Environment-Friendly Safety

Orna Kupferman and Sigal Weiner

School of Computer Science and Engineering, Hebrew University, Israel

Abstract. Of special interest in verification are safety properties, which assert that the system always stays within some allowed region. For closed systems, the theoretical properties of safety properties as well as their practical advantages with respect to general properties are well understood. For open (a.k.a. reactive) systems, whose behavior depends on their on-going interaction with the environment, the common practice is to use the definition and algorithms of safety for closed systems, ignoring the distinction between input and output signals. In a recent work, Ehlers and Finkbeiner introduced *reactive safety* – a definition of safety for the setting of open systems. Essentially, reactive safety properties require the system to stay in a region of states that is both allowed and from which the environment cannot force it out. In this paper we continue their study and extend it to other families of properties. In the setting of closed systems, each safety property induces a set of finite bad prefixes – ones after which the property must be violated. The notion of bad prefixes enables a reduction of reasoning about safety properties to reasoning about properties of finite computations. We study reactive bad prefixes, their detection in theory and in practice, and their approximation by either a non-reactive safety property or by reasoning about the syntax of the formula. We study the dual notion, of reactive co-safety properties, and the corresponding theory of reactive good prefixes. For both safety and co-safety properties, we relate the definitions in the closed and open settings, and argue that our approach strictly extends the range of properties for which we can apply algorithms that are based on finite computations. Since the reactive setting is particularly challenging for general properties, such an application is significant in practice.

1 Introduction

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 the 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. [1]

[1] Note that the adjective safety describes the properties and not the system. One may say that a system is safe if it satisfies safety specifications, but our use here refers to the specifications

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 u over Σ is a *bad prefix* for L iff for all infinite words v over Σ, the concatenation $u \cdot v$ of u and v 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.

Safety has been widely studied in the formal-verification community; c.f., [1, 8, 14]. The theoretical properties of safety properties as well as their practical advantages with respect to general properties are well understood. The definition and studies of safety, however, treat all the atomic propositions as equal. Thus, they do not distinguish between input and output signals and are suited for closed systems – ones that do not maintain an interaction with their environment. In open (also called *reactive*) systems [6, 12], the system interacts with the environment, and a correct system should satisfy the specification with respect to all environments. A good way to think about the open setting is to consider the situation as a game between the system and the environment. The interaction between the players in this game generates a computation, and the goal of the system is that only computations that satisfy the specification will be generated.

Technically, one has to partition the set AP of atomic propositions to a set I of input signals, which the environment controls, and a set O of output signals, which the system controls. An open system is then an *I/O-transducer* – a deterministic automaton over the alphabet 2^I in which each state is labeled by an output in 2^O. Given a sequence of assignments to the input signals (each assignment is a letter in 2^I), the run of the transducer on it induces a sequence of assignments to the output signals (that is, letters in 2^O). Together these sequences form a computation, and the transducer *realizes* a specification ψ if all its computations satisfy ψ [12].

The transition from the closed to the open setting modifies the questions we typically ask about systems. Most notably, the *synthesis* challenge, of generating a system that satisfies the specification, corresponds to the satisfiability problem in the closed setting and to the realizability problem in the open setting. As another example, the equivalence problem between LTL specifications is different in the closed and open settings [5]. That is, two specifications may not be equivalent when compared with respect to arbitrary systems on $I \cup O$, but be *open equivalent*; that is, equivalent when compared with respect to I/O-transducers. To see this, note for example that a satisfiable yet non-realizable specification is equivalent to false in the open but not in the closed setting.

As mentioned above, the classical definition of safety does not distinguish between input and output signals. The definition can still be applied to open systems, as a special case of closed systems with $\Sigma = 2^{I \cup O}$. In [2], the authors introduced *reactive safety* – a definition of safety for the setting of open systems. The definition in [2] is by means of sets of trees with directions in 2^I and labels in 2^O. The use of trees naturally locate reactive safety between linear and branching safety. Here, we suggest an equivalent yet differently presented definition, which explicitly use realizability. In our definition, a prefix $u \in (2^{I \cup O})^*$ is bad with respect to a property ψ if the system cannot realize ψ after the generation of u. Thus, reactive safety properties require the system to stay in a region of states that is both allowed and from which the environment cannot force it out. In order to indicate that in the open setting we take the environment into an account, we use the term *green safety* to refer to safety in the open setting, and refer to classical safety as *black safety*, or, when clear from the context, safety. To see the difference between the green and black definitions, consider the specification $\psi = G(err \rightarrow F\mathit{fix})$, with $I = \{\mathit{fix}\}$ and $O = \{err\}$. Thus, the system controls the generation of errors, the environment controls the fixes, and the specification is satisfied if every error is eventually fixed. Note that ψ is realizable using the system strategy "never err". Also, ψ is clearly not a safety property, as every prefix can be extended to one that satisfies ψ. On the other hand, ψ is green safe. Indeed, every computation that violates ψ has a green bad prefix – a prefix that ends when the system errs. Once this prefix has been generated, the system has no way to realize the specification, as it is the environment that controls the fixes.

We continue the study of green safety in [2]. We first give further examples to specifications that are green safe but not safe and study their properties. We study green bad prefixes and show that, unlike the closed setting, they are not closed under extensions, and we relate their closure under extension to black safety. We show how one can take advantage of green safety when the specification is not safe (but is green safe) and lift the algorithmic advantages of safety properties to green safety properties. We do so by mapping green safety properties to open-equivalent black safety properties. The mapping is the same as a mapping suggested in [2] by means of nodes in the tree in which a violation starts. In addition to the fact that our definition uses realizability explicitly, which we find simpler, our definition and results apply to general languages, and not only to green or black safety languages. We further formalize the connection between green and black safety by showing that a property is green safe iff it is open equivalent to a black safe property.

We extend the green approach to other families of properties. In the setting of closed systems, the fragment of *co-safety* properties dualizes the one of safety properties: a property is co-safe if its complement is safe. Equivalently, a property is co-safe if every computation that satisfies it has a good prefix – one after which the property aught to hold. In the open setting, dualization is more involved, as one has not only to complement the property but to also to dualize the roles of the system and the environment. Since the game between the system and the environment is determined [4], in the sense that either there is an I/O-transducer that realizes ψ (that is, the system wins) or there is an O/I-transducer that realizes $\neg\psi$ (that is, the environment wins), such a dualization is possible, and we actually have four fragments of languages that are induced by dualization of the green safety definition. The different fragments correspond to whether we talk about safety or co-safety, and whether it is the system or the environment that we consider. We study the theoretical properties of the fragments and the connections among them.

