On Bounded Specifications

Orna Kupferman
Hebrew University*

Moshe Y. Vardi
Rice University†

September 12, 2002

Abstract

Bounded model checking methodologies check the correctness of a system with respect to a given specification by examining computations of a bounded length. Results from set-theoretic topology imply that sets in Σ^ω that are both open and closed (*clopen sets*) are precisely bounded sets: membership of a word in a clopen set can be determined by examining a bounded prefix of it. Clopen sets correspond to specifications that are both safety and co-safety. In this paper we study bounded specifications from this perspective. We consider both the linear and the branching frameworks. In the linear framework, we show that when clopen specifications are given by word automata or temporal logic formulas, we can identify a bound and translate the specification to bounded formalisms such as cycle-free automata and bounded LTL. In the branching framework, we show that while clopen sets of trees with infinite branching degrees may not be bounded, we can extend the results from the linear framework to clopen specifications given by tree automata or temporal logic formulas, even for trees with infinite branching degrees. There, we can identify a bound and translate clopen specifications to cycle-free automata and modal logic. Finally, we show how our results imply that the bottom levels of the μ-calculus hierarchy coalesce.

*Address: School of Computer Science and Engineering, Jerusalem 91904, Israel. Email: orna@cs.huji.ac.il
†Address: Department of Computer Science, Houston, TX 77251-1892, U.S.A. Email: vardi@cs.rice.edu.

1 Introduction

Today's rapid development of complex and safety-critical systems requires reliable verification methods. In model checking, 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 [CGP99]. For example, we can view computations of a nonterminating system $\mathcal{S}$ as infinite words over an alphabet Σ (typically, $\Sigma = 2^{AP}$, where AP is the set of the system's atomic propositions). Then, $\mathcal{S}$ induces a language $\mathcal{L}(\mathcal{S}) \subseteq \Sigma^\omega$. Similarly, we can view a property ψ of the system as a language $\mathcal{L}(\psi) \subseteq \Sigma^\omega$ of all the computations that satisfy ψ. Verification that $\mathcal{S}$ satisfies ψ can then be reduced to checking that $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\psi)$ [Kur94, VW94].

Of special interest are properties asserting that the system always stays within some allowed region 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 [Kin94].

Model checking of general properties considers infinite computations. Indeed, $\mathcal{L}(\mathcal{S})$ and $\mathcal{L}(\psi)$ are languages in Σ^ω, and checking whether $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\psi)$ involves a search for bad cycles [VW94]. A symbolic implementation of such a search may be very expensive [HKSV97, BGS00]. On the other hand, model checking of safety properties involves a search for finite bad prefixes. Therefore, such a search considered only finite computations and is much simpler than general model-checking search [KV99]. The simplicity of the search for finite bad prefixes has motivated the development of *bounded model checking* methodologies, which consider computations of a bounded length. For example, in *SAT-based* model checking, we generate a propositional formula φ_k, for a fixed $k \geq 0$, such that φ_k is satisfiable iff the property is violated by a prefix of length k of some computation [BCC$^+$99, BCRZ99]. In *symbolic trajectory evaluation* (STE), we try to falsify the correctness of a computation by referring only to a bounded prefix of it. The method is sound but not complete: we may terminate with no answer to the model-checking problem [SB95]. While it is possible to extend both SAT-based model checking and STE to handle ω-regular properties[1], the key idea of bounded model-checking methodologies is to reason about prefixes of a bounded length.

Recall that if a safety property is violated, then there is a finite prefix along which the violation has occurred. While we know that such a bad prefix exists, we cannot in general bound its length a-priori. Moreover, it may be that no such bound exists. For example, no bound exists for the safety property Gp (p is always true). Indeed, for every $k \geq 0$, the prefix p^k can be extended to a computation that satisfies Gp and can also be extended to a computation that does not satisfy Gp. We say that a property ψ is *bounded* if there is $k \geq 0$ such that for every computation π, the satisfaction of ψ in π can be determined by observing only the prefix of length k of π. It is clear that all bounded properties are also safety properties, but, as Gp demonstrates, safety is not a sufficient condition to boundedness.

The recent developments in bounded model checking have led to growing interest in bounded properties and their power. Motivated by these developments, we set out to characterize the expressive power of bounded properties. The initial goal of this research was to lift results about

[1]In SAT-based model checking, it is possible to reason about cycles of a bounded length [BCCZ99], and in STE it is possible to combine several checks in order to reason about cycles [Yan00].

bounded sets in set-theoretic topology to results about bounded properties. Several basic topological notions have natural meaning in the context of formal languages and have been useful in studying the latter [HR86]. In particular, the study of the *Borel hierarchy* in set-theoretic topology was helpful to the study of the various types of automata on infinite words, whose acceptance conditions can be classified in terms of their location in the Borel hierarchy [Lan69, Tho90].

Let us first review some of the relevant terminology from set-theoretic topology. Consider a set X and a distance function $d : X \times X \rightarrow \mathbb{R}$ between the elements of X. For an element $x \in X$ and $\gamma \geq 0$, let $K(x, \gamma)$ be the set of elements x' such that $d(x, x') \leq \gamma$. Consider a set $S \subseteq X$. An element $x \in S$ is called an *interior element* of S if there is $\gamma > 0$ such that $K(x, \gamma) \subseteq S$. The set S is *open* if all the elements in S are interior. A set S is *closed* if $X \setminus S$ is open. So, a set S is open if every element in S has a nonempty "neighborhood" contained in S, and a set S is closed if every element not in S has a nonempty neighborhood whose intersection with S is empty. A set that is both open and close is called a *clopen* set.

A *Cantor space* consists of $X = D^\omega$, for some finite D, and d defined by $d(w, w') = \frac{1}{2^n}$, where n is the first position where w and w' differ. Thus, elements of X can be viewed as infinite words over D and two words are close to each other if they have a long common prefix. If $w = w'$, then $d(w, w') = 0$. It is known that clopen sets in Cantor space are *bounded*, where a set S is bounded if it is of the form $W \cdots D^\omega$ for some finite set $W \subseteq D^*$. Hence, clopen sets in our Cantor space correspond exactly to the bounded properties we are looking for: each clopen language $L \subseteq \Sigma^\omega$ has a bound $k \geq 0$ such that membership in L can be determined by the prefixes of length k of words in Σ^ω.

What are these clopen sets in Σ^ω? It turns out that topology has an answer to this question as well [MP89, Gum93]: recall that a language $L \subseteq \Sigma^\omega$ is a *safety* language[2] iff every $w \notin L$ has a *bad prefix* $x \in \Sigma^*$ such that for all $y \in \Sigma^\omega$, we have $x \cdot y \notin L$. A language $L \subseteq \Sigma^\omega$ is a *co-safety* language[3] iff $\Sigma^\omega \setminus L$ is a safety 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 $x \cdot y \in L$. It is not hard to see that a language $L \subseteq \Sigma^\omega$ is co-safety iff L is an open set in our Cantor space. To see that, consider a word w in a co-safety language L, and let x be a good prefix of w. All the words w' with $d(w, w') \leq \frac{1}{2^{|x|}}$ have x as their prefix, so they all belong to L. For the second direction, consider a word w in an open set L, and let $\gamma > 0$ be such that $K(w, \gamma) \subseteq L$. The prefix of w of length $\lfloor \log \frac{1}{\gamma} \rfloor$ is a good prefix for L. It follows that the clopen sets in Σ^ω, namely the bounded properties we are after, are exactly these properties that are both safety and co-safety!

While topology immediately solved our initial question about bounded properties (c.f., [Sta97]), it has led to many new questions. The properties we are interested in are not general subsets of Σ^ω. Rather, they are ω-regular, given by an automaton or a temporal-logic formula. Can we make use of this extra structure? For example, can we identify a bound for a given clopen property? Can we identify a tight bound? Once we found a bound k, can we translate clopen properties to formalisms that refer to the prefix of length k only? (We call such formalisms *bounded formalisms.*) What would be the blow-up of such a translation? Also, sometime we want to verify branching temporal properties (that is, properties that describe the whole computation tree of a system, and not its

[2]The definition of safety we consider here is given in [AS85], it coincides with the definition of limit closure defined in [Eme83], and is different from the definition in [Lam85], which also refers to the property being closed under stuttering.

[3]The term used in [MP92] is *guarantee* language.

individual computations) [Lam80, EH86]. Can we extend the results from the linear framework to the branching one?

