Formal Methods in System Design, 19, 291–314, 2001

Model Checking of Safety Properties

ORNA KUPFERMAN orna@cs.huji.ac.il
School of Computer Science and Engineering, Hebrew University, Jerusalem 91904, Israel

MOSHE Y. VARDI vardi@cs.rice.edu
Department of Computer Science, Rice University, Houston, TX 77251-1892, USA

Received August 25, 1999; Accepted September 15, 2000

Abstract. Of special interest in formal verification are safety properties, which assert that the system always stays within some allowed region. Proof rules for the verification of safety properties have been developed in the proof-based approach to verification, making verification of safety properties simpler than verification of general properties. In this paper we consider model checking of safety properties. 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 infeasible. 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.

Keywords: model checking, safety properties, automata

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 formally define what safety properties are, 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 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, if $\Sigma = \{0, 1\}$, then $L = \{0^\omega, 1^\omega\}$ is a safety language. To see this, note that every word not in L contains either the sequence 01 or the sequence 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 [2], it coincides with the definition of limit closure defined in [13], and is different from the definition in [24], 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 [40] (see also [3]). On the other hand, when the Büchi automaton is deterministic, the problem can be solved in linear time [31]. 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 [40]. 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 [15]. 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 [31, 32]. The definition of safety formulas considered in [31, 32] 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 [31]. Proof-based methods are also known for the verification of *liveness* properties [35], which assert that the system eventually reaches some good set of states. While proof-rule 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* [8, 36] 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 and liveness properties such as invariance checking [17, 28, 33, 43], or assume that a general safety property is given by the set of its bad prefixes [17].

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 [25, 47]. 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 $\mathcal{A}_{\neg\psi}$ [45]. This check can be performed on-the-fly and symbolically [12, 16, 42]. 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 [39].

Consider a safety property ψ. Let $pref(\psi)$ denote the set of all bad prefixes for ψ. For example, $pref(Gp)$ contains all finite words that have a position in which p does not hold. Recall that every computation that violates ψ has a prefix in $pref(\psi)$. We say that an automaton on finite words is *tight* for a safety property ψ if it recognizes $pref(\psi)$. Since every system that violates ψ has a computation with a prefix in $pref(\psi)$, 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. [18]). In particular, when the words are finite, we can use backward or forward symbolic reachability analysis [5, 19]. 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. [38]) 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* [15, 31]. 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 [5, 19] and deductive verification techniques [7, 27, 41]. 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 $\mathcal{A}_{\psi}$.

To formalize the notion of "natural safety formulas," consider the safety LTL formula $\psi = G(p \vee (Xq \wedge X\neg q))$. A single state in which p does not hold is a bad prefix for ψ. Nevertheless, this prefix 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. So, while some bad prefixes are *informative*, namely, they tell the whole violation story, other bad prefixes may not be informative, and some user intelligence is required in order to understand why they are bad prefixes (the formal definition of informative prefixes is similar to the semantics of LTL over finite computations in which $X\mathbf{true}$ does not hold in the final position).

The notion of informative prefixes is the base for a classification of safety properties into three distinct safety levels. A property is *intentionally safe* if all its bad prefixes are informative. For example, the formula Gp is intentionally safe. A property ψ is *accidentally safe* if every computation that violates ψ has an informative bad prefix. For example, the formula $G(p \vee (Xq \wedge X \neg q))$ is accidentally safe. Finally, a property 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. While intentionally safe properties are natural, 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 $\mathcal{A}_{\neg\psi}^{true}$ on finite words, which is tight for ψ that is intentionally safe, and is fine for ψ that is accidentally safe.

We suggest a verification methodology that is based on the above observations. Given a system M and a safety formula ψ, we first construct the automaton $\mathcal{A}_{\neg\psi}^{true}$, regardless of the type of ψ. If the intersection of M and $A_{\neg\psi}^{true}$ is not empty, we get an error trace. Since $\mathcal{A}_{\neg\psi}^{true}$ runs on finite words, nonemptiness can be checked using forward reachability symbolic methods. If the product is empty, then, as $\mathcal{A}_{\neg\psi}^{true}$ is tight for intentionally safe formulas and is fine for 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. To distinguish between these two cases, we check whether ψ is pathologically safe. This check requires space polynomial in the length of ψ. If ψ is pathologically safe, we turn the user's attention to the fact that his specification is needlessly complicated. According to the user's preference, we then either construct an automaton tight for ψ, proceed with usual LTL verification, or wait for an alternative specification.

So far, we discussed safety properties in the linear paradigm. One can also define safety in the branching paradigm [6]. Then, a property, which describes trees, is a safety property if every tree that violates it has a finite prefix all whose extensions violate the property as well. We define safety in the branching paradigm and show that the problems of determining whether a CTL or a universal CTL formula is safety are EXPTIME-complete and PSPACE-complete, respectively. Given the linear complexity of CTL model checking, it is not clear yet whether safety is a helpful notion for the branching paradigm. On the other hand, we show that safety is a helpful notion for the assume-guarantee paradigm, where safety of either the assumption or the guarantee is sufficient to improve general verification methods.

2. Preliminaries

2.1. *Linear temporal logic*

The logic *LTL* is a linear temporal logic. Formulas of LTL are constructed from a set AP of atomic propositions using the usual Boolean operators and the temporal operators X ("next time"), U ("until"), and V ("duality of until"). Formally, given a set AP, an LTL formula in a positive normal form is defined as follows:

- **true**, **false**, p, or $\neg p$, for $p \in AP$.
- $\psi_1 \vee \psi_2$, $\psi_1 \wedge \psi_2$, $X\psi_1$, $\psi_1 U \psi_2$, or $\psi_1 V \psi_2$, where ψ_1 and ψ_2 are LTL formulas.

For an LTL formula ψ, we use $cl(\psi)$ to denote the closure of ψ, namely, the set of ψ's subformulas. We define the semantics of LTL with respect to a *computation* $\pi = \sigma_0, \sigma_1, \sigma_2, \ldots$, where for every $j \geq 0$, σ_j is a subset of AP, denoting the set of atomic propositions that hold in the j's 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 $p \in AP$, $\pi \models p$ iff $p \in \sigma_0$ and $\pi \models \neg p$ iff $p \notin \sigma_0$ and
- $\pi \models \psi_1 \vee \psi_2$ iff $\pi \models \psi_1$ or $\pi \models \psi_2$.
- $\pi \models \psi_1 \wedge \psi_2$ iff $\pi \models \psi_1$ and $\pi \models \psi_2$.
- $\pi \models X\psi_1$ iff $\pi^1 \models \psi_1$.
- $\pi \models \psi_1 U \psi_2$ iff there exists $k \geq 0$ such that $\pi^k \models \psi_2$ and $\pi^i \models \psi_1$ for all $0 \leq i < k$.
- $\pi \models \psi_1 V \psi_2$ iff for all $k \geq 0$, if $\pi^k \not\models \psi_2$, then there is $0 \leq i < k$ such that $\pi^i \models \psi_1$.

Often, we interpret linear temporal logic formulas over a *system* with many computations. Formally, a system is $M = \langle AP, W, R, W_0, L\rangle$ where W is the set of states, $R \subseteq W \times W$ is a total transition relation (that is, for every $w \in W$, there is at least one w' such that $R(w, w')$, the set 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. A computation of M is a sequence $w_0, w_1, \ldots$ such that $w_0 \in W_0$ and for all $i \geq 0$ we have $R(w_i, w_{i+1})$.

The *model-checking problem* for LTL is to determine, given an LTL formula ψ and a system M, whether all the computations of M satisfy ψ. The problem is known to be PSPACE-complete [39].

2.2. *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 $x \cdot y \notin L$. Thus, a bad prefix is a finite word that cannot be extended to an infinite word in L. Note that if x is a bad prefix, then all the finite extensions of x are also bad prefixes. We say that a bad prefix x is *minimal* iff all the strict prefixes of x are not bad. 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 bad prefix in X. Thus, while X need not contain all the bad prefixes for L, it must contain sufficiently many prefixes to "trap" all the words not in L. 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 [31] 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.

