The Concurrency Column

BY

Luca Aceto

BRICS, Department of Computer Science
Aalborg University, 9220 Aalborg Ø, Denmark

Dept. of Computer Science, School of Science and Engineering
Reykjavik University, 103 Reykjavik, Iceland

luca@{cs.auc.dk,ru.is}, http://www.cs.auc.dk/~luca/BEATCS

The automata-theoretic approach to the specification and verification of reactive systems is one of my favourite examples of a beautiful mathematical theory that is having increasing impact on the practice of computer science. The importance of work done in the automata-theoretic approach to system analysis has been recognized by the computer science community at large with the award of the ACM's Paris Kanellakis Theory and Practice Award for 2005 to Gerard J. Holzmann, Robert P. Kurshan, Moshe Y. Vardi, and Pierre Wolper, who "demonstrated that checking the correctness of reactive systems can be achieved using a mathematical analysis of abstract machines".

The application of the theory of automata on infinite words or trees in the algorithmic analysis of reactive systems has motivated a deep examination of the complexity of algorithmic problems and constructions over those structures. The result is a very satisfactory algorithmic theory of automata in which the complexity of most natural and useful problems is well understood. However, despite determined efforts by the research community over the years, there are still some fundamental problems for which the gap between the known upper and lower bounds on their complexity is at least exponential.

In the present installment of the concurrency column, Orna Kupferman, one of the prime movers in the study of automata-theoretic constructions and in their application to the verification of reactive systems, presents a survey of several problems in which the gap in our knowledge is exponential, and describes recent efforts to close the gaps.

I am glad to be able to offer this piece to the readership of the concurrency column, and I trust that you will enjoy reading it as much as I did.

Exponential Gaps in our Knowledge

Orna Kupferman
School of Engineering and Computer Science
Hebrew University
Jerusalem 91904, Israel
Email: orna@cs.huji.ac.il

Abstract

Automata on infinite objects were the key to the solution of several fundamental decision problems in mathematics and logic. Today, automata on infinite objects are used for formal specification and verification of reactive systems. The practical importance of automata in formal methods has motivated a re-examination of the complexity of problems and constructions involving automata. For most problems and constructions, the situation is satisfying, in the sense that even if there is a gap between the upper and the lower bound, it is small. For some fundamental and highly practical cases, however, the gap between the upper and the lower bound is exponential or even larger. The article surveys several such frustrating cases, studies features that they share, and describes recent efforts (with partial success) to close the gaps.

1 Introduction

Finite *automata on infinite objects* were first introduced in the 60's, and were the key to the solution of several fundamental decision problems in mathematics and logic [7, 34, 43]. Today, automata on infinite objects are used for *specification* and *verification* of reactive systems. The automata-theoretic approach to verification reduces questions about systems and their specifications to questions about automata [26, 55]. Recent industrial-strength property-specification languages such as Sugar [4], ForSpec [1], and the recent standard PSL 1.01 [20] include regular expressions and/or automata, making specification and verification tools that are based on automata even more essential and popular.

Early automata-based algorithms aimed at showing decidability. The complexity of the algorithm was not of much interest. For example, the fundamental automata-based algorithms of Büchi and Rabin, for the decidability of S1S and

SnS (the monadic second-order theories of infinite words and trees, respectively) [7, 43] are of non-elementary complexity (i.e., the complexity can not be bounded by a stack of exponentials of a fixed height [35]). Proving the decidability of a given logic was then done by translating the logic to a monadic second-order theory [15, 24], ignoring the fact that a direct algorithm could have been more efficient. Things have changed in the early 80's, when decidability of highly expressive logics became of practical importance in areas such as artificial intelligence and formal reasoning about systems [16, 23]. The change was reflected in the development of two research directions: (1) direct and efficient translations of logics to automata [57, 48, 50, 54], and (2) improved algorithms and constructions for automata on infinite objects [46, 11, 42].

Both research directions are relevant not only for solving the decidability problem, but also for solving other basic problems in formal methods, such as model checking [9] and synthesis [42]. Moreover, input from the industry continuously brings to the field new problems and challenges, requiring the development of new translations and algorithms.[1] For many problems and constructions, our community was able to come up with satisfactory solutions, in the sense that the upper bound (the complexity of the best algorithm or the blow-up in the best known construction) coincides with the lower bound (the complexity class in which the problem is hard, or the blow-up that is known to be unavoidable). For some problems and constructions, however, the gap between the upper bound and the lower bound is significant. This situation is especially frustrating, as it implies that not only we may be using algorithms that can be significantly improved, but also that something is missing in our understanding of automata on infinite objects.

Before turning to the frustrating cases, let us first describe one "success story" — the complementation construction for nondeterministic Büchi automata on infinite words (NBWs). Translating S1S into NBWs, Büchi had to prove the closure of NBWs under complementation. For that, Büchi suggested in 1962 a doubly-exponential construction. Thus, starting with an NBW with n states, the complementary automaton had $2^{2^{O(n)}}$ states [7]. The lower bound known then for NBW complementation was 2^n, which followed from the complementation of automata on finite words [45]. Motivated by problems in formal methods, Sistla, Vardi, and Wolper developed in 1985 a better complementation construction with only a $2^{O(n^2)}$ blow-up [51]. Only in 1988, Safra introduced a determinization construction for NBWs that enabled a $2^{O(n \log n)}$ complementation construction [46], and Michel proved a matching lower bound [37]. The story, however, was not over.

[1]In fact, the practical importance of automata has lead to a reality in which the complexity of a solution or a construction is only one factor in measuring its quality. Other measures, such as the feasibility of a symbolic implementation or the effectiveness of optimizations and heuristics in the average case are taken into an account too. In this article, however, we only consider worst-case complexity.

A careful analysis of the lower and upper bounds reveals an exponential gap hiding in the constants of the $O()$ notations. While the upper bound of Safra is n^{2n}, the lower bound of Michel is only $n!$, which is roughly $(n/e)^n$. Only recently, a new complementation construction, which avoids determinization, has led to an improved upper bound of $(0.97n)^n$ [14], and a new concept, of full automata, has led to an improved lower bound of $(0.76n)^n$ [58]. Thus, a gap still exists, but it is an acceptable one, and it probably does not point to a significant gap in our understanding of nondeterministic Büchi automata.

In the article, we survey two representative problems for which the gap between the upper and the lower bound is still exponential. In Section 3, we consider safety properties and the problem of translating safety properties to nondeterministic automata on finite words. In Section 4, we consider the problem of translating nondeterministic Büchi word automata to nondeterministic co-Büchi word automata. Both problems have strong practical motivation, and in both progress has been recently achieved. We study the problems, their motivation, and their current status. The study is based on joint work with Moshe Vardi [27], Robby Lampert [21], and Benjamin Aminof [2].

