Very good paper. Take as first reference for automata-theoretic model checking.

Automata-Theoretic Model Checking Revisited [⋆]

Moshe Y. Vardi [⋆⋆]

Rice University, Department of Computer Science, Houston, TX 77251-1892, U.S.A.
Email:vardi@cs.rice.edu, URL: http://www.cs.rice.edu/~vardi

system

Abstract. In automata-theoretic model checking we compose the design under verification with a Büchi automaton that accepts traces violating the specification. We then use graph algorithms to search for a counterexample trace. The basic theory of this approach was worked out in the 1980s, and the basic algorithms were developed during the 1990s. Both explicit and symbolic implementations, such as SPIN and and SMV, are widely used. It turns out, however, that there are still many gaps in our understanding of the algorithmic issues involved in automata-theoretic model checking. This paper covers the fundamentals of automata-theoretic model checking, reviews recent progress, and outlines areas that require further research.

1 Introduction

Formal verification is a process in which mathematical techniques are used to guarantee the correctness of a design with respect to some specified behavior. Automated formal-verification tools, such as COSPAN [48], SPIN [50] and SMV [17,63], based on *model-checking technology* [21,67], have enjoyed a substantial and growing use over the last few years, showing an ability to discover subtle flaws that result from extremely improbable events [23]. While until recently these tools were viewed as of academic interest only, they are now routinely used in industrial applications, resulting in decreased time to market and increased product integrity [24,25,58]. It is fair to say that automated verification is one of the most successful applications of automated reasoning in computer science.

As model-checking technology matured, the demand for specification language of increased expressiveness increased interest in linear-time formalisms [3]. The automata-theoretic approach offers a uniform algorithmic framework for model checking linear-time properties [57,81,83] It turns out, however, that there are still many gaps in our understanding of the algorithmic issues involved in automata-theoretic model checking. This paper covers the fundamentals of automata-theoretic model checking, reviews recent progress, and outlines areas that require further research.

[⋆] I am grateful to Orna Kupferman and Kristin Rozier for their comments on earlier drafts of this article.

[⋆⋆] Supported in part by NSF grants CCR-9988322, CCR-0124077, CCR-0311326, and ANI-0216467, by BSF grant 9800096, and by a grant from the Intel Corporation.

2 Basic Theory

Specification

The first step in formal verification is to come up with a *formal specification* of the design, consisting of a description of the desired behavior. One of the more widely used specification languages for designs is *temporal logic* [65]. In *linear* temporal logics, time is treated as if each moment in time has a unique possible future. Thus, linear temporal formulas are interpreted over linear sequences, and we regard them as describing the behavior of a single computation of a system. (An alternative approach is to use *branching* time. For a discussion of linear vs. branching time, see [82].)

In the linear temporal logic LTL, formulas are constructed from a set $Prop$ of atomic propositions using the usual Boolean connectives as well as the unary temporal connectives X ("next"), F ("eventually"), G ("always"), and the binary temporal connective U ("until"). For example, the LTL formula $G(\mathit{request} \rightarrow F\ \mathit{grant})$, which refers to the atomic propositions *request* and *grant*, is true in a computation precisely when every state in the computation in which *request* holds is followed by some state in the future in which *grant* holds. The LTL formula $G(\mathit{request} \rightarrow (\mathit{request}\ U\ \mathit{grant}))$ is true in a computation precisely if, whenever *request* holds in a state of the computation, it holds until a state in which *grant* holds is reached. In LTL model checking we assume that the specification is given in terms of properties expressed by LTL formulas.

Kripke structures

Nice summary of the LTL semantics

LTL is interpreted over *computations*, which can be viewed as infinite sequences of truth assignments to the atomic propositions; i.e., a computation is a function $\pi : \mathbb{N} \rightarrow 2^{Prop}$ that assigns truth values to the elements of $Prop$ at each time instant (natural number). For a computation π and a point $i \in \mathbb{N}$, the notation $\pi, i \models \varphi$ indicates that a formula φ holds at the point i of the computation π. In particular, $\pi, i \models X\varphi$ if $\pi, i+1 \models \varphi$, and $\pi, i \models \varphi U \psi$ if for some $j \geq i$, we have $\pi, j \models \psi$ and for all k, $i \leq k < j$, we have $\pi, k \models \varphi$. The connectives F and G can be defined in terms of the connective U: $F\varphi$ is defined as $\mathbf{true}\ U\varphi$, and $G\varphi$ is defined as $\neg F \neg \varphi$. We say that π *satisfies* a formula φ, denoted $\pi \models \varphi$, iff $\pi, 0 \models \varphi$. We denote by $\mathrm{models}(\varphi)$ the set of computations satisfying φ.