In the closed setting, the intersection of safe and co-safe properties induces the fragment of *bounded* properties – there is an integer $k \geq 0$ such that every word of length k is either a good or a bad prefix [9]. We study boundedness in the open setting and show that the fact green bad and good prefixes are not closed under extension makes the boundedness issue more complicated, as a computation may have both infinitely many good and infinitely many bad prefixes.

In the closed setting, detection of special (bad or good) prefixes has the flavor of validity checking. Accordingly, the problem of deciding whether an LTL specification is safe or co-safe is PSPACE-complete [14]. In the setting of open systems, detection of special prefixes has the flavor of realizability. Thus, reasoning about special prefixes is more complicated. In particular, it is shown in [2] that the problem of deciding whether an LTL formula is reactive safe is 2EXPTIME-complete. Similar bounds hold for the problem of detecting special prefixes. Thus, especially in the open setting, it is interesting to find efficient ways to approximate the language of special prefixes and their detection. We suggest such an approximation by means of *informative green prefixes*. The notion of informative prefixes was introduced for the closed setting in [8]. Essentially, a prefix is informative for a safety property ψ if the syntax of ψ explains why it is a bad prefix. Lifting the notion to open systems involves an approximation that is based both on examining the syntax, rather than the semantics of the property, and an approximation of realizability by satisfiability. We argue that for natural specifications, the approximations are accurate.

Finally, our ability to replace green safe properties by simpler safe properties as well as the fact that our syntactic-based approximation is accurate for natural specifications are useful not only for easier reasoning about but also in order to assess the quality of specifications. This later point is very important in the context of property-based design [13]. The setting of open systems is particularly challenging for property assurance: solving the synthesis problem, decomposition of specifications is not always possible, making the detection of dependencies among different components of the specification much more difficult.

Due to the lack of space, some proofs are omitted from this version and can be found in the full version, in the authors' URLs.

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 proposition using the usual Boolean operators and the temporal operators G ("always"), F ("eventually"), X ("next time"), and U ("until"). We define the semantics of LTL with respect to a *computation* $\pi = \sigma_0, \sigma_1, \sigma_2, \ldots$, where for every $j \geq 0$, we have that σ_j is a subset of AP, denoting the set of atomic propositions that hold in the j-th position of π. We use $\pi \models \psi$ to indicate that an LTL formula ψ holds in the computation π. We use $\|\psi\|$ to denote the set of computations in $(2^{AP})^\omega$ that satisfy ψ. A full definition of the syntax and semantics of LTL can be found in [11].

2.2 Safety languages and formulas

Consider a language $L \subseteq \Sigma^\omega$ of infinite words over the alphabet Σ. A finite word $u \in \Sigma^*$ is a *bad prefix* for L if for all $v \in \Sigma^\omega$, we have $u \cdot v \notin L$. Thus, a bad prefix is a finite word that cannot be extended to an infinite word in L. Note that if u is a bad prefix, then all the finite extensions of u are also bad prefixes. A language L is a *safety* language if every word not in L has a finite bad prefix [1, 8, 14]. For a language L, we denote by $bp(L)$ the set of all bad prefixes for L. We say that an LTL formula ψ is a safety formula iff $\|\psi\|$ is a safety language.

2.3 Open systems

We model open systems by *transducers*. Let I and O be finite sets of input and output signals, respectively. Given $x = i_0 \cdot i_1 \cdot i_2 \cdots \in (2^I)^\omega$ and $y = o_0 \cdot o_1 \cdot o_2 \cdots \in (2^O)^\omega$, we denote their composition by $x \oplus y = (i_0, o_0) \cdot (i_1, o_1) \cdot (i_2, o_2) \cdots \in (2^{I \cup O})^\omega$. An I/O-transducer is a tuple $\mathcal{T} = \langle I, O, S, s_0, \eta, L \rangle$, where S is a set of states, $s_0 \in S$ is an initial state, $\eta : S \times 2^I \to S$ is a transition function, and $L : S \to 2^O$ is a labeling function. The *run* of $\mathcal{T}$ on a (finite or infinite) input sequence $x = i_0 \cdot i_1 \cdot i_2 \cdots$, with $i_j \in 2^I$, is the sequence $s_0, s_1, s_2, \ldots$ of states such that $s_{j+1} = \eta(s_j, i_{j+1})$ for all $j \geq 0$. The *computation* of $\mathcal{T}$ on x is then $x \oplus y$, for $y = L(s_0) \cdot L(s_1) \cdot L(s_2) \cdots$ Note that $\mathcal{T}$ is responsive and deterministic (that is, it suggests exactly one successor state for each input letter), and thus $\mathcal{T}$ has a single run, generating a single computation, on each input sequence. We extend η to finite words over 2^I in the expected way. In particular, $\eta(s_0, x)$, for $x \in (2^I)^*$ is the $|x|$-th state in the run on x. A transducer $\mathcal{T}$ induces a *strategy* $f : (2^I)^* \to 2^O$ such that for all $x \in (2^I)^*$, we have that $f(x) = L(\eta(s_0, x))$. Given an LTL formula ψ over $I \cup O$, we say that ψ is *I/O-realizable* if there is a finite-state I/O-transducer $\mathcal{T}$ such that all the computations of $\mathcal{T}$ satisfy ψ [12]. We then say that $\mathcal{T}$ realizes ψ. When it is clear from the context, we refer to I/O-realizability as *realizability*, or talk about realizability of languages over the alphabet $2^{I \cup O}$.

Since the realizability problem corresponds to deciding a game between the system and the environment, and the game is determined [4], realizability is determined too, in the sense that either there is an I/O-transducer that realizes ψ (that is, the system wins) or there is an O/I-transducer that realizes $\neg\psi$ (that is, the environment wins). Note that in an O/I-transducer the system and the environment "switch roles" and the system is the one that provides the inputs to the transducer. A technical detail is that in order for the setting of O/I-realizability to be dual to the one in I/O-realizability we need, in addition to switching the roles and negating the specification, to switch the player that moves first and consider transducers in which the environment initiates the interaction and moves first. Since we are not going to delve into constructions, we ignore this point, which is easy to handle.

