On Locally Checkable Properties

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

[1] Hebrew University, School of Eng. and Computer Science, Jerusalem 91904, Israel
Email: {orna,yoadl}@cs.huji.ac.il, URL: http://www.cs.huji.ac.il/~{orna,yoadl}
[2] Rice University, Department of Computer Science, Houston, TX 77251-1892, U.S.A., and Microsoft Research, Cambridge, U.K. Email:vardi@cs.rice.edu, URL: http://www.cs.rice.edu/~vardi

Abstract. The large computational price of formal verification of general ω-regular properties has led to the study of restricted classes of properties, and to the development of verification methodologies for them. Examples that have been widely accepted by the industry include the verification of safety properties, and bounded model checking. We introduce and study another restricted class of properties – the class of *locally checkable* properties. For an integer $k \geq 1$, a language $L \subseteq \Sigma^\omega$ is *k-checkable* if there is a language $R \subseteq \Sigma^k$ (of "allowed subwords") such that a word w belongs to L iff all the subwords of w of length k belong to R. A property is locally checkable if its language is k-checkable for some k. Locally checkable properties, which are a special case of safety properties, are common in the specification of systems. In particular, one can often bound an eventuality constraint in a property by a fixed time frame.
The practical importance of locally checkable properties lies in the low memory demand for their run-time verification. A monitor for a k-checkable property needs only a record of the last k computation cycles. Furthermore, even if a large number of k-checkable properties are monitored, the monitors can share their memory, resulting in memory demand that do not depend on the number of properties monitored. This advantage of locally checkable properties makes them particularly suitable for run-time verification. In the paper, we define locally checkable languages, study their relation to other restricted classes of properties, study the question of deciding whether a property is locally checkable, and study the relation between the size of the property (specified by an LTL formula or an automaton) and the smallest k for which the property is k-checkable.

1 Introduction

It is generally acknowledged that one of the main obstacles to the development of complex computerized systems lies in the process of system verification. In system verification we try to ensure that a model of the system satisfies some specification.

* Supported in part by BSF grant 9800096, and by a grant from Minerva.
** Supported in part by NSF grants CCR-9988322, CCR-0124077, CCR-0311326, and ANI-0216467, by BSF grant 9800096, and by Texas ATP grant 003604-0058-2003. Part of this work was done while the author was visiting the Isaac Newton Institute for Mathematical Science, as part of a Special Programme on Logic and Algorithms, supported by a Guggenheim Fellowship.

Common specification formalisms are based on temporal logic [Pnu81] and automata on infinite words [Kur94,VW94]. Such formalisms are very expressive, for example automata over infinite objects can specify all ω-regular properties. Expressiveness, however, comes at a cost, as verifying a general ω-regular property is sometimes very costly in terms of the resources needed for the verification. As a result, an effort is being made to identify and study less expressive formalisms that allow for an easier verification process. An example of a class of properties that gain a lot of attention is the class of *safety properties* [AS85,Sis94]. Safety properties require the system to always stay within some allowed region, and their verification can be reduced to invariant checking [KV01a]. A large portion of the properties actually checked in the industry are safety properties. As a result, specialized algorithms for safety properties were developed and are successfully used in practice [BBL98].

Even a larger effect on industrial verification was made by the introduction of *bounded model-checking* (BMC) [CBRZ01]. In BMC, only a bounded interval of time at the beginning of the computation is checked. While BMC techniques can be applied to general temporal logic properties, they correspond the class of *bounded properties* – properties that regard only an interval of length k, for some $k \geq 0$, in the beginning of the computation [KV01b]. In practice, it is possible to apply BMC to significantly large systems. Thus, focusing on a bounded interval of interest at the beginning of the computation is extremely fruitful in practice.

While BMC techniques check bounded properties, and thus computations are evaluated from their initial state, it is possible to check bounded properties also from an arbitrary state of the computation. This is done in symbolic trajectory evaluation (STE) [SB95,AJS98]. In this work we study properties that are similar to properties checked by STE: while still focusing our interest on intervals of a bounded length $k \geq 0$, we would like to regard every k-long time interval throughout the computation.

Let us start with an example: a classical no-starvation specification has the form "throughout the computation, whenever a *req* signal is raised, it is followed by an *ack* signal raised sometime in the future". Such a specification can be seen as characterizing "events" of an unbounded length. The event begins when a *req* signal is raised, and ends when an *ack* signal is raised. In real systems, one can often bound the time frame within which the event should occur. The user may expect, for example, that the following bounded version of the no-starvation specification holds: "throughout the computation, whenever a *req* signal is raised, it is followed by an *ack* signal raised within seven computation cycles".

We introduce in this paper the novel concept of checkability. A language $L \subseteq \Sigma^\omega$ is *k-checkable* if there exists a finite language $R \subseteq \Sigma^k$ such that for every word $w \in \Sigma^\omega$ it holds that $w \in L$ iff all the k-long subwords of w are elements of R. In such a case, we say that L is induced by R. A language is *locally checkable* (or simply checkable) if it is k-checkable for some k. Note that our bounded version of "no starvation" can be characterized by an 8-checkable language. For example, a language in which all 8-long subwords are permissible except those in which a *req* was raised in the first cycle but no *ack* was raised later.

Intuitively, checkable languages are languages that can be verified by a verifier with a bounded memory – one that has access only to the last k-computation cycles. The practical importance of this distinction lies the context of runtime verification [HR02,BGHS04,dR05]. Run-time verification of a property amounts to executing a monitor together with the system allowing the detection of errors in run time. Run-time monitors for checkable specifications have low memory demand. Furthermore, in the case of general ω-regular properties, when several properties are checked, we need a monitor for each property, and since the properties are independent of each other, so are the state spaces of the monitors. Thus, the memory demand (as well as the resources needed to maintain the memory) grow linearly with the number of properties monitored. Such a memory demand is a real problem in practice. In contrast, we show that a monitor for a k-checkable property needs only a record of the last k computation cycles. Furthermore, even if a large number of k-checkable properties are monitored, the monitors can share their memory, resulting in memory demand that do not depend on the number of properties monitored. This advantage of checkable languages make them particularly suited to be used as specification formalism for run-time verification.

An extensively studied family of languages related to checkable languages is that of *locally testable* languages [MP71,Wil93]. A language L is locally testable if the answer to the question "is the word w in L?" is determined by a bounded prefix of w and by the set of subwords of w of some bounded length k. Thus, checkable languages are a special case of testable languages: in the case of checkable languages, the only question one might ask of the set of subwords of length k is about their containment in a set R. Note that the bounded-memory advantage that holds for checkable languages does not hold for general testable languages. Indeed, in the case of testable languages, one must maintain the set of all subswords of length k seen so far in order to establish membership, and this involves remembering which subwords were seen at the beginning of the computation. In fact, we prove that locally testable languages constitute a much more expressive formalism. In particular, there are locally testable properties that are not locally checkable at all, and for an arbitrarily large k, there are 2-testable properties that are k-checkable but not $(k-1)$-checkable.

In this paper we define k-checkable and locally checkable languages and study their properties. We provide some basic constructions and observations, and study the relation between locally checkable languages and other fragments of ω-regular properties such as safety properties, bounded properties, uninitialized properties, and testable properties. In addition, we study the problem of deciding whether a specification is locally checkable, or k-checkable, and the relation between the size of the smallest Büchi automaton or LTL formula for a checkable specification and the smallest k for which the specification is k-checkable.

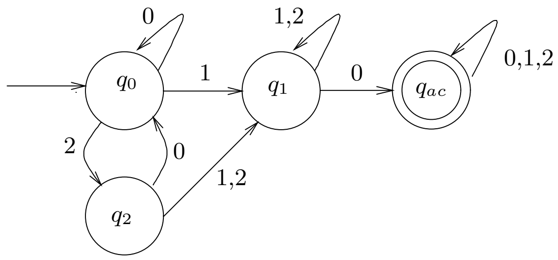
2 Preliminaries

Consider an alphabet Σ. For a word $w = w_1 w_2 w_3 \ldots$ over Σ, we denote the length of w by $|w|$. Note that $|w|$ is either a natural number (in case $w \in \Sigma^*$), or the first infinite ordinal ω (in case $w \in \Sigma^\omega$). For $i, j \in \mathbb{N}$ such that $i \leq j \leq |w|$, we denote by $w^i = w_i w_{i+1} \ldots$ the suffix of w starting at the ith letter and denote by $w[i..j] = w_i w_{i+1} \ldots w_j$ the subword between the ith and jth letters. For $w \in \Sigma^\omega$, we denote by $\mathit{suff}(w)$ the set of suffixes of w, i.e. $\mathit{suff}(w) = \{w^i \mid i \geq 0\}$. For a word $w \in \Sigma^\omega$, we denote by $sub(w)$ the set of finite subwords of w, formally, $sub(w) = \{y \in \Sigma^* \mid \exists x \in \Sigma^*, z \in \Sigma^\omega \text{ such that } w = xyz\}$. For $k \geq 0$, we denote by $sub(w, k)$ the set of subwords of w of length k, i.e., $sub(w, k) = sub(w) \cap \Sigma^k$.