2 Preliminaries

Word automata. Given an alphabet Σ, an *infinite word over* Σ is an infinite sequence $w = \sigma_1 \cdot \sigma_2 \cdots$ of letters in Σ. A *nondeterministic Büchi word automaton* (NBW, for short) is $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, F \rangle$, where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $Q_0 \subseteq Q$ is a set of initial states, and $F \subseteq Q$ is a set of accepting states. If $|Q_0| = 1$ and δ is such that for every $q \in Q$ and $\sigma \in \Sigma$, we have that $|\delta(q, \sigma)| = 1$, then $\mathcal{A}$ is a *deterministic* Büchi word automaton (DBW).

Given an input word $w = \sigma_0 \cdot \sigma_1 \cdots$ in Σ^ω, a *run* of $\mathcal{A}$ on w is a sequence $r_0, r_1, \ldots$ of states in Q such that $r_0 \in Q_0$ and for every $i \geq 0$, we have $r_{i+1} \in \delta(r_i, \sigma_i)$; i.e., the run starts in one of the initial states and obeys the transition function. Note that a nondeterministic automaton can have many runs on w. In contrast, a deterministic automaton has a single run on w. For a run r, let $inf(r)$ denote the set of states that r visits infinitely often. That is,

$$inf(r) = \{q \in Q \ : r_i = q \text{ for infinitely many } i \geq 0\}.$$

As Q is finite, it is guaranteed that $inf(r) \neq \emptyset$. The run r is *accepting* iff $inf(r) \cap F \neq \emptyset$. That is, a run r is accepting iff there exists a state in F that r visits infinitely often. A run that is not accepting is *rejecting*. An NBW $\mathcal{A}$ accepts an input word w iff there exists an accepting run of $\mathcal{A}$ on w. The *language* of an NBW $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts. We assume that a given NBW $\mathcal{A}$ has no empty states (that is, at least one word is accepted from each state – otherwise we can remove the state).

Linear Temporal Logic. The logic *LTL* is a linear temporal logic. Formulas of LTL are constructed from a set AP of atomic propositions using the usual Boolean operators and the temporal operators G ("always"), F ("eventually"), X ("next time"), and U ("until"). Formulas of LTL describe computations of systems over AP. For example, the LTL formula $G(req \rightarrow Fack)$ describes computations in which every position in which *req* holds is eventually followed by a position in which *ack* holds. For the detailed syntax and semantics of LTL, see [41]. The *model-checking problem* for LTL is to determine, given an LTL formula ψ and a system M, whether all the computations of M satisfy ψ.

General methods for LTL model checking are based on translation of LTL formulas to nondeterministic Büchi word automata:

Theorem 1. [55] *Given an LTL formula ψ, one can construct an NBW $\mathcal{A}_\psi$ that accepts exactly all the computations that satisfy ψ. The size of $\mathcal{A}_\psi$ is, in the worst case, exponential in the length of ψ.*

Given a system M and a property ψ, model checking of M with respect to ψ is reduced to checking the emptiness of the product of M and $\mathcal{A}_{\neg\psi}$ [55]. This check can be performed on-the-fly and symbolically [10, 17, 52], and the complexity of model checking that follows is PSPACE, with a matching lower bound [47].

3 Translating Safety Properties to Automata

In this section we describe the first exponential gap in our knowledge: the complexity of translating an NBW $\mathcal{A}$ that recognizes a co-safety language to a nondeterministic automaton on finite words (NFW) $\mathcal{A}'$ such that $\mathcal{A}'$ accepts only good prefixes for $L(\mathcal{A})$, and every word that is accepted by $\mathcal{A}$ has at least one good prefix accepted by $\mathcal{A}'$. We first define safety and co-safety languages, define bad and good prefixes, and motivate the above construction. We then describe a recent result that describes a construction of an NFW for bad and good prefixes for the case the safety or the co-safety property is given by means on an LTL formula.

3.1 Safety properties and their verification

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 two processes are never simultaneously in the critical section. 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 two processes are in the critical section simultaneously, this occurs after some finite execution of the system.

In order to formally define what safety properties are, we refer to computations of a nonterminating system as infinite words over an alphabet Σ. Typically, $\Sigma = 2^{AP}$, where AP is the set of the system's atomic propositions. Consider a language $L \subseteq \Sigma^\omega$ of infinite words over the alphabet Σ. A finite word $x \in \Sigma^*$ is a *bad prefix* for L iff for all $y \in \Sigma^\omega$, we have $x \cdot y \notin L$. Thus, a bad prefix is a finite word that cannot be extended to an infinite word in L. Note that if x is a bad prefix, then all the finite extensions of x are also bad prefixes. A language L is a *safety* language iff every infinite word $w \notin L$ has a finite bad prefix.[2] For a safety language L, we denote by *bad-pref*(L) the set of all bad prefixes for L. For example, if $\Sigma = \{0, 1\}$, then $L = \{0^\omega, 1^\omega\}$ is a safety language. To see this, note that every word not in L contains either the sequence 01 or the sequence 10, and a prefix that ends in one of these sequences cannot be extended to a word in L. Thus, *bad-pref*(L) is the language of the regular expression $(0^* \cdot 1 + 1^* \cdot 0) \cdot (0 + 1)^*$.

For a language $L \subseteq \Sigma^\omega$ (Σ^*), we use *comp*(L) to denote the complement of L; i.e., *comp*$(L) = \Sigma^\omega \setminus L$ $(\Sigma^* \setminus L$, respectively). We say that a language $L \subseteq \Sigma^\omega$ is a *co-safety* language iff *comp*(L) is a safety language. (The term used in [38] is *guarantee* language.) Equivalently, L is co-safety iff every infinite word $w \in L$ has a *good prefix* $x \in \Sigma^*$: for all $y \in \Sigma^\omega$, we have $x \cdot y \in L$. For a co-safety language L, we denote by *good-pref*(L) the set of good prefixes for L. Note that for a safety language L, we have that *good-pref*$($*comp*$(L)) =$ *bad-pref*(L). Thus, in order to construct the set of bad prefixes for a safety property, one can construct the set of good prefixes for its complementary language.

We say that an NBW $\mathcal{A}$ is a safety (co-safety) automaton iff $\mathcal{L}(\mathcal{A})$ is a safety (co-safety) language. We use *bad-pref*$(\mathcal{A})$, *good-pref*$(\mathcal{A})$, and *comp*$(\mathcal{A})$ to abbreviate *bad-pref*$(\mathcal{L}(\mathcal{A}))$, *good-pref*$(\mathcal{L}(\mathcal{A}))$, and *comp*$(\mathcal{L}(\mathcal{A}))$, respectively.