2.3. 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. For a given set X, let $\mathcal{B}^+(X)$ be the set of positive Boolean formulas over X (i.e., Boolean formulas built from elements in X using $\wedge$ and $\vee$), where we also allow the formulas **true** and **false**. For $Y \subseteq X$, we say that Y *satisfies* a formula $\theta \in \mathcal{B}^+(X)$ iff the truth assignment that assigns *true* to the members of Y and assigns *false* to the members of $X \backslash Y$ satisfies θ. For example, the sets $\{q_1, q_3\}$ and $\{q_2, q_3\}$ both satisfy the formula $(q_1 \vee q_2) \wedge q_3$, while the set $\{q_1, q_2\}$ does not satisfy this formula. The transition function $\rho : Q \times \Sigma \to 2^Q$ of a nondeterministic automaton with state space Q and alphabet Σ can be represented using $\mathcal{B}^+(Q)$. For example, a transition $\rho(q, \sigma) = \{q_1, q_2, q_3\}$ can be written as $\rho(q, \sigma) = q_1 \vee q_2 \vee q_3$. While transitions of nondeterministic automata correspond to disjunctions, transitions of *alternating automata* can be arbitrary formulas in $\mathcal{B}^+(Q)$. 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.

Formally, an alternating 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, $\delta : Q \times \Sigma \to \mathcal{B}^+(Q)$ is a transition function, $Q_0 \subseteq Q$ is a set of initial states, and $F \subseteq Q$ is a set of accepting states. While a run of a nondeterministic automaton over a word w can be viewed as a function $r : \mathbb{N} \to Q$, a run of an alternating automaton on w is a tree whose nodes are labeled by states in Q. Formally, a tree is a nonempty set $T \subseteq \mathbb{N}^*$, where for every $x \cdot c \in T$ with $x \in \mathbb{N}^*$ and $c \in \mathbb{N}$, we have $x \in T$. The elements of T are called *nodes*, and the empty word ε is the *root* of T. For every $x \in T$, the nodes $x \cdot c \in T$ where $c \in \mathbb{N}$ are the *children* of x. A node with no children is a *leaf*. A *path* π of a tree T is a set $\pi \subseteq T$ such that $\varepsilon \in \pi$ and for every $x \in \pi$, either x is a leaf, or there exists a unique $c \in \mathbb{N}$ such that $x \cdot c \in \pi$. Given a finite set Σ, a Σ-*labeled tree* is a pair $\langle T, V \rangle$ where T is a tree and $V : T \to \Sigma$ maps each node of T to a letter in Σ. A run of $\mathcal{A}$ on an infinite word $w = \sigma_1 \cdot \sigma_2 \cdots$ is a Q-labeled tree $\langle T_r, r \rangle$ with $T_r \subseteq \mathbb{N}^*$ such that $r(\varepsilon) \in Q_0$ and for every node $x \in T_r$ with $\delta(r(x), \sigma_{|x|+1}) = \theta$, there is a (possibly empty) set $S = \{q_1, \ldots, q_k\}$ such that S satisfies θ and for all $1 \leq c \leq k$, we have $x \cdot c \in T_r$ and $r(x \cdot c) = q_c$. For example, if $\delta(q_{in}, \sigma_0) = (q_1 \vee q_2) \wedge (q_3 \vee q_4)$, 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. Note that if for some y the function δ has the value **true**, then y need not have successors. Also, δ can never have the value **false** in a run. For a run r and an infinite path $\pi \subseteq T_r$, let $inf(r \,|\, \pi)$ denote the set of states that r visits infinitely often along π. That is, $inf(r \,|\, \pi) = \{q \in Q :$ for infinitely many $x \in \pi$, we have $r(x) = q\}$. As Q is finite, it is guaranteed that $inf(r \,|\, \pi) \neq \emptyset$. When $\mathcal{A}$ is a *Büchi* automaton on infinite words, the run r is *accepting* iff $inf(r \,|\, \pi) \cap F \neq \emptyset$ for all infinite path in T_r. That is, iff every path in the run visits at least one state in F infinitely often.

The automaton $\mathcal{A}$ can also run on finite words in Σ^*. A run of $\mathcal{A}$ on a finite word $w = \sigma_1 \cdots \sigma_n$ is a Q-labeled tree $\langle T_r, r\rangle$ with $T_r \subseteq \mathbb{N}^{\leq n}$, where $\mathbb{N}^{\leq n}$ is the set of all words of length at most n over the alphabet $\mathbb{N}$. The run proceeds exactly like a run on an infinite word, only that all the nodes of level n in T_r are leaves. A run $\langle T_r, r\rangle$ is accepting iff all its nodes of level n visit accepting states. Thus, if for all nodes $x \in T_r \cap \mathbb{N}^n$, we have $r(x) \in F$.

A word (either finite or infinite) is accepted by $\mathcal{A}$ iff there exists an accepting run on it. Note that while conjunctions in the transition function of $\mathcal{A}$ are reflected in branches of $\langle T_r, r\rangle$, disjunctions are reflected in the fact we can have many runs on the same word. The language of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts. As we already mentioned, 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 $\delta(q, \sigma) \in Q \cup \{\mathbf{false}\}$, and it is nondeterministic if $\delta(q, \sigma)$ is always a disjunction over Q.

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* if $\mathcal{L}(\mathcal{A}) = \emptyset$. A state $q \in Q$ is *nonempty* if $\mathcal{L}(\mathcal{A}^{\{q\}}) \neq \emptyset$. 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 [34, 48].

We can now state the basic result concerning the analysis of safety, which generalizes Sistla's result [40] concerning safety of LTL formulas.

Theorem 2.1. *Checking whether an alternating Büchi automaton is a safety* (*or a co-safety*) *automaton is PSPACE-complete.*

Proof: Let $\mathcal{A}$ be a given alternating Büchi automaton. There is an equivalent nondeterministic Büchi automaton $\mathcal{N}$, whose size is at most exponential in the size of $\mathcal{A}$ [30]. We assume that each state in $\mathcal{N}$ accepts at least one word (otherwise, we can remove the state and simplify the transitions relation). Let $\mathcal{N}^{loop}$ be the automaton obtained from $\mathcal{N}$ by taking all states as accepting states. As shown in [3, 40], $\mathcal{A}$ is a safety automaton iff $\mathcal{L}(\mathcal{N}^{loop})$ is contained in $\mathcal{L}(\mathcal{A})$. In order to check the latter, we first construct from $\mathcal{A}$ a nondeterministic automaton $\tilde{\mathcal{N}}$ such that $\mathcal{L}(\tilde{\mathcal{N}}) = comp(\mathcal{A})$. To construct $\tilde{\mathcal{N}}$, we first complement $\mathcal{A}$ with a quadratic blow-up [21], and then translate the result (which is an alternating Büchi automaton) to a nondeterministic Büchi automaton, which involves an exponential blow up [30]. Thus, the size of $\tilde{\mathcal{N}}$ is at most exponential in the size of $\mathcal{A}$. Now, $\mathcal{L}(\mathcal{N}^{loop})$ is contained in $\mathcal{L}(\mathcal{A})$ iff the intersection $\mathcal{L}(\mathcal{N}^{loop}) \cap \mathcal{L}(\tilde{N})$ is empty. Since the constructions described above can be performed on the fly, the emptiness of the intersection can be checked in space polynomial in the size of $\mathcal{A}$. The claim for co-safety follows since, as noted, alternating Büchi automata can be complemented with a quadratic blow-up [21]. The lower bound follows from Sistla's lower bound for LTL [40], since LTL formulas can be translated to alternating Büchi automata with a linear blow-up (see Theorem 2.2). □

We note that the nonemptiness tests required by the algorithm can be performed using model checking tools (cf. [12, 42]).

2.4. *Automata and temporal logic*

Given an LTL formula ψ in positive normal form, one can build a nondeterministic Büchi automaton $\mathcal{A}_\psi$ such that $\mathcal{L}(\mathcal{A}_\psi) = \|\psi\|$ [47]. The size of $\mathcal{A}_\psi$ is exponential in $|\psi|$. It is shown in [23, 44] that when alternating automata are used, the translation of ψ to $\mathcal{A}_\psi$ as above involves only a linear blow up[1] The translation of LTL formulas to alternating Büchi automata is going to be useful also for our methodology, and we describe it below.