A language $L \subseteq \Sigma^\omega$ is *k-checkable* if there exists a finite language $R \subseteq \Sigma^k$ such that $w \in L$ iff all the k-long subwords of w are elements of R. That is, $L = \{w \in \Sigma^\omega \mid sub(w,k) \subseteq R\}$. In such a case, we say that L is induced by R. A language is *locally checkable* (or simply *checkable*) if it is k-checkable for some k. A language $L \subseteq \Sigma^\omega$ is *k-co-checkable* if there exists a finite language $R \subseteq \Sigma^k$ such that $w \in L$ iff there exists a k-long subword of w that is a an element of R. That is, $L = \{w \in \Sigma^\omega \mid sub(w,k) \cap R \neq \emptyset\}$. A language is *locally co-checkable* (or simply *co-checkable*) if it is k-co-checkable for some k.

We assume the reader is familiar with nondeterministic Büchi word automata (NBWs) [Tho90]. We describe an NBW by a tuple $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, F \rangle$, where Σ is the alphabet, Q is the set of states, $Q_0 \subseteq Q$ is the set of initial states, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, and $F \subseteq Q$ is the set of accepting states. When $|Q_0| = 1$ and $|\delta(q,\sigma)| = 1$ for all $q \in Q$ and $\sigma \in \Sigma$, we say that $\mathcal{A}$ is a deterministic Büchi automaton (DBW). For $S \subseteq Q$, we denote by $\mathcal{A}^S$ the NBW obtained from $\mathcal{A}$ by changing the set of initial states to S; i.e. $\mathcal{A}^S = \langle \Sigma, Q, S, \delta, F \rangle$. For $s \in Q$ we use A^s to denote $A^{\{s\}}$. We denote the language of $\mathcal{A}$ by $L(\mathcal{A})$. For a fragment of the ω-regular languages (e.g., k-checkable) and an NBW $\mathcal{A}$, we say that $\mathcal{A}$ is in the fragment (e.g., $\mathcal{A}$ is a k-checkable NBW) iff $L(\mathcal{A})$ is in the fragment.

Example 1 Let $\Sigma = \{0,1,2\}$. The DBW $\mathcal{A}$ below recognizes the language L of all the words that contain 10, 120 or 220 as subwords. Note that L is the 3-co-checkable language L co-induced by $R = \{010, 110, 210, 100, 101, 102, 120, 220\}$. Indeed, a word w is in L iff $sub(w,3) \cap R \neq \emptyset$.



□

The DFA in Example 1 exhibits some of the subtlety to be found in co-checkable languages. At first glance, one would assume that the traversal through a minimal sub-word of the inducing set should not contain a cycle. This impression, however, is misleading, as demonstrated in the DBW above, where a traversal through the subword 120 contains a cycle.

Consider a language $R \subseteq \Sigma^k$ of words of length k. We denote by $check(R)$ the checkable language induced by R. Formally, $check(R) = \{w \in \Sigma^\omega | sub(w,k) \subseteq R\}$. Similarly, we denote by $co\text{-}check(R)$ the language $\{w \in \Sigma^\omega | sub(w,k) \cap R \neq \emptyset\}$. For a checkable (co-checkable) language L, we denote by $width(L)$ the smallest natural number k for which there exists a set $R \subseteq \Sigma^k$ such that $L = check(R)$ (resp. $L = co\text{-}check(R)$).

Note that a checkable language L may be induced by more than one language of words of length $width(L)$. For example, let $\Sigma = \{a, b, c\}$ and $L = a^\omega + a^* c^\omega + c^\omega$. Both $R_1 = \{aa, ac, cc\}$ and $R_2 = \{aa, ac, cc, ab\}$ induce L. In this case, however, R_2 clearly contains a redundant word.

Proposition 2

1. *For each checkable language L there exists a unique minimal inducing set, denoted $induce(L)$.*
2. *For each co-checkable language L there exists a unique maximal inducing set, denoted $co\text{-}induce(L)$.*
3. *For every checkable language L, it holds that $check(induce(L)) = L$. For every co-checkable language L, it holds that $co\text{-}check(co\text{-}induce(L)) = L$.*

Proof: For a checkable language $L \subseteq \Sigma^\omega$, we define $induce(L)$ to be the intersection of all languages $R \subseteq \Sigma^{width(L)}$ that induce L. Formally, $induce(L) = \bigcap\{R \subseteq \Sigma^{width(L)} \mid check(R) = L\}$. For a co-checkable language $L \subseteq \Sigma^\omega$, we define $co\text{-}induce(L)$ to be the union of all languages $R \subseteq \Sigma^{width(L)}$ that co-induce L. Formally, $co\text{-}induce(L) = \bigcup\{R \subseteq \Sigma^{width(L)} \mid co\text{-}check(R) = L\}$.

For the third claim, consider some $k \geq 0$, and some k-checkable language L. We first prove that $L \subseteq check(induce(L))$. Let w be a word in L. Then, for every set $R \subseteq \Sigma^k$ that induces L, it holds that $sub(w, k) \subseteq R$. Therefore, $sub(w, k) \subseteq induce(L)$, and $w \in check(induce(L))$. For the other direction, let w be a word in $check(induce(R)$. For every set $R \subseteq \Sigma^k$ that induces L, it holds that $sub(w, k) \subseteq R$. Since L is k-checkable, there exists at least one such set R that induces L so w must be in L. The proof for the co-checkable case follows similar lines. $\square$

3 Basic observations

In this section we study some basic properties of checkable (and co-checkable) languages.

We begin with the closure of checkable languages with respect to set operations. For a language $R \subseteq \Sigma^*$, the complement of R is the language $\Sigma^* \setminus R$. For a language $L \subseteq \Sigma^\omega$ the complement of L is the language $\Sigma^\omega \setminus L$. In either case, we denote the complement of L by $comp(L)$. It is easy to see that checkable languages are not closed under complementation. For example, the language $\{a^\omega\}$ over the alphabet $\{a, b\}$ is 1-checkable whereas its complement $\{a, b\}^\omega \setminus \{a^\omega\}$ is not checkable. As we now show, however, checkable and co-checkable languages complement each other.

Proposition 3 *Checkable and co-checkable languages complement each other in the following ways:*

1. *The complement of a k-checkable language is k-co-checkable and vice-versa.*
2. *A finite language $R \subseteq \Sigma^k$ induces the k-checkable language L iff $\Sigma^k \setminus R$ co-induces the k-co-checkable language $comp(L)$.*
3. *For a co-checkable language L, $co\text{-}induce(L) = \Sigma^{width(L)} \setminus induce(comp(L))$.*

Proof: The second proposition follows from noting that for every word $w \in \Sigma^\omega$, it holds that $sub(w, k) \subseteq R$ iff $sub(w, k) \cap (\Sigma^k \setminus R) = \emptyset$. Both the first and the third propositions follow from the second proposition. $\square$

Note that, in the context of verification, we expect specifications of the type "whenever a *req* signal is raised, it is followed by an *ack* signal raised within seven computation cycles", rather then specifications of the type "there exists time window in which a *req* signal is raised, and it is followed by an *ack* signal raised within seven computation cycles". Thus, checkable specifications are more suitable for verification then co-checkable specifications[3].