3 Green Safety

Let I and O be sets of input and output signals, respectively. Consider a language $L \subseteq (2^{I \cup O})^\omega$. For a finite word $u \in (2^{I \cup O})^*$, let $L^u = \{s : \ u \cdot s \in L\}$ be the set of all infinite words s such that $u \cdot s \in L$. Thus, if L describes a set of allowed computations, then L^u describes the set of allowed suffixes of computations starting with u.

We say that a finite word $u \in (2^{I \cup O})^*$ is a *system bad prefix* for L iff L^u is not realizable. Thus, a system bad prefix is a finite word u such that after traversing u, the system does not have a strategy to ensure that the interaction with the environment would generate a computation in L. We use $sbp(L)$ to denote the set of system bad prefixes for L. Note that by determinacy of games, whenever L^u is not realizable by the system, then its complement is realizable by the environment. Thus, once a bad prefix has been generated, the environment has a strategy to ensure that the entire generated behavior is not in L.

A language $L \subseteq (2^{I \cup O})^\omega$ is a *green safety language* if every word not in L has a system bad prefix.

Example 1. Let $I = \{q\}$, $O = \{p\}$, $\psi = Gp \vee FGq$, and $L = \|\psi\|$. Note that ψ is realizable using the system strategy "always output p". We show L is green safe. Consider a word $w \notin L$. Since w does not satisfy Gp, there must be a prefix u of w such that u contains a position satisfying $\neg p$. Since words with prefix u do not satisfy Gp, we have that $L^u = \|FGq\|$. Since $q \in I$, the specification FGq is not realizable. Thus, u is a system bad prefix and L is green safe.

On the other hand, L is not safe. Consider for example the word $w = \emptyset^\omega$. While w is not in L, for every finite computation u of w, the suffix $s = \{q\}^\omega$ is such that $u \cdot s \models FGq$, implying that $u \cdot s \in L$. Thus, w has no bad prefix, implying that L is not safe.

Example 2. Let $I = \{q\}$, $O = \{p\}$, $\psi = G(p \to Fq)$, and $L = \|\psi\|$. Note that ψ is realizable using the system strategy "never output p". Also, ψ is clearly not a safety property, as every prefix can be extended to one that satisfies it. On the other hand, L is green safe. Indeed, every word not in L must have a prefix u that ends with $\{p\}$. Since $L^u = \|Fq\|$ and $q \in I$, so the specification Fq is not realizable, we have that u is a system bad prefix and L is green safe.

Note that when $I = \emptyset$, which corresponds to the case of closed systems, we have that L^u is not realizable iff L^u is empty. Thus, when $I = \emptyset$, safety coincides with green safety.

Explaining the intuition behind green safety, we are going to use the following terminology. We say that the system *errs* when it generates a system bad prefix. The environment, however, may *forgive* these errors and not follow a winning strategy after it. In Example 1, the system errs whenever it outputs $\neg p$. In Example 2, the system errs whenever it outputs p. In both cases, when this happens, the environment may follow a strategy with which the generated computations do not satisfy ψ, say by always inputting $\emptyset$, but it may also forgive the errors by following a strategy with which ψ still holds, say by always inputting $\{q\}$.

Remark 1. While presented differently, our definition of green safety is equivalent to the definition of reactive safety in [2]. The definition there is by means of sets of trees with directions in 2^I and labels in 2^O. The use of trees naturally locate reactive safety between linear and branching safety. On the other hand, we find the explicit use of realizability in our definition much simpler and easier to work with, as it naturally conveys the intuition of safety in the open setting.

3.1 Properties of green safety

We start by checking some theoretical properties of green safety.

Proposition 1. *Every non-realizable language is green safe, with ϵ being a system bad prefix.*

Proof: Since $L^{\epsilon} = L$, we have that L is not realizable iff L^{ϵ} is not realizable, which holds iff ϵ is a system bad prefix. Therefore, if L is not realizable, every word not in L has ϵ as a system bad prefix, and so L is green safe. $\square$

As pointed out in [2], green safety is strictly weaker than safety. We present here the proof using our alternative definition of green safety.

Proposition 2. *Every safe language is green safe, but the other direction is not necessarily true.*

Proof: Let L be a safe language. Consider a word $w \notin L$ and a bad prefix $u \in (2^{I \cup O})^*$ of w. Since u is a bad prefix, the set L^u is empty, and is therefore unrealizable, so u is also a system bad prefix. Thus, every word not in L has a system bad prefix, implying that L is green safe. Strictness is demonstrated in Example 1. $\square$

In the closed settings, the set $bp(L)$ is closed under finite extensions for all languages $L \subseteq \Sigma^\omega$. That is, for every finite word $u \in bp(L)$ and finite extension $v \in \Sigma^*$, we have that $u \cdot v \in bp(L)$. As we shall see now, the set of system bad prefixes is not closed under finite extensions. The reason is that the environment need not take advantage of errors made by the system, possibly giving the system another chance to win. Below we give two such examples.

Example 3. Let $I = \{\mathit{fix}\}$, $O = \{\mathit{err}\}$, and $\psi = G(\mathit{err} \rightarrow X\mathit{fix}) \wedge FG\neg \mathit{err}$. Thus, ψ states that every error the system makes is fixed by the environment in the following step, and that there is a finite number of errors. Let $L = \|\psi\|$. Clearly, L is realizable, as the strategy "make no errors" is a winning strategy for the system.

We first show that L is green safe. Consider a word $w \notin L$. Since $w \not\models \psi$, there must be a prefix u of w such that u ends in a position satisfying err. We claim that u is a system bad prefix. Indeed, an environment strategy starting with $\neg\mathit{fix}$ guarantees that the condition $G(\mathit{err} \rightarrow X\mathit{fix})$ is not satisfied, and hence is a winning strategy for the environment after u was generated. Hence, L^u is not realizable, implying that L is green safe.