In addition to proof-based methods for the verification of safety properties [38, 39], there is extensive work on model checking of safety properties. Recall that general methods for model checking of linear properties are based on a construction of an NBW $\mathcal{A}_{\neg\psi}$ that accepts exactly all the infinite computations that violate the property ψ and is of size exponential in ψ [32, 55]. Verification of a system M with respect to ψ is then reduced to checking the emptiness of the product of M and $\mathcal{A}_{\neg\psi}$ [53].

When ψ is a safety property, the NBW $\mathcal{A}_{\neg\psi}$ can be replaced by *bad-pref*$(\mathcal{A}_\psi)$

[2] The definition of safety we consider here is given in [3], it coincides with the definition of limit closure defined in [12], and is different from the definition in [30], which also refers to the property being closed under stuttering.

– an NFW that accepts exactly all the bad prefixes of ψ [27]. This has several advantages, as reasoning about finite words is simpler than reasoning about infinite words: symbolic reasoning (in particular, bounded model checking procedures) need not look for loops (cf. [18]) and can, instead, apply backward or forward reachability analysis [5, 19, 8]. In fact, the construction of *bad-pref*$(\mathcal{A}_\psi)$ reduces the model-checking problem to the problem of invariance checking [38], which is amenable to both model-checking techniques [5, 19] and deductive verification techniques [6, 40, 33]. In addition, using *bad-pref*$(\mathcal{A}_\psi)$, we can return to the user a finite error trace, which is a bad prefix, and which is often more helpful than an infinite error trace.

Consider a safety NBW $\mathcal{A}$. The construction of *bad-pref*$(\mathcal{A})$ was studied in [27]. If $\mathcal{A}$ is deterministic, we can construct a *deterministic* automaton on finite words (DFW) for *bad-pref*$(\mathcal{A})$ by defining the set of accepting states to be the set of states s for which $\mathcal{A}$ with initial state s is empty. Likewise, if $\mathcal{A}$ is a co-safety automaton, we can construct a DFW for *good-pref*$(\mathcal{A})$ by defining the set of accepting states to be the set of states s for which $\mathcal{A}$ with initial state s is universal.

When $\mathcal{A}$ is nondeterministic, the story is more complicated. Even if we are after a nondeterministic, rather than a deterministic, automaton for the bad or good prefixes, the transition from infinite words to finite words involves an exponential blow-up. Formally, we have the following.

Theorem 2. [27] *Consider an NBW $\mathcal{A}$ of size n.*

1. *If $\mathcal{A}$ is a safety automaton, the size of an NFW for bad-pref$(\mathcal{A})$ is $2^{\Theta(n)}$.*
2. *If $\mathcal{A}$ is a co-safety automaton, the size of an NFW for good-pref$(\mathcal{A})$ is $2^{\Theta(n)}$.*

The lower bound in Theorem 2 for the case $\mathcal{A}$ is a safety automaton is not surprising. Essentially, it follows from the fact that *bad-pref*$(\mathcal{A})$ refers to words that are not accepted by $\mathcal{A}$. Hence, it has the flavor of complementation, and complementation of nondeterministic automata involves an exponential blow-up [36]. The second blow up, however, in going from a co-safety automaton to a nondeterministic automaton for its good prefixes is surprising. Its proof in [27] highlights the motivation behind the definition of a *fine automaton* for safety properties, and we describe it below.

For $n \geq 1$, let $\Sigma_n = \{1, \ldots, n, \&\}$. We define L_n as the language of all words $w \in \Sigma_n^\omega$ such that w contains at least one & and the letter after the first & is either & or it has already appeared somewhere before the first &. The language L_n is a co-safety language. Indeed, each word in L_n has a good prefix (e.g., the one that contains the first & and its successor). We can recognize L_n with an NBW with $O(n)$ states (the NBW guesses the letter that appears after the first &).

Obvious good prefixes for L_n are 12&&, 123&2, etc. That is, prefixes that end one letter after the first &, and their last letter is either & or has already appeared somewhere before the &. We can recognize these prefixes with an NFW with $O(n)$ states. But L_n also has some less obvious good prefixes, like $1234 \cdots n\&$ (a permutation of $1 \ldots n$ followed by &). These prefixes are indeed good, as every suffix we concatenate to them would start with & or with a letter in $\{1, \ldots, n\}$, which has appeared before the &. To recognize these prefixes, an NFW needs to keep track of subsets of $\{1, \ldots, n\}$, for which it needs 2^n states. Consequently, an NFW for *good-pref*(L_n) must have at least 2^n states.

It is also shown in [27] that the language L_n can be encoded by an LTL formula of length quadratic in n. This implies that the translation of safety and co-safety LTL formulas to NFWs for the bad and good prefixes is doubly exponential. Formally, we have the following.

Theorem 3. [27] *Given a safety LTL formula ψ of size n, the size of an NFW for bad-pref(ψ) is $2^{2^{\Omega(\sqrt{n})}}$.*

3.2 Fine automata and their construction

As described in the proof of Theorem 2, some good prefixes for L_n (the "obvious prefixes") can be recognized by a small NFW. What if we give up the non-obvious prefixes and construct an NFW $\mathcal{A}'$ that accepts only the "obvious subset" of L_n? It is not hard to see that each word in L_n has an obvious prefix. Thus, while $\mathcal{A}'$ does not accept all the good prefixes, it accepts at least one prefix of every word in L. This useful property of $\mathcal{A}'$ is formalized below.

Consider a safety language L. We say that a set $X \subseteq$ *bad-pref*(L) is a *trap* for L iff every word $w \notin L$ has at least one bad prefix in X. Thus, while X need not contain all the bad prefixes for L, it must contain sufficiently many prefixes to "trap" all the words not in L. Dually, a trap for a co-safety language L is a set $X \subseteq$ *good-pref*(L) such that every word $w \in L$ has at least one good prefix in X. We denote the set of all the traps, for an either safety or co-safety language L, by *trap*(L).

An NFW $\mathcal{A}$ is *fine* for a safety or a co-safety language L iff $\mathcal{A}$ accepts a trap for L. For example, an NFW that accepts $0^* \cdot 1 \cdot (0 + 1)$ does not accept all the bad prefixes of the safety language $\{0^\omega\}$; in particular, it does not accept the minimal bad prefixes in $0^* \cdot 1$. Yet, such an NFW is fine for $\{0^\omega\}$. Indeed, every infinite word that is different from 0^ω has a prefix in $0^* \cdot 1 \cdot (0+1)$. Likewise, the NFW is fine for the co-safety language $0^* \cdot 1 \cdot (0 + 1)^\omega$. In practice, almost all the benefit that one obtains from an NFW that accepts all the bad/good prefixes can also be obtained from a fine automaton. It is shown in [27] that for natural safety formulas ψ, the construction of an NFW fine for ψ is as easy as the construction of $\mathcal{A}_{\neg\psi}$. In more details, if we regard $\mathcal{A}_{\neg\psi}$ as an NFW, with an appropriate definition of the set of accepting states, we get an automaton fine for ψ. For general safety formulas, the problem of constructing small fine automata was left open in [27] and its solution in [21] has led to new mysteries in the context of safety properties. Let us first describe the result in [21].