Proposition 4 *Let L_1 and L_2 be k-checkable languages. Then, $L_1 \cap L_2$ is k-checkable, but $L_1 \cup L_2$ need not be checkable.*

Proof: For intersection, it is not hard to see that $induce(L_1) \cap induce(L_2)$ induces $L_1 \cap L_2$. For union, consider the language $L_1 = a^\omega + a^* c^\omega + c^\omega$ (induced by $\{aa, ac, cc\}$), and $L_2 = a^\omega + c^* a^\omega + c^\omega$ (induced by $\{aa, ca, cc\}$). For $k \geq 0$, note that every k-long subword of $a^k c^k a^\omega$, appears either in $a^* c^\omega$ or in $c^* a^\omega$. Therefore, every k-checkable language that contains both $a^* c^\omega$ and $c^* a^\omega$ must contain $a^k c^k a^\omega$. Therefore $L_1 \cup L_2$ cannot be k-checkable for any $k \geq 0$. $\square$

As we discuss further in the sequel, the closure of checkable properties under intersection is crucial for efficient run-time verification.

We now proceed with generic constructions of automata for checkable and co-checkable languages.

Theorem 5.

1. *For every $R \subseteq \Sigma^k$ there exists a DBW $\mathcal{A}$, with at most $O(|\Sigma|^k)$ states, such that $L(\mathcal{A}) = check(R)$. Similarly, there exists a DBW $\mathcal{A}'$, with the same parameters, for which $L(\mathcal{A}') = co\text{-}check(R)$. Furthermore, both $\mathcal{A}$ and $\mathcal{A}'$ can be constructed using polynomial space.*
2. *For every $R \subseteq \Sigma^k$, there exists an NBW $\mathcal{A}$, with at most $O(k|R|)$ states, such that $L(\mathcal{A}) = co\text{-}check(R)$. Furthermore, $\mathcal{A}$ can be constructed in polynomial time.*

Proof: Let $R \subseteq \Sigma^k$ be some finite inducing (or co-inducing) language. We start with the construction of an NBW $\mathcal{A}$ for *co-check*(R) Let $\mathcal{A}'$ be DFW of size $O(k|R|)$ recognizing R. It is easy to construct $\mathcal{A}'$ by letting it guess a word in R, and then either reach an accepting sink in case the guess is correct or a rejecting sink in case it is not. In order to get the NBW $\mathcal{A}$ for *co-check*(R), we add to the initial states of $\mathcal{A}'$ self loops.

We proceed with the construction of a DBW $\mathcal{A}$ for *check*(R). The main idea is to construct $\mathcal{A}$ so that throughout its run, $\mathcal{A}$'s state either maintains the last $k-1$ letters read, or is a rejecting sink. For this purpose, $\mathcal{A}$'s set of states is $\bigcup_{i=0}^{k-1} \Sigma^i$ as well as a rejecting sink. The initial state is ϵ, and all states but the sink are accepting. The run proceeds so that the state of $\mathcal{A}$ maintains the last $k-1$ letters read. In case $\mathcal{A}$ encounters some k-long subword that is not in R, then $\mathcal{A}$ moves into the rejecting sink. It is not hard to see that the language of $\mathcal{A}$ is *check*(R). The construction of $\mathcal{A}'$, for the co-checkable case, differs only slightly: the sink is accepting while all other states are not.

[3] This is the case since checkable specifications are safety while co-checkable specifications are co-safety (see Proposition 7).

We turn now to prove the optimality of the last construction. Let $R = \{\sigma_1 \cdots \sigma_k \in \Sigma^k \mid \sigma_1 = \sigma_k\}$. For every $(k-1)$-long word $u \in \Sigma^{k-1}$ we have $u^\omega \in check(R)$. On the other hand, for two different $(k-1)$-long words we have $u \cdot v^\omega \notin check(R)$. Assume, towards contradiction, that there exists an NBW $\mathcal{A}$ with less than $|\Sigma|^{k-1}$ states for $check(R)$. For each $u \in \Sigma^{k-1}$, let $r^u = r_0^u r_1^u \ldots$ be an accepting run of $\mathcal{A}$ on u^ω. Since $\mathcal{A}$ has less states then the number of words in Σ^{k-1}, there must exist two different words $u, v \in \Sigma^{k-1}$ for which $r_{k-1}^u = r_{k-1}^v$. In such a case, however, $r_0^u \ldots r_{k-1}^u r_k^v r_{k+1}^v \ldots$ is an accepting run of $\mathcal{A}$ on $u \cdot v^\omega$. We reached contradiction implying that such an $\mathcal{A}$ with less than $|\Sigma|^{k-1}$ states cannot exist. $\square$

Note that, as shown in the proof of Theorem 5, there exists $R \subseteq \Sigma^k$ for which the smallest NBW for *check*(R) has at least $|\Sigma|^{k-1}$ states. Thus, applying nondeterminizm in order to get an improved construction, is possible only for *co-check*.

The basic observations made so far suggest that locally checkable specifications have properties that are desirable in the context of verification. The closure of checkable properties under intersection is crucial for the fact, discussed in Section 1, that run-time verification of several checkable properties requires bounded memory, which does not depend on the number of properties checked. Indeed, combining the monitors of several k-checkable properties, one gets a new single monitor, with the same state space as each of the single monitors. Indeed, if the languages $L_1, L_2, \ldots, L_n$, for n different k-checkable properties, are induced by $R_1, R_2, \ldots, R_n$, respectively, then the intersection $R = R_1 \cap R_2 \cap \cdots \cap R_n$ induces $L_1 \cap L_2 \cap \cdots \cap L_n$. A DBW for *check*$(R)$, described in Theorem 5, can then serve as a monitor for all the properties: the DBW is run in parallel to the verified system, and once it moves to the rejecting sink (note that all other states are accepting) an error is reported. Note also that while the number of states of the single monitor is exponential in k, the memory demand is linear in k.

Finally, in the context of model checking we enjoy both the fact that we do not need our monitors to be deterministic, and the fact that we work with complemented specifications, which are co-checkable. Therefore, in the context of model checking, the smaller nondeterministic monitors for co-checkable specifications can be used.

4 Deciding checkability

In this section we study decision problems related to checkability.

4.1 Relation to other families of languages

We start by investigating the relations between checkable and co-checkable languages and other restricted fragments of ω-regular languages. In section 4.2, we use these observations in order to decide checkability of languages.

We consider safety, co-safety, and bounded languages first. Let L be a language of infinite words over Σ. A finite word $x \in \Sigma^*$ is a *bad prefix* for L if for all infinite words $y \in \Sigma^\omega$, 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 into an infinite word in L. In a similar fashion, a finite word $x \in \Sigma^*$ is a *good prefix* for L, if for all infinite words $y \in \Sigma^\omega$, the concatenation $x \cdot y$ of x and y is in L. A language L is a *safety language* if every word not in L has a finite bad prefix. A language L is a *co-safety language* if every word in L has a finite good prefix [AS85]. For a co-safety language L we denote by $good(L)$ the set of good prefixes for L. For $k \geq 0$, a language L is *bounded with bound* k if every k-long word $x \in \Sigma^k$ is either a good prefix, or a bad prefix for L. A language L is *bounded* if there exists $k \geq 0$ for which the language is bounded with bound k. A language is bounded iff it is both safety and co-safety [KV01b].

Two other related families of languages we study are *uninitialized* [HKKM02] (also called suffix closed) and *liveness* [AS85] languages. A language is uninitialized if for every word w in L, every suffix of w is also in L. Thus, a language L is uninitialized iff $suff(L) \subseteq L$. A language L is *liveness* if for every word w in L and every word $v \in \Sigma^\omega$, if w is a suffix of v, then $v \in L$. Thus, L is liveness iff $\Sigma^* \cdot L \subseteq L$.

Proposition 6 *A language $L \subseteq \Sigma^\omega$ is uninitialized iff $comp(L)$ is liveness*

Proof: We prove that L is not uninitialized iff $comp(L)$ is not liveness. Assume that $comp(L)$ is not liveness. Then there is $w \in comp(L)$ and $u \in \Sigma^*$ such that $u \cdot w \notin comp(L)$. This means that $u \cdot w \in L$ but $w \notin L$, so L is not uninitialized.