We now show that $sbp(L)$ is not closed under finite extensions. Consider the word $w = (\{\mathit{err}, \mathit{fix}\} \cdot \{\mathit{fix}\})^\omega$. That is, the system makes an error on every odd position, and the environment always fixes errors. Since there are infinitely many errors in w, it does not satisfy ψ. The prefix $u = \{\mathit{err}, \mathit{fix}\}$ of w is a system bad prefix. Indeed, an environment strategy that starts with $\neg\mathit{fix}$ is a winning strategy. On the other hand, u's extension $v = \{\mathit{err}, \mathit{fix}\} \cdot \{\mathit{fix}\}$ is not a system bad prefix. Indeed, L^v is realizable using the winning system strategy "make no errors".

Note that w has infinitely many system bad prefixes and infinitely many undetermined prefixes. For the same language, we can also point to a word with only one system bad prefix. Consider the word $w' = \{\mathit{err}, \mathit{fix}\} \cdot \{\mathit{fix}\}^\omega$. Note that w' is in L. Here, the system makes only one error, which is fixed, and then makes no more errors. While $\{\mathit{err}, \mathit{fix}\}$ is a system bad prefix, every longer prefix of w' contains the fix for the first error, and does not contain further errors by the system, Therefore, it is not a system bad prefix.

Example 4. In the previous example, we saw a word w' with only one system bad prefix, but w' was in L. Let $I = \{\mathit{fix}\}$, $O = \{\mathit{err}, \mathit{ack}\}$, and $\psi = G(\mathit{err} \rightarrow X(\mathit{fix} \wedge F\,\mathit{ack}))$. Thus, ψ states that after the system makes an error, the environment must fix it, and the system must also eventually acknowledge the error. Let $L = \|\psi\|$. In the full version we show that L is green safe and not safe and that here is a word not in L that has only one system bad prefix.

We can conclude with the following:

Proposition 3. *The set of system bad prefixes is not closed under extension.*

3.2 From green to black safety

As studied in [8], reasoning about safety properties is easier than reasoning about general properties. In particular, rather than working with automata on infinite words, one can model check safety properties using automata (on finite words) for bad prefixes. In the open setting, when the specification we reason about is safe, we can use algorithms developed for safety languages. The question is whether and how we can take advantage of green safety when the specification is not safe (but is green safe). In this section we answer this question positively and lift the algorithmic advantages of safety properties to green safety properties. We do so by mapping green safety properties to open-equivalent black safety properties.

For a language $L \subseteq (2^{I \cup O})^\omega$, we define $\mathit{black}(L) = L \cap \{w\ :\ w$ has no system bad prefix for $L\}$. Equivalently, $\mathit{black}(L) = L \setminus \{w\ : w$ has a system bad prefix for $L\}$. Intuitively, we obtain $\mathit{black}(L)$ by defining all the finite extensions of $\mathit{sbp}(L)$ as bad prefixes. Accordingly, it is easy to see that $\mathit{sbp}(L) \subseteq \mathit{bp}(\mathit{black}(L))$. We sometimes apply black on LTL formulas, mapping formulas to formulas.

Example 5. Consider the specification $\psi = G(\mathit{err} \rightarrow X\mathit{fix}) \wedge FG\neg \mathit{err}$, with $I = \{\mathit{fix}\}$, $O = \{\mathit{err}\}$. In Example 3 we saw that ψ is green safe. Moreover, an infinite word contains a system bad prefix for ψ iff it has a position that satisfies err. Accordingly, $\mathit{black}(\psi) = G\neg \mathit{err}$. The specification ψ is a basis to similar specifications. For example, in a thread-management context, if we replace err by $\mathit{Zero\_x}$ and fix by $\mathit{Interrupt}$, where interrupt stands for the operating system interrupting the system thread, then the formula $\psi = G(\mathit{Zero\_x} \rightarrow X\,\mathit{Interrupt}) \wedge FG\neg \mathit{Zero\_x}$ states that the value of x, which the system controls, can be 0 only finitely often and that whenever it is 0, the environment must not interrupt the system in the next transition. For this formula, we get that $\mathit{black}(\psi) = G\neg \mathit{Zero\_x}$. This matches our intuition: If an interrupt can occur at any time, and we want to avoid an interrupt when x is 0, we must never set x to 0.

Example 6. Consider the specification $\psi = G(\mathit{err} \rightarrow X(\mathit{fix} \wedge F\,\mathit{ack}))$, with $I = \{\mathit{fix}\}$, $O = \{\mathit{err}, \mathit{ack}\}$. In Example 4 we saw that ψ is green safe. Moreover, an infinite word contains a system bad prefix for ψ iff it has a position that satisfies err. Accordingly, $\mathit{black}(\psi) = G\neg \mathit{err}$. Here too, the structure of ψ is a basis to similar specifications. For example, in a network with packet loss, replacing err with $\neg legal$ (for sending an illegal packet), *fix* with $drop$ (for packet dropped by the network), and ack with $resend$, we get the specification "illegal packets are eventually resent, and no illegal packet reaches its destination". For this formula, we get that $black(\psi) = Glegal$. This matches our intuition: the only way to avoid an arrival of an illegal packet to its destination is to never send one.

Remark 2. A similar transition from green to black safety is described in [2], by means of nodes in the tree in which a violation starts, which are analogous to our system bad prefixes. In addition to the fact that our definition uses realizability explicitly, which we find simpler, our definition and results apply to general languages, and not only to green or black safety languages.

Theorem 1. *Consider a language $L \subseteq (2^{I\cup O})^\omega$. The following are equivalent:*

1. *L is green safe.*
2. *$\{w : w$ has no system bad prefix$\} \subseteq L$; that is, $black(L) = \{w : w$ has no system bad prefix$\}$.*
3. *$black(L)$ is black safe.*

Proof: We first prove if L is green safe then $\{w : w$ has no system bad prefix$\} \subseteq L$. Assume that L is green safe. Consider a word $w \in \{w : w$ has no system bad prefix$\}$, and assume by way of contradiction that $w \notin L$. Since L is green safe and $w \notin L$, we have that w has a system bad prefix for L, contradicting the fact that $w \in \{w : w$ has no system bad prefix$\}$.