Recall that the transition from a safety NBW to an NFW for its bad prefixes is exponential, and that the exponential blow up follows from the fact that a complementing NBW can be construction from a tight NFW. When we consider fine automata, things are more complicated, as the fine NFW need not accept all bad prefixes. As we show below, however, a construction of fine automata still has the flavor of complementation, and must involve an exponential blow up.

Theorem 4. [21] *Given a safety NBW $\mathcal{A}$ of size n, the size of an NFW fine for $\mathcal{A}$ is $2^{\Theta(n)}$.*

We now move on to consider co-safety NBWs. Recall that, as with safety properties and bad prefixes, the transition from a co-safety NBW to an NFW for its good prefixes is exponential. We show that a fine NFW for a co-safety property can be constructed from the NBWs for the property and its negation. The idea is that it is possible to bound the number of times that a run of $\mathcal{A}$ visits the set of accepting states when it runs on a word not in $\mathcal{L}(\mathcal{A})$. Formally, we have the following:

Lemma 5. [21] *Consider a co-safety NBW $\mathcal{A}$. Let F be the set of accepting states of $\mathcal{A}$ and let $\overline{\mathcal{A}}$ be an NBW with $\overline{n}$ states such that $\mathcal{L}(\overline{\mathcal{A}}) = comp(\mathcal{L}(\mathcal{A}))$. If a run of $\mathcal{A}$ on a finite word $h \in \Sigma^*$ visits F more than $|F| \cdot \overline{n}$ times, then h is a good prefix for $\mathcal{L}(\mathcal{A})$.*

Proof: Since $\mathcal{A}$ is a co-safety NBW, $\overline{\mathcal{A}}$ is a safety NBW. Recall that no state of $\overline{\mathcal{A}}$ is empty. Therefore, by [49], every infinite run of $\overline{\mathcal{A}}$ is accepting[3]. Let $r = r_0 r_1 \ldots r_l$ be a run of $\mathcal{A}$ on h that visits F more than $|F| \cdot \overline{n}$ times. Assume, by way of contradiction, that h is not a good prefix. Then, h can be extended to a word accepted by $\overline{\mathcal{A}}$, and thus, there is a (finite) run $r' = r'_0 r'_1 \ldots r'_l$ of $\overline{\mathcal{A}}$ on h. Since r visits F more than $|F| \cdot \overline{n}$ times, there exist $0 \leq i < j \leq l$ such that $r_j = r_i \in F$ and $r'_j = r'_i$. Let $w = h_1 \ldots h_i (h_{i+1} \ldots h_j)^\omega$. Since $r_j = r_i \in F$, the run $r_0 r_1 \ldots r_i (r_{i+1} \ldots r_j)^\omega$ is an accepting run of $\mathcal{A}$ on w. On the other hand, $r'_0 r'_1 \ldots r'_i (r'_{i+1} \ldots r'_j)^\omega$ is an accepting run of $\overline{\mathcal{A}}$ on w. Hence, w is accepted by both $\mathcal{A}$ and $\overline{\mathcal{A}}$, and we have reached a contradiction. □

[3] Note that $\overline{\mathcal{A}}$ is equivalent to the nondeterministic word automaton obtained by making all its states accepting. Such automata are termed *looping*.

Consider a co-safety NBW $\mathcal{A}$ with n states, m of them accepting. Let $\overline{\mathcal{A}}$ be an NBW with $\overline{n}$ states such that $\mathcal{L}(\overline{\mathcal{A}}) = comp(\mathcal{L}(\mathcal{A}))$. Following Lemma 5, we can construct an NFW fine for $\mathcal{A}$ by taking $(m \cdot \overline{n}) + 1$ copies of $\mathcal{A}$, and defining the transition function such that when a run of $\mathcal{A}'$ visits F in the j-th copy of $\mathcal{A}$, it moves to the $(j + 1)$-th copy. The accepting states of $\mathcal{A}'$ are the states of F in the $(m \cdot \overline{n} + 1)$-th copy. This implies the following theorem.

Theorem 6. [21] *Consider a co-safety NBW $\mathcal{A}$ with n states, m of them accepting. Let $\overline{\mathcal{A}}$ be an NBW with $\overline{n}$ states such that $\mathcal{L}(\overline{\mathcal{A}}) = comp(\mathcal{L}(\mathcal{A}))$. There exists an NFW $\mathcal{A}'$ with $n \cdot (m \cdot \overline{n} + 1)$ states such that $\mathcal{A}'$ is fine for $\mathcal{L}(\mathcal{A})$.*

Given a safety NBW, its complement NBW is co-safety. Thus, dualizing Theorem 6, we get the following.

Theorem 7. [21] *Consider a safety NBW with n states. Let $\overline{\mathcal{A}}$ be an NBW with $\overline{n}$ states, $\bar{m}$ of them accepting, such that $\mathcal{L}(\overline{\mathcal{A}}) = comp(\mathcal{L}(\mathcal{A}))$. There exists an NFW $\mathcal{A}'$ with $\bar{n} \cdot (\bar{m} \cdot n + 1)$ states such that $\mathcal{A}'$ is fine for $\mathcal{L}(\mathcal{A})$.*

By Theorem 1, given an LTL formula ψ, we can construct NBWs $\mathcal{A}_\psi$ and $\mathcal{A}_{\neg\psi}$ for ψ and $\neg\psi$, respectively. The number of states in each of the NBWs is at most $2^{O(|\psi|)}$. Hence, by Theorem 6, we can conclude:

Theorem 8. [21] *Consider a safety LTL formula φ of length n. There exists an NFW fine for φ with at most $2^{O(n)}$ states.*

It follows from Theorem 8 that the transition from a tight NFW (one that accepts exactly all bad or good prefixes) to a fine NFW is significant, as it circumvents the doubly exponential blow-up in Theorem 3.

3.3 Discussion

The work in [21] has answered positively the question about the existence of exponential fine automata for general safety LTL formulas, improving the doubly-exponential construction in [27]. Essentially, the construction adds a counter on top of the NBW for the formula. The counter is increased whenever the NBW visits an accepting state, and a computation is accepted after the counter reaches a bound that depends on the size of the formula. For a discussion on the application of the result in the context of bounded model checking and run-time verification see [21]. Here, we discuss the theoretical aspects of the result.