Assume now that L is not uninitialized. Then there is $w \in L$ such that $w = u \cdot v$ and $v \notin L$. This means that $v \in comp(L)$ but $u \cdot v \notin comp(L)$, so $comp(L)$ is not liveness. $\square$

Proposition 7

1. *Checkable languages are both safety and uninitialized. Co-checkable languages are both co-safety and liveness.*
2. *There exists a safety uninitialized language that is not checkable. There exists a co-safety liveness language that is not co-checkable.*
3. *No language, other then $\emptyset$ and Σ^ω, is both bounded and checkable (or co-checkable).*

Proof:
1. To see that a checkable language is safety, let L be a checkable language and $x \in \Sigma^\omega$ a word not in L. Then x must contain some k-long subword $x[i..i+k-1]$ that is not in $induce(L)$. Therefore, the prefix $x[1..i+k-1]$ is a bad prefix for L. The fact a checkable language is uninitialized follows from the easy observation that for all $i \geq 0$ it holds that $sub(w^i, k) \subseteq sub(w, k)$, therefore if $sub(w, k) \subseteq R$ then $sub(w^i, k) \subseteq R$. The results for co-checkable languages follow.
2. We proceed To show a safety uninitialized language that is not checkable. Let $\Sigma = 2^{\{p,q,r\}}$, and $L \subseteq \Sigma^\omega$ be the language of words in which whenever a letter containing q appears, then either a letter containing r appears eventually and until then only letters containing p appear, or all the letters that appear in the suffix after the word containing q contains p. For readers familiar with LTL, the language L corresponds to the formula $G(q \rightarrow (pWr))$.
   It is easy to see that L is both uninitialized and safety. We prove that L is not checkable. Assume by way of contradiction that for some $k \geq 0$ the language L is k-checkable. Consider the word $w = \{p, q\} \cdot \{p\}^k \cdot \emptyset \cdot \{p\}^\omega$. Clearly, $w \notin L$. Therefore, there exists some $i \geq 0$ such that $w[i..i+k-1] \notin induce(L)$. There are three forms of k-long subwords of w: the subword $\{p, q\} \cdot \{p\}^{k-1}$, the subword $\{p\}^k$, and for every $i < k$, the subword $\{p\}^i \cdot \emptyset \cdot \{p\}^{k-i-1}$. Subwords of the first

two forms are also subwords of $\{p,q\} \cdot \{p\}^\omega$, which is in L. Therefore, subwords of the first two forms must be in $induce(L)$. Similarly, subwords of the third form are subwords of $\{p\}^k \cdot \emptyset \cdot \{p\}^\omega$, which is in L. Therefore, all of the subwords of w are in $induce(L)$ in contradiction to the fact that $w \notin L$.

The complementary language is co-safety liveness language that is not co-checkable.

3. Finally, we show that a non trivial bounded language is neither checkable nor co-checkable. Assume, toward contradiction, that there is a language L that is both bounded, with bound b, and checkable. Let w be some word in L and v some word not in L. Consider $u = w[1..b] \cdot v$. Since L is bounded u must be in L. On the other hand, since L is checkable and v is not in L, the word u (that contain all the k-long subwords contained in v) cannot be in L. Thus a contradiction is reached.

   Since bounded languages are closed under complementation, the result for co-checkable languages follows by looking at the complementary languages.

□

As discussed in Section 1, checkable languages are a special case of testable languages. A language L is *k-testable* [MP71,Wil93] if for every two words $w_1, w_2 \in \Sigma^\omega$, if $w_1[0..k-1] = w_2[0..k-1]$ and $sub(w_1,k) = sub(w_2,k)$ then $w_1 \in L$ iff $w_2 \in L$. A language is *locally testable* (or simply *testable*) if it is k-testable for some $k \geq 0$. Every checkable (or co-checkable) language is trivially testable. The other direction, however, does not hold. As discussed in Section 1, the expressive power of testable languages has a computational price, as the memory demand for a general k-testable property is exponential in k. Indeed, while a monitor for a k-checkable language only has to remember the last k letters, a monitor for a k-testable language has to remember all the subwords of length k seen so far.

Proposition 8

1. *There are testable languages that are neither checkable nor co-checkable.*
2. *For every $k \geq 0$, there exists a 2-testable language that is k-checkable but not $(k-1)$-checkable.*
3. *For every k-testable property there is an NBW with at most $2^{|\Sigma|^{O(k)}}$ states.*
4. *There are k-testable properties for which every NBW has at least $2^{|\Sigma|^{\Omega(k)}}$ states.*

Proof: 1. First, we show a testable language neither checkable nor co-checkable. Let $\Sigma = 2^{\{p,q\}}$. The language L of all words in which all letters contain p and at least one letter contain q is testable and is neither checkable nor co-checkable.

2. We proceed to show, for every $k > 0$, a language that is 2-testable and k-checkable but not $(k-1)$-checkable. Consider the alphabet $\Sigma = \{1, \ldots, k\}$, and the language $L_1 = suff((1 \cdot 2 \cdots k)^\omega)$. It is not hard to see that L is 2-checkable, and is induced by $R_1 = \{1 \cdot 2, 2 \cdot 3, \ldots k-1 \cdot k, k \cdot 1\}$. Thus, L_1 is also 2-testable. Consider now the alphabet $\Sigma' = \{1, \ldots, k, k'\}$ and the languages $L_2 = suff((1 \cdot 2 \cdots k-1 \cdot k')^\omega)$. Note that L_2 is identical to L_1 up to substitution of one letter. Thus, L_2 is 2-checkable, and is induced by $R_2 = \{1 \cdot 2, 2 \cdot 3, \ldots k-1 \cdot k', k' \cdot 1\}$. Now, let $L = L_1 \cup L_2$. Since a word w is in L iff all its subwords of length 2 are either all contained in R_1 or all contained in R_2, the language L is 2-testable.

The language L, however, contains no word in which both k and k' appear. In order to enforce such uniformity, at least two consecutive appearances of k or k' must be seen. It follows that L is $k+1$-checkable but its width of cannot be smaller than $k+1$.
3. We proceed to the construction of an NBW for a k-testable language. It is not hard to see, that it is possible to keep track of the set of k-long subwords seen so far, using an NBW of size $2^{|\Sigma|^{O(k)}}$. Therefore, every k-testable language has an NBW of size $2^{|\Sigma|^{O(k)}}$.
4. Finally, we show the optimality of the last construction. Let $S \subseteq \Sigma^k$ be a set of size $|\Sigma|^{k-1}$. Let T_S be the testable specification that is satisfied by a word $w \in \Sigma^\omega$ iff $sub(w,k) = S$. It is not hard to see that a monitor for T_S must to have at least $2^{|\Sigma|^{k-1}}$ states.

□

4.2 Decision problems

We now turn to the problem of deciding whether a property is checkable or co-checkable. We consider properties given as NBWs. We first need to study some more properties of checkable languages.

Consider a language L. A word $x \in \Sigma^*$ is a *minimal good prefix* for L if it is a good prefix, and no strict prefix or strict suffix of x are good prefixes. Consider for example the language $L = a\Sigma^\omega \cup bc^*a\Sigma^\omega$. It is easy to see that L is a co-safety language and that the good prefixes of L are the words in $a\Sigma^*$ and $bc^*a\Sigma^*$. Note that a is a good prefix, and since ϵ is not a good prefix, clearly a is a minimal good prefix. On the other hand, a appears as a subword in any good prefix, and therefore a is the only minimal good prefix. The set of minimal good prefixes of L is denoted $min(L)$. For an automaton $\mathcal{A}$, we denote by $min(\mathcal{A})$ the set $min(L(\mathcal{A}))$.

The decision criteria is based on the intuition that a co-checkable language has a finite set of minimal good prefixes. We would have liked to argue that this is a sufficient condition for a language to be co-checkable. This, however, is not true as can be seen by the previous example. Indeed $L = a\Sigma^\omega \cup bc^*a\Sigma^\omega$ has a finite set of minimal good prefixes (namely $\{a\}$), but is not co-checkable.

Theorem 9. *A language L is co-k-checkable iff L is co-safety, liveness, and $min(L)$ is contained in Σ^k. Dually, a language L is k-checkable iff L is safety, uninitialized, and $min(comp(L))$ is contained in Σ^k.*

Proof: We prove the characterization for co-k-checkable languages. The one for k-checkable languages is dual. Assume first that L is co-k-checkable. Then, by Theorem 7, L is co-safety and liveness. We prove that $min(L)$ is is contained in Σ^k. Let $R = \textit{co-induce}(L)$, and let $sub(R)$ denote all the subwords of words in R. We prove that $min(L) \subseteq sub(R)$. Since $R \subseteq \Sigma^k$, so is $sub(R)$, implying the same for $min(L)$.