So, the enhanced goal of this research has become the study of clopen ω-regular linear and branching properties. We start with the linear framework. We first show that for an ω-regular clopen language $L \subseteq \Sigma^\omega$, we can identify a bound. We describe two incomparable bounds. The first refers to the deterministic automaton for L and the second refers to the nondeterministic automata for L and $\Sigma^\omega \setminus L$. Using the bound, we translate L to bounded formalisms, specifically, to a cycle-free automaton and to an LTL formula whose only temporal operator is X ("next"), and we study the blow up of these translations. In particular, we show that the translation of a clopen LTL formula to a formula whose only temporal operator is X is tightly exponential (that is, exponential in both the upper and lower bound senses).

We then turn to the branching framework. The definition of safety and co-safety properties can be extended to the branching framework in a straightforward way [BFG$^+$91, KV99, MT01]. Let $\tau(\Sigma, \Upsilon)$ be the set of Σ-labeled trees with directions in Υ; that is, the trees are prefix-closed subsets of Υ^*, and each node of the tree is labeled by a letter from Σ. Intuitively, a language $L \subseteq \tau(\Sigma, \Upsilon)$ is a safety language if every tree not in L has a bad prefix (which is a tree of a finite height) all of whose extensions are not in L. A clopen tree language is a language that is both safety and co-safety. We show that a distinction should be made between trees of finite branching degrees and trees with possibly infinite branching degrees. For the first type of trees, we can prove boundedness using the same considerations as in the linear framework. On the other hand, for trees with infinite branching degrees, it is not true that general clopen tree properties are bounded! Nevertheless, when the clopen tree languages are ω-regular[4], it is possible extend the results from the linear framework to both types of trees: we are able to identify a bound and to translate clopen properties to cycle-free tree automata[5]. To obtain our result, we use *symmetric nondeterministic automata* as a novel automata-theoretic tool (symmetry has been previously applied only to alternating automata). The advantage of working with nondeterministic automata is the ability to use pumping arguments.

The understanding that ω-regular clopen tree languages are bounded enables us to show that the bottom levels of the μ-calculus expressiveness hierarchy coalesce. The *μ-calculus* is an expressive and important specification language in which formulas are built from Boolean operators, next-times modalities, and least and greatest fixed-point operators [Koz83]. μ-calculus formulas are classified according to their *alternation depth*, which is the maximal number of alternations between nested least and greatest fixed-point operators. From a practical point of view, the classification is important, as the alternation depth is the major factor in the complexity of μ-calculus model checking [EL86]. A more refined classification also distinguishes between formulas in which the outermost fixed-point operator in the nested chain is a least fixed-point operator (Σ_i formulas, where i is the alternation depth) and formulas where it is a greatest fixed-point operator (Π_i formulas). *Modal Logic* (ML) consists of μ-calculus formulas with no fixed-point operators. The

[4]A tree language is ω-regular if it can be recognized by a tree automaton. When the language contains trees with infinite branching degrees, the automata are *amorphous*, capable of handling infinite branching degrees. Several types of amorphous automata are studies in the literature. In particular, in *symmetric automata* [JW95, Wil99b], the transition function describes universal and existential requirements on the successors of the current node, it is independent of the branching degree, and can handle trees with an infinite branching degree.

[5]We assume that Υ is known; thus finite branching degrees are bounded by $|\Upsilon|$. We could have considered also trees with finite but unbounded branching degrees. As we note in the sequel, general clopen properties of such trees may not be bounded, yet ω-regular properties of such trees are bounded.

μ-calculus is more expressive than ML, and in fact, it is possible to decide, given a μ-calculus formula, whether it has and equivalent ML formula [Ott99]. Moreover, it was recently proved that the μ-calculus hierarchy is strict (i.e., there is no $d \geq 1$ such that all μ-calculus formulas can be translated to formulas of alternation depth d) [Bra98]. For several hierarchies in computer science, even strict ones, it is possible to show local *coalescence*, where membership in some class of the hierarchy and in its complementary class implies membership in a lower class. For example, the equation RE$\cap$co-RE=Recursive implies a coalescence at the bottom of the arithmetical hierarchy.[6] It is shown in [BM78] that if a property describing classes of structures can be expressed both as a least fixed-point and a greatest fixed-point of a first-order formula, then it is equivalent to a first-order formula with no fixed points. Since first-order formulas that are preserved under bisimulation are equivalent to ML formulas [BB93, Ben91], the result in [BM78] implies a coalescence at the bottom of the μ-calculus expressiveness hierarchy, namely $\Sigma_1 \cap \Pi_1 = ML$. Using the fact that μ-calculus formulas of alternation depth 1 induce either safety or co-safety symmetric languages, we are able to get a constructive proof to the above coalescence, and to extend it to finite structures.[7]

2 Preliminaries

2.1 Safety and co-safety languages

Consider a language $L \subseteq \Sigma^\omega$ of infinite words over the alphabet Σ. A finite word $x \in \Sigma^*$ is a *bad prefix* for L if for all $y \in \Sigma^\omega$, we have $x \cdot y \notin L$. Thus, a bad prefix is a finite word that cannot be extended to an infinite word in L. Note that if x is a bad prefix, then all the finite extensions of x are also bad prefixes. We say that a bad prefix x is *minimal* if all the strict prefixes of x are not bad. A language L is a *safety* language iff every $w \notin L$ has a finite bad prefix. For a safety language L, we denote by $\mathit{bad\_pref}(L)$ the set of all bad prefixes for L.

For a language $L \subseteq \Sigma^\omega$, we use $\mathit{comp}(L)$ to denote the complement of L; i.e., $\mathit{comp}(L) = \Sigma^\omega \setminus L$. We say that a language $L \subseteq \Sigma^\omega$ is a *co-safety* language iff $\mathit{comp}(L)$ is a safety 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 $\mathit{good\_pref}(L)$ the set of good prefixes for L. Note that $\mathit{good\_pref}(L) = \mathit{bad\_pref}(\mathit{comp}(L))$.

For a set Υ of directions, an Υ*-tree* is a nonempty set $T \subseteq \Upsilon^*$, where for every $x \cdot \upsilon \in T$ with $x \in \Upsilon^*$ and $\upsilon \in \Upsilon$, we have $x \in T$. The elements of T are called *nodes*, and the empty word ε is the *root* of T. For every $x \in T$, the nodes $x \cdot \upsilon \in T$ where $\upsilon \in \Upsilon$ are the *children* of x. A node with no children is a *leaf*. The *degree* of a node x, denoted $\mathit{deg}(x)$, is the number of children x has. Note that $\mathit{deg}(x) \leq |\Upsilon|$. We assume that Υ is finite or $\Upsilon = \mathbb{N}$, in which case we may have nodes with an infinite degree. The *height* of a tree T is the length (possibly ∞) of the longest node in T. Note that when $\Upsilon = \mathbb{N}$, there may be infinite trees with a finite height. A tree is *leafless* iff it has no leaves. Note that a tree may be infinite and still have leaves. A *path* π of a tree T is a set $\pi \subseteq T$ such that $\varepsilon \in \pi$ and for every $x \in \pi$, either x is a leaf, or there exists a unique $\upsilon \in \Upsilon$ such that $x \cdot \upsilon \in \pi$. For an integer d, a d-Υ*-cone* is *an*Υ-tree all of whose paths are finite and have

[6] On the other hand, the analogous coalescence for the polynomial hierarchy is not known. It is a major open question whether NP$\cap$co-NP=PTIME [GJ79]

[7] The result in [BM78] appeals to the Compactness Theorem, so it is not constructive and does not carry over to finite structures.

leaves in Υ^d. An Υ-cone is a d-Υ-cone for some d. For a tree T with a finite height, an *extension* of T is a tree T' such that $T \subseteq T'$ and every node $z \in T' \setminus T$ has a leaf x of T for which $z = x \cdot y$ for some $y \in \Upsilon^+$.

Given a finite set Σ, a Σ-*labeled* Υ-*tree* is a pair $\langle T, V \rangle$ where T is an Υ-tree and $V : T \to \Sigma$ maps each node of T to a letter in Σ. We use $\tau(\Sigma, \Upsilon)$ to denote the set of all Σ-labeled leafless Υ-trees. For a language $L \subseteq \tau(\Sigma, \Upsilon)$, we use $comp(L)$ to denote the complement of L; i.e., $comp(L) = \tau(\Sigma, \Upsilon) \setminus L$.