Theorem 2.2 [23, 44]**.** *Given an LTL formula ψ, we can construct, in linear running time, an alternating Büchi automaton $\mathcal{A}_\psi = \langle 2^{AP}, cl(\psi), \delta, \{\psi\}, F\rangle$, such that $\mathcal{L}(\mathcal{A}_\psi) = \|\psi\|$.*

Proof: The set F of accepting states consists of all the formulas of the form $\varphi_1 V \varphi_2$ in $cl(\psi)$. It remains to define the transition function δ. For all $\sigma \in 2^{AP}$, we define:

- $\delta(\mathbf{true}, \sigma) = \mathbf{true}$.
- $\delta(p, \sigma) = \mathbf{true}$ if $p \in \sigma$.
- $\delta(\neg p, \sigma) = \mathbf{true}$ if $p \notin \sigma$.
- $\delta(\mathbf{false}, \sigma) = \mathbf{false}$.
- $\delta(p, \sigma) = \mathbf{false}$ if $p \notin \sigma$.
- $\delta(\neg p, \sigma) = \mathbf{false}$ if $p \in \sigma$.
- $\delta(\varphi_1 \wedge \varphi_2, \sigma) = \delta(\varphi_1, \sigma) \wedge \delta(\varphi_2, \sigma)$.
- $\delta(\varphi_1 \vee \varphi_2, \sigma) = \delta(\varphi_1, \sigma) \vee \delta(\varphi_2, \sigma)$.
- $\delta(X\varphi, \sigma) = \varphi$.
- $\delta(\varphi_1 U \varphi_2, \sigma) = \delta(\varphi_2, \sigma) \vee (\delta(\varphi_1, \sigma) \wedge \varphi_1 U \varphi_2)$.
- $\delta(\varphi_1 V \varphi_2, \sigma) = \delta(\varphi_2, \sigma) \wedge (\delta(\varphi_1, \sigma) \vee \varphi_1 V \varphi_2)$. □

Using the translation described in [30] from alternating Büchi automata to nondeterministic Büchi automata, we get:

Corollary 2.3 [47]**.** *Given an LTL formula ψ, we can construct, in exponential running time, a nondeterministic Büchi automaton $\mathcal{N}_\psi$ such that $\mathcal{L}(\mathcal{N}_\psi) = \|\psi\|$.*

Combining Corollary 2.3 with Theorem 2.1, we get the following algorithm for checking safety of an LTL formula ψ. The algorithm is essenttially as described in [40], rephrased somewhat to emphasize the usage of model checking.

1. Construct the nondeterministic Büchi automaton $\mathcal{N}_\psi = \langle 2^{AP}, Q, \delta, Q_0, F\rangle$.
2. Use a model checker to compute the set of nonempty states, eliminate all other states, and take all remaining states as accepting states. Let $\mathcal{N}_\psi^{loop} = \langle 2^{AP}, S, \delta, S_0, S\rangle$ be the resulting automaton. By Theorem 2.1, ψ is a safety formula iff $\|\mathcal{N}_\psi^{loop}\| \subseteq \|\mathcal{N}_\psi\|$; thus all the computations accepted by $\mathcal{N}_\psi^{loop}$ satisfy ψ.
3. Convert $\mathcal{N}_\psi^{loop}$ into a system $M_\psi^{loop} = \langle AP, S \times 2^{AP}, R, S_0 \times 2^{AP}, L\rangle$, where the transition relation $R = \{\langle (s, \sigma), (s', \sigma')\rangle : s' \in \delta(s, \sigma)\}$, and the labeling function is such that $L(s, \sigma) = \sigma$. Thus, the system M_ψ^{loop} has exactly all the computations accepted by $\mathcal{N}_\psi^{loop}$.
4. Use a model checker to verify that $M_\psi^{loop} \models \psi$.

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 [16, 47]. In practice, when given a property that happens to be safe, what we want is an 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 $\mathit{pref}(\mathcal{A})$ 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 3.1 below shows that when $\mathcal{A}$ is a nondeterministic automaton, things are not that simple. While we can avoid a difficult determinization of $\mathcal{A}$ (which may also require an acceptance condition that is stronger than Büchi) [38], we cannot avoid an exponential blow-up.

Theorem 3.1. *Given a safety nondeterministic Büchi automaton $\mathcal{A}$ of size n, the size of an automaton that recognizes $\mathit{pref}(\mathcal{A})$ is $2^{\Theta(n)}$.*

Proof: We start with the upper bound. Let $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, F \rangle$. Recall that $\mathit{pref}(\mathcal{L}(\mathcal{A}))$ 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 $\mathit{pref}(\mathcal{A})$ 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 δ' and F' are as follows.

- The transition function δ' follows the subset construction induced by δ; that is, for every $S \in 2^Q$ and $\sigma \in \Sigma$, we have $\delta'(S, \sigma) = \bigvee_{s \in S} \delta(s, \sigma)$.
- The set of accepting states contains all the rejecting sets of $\mathcal{A}$.

We now turn to the lower bound. Essentially, it follows from the fact that $\mathit{pref}(\mathcal{A})$ 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 [29]. 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 $\mathit{comp}(\mathcal{A})$ has $2^{\Theta(|Q|)}$ states. (To see this, consider the language L_n consisting of 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. We claim that $\mathit{pref}(\mathcal{A}') = \mathit{comp}(\mathcal{A})$. To see this, note that since all the states in $\mathcal{A}$ are accepting, a word w is rejected by $\mathcal{A}$ iff all the runs of $\mathcal{A}$ on w get stuck while reading it, which, as all the states in $\mathcal{A}'$ are accepting, holds iff w is in $\mathit{pref}(\mathcal{A}')$. □

We note that while constructing the deterministic automaton $\mathcal{A}'$, one can apply to it minimization techniques as used in the verification tool Mona [20]. The lower bound in Theorem 3.1 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 3.2 and 3.3 we refute this hope.

Theorem 3.2. *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 3.1, 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 3.2 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 3.2 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 [22]. In order to prove the latter, [22] define the language $L_n \subseteq \{0, 1, \#, \&\}^*$ by

$$L_n = \{\{0, 1, \#\}^* \cdot \# \cdot w \cdot \# \cdot \{0, 1, \#\}^* \cdot \& \cdot w \ : w \in \{0, 1\}^n\}.$$

A word w is in L_n iff the suffix of length n that comes after the single & in w appears somewhere before the &. By [11], the smallest deterministic automaton on finite words that accepts L_n has at least 2^{2^n} states (reaching the &, the automaton should remember the possible set of words in $\{0, 1\}^n$ that have appeared before). On the other hand, we can specify L_n with the following LTL formula of length quadratic in n (we ignore here the technical fact that Büchi automata and LTL formulas describe infinite words).

$$[(\neg\&)U(\& \wedge XG\neg\&)] \wedge F\left[\# \wedge X^{n+1}\# \wedge \bigwedge_{1 \leq i \leq n} ((X^i 0 \wedge G(\& \rightarrow X^i 0)) \vee (X^i 1 \wedge G(\& \rightarrow X^i 1)))\right].$$

Theorem 3.3. *Given a safety LTL formula ψ of size n, the size of an automaton for pref(ψ) is $2^{2^{O(n)}}$ and $2^{2^{\Omega(\sqrt{n})}}$.*

Proof: The upper bounds follows from the exponential translation of LTL formulas to nondeterministic Büchi automata [47] and the exponential upper bound in Theorem 3.1. For the lower bound, we define, for $n \geq 1$, the language L'_n of infinite words over $\{0, 1, \#, \&\}$ where every word in L'_n contains at least one &, and after the first & either there is a word in $\{0, 1\}^n$ that has appeared before, or there is no word in $\{0, 1\}^n$ (that is, there is at least one # or & in the first n positions after the first &). The language L'_n is a co-safety language. As in the proof of Theorem 3.2, a prefix of the form $x\&$ such that $x \in \{0, 1, \#\}^*$ contains all the words in $\{0, 1\}^n$ is a good prefix, and a nondeterministic automaton needs 2^{2^n} states to detect such good prefixes. This makes the automaton for *co-pref*(L'_n) doubly exponential. On the other hand, we can specify L'_n with an LTL formula ψ_n that is quadratic in n. The formula is similar to the one for L_n, only that it is satisfied also by computations in which the first & is not followed by a word in $\{0, 1\}^n$. Now, the LTL formula $\neg\psi_n$ is a safety formula of size quadratic in n and the size of the smallest nondeterministic Büchi automaton for *pref*$(\neg\psi)$ is 2^{2^n}. □

In order to get the upper bound in Theorem 3.3, we applied the exponential construction in Theorem 3.1 to the exponential Büchi automaton $\mathcal{A}_\psi$ for $\|\psi\|$. The construction in Theorem 3.1 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 the following lemma.