We now prove that if $\{w : w$ has no system bad prefix$\} \subseteq L$ then $black(L)$ is black safe. Consider a word $w \notin black(L)$. By definition, $black(L) = L \cap \{w : w$ has no system bad prefix$\}$, and since $\{w : w$ has no system bad prefix$\} \subseteq L$, we have that $black(L) = \{w : w$ has no system bad prefix$\}$. Therefore, w has a system bad prefix u. For every suffix $s \in (2^{I\cup O})^\omega$, the word $w' = u \cdot s$ contains the system bad prefix u and therefore $w' \notin black(L)$. Thus, u is a bad prefix in $black(L)$, implying that $black(L)$ is black safe.

Finally, we prove that if $black(L)$ is black safe then L is green safe. Assume that $black(L)$ is black safe, and consider a word $w \notin L$. Since $black(L) \subseteq L$, we have that $w \notin black(L)$. Therefore, w has a bad prefix u in $black(L)$. If $u \in sbp(L)$, we are done since w indeed has a system bad prefix. Otherwise, we claim that u has a prefix v such that $v \in sbp(L)$. Since u is not a system bad prefix, the system has a winning strategy from u, and that strategy generates a suffix $s \in (2^{I\cup O})^\omega$ such that $w' = u \cdot s \in L$. Since $u \in bp(black(L))$, we have that $w' \notin black(L)$, so w' has a prefix $v \in sbp(L)$. We claim that $|v| \leq |u|$. Indeed, every prefix of w' that is not a prefix of u was generated by a winning strategy for the system. Therefore, it cannot be a system bad prefix. Now, if $|v| \leq |u|$ then v is also a prefix of w, so w has a system bad prefix. Therefore, L is green safe. $\square$

While L and $black(L)$ are not equivalent, they are *open equivalent*, in the sense of [5]. Formally, we have the following.

Theorem 2. *For every language $L \subseteq (2^{I \cup O})^\omega$ and I/O-transducer $\mathcal{T}$, we have that $\mathcal{T}$ realizes L iff $\mathcal{T}$ realizes $black(L)$.*

Proof: Since $black(L) \subseteq L$, then clearly every transducer that realizes $black(L)$ also realizes L. For the other direction, let L be some language and consider an I/O-transducer $\mathcal{T}$ that realizes L. Assume by contradiction that $\mathcal{T}$ does not realize $black(L)$. Then, there is a computation w of $\mathcal{T}$ such that $w \in L \setminus black(L)$. Since $black(L) = L \cap \{w : w \text{ has a system bad prefix for } L\}$ and $w \in L \setminus black(L)$, it must be that $w \notin \{w : w \text{ has a system bad prefix for } L\}$. Thus, w has a system bad prefix u. Since u is a system bad prefix, we have that L^u is not realizable, which means that after u was generated, the environment has a winning strategy. Since L is ω-regular, there is also an O/I-transducer that implements such a winning strategy. Let $\mathcal{T}'$ be such an O/I-transducer. Consider the word $w' = u \cdot (x \oplus y)$, where $x \in (2^I)^\omega$ and $y \in (2^O)^\omega$ are the input and output sequences generated when the environment follows $\mathcal{T}'$, and the system follows $\mathcal{T}^u$ (that is, T after u has been generated). So, $x = \mathcal{T}'(y)$ and $y = \mathcal{T}^u(x)$. Since $y = \mathcal{T}^u(x)$, we have that $w' = u \cdot (x \oplus T^u(x))$ is a computation of $\mathcal{T}$. Since $\mathcal{T}'$ is a winning strategy for the environment, we have that $w' = u \cdot (\mathcal{T}'(y) \oplus y) \notin L$. On the one hand, since $\mathcal{T}$ realizes L, all the traces of $\mathcal{T}$ are in L. On the other hand, w' is a trace of $\mathcal{T}$, so we have reached a contradiction. Therefore, T also realizes $black(L)$. $\square$

Note that Theorem 2 applies to arbitrary languages and not only for green safe ones.

Theorem 2 suggests that we can reason about ψ, and in particular solve its model-checking (with respect to transducers) and synthesis problems by reasoning about $black(\psi)$. Consider for example the green safety property $\psi = G(p \to Fq)$, where $black(\psi) = G\neg p$ (recall that p is an output signal, see Example 2). Our ability to replace ψ by the much simpler formula $black(\psi)$ is similar to our ability to simplify specifications with *inherent vacuity* [3]. Indeed, green-but-not-black safety typically indicates that the specifier is not fully aware of the many aspects of the specification. Thus, green safety is useful not only for reasoning about simpler specifications but also in order to assess the quality of specifications, which is very important in the context of property-based design [13], especially in the setting of open systems. The setting of open systems is indeed particularly challenging for property assurance: solving the synthesis problem, decomposing of specifications is not always possible, making the detection of dependencies among different components of the specification much more difficult.

It is shown in [2], that given an LTL formula ψ, it is possible to construct a deterministic looping word automaton for $black(\psi)$ with doubly-exponential number of states.[2] In fact, as suggested in [8], it is then possible to generate also a deterministic automaton for the bad prefixes of $black(\psi)$. Note that when L is not realizable, we have that $\epsilon \in sbp(L)$, implying that $black(L) = \emptyset$. It follows that we cannot expect to construct small automata for $black(L)$, even nondeterministic ones, as the realizability problem for LTL can be reduced to easy questions about them.

Theorem 2 implies that a green safety language L is open equivalent to a safe language, namely $black(L)$. We complete the picture by showing that open equivalence to a safe language implies green safety.

[2] A looping automaton is a Büchi automaton in which all states are accepting. It is known [8, 14] that safety properties can be translated to looping automata.

Theorem 3. *A language L is green safe iff L is open equivalent to a safe language.*

Proof: First, if L is green safe, then, by Theorem 1, we have that $black(L)$, which is open equivalent to L, is safe.