Consider an Υ-tree $T \subseteq \Upsilon^*$. A prefix of T is a nonempty prefix-closed subset of T with a finite height. A prefix of a tree $\langle T, V \rangle \in \tau(\Sigma, \Upsilon)$ is a Σ-labeled Υ-tree $\langle P, V \rangle$, where P is a prefix of T. An *extension* of $\langle P, V \rangle$ is a tree $\langle T', V' \rangle \in \tau(\Sigma, \Upsilon)$ such that T' is an extension of P, and V and V' agree on the labels of the nodes in P. We say that a language $L \in \tau(\Sigma_\Upsilon)$ is a safety language if every tree $\langle T, V \rangle \notin L$ has a prefix $\langle P, V \rangle$ all whose extensions are not in L. The prefix $\langle P, V \rangle$ is then a *bad prefix* for L. Dually, $L \in \tau(\Sigma, \Upsilon)$ is co-safety if every tree $\langle T, V \rangle$ in L has a prefix $\langle P, V \rangle$ all whose extensions are in L. The prefix $\langle P, V \rangle$ is then a *good prefix* for L. Note that, as in the linear case, $L \subseteq \tau(\Sigma, \Upsilon)$ is co-safety iff $comp(L)$ is safety. A *cone prefix* is a prefix which is a cone. Since prefixes have a finite height, it is easy to see that each bad prefix induces a bad cone prefix, and similarly for good prefixes.

In both the linear and the branching frameworks, we say that a language is *clopen* if it is both safety and co-safety (or, equivalently, if both L and $comp(L)$ are safety.) Note that the set of clopen languages is closed under complementation. For a temporal logic formula ψ, we say that ψ is a *safety formula* iff the set of words/trees that satisfy ψ is a safety language. Similarly, ψ is a *co-safety formula* iff this set is a co-safety language, or, equivalently, $\neg\psi$ is a safety formula. For an LTL formula ψ over a set AP of atomic propositions, let $\|\psi\|$ denote the set of computations in $(2^{AP})^\omega$ that satisfy ψ. Similarly, for a CTL* formula over AP, and a set Υ of directions, let $\|\psi\|_\Upsilon$ denote the set of computation trees in $\tau((2^{AP}), \Upsilon)$ that satisfy ψ. We say that ψ is a *safety formula* iff $\|\psi\|_\Upsilon$ is a safety language for all Υ. Also, ψ is a *co-safety formula* iff $\|\psi\|_\Upsilon$ is a co-safety language for all Υ or, equivalently, $\neg\psi$ is a safety formula.

2.2 Automata

Let Σ be a finite alphabet. For a word $w = \sigma_0 \cdot \sigma_1 \cdots$ over Σ and integers i and j, we use $w[i, \ldots, j]$ to denote the infix (possibly prefix of suffix) $\sigma_i \cdots \sigma_j$ of w. A *looping word automaton* is $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0 \rangle$, where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, and $Q_0 \subseteq Q$ is a set of initial states. The set Q contains a state q_{acc} designated as an *accepting sink*. If $|Q_0| = 1$ and δ is such that for every $q \in Q$ and $\sigma \in \Sigma$, we have that $|\delta(q, \sigma)| \leq 1$, then $\mathcal{A}$ is a *deterministic* automaton.

Given an input word $w = \sigma_0 \cdot \sigma_1 \cdots$ in Σ^ω, a *run* of $\mathcal{A}$ on σ is a function $r : \mathbb{N} \to Q$ where $r(0) \in Q_0$ and for every $i \geq 0$, either $r(i) = q_{acc}$ or $r(i+1) \in \delta(r(i), \sigma_i)$; i.e., the run starts in one of the initial states and obeys the transition function. Once the run reaches the accepting sink, its continuation is not important. Note that a nondeterministic automaton can have many runs on σ. In contrast, a deterministic automaton has a single run on σ. An automaton $\mathcal{A}$ accepts an input word w iff there exists a run of $\mathcal{A}$ on w.

Amorphous tree automata run on Σ-labeled Υ-trees. A *looping tree automaton* is $\mathcal{A} = \langle \Sigma, \Upsilon, Q, \delta, Q_0 \rangle$, where Σ, Q, and Q_0, are as in Büchi word automata (in particular, Q contains an accepting sink q_{acc}), and $\delta : Q \times \Sigma \times \{1, \ldots, |\Upsilon|\} \to 2^{Q^*}$ is a (nondeterministic) transition function, with $\delta(q, \sigma, k) \subseteq Q^k$. When Υ is infinite and $k = \mathbb{N}$, the tuples in Q^k are represented in some succinct way. We will return to this point shortly. Intuitively, in each of its transitions, $\mathcal{A}$ splits into several copies. Each copy proceeds to a subtree of the current node. A k-tuple $\langle q_1, q_2, \ldots, q_k \rangle \in \delta(q, \sigma, k)$ means that if $\mathcal{A}$ is now in state q and it reads a node of degree k labeled by σ, then a possible transition is one in which the copy that proceeds to direction leftmost subtree moves to state q_1, the copy that proceeds to the subtree near the leftmost one moves to state state q_1, and so on. A *run* of $\mathcal{A}$ on an input Σ-labeled Υ-tree $\langle T, V \rangle$ is a Q-labeled Υ tree $\langle T, r \rangle$ such that $r(\varepsilon) \in Q_0$ and for every $x \in T$ with successors $x \cdot \upsilon_1, x \cdot \upsilon_2, \ldots, x \cdot \upsilon_{deg(x)}$ in T, either $r(x) = q_{acc}$ or $\langle r(x \cdot \upsilon_1), r(x \cdot \upsilon_2), \ldots, r(x \cdot \upsilon_{deg(x)}) \rangle \in \delta(r(x), V(x), deg(x))$. If, for instance, $\Upsilon = \{0, 1\}$, $r(0) = q_2$, $V(0) = a$, $deg(x) = 2$, and $\delta(q_2, a, 2) = \{\langle q_1, q_2 \rangle, \langle q_4, q_5 \rangle\}$, then either $r(0 \cdot 0) = q_1$ and $r(0 \cdot 1) = q_2$, or $r(0 \cdot 0) = q_4$ and $r(0 \cdot 1) = q_5$. An automaton $\mathcal{A}$ accepts $\langle T, V \rangle$ iff there exists an run of $\mathcal{A}$ on $\langle T, V \rangle$.

Recall that amorphous automata are capable of reading trees with infinite branching degrees. For that, the tuples in the transition function are described in some succinct way. To handle trees with arbitrary branching degree, we introduce here *symmetric* nondeterministic looping automata, which are the nondeterministic counterpart of symmetric alternating looping automata [JW95, Wil99b]. In a symmetric looping automaton $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0 \rangle$, the state space is $Q = 2^S$ for some set S of *underlying states*, and the transition function $\delta : Q \times \Sigma \to 2^{Q \times Q}$ maps a state and a letter to sets of pairs $\langle S_U, S_E \rangle$ of subsets of S. The set $S_U \subseteq S$ is the *universal set* and it describes the underlying states that should be members in all the successor states. The set $S_E \subseteq S$ is the *existential set* and it describes underlying states each of which has to be a member in at least one successor state. Formally, the tuples induced by $\delta(q, \sigma)$ are $\langle S_1, \ldots, S_k \rangle$ for an arbitrary k (possibly $k = \omega$), such that there is $\langle S_U, S_E \rangle$ in $\delta(q, \sigma)$ such that for all $1 \leq i \leq k$ we have $S_U \subseteq S_i$ and for all $s \in S_E$ there is $1 \leq i \leq k$ such that $s \in S_i$. Runs of symmetric automata are then defined as runs of usual nondeterministic automata, by means of the tuples induced by the transition function. A language is *symmetric* if there is a nondeterministic symmetric automaton recognizing it[8].

We now define a bounded form of automata. These are automata that essentially read only a bounded prefix of their input. A word automaton $\mathcal{A}$ is a *cycle-free* automaton if it contains no cycles. Formally, $\mathcal{A}$ is cycle free if for all runs r of $\mathcal{A}$ and two positions $i, j \geq 0$, either $r(i) \neq r(j)$ or $r(i) = q_{acc}$. When $\mathcal{A}$ is a tree automaton, it is cycle free if for all runs $\langle T, r \rangle$ of $\mathcal{A}$ and two nodes $x, y \in T$ such that x is a prefix of y, we have $r(x) \neq r(y)$ or $r(x) = q_{acc}$. The *diameter* of a deterministic cycle-free word automaton $\mathcal{A}$, denoted $diameter(\mathcal{A})$, is the length of the longest path from the initial state of $\mathcal{A}$ to the accepting sink of $\mathcal{A}$.