Lemma 3.4. *Consider an LTL formula ψ and its nondeterministic Büchi automaton $\mathcal{A}_\psi$. Let Q be a set of states of $\mathcal{A}_\psi$. The universality problem for Q can be checked using space polynomial in $|\psi|$.*

Proof: Every state in $\mathcal{A}_\psi$ is associated with a set η of subformulas of ψ. A set Q of states of $\mathcal{A}_\psi$ then corresponds to a set $\{\eta_1, \eta_2, \ldots, \eta_k\}$ of sets of subformulas. Let $\eta_i = \{\varphi^i_1, \ldots, \varphi^i_{l_i}\}$. The set Q is universal iff the LTL formula $\psi_Q = \bigvee_{1\leq i\leq k} \bigwedge_{1\leq j\leq l_i} \varphi^i_j$ is valid. Though the formula ψ_Q may be exponentially longer than ψ, we can check its validity in PSPACE. To do this, we first negate it and get the formula $\neg\psi_Q = \bigwedge_{1\leq i\leq k} \bigvee_{1\leq j\leq l_i} \neg\varphi^i_j$. Clearly, ψ_Q is valid iff $\neg\psi_Q$ is not satisfiable. But $\neg\psi_Q$ is satisfiable iff at least one conjunction $\bigwedge_{1\leq i\leq k} \neg\varphi^i_{j_i}$ in the disjunctive normal form of $\neg\psi_Q$ is satisfiable, where $1 \leq j_i \leq l_i$. Thus, to check if $\neg\psi_Q$ is satisfiable we have to enumerate all such conjunctions and check whether one is satisfiable. Since each such conjunction is of polynomial size, as the number of literals is bounded by $|\psi|$, the claim follows. □

Note that the satisfiability problem for LTL (and thus for all the $\neg\varphi^i_{j_i}$'s) can be reduced to the nonemptiness problem for nondeterministic Büchi automata, and thus also to model checking [12, 42]. In fact, the nondeterministic Büchi automaton $\mathcal{A}_\psi$ constructed in [47] contains all sets of subformulas of ψ as states. To run the universality test it suffices to compute the set of states of $\mathcal{A}_\psi$ accepting some infinite word. Then a conjunction $\bigwedge_{1 \le i \le k} \neg\varphi^i_{j_i}$ is satisfiable iff the set $\{\neg\varphi^1_{j_1}, \ldots, \neg\varphi^k_{j_k}\}$ is contained in such a state.

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 [5, 19]. 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 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 $M = \langle AP, W, R, W_0, L\rangle$. 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 3.2, 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 [37, 49], including standard simulation techniques [26]. Also, one could use *bounded model-checking* techniques, in which a reduction to the propositional satisfiability problem is used, to check whether there is a path of bounded length from an initial state to an accepting state in $\mathit{fin}(M) \times \mathcal{A}'$ [4]. 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. It is conceivable, however, that due to the determinism of $\mathcal{A}'$, such a BDD would have in practice a not too large width, and therefore would be of a manageable size (see Section 6.2 for a related discussion.)

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 *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 $S_{i+1} = post(S_i)$. One could therefore say that this construction implements the subset construction dynamically, "on the fly", during model checking, rather than statically, before model checking. 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 $|\psi|$. More experimental work is needed to compare the merit of the two approaches (i.e., static vs. dynamic subset construction).

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. By Lemma 3.4, 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 $|\psi|$. (Repeatedly, select a state $w \in W$, analyze Q_i^w, and then remove all states $u \in W$ such that $Q_i^u = Q_i^w$.) By Lemma 4.1, encountering a universal Q_i^w solves the model-checking problem without further traversal of the system.

Lemma 4.1. *If ψ is a safety formula, then $M \times \mathcal{A}_{\neg\psi}$ is nonempty iff Q_i^w is universal for some $w \in W$ and $i > 0$.*

Proof: Suppose Q_i^w is universal. Consider any infinite trace y that starts in w. Since Q_i^w is universal, there is some state $q \in Q_i^w$ such that y is accepted by $\mathcal{A}^q_{\neg\psi}$. In addition, by the definition of S_i, there is a finite trace x from some state in W_0 to w such that, reading x, the automaton $\mathcal{A}_{\neg\psi}$ reaches q. Hence, the infinite trace $x \cdot y$ does not satisfy ψ.

Suppose now that $M \times \mathcal{A}_{\neg\psi}$ is nonempty. Then there is an infinite trace y of M that is accepted by $\mathcal{A}_{\neg\psi}$. As ψ is a safety formula, y has a bad prefix x of length i such that $\delta(Q_0, x)$ is universal. If x ends in the state w, then Q_i^w is universal. □

Let j be the length of the shortest bad prefix for ψ that exists in M. The direction from left to right in Lemma 4.1 can be strengthened, as the nonemptiness of $M \times \mathcal{A}_{\neg\psi}$ implies that for every $i \geq j$, there is $w \in W$ for which Q_w^i is universal. While we do not want to (and sometime we cannot) calculate j in advance, the stronger version gives more information on when and why the method we discuss above is not only sound but also complete.

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) [37, 49]. We have to ensure, however, that S'_{i+1} is *saturated* with respect to states of $\mathcal{A}_{\neg\psi}$ [26]. 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 $\langle w, q'\rangle \in S'_{i+1}$ (in other words, if some pair in which the M-state element is w stays in the subset S'_{i+1} of S_{i+1}, then all the pairs in which the M-state element is w should stay). This ensures that if the semi-exhaustive analysis follows a bad prefix of length i in M, then $Q_i'^w = \{q : \langle w, q\rangle \in S'_i\}$ is universal. In the extreme case, we follow only one trace of M, i.e., we simulate M. In that case, we have that $S'_{i+1} = \{w\} \times Q_i'^w$. For related approaches see [1, 10]. Note that while such a simulation cannot in general be performed for ψ that is not a safety formula, we can use it as a heuristic also for general formulas. We will get back to this point in Remarks 5.3 and 6.1.

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*. Recall that 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).

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 $L : \{1, \ldots, n+1\} \to 2^{cl(\neg\psi)}$ such that the following hold:

1. $\neg\psi \in L(1)$.
2. $L(n+1)$ is empty.
3. For all $1 \le i \le n$ 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 $\varphi_1 V \varphi_2 \in L(i+1)]$.

If π is informative for ψ, the existing mapping L is called the *witness* for $\neg\psi$ in π. 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 (e.g., with a witness L for which $L(1) = \{F\neg p\}$, $L(2) = \{F\neg p, \neg p\}$, 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.

Theorem 5.1. *Given an LTL formula ψ and a finite computation π of length n, the problem of deciding whether π is informative for ψ can be solved in time $O(n \cdot |\psi|)$.*

Proof: Intuitively, since π has no branches, deciding whether π is informative for ψ can proceed similarly to CTL model checking. Given ψ and $\pi = \sigma_1 \cdot \sigma_2 \cdots \sigma_n$, we construct a mapping $L_{\max} : \{1, \ldots, n\} \to 2^{cl(\neg\psi)}$ such that $L_{\max}(i)$ contains exactly all the formulas $\neg\varphi \in cl(\neg\psi)$ such that the suffix $\sigma_i, \ldots, \sigma_n$ is informative for φ. Then, π is informative for ψ iff $\neg\psi \in L_{\max}(1)$. The construction of $L_{\max}$ proceeds in a bottom-up manner. Initially $L_{\max}(i) = \emptyset$ for all $1 \leq i \leq n$. Then, for each $1 \leq i \leq n$, we insert to $L_{\max}(i)$ all the propositional assertions in $cl(\neg\psi)$ that are satisfied by σ_i. Then, we proceed by induction on the structure of the formula, inserting a subformula φ to $L_{\max}(i)$ iff the conditions from item **(3)** above are satisfied for it, taking $L_{\max}(n+1) = \emptyset$. In order to cope with the circular dependency in the conditions for φ of the forms $\varphi_1 U \varphi_2$ and $\varphi_1 V \varphi_2$, insertion of formulas proceeds from $L_{\max}(n)$ to $L_{\max}(1)$. Thus, for example, the formula $\varphi_1 \vee \varphi_2$ is added to $L_{\max}(i)$ iff $\varphi_1 \in L_{\max}(i)$ or $\varphi_2 \in L_{\max}(i)$, the formula $X\varphi_1$ is added to $L_{\max}(i)$ iff $\varphi_1 \in L_{\max}(i+1)$ (thus, $L_{\max}(n)$ contains no formulas of the form $X\varphi_1$), and the formula $\varphi_1 U \varphi_2$ is added to $L_{\max}(i)$ iff $\varphi_2 \in L_{\max}(i)$ or both $\varphi_1 \in L_{\max}(i)$ and $\varphi_1 U \varphi_2 \in L_{\max}(i+1)$ (recall that we insert $\varphi_1 U \varphi_2$ to $L_{\max}(i+1)$, if appropriate, before we examine the insertion of $\varphi_1 U \varphi_2$ to $L_{\max}(i)$). We have at most $|\psi|$ subformulas to examine, each of which requires time linear in n, thus the overall complexity is $O(n \cdot |\psi|)$. □