While [21] has solved the problem of constructing exponential fine automata for LTL formulas, the problem of constructing polynomial fine automata for co-safety NBW is still open. The challenge here is similar to other challenges in automata-theoretic constructions in which one needs both the NBW and its complementing NBW — something that is easy to have in the context of LTL, but difficult in the context of NBW. More problems in this status are reported in [29]. For example, the problem of deciding whether an LTL formula ψ can be translated to a DBW can be solved by reasoning about the NBWs for ψ and $\neg\psi$. This involves an exponential blow up in the length of ψ, but, as in our case, no additional blow-up for complementation. The problem of deciding whether an NBW can be translated to a DBW cannot be solved using the same lines, as here complementation does involve an exponential blow up. From a practical point of view, however, the problem of going from a co-safety automaton to a fine NFW is of less interest, as users that use automata as their specification formalism are likely to start with an automaton for the bad or the good prefixes anyway. Thus, the problem about the size of fine automata is interesting mainly for the specification formalism of LTL, which [21] did solve.

4 From Büchi to co-Büchi Automata

The second open problem we describe is the problem of translating, when possible, a nondeterministic Büchi word automaton to an equivalent nondeterministic co-Büchi word automaton (NCW). The *co-Büchi* acceptance condition is dual to the Büchi acceptance condition. Thus, $F \subseteq Q$ and a run r is accepting if it visits F only finitely many times. Formally, $inf(r) \cap F = \emptyset$. NCWs are less expressive than NBWs. For example, the language $\{w : w \text{ has only finitely many 0s}\} \subseteq \{0, 1\}^\omega$ cannot be recognized by an NCW. In fact, NCWs are as expressive as deterministic co-Büchi automata (DCWs). Hence, as DBWs are dual to DCWs, a language can be recognized by an NCW iff its complement can be recognized by a DBW.

The best translation of NBW to NCW (when possible) that is currently known actually results in a deterministic co-Büchi automaton (DCW), and it goes as follows. Consider an NBW $\mathcal{A}$ that has an equivalent NCW. First, co-determinize $\mathcal{A}$ and obtain a deterministic Rabin automaton (DRW) $\tilde{\mathcal{A}}$ for the complement language. By [25], DRWs are *Büchi type*. That is, if a DRW has an equivalent DBW, then the DRW has an equivalent DBW of the same structure. Let $\tilde{\mathcal{B}}$ be the DBW equivalent to $\tilde{\mathcal{A}}$ (recall that since $\mathcal{A}$ can be recognized by an NCW, its complement can be recognized by a DBW). By dualizing $\tilde{\mathcal{B}}$ one gets a DCW equivalent to $\mathcal{A}$. The co-determinization step involves an exponential blowup in the number of states [46]. Hence, starting with an NBW with n states, we end up with an NCW with $2^{O(n \log n)}$ states. This is particularly annoying as even a lower bound showing that an NCW needs one more state is not known. As we discuss below, the translation of NBW to an equivalent NCW is of practical importance because of its relation to the problem of translating LTL formulas to equivalent alternation-free

μ-calculus (AFMC) formulas (when possible).

It is shown in [28] that given an LTL formula ψ, there is an AFMC formula equivalent to $\forall\psi$ iff ψ can be recognized by a DBW. Evaluating specifications in the alternation-free fragment of μ-calculus can be done with linearly many symbolic steps. In contrast, direct LTL model checking reduces to a search for bad-cycles and its symbolic implementation involves nested fixed-points, and is typically quadratic [44]. Hence, identifying LTL formulas that can be translated to AFMC, and coming up with an optimal translation, is a problem of great practical importance. The best known translations of LTL to AFMC first translates the LTL formula ψ to a DBW, which is then linearly translated to an AFMC formula for $\forall\psi$. The translation of LTL to DBW, however, is doubly exponential, thus the overall translation is doubly-exponential, with only an exponential matching lower bound [28].

The reason that current translations go through an intermediate deterministic automaton is the need to run this automaton on all the computations of the system in a way that computations with the same prefix follow the same run. A similar situation exists when we expand a word automaton to a tree automaton [13] — the word automaton cannot be nondeterministic, as different branches of the tree that have the same prefix u may be accepted by runs of the word automaton that do not agree on the way they proceed on u. A promising direction for coping with this situation was suggested in [28]: Instead of translating the LTL formula ψ to a DBW, one can translate $\neg\psi$ to an NCW. This can be done either directly, or by translating the NBW for $\neg\psi$ to an equivalent NCW. Then, the NCW can be linearly translated to an AFMC formula for $\exists\neg\psi$, whose negation is equivalent to $\forall\psi$. The fact that the translation can go through a nondeterministic rather than a deterministic automaton is very promising, as nondeterministic automata are typically exponentially more succinct than deterministic ones.[4] Nevertheless, the problem of translating LTL formulas to NCWs of exponential size is still open. [5]The best translation that is known today involves a doubly-exponential blow up, and it actually results in a DCW, giving up the idea that the translation of LTL to AFMC can be exponentially more efficient by using intermediate nondeterministic automata. Note that a polynomial translation of NBW to NCW will imply a singly-exponential translation of LTL to AFMC, as the only exponential step in the procedure will be the translation of LTL to NBW.[6]

[4]Dually, we can translate the LTL formula to a *universal* Büchi automaton and translate this automaton to an AFMC formula. The universal Büchi automaton for ψ is dual to the nondeterministic co-Büchi automaton for $\neg\psi$.

[5]As mentioned above, not all LTL formulas can be translated to NCWs. When we talk about the blow up in a translation, we refer to formulas for which a translation exists.

[6] Wilke [56] proved an exponential lower-bound for the translation of an NBW for an LTL formula ψ to and AFMC formula equivalent to $\forall\psi$. This lower-bound does not preclude a polynomial

Recall that while the best upper bound for an NBW to NCW translation is $2^{O(n \log n)}$, we do not even have a single example to a language whose NBW is smaller than its NCW. In fact, it was only recently shown that NBWs are not *co-Büchi-type*. That is, there is an NBW $\mathcal{A}$ such that $L(\mathcal{A})$ can be recognized by an NCW, but an NCW for $L(\mathcal{A})$ must have a different structure than $\mathcal{A}$. We describe such an NBW in the proof below (the original proof, in [22], has a different example).

Lemma 9. [22] *NBW are not co-Büchi-type.*

Proof: Consider the NBW $\mathcal{A}$ described in Figure 1. The NBW recognizes the language L of all words with at least one a and at least one b. This language can be recognized by an NCW, yet it is easy to see that there is no way to define F on top of $\mathcal{A}$ such that the result is an NCW that recognizes L. □

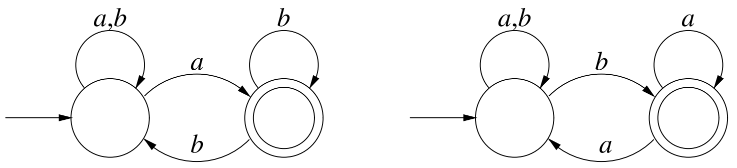

Figure 1: An NBW for "at last one a and at least one b"

During our efforts to solve the NBW to NCW problem, we have studied the related problem of translating NBWs to NFWs. In the next section we describe the problem, its relation to the NBW to NCW problem, and our partial success in this front.