For the other direction, assume that L is open equivalent to a safe language L'. We show that L is green safe. Assume by way of contradiction that L is not green safe. Then, there is a word $w \notin L$ with no system bad prefix. In the full version we show that the above implies that the word w also has no system bad prefix in L', which implies, as L' is a safe language, that $w \in L'$. Consider the following (infinite) transducer T: As long as T gets inputs that agree with w, it generates outputs that agree with w and continues. Once the input does not agree with w, the prefix generated so far is a prefix of w. Since w has no system bad prefix in L', there is a system winning strategy in L' from this prefix, and T plays that strategy. Since T either generates $w \in L'$, or reaches a position from which it plays a system winning strategy in L', it follows that T realizes L'. Since, however, T generates w, which is not in L, it does not realize L, contradicting the fact that L and L' are open equivalent. We note that, by [5], the existence of an infinite transducer that distinguishes between L and L' implies the existence of such a finite transducer. $\square$

4 Green Co-Safety

For a language $L \subseteq \Sigma^\omega$, we use $comp(L)$ to denote the complement of L; i.e., $comp(L) = \Sigma^\omega \setminus L$. In the closed setting, we say that a language $L \subseteq \Sigma^\omega$ is a *co-safety* language if $comp(L)$ is a safety language. (The term used in [10] 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 $gp(L)$ the set of good prefixes for L. Note that $gp(L) = bp(comp(L))$ [8]. Finally, an LTL formula ψ is a co-safety formula iff $\|\psi\|$ is a co-safety language or, equivalently, $\|\neg\psi\|$ is a safety language.

In the setting of open systems, dualization of specifications is more involved, as one has not only to complement the language but to also dualizes the roles of the system and the environment. Accordingly, we actually have four fragments of languages that are induced by dualization of the green safety definition. We define them by means of bad and good prefixes.

Consider a language $L \subseteq (2^{I\cup O})^\omega$ and a prefix $u \in (2^{I\cup O})^*$. We say that

– u is a *system bad prefix* if L^u is not I/O-realizable.
– u is a *system good prefix* if L^u is I/O-realizable.
– u is an *environment bad prefix* if L^u is not O/I-realizable.
– u is an *environment good prefix* if L^u is O/I-realizable.

Now, a language $L \subseteq (2^{I\cup O})^\omega$ is a *system (environment) safety language* if every word not in L has a system (environment, respectively) bad prefix. The language L is a *system (environment) co-safety language* if every word in L has a system (environment, respectively) good prefix. Note that system safety coincides with green safety. Here, that we parametrize safety with either a system or an environment, we simplify the notation and omit "green".

Since each language L^u is either I/O-realizable or not I/O-realizable, and the same for O/I-realizability, all finite words are determined, in the following sense.

Proposition 4. *Consider a language $L \subseteq (2^{I \cup O})^\omega$. All finite words in $(2^{I \cup O})^*$ are determined with respect to L. That is, every prefix is either system good or system bad, and either environment good or environment bad, with respect to L.*

Note that while every prefix is determined, a word may have both system bad and system good prefixes, and similarly for the environment, which is not the case in the setting of closed systems. For example, recall the language $L = \|G(err \to X\mathit{fix}) \wedge FG\neg err\|$, for $I = \{\mathit{fix}\}$ and $O = \{err\}$. In Example 3 we saw that the word $(\{err, fix\} \cdot \{fix\})^\omega$ has both a system bad prefix $\{err, fix\}$, and a system good prefix $\{err, fix\} \cdot \{fix\}$.

In a dual manner to Proposition 1, every realizable language is system co-safe with ϵ being a system good prefix for every word in L. Accordingly, our goal in studying co-safety is two fold. First, since a system good prefix u is such that L^u is I/O-realizable, then the set of system good prefixes describe the "hopeful scenarios" for the system – ones after which it would be able to realize a non-realizable specification. Second, the story of safety and co-safety is told about both the system and the environment. As we shall now see, system safety and environment co-safety dualize each other.

Proposition 5. *For every language $L \subseteq (2^{I \cup O})^\omega$, we have that L is system safe iff $comp(L)$ is environment co-safe.*

By switching the roles of the system and the environment, we get that L is system co-safe iff $comp(L)$ is environment safe.

It is interesting to consider the special case when $I = \emptyset$. There, O/I-realizability coincides with validity. Therefore, given a language $L \subseteq (2^O)^\omega$, a prefix u is an environment good prefix iff $L^u = \Sigma^\omega$, which coincides with the definition of a good prefix in the closed settings. Therefore, when $I = \emptyset$, environment co-safety coincides with co-safety.

4.1 Boundness

We say a property ψ is *bounded* if there is an integer $k \geq 0$ such that every word of length k is either a good or a bad prefix for ψ. In the closed settings, a language that is both safe and co-safe is bounded [9]. In the open setting, we can talk about two relevant intersections. The first is languages that are both system safe and system co-safe (or dually, both environment safe and environment co-safe). The second is languages that are both system safe and environment co-safe (or dually, both environment safe and system co-safe). In this section we consider the fragments corresponding to both types of intersection.

We start with the first fragment. We denote by $[\![P]\!]$ the set of languages that have the property P. As we have previously seen, every unrealizable language is system safe, and every realizable language is system co-safe. Therefore, $[\![\text{system safe}]\!] \cap [\![\text{system co-safe}]\!] = ([\![\text{realizable}]\!] \cap [\![\text{system safe}]\!]) \cup ([\![\text{unrealizable}]\!] \cap [\![\text{system co-safe}]\!])$. As we have seen in Section 3, system safety is of interest in the case of realizable languages, and the realizable languages that are system safe are not bounded. Likewise, unrelizability does not impose boundedness on specifications that are system co-safe. Thus, there is no reason to expect a language that is both system safe and system co-safe to be bounded. We are going to confirm this intuition in Example 7 below. Thus, interestingly, the intersection system safe and system co-safe properties is not related to boundedness and instead suggests a characterization of realizable and non-realizable specifications.

We continue to the second fragment. Let L be a language that is both system safe and environment co-safe. Consider a word $w \in (2^{I \cup O})^\omega$. If $w \in L$, then, as L is environment co-safe, w has a good environment prefix. If $w \notin L$, then, as L is system safe, w has a bad system prefix. As in the closed setting, it follows that w must have a "special" – either environment co-safe or system safe prefix. In the closed setting, it was possible to use this information in order to bound the length of the shortest such prefix. As we shall see now, this strongly depends on the fact the bad and good prefixes in the closed setting are closed under extensions, and is no longer valid in the open setting.