Remark 5.2. A very similar argument shows that one can check in linear running time whether an infinite computation π, represented as a prefix followed by a cycle, satisfies an LTL formula ψ.

Remark 5.3. Clearly, if an infinite computation ρ has a prefix π informative for ψ, then ρ does not satisfy ψ. On the other hand, it may be that ρ does not satisfy ψ and all the prefixes of ρ up to a certain length (say, the length where the BDDs described in Section 4 explode) are not informative. Hence, in practice, one may want to apply the check in Theorem 5.1 to both ψ and $\neg\psi$. Then, one would get one of the following answers: fail (a prefix that is informative for ψ exists, hence ρ does not satisfy ψ), pass (a prefix that is informative for $\neg\psi$ exists, hence ρ satisfies ψ), and undetermined (neither prefixes were found). Note that the above methodology is independent of ψ being a safety property.

We now use the notion of informative prefix in order to 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 $[G(q \vee FGp) \wedge G(r \vee FG\neg p)] \vee Gq \vee Gr$

Sistla has shown that all temporal formulas in positive normal form constructed with the temporal connectives X and V are safety formulas [40]. We call such formulas *syntactically safe*. The following strengthens Sistla's result.

Theorem 5.4. *If ψ is syntactically safe, then ψ is intentionally or accidentally safe.*

Proof: Let ψ be a syntactically safe formula. Then, the only temporal operators in $\neg\psi$ are X and U. Consider a computation $\pi = \sigma_1 \cdot \sigma_2 \cdots$ that satisfies $\neg\psi$. Since π satisfies $\neg\psi$, then, by the semantics of LTL, there is a mapping $L : \mathbb{N} \to 2^{cl(\neg\psi)}$ such that conditions (1) and (3) for a witness mapping hold for L (with $n = \infty$), and there is $i \in \mathbb{N}$ such that $L(i + 1)$ is empty. The prefix $\sigma_1 \cdots \sigma_i$ of π is then informative for ψ. It follows that every computation that violates ψ has a prefix informative for ψ, thus ψ is intentionally or accidentally safe. □

As described in Section 2.4, given an LTL formula ψ in positive normal form, one can build an alternating Büchi automaton $\mathcal{A}_\psi = \langle 2^{AP}, 2^{cl(\psi)}, \delta, \psi, F\rangle$ 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. We define the alternating Büchi automaton $\mathcal{A}_\psi^{true} = \langle 2^{AP}, 2^{cl(\psi)}, \delta, \psi, \emptyset\rangle$ 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 $\mathcal{A}_\psi^{true}$ all copies must reach a state from which they proceed to **true**. Accordingly, $\mathcal{A}_\psi^{true}$ 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*$(\mathcal{A}_\psi^{true})$ be $\mathcal{A}_\psi^{true}$ when regarded as an automaton on finite words.

Theorem 5.5. *For every safety formula ψ, the automaton fin$(\mathcal{A}_{\neg\psi}^{true})$ accepts exactly all the prefixes that are informative for ψ.*

Proof: Assume first that $\pi = \sigma_1 \cdots \sigma_n$ is a prefix informative for ψ in π. Then, there is a witness mapping $L: \{1, \ldots, n\} \to 2^{cl(\neg\psi)}$ for $\neg\psi$ in π. The witness L induces a run r of *fin*$(\mathcal{A}_{\neg\psi}^{true})$. Formally, the set of states that r visits when it reads the suffix π^i of π coincides with $L(i)$. By the definition of a witness mapping, all the states q that r visits when it reads π^n satisfy $\delta(q, \sigma_n) = $ **true**. therefore, r is accepting.

The other direction is similar, thus every accepting run of *fin*$(\mathcal{A}_{\neg\psi}^{true})$ on π induces a witness for $\neg\psi$ in π. □

Corollary 5.6. *Consider a safety formula ψ.*

1. *If ψ is intentionally safe, then fin$(\mathcal{A}_{\neg\psi}^{true})$ is tight for ψ.*
2. *If ψ is accidentally safe, then fin$(\mathcal{A}_{\neg\psi}^{true})$ is fine for ψ.*

Theorem 5.7. *Deciding whether a given formula is pathologically safe is PSPACE-complete.*

Proof: Consider a formula ψ. Recall that the automaton $\mathcal{A}_{\psi}^{true}$ 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 $\mathcal{A}_{\neg\psi}^{true}$. Accordingly, checking whether ψ is pathologically safe can be reduced to checking the containment of $\mathcal{L}(\mathcal{A}_{\neg\psi})$ in $\mathcal{L}(\mathcal{A}_{\neg\psi}^{true})$. 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 [21], 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 $\varphi = [G(q \vee FGp) \wedge G(r \vee FG\neg p)] \vee Gq \vee Gr$ 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 5.7 implies that the reverse direction of Theorem 5.4 does not hold.

Theorem 5.8. *Deciding whether a given formula is intentionally safe is in EXPSPACE.*

Proof: Consider a formula ψ of size n. By Theorem 3.3, we can construct an automaton of size $2^{2^{O(n)}}$ for $pref(\psi)$. By Theorem 5.5, $fin(\mathcal{A}_{\neg\psi}^{true})$ accepts all the prefixes that are informative for ψ. Note that ψ is intentionally safe iff every prefix in $pref(\psi)$ is an informative prefix for ψ. Thus, to check that ψ is intentionally safe, one has to complement $fin(\mathcal{A}_{\neg\psi}^{true})$ and check that its intersection with the automaton for $pref(\psi)$ is empty. A nondeterministic automaton that complements $fin(\mathcal{A}_{\neg\psi}^{true})$ is exponential in n [30], and its product with the automaton for $pref(\psi)$ is doubly exponential in n. Since emptiness can be checked in nondeterministic logarithmic space, the claim follows. □

6. A Methodology

6.1. Exploiting the classification

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 $fin(\mathcal{A}_{\neg\psi}^{true})$, 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 $fin(M)$ be a nondeterministic automaton on finite words that accepts the prefixes of computations of M, and let $\mathcal{U}_{\neg\psi}^{true}$ be the nondeterministic automaton on finite words equivalent to the alternating automaton $fin(\mathcal{A}_{\neg\psi}^{true})$ [11]. The size of $\mathcal{U}_{\neg\psi}^{true}$ is exponential in the size of $fin(\mathcal{A}_{\neg\psi}^{true})$, hence it is exponential in the length of ψ. Given M and ψ, we suggest to proceed as follows (see figure 1).

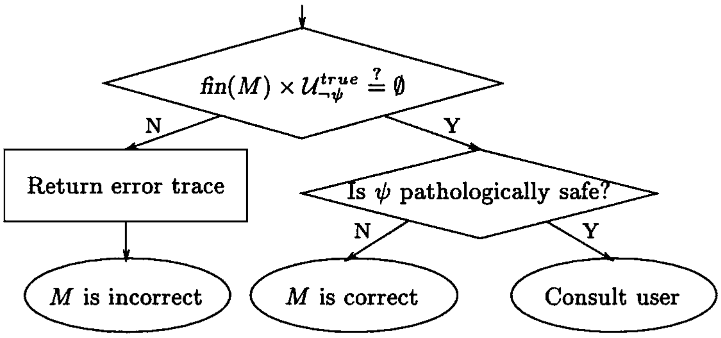
$\mathit{fin}(M) \times \mathcal{U}^{true}_{\neg\psi} \overset{?}{=} \emptyset$
N
Y
Return error trace
Is ψ pathologically safe?
N
Y
M is incorrect
M is correct
Consult user

Figure 1. Verification of safety formulas.