System

Designs can be described using a variety of formalisms. Regardless of the formalism used, a *finite-state design* can be abstractly viewed as a *labeled transition system*, i.e., as a structure of the form $M = (W, W_0, R, V)$, where W is the finite set of states that the system can be in, $W_0 \subseteq W$ is the set of initial states of the system, $R \subseteq W^2$ is a transition relation that indicates the allowable state transitions of the system, and $V : W \rightarrow 2^{Prop}$ assigns truth values to the atomic propositions in each state of the system. (A labeled transition system is essentially a Kripke structure.) A *path* in M that *starts at* u is a possible infinite behavior of the system starting at u, i.e., it is an infinite sequence $u_0, u_1 \ldots$ of states in W such that $u_0 = u$, and $(u_i, u_{i+1}) \in R$ for all $i \geq 0$. The sequence $V(u_0), V(u_1) \ldots$ is a *computation* of M that *starts at* u. It is the sequence of truth assignments visited by the path, and can be viewed as a function from $\mathbb{N}$ to 2^{Prop}. The *language* of M, denoted $L(M)$, consists of all computations of M that start at a state in W_0. Note that $L(M)$ can be viewed as a language of infinite words over the alphabet 2^{Prop}. The language $L(M)$ can be viewed as an abstract description of the system M, describing all possible "traces". We say that M *satisfies* an LTL formula φ if all computations in $L(M)$ satisfy φ, that is, if $L(M) \subseteq \mathrm{models}(\varphi)$. When M satisfies

Interpretation for an LTL formula
(specification)

φ we also say that M is a model of φ, which explains why the technique is known as *model checking* [23].

One of the major approaches to automated verification is the *automata-theoretic approach*, which underlies model checkers that can handle linear-time specifications (for a precursor, see [61]). The key idea underlying the automata-theoretic approach is that, given an LTL formula φ, it is possible to construct a finite-state automaton A_φ on infinite words that accepts precisely all computations that satisfy φ. The type of finite automata on infinite words we consider is the one defined by Büchi [13]. A *Büchi automaton* is a tuple $A = (\Sigma, S, S_0, \rho, F)$, where Σ is a finite alphabet, S is a finite set of states, $S_0 \subseteq S$ is a set of initial states, $\rho : S \times \Sigma \rightarrow 2^S$ is a nondeterministic transition function, and $F \subseteq S$ is a set of accepting states. A *run* of A over an infinite word $w = a_1 a_2 \cdots$, is a sequence $s_0 s_1 \cdots$, where $s_0 \in S_0$ and $s_i \in \rho(s_{i-1}, a_i)$ for all $i \geq 1$. A run $s_0, s_1, \ldots$ is *accepting* if there is some accepting state that repeats infinitely often, i.e., for some $s \in F$ there are infinitely many i's such that $s_i = s$. The infinite word w is *accepted* by A if there is an accepting run of A over w. The *language* of infinite words accepted by A is denoted $L(A)$. The following fact establishes the correspondence between LTL and Büchi automata [84] (for a tutorial introduction to this correspondence, see [81]):

Theorem 1. *Given an LTL formula φ, one can build a Büchi automaton $A_\varphi = (\Sigma, S, S_0, \rho, F)$, where $\Sigma = 2^{Prop}$ and $|S| \leq 2^{O(|\varphi|)}$, such that $L(A_\varphi) = models(\varphi)$.*

All computations (Kripke structures) that satisfy phi

This correspondence reduces the verification problem to an automata-theoretic problem as follows [83]. Suppose that we are given a system M and an LTL formula φ. We check whether $L(M) \subseteq \text{models}(\varphi)$ as follows: (1) construct the automaton $A_{\neg\varphi}$ that corresponds to the *negation* of the formula φ (this automaton is called the *complementary* automaton), (2) take the *cross product* of the system M and the automaton $A_{\neg\varphi}$ to obtain an automaton $A_{M,\varphi}$, such that $L(A_{M,\varphi}) = L(M) \cap L(A_{\neg\varphi})$, and (3) check whether the language $L(A_{M,\varphi})$ is empty, i.e., $A_{M,\varphi}$ accepts *no* input.

Theorem 2. *Let M be a labeled transition system and φ be an LTL formula. Then M satisfies φ iff $L(A_{M,\varphi}) = \emptyset$.*

If $L(A_{M,\varphi})$ is empty, then the design is correct. Otherwise, the design is incorrect and the word accepted by $L(A_{M,\varphi})$ is an incorrect computation.