In order to prove that $min(L) \subseteq sub(R)$, we first prove that $min(L)$ contains only finite words of length at most k. To see this, assume, towards contradiction, that a word $w = w_1 \cdots w_n$ of length $n > k$ is in $min(L)$. Consider the strict suffix u of length k of w. Thus, $u = w_{n-k+1} \cdots w_n$. If u is a good prefix for L, then w is not minimal. Otherwise, there is a word $t \in \Sigma^\omega$ such that $u \cdot t \notin L$. Since R co-induces L, it follows that $sub(u \cdot t, k) \cap R = \emptyset$. On the other hand, as w is a good prefix, we know that $w \cdot t \in L$, implying that $sub(w \cdot t, k) \cap R \neq \emptyset$. Since every k long subword of $w \cdot t$ that is not a subword of $u \cdot t$ must be a subword of $w[1..n-1]$, we get that $sub(w[1..n-1]) \cap R \neq \emptyset$. Therefore, $w[1..n-1]$ is a good prefix for L, and w is not minimal. We reached a contradiction, implying that $min(L)$ contains only words of length k or less.

Consider a word $w \in min(L)$. By the above, every word that has w as a prefix is in L. Moreover, since L is liveness, every word that has w as a subword is in L. Recall that the length of w is bounded by k. Therefore, by the maximality of R, every k-long subword that contains w as a subword is in R, and $w \in sub(R)$.

For the other direction, let L be a co-safety and liveness language with $min(L) \subseteq \Sigma^k$. Let R be the set of k-long words that contain minimal good prefixes as subwords. We prove that $L = co\text{-}check(R)$. To see that $L \subseteq co\text{-}check(R)$, note that since L is co-safety, every word $w \in L$ starts with a good prefix. Every good prefix contains a minimal good prefix as a prefix or a suffix. Therefore, w contains a minimal good prefix as a subword. Hence, w contains a word from R as a subword.

To see that $co\text{-}check(R) \subseteq L$, consider some $w \in co\text{-}check(R)$. By the definition of co-checkability, $w = x \cdot y \cdot z$ for $y \in R$. Since $y \in R$, it has a sub word y' which is in $min(L)$, and hence is a good prefix for L. Let i be the index of the first letter of y' in (the first occurrence of) y' in w. Then, $w^i \in L$ since it starts with a good prefix, and $w \in L$ since it has a suffix in L and L is liveness. □

Corollary 1. *A language L is co-checkable iff L is co-safety, liveness, and $min(L)$ is finite. Dually, a language L is checkable iff L is safety, uninitialized, and $min(comp(L))$ is finite.*

We can now use the criteria from Corollary 1 in order to decide whether a given NBW $\mathcal{A}$ is checkable or co-checkable. The most challenging part is the test for the finiteness of the set of minimal good prefixes. For the proof of the next claim we need the following definition: an NBW $\mathcal{A}$ is *co-looping* if it has a single accepting state that is a sink.

Theorem 10.

1. *For a co-safety DBW $\mathcal{A}$, deciding whether $min(\mathcal{A})$ is finite is in NLOGSPACE.*
2. *For a co-safety NBW $\mathcal{A}$, deciding whether $min(\mathcal{A})$ is finite is in PSPACE.*

Proof: The NBW case reduces to the DBW case by using the polynomial space determinization of safety and co-safety NBWs [KV01a]. As the test described below for DBW uses only nondeterministic logarithmic space, we get a polyspace test for NBW.

Given a DBW $\mathcal{A}$, our strategy is to construct a DFW $\mathcal{A}'$ whose language is $min(\mathcal{A})$, and decide whether the language of $\mathcal{A}'$ is finite. We now show how to construct $\mathcal{A}'$.

Let $\mathcal{A}$ be an n-states co-safety DBW with language $L \subseteq \Sigma^\omega$. As a first step we construct a co-looping DFW $\mathcal{A}_g$ with $O(n)$ states such that $L(\mathcal{A}_g) = good(L(\mathcal{A}))$.

We say that a state q of $\mathcal{A}$ is *universal* if $L(\mathcal{A}^q) = \Sigma^\omega$. Clearly, replacing all universal states by a single accepting sink does not change the language of $\mathcal{A}$. We denote by $\mathcal{A}'$ the resulting automaton. Let $w \in good(L)$, since $\mathcal{A}$ is deterministic, after reading w, the automaton $\mathcal{A}$ must be in the universal state. On the other hand, if $\mathcal{A}$ is in the universal state after reading some finite word $w \in \Sigma^*$, then $\mathcal{A}$ cannot leave the accepting sink so it must be that $w \in good(L)$. If we view $\mathcal{A}'$ as a DFW (rather then a DBW) we get a DFW for $good(L)$ as needed.

Note that the complexity of the construction involves the computation of the universal states set (as well as logarithmic space computations). Since a state is universal iff no cycle of non-accepting states is reachable from it, the universality of a state can be decided in non-deterministic logarithmic space.

Let $nonmin(L) = good(L) \setminus min(L)$ be the set of non-minimal good prefixes of L. Given a word $w = w_1 \cdots w_l \in good(L)$, the word w is in $nonmin(L)$ iff either of the following holds.

1. $w[2..l] \in good(L)$ (we classify such words as words of type 1), or
2. $w[1..l-1] \in good(L)$ (we classify such words as words of type 2).

We can construct a DFW $\mathcal{A}_1$, for words of type 1, by adding to $\mathcal{A}_g$ a new state q^1_{in}, making it the initial state, and adding to the transition function of $\mathcal{A}_g$ a transition from q^1_{in} to the original initial state of $\mathcal{A}_g$, labeled by all the letters in Σ. We denote the set of states of $\mathcal{A}_1$ by Q_1, and its accepting sink by q^1_{ac}.

We can also construct a DFW $\mathcal{A}_2$, for words of type 2, by adding a new accepting sink state q^2_{ac2} to $\mathcal{A}_g$, making the former accepting state q^2_{ac} non-accepting, and adding to the transition function of $\mathcal{A}_2$ a transition from q^2_{ac} to q^2_{ac2} labeled by all the letters in Σ. Note that a word w is in $good(L)$ iff a run of $\mathcal{A}_2$ on w ends in either q^2_{ac} or q^2_{ac2}.

We denote by $\mathcal{A}_{min}$ the cross product of $\mathcal{A}_1$ and $\mathcal{A}_2$, where the accepting states set is $(Q_1 \setminus \{q^1_{ac}\}) \times \{q^2_{ac}\}$. The words accepted by $\mathcal{A}_{min}$ are exactly $good(L) \setminus nonmin(L)$ as needed.

The number of state of $\mathcal{A}_{min}$ is quadratic in the number of states of the DBW $\mathcal{A}$. Since deciding DFW finiteness amounts to deciding if there exists a cycle on a path between the initial state and an accepting state, deciding finiteness of DFW is in NLOGSPACE and the complexity result follows. □

We can now combine the tests as suggested in Corollary 1.

Theorem 11. *The problem of deciding checkability is*

1. *NLOGSPACE-complete for DBWs.*
2. *PSPACE-complete for NBWs.*

Proof: For the upper bounds, we follow the characterization in Corollary 1 and check whether $L(\mathcal{A})$ is safety [AS87], uninitialized [HKKM02], and $min(comp(L))$ is finite. In order to check the latter, we construct a deterministic DBW $\tilde{\mathcal{A}}_g$ for $good(comp(L(\mathcal{A})))$. In case $\mathcal{A}$ is an NBW, we construct a DBW $\tilde{\mathcal{A}}$ for $comp(\mathcal{A})$ in polynomial space [KV01a], and can proceed as in Theorem 10.

In case $\mathcal{A}$ is DBW, we proceed as follows: We say that a state q of $\mathcal{A}$ is *useless* if $L(\mathcal{A}^q) = \emptyset$. Since $\mathcal{A}$ is deterministic, after reading a bad prefix $\mathcal{A}$ must be in a useless state. Thus, replacing all the useless states by a single sink, marking the new sink accepting, and the rest of the states non-accepting, results in a DFW $\mathcal{A}_g$ for the bad prefixes of $L(\mathcal{A})$ which are the good prefixes of $comp(\mathcal{A})$.

Note that deciding whether a state is not useless amounts to searching for a cycle with an accepting state reachable from it. Therefore, deciding uselessness in can be done in nondeterministic logarithmic space.