For both word and tree automata, we use $\mathcal{A}^q$, for $q \in Q$, to denote the automaton $\mathcal{A}$ with initial set $\{q\}$. The language $\mathcal{L}(\mathcal{A})$ is the set of words/trees accepted by $\mathcal{A}$. We say that an automaton $\mathcal{A}$ is a clopen automaton if $\mathcal{L}(\mathcal{A})$ is clopen. As shown in [AS87, BFG$^+$91, Sis94, KV99], looping automata recognize safety languages. Moreover, if a safety language L is recognized by an automaton $\mathcal{A}$ with an acceptance condition such as Büchi, Rabin, etc., then L is also recognized by the looping automaton obtained by ignoring the acceptance condition. It follows that every safety ω-regular word language can be recognized by a looping automaton. Note that looping word automata can be determinized by an application of the standard *subset construction* [RS59], thus

[8]Readers familiar with alternating automata can see that symmetric nondeterministic looping automata are essentially symmetric alternating looping automata with transitions in DNF.

a safety ω-regular word language can be recognized by a deterministic looping word automaton.

3 Clopen Properties: The Linear Framework

In this section we study linear clopen properties. We first give a direct proof, independent of set-theoretic topology, that linear clopen properties are bounded. We then consider ω-regular clopen properties, obtain two bounds for them, and consider their translation to bounded formalisms.

Consider a clopen language $L \subseteq \Sigma^\omega$. For a finite word $x \in \Sigma^*$, we say that x is *determined* with respect to L if x is a bad or a good prefix for L. Accordingly, x is *undetermined* with respect to L if there are $y \in \Sigma^\omega$ and $z \in \Sigma^\omega$ such that $x \cdot y \in L$ and $x \cdot z \notin L$.

Lemma 3.1 *If $L \subseteq \Sigma^\omega$ is clopen, then every word $w \in \Sigma^\omega$ has only finitely many prefixes that are undetermined with respect to L.*

Proof: Consider a word $w \in \Sigma^\omega$. If $w \notin L$, let x be the minimal bad prefix of w. Since L is a safety language, such a prefix exists. Clearly, w has $|x|$ undetermined prefixes (exactly all the strict prefixes of x). Similarly, if $w \in L$, we take x to be the minimal good prefix of w. Again, w has $|x|$ undetermined prefixes. □

We say that a clopen language L is *bounded* if there are only finitely many words in Σ^* that are undetermined with respect to L. For an integer k, we say that L is *bounded by k* if all the words $x \in \Sigma^*$ such that $|x| \geq k$ are determined with respect to L. Note that each bounded language L has an integer k such that L is bounded by k.

Theorem 3.2 [Sta97] *All clopen languages $L \subseteq \Sigma^\omega$ are bounded.*

Proof: Assume by way of contradiction that there is a clopen language $L \subseteq \Sigma^\omega$ that is not bounded. Thus, there are infinitely many $x \in \Sigma^*$ such that x is undetermined with respect to L. Since Σ is finite and the set of undetermined words is prefix closed, it follows, by König's Lemma, that there is an infinite word in Σ^ω all of whose prefixes are undetermined. This, however, contradicts Lemma 3.1. □

As discussed in Section 1, the clopen requirement is essential. Indeed, the language induced by the safety property Gp is not bounded, and so is the language induced by the co-safety property $F\neg p$.

Theorem 3.2 applies to languages L that are not necessarily ω-regular. We now show that when L is ω-regular, it is possible to obtain a bound for L. We first present a bound that refer to this deterministic automaton, and then use some observations about automata to present a bound that refer to the nondeterministic automata for L and $comp(L)$.

For a clopen ω-regular language L, let $det(L)$ be the size of the minimal deterministic looping automaton that recognizes L. Note that deterministic looping automata can be minimized in a way that is analogous to way deterministic automata on finite words are minimized. Furthermore, there is unique minimal deterministic automaton for L.

Lemma 3.3 *All minimal deterministic looping automata that recognize a clopen language are cycle free.*

Proof: Consider a minimal deterministic looping automaton $\mathcal{A} = \langle \Sigma, Q, \delta, \{q_0\}, Q\rangle$ such that $\mathcal{L}(\mathcal{A})$ is clopen. Note that since $\mathcal{A}$ is minimal and deterministic, then $\mathcal{A}$ has at most one accepting sink q_{acc}, and all the states $q \neq q_{acc}$ are such that $\mathcal{L}(\mathcal{A}^q) \neq \Sigma^\omega$. Assume by way of contradiction that $\mathcal{A}$ is not cycle free. Then there is a state $q \neq q_{acc}$ and two words x and y in Σ^* such that $\delta(q_0, x) = q$ and $\delta(q, y) = q$. Let $z \in \Sigma^\omega$ be such that $z \notin \mathcal{L}(\mathcal{A}^q)$. Since $\mathcal{A}$ is a looping automaton, the word $x \cdot y^\omega$ is accepted by $\mathcal{A}$. Also, since $\mathcal{L}(\mathcal{A})$ is clopen, there is $i \geq 0$ such that $x \cdot y^i$ is a good prefix for $\mathcal{L}(\mathcal{A})$. On the other hand, since $\mathcal{A}$ is deterministic, $x \cdot y^i \cdot z$ is not accepted by $\mathcal{A}$, contradicting the fact that $x \cdot y^i$ is a good prefix. $\square$

So, the minimal deterministic looping automaton that recognizes a clopen language L is cycle free. Thus, we can talk about diameters of clopen ω-regular languages and define $diameter(L)$ as the diameter of the minimal automaton for L.

Lemma 3.4 *A clopen ω-regular language $L \subseteq \Sigma^\omega$ is bounded by $diameter(L)$.*

Proof: Let $\mathcal{A}$ be the minimal deterministic looping automaton for L. Consider a word $x \in \Sigma^*$ with $|x| \geq diameter(L)$. By Lemma 3.3, $\mathcal{A}$ is cycle free. Hence, the run of $\mathcal{A}$ on x either reaches an accepting sink, in which case x is a good prefix, or it does not reach an accepting sink, in which case, by the definition of $diameter(\mathcal{A})$, we cannot extend x to a word accepted by $\mathcal{A}$, thus x is a bad prefix. $\square$

It follows that a clopen ω-regular language $L \subseteq \Sigma^\omega$ is bounded by $diameter(L)$ and can be recognized by a cycle-free automaton with $det(L)$ states. The weakness of these immediate bounds is that they refer to a deterministic automaton for L, which may be exponentially bigger than a nondeterministic automaton for it. We now show it is possible to obtain a bound of a clopen ω-regular language that refers to nondeterministic automata for L and $comp(L)$.

For safety language L, the *in index* of L, denoted $in\_index(L)$, is the minimal number of states that a nondeterministic looping automaton recognizing L has. Similarly, the *out index* of a co-safety language L, denoted $out\_index(L)$, is the minimal number of states that a nondeterministic looping automaton recognizing $comp(L)$ has. If L is clopen, we also refer to the *index* of L, denoted $index(L)$, which is the product of the in and out indices of L.

Lemma 3.5 *A clopen ω-regular language $L \subseteq \Sigma^\omega$ is bounded by $index(L)$.*

Proof: Assume by way of contradiction that there is a word $x \in \Sigma^*$ such that $|x| \geq index(L)$ and x is undetermined with respect to L. Thus, there are suffixes y and z such that $x \cdot y \in L$ and $x \cdot z \notin L$. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be nondeterministic looping automata such that $\mathcal{L}(\mathcal{A}_1) = L$, $\mathcal{L}(\mathcal{A}_2) = comp(L)$, and $\mathcal{A}_1$ and $\mathcal{A}_2$ have $in\_index(L)$ and $out\_index(L)$ states, respectively. Consider two accepting runs r_1 and r_2 of $\mathcal{A}_1$ and $\mathcal{A}_2$ on $x \cdot y$ and $x \cdot z$, respectively. Since $|x| \geq in\_index(L) \cdot out\_index(L)$, there are two prefixes $x[1, \ldots, i]$ and $x[1, \ldots, j]$ of x such that $i < j$ and both runs repeat their state after these two prefixes; i.e., $r_1(i) = r_1(j)$ and $r_2(i) = r_2(j)$. Consider the word $x' = x[1, \ldots, i] \cdot x[i+1, \ldots, j]^\omega$.