4.1 From Büchi to limit finite automata

Recall that DBWs are less expressive than NBWs. Landweber characterizes languages $L \subseteq \Sigma^\omega$ that can be recognized by a DBW as those for which there is a regular language $R \subseteq \Sigma^*$ such that L is the *limit* of R. Formally, w is in the limit of R iff w has infinitely many prefixes in R [31]. It is not hard to see that a DBW for L, when viewed as a DFW, recognizes a language whose limit is L, and vice versa – a DFW for R, when viewed as a DBW, recognizes the language that is the limit of R. What about the case R and L are given by nondeterministic automata? It is not hard to see that the simple transformation between the two formalisms

upper-bound for the translation of an NBW for $\neg\psi$ to an AFMC formula equivalent to $\exists\neg\psi$, which is our goal.

no longer holds. For example, the NBW $\mathcal{A}$ in Figure 2 recognizes the language L of all words with infinitely many 1s, yet when viewed as an NFW, it recognizes $(0+1)^+$, whose limit is $(0+1)^\omega$. As another example, the language of the NBW $\mathcal{A}'$ is empty, yet when viewed as an NFW, it recognizes the language $(0+1)^* \cdot 1$, whose limit is L. As demonstrated by the examples, the difficulty of the nondeterministic case originates from the fact that different prefixes of the infinite word may follow different accepting runs of the NFW, and there is no guarantee that these runs can be merged into a single run of the NBW. Accordingly, the best translation that was known until recently for going from an NFW to an NBW accepting its limit, or from an NBW to a limit NFW, is to first determinize the given automaton. This involves a $2^{O(n \log n)}$ blow up and gives up the potential succinctness of the nondeterministic model. On the other hand, no lower bound above $\Omega(n \log n)$ is known.

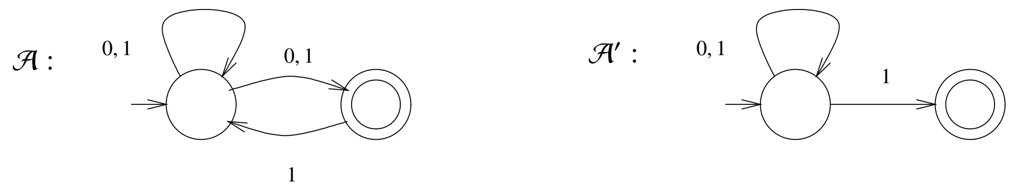

Figure 2: Relating NBWs and limit NFWs.

In [2] we study this exponential gap and tried to close it. In addition to the limit operator introduced by Landweber, we introduce and study two additional ways to induce a language of infinite words from a language of finite words: the *co-limit* of R is the set of all infinite words that have only finitely many prefixes in R. Thus, co-limit is dual to Landweber's limit. Also, the *persistent limit* of R is the set of all infinite words that have only finitely many prefixes not in R. Thus, eventually all the prefixes are in R.

Formally, we have the following.

Definition 10. Consider a language $R \subseteq \Sigma^*$. We define three languages of infinite words induced by R.

1. **[limit]** $lim(R) \subseteq \Sigma^\omega$ is the set of all words that have infinitely many prefixes in R. I.e., $lim(R) = \{w \mid w[1,i] \in R$ for infinitely many i's$\}$ [31].

2. **[co-limit]** $co\text{-}lim(R) \subseteq \Sigma^\omega$ is the set of all words that have only finitely many prefixes in R. I.e., $co\text{-}lim(R) = \{w \mid w[1,i] \in R$ for finitely many i's$\}$.

3. **[persistent limit]** $plim(R) \subseteq \Sigma^\omega$ is the set of all words that have only finitely many prefixes not in R. I.e., $plim(R) = \{w \mid w[1,i] \in R$ for almost all i's$\}$.

For example, for $R = (a + b)^*b$, the language $lim(R)$ consists of all words that have infinitely many b's, $co\text{-}lim(R)$ is the language of words that have finitely many b's, and $plim(R)$ is the language of words that have finitely many a's. For an NFW $\mathcal{A}$, we use $lim(\mathcal{A})$, $co\text{-}lim(\mathcal{A})$, and $plim(\mathcal{A})$, to denote $lim(L(\mathcal{A}))$, $co\text{-}lim(L(\mathcal{A}))$, and $plim(L(\mathcal{A}))$, respectively. The three limit operators are dual in the following sense:

Lemma 11. *For every $R \subseteq \Sigma^*$, we have*

$$comp(lim(R)) = co\text{-}lim(R) = plim(comp(R)).$$

Below we describe the main results of [2], which studies the relative succinctness of NBWs, NCWs, and NFWs whose limit, co-limit, and persistent limit correspond to the NBW and NCW.

We first need some notations. Consider an NFW $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, F \rangle$. For two sets of states $P, S \subseteq Q$, we denote by $L_{P,S}$ the language of $\mathcal{A}$ with initial set P and accepting set S. Formally, $L_{P,S}$ is the language accepted by the NFW $\langle \Sigma, Q, \delta, P, S \rangle$. If S or P are singletons we omit the curly braces, so instead of $L_{\{p\},S}$ we write $L_{p,S}$, etc.

Theorem 12 is a key theorem in beating the "determinize first" approach. It implies that the transition from an NFW $\mathcal{A}$ to an NBW for $lim(\mathcal{A})$ need not involve a determinization of $\mathcal{A}$. Indeed, we can specify $lim(\mathcal{A})$ as the union of languages that are generated by automata with a structure similar to the structure of $\mathcal{A}$. Formally, we have the following.

Theorem 12. [2] *For every NFW $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, F \rangle$,*

$$lim(\mathcal{A}) = \bigcup_{p \in Q} L_{Q_0,p} \cdot (L_{p,p} \cap L_{p,F})^\omega.$$

Given $\mathcal{A}$ with n states, Theorem 12 implies that an NBW accepting $lim(\mathcal{A})$ can be constructed by intersection, application of ω, concatenation, and union, starting with NFWs with n states. Exploiting the the similarity in the structure of the involved NFWs, the resulting NBW has $O(n^2)$ states.

Corollary 13. *Given an NFW $\mathcal{A}$ with n states, there is an NBW $\mathcal{A}'$ with $O(n^2)$ states such that $L(\mathcal{A}') = lim(L(\mathcal{A}))$.*

Corollary 13 implies that going from an NFW to an NBW for its limit, it is possible to do better than determinize the NFW. On the other hand, it is shown in [2] that going from an NFW to an NCW for its co-limit or persistent limit, an exponential blow-up cannot be avoided, and determinization is optimal.