Instead of checking the emptiness of $M \times \mathcal{A}_{\neg\psi}$, verification starts by checking *fin*(M) with respect to $\mathcal{U}^{true}_{\neg\psi}$. Since both automata refer to finite words, this can be done using finite forward reachabilityanalysis.[2] If the product *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[3]. If the product *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 5.4.) If ψ 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[4]). 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 [18, 42].) 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 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.

Remark 6.1. In fact, our methodology can be adjusted to formulas ψ that are not (or not known to be) safety formulas, and it can often terminate with a helpful output for such formulas. As with safety formulas, we start by checking the emptiness of *fin*$(M) \times \mathcal{U}^{true}_{\neg\psi}$ (note that $\mathcal{U}^{true}_{\neg\psi}$ is defined also for formulas that are not safety). If the intersection is not empty, it contains an error trace, and M is incorrect. If the intersection is empty, we check whether ψ is an intentionally or accidentally safe formula. If it is, we conclude that M is correct. Otherwise, we consult the user. Note also that by determinizing $\mathcal{U}^{true}_{\neg\psi}$, we can get a checker that can be used in simulation [1].

6.2. *On going backwards*

As detailed above, given a system M and a safety formula ψ, our method starts by checking whether there is finite prefix of a computation of M (a word accepted by $\mathit{fin}(M)$) that is an informative bad prefix for ψ (a word accepted by $\mathcal{U}^{\mathit{true}}_{\neg\psi}$). Since both $\mathit{fin}(M)$ and $\mathcal{U}^{\mathit{true}}_{\neg\psi}$ are automata on finite words, so is their product, thus a search for a word in their intersection can be done using finite forward reachability analysis. In this section we discuss another approach for checking that no finite prefix of a computation of M is an informative bad prefix.

We say that a nondeterministic automaton $\mathcal{U} = \langle \Sigma, Q, \rho, Q_0, F \rangle$ is *reverse deterministic* if for every state $q \in Q$ and letter $\sigma \in \Sigma$, there is at most one state $q' \in Q$ such that $q \in \rho(q', \sigma)$. Thus, given the current state of a run of $\mathcal{U}$ and the last letter read from the input, one can determine the state visited before the current one. Let $\rho^{-1} : Q \times \Sigma \to 2^Q$ denote the reverse function of ρ, thus $\rho^{-1}(q, \sigma) = \{q' : q \in \rho(q', \sigma)\}$. By the above, when $\mathcal{U}$ is reverse deterministic, all the sets in the range of ρ^{-1} are either empty or singletons. We extend ρ^{-1} to sets in the natural way as follows. For a set $Q' \subseteq Q$, we define $\rho^{-1}(Q', \sigma) = \{q' : Q' \cap \rho(q', \sigma) \neq \emptyset\}$; thus, $\rho^{-1}(Q', \sigma)$ contains all the states that may lead to some state in Q' when the letter in the input is σ.

Assume that we have a reverse deterministic fine automaton $\mathcal{U}_{\neg\psi}$ for ψ; thus $\mathcal{U}_{\neg\psi}$ accepts exactly all the bad prefixes informative for ψ. Consider the product $P = \mathit{fin}(M) \times \mathcal{U}_{\neg\psi}$. Let $P = \langle 2^{AP}, S, \rho, S_0, F \rangle$. Recall that M has a finite prefix of a computation that is an informative bad prefix iff P is nonempty; namely, there is a path in P from some state in S_0 to some state in F. Each state in P is a pair $\langle w, q \rangle$ of a state w of M and a state q of $\mathcal{U}_{\neg\psi}$. We say that a set $S' \subseteq S$ of states of P is Q-homogeneous if there is some $q \in Q$ such that $S' \subseteq W \times \{q\}$, where W is the state of states of M; that is, all the pairs in S' agree on their second element. For every state $\langle w, q \rangle$ and letter $\sigma \in 2^{AP}$, the set $\rho^{-1}(\langle w, q \rangle, \sigma)$ may contain more than one state. Nevertheless, since $\mathcal{U}_{\neg\psi}$ is reverse deterministic, the set $\rho^{-1}(\langle w, q \rangle, \sigma)$ is Q-homogeneous. Moreover, since $\mathcal{U}_{\neg\psi}$ is reverse deterministic, then for every Q-homogeneous set $S' \subseteq S$ and for every $\sigma \in 2^{AP}$, the set $\rho^{-1}(S', \sigma)$ is Q-homogeneous as well. Accordingly, if we start with some Q-homogeneous set and traverse P backwards along one word in Σ^*, we need to maintain only Q-homogeneous sets. In practice, it means that instead of maintaining sets in $2^W \times 2^Q$ (which is what we need to do in a forward traversal), we only have to maintain sets in $2^W \times Q$. If we conduct a backwards breadth-first search starting from F, we could hope that the sets maintained during the search would be smaller, though not necessarily homogeneous, due to the reverse determinism. The above suggests that when the fine automaton for ψ is reverse deterministic, it may be useful to check the nonemptiness of P using a backwards search, starting from the fine automaton's accepting states.

The automaton $\mathcal{U}^{\mathit{true}}_{\neg\psi}$ is defined by means of the alternating word automaton $\mathcal{A}_{\neg\psi}$, and is not reverse deterministic. For example, if $\neg\psi = X(Xp \vee Xq) \vee X(Xq \vee Xr)$ then $\mathcal{U}^{\mathit{true}}_{\neg\psi}$ can reach the state q from both states $Xp \vee Xq$ and $Xq \vee Xr$. Below we describe a fine reverse deterministic automaton $\mathcal{N}_{\neg\psi}$ for ψ, of size exponential in the length of ψ. The automaton is based on the reverse deterministic automata defined in [47] for LTL. As in [47], each state of the automaton $\mathcal{N}_{\neg\psi}$ is associated with a set S of formulas in $cl(\neg\psi)$.

When the automaton is in a state associated with S, it accepts exactly all infinite words that satisfy all the formulas in S. Unlike the automata in [47], a state of $\mathcal{N}_{\neg\psi}$ that is associated with S imposes only requirements on the formulas (these in S) that should be satisfied, and imposes no requirements on formulas (these in $cl(\neg\psi) \setminus S$) that should not be satisfied. This property is crucial for $\mathcal{N}_{\neg\psi}$ being fine. When the automaton $\mathcal{N}_{\neg\psi}$ visits a state associated with the empty set, it has no requirements. Accordingly, we define $\{\emptyset\}$ to be the set of accepting states (note that the fact that the set of accepting states is a singleton implies that in the product P we can start with the single Q-homogeneous set $W \times \{\emptyset\}$).

It is easy to define $\mathcal{N}_{\neg\psi}$ formally in terms of its reverse deterministic function ρ^{-1}. Consider a state $S \subseteq cl(\neg\psi)$ of $\mathcal{N}_{\neg\psi}$ and a letter $\sigma \in 2^{AP}$. The single state S' in $\rho^{-1}(S, \sigma)$ is the maximal subset of $cl(\neg\psi)$ such that if a computation π satisfies all the formulas in S' and its first position is labeled by σ, then its suffix π^1 satisfies all the formulas in S. Formally, S' contains exactly all the propositional assertion in $cl(\neg\psi)$ that are satisfied by σ, and for all formulas φ in $cl(\neg\psi)$ for which the following hold.

- If $\varphi = \varphi_1 \vee \varphi_2$, then $\varphi \in S'$ iff $\varphi_1 \in S'$ or $\varphi_2 \in S'$.
- If $\varphi = \varphi_1 \wedge \varphi_2$, then $\varphi \in S'$ iff $\varphi_1 \in S'$ and $\varphi_2 \in S'$.
- If $\varphi = X\varphi_1$, then $\varphi \in S'$ iff $\varphi_1 \in S$.
- If $\varphi = \varphi_1 U \varphi_2$, then $\varphi \in S'$ iff $\varphi_2 \in S'$ or $[\varphi_1 \in S'$ and $\varphi_1 U \varphi_2 \in S]$.
- If $\varphi = \varphi_1 V \varphi_2$, then $\varphi \in S'$ iff $\varphi_2 \in S'$ and $[\varphi_1 \in S'$ or $\varphi_1 V \varphi_2 \in S]$.