Since $\mathcal{A}_1$ is a looping automaton, the run r_1 induces an accepting run r'_1 of $\mathcal{A}_1$ on x'. Formally, for all $l \leq i$ we have $r'_1(l) = r_1(l)$ and for all $l > i$, we have $r'_1(l) = r_1(i + ((l-i) \bmod (j-i)))$. Similarly, the run r_2 induces an accepting run of $\mathcal{A}_2$ on x'. It follows that x' is accepted by both $\mathcal{A}_1$ and $\mathcal{A}_2$, contradicting the fact that $\mathcal{L}(\mathcal{A}_2) = comp(\mathcal{L}(\mathcal{A}_1))$. $\square$

Theorem 3.6 *If an ω-regular language $L \subseteq \Sigma^\omega$ is clopen, then there is a cycle-free nondeterministic word automaton of size $in\_index(L) \cdot index(L)$ that recognizes L.*

Proof: Let $\mathcal{A}$ be a looping automaton for L. Consider a word $w \in \Sigma^\omega$. If w has a prefix x that cannot be extended to a word accepted by $\mathcal{A}$, then x is a bad prefix and $w \notin L$. On the other hand, if w has a prefix x with $|x| = index(L)$ and x can be extended to a word accepted by $\mathcal{A}$, then $w \in L$. Indeed, by Theorem 3.5, x is determined, so it must be a good prefix. It follows that $w \in L$ iff $\mathcal{A}$ can proceed over the prefix of length $index(L)$ of w without getting stuck. Accordingly, we define the cycle-free automaton for L as follows. Let $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0 \rangle$. We define $\mathcal{A}' = \langle \Sigma, (Q \times \{1, \ldots, index(L)\}) \cup \{q_{acc}\}, \delta', Q_0 \times \{1\} \rangle$, where for all $q \in Q$, $i \in \{1, \ldots, index(L) - 1\}$, and $\sigma \in \Sigma$, the set $\delta'(\langle q, i \rangle, \sigma)$ is obtained from $\delta(q, \sigma)$ by replacing a state $s \neq q_{acc}$ by the state $\langle s, i+1 \rangle$. Also, $\delta'(\langle q, index(L) \rangle, \sigma) = q_{acc}$. $\square$

Since LTL formulas can be translated to nondeterministic Büchi word automata with an exponential blow up [VW94], it follows from Theorem 3.6 that clopen LTL formulas can be translated to nondeterministic cycle-free automata with an exponential blow up. LTL formulas can also be translated to alternating Büchi word automata, with only a linear blow up [Var96]. Can clopen LTL formulas then be translated to alternating cycle-free automata with a linear blow up? In Theorem 3.7 below we answer this question to the negative. The idea is that a cycle-free automaton, even an alternating one, needs to visit k different states in order to read the k-th letter of the input.

Theorem 3.7 *The translation of clopen LTL formulas or clopen alternating word automata to to nondeterministic or alternating cycle-free word automata is tightly exponential.*

Proof (sketch): As discussed above, the upper bound follows from Theorem 3.6 and the exponential translation of LTL formulas to nondeterministic looping automata [VW94, Sis94]. For the lower bound, we describe a clopen property that can be specified by a short LTL formula but requires a large cycle-free alternating automaton. Let $\Sigma = \{0, 1, \#, \$\}$. For $n \geq 1$, we define the language $\mathcal{L}_n$ of all words over Σ that have in their prefix a description of an n-bit counter that counts from 0 to $2^n - 1$ (the letter # is used to separate successive values of the counter, and the letter \$ is used to separate the counter from the suffix of the word). For example,

$$\mathcal{L}_3 = 000\#001\#010\#011\#100\#101\#110\#111\$(0 + 1 + \# + \$)^\omega.$$

Clearly, $\mathcal{L}_n$ is a clopen language. It can be shown that $\mathcal{L}_n$ can be specified by an LTL formula of length $O(n)$. Essentially, the formula checks that as long as we are in the counting prefix of the word (that is, before the first \$), then whenever we see a letter $b \in \{0, 1\}$, the letter at distance $n + 1$ matches our expectations; i.e., b changes its value iff it is the last bit before the next #, or that all the bits up to the next # are 1's. In addition, the word should start with a sequence of n 0's, and the #'s and the \$ should be labeled appropriately — all these are easy to check.

On the other hand, since the membership of a word in $\mathcal{L}_n$ depends on all its first $2^n(n+1)$ letters, a cycle-free automaton has to proceed over all the first $2^n(n+1)$ letters. Being cycle free, it must have at least $2^n(n+1)+1$ states to do so. Formally, the run tree of a cycle-free alternating automaton with m states has nodes of height at most m, thus it can read only the first $m-1$ letters of the input word. In fact, just checking that the first \$ is located at position $2^n(n+1)$ requires that many states. $\square$

The translation of LTL formulas or alternating word automata to nondeterministic word automata involves an exponential blow up [MH84, VW94]. Hence, the cost of cycle-freeness is reflected only in the exponent, which can be three times larger in the cycle-free case. We note that, as with the translation of LTL formulas to nondeterministic automata, the exponential blow up refers to the worst case, and it does rarely appear in practice.

It is shown in [KV99] that the blow-up going from a co-safety LTL formula to a nondeterministic word automaton recognizing its good prefixes involves a doubly exponential blow up. Thus, for every n, there is a co-safety formula ψ of size $O(n)$ such that the minimal automaton for $good\_pref(\psi)$ has $2^{2^{O(n)}}$ states. The proof in [KV99] can be extended to ψ that is clopen. Hence, while we are able to translate a clopen LTL formula to a cycle-free automaton with an exponential blow up, we cannot expect an exponential translation of clopen LTL formulas to a cycle-free automaton for its good prefixes.

Bounded LTL is a fragment of the linear temporal logic LTL in which the only temporal operator is X ("next"). Since nondeterministic cycle-free word automata can be linearly translated to bounded LTL [Wil99a], it follows that clopen LTL formulas can be exponential translated to bounded LTL.

Proof: Let $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0 \rangle$ be a nondeterministic cycle-free word automaton. With each state q of $\mathcal{A}$ and letter σ, we associate a bounded LTL formula $\Psi(q,\sigma)$. The formula $\Psi(q)$ associated with q is then the disjunction $\bigvee_{\sigma\in\Sigma} \Psi(q,\sigma)$. For the accepting sink q_{acc}, we define $\Psi(q_{acc}) = \mathbf{true}$. For all other $q \in Q$ and $\sigma \in \Sigma$, we define $\Psi(q,\sigma)$ as follows.

- If $\delta(q,\sigma) = \emptyset$, then $\Psi(q,\sigma) = \mathbf{false}$.
- If $\delta(q,\sigma) = \{q_1,\ldots,q_l\}$, then $\Psi(q,\sigma) = \sigma \wedge \bigvee_{1\leq i\leq l} X\Psi(q_i)$.

Since $\mathcal{A}$ is cycle-free, the definition is well founded. Finally, the formula $\Psi(\mathcal{A})$ of $\mathcal{A}$ is the disjunction $\bigvee_{q\in Q_0} \Psi(q)$. When represented by a DAG, the length of $\Psi(q,\sigma)$ of linear in the size of $\delta(q,\sigma)$. Hence, the overall blow up is linear. $\square$

4 Clopen Properties: The Branching Framework

In this section we study clopen branching properties. We first restrict attention to the case Υ is finite. We show that in this case, we can prove boundedness using the same considerations as in the linear framework. On the other hand, when Υ is infinite and the trees may have infinite branching degrees, it is not true that general clopen tree properties are bounded. We then show that for clopen ω-regular tree languages, it is possible extend the results from the linear framework to both types of trees: we are able to identify a bound and to translate clopen properties to cycle-free tree automata.

Consider a clopen language $L \subseteq \tau(\Sigma, \Upsilon)$. For a Σ-labeled Υ-tree $\langle P, V\rangle$ with a finite height, we say that $\langle P, V\rangle$ is *determined* with respect to L if it is a bad or a good prefix for L. Accordingly, $\langle P, V\rangle$ is *undetermined* with respect to L if there are extensions $\langle T_1, V_1\rangle$ and $\langle T_2, V_2\rangle$ of $\langle P, V\rangle$ such that $\langle T_1, V_1\rangle \in L$ and $\langle T_2, V_2\rangle \notin L$. We say that a clopen language $L \in \tau(\Sigma, \Upsilon)$ is *bounded* if there are only finitely many integers $d \geq 0$ for which there is a Σ-labeled d-Υ-cone that is undetermined with respect to L. For an integer k, we say that L is *bounded by* k if all the Σ-labeled d-Υ-cones, with $d \geq k$, are determined with respect to L.

Lemma 4.1 *If $L \subseteq \tau(\Sigma, \Upsilon)$ is clopen, then every tree in $\tau(\Sigma, \Upsilon)$ has only finitely many cone prefixes that are undetermined with respect to L.*

Proof: Consider a tree $\langle T, V\rangle \in \tau(\Sigma, \Upsilon)$. If $\langle T, V\rangle \notin L$, let d be such that the minimal bad cone prefix of $\langle T, V\rangle$ is a d-Υ-cone. Since L is a safety language, such d exists. Clearly, $\langle T, V\rangle$ has d undetermined cone prefixes (all the c-Υ-cones, for $0 \leq c < d$). Similarly, if $\langle T, V\rangle \in L$, we take d so that the minimal good cone prefix of $\langle T, V\rangle$ is a d-Υ-cone. Again, $\langle T, V\rangle$ has d undetermined prefixes. $\square$

When Υ is finite, König's Lemma can be applied, as in Section 3, in order to prove that clopen tree languages are bounded.