Further results of [2] study succinctness among NFWs to which different limit operators are applied. For example, in Theorem 14 below we prove that going from a persistent limit NFW to a limit NFW involves an exponential blow up. In other words, given an NFW $\mathcal{A}$ whose persistent limit is L, translating $\mathcal{A}$ to an NFW whose limit is L may involve an exponential blow up. Note that persistent limit and limit are very similar – both require the infinite word to have infinitely many prefixes in $L(\mathcal{A})$, only that the persistent limit requires, in addition, that only finitely many prefixes are not in $L(\mathcal{A})$. This difference, which is similar to the difference between NBW and NCW, makes persistent limit exponentially more succinct. Technically, it follows from the fact that persistent limit NFWs inherit the power of alternating automata. In a similar, though less surprising way, co-limit NFWs inherit the power of complementation, and are also exponentially more succinct.

Theorem 14. [2] *For every $n \geq 1$, there is a language $L_n \subseteq \Sigma^\omega$ such that there are NFWs $\mathcal{A}$ with $O(n)$ states, and $\mathcal{A}'$ with $O(n^2)$ states, such that co-lim$(\mathcal{A})$ = plim$(\mathcal{A}')$ = L_n but an NFW $\mathcal{A}''$ such that lim$(\mathcal{A}'') = L_n$ must have at least 2^n states.*

Proof: Consider the language $L_n \subseteq \{0,1\}^\omega$ of all words w such that $w = uuz$, with $|u| = n$. We prove that an NFW $\mathcal{A}''$ such that $lim(\mathcal{A}'') = L_n$ must remember subsets of size n, and thus must have at least 2^n states. For a word $u \in \Sigma^n$, let S_u be the set of states that $\mathcal{A}''$ can reach after reading u. Since $u \cdot u \cdot 0^\omega \in L_n$, the NFW $\mathcal{A}''$ must accept from S_u infinitely many prefixes of $u \cdot 0^\omega$. Since S_u is finite, there is a subset $S'_u \subseteq S_u$, such that for every $s \in S'_u$ we have that $\mathcal{A}''$ accepts infinitely many prefixes of $u \cdot 0^\omega$. Assume by way of contradiction that $\mathcal{A}''$ has less than 2^n states. Then, there are two different words $u, v \in \Sigma^n$, such that $S'_u \cap S'_v \neq \emptyset$. Consider a state $s \in (S'_u \cap S'_v)$. It follows that $\mathcal{A}''$ can reach s by reading u, and accept from s infinitely many prefixes of $v \cdot 0^\omega$, contradicting the fact that $u \cdot v \cdot 0^\omega \notin L_n$.

In order to construct small NFW for the co-limit and persistent limit operators, we observe that a word w is in L_n iff $\bigwedge_{i=1}^{n}(w[i] = w[n+i])$. In the case of co-limit, we can check that only finitely many (in fact, 0) prefixes h of an input word are such that $h[i] \neq h[i+n]$ for some $1 \leq i \leq n$. This is done by letting $\mathcal{A}$ guess a position $1 \leq i \leq n$, remember $h[i]$, and accept a word iff the letter that comes n positions after it (that is, in $h[i+n]$) is different. It is easy to see that $\mathcal{A}$ requires only $O(n)$ states. A word w has finitely many prefixes in $L(\mathcal{A})$ iff $w \in L_n$. Hence, $co\text{-}lim(\mathcal{A}') = L_n$.

The case of persistent limit is much harder, as we cannot use the implicit complementation used in the co-limit case. Instead, we use the universal nature of persistence. We define the NFW $\mathcal{A}'$ as a union of n gadgets $\mathcal{A}'_1, \ldots, \mathcal{A}'_n$. The gadget $\mathcal{A}'_i$ is responsible for checking that $w[i] = w[n+i]$. In order to make sure that the conjunction on all $1 \leq i \leq n$ is satisfied, we further limit $\mathcal{A}'_i$ to accept only words of length $i \bmod n$. Hence, $\mathcal{A}'_i$ accepts a word $u \in \Sigma^*$ iff $u[i] = u[n+i] \wedge |u| = i \bmod n$. Consequently, if $w[i] \neq w[n+i]$, then all the prefixes of w of length $i \bmod n$ are rejected by $\mathcal{A}'$. On the other hand, if only a finite number of prefixes of an infinite word are not accepted by $\mathcal{A}'$, then for all $1 \leq i \leq n$, only a finite number of prefixes of length $i \bmod n$ are not accepted by $\mathcal{A}'_i$. Thus, a word w is in $plim(\mathcal{A}')$ iff for every $1 \leq i \leq n$, almost all the prefixes of w of length $i \bmod n$ are accepted by $\mathcal{A}'_i$. Hence, $w \in plim(\mathcal{A}')$ iff for all $1 \leq i \leq n$ we have that $w[i] = w[n+i]$, and we are done. Since each of the gadgets has $O(n)$ states, and $\mathcal{A}'$ needs n gadgets, it has $O(n^2)$ states. Figure 3 describes the gadget $\mathcal{A}'_i$ (all non-labeled transitions in the figure are labeled by both 0 and 1). $\square$

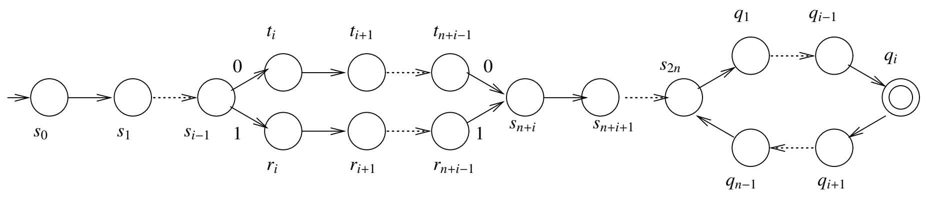

Figure 3: The gadget $\mathcal{A}'_i$

4.2 Discussion

The exponential gap between the known upper and the lower bound in the translation of NBW to NCW is particularly annoying: the upper bound is $2^{O(n \log n)}$ and for the lower bound we do not even have an example of a language whose NCW needs one more state than the NBW. The example in the proof of Lemma 9 shows an advantage of the Büchi condition. In a recent work with Benjamin Aminof, we hope to turn this advantage into a lower bound. The idea is as follows. NCWs cannot recognize the language of all words that have infinitely many occurrences of some letter. Indeed, DBWs cannot recognize the complement language [31]. Thus, a possible way to obtain a lower bound for the translation of NBW to NCW is to construct a language that is recognizable by an NCW, but for which an NCW needs more states than an NBW due to its inability to recognize infinitely many occurrences of a letter. One such candidate is the family of languages $L_1, L_2, \ldots$ over the alphabet $\{a, b\}$, where L_k contains exactly all words that have at least k occurrences of the letter a and at least k occurrences of the letter b. An NBW can follow the idea of the NBW in Figure 1: since every infinite word has infinitely many a's or infinitely many b's, the NBW for L can guess which of the two letters occurs infinitely often, and count k occurrences of the second letter. Thus, the NBW is the union of two components, one looking for k occurrences of a followed by infinitely many b's and the other looking for k occurrences of b followed by infinitely many a's. This can be done with $2k + 1$ states. We conjecture that an NCW must have roughly twice as many states. The reason is that an NCW with less than k states accepting all words with infinitely many a's, inevitably also accepts a word with less than k a's. Thus, we suspect that the best an NCW can do is to decompose the problem to "k occurrences of a followed by k occurrences of b or k occurrences of b followed by k occurrences a", which requires four counters to k.