The *emptiness* problem for an automaton is to decide, given an automaton A, whether $L(A) = \emptyset$, i.e., if the automaton accepts no word. Algorithms for emptiness are based on testing *fair reachability* in graphs: an automaton is *nonempty* if starting from some initial state we can reach an accepting state from where there is a cycle back to itself [16]. An algorithm for nonemptiness is the following: (i) decompose the transition graph of the automaton into *maximal strongly connected components (*MSCC*s)* (linear cost depth-first search [27]); (ii) verify that one of the MSCCs intersects with F (linear cost). More sophisticated Büchi nonemptiness algorithms have been studied, e.g., [28,34]. When the automaton is nonempty, nonemptiness algorithms return a witness in the shape of a "lasso": an initial finite prefix followed by a finite cycle. (If the accepting states are "sink" states, then the finite cycle following the initial prefix can be ignored.) Thus, once the automaton $A_{\neg\varphi}$ is constructed, the verification task is reduced to automata-theoretic problems, namely, intersecting automata and testing emptiness of automata, which have highly efficient solutions [81]. Furthermore, using data structures that enable compact representation of very large state spaces makes it possible to verify designs of significant complexity [8,14].

The linear-time framework is not limited to using LTL as a specification language. ForSpec and PSL are recent extensions of LTL, designed to address the need of the semiconductor industry [1,3]. There are also those who prefer to use automata on infinite words as a specification formalism [84]; in fact, this is the approach of COSPAN [48,57]. In this approach, we are given a design represented as a finite transition system M and a property represented by a Büchi (or a related variant) automaton P. The design is correct if all computations in $L(M)$ are accepted by P, i.e., $L(M) \subseteq L(P)$. This approach is called the *language-containment* approach. To verify M with respect to P, we: (1) construct the automaton P^c that *complements* P, (2) take the product of the system M and the automaton P^c to obtain an automaton $A_{M,P}$, and (3) check that the automaton $A_{M,P}$ is nonempty. As before, the design is correct iff $A_{M,P}$ is empty. Thus, the verification task is again reduced to automata-theoretic problems, namely complementing and intersecting automata and testing emptiness of automata.

Use an omega-automaton
directly for the specification

3 Automata-Theoretic Model Checking Revisited

By the late 1990s, the automata-theoretic approach to model checking seems to have stabilized. The algorithms developed can be classified as as *explicit*, based on explicit state enumeration, e.g., [28,43], or *implicit/symbolic*, based on a symbolic encoding of the state space, using binary decision diagrams (BDDs) [14] or satisfiability solving [8]. These algorithms have been implemented in various model-checking tools [17,50,63].

In the last few years, further progress has been been on several aspects of automata-theoretic model checking. As a result of this progress, we know both more and less. We now know that the simple picture that prevailed by the late 1990s is too simplistic, but we do not have a clear understanding of the space of relevant algorithms. In the rest of this section, we survey the progress made over the last few years and highlight the questions that have been opened by this progress.

3.1 Translating LTL Formulas to Büchi Automata

Translating LTL formulas to automata is a key building block in linear-time model checking. While the focus of the original translation [87,84] was on mathematical simplicity, it was not appropriate for explicit model checking, since the automata constructed were always exponential in the size of the formula. Already in [86] it was shown that instead of starting with an exponentially large state pace, the translation can create states on a demand-driven basis. The optimized translation of [43] avoided the exponential blow-up in many cases of practical interest and was was used in the explicit model checker SPIN [50]. The original translation of [84], was appropriate for symbolic model checking and, after appropriate optimization [22], is used in symbolic model checkers such as NuSMV [17]. An approach to LTL translation via alternating automata was described in [80], again motivated by mathematical simplicity.

Two papers published in 1999 [29,31] showed that [43] is not the last word on explicit LTL translation, which opened the door to many more papers [32,35,38,39, 46,44,40,76,73,79]. In fact, so many papers have been published over the last few years on this topic that it is difficult to say what is the best approach to translating LTL to automata. This is compounded by several issues:

- All the cited papers focus on optimizing automata generation (with respect to time and/or space), rather than optimizing model checking. It is not clear, however,that improving automata-generation performance yields an improvement in model-checking performance. One exception is [73], which aims at optimizing model checking by generating "more deterministic" automata, but again does not offer any evidence of improvement in model checking.
- There are reasons to believe that none of the existing LTL translators perform well on nontrivial formulas. For example, [74] reports not being able to translate the following formula, expressing a conjunction of fairness conditions, by any of the available tools.

$$\begin{array}{l}((GFp_0 \rightarrow GFp_1)\&(GFp_2 \rightarrow GFp_0)\& \\ (GFp_3 \rightarrow GFp_2)\&(GFp_4 \rightarrow GFp_2)\& \\ (GFp_5 \rightarrow GFp_3)\&(GFp_6 \rightarrow GF(p_5 \vee p_4))\& \\ (GFp_7 \rightarrow GFp_6)\&(GFp_1 \rightarrow GFp_7)) \rightarrow GFp_8\end{array}$$