Example 7. Consider the formula $\psi = G(err \to Gfix)$, for $I = \{fix\}$ and $O = \{err\}$. Let $L = \|\psi\|$. It is easy to see that L is I/O-realizable with the system strategy "make no errors". Thus, L is system co-safe. For every word $w \notin L$, we have that $w \models F(err \wedge F\neg fix)$. Therefore, every word $w \notin L$ has a prefix that contains a position satisfying err, and ends in a position satisfying $\neg fix$. Such a prefix is a black bad prefix, and is thus both a system bad and an environment bad prefix. Therefore, L is both environment safe and system safe. Finally, L is also O/I-realizable, with the environment environment strategy "always fix". It follows that L is also co-safe.

Hence, L belongs to the four green safety and co-safety fragments. On the other hand, L is not bounded. To see this, consider the word $w = \emptyset^\omega$. For every prefix u of w, the suffix $s = \{err\}^\omega$ is such that $u \cdot s \notin L$, and the suffix $s' = \emptyset^\omega$ is such that $u \cdot s' \in L$. Thus, w has undetermined prefixes of unbounded length, and so L is not bounded.

Since in this example we show a language that has all four green safety properties, but is not bounded, we can conclude with the following.

Proposition 6. *A language in an intersection of system safety, system co-safety, environment safety, and environment co-safety, need not be bounded.*

In the full version, we consider a dualization of $black(L)$, namely the set $white(L)$ obtained by adding all the infinite extensions of environment good prefixes to L. An environment good prefix in L is thus a good prefix in $white(L)$.

We show that for every language $L \subseteq (2^{I \cup O})^\omega$, we have $comp(white(L)) = black(comp(L))$. By dualizing our results on green and black safety, we thus have that L is environment co-safe iff $white(L)$ is co-safe, and that L and $white(L)$ are co-open-equivalent.

5 Green Informative Prefixes

In the closed setting, detection of special (bad or good) prefixes has the flavor of validity checking. Accordingly, the problem of deciding whether an LTL specification is safe or co-safe is PSPACE-complete [14], and the size of an automaton for the special prefixes is doubly-exponential in the LTL formula that describes the specification

[8]. The doubly-exponential blow up is present even when the automaton is nondeterministic. Intuitively, the need to accept all the special prefixes requires the construction to have the flavor of determinization, as one has to relate different components of the specification. In the setting of open systems, detection of special prefixes has the flavor of realizability. Thus, reasoning about special prefixes is more complicated. In particular, it is shown in [2] that the problem of deciding whether an LTL formula is reactive safe is 2EXPTIME-complete. In fact, as we show in the full version, the problem is 2EXPTIME-hard even for specifications that are known to be realizable. Similarly, as showed in [2], automata that recognize the system bad prefixes of a reactive safety property are of size doubly-exponential in the LTL formula.

In [8], the authors introduced the notion of *informative prefixes* in the context of closed systems. Given an LTL formula ψ, the set of informative prefixes for ψ is a subset of $bp(\psi)$ that is easier to detect. Essentially, a prefix is informative for ψ if the syntax of ψ explains why it is a bad prefix. In this section we lift the notation of informative prefixes and their applications to the open setting. We first need the following definition and notations. We assume that LTL formulas are written in a positive normal form, where negation is pushed inward and is applied only to atomic propositions. For this, we have to introduce the dual R ("release") of U ("until"). We use $cl(\psi)$ to denote the set of subformulas of ψ (after transferring ψ to a positive normal form).

For an LTL formula ψ over $AP = I \cup O$ and a finite computation $\pi = \sigma_1 \cdot \sigma_2 \cdots \sigma_n$, with $\sigma_i \in 2^{I \cup O}$, we say that π is *green informative* for ψ if 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. $L(n+1)$ contains only formulas over I, and the formula $\bigwedge_{\varphi \in L(n+1)} \varphi$ is satisfiable.
3. For all $1 \leq j \leq n$ and $\varphi \in L(j)$, the following hold:
   - If φ is a propositional assertion, it is satisfied by σ_j.
   - If $\varphi = \varphi_1 \vee \varphi_2$ then $\varphi_1 \in L(j)$ or $\varphi_2 \in L(j)$.
   - If $\varphi = \varphi_1 \wedge \varphi_2$ then $\varphi_1 \in L(j)$ and $\varphi_2 \in L(j)$.
   - If $\varphi = X\varphi_1$, then $\varphi_1 \in L(i+1)$.
   - If $\varphi = \varphi_1 U \varphi_2$, then $\varphi_2 \in L(j)$ or $[\varphi_1 \in L(i)$ and $\varphi_1 U \varphi_2 \in L(j+1)]$.
   - If $\varphi = \varphi_1 R \varphi_2$, then $\varphi_2 \in L(j)$ and $[\varphi_1 \in L(i)$ or $\varphi_1 V \varphi_2 \in L(j+1)]$.

If π is informative for ψ, then the mapping L is called the *green witness* for $\neg\psi$ in π. Intuitively, $L(j)$, for $j \geq 0$, is the set of subformulas in $cl(\neg\psi)$ that are yet to be satisfied in order for $\neg\psi$ to be satisfied in a computation that has $\sigma_1 \cdot \sigma_2 \cdots \sigma_{j-1}$ as a prefix. In the closed setting, the requirement on $L(n+1)$ is to be empty, corresponding to the requirement that no more obligations have to be satisfied in order for $\neg\psi$ to hold in all possible suffixes. In the open setting, the corresponding requirement would have been that $L(n+1)$ is such that the conjunction of the formulas in it is O/I-realizable. We refer to prefixes that satisfy the above as *strong green informative prefixes*. As we shall see below, while such prefixes are more precise, they are harder to detect. In the other extreme, we could have require the formulas in $L(n+1)$ to only refer to I and give up the satisfiability checking. We call such prefixes *weak informative green prefixes*. While checking for weak prefixes is easier, they do not guarantee that the prefix is system bad.

In the definition above, the requirements left to be checked in $L(n+1)$ are on I and their conjunction has to be satisfiable. Since all the requirements are on I, satisfiability and realizability coincide, which guarantees that a green informative prefix is indeed a system bad prefix.

Note that when $I = \emptyset$, the requirement above for $L(n+1)$ is equivalent to the requirement $L(n+1) = \emptyset$, thus the definition of a green informative prefix coincides with the definition of informative prefix.