Theorem 4.2 *Let Υ be a finite set of directions. All clopen languages $L \subseteq \tau(\Sigma, \Upsilon)$ are bounded.*

Proof: Assume by way of contradiction that there is a clopen language $L \subseteq \tau(\Sigma, \Upsilon)$ that is not bounded. Thus, there are infinitely many integers d such that there is a d-Υ-cone that is undetermined with respect to L. The set of undetermined Υ-cones is prefix closed in the sense that if a $(d+1)$-Υ-cone is undetermined, so is its d-Υ-cone prefix. Hence, we can arrange the set of undetermined Υ-cones in a tree (it is really a forest, but we can connect all its trees by a new root), where the i'th level of the tree contains all the undetermined i-Υ-cones, and the successors of a node in level i are all its possible extensions to an undetermined $(i+1)$-Υ-cone. Since Υ and Σ are finite, so is the branching degree of the tree. It follows, by König's Lemma, that there is an infinite path in the tree, corresponding to tree in $\tau(\Sigma, \Upsilon)$ all of whose cone prefixes are undetermined. This, however, contradicts Lemma 4.1. $\square$

We now turn to consider $\mathbb{N}$-trees. Recall that $\mathbb{N}$-trees may have nodes with an infinite degree. While Lemma 4.1 does not depend on Υ being finite, the finiteness of Υ is crucial for the application of König's Lemma in the proof of Theorem 4.2. In fact, as we prove in Theorem 4.3 below, it is not true that all clopen languages in $\tau(\Sigma, \mathbb{N})$ are bounded.

Theorem 4.3 *There is a clopen language in $\tau(\Sigma, \mathbb{N})$ that is not bounded.*

Proof: Consider the language $L \subseteq \tau(\{0,1\}, \mathbb{N})$, where $\langle T, V\rangle \in L$ iff either $deg(\varepsilon) = \infty$, or there is a node x with $|x| \leq deg(\varepsilon)$ and $V(x) = 1$. Thus, L contains exactly all $\{0,1\}$-labeled $\mathbb{N}$-trees in which, if the root has a finite degree k, then a node labeled 1 is reachable within k steps. It is not hard to see that L is clopen: if $\langle T, V\rangle \notin L$, its $deg(\varepsilon)$-$\mathbb{N}$-cone prefix is a bad prefix. Also, if $\langle T, V\rangle \in L$, either its 1-$\mathbb{N}$-cone prefix or its $deg(\varepsilon)$-$\mathbb{N}$-cone prefix is a good prefix. Now, for every a candidate bound k for L, the tree $\langle T, V\rangle$ with $deg(\varepsilon) = k+1$ and $V(x) = 0$ for all $x \in T$ has an undetermined k-$\mathbb{N}$-cone prefix. Hence, L is not bounded.[9] $\square$

The language L examined in Theorem 4.3 is quite unnatural. In practice, we are concerned with ω-regular languages. In Section 3 we show that in the linear framework, ω-regularity enables us to identify a bound for the clopen language. We now show that in the branching framework ω-regularity enables us not only to identify a bound but also to handle $\mathbb{N}$-trees, where the branching degree is infinite or finite but unbounded. We first need some notations.

A *frontier* of an Υ-tree T is a set $E \subseteq T$ of nodes such that for every path $\pi \subseteq T$, we have $|\pi \cap E| = 1$. For a node x and a frontier E, we say that $x \leq E$ if x is a prefix of some node in E. A *roof* of a frontier E is a set E' such that every node in E has a strict prefix in E'. For a Σ-labeled Υ-tree $\langle T, V\rangle$, a frontier E of T and a roof E' of E, the tree $\langle T^{E\leftarrow E'}, V^{E\leftarrow E'}\rangle$ is obtained from $\langle T, V\rangle$ by pumping the difference between E' and E infinitely often. Note that a node $x \in E$ may have several prefixes in E'. Let $up(x)$ be the longest prefix of x in E'. When we pump the difference between E' and x, we refer to $up(x)$. Formally, for $x \in E$, we define $pump(x, E)$ as the set of all words $y_0 \cdot y_1 \cdots y_n \in \Upsilon^*$, with $n \geq 1$, such that $y_0 = x$, for all $0 < i < n-1$ we have $up(y_i) \cdot y_{i+1} \in E$, and $up(y_{n-1}) \cdot y_n \leq E$. Note that since E is a frontier, every word $z \in \Upsilon^*$ is a member of $pump(x, E)$ for at most one $x \in E$, in which case it has a single partition to $y_0 \cdot y_1 \cdots y_n$ as above. Accordingly, for $z \in pump(x, E)$, we can define $tail(z)$ as $up(x) \cdot y_n$. Now,

$$T^{E\leftarrow E'} = \bigcup_{x \in E} \{z : z \text{ is a prefix of } x\} \cup pump(x, E),$$

and

$$V^{E\leftarrow E'}(z) = \left[\begin{array}{ll} V(z) & \text{if } z \leq E, \\ V(tail(z)) & \text{if } z \in pump(x, E) \text{ for } x \in E. \end{array}\right.$$

For simplicity, we denote $\langle T^{E\leftarrow E'}, V^{E\leftarrow E'}\rangle$ by $\langle T, V\rangle^{E\leftarrow E'}$.

When the clopen language L is ω-regular, we can talk about the size of the nondeterministic automaton that accepts L [10]. We say that L is *strongly* ω-regular if both L and $comp(L)$ are ω-regular. When Υ is finite, all ω-regular languages are strongly ω-regular. Since amorphous automata are not closed under complementation, not all ω-regular languages in $\tau(\Sigma, \mathbb{N})$ are strongly ω-regular. In order to talk about the in and out indices of a clopen tree language, we restrict attention to strongly ω-regular languages. It can be shown that the set of symmetric looping nondeterministic automata that recognize clopen languages is closed under complementation, thus, in particular, our results apply to symmetric clopen languages.

Lemma 4.4 *A strongly ω-regular clopen language $L \subseteq \tau(\Sigma, \Upsilon)$ is bounded by $index(L)$.*

[9] Note that L is bounded by the size of the set Υ of directions (in the theorem, L is defined with respect to Υ-trees with $\Upsilon = \mathbb{N}$, thus $|\Upsilon|$ is infinite and L is unbounded). So, when defined with respect to Υ-trees with a finite Υ, the language L is bounded. Also, note that it is the unboundedness of Υ, rather than its infiniteness, that makes L unbounded.

[10] It is possible to extend also the bound based on the diameter of L to the branching framework. The extension, however, is very technical and not very enlightening.

Proof: Assume by way of contradiction that there is a d-Υ-cone $\langle P, V\rangle$ such that $d \geq index(L)$ and $\langle P, V\rangle$ is undetermined. Thus, there are extensions $\langle T_1, V_1\rangle$ and $\langle T_2, V_2\rangle$ of $\langle P, V\rangle$ such that $\langle T_1, V_1\rangle \in L$ and $\langle T_2, V_2\rangle \notin L$. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be nondeterministic looping automata such that $\mathcal{L}(\mathcal{A}_1) = L$, $\mathcal{L}(\mathcal{A}_2) = comp(L)$, and $\mathcal{A}_1$ and $\mathcal{A}_2$ have $in\_index(L)$ and $out\_index(L)$ states, respectively. By the above, there are accepting runs $\langle T_1, r_1\rangle$ and $\langle T_2, r_2\rangle$ of $\mathcal{A}_1$ and $\mathcal{A}_2$ on $\langle T_1, V_1\rangle$ and $\langle T_2, V_2\rangle$, respectively. Both T_1 and T_2 have P as their d-Υ-cone prefix. Since $d \geq in\_index(L) \cdot out\_index(L)$, every path $\pi \subseteq P$ has two nodes x_1 and x_2 such that x_1 is a strict prefix of x_2, $r_1(x_1) = r_1(x_2)$, and $r_2(x_1) = r_2(x_2)$. Thus, both runs repeat their state at x_1 and x_2. The set of nodes x_2 as above is a frontier E and the set of nodes x_1 as above is a roof E' of E. Consider the tree $\langle P, V\rangle^{E\leftarrow E'}$. The accepting run of $\mathcal{A}_1$ on $\langle T_1, V_1\rangle$ induces an accepting run of $\mathcal{A}_1$ on $\langle P, V\rangle^{E\leftarrow E'}$. Formally, the tree $\langle P, r_1\rangle^{E\leftarrow E'}$ is an accepting run of $\mathcal{A}_1$ on $\langle P, V\rangle^{E\leftarrow E'}$. Similarly, the accepting run of $\mathcal{A}_2$ on $\langle T_2, V_2\rangle$ induces an accepting run of $\mathcal{A}_2$ on $\langle P, V\rangle^{E\leftarrow E'}$. It follows that $\langle P, V\rangle^{E\leftarrow E'}$ is accepted by both $\mathcal{A}_1$ and $\mathcal{A}_2$, contradicting the fact that $\mathcal{L}(\mathcal{A}_2) = comp(\mathcal{L}(\mathcal{A}_1))$. $\square$