Once we constructed $\mathcal{A}_g$, we proceed by checking the finiteness of $min(\mathcal{A}_g)$ as in Theorem 10.

The lower bound for NBWs is proven by a standard reduction from NBW universality. The same argument, applied to DBW results in a NLOGSPACE-hardness. We proceed with the details of proofs.

Given an NBW $\mathcal{A}$, we construct an NBW $\mathcal{A}'$ such that $\mathcal{A}'$ 'is checkable iff $\mathcal{A}$ is universal. We denote the language of $\mathcal{A}$ by L and the alphabet of $\mathcal{A}$ by Σ. Let $\mathcal{A}'$ be an NBW over an alphabet Σ' such that the language L' of $\mathcal{A}'$ is not checkable.

For two words $w = w_1 w_2 \ldots \in \Sigma^\omega$ and $v = v_1 v_2 \ldots \in \Sigma'^\omega$, let $w \oplus v = \langle w_1, v_1 \rangle \cdot \langle w_2, v_2 \rangle \cdots \in (\Sigma \times \Sigma')^\omega$. It is easy to build an NBW $\mathcal{A}_\times$ over the alphabet $\Sigma \times \Sigma'$ such that $\mathcal{A}_\times$ accepts a word $w \oplus v \in (\Sigma \times \Sigma')^\omega$ iff either $\mathcal{A}$ accepts w or $\mathcal{A}'$ accepts v. We denote the language of $\mathcal{A}_\times$ by $L_\times$.

We prove that $L_\times$ is checkable iff $\mathcal{A}$ is universal. First, if $\mathcal{A}$ is universal so is $\mathcal{A}_\times$, and therefore $L_\times$ is checkable. For the other direction, assume by way of contradiction that $\mathcal{A}$ is not universal and yet $L_\times$ is k-checkable for some k. Consider a word π not in $L_\times$. We say that a word $w_\times = w \oplus w' \in (\Sigma \times \Sigma)^k$ *agrees* with π when its projection on Σ (i.e. w) is a subword of π. Let $R_\times$ be the set of words in $induce(L_\times)$ that agree with π. Let by $R' \subseteq \Sigma'^k$ be the projection of $R_\times$ on Σ'. We prove that R' induces L' in contradiction to the choice of A'.

For every $\pi' \in \Sigma'^\omega$, the word $\pi \oplus \pi' \in (\Sigma \times \Sigma')^\omega$ is in $L_\times$ iff $sub(\pi \oplus \pi', k) \subseteq induce(L_\times)$. Since $\pi \oplus \pi'$ projection on Σ is π, the question whether $sub(\pi \oplus \pi', k) \subseteq induce(L_\times)$ reduces to the question whether $sub(\pi', k) \subseteq R'$. On the other hand, since $\pi \notin L$, the definition of $\mathcal{A}_\times$ implies that $\pi \oplus \pi' \in L_\times$ iff $\pi' \in L'$. We get that $\pi' \in L'$ iff $sub(\pi', k) \subseteq R$, i.e., that R induces L'.

□

We now turn to the dual case, where we want to test an automaton for being co-checkable. Again, we use the characterization in Corollary 1.

Theorem 12. *Deciding co-checkability is*

1. *NLOGSPACE-complete for DBWs.*
2. *PSPACE-complete for NBWs.*

Proof: For the upper bounds, we apply the criteria of Corollary 1. We have to check co-safety, liveness and finiteness of $min(L)$. We start with the NBW case. To check whether $L(\mathcal{A})$ is co-safety we apply the procedure of [Sis94][4]. Checking for liveness can be done as in [AS87]; note that the procedure suggested in [AS87] can be done in polynomial space. Checking the finiteness of $min(L)$ can be done as in Theorem 10.

As for DBWs, checking for both liveness and co-safety can be done in NLOGSPACE. The [AS87] procedure for checking liveness is to construct an automaton $\mathcal{A}'$ with

[4] [Sis94] suggests a procedure for deciding and automaton is safety. We can check for co-safety by complementing $\mathcal{A}$ and checking $comp(\mathcal{A})$ for safety. While complementing $\mathcal{A}$ has an exponential blow-up, this blow-up does not increase the complexity of the decision procedure, since the decision procedure in [Sis94] already involves complementation.

the same structure as $\mathcal{A}$ in which all the sates are accepting, and then check whether the language of $\mathcal{A}'$ is Σ^ω. Note that [AS87] assume that $\mathcal{A}$ is reduced, i.e., from every state in $\mathcal{A}$ an accepting state can be reached. It is not hard to see that in the case $\mathcal{A}$ is a DBW, the entire procedure (including reducing $\mathcal{A}$) can be done in NLOGSPACE.

As for checking for co-safety: recall that a state q is *universal* if $L(\mathcal{A}^q) = \Sigma^\omega$. We construct a new DBW $\mathcal{A}'$ similar to $\mathcal{A}$ in which all universal sates are replaced by a single accepting sink, and no other state is accepting. It is not hard to see that $\mathcal{A}$ is co-safety iff $L(\mathcal{A}) = L(\mathcal{A}')$. Thus, checking whether a DBW is co-safety can be done in NLOGSPACE.

Checking for finiteness of $min(L(\mathcal{A}))$ can be done as in Theorem 10.
The lower bound for deciding NBWs co-checkability is proven by a reduction from NBW universality, using the same argument as in Theorem 13. For DBWs, NLOGSPACE-hardness can be proven by a reduction from graph non reachability.

Given a graph $G = \langle V, E\rangle$ and two vertices $s, t \in V$ we construct a DBW $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F\rangle$ that is co-checkable iff t is not reachable from s in G. Let $\mathcal{A}' = \langle \Sigma', Q', q_0', \delta', F'\rangle$ be some DFW whose language is not liveness. The alphabet Σ of $\mathcal{A}$ is $\Sigma' \cup V$. The set of states V of $\mathcal{A}$ is $V \cup Q'$ where we identify the vertex t with q_0'. The initial state of $\mathcal{A}$ is s. The transition function of $\mathcal{A}$ is defined as follows: for state $v \in V$ and letter $\sigma \in \Sigma$ let $\delta(v, \sigma) = \sigma$ if $(v, \sigma) \in E$ or $\delta(v, \sigma) = v$ otherwise. For state $s \in Q'$ and letter $\sigma \in \Sigma$ let $\delta(s, \sigma) = \delta'(s, \sigma)$ if $\sigma \in \Sigma'$ or $\delta(s, \sigma) = s$ otherwise. The accepting states set F is F'. If t is not reachable from s in G, then the language of $\mathcal{A}$ is empty, and thus co-checkable. If, on the other hand, t is reachable in G from t, we show that the language of $\mathcal{A}$ is not liveness, and therefore $\mathcal{A}$ is not checkable. Since $\mathcal{A}'$ is not liveness there exists an infinite word x in $L(\mathcal{A}')$ and a finite word $y \in \Sigma^*$ for which $y \cdot x$ is not in $L(\mathcal{A}')$. Denote by $p \in V^*$ a path from s to t in G. Clearly $p \cdot x$ is in $L(\mathcal{A})$. However, $p \cdot y \cdot p \cdot x$ is not in $L(\mathcal{A}')$. Thus, $L(\mathcal{A})$ is not liveness, and hence not co-checkable. □

We now turn to study the problem of deciding whether L is k-checkable or k-co-checkable for a given k. We describe the reasoning for the case of co-checkability. The considerations in the case of k-checkability are dual.

¿From Theorem 9 we know that finding the width of a co-checkable language L can be reduced to finding the length of its longest minimal good prefix, and deciding whether L is k-co-checkable can be reduced to checking that L is checkable yet no minimal good prefix of it is of length $k' > k$. In Theorem 10, we constructed a DFW $\mathcal{A}_{min}$ that accepts exactly all the minimal good prefixes of L. Recall that when L is co-checkable, the language of $\mathcal{A}_{min}$ is finite, so $\mathcal{A}_{min}$ contains no cycles. Hence, checking a bound on the length of accepting paths in $\mathcal{A}_{min}$ can be done nondeterministically in space that is logarithmic in the size of $\mathcal{A}_{min}$ (a longer path can be guessed). Accordingly, we have the following.

Theorem 13.