Example 8. Let $I = \{q\}$, $O = \{p\}$ and let $\psi_1 = G(p \rightarrow Fq)$. Using the positive normal form, we have that $\neg\psi_1 = F(p \wedge G\neg q)$, where we use $F\varphi$ as an abbreviation for $trueU\varphi$, and $G\varphi$ as an abbreviation for $falseR\varphi$. The finite computation $\pi = \emptyset \cdot \{p\}$ is a green informative prefix for ψ_1, as witnessed by the mapping L with $L(1) = \{F(p \wedge G\neg q)\}$, $L(2) = \{F(p \wedge G\neg q), p \wedge G\neg q, p, G\neg q, \neg q\}$, $L(3) = \{G\neg q\}$. Indeed, $|\pi| = 2$ and $L(2+1)$ contains a satisfiable formula over I.

We now consider two variants of the previous example. The first is $\psi_2 = G(p \rightarrow (Fq \vee (Xr \wedge X\neg r))$, where $I = \{q\}$, $O = \{p, r\}$. Note that since $Xr \wedge X\neg r$ is not satisfiable, the specifications ψ_1 and ψ_2 are equivalent. Still, informative prefixes consider the syntax of the formula. To see that the syntax may be crucial, let us examine π again, now with respect to $\neg\psi_2 = F(p \wedge (G\neg q \wedge ((X\neg r) \vee Xr)))$. We can see π is not a green informative prefix for ψ_2, as such a prefix must contain at least one state after the first state in which p holds, to syntactically verify that $(X\neg r) \vee Xr$ holds. Note that if r had been an input, then π would have been a green informative prefix.

The second variant is $\psi_3 = G(p \rightarrow ((X\neg q) \wedge Xq))$, where $I = \{q\}$ and $O = \{p\}$. Now, $\neg\psi_3 = F(p \wedge (Xq \vee X\neg q))$. We can see that π is a green informative prefix, as $((X\neg q) \vee Xq)$ is over I and is satisfiable. Formally, the mapping L with $L(1) = \{\neg\psi_3\}$, $L(2) = \{\neg\psi_3, p \wedge (Xq \vee X\neg q), p, Xq \vee X\neg q, Xq\}$, and $L(3) = \{q\}$ is a green witness for $\neg\psi_3$. On the other hand, in the closed setting π is not an informative prefix for ψ_3, as such a prefix must contain at least one state after the first state in which p holds, to syntactically verify that $((X\neg q) \vee Xq)$ holds.

The fact that the requirement about $L(n+1)$ is easier to satisfy in the open rather than in the closed setting, together with the example of ψ_3 above, imply the following.

Theorem 4. *Green information is weaker than black information. That is, every informative prefix is also a green informative prefix, but the other direction is not necessarily true.*

The syntax-based definition leads to an easier detection of bad prefixes:

Theorem 5. *Given an LTL formula ψ and a finite computation π, the problem of deciding whether π is green informative for ψ is PSPACE-complete.*

Proof: We start with the upper bound. Consider a prefix $\pi = \sigma_1, \ldots, \sigma_n$ and an LTL formula ψ. As shown in [8], it is possible to construct in time $O(n \cdot |\psi|)$ a mapping $L_{max} : \{1, \ldots, n+1\} \rightarrow 2^{cl(\neg\psi)}$ such that $L_{max}(j)$ contains exactly all the formulas $\neg\varphi$ such that the suffix $\sigma_j, \ldots, \sigma_n$ is informative for φ. Extending this construction to the open setting requires a guess of the formulas in $L(n+1)$, making the guess and the check that the conjunction of the formulas is satisfiable the computational bottleneck. Since satisfiability, as well as going over all possible guesses, can be done in PSPACE, we are done.

For the lower bound, we show a reduction from LTL satisfiability problem, which is PSPACE-complete. Given an LTL formula ψ over AP, we consider the specification $\theta = \neg\psi$, with $I = AP$ and $O = \emptyset$. It is easy to see ϵ is a green informative prefix for θ iff ψ is satisfiable. $\square$

Remark 3. Since the generation of $L(n+1)$ is the computational bottleneck, working with the strong and weak green informative prefix definition results in detection problems that are 2EXPTIME-complete and linear-time, respectively.

Finally, as in the closed setting, it is possible to define an automaton that recognizes exactly all the informative green prefixes of a given safety formula. It is also possible to use the notion of informative green prefixes in order to classify green safety formulas according to the level in which informative prefixes approximate the set of all bad prefixes. The technical details are similar to these in [8], with the different conditions on $L(n+1)$ imposing the expected changes, in both the algorithms and the complexity. We describe the full details in the full version.

References

1. B. Alpern and F.B. Schneider. Recognizing safety and liveness. *Distributed computing*, 2:117–126, 1987.
2. R. Ehlers and B. Finkbeiner. Reactive safety. In *Proc. 2nd GANDALF*, EPTCS 54, pages 178–191, 2011.
3. D. Fisman, O. Kupferman, S. Seinvald, and M.Y. Vardi. A framework for inherent vacuity. In *HVC*, LNCS 5394, pages 7–22, 2008.
4. D. Gale and F. M. Stewart. Infinite games of perfect information. *Ann. Math. Studies*, 28:245–266, 1953.
5. K. Greimel, R. Bloem, B. Jobstmann, and M.Y. Vardi. Open implication. In *Proc. 35th ICALP*, LNCS 5126, pages 361–372, 2008.
6. D. Harel and A. Pnueli. On the development of reactive systems. In *Logics and Models of Concurrent Systems*, *NATO Advanced Summer Institutes*, F-13, pages 477–498, 1985.
7. H. Kress-Gazit, G.E. Fainekos, and G.J. Pappa. Temporal-logic-based reactive mission and motion planning. *IEEE Trans. on Robotics*, 25(6):1370–1381, 2009.
8. O. Kupferman and M.Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.
9. O. Kupferman and M.Y. Vardi. On bounded specifications. In *Proc. 8th LPAR*, LNCS 2250, pages 24–38, 2001.
10. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1992.
11. A. Pnueli. The temporal logic of programs. In *Proc. 18th FOCS*, pages 46–57, 1977.
12. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th POPL*, pages 179–190, 1989.
13. PROSYD. The Prosyd project on property-based system design. http://www.prosyd.org, 2007.
14. A.P. Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6:495–511, 1994.