It is easy to see that for every S and σ, there is a single S' that satisfies the above conditions. Also, a sequence of states in $\mathcal{N}_{\neg\psi}$ that starts with some S_0 containing $\neg\psi$ and leads via the finite computation $\pi = \sigma_1 \cdot \sigma_2 \cdots \sigma_n$ to $S_n = \emptyset$ induces a mapping showing that π is informative for ψ. It follows that $\mathcal{N}_{\neg\psi}$ is a reverse deterministic fine automaton for ψ.

6.3. *Safety in the assume-guarantee paradigm*

Given a system M and two LTL formulas φ_1 and φ_2, the linear *assume-guarantee* specification $\langle\varphi_1\rangle M \langle\varphi_2\rangle$ holds iff for every system M' such that the composition $M \parallel M'$ satisfies (the assumption) φ_1, the composition $M \parallel M'$ also satisfies (the guarantee) φ_2. Testing assume-guarantee specifications as above can be reduced to LTL model checking. Indeed, it is not hard to see that $\langle\varphi_1\rangle M \langle\varphi_2\rangle$ iff M satisfies the LTL formula $\psi = \varphi_1 \rightarrow \varphi_2$. A system M satisfies ψ iff the intersection $M \times \mathcal{A}_{\varphi_1}$ satisfies φ_2.

It may be that while ψ is not a safety formula, φ_1 is a safety formula. Then, by the proof of Theorem 2.1 the analysis of $M \times \mathcal{A}_{\varphi_1}$ can ignore the fairness conditions of $\mathcal{A}_{\varphi_1}$ and can proceed with model checking φ_2. (Note, however, that the system $M \times \mathcal{A}_{\varphi_1}$ may not be total, i.e., it may have dead-end states, which need to be eliminated before model checking φ_2.) Suppose now that φ_2 is a safety formula, while φ_1 is not a safety formula. We can then proceed as follows. We first ignore the fairness condition in $M \times \mathcal{A}_{\varphi_1}$ and use the techniques above to model check the safety formula φ_2. Suppose we found a bad prefix that ends with the state $\langle w, q\rangle$ of $M \times \mathcal{A}_{\varphi_1}$. It remains to check that $\langle w, q\rangle$ is a fair state, i.e., that there is a fair path starting from $\langle w, q\rangle$. Instead of performing fair reachability analysis over the entire state space of $M \times \mathcal{A}_{\varphi_1}$ or on the reachable state space of $M \times \mathcal{A}_{\varphi_1}$, it suffices to perform this analysis on the set of states that are reachable from $\langle w, q\rangle$. This could potentially be much easier than doing full fair reachability analysis.

In conclusion, when reasoning about assume-guarantee specifications, it is useful to consider the safety of the assumptions and the guarantee separately.

6.4. Safety in the branching paradigm

Consider a binary tree $T = \{0, 1\}^*$. A prefix of T is a nonempty prefix-closed finite subset of T. For a labeled tree $\langle T, V\rangle$ and a prefix P of T, a *P-extension* of $\langle T, V\rangle$ is a labeled tree $\langle T, V'\rangle$ in which V and V' agree on the labels of the nodes in P. We say that a branching formula ψ is a safety formula iff for every tree $\langle T, V\rangle$ that violates ψ, there exists a prefix P such that all the P-extensions of $\langle T, V\rangle$ violates ψ. The logic CTL is a branching temporal logic. In CTL, every temporal operator is preceded by a path quantifier, E ("for some path") or A ("for all paths").

Theorem 6.2. *Given a CTL formula ψ, deciding whether ψ is a safety formula is EXPTIME-complete.*

Proof: Sistla's algorithm for checking safety of LTL formulas [40] can be adapted to the branching paradigm as follows. Consider a CTL formula ψ. Recall that ψ is not safe iff there is a tree that does not satisfy ψ and all of whose prefixes have at least one extension that satisfies ψ. Without loss of generality we can assume that the tree has a branching degree bounded by $d = |\psi|$ [14]. Let $\mathcal{A}_\psi^d$ be a nondeterministic automaton for ψ; thus $\mathcal{A}_\psi^d$ accepts exactly these d-ary trees that a satisfy ψ. We assume that each state in $\mathcal{A}_\psi^d$ accepts at least one tree (otherwise, we can remove it and simplify the transitions relation). Let $\mathcal{A}_\psi^{d,loop}$ be the automaton obtained from $\mathcal{A}_\psi^d$ by taking all states to be accepting states. Defining the set of accepting states as the set of all states. Thus, $\mathcal{A}_\psi^{d,loop}$ accepts exactly all d-ary trees all of whose prefixes that have at least one extension accepted by $\mathcal{A}_\psi$. Hence, ψ is not safety iff $\mathcal{L}(\mathcal{A}_\psi^{loop}) \cap \mathcal{L}(\mathcal{A}_{\neg\psi})$ is not empty. Since the size of the automata is exponential in ψ and the nonemptiness check is quadratic [46], the EXPTIME upper bound follows.

For the lower bound, we do a reduction from CTL satisfiability. Given a CTL formula ψ, let p be a proposition not in ψ, and let $\varphi = \psi \wedge AFp$. We claim that φ is safe iff ψ is not satisfiable. First, if ψ is not satisfiable, then so is φ, which is therefore safe. For the other direction assume, by way of contradiction, that φ is safe and ψ is satisfied by some tree $\langle T, V\rangle$. The tree $\langle T, V\rangle$ is labeled only by the propositions appearing in ψ. Let $\langle T, V'\rangle$ be an extension of $\langle T, V\rangle$ that refers also to the proposition p and labels T so that $\langle T, V'\rangle$ violates AFp. Clearly, $\langle T, V'\rangle$ does not satisfy φ. Since φ is safe, $\langle T, V'\rangle$ should have a bad prefix P all of whose extensions violate φ. Consider a P extension that agrees with $\langle T, V\rangle$ about the propositions in ψ and has a frontier of p's. Such a P-extension satisfies both ψ and AFp, contradicting the fact that P is a bad prefix. □

Using similar arguments, we prove the following theorem, showing that when we disable alternation between universal and existential quantification in the formula, the problem is as complex as in the linear paradigm.

Theorem 6.3. *Given an ACTL formula ψ, deciding whether ψ is a safety formula is PSPACE-complete.*

Since CTL and ACTL model checking can be completed in time linear [9], and can be performed using symbolic methods, a tree automaton of exponential size that detects finite bad prefixes is not of much help. On the other hand, perhaps safety could offer an advantage in the alternating-automata-theoretic framework of [23]. At this point, it is an open question whether safety is a useful notion in the branching-time paradigm.

Acknowledgment

This research is supported by BSF grant 9800096. M.Y. Vardi is supported in part by NSF grant CCR-9700061 and by a grant from the Intel Corporation.

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

Notes

1. The automaton $\mathcal{A}_\psi$ has linearly many states. Since the alphabet of $\mathcal{A}_\psi$ is 2^{AP}, which may be exponential in the formula, a transition function of a linear size involves an implicit representation.
2. See Section 6.2 for an alternative approach.
3. Note that since ψ may not be intentionally safe, the automaton $\mathcal{U}_{\neg\psi}^{true}$ may not be tight for ψ, thus while w is a minimal informative bad prefix, it may not be a minimal bad prefix.
4. An automatic translation of pathologically safe formulas to intentionally safe formulas is an open problem. Such a translation may proceed through the automaton for the formula's bad prefixes, in which case it would be nonelementary.