  A specialized tool generated an automaton with 1281 states from this formula. Note that symbolic model checkers routinely handle BDDs with millions of nodes. It is not clear why LTL translators cannot handle automata with only thousands of states.
- A *generalized* Büchi automaton is a tuple $A = (\Sigma, S, S_0, \rho, \mathbf{F})$, where $\mathbf{F}$ is a set $\{F_1, \ldots, F_k\}$ of subsets of S, called *accepting* sets. A run of A is accepting if each accepting set is visited infinitely often. It is known that a generalized Büchi automaton with k accepting sets can be *degeneralized*, that is, converted to an equivalent Büchi automaton, at the cost of multiplying the number of states by k [16]. As is shown in [43], it is natural to translate LTL to generalized Büchi automata. While symbolic model checkers support generalized Büchi automata, SPIN does not support them and requires degeneralization. There are, however, some who argue that it may be advantageous to avoid degeneralization [78]; see discussion of nonemptiness algorithms below.
- Industrial experience has shown that LTL is too weak expressively for industrial applications (see [85] for theoretical justification), resulting in more expressive industrial language such as ForSpec and PSL [1,3]. So far there he been no report of an effort to develop an explicit translator for ForSpec or PSL. Some industrial symbolic implementations of ForSpec and PSL are known to exists; for example, Intel has a symbolic translator for ForSpec [4], but little is known about them. See [15,18,66] for recent descriptions of symbolic translations for certain fragments of PSL. None of these translations handle all features of PSL.

3.2 Deterministic vs. Nondeterministic Automata

Translate a complemented property (temporal logic formula) to a deterministic or non-deterministic automaton?

For certain formulas, the very approach of translating temporal assertions to *nondeterministic* Büchi automata should be re-visited. The majority of the properties being verified are *safety* properties, whose violation can be witnessed by a finite counterexample. It is known that in such cases the complemented properties can be translated into automata on finite words [53] (see also [52]).[1] Such automata can be determinized, though at a possibly exponential cost [68]. It may seem that such blow-up should be avoided, but symbolic model checking can be viewed as online determinization of the assertion automaton [53]. Thus, determinization is in some sense inherent to symbolic model checking.

Recent results point to the advantage of translation to deterministic automata in the context of SAT-based model checking [2]. Unlike the standard propositional encoding of LTL formulas [8, 19, 59], which is polynomial in the size of the formula, the encoding in [2] is exponential. It is shown in [2] that such encoding can nevertheless lead to improved model-checking performance. When the automaton is nondeterministic, the model checker has to find a bad behavior of the design under verification as well as an accepting run of the automaton on that behavior. When the automaton is deterministic, the search for an accepting run is avoided. This result raises the possibility that translation to deterministic automata would also be advantageous in the context of explicit and BDD-based model checking. (A theoretical advantage of translating to deterministic automata is described in [56], but it is not clear if this leads also to a practical advantage.)

3.3 Nonemptiness Algorithms

Is the language accepted by the automaton the empty language or not?

There are three types of nonemptiness algorithms for Büchi automata: explicit, BDD-based, and SAT-based.

Explicit Algorithms As mentioned earlier, an obvious algorithm for nonemptiness of Büchi automata is the following: (i) decompose the transition graph of the input automaton into MSCCs using depth-first search, and (ii) verify that one of the MSCCs intersects with the accepting set F (or with all members of $\mathbf{F}$ for generalized automata).

For large state spaces, maintaining the required data structures in main memory might be infeasible. An alternative algorithm, NDFS, was proposed in [28]. NDFS conducts two depth-first searches, but does not require a decomposition into maximal strongly connected components. This algorithm, with some modifications [45, 51], is the algorithm implemented in SPIN. NDFS was improved further in [71].

Other works [29, 41] developed optimized versions of the MSCC-based algorithm and argued that it performs better than NDFS and its variants. The experimental evidence is limited, however, to automata with state spaces of moderate size, while NDFS was designed for very large state spaces, where MSCC decomposition cannot be carried out in main memory. (NDFS can use state hashing, which underapproximates the set visited by the search [28].) The emerging picture is that MSCC-based algorithms are appropriate for main-memory implementations, whereas NDFS algorithms are appropriate when the state space is too large for a main-memory implementation.

[1] Interestingly, only few model-checking tools, e.g., VIS [11], take advantage of this observation, even though nonemptiness algorithms for automata on finite words are significantly simpler than those for automata on infinite words [81].

NDFS was extended to generalized Büchi automata in [78]; instead of conducting two depth-first searches, we may need to conduct $k + 1$ depth-first searches, where k is the number of accepting sets of the automaton. Thus, the blow-up in the size of the state space is replaced by a blow-up in the number of depth-first searches. It is not clear that this yields an improvement in performance.

A thorough discussion and experiments involving nonemptiness algorithms