1. *Given a DBW $\mathcal{A}$ and an integer k, deciding whether $\mathcal{A}$ is k-checkable (or co-k-checkable) is NLOGSPACE-complete.*
2. *Given an DBW $\mathcal{A}$ and an integer k, deciding whether $\mathcal{A}$ is k-checkable (or co-k-checkable) is PSPACE-complete.*

By translating LTL formulas to NBWs, the results above imply an EXPSPACE upper bound to the problem of deciding whether an LTL formula is checkable or k-checkable, for a given k. We leave open the question of the tightness of this bound. As we shall see in Section 5, the fact LTL formulas can be easily complemented leads, in the case of width bound, to an upper bound that is tighter than the one obtained by going through NBWs.

5 Bounding the width

In this section we study the relations between the width of a checkable (or co-checkable) language, and the size of automata or LTL formulas for the language.

5.1 Width vs. automata size

We start with Büchi automata. At first sight, it seems that the width of a language of a checkable language can be bounded by the diameter of the smallest DBW recognizing the language. Indeed, it appears that in an accepting run, the traversal through the minimal good prefix should not contain a cycle. This impression, however, is misleading, as demonstrated in the DBW $\mathcal{A}$ from Example 1, where a traversal through the subword 120 contains a cycle. The diameter of the DBW $\mathcal{A}$ is 3, so a bound by the diameter is still possible, but remains open. As detailed below, our bound depends on the size of $\mathcal{A}$ and not only in its diameter. We start with an upper bound:

Theorem 14.

1. *For a checkable (or co-checkable) DBW $\mathcal{A}$ with n states, the width of $L(\mathcal{A})$ is bounded by $O(n^2)$.*
2. *For a checkable (or co-checkable) NBW $\mathcal{A}$ with n states, the width of $L(\mathcal{A})$ is bounded by $2^{O(n)}$.*

Proof: We consider the co-checkable case first. In the proof of Theorem 10, we constructed a DFW $\mathcal{A}_{min}$ such that $L(\mathcal{A}_{min}) = min(\mathcal{A})$. Since $min(\mathcal{A})$ is finite, no accepting run of $\mathcal{A}$ contains a cycle. Therefore, the size of $\mathcal{A}_{min}$ provides a trivial bound for the longest word in $min(\mathcal{A})$, and therefore on width $L(\mathcal{A})$.

As for the checkable case, we can consider the complement language. Since the construction of $\mathcal{A}_{min}$ already involved determinization and complementation the complexity remains the same. □

We now prove an exponential lower bounds on the gap between width and size of automata by presenting small NBWs that accept checkable and co-checkable languages of large width. The crucial observation is that an NBW with n states can "characterize" a word w of length exponential in n, in the sense it accepts all strings but w.

For natural numbers $i, n \geq 0$, we denote by $m_n(i) \in \{0,1\}^n$ the n-bit binary string encoding of i mod 2^n (e.g. $m_4(5) = 0101$). We denote by $counter(n)$ the string $m_n(0) \cdot \# \cdot m_n(1) \cdots \# \cdot m_n(2^n - 1) \cdot \#$. For example, $counter(3) =$ 000#001#010#011#100#101#110#111#. Note that the length of $counter(n)$ is

$(n+1)2^n$. The word *counter*(n) is characterized by its first $n+1$ letters (i.e. $00\cdots0\#$), and by the fact that when a letter σ is read, the letter σ' at distance $n+1$ from σ is fixed by simple rules: if σ is #, then so is σ'. If σ is 0, then σ' is 1 if all the letters between σ and the next # are 1 and is 0 otherwise. Similar rules hold if σ is 1. We refer to these rules as the $(n+1)$-distance rules.

Each of the $(n+1)$-distance rules, as well as the contents of the first $(n+1)$ letters, can be easily checked. Therefore, we can construct an NBW that accepts words that violate one of those rules (simply by guessing which rule is violated and where).

Theorem 15. *There exist an NBW $\mathcal{A}$ with $O(n)$ states such that $L(\mathcal{A})$ is k-checkable but not $(k-1)$-checkable, for $k = (n+1)2^n + 2$.*

Proof: Let Σ be $\{0, 1, \#, b, e\}$. For $n \geq 0$, let x_n be the word $b \cdot counter(n) \cdot e$, and let L be the language of all words that do not contain x_n as a subword. Thus, $L = check(\Sigma^{|x_n|} \setminus \{x_n\})$ and therefore L is $(|x_n|)$-checkable. On the other hand, every word of length $|x_n| - 1$ can be extended to a word in L, so L is not $|x_n| - 1$ checkable.

It is left to see that L can be accepted by an NBW $\mathcal{A}$ of size $O(n)$. In fact, $\mathcal{A}$ is a slight variation of the NBW that detects violations in *counter*(n). The initial state of $\mathcal{A}$ is an "idle" state that scans the input waiting for a letter b. When a letter b is read, $\mathcal{A}$ attempts to detect a violation of that ensures the word read is not *counter*(n). If such violation is detected then $\mathcal{A}$ returns to its idle state. If, on the other hand, an e is seen before a violation was detected then $\mathcal{A}$ moves to a rejecting sink. Thus, $\mathcal{A}$ is a co-checkable NBW with $O(n)$ states whose language is of width $(n+1)2^n + 2$. □

Theorem 16. *There exist an NBW $\mathcal{A}$ with $O(n^2)$ states such that $L(\mathcal{A})$ is k-co-checkable but not $(k-1)$-co-checkable, for $k = 2(n+1)2^n$.*

Proof: We prove the theorem in two steps. First, we describe a language (based on the word *counter*(n)) that is $(n+1)$-co-checkable and has an NBW of size $O(n)$. Next, we examine a small variation of the language, one that still has a small NBW accepting it, but is only k-co-checkable for k exponential in n.

For $n \geq 0$, we denote by L the language *suff*$(counter(n)^\omega)$ of all the suffixes of the word $counter(n)^\omega$. Like *counter*(n), all the words in L follow the $(n+1)$-distance rules. Furthermore, every word that begins with an $(n+1)$-long subword of *counter*(n) and follows the $(n+1)$-distance rules, is in L. Since the $(n+1)$-distance rules specify a letter by its preceding $(n+1)$ letters, these rules can be seen as a set of permissible subwords of length $(n+2)$. Therefore, L is $(n+2)$-checkable, and *comp*(L) is $(n+2)$-co-checkable. It is also not hard to see that *comp*(L) is accepted by an NBW of size $O(n)$ that looks for a violation of the $(n+1)$-distance rules, or a flaw in the first $n+1$ letters (i.e., two or more # letters appearing within the first $n+1$ letters).

We now look at a variation of *counter*(n). Let \$ be a new letter. For $n \geq 0$, we denote by $counter^\$(n)$ the word $m_n(0) \cdot \# \cdot m_n(1) \cdots \# \cdot m_n(2^n - 1) \cdot \$$ which differs from *counter*(n) only in the last letter. The word $counter^\$(n)$ is characterized by rules slightly different from the $(n+1)$-distance rules: the rules now allow \$ to appear at distance $n+1$ from #, but require that \$ is preceded by a block of n 1's.

We refer to these rules as the $(n+1)$-\$-distance rules. As in the case of $counter(n)$, there exists an NFW of size $O(n)$ that detects violations of the $(n+1)$-\$-distance rules.

Consider now the language $L' = \mathit{suff}(counter(n)^\omega)) \cup \mathit{suff}(counter^\$(n)^\omega)$, i.e., the language of all words that are either a suffix of $counter(n)^\omega$ or a suffix of $counter^\$(n)^\omega$. The crucial point is that the word is of one of these types, and therefore the letter after a block of n 1's is either always #, or always \$. We refer to a letter that appears after n 1's as a *critical* letter. Since every subword of length $2|counter(n)|-1$ contains at most one critical letter, it is impossible to enforce the uniformity of all critical letters in such a subword, thus L' is not $(2|counter(n)|-1)$-checkable. On the other hand, it is clear that L' is $2|counter(n)|$-checkable, and $comp(L')$ is $2|counter(n)|$-co-checkable.