References

1. Y. Abarbanel, I. Beer, L. Gluhovsky, S. Keidar, and Y. Wolfstal, "FoCs—Automatic generation of simulation checkers from formal specifications," in *Computer Aided Verification, Proc. 12th Int. Conference*, Lecture Notes in Computer Science, Vol. 1855, Springer-Verlag, 2000, pp. 538–542.
2. B. Alpern and F.B. Schneider, "Defining liveness," *Information Processing Letters*, Vol. 21, pp. 181–185, 1985.
3. B. Alpern and F.B. Schneider, "Recognizing safety and liveness," *Distributed Computing*, Vol. 2, pp. 117–126, 1987.
4. A. Biere, A. Cimatti, E.M. Clarke, M. Fujita, and Y. Zhu, "Symbolic model checking using SAT procedures instead of BDDs," in *Proc. 36th Design Automaion Conference*, IEEE Computer Society, 1999, pp. 317–320.
5. 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*, Vol. 98, No. 2, pp. 142–170, 1992.
6. A. Boujjani, J.-C. Fernandez, S. Graf, C. Rodriguez, and J. Sifakis, "Safety for branching time," in *Proc. 18th International Colloquium on Automata Languages and Programming*, LNCS, Springer-Verlag, 1991, pp. 76–92.
7. 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, 1983.
8. 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*, Lecture Notes in Computer Science, Vol. 131, Springer-Verlag, 1981, pp. 52–71.

9. 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*, Vol. 8, No. 2, pp. 244–263, 1986.
10. W. Canfield, E.A. Emerson, and A. Saha, "Checking formal specifications under simulation," in *Proc. International Conference on Computer Design*, 1997, pp. 455–460.
11. A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer, "Alternation," *Journal of the Association for Computing Machinery*, Vol. 28, No. 1, pp. 114–133, 1981.
12. C. Courcoubetis, M.Y. Vardi, P. Wolper, and M. Yannakakis, "Memory efficient algorithms for the verification of temporal properties," *Formal Methods in System Design*, Vol. 1, pp. 275–288, 1992.
13. E.A. Emerson, "Alternative semantics for temporal logics," *Theoretical Computer Science*, Vol. 26, pp. 121–130, 1983.
14. E.A. Emerson, "Temporal and modal logic," in *Handbook of Theoretical Computer Science*, 1990, pp. 997–1072.
15. N. Francez, *Program Verification*. International Computer Science, Addison-Weflay, 1992.
16. R. Gerth, D. Peled, M.Y. Vardi, and P. Wolper, "Simple on-the-fly automatic verification of linear temporal logic," in P. Dembiski and M. Sredniawa (Eds), *Protocol Specification, Testing, and Verification*, Chapman & Hall, 1995, pp. 3–18.
17. P. Godefroid and P. Wolper, "Using partial orders for the efficient verification of deadlock freedom and safety properties," in *Proc. 3rd Conference on Computer Aided Verification*, Lecture Notes in Computer Science, Vol. 575, Springer-Verlag, 1991, pp. 332–342.
18. R.H. Hardin, R.P. Kurshan, S.K. Shukla, and M.Y. Vardi, "A new heuristic for bad cycle detection using BDDs," in *Computer Aided Verification, Proc. 9th Int. Conference*, Lecture Notes in Computer Science, Vol. 1254, Springer-Verlag, 1997, pp. 268–278.
19. H. Iwashita and T. Nakata, "Forward model checking techniques oriented to buggy designs," in *Proc. IEEE/ACM International Conference on Computer Aided Design*, 1997, pp. 400–404.
20. N. Klarlund, "Mona & Fido: The logic-automaton connection in practice," in *Computer Science Logic, CSL '97*, Lecture Notes in Computer Science, 1998.
21. O. Kupferman and M.Y. Vardi, "Weak alternating automata are not that weak," in *Proc. 5th Israeli Symposium on Theory of Computing and Systems*, IEEE Computer Society Press, 1997, pp. 147–158.
22. O. Kupferman and M.Y. Vardi, "Freedom, weakness, and determinism: From linear-time to branching-time," in *Proc. 13th IEEE Symposium on Logic in Computer Science*, 1998, pp. 81–92.
23. O. Kupferman, M.Y. Vardi, and P. Wolper, "An automata-theoretic approach to branching-time model checking," *Journal of the ACM*, Vol. 47, No. 2, 2000.
24. L. Lamport, "Logical foundation," in *Distributed systems—Methods and Tools for Specification*, Lecture Notes in Computer Science, Vol. 190, Springer-Verlag, 1985.
25. O. Lichtenstein and A. Pnueli, "Checking that finite state concurrent programs satisfy their linear specification," in *Proc. 12th ACM Symposium on Principles of Programming Languages*, New Orleans, 1985, pp. 97–107.
26. Y. Luo, T. Wongsonegoro, and A. Aziz, "Hybrid techniques for fast functional simulation," in *Proc. 35th Design Automation Conference*, IEEE Computer Society, 1998.
27. 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.
28. K.L. McMillan, "Using unfolding to avoid the state explosion problem in the verification of asynchronous circuits," in *Proc. 4th Conference on Computer Aided Verification*, Lecture Notes in Computer Science, Vol. 663, Springer-Verlag, 1992, pp. 164–174.
29. 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*, 1971, pp. 188–191.
30. S. Miyano and T. Hayashi, "Alternating finite automata on ω-words," *Theoretical Computer Science*, Vol. 32, pp. 321–330, 1984.
31. Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Specification*, Springer-Verlag, Berlin, 1992.
32. Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Safety*, Springer-Verlag, New York, 1995.

33. S. Melzer and S. Roemer, "Deadlock checking using net unfoldings," in *Computer Aided Verification, Proc. 9th Int. Conference*, Lecture Notes in Computer Science, Vol. 1254, Springer-Verlag, 1997, pp. 364–375.
34. 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*, 1972, pp. 125–129.
35. S. Owicki and L. Lamport, "Proving liveness properties of concurrent programs," *ACM Transactions on Programming Languages and Systems*, Vol. 4, No. 3, pp. 455–495, 1982.
36. J.P. Queille and J. Sifakis, "Specification and verification of concurrent systems in Cesar," in *Proc. 5th International Symp. on Programming*, Lecture Notes in Computer Science, Vol. 137, Springer-Verlag, 1981, pp. 337–351.
37. K. Ravi and F. Somenzi, "High-density reachability analysis," in *Proc. Int'l Conf. on Computer-Aided Design*, San Jose, 1995, pp. 154–158.
38. S. Safra, "On the complexity of ω-automata," in *Proc. 29th IEEE Symposium on Foundations of Computer Science*, White Plains, October 1988, pp. 319–327.
39. A.P. Sistla and E.M. Clarke, "The complexity of propositional linear temporal logic," *Journal ACM*, Vol. 32, pp. 733–749, 1985.
40. A.P. Sistla, "Satefy, liveness and fairness in temporal logic," *Formal Aspects of Computing*, Vol. 6, pp. 495–511, 1994.
41. 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, 1993.
42. H.J. Touati, R.K. Brayton, and R. Kurshan, "Testing language containment for ω-automata using BDD's," *Information and Computation*, Vol. 118, No. 1, pp. 101–109, 1995.
43. A. Valmari, "On-the-fly verification with stubborn sets," in *Proc. 5nd Conference on Computer Aided Verification*, Lecture Notes in Computer Science, Vol. 697, Springer-Verlag, 1993.
44. M.Y. Vardi, "An automata-theoretic approach to linear temporal logic," in F. Moller and G. Birtwistle (Eds.), *Logics for Concurrency: Structure versus Automata*, Lecture Notes in Computer Science, Vol. 1043, Springer-Verlag, Berlin, 1996, pp. 238–266.
45. M.Y. Vardi and P. Wolper, "An automata-theoretic approach to automatic program verification," in *Proc. First Symposium on Logic in Computer Science*, Cambridge, 1986, pp. 332–344.
46. M.Y. Vardi and P. Wolper, "Automata-theoretic techniques for modal logics of programs," *Journal of Computer and System Science*, Vol. 32, No. 2, pp. 182–221, 1986.
47. M.Y. Vardi and P. Wolper, "Reasoning about infinite computations," *Information and Computation*, Vol. 115, No. 1, pp. 1–37, 1994.
48. P. Wolper, "Synthesis of Communicating Processes from Temporal Logic Specifications," Ph.D. Thesis, Stanford University, 1982.
49. J. Yuan, J. Shen, J. Abraham, and A. Aziz, "On combining formal and informal verification," in *Computer Aided Verification, Proc. 9th Int. Conference*, Lecture Notes in Computer Science, Vol. 1254, Springer-Verlag, 1997, pp. 376–387.