Theorem 4.5 *If a strongly ω-regular language $L \subseteq \tau(\Sigma, \Upsilon)$ is clopen, then there is a cycle-free nondeterministic tree automaton of size $in\_index(L) \cdot index(L)$ that recognizes L. If L is symmetric, so is the cycle-free automaton.*

Proof: Let $\mathcal{A}$ be a looping automaton for L. Consider a tree $\langle T, V\rangle \in \tau(\Sigma, \Upsilon)$. If $\langle T, V\rangle$ has a prefix $\langle P, V\rangle$ that cannot be extended to a tree accepted by $\mathcal{A}$, then $\langle P, V\rangle$ is a bad prefix and $\langle T, V\rangle \notin L$. On the other hand, if $\langle T, V\rangle$ has an $index(L)$-Υ-cone prefix $\langle P, V\rangle$ that can be extended to a tree accepted by $\mathcal{A}$, then $\langle T, V\rangle \in L$. Indeed, by Theorem 4.4, $\langle P, V\rangle$ is determined, so it must be a good prefix. It follows that $\langle T, V\rangle \in L$ iff $\mathcal{A}$ can proceed over the $index(L)$-Υ-cone prefix of $\langle T, V\rangle$ without getting stuck. Accordingly, we define the cycle-free automaton for L as follows. Let $\mathcal{A} = \langle \Sigma, \Upsilon, Q, \delta, Q_0\rangle$. We define $\mathcal{A}' = \langle \Sigma, \Upsilon, (Q \times \{1, \ldots, index(L)\}) \cup \{q_{acc}\}, \delta', Q_0 \times \{1\}\rangle$, where for all $q \in Q$, $i \in \{1, \ldots, index(L) - 1\}$, $\sigma \in \Sigma$, and $k \in \{1, \ldots, |\Upsilon|\}$, the tuples in $\delta'(\langle q, i\rangle, \sigma, k)$ are obtained from these in $\delta(q, \sigma, k)$ by replacing a state $s \neq q_{acc}$ by the pair $\langle s, i + 1\rangle$. Finally, $\delta'(\langle q, index(L)\rangle, \sigma, k) = \langle q_{acc}, \ldots, q_{acc}\rangle$.

When the automaton $\mathcal{A}$ is symmetric, the sets in $\mathcal{A}'$ are obtained from these in $\mathcal{A}$ by attributing them by the corresponding elements in $\{1, \ldots, index(L)\}$, thus $\mathcal{A}'$ is symmetric as well. $\square$

The proof of Theorem 3.7 can be adapted to the branching case, where the specification language is CTL. The upper bound follows from the exponential translation of CTL to Büchi tree automata [VW86]. For the lower bound, the same example as the linear case works (with the specification being universally quantified). Hence we have the following.

Theorem 4.6 *The translation of clopen CTL formulas and clopen alternating tree automata to nondeterministic or alternating cycle-free tree automata is tightly exponential.*

5 Clopen μ-calculus specifications

The *propositional μ-calculus* is a propositional modal logic augmented with least and greatest fixed-point operators [Koz83]. Specifically, we consider a μ-calculus where formulas are constructed from Boolean propositions with Boolean connectives, the temporal operators EX ("exists next") and

AX ("for all next"), as well as least (μ) and greatest (ν) fixed-point operators. We assume that μ-calculus formulas are written in positive normal form (negation only applied to atomic propositions constants and variables).

Formally, given a set AP of atomic proposition constants and a set APV of atomic proposition variables, a μ-calculus formula is either:

- **true**, **false**, p or $\neg p$ for all $p \in AP$.
- y for all $y \in APV$;
- $\varphi \wedge \psi$, $\varphi \vee \psi$, $EX\varphi$, or $AX\varphi$, where φ and ψ are μ-calculus formulas;
- $\mu y.\varphi(y)$ or $\nu y.\varphi(y)$, where $y \in APV$ and $\varphi(y)$ is a μ-calculus formula containing y as a free variable.

We classify formulas according to the nesting of fixed-point operators in them. Several versions to such a classification can be found in the literature [EL86, Niw86, Bra98]. We describe here a simple version that counts syntactic alternations of μ and ν, resulting in the following definition.

- A formula is in $\Sigma_0 = \Pi_0$ if it contains no fixed-point operators.
- A formula is in Σ_{i+1} if it is of the form φ_i, ψ_i, $\varphi_{i+1} \wedge \varphi'_{i+1}$, $\varphi_{i+1} \vee \varphi'_{i+1}$, $AX\varphi_{i+1}$, and $EX\varphi_{i+1}$, $\mu y.\psi_i(y)$, or $\mu y.\varphi_{i+1}(y)$, where φ_i is a Σ_i formula, ψ_i is a Π_i formula, and φ_{i+1} and φ'_{i+1} are Σ_{i+1} formulas.
- A formula is in Π_{i+1} if it is of the form ψ_i, φ_i, $\psi_{i+1} \wedge \psi'_{i+1}$, $\psi_{i+1} \vee \psi'_{i+1}$, $AX\psi_{i+1}$, and $EX\psi_{i+1}$, $\nu y.\varphi_i(y)$, or $\nu y.\psi_{i+1}(y)$, where ψ_i is a Π_i formula, φ_i is a Σ_i formula, and ψ_{i+1} and ψ'_{i+1} are Π_{i+1} formulas.

Intuitively, Σ_i contains all Boolean and modal combinations of formulas in which there are at most $i-1$ alternations of μ and ν, with the external fixed-point being a μ. Similarly, Π_i contains all Boolean and modal combinations of formulas in which there are at most i alternations of μ and ν, with the external fixed-point being a ν.

The different versions of definitions of Σ_i and Π_i in the literature induce different classes. All versions, however, agree with our simple version with respect to Σ_1 and Π_1, which we study in this section. Indeed, Σ_1 contains all alternation-free μ-calculus formulas whose only fixed-point operator is μ, and dually, Π_1 contains all alternation-free μ-calculus formulas whose only fixed-point operator is ν. *Modal logic* (ML) is a branching temporal logic in which the only temporal operators are EX and AX. Note that ML coincides with Σ_0 and Π_0. The problem of deciding whether a μ-calculus formula has an equivalent ML formula can be decided in exponential time [Ott99].

We now relate the bottom of the μ-calculus hierarchy to safety and co-safety languages.

Lemma 5.1 *All Π_1 formulas induce symmetric safety languages, and all Σ_1 formulas induce symmetric co-safety languages.*

Proof (sketch): Using the techniques of [KVW00], it can be shown that Π_1 formulas can be translated to symmetric alternating looping automata with no ε-transitions. We can then use a subset construction to translate such automata to symmetric nondeterministic looping automata, implying that Π_1 formulas induce symmetric safety languages. Formally, if the alternating automaton is $\mathcal{A}$ with state space S and transitions function ρ, then the nondeterministic automaton is $\mathcal{A}'$ with state space 2^S and transition function δ such that the pairs $\langle S_U, S_E\rangle \in \delta(q,\sigma)$ are these that satisfy all the formulas $\rho(s,\sigma)$ for $s \in q$. The argument for Σ_1 formulas is dual. $\square$

In particular, since Σ_1 formulas induce ω-regular languages, whatever we say in this section about μ-calculus, is valid for trees with arbitrary branching degree.

Lemma 5.2 *A nondeterministic symmetric cycle-free automaton can be translated to an ML formula with a linear blow up.*

Proof: Let $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, F\rangle$ be a symmetric nondeterministic cycle-free automaton. Let $Q = 2^S$. With each underlying state $s \in S$, we associate a formula $\Psi(s,\sigma)$. The formula $\Psi(s)$ associated with s is then the disjunction $\bigvee_{\sigma\in\Sigma}\Psi(q,\sigma)$. If $\{s\}$ is an accepting sink, then $\Psi(s) = \mathbf{true}$. For all other $s \in S$ and $\sigma \in \Sigma$, we define $\Psi(s,\sigma)$ as follows.

