to branching-time. In *Proc. 13th LICS*, pp. 81–92, June 1998.

[Lam85] L. Lamport. Logical foundation. In *Distributed systems - methods and tools for specification*, LNCS 190, 1985.

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

[LWA98] Y. Luo, T. Wongsonegoro, and A. Aziz. Hybrid techniques for fast functional simulation. In *Proc. 35th DAC*. IEEE Computer Society, 1998.

[MAB+94] Z. Manna, A. Anuchitanukul, N. Bjorner, A. Browne, E. Chang, M. Colon, L. De Alfaro, H. Devarajan, H. Sipma, and T. Uribe. STeP: The Stanford Temporal Prover. Technical Report STAN-CS-TR-94-1518, Dept. of Computer Science, Stanford University, 1994.

[McM92] K. McMillan. Using unfolding to avoid the state explosion problem in the verification of asynchronous circuits. In *Proc. 4th CAV*, LNCS 663, pp. 164–174, 1992.

[MF71] A.R. Meyer and M.J. Fischer. Economy of description by automata, grammars, and formal systems. In *Proc. 12th IEEE Symp. on Switching and Automata Theory*, pp. 188–191, 1971.

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

[MP95] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Safety*. Springer-Verlag, New York, 1995.

[MR97] S. Melzer and S. Roemer. Deadlock checking using net unfoldings. In *Proc. 9th CAV*, LNCS 1254, pp. 364–375, 1997.

[MS72] A.R. Meyer and L.J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time. In *Proc. 13th IEEE Symp. on Switching and Automata Theory*, pp. 125–129, 1972.

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

[RS95] K. Ravi and F. Somenzi. High-density reachability analysis. In *Proc. CAD*, pp. 154–158, 1995.

[Saf88] S. Safra. On the complexity of ω-automata. In *Proc. 29th FOCS*, pp. 319–327, White Plains, 1988.

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

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

[SOR93] R.E. Shankar, S. Owre, and J.M. Rushby. The PVS proof checker: A reference manual (beta release). Technical report, Computer Science laboratory, SRI International, Menlo Park, California, March 1993.

[TBK95] H.J. Touati, R.K. Brayton, and R. Kurshan. Testing language containment for ω-automata using BDD's. *Information and Computation*, 118(1):101–109, April 1995.

[Val93] A. Valmari. On-the-fly verification with stubborn sets. In *Proc. 5nd CAV*, LNCS 697, 1993.

[Var96] M.Y. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, LNCS 1043, pp. 238–266, 1996.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st LICS*, pp. 322–331, 1986.

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

[Wol82] P. Wolper. *Synthesis of Communicating Processes from Temporal Logic Specifications*. PhD thesis, Stanford University, 1982.

[YSAA97] J. Yuan, J. Shen, J. Abraham, and A. Aziz. On combining formal and informal verification. In *Proc 9th CAV*, LNCS 1254, pp. 376–387, 1997.