It is left to show that $comp(L')$ has a small NBW. Note that $comp(L')$ is the intersection of $comp(\mathit{suff}(counter(n)^\omega)$ and $comp(\mathit{suff}(counter^\$(n)^\omega)$, each of which has an NBW of size $O(n)$. Therefore, by the standard construction of NBWs intersection, $comp(L')$ has an NBW $\mathcal{A}$ of size $O(n^2)$. Thus, $\mathcal{A}$ is a checkable NBW with $O(n^2)$ states whose language is of width $2(n+1)2^n$. $\square$

5.2 Width vs. formula size

We now turn to consider bounds on the width of a language in terms of an LTL formula defining the language. The main technical tool used in the proof of Theorem 16 is the fact that there is a small NBW detecting violations of the $(d+1)$-distance rules. Since these rules can be easily specified by an LTL formula of size $O(d)$, a lower bound on the width of languages of LTL formulas follows:

Theorem 17.

(1) There exists an LTL formula φ such that the language $L(\varphi)$ is checkable and of width $2^{\Omega(|\varphi|)}$.

(2) There exists an LTL formula φ such that the language $L(\varphi)$ is co-checkable and of width $2^{\Omega(|\varphi|)}$.

Note that since LTL has a negation operand, Claim (2) above follows trivially from Claim (1).

It follows that the gap between the size of an LTL formula defining a checkable language and its width might be exponential. In fact, since LTL formulas are exponentially more succinct than NBWs, Theorems 15 and 16 hint that the gap may be even doubly exponential. Nevertheless, we now show that the gap cannot be larger then an exponential.

Theorem 18. *For a checkable (or co-checkable) LTL formula φ, the width of $L(\varphi)$ is bounded by $2^{|\varphi|^3+2}$.*

Proof: Let φ be a checkable LTL formula, and let L be its language. We denote the of width of L by k. Let $\mathcal{A}$ be an n states NBW for L, and let $\tilde{\mathcal{A}}$ be an $\tilde{n}$ states NBW for $comp(L)$. The crux of the proof is the observation that while k may depend exponentially on n, it depends polynomially on $\max(n, \tilde{n})$. Since complementation in LTL is trivial, the exponential construction of an NBW for the language of an

LTL formula bounds both n and $\tilde{n}$ [VW86]. We prove, by a pumping-lemma type argument, that k is bounded by $n^2\tilde{n}+3$.

By Theorem 3, we have that $comp(L)$ is co-induced by $\Sigma^k \setminus induce(L)$. We first claim that there is a word $w \in \Sigma^{k-2}$ such that there are $\sigma, \tau \in \Sigma$, for which the following hold.

(1) $\sigma \cdot w \cdot \tau \in \Sigma^k \setminus induce(L)$.
(2) There is $\sigma' \in \Sigma$ such that $\sigma' \cdot w \cdot \tau \in induce(L)$.
(3) There is $\tau' \in \Sigma$ such that $\sigma \cdot w \cdot \tau' \in induce(L)$.

The existence of w that satisfies Conditions 1-3 follows from the minimality of k. Since $induce(L)$ contains no redundancies, we also know that

(4) There is $t' \in \Sigma^\omega$ such that $\sigma' \cdot w \cdot \tau \cdot t' \in L$.
(5) There is $t \in \Sigma^\omega$ such that $\sigma \cdot w \cdot \tau' \cdot t \in L$.

Assume by way of contradiction that $k \geq n^2\tilde{n}+3$. Consider the infinite words $u = \sigma \cdot w \cdot \tau \cdot t'$, $v = \sigma' \cdot w \cdot \tau \cdot t'$, and $p = \sigma \cdot w \cdot \tau' \cdot t$. The word u contains the subword $\sigma \cdot w \cdot \tau$, which is (Condition 1) in $\Sigma^k \setminus induce(L)$. Therefore, $u \notin L$. By Conditions 4 and 5, the words v and p are in L.

Let r_u be an accepting run of $\tilde{\mathcal{A}}$ on u, and let r_v and r_p be accepting runs of $\mathcal{A}$ on v and p, respectively. Since $k \geq n^2\tilde{n}+3$, we also have $k-2 \geq n^2\tilde{n}+1$, thus there is a triple of states s_u, s_v, and s_p, of $\tilde{\mathcal{A}}$, $\mathcal{A}$, and $\mathcal{A}$ respectively, and there is a partition of w to $x \cdot y \cdot z$, with $y \neq \epsilon$, such that the run r_u visits s_u after reading $\sigma' \cdot x$ and after reading $\sigma' \cdot x \cdot y$, the run r_v visits s_v after reading $\sigma \cdot x$ and after reading $\sigma \cdot x \cdot y$, and the run r_p visits s_p after reading $\sigma \cdot x$ and after reading $\sigma \cdot x \cdot y$,

If follows that for all $i \geq 0$, we have the following.

- $\sigma \cdot x \cdot y^i \cdot z \cdot \tau \cdot t' \notin L$.
- $\sigma' \cdot x \cdot y^i \cdot z \cdot \tau \cdot t' \in L$.
- $\sigma \cdot x \cdot y^i \cdot z \cdot \tau' \cdot t \in L$.

From the last two facts, all subwords of length k of $\sigma' \cdot x \cdot y^2 \cdot z \cdot \tau \cdot t'$ and of $\sigma \cdot x \cdot y^2 \cdot z \cdot \tau' \cdot t$ are in $induce(L)$. Hence, so are all the subwords of length k of $\sigma \cdot x \cdot y^2 \cdot z \cdot \tau \cdot t'$, contradicting the fact it is not in L. □

6 Conclusions

We defined k-checkable and locally checkable languages and studied their properties. We showed that the memory demand for monitoring k-checkable properties is independent of the number of properties checked. This advantage of checkable languages make them particularly suited to be used as specification formalism for run-time verification.

We studied the relation between locally checkable languages and other fragments of ω-regular properties and showed that safety properties, uninitialized properties, and testable properties, all strictly contain checkable properties. We considered the problem of deciding whether a specification is locally checkable, or k-checkable for a given k, and showed that both problems are PSPACE-complete. Finally, we studied the relation between the width of a checkable language and the size of an NBW or

LTL formula for the language, and showed that NBWs and LTL formulas can define checkable languages with an exponentially larger width. An interesting problem that remains open is the relation between the width of a co-checkable language and the size of a DBW for it.

References

[AJS98] M. Aagaard, R.B. Jones, and C.-J.H Seger. Combining theorem proving and trajectory evaluation in an industrial environment. In *Proc. 35th DAC*, 538–541, 1998.

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

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

[BBL98] I. Beer, S. Ben-David, and A. Landver. On-the-fly model checking of RCTL formulas. In *Proc. 10th CAV*, LNCS, 1998.

[BGHS04] H. Barringer, A. Goldberg, K. Havelund, and K. Sen. Rule-based runtime verification. In *Proc. 5th VMCAI*, LNCS 2937, pages 44–57, 2004.

[CBRZ01] E. M. Clarke, A. Bierea, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *FMSD*, 19(1):7–34, 2001.

[dR05] M. d'Amorim and G. Rosu. Efficient monitoring of omega-languages. In *Proc. 17th CAV*, LNCS, 2005.

[HKKM02] T.A. Henzinger, S.C. Krishnan, O. Kupferman, and F.Y.C. Mang. Synthesis of uninitialized systems. In *Proc. 29th ICALP*, LNCS 2380, pages 644–655, 2002. 2002.

[HR02] K. Havelund and G. Rosu. Synthesizing monitors for safety properties. In *Proc. 8th TACAS*, LNCS 2280, pages 342–356. 2002.

[Kur94] R.P. Kurshan. *Computer Aided Verification of Coordinating Processes*. Princeton Univ. Press, 1994.

[KV01a] O. Kupferman and M.Y. Vardi. Model checking of safety properties. *FMSD*, 19(3):291–314, November 2001.

[KV01b] O. Kupferman and M.Y. Vardi. On bounded specifications. In *Proc. 8th LPAR*, LNCS 2250, pages 24–38, 2001.

[MP71] R. McNaughton and S. Papert. *counter-free automata*. MIT Pres, 1971.

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

[Pnu81] A. Pnueli. The temporal semantics of concurrent programs. *TCS*, 13:45–60, 1981.

[Saf88] S. Safra. On the complexity of ω-automata. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 319–327, White Plains, October 1988.

[SB95] C.J.H. Seger and R.E. Bryant. Formal verification by symbolic evaluation of partially-ordered trajectories. *FMSD*, 6:147–189, 1995.

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

[Tho90] W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science*, pages 133–191, 1990.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st LICS*, 332–344, 1986.

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

[Wil93] Thomas Wilke. Locally threshold testable languages of infinite words. In *Proc. 10th STACS*, LNCS 665, pages 607–616, 1993.