References

[1] R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M.Y. Vardi, and Y. Zbar. The ForSpec temporal logic: A new temporal property-specification logic. In *Proc. 8th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2280 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 2002.

[2] B. Aminof and O. Kupferman. On the succinctness of nondeterminizm. In *4th Int. Symp. on Automated Technology for Verification and Analysis*, volume 4218 of *Lecture Notes in Computer Science*, pages 125–140. Springer, 2006.

[3] B. Alpern and F.B. Schneider. Defining liveness. *Infornation Processing Letters*, 21:181–185, 1985.

[4] I. Beer, S. Ben-David, C. Eisner, D. Fisman, A. Gringauze, and Y. Rodeh. The temporal logic Sugar. In *Proc 13th Int. Conf. on Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 363–367. Springer, 2001.

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

[6] R.S. Boyer and J.S. Moore. Proof-checking, theorem-proving and program verification. Technical Report 35, Institute for Computing Science and Computer Applications, 1983.

[7] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Int. Congress on Logic, Method, and Philosophy of Science. 1960*, pages 1–12. Stanford University Press, 1962.

[8] E. M. Clarke, A. Bierea, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

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

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

[11] E.A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 328–337, 1988.

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

[13] A.E. Emerson and A.P. Sistla. Deciding full branching time logics. *Information and Control*, 61(3):175–201, 1984.

[14] E. Friedgut, O. Kupferman, and M.Y. Vardi. Büchi complementation made tighter. In *2nd Int. Symp. on Automated Technology for Verification and Analysis*, volume 3299 of *Lecture Notes in Computer Science*, pages 64–78. Springer, 2004.

[15] D.M. Gabbay. Applications of trees to intermediate logics i. *Journal of Symbolic Logic*, 37:135–138, 1972.

[16] G. De Giacomo and M. Lenzerini. Concept languages with number restrictions and fixpoints, and its relationship with μ-calculus. In *Proc. 11th European Conf. on Artificial Intelligence (ECAI-94)*, pages 411–415. John Wiley and Sons, 1994.

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

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

[19] H. Iwashita and T. Nakata. Forward model checking techniques oriented to buggy designs. In *Proc. IEEE/ACM Int. Conf. on Computer Aided Design*, pages 400–404, 1997.

[20] Accellera Organization Inc. http://www.accellera.org, 2006.

[21] O. Kupferman and R. Lampert. On the construction of fine automata for safety properties. In *4th Int. Symp. on Automated Technology for Verification and Analysis*, volume 4218 of *Lecture Notes in Computer Science*, pages 110–124. Springer, 2006.

[22] O. Kupferman, G. Morgenstern, and A. Murano. Typeness for ω-regular automata. In *2nd Int. Symp. on Automated Technology for Verification and Analysis*, volume 3299 of *Lecture Notes in Computer Science*, pages 324–338. Springer, 2004.

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

[24] D. Kozen and R. Parikh. A decision procedure for the propositional μ-calculus. In *Logics of Programs*, volume 164 of *Lecture Notes in Computer Science*, pages 313–325. Springer, 1984.

[25] S.C. Krishnan, A. Puri, and R.K. Brayton. Deterministic ω-automata vis-a-vis deterministic Büchi automata. In *Algorithms and Computations*, volume 834 of *Lecture Notes in Computer Science*, pages 378–386. Springer, 1994.

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

[27] O. Kupferman and M.Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

[28] O. Kupferman and M.Y. Vardi. From linear time to branching time. *ACM Transactions on Computational Logic*, 6(2):273–294, 2005.

[29] O. Kupferman and M.Y. Vardi. Safraless decision procedures. In *Proc. 46th IEEE Symp. on Foundations of Computer Science*, pages 531–540, 2005.

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

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

[32] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th ACM Symp. on Principles of Programming Languages*, pages 97–107, 1985.

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

[34] R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.

[35] A. R. Meyer. Weak monadic second order theory of successor is not elementary recursive. In *Proc. Logic Colloquium*, volume 453 of *Lecture Notes in Mathematics*, pages 132–154. Springer, 1975.

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

[37] M. Michel. Complementation is more difficult with automata on infinite words. CNET, Paris, 1988.

[38] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1992.

[39] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Safety*. Springer, 1995.

[40] S. Owre, R.E. Shankar, and J.M. Rushby. *User guide for the PVS specification and verification system*. CSL, 1995.

[41] A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.

[42] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, 1989.

[43] M.O. Rabin. Decidability of second order theories and automata on infinite trees. *Transaction of the AMS*, 141:1–35, 1969.

[44] K. Ravi, R. Bloem, and F. Somenzi. A comparative study of symbolic algorithms for the computation of fair cycles. In *Proc. 3rd Int. Conf. on Formal Methods in Computer-Aided Design*, number 1954 in Lecture Notes in Computer Science, pages 143–160. Springer, 2000.

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

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

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

[48] R.S. Street and E.A. Emerson. An elementary decision procedure for the μ-calculus. In *Proc. 11th Int. Colloq. on Automata, Languages, and Programming*, volume 172, pages 465–472. Springer, 1984.

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

[50] R.S. Streett. Propositional dynamic logic of looping and converse. *Information and Control*, 54:121–141, 1982.

[51] A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. In *Proc. 12th Int. Colloq. on Automata, Languages, and Programming*, volume 194, pages 465–474. Springer, 1985.

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

[53] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st IEEE Symp. on Logic in Computer Science*, pages 332–344, 1986.

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

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

[56] T. Wilke. CTL$^+$ is exponentially more succinct than CTL. In *Proc. 19th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 1738 of *Lecture Notes in Computer Science*, pages 110–121. Springer, 1999.

[57] P. Wolper, M.Y. Vardi, and A.P. Sistla. Reasoning about infinite computation paths. In *Proc. 24th IEEE Symp. on Foundations of Computer Science*, pages 185–194, 1983.

[58] Q. Yan. Lower bounds for complementation of ω-automata via the full automata technique. In *Proc. 33rd Int. Colloq. on Automata, Languages, and Programming*, volume 4052 of *Lecture Notes in Computer Science*, pages 589–600. Springer, 2006.