- If $\delta(\{s\},\sigma) = \emptyset$, then $\Psi(s,\sigma) = \mathbf{false}$.
- If $\delta(\{s\},\sigma) = \{\langle S_U^1, S_E^1\rangle, \ldots, \langle S_U^l, S_E^l\rangle\}$, then

$$\Psi(s,\sigma) = \sigma \wedge \bigvee_{1\le i\le l} [\bigwedge_{s_u\in S_U^i} AX\Psi(s_u) \wedge \bigwedge_{s_e\in S_E^i} EX\Psi(s_e)].$$

  Since $\mathcal{A}$ is cycle-free, the definition is well founded. Finally, the formula $\Psi(\mathcal{A})$ of $\mathcal{A}$ is the formula $\bigvee_{S\in Q_0}\bigwedge_{s\in S}\Psi(s)$. When represented by a DAG, the length of $\Psi(s,\sigma)$ of linear in the size of $\delta(\{s\},\sigma)$. Hence, the overall blow up is linear.

$\square$

Lemmas 5.1 and 5.2 together imply that the levels at the bottom of the μ-calculus expressiveness hierarchy coalesce.

Theorem 5.3 *If $\psi \in \Sigma_1 \cap \Pi_1$, then $\psi \in ML$. The translation of $\Sigma_1 \cap \Pi_1$ to ML is tightly exponential.*

Proof (sketch): We start with the upper bound. By Lemma 5.1, a formula $\psi \in \Sigma_1 \cap \Pi_1$ is symmetric and clopen. Hence, by Theorem 4.5, there is a symmetric nondeterministic cycle-free automaton that recognizes ψ. So, by Lemma 5.2, ψ can be translated to an ML formula. Since the translation of μ-calculus to symmetric nondeterministic automata involves an exponential blow up, thus the index of the language of trees satisfying ψ is exponential in $|\psi|$, it follows, by Theorems 4.5 and 5.2, that the translation of $\Sigma_1 \cap \Pi_1$ to ML is exponential as well. The lower bound follows from Theorem 4.6. $\square$

It can be shown that the pumping argument used in the proof of Lemma 4.4 can yield finitely-generated trees, which are trees that are generated by unfolding finite graphs. This can be used to show that Theorem 5.3 holds even if we restrict attention to finite structures. It is an interesting open question whether Theorem 5.3 can be generalized to a characterization of $\Sigma_i \cap \Pi_i$, for $i > 1$.

Acknowledgment We thank Vaughan Pratt for very helpful explanations about clopen sets in Cantor spaces.

References

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

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

[BB93] J. Benthem and J. Bergstra. Logic of transition systems. Technical Report P9308, Programing research group, University of Amsterdam, 1993.

[BCC+99] A. Biere, A. Cimatti, E.M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *Proc. 36th Design Automation Conference*, pages 317–320. IEEE Computer Society, 1999.

[BCCZ99] A. Biere, A. Cimatti, E.M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Analysis and Construction of Systems*, volume 1579 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[BCRZ99] A. Biere, E.M. Clarke, R. Raimi, and Y. Zhu. Verifying safety properties of a PowerPC[tm] microprocessor using symbolic model checking without BDDs. In *Computer Aided Verification, Proc. 11th International Conference*, volume 1633 of *Lecture Notes in Computer Science*, pages 172–183. Springer-Verlag, 1999.

[Ben91] J. Benthem. Languages in actions: categories, lambdas and dynamic logic. *Studies in Logic*, 130, 1991.

[BFG+91] A. Bouajjani, J.-C. Fernandez, S. Graf, C. Rodriguez, and J. Sifakis. Safety for branching semantics. In *Proc. 18th International Colloquium on Automata, Languages and Programming*, pages 76–92. Lecture Notes in Computer Science, Springer-Verlag, July 1991.

[BGS00] R. Bloem, H.N. Gabow, and F. Somenzi. An algorithm for strongly connected component analysis in $n \log n$ symbolic steps. In *Formal Methods in Computer Aided Design*, volume 1954 of *Lecture Notes in Computer Science*, pages 37–54. Springer-Verlag, 2000.

[BM78] J. Barwise and Y.N. Moschovakis. Global inductive definability. *Journal of Symbolic Logic*, 43(3):521–534, 1978.

[Bra98] J.C. Bradfield. The modal μ-calculus alternation hierarchy is strict. *Theoretical Computer Science*, 195(2):133–153, March 1998.

[CGP99] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[EH86] E.A. Emerson and J.Y. Halpern. Sometimes and not never revisited: On branching versus linear time. *Journal of the ACM*, 33(1):151–178, 1986.

[EL86] E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional μ-calculus. In *Proc. 1st Symp. on Logic in Computer Science*, pages 267–278, Cambridge, June 1986.

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

[GJ79] M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. Freeman and Co., San Francisco, 1979.

[Gum93] H.P. Gumm. Another glance at the Alpern-Schneider characterization of safety and liveness in concurrent executions. *Information Processing Letters*, 47:291–294, 1993.

[HKSV97] R.H. Hardin, R.P. Kurshan, S.K. Shukla, and M.Y. Vardi. A new heuristic for bad cycle detection using BDDs. In *Computer Aided Verification, Proc. 9th International Conference*, volume 1254 of *Lecture Notes in Computer Science*, pages 268–278. Springer-Verlag, 1997.

[HR86] H.J. Hoogeboom and G. Rozenberg. Infinitary languages: basic theory and applications to concurrent systems. In *Proc. Advanced School on Current Trends in Concurrency*, volume 224 of *Lecture Notes in Computer Science*, pages 266–342. Springer-Verlag, 1986.

[JW95] D. Janin and I. Walukiewicz. Automata for the modal μ-calculus and related results. In *Proc. 20th International Symp. on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, pages 552–562. Springer-Verlag, 1995.

[Kin94] E. Kindler. Safety and liveness properties: A survey. *Bulletin of the EATCS*, 53:268 – 272, 1994.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

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

[KV99] O. Kupferman and M.Y. Vardi. Model checking of safety properties. In *Computer Aided Verification, Proc. 11th International Conference*, volume 1633 of *Lecture Notes in Computer Science*, pages 172–183. Springer-Verlag, 1999.

[KVW00] O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, March 2000.

[Lam80] L. Lamport. Sometimes is sometimes "not never" - on the temporal logic of programs. In *Proc. 7th ACM Symp. on Principles of Programming Languages*, pages 174–185, January 1980.

[Lam85] L. Lamport. Logical foundation. In *Distributed systems - methods and tools for specification*, volume 190 of *Lecture Notes in Computer Science*. Springer-Verlag, 1985.

[Lan69] L.H. Landweber. Decision problems for ω–automata. *Mathematical Systems Theory*, 3:376–384, 1969.

[MH84] S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330, 1984.

[MP89] Z. Manna and A. Pnueli. The anchored version of the temporal framework. In *Linear time, branching time, and partial order in logics and models for concurrency*, volume 345 of *Lecture Notes in Computer Science*, pages 201–284. Springer-Verlag, 1989.

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

[MT01] P. Manolios and R. Trefler. Safety and liveness in branching time. In *Proc. 16th IEEE Symp. on Logic in Computer Science*, June 2001.

[Niw86] D. Niwiński. On fixed point clones. In *Proc. 13th International Colloquium on Automata, Languages and Programming*, volume 226 of *Lecture Notes in Computer Science*, pages 464–473. Springer-Verlag, 1986.

[Ott99] M. Otto. Eliminating recursion in the mu-calculus. In *Proc. 16th Symp. on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pages 531–540. Springer-Verlag, 1999.

[RS59] M.O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:115–125, 1959.

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

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

[Sta97] L. Staiger. ω-languages. *Handbook of Formal Languages*, pages 339–388, 1997.

[Tho90] W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science*, pages 165–191, 1990.

[Var96] M.Y. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer-Verlag, Berlin, 1996.

[VW86] M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Science*, 32(2):182–221, April 1986.

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

[Wil99a] T. Wilke. Classifying discrete temporal properties. In *Proc. 16th Symp. on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pages 32–46. Springer-Verlag, 1999.

[Wil99b] T. Wilke. CTL^+ is exponentially more succinct than CTL. In C. Pandu Ragan, V. Raman, and R. Ramanujam, editors, *Proc. 19th conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1738 of *Lecture Notes in Computer Science*, pages 110–121. Springer-Verlag, 1999.

[Yan00] J. Yang. A theory for generalized symbolic trajectory evaluation. In *Symposium on Symbolic Trajectory Evaluation*, Chicago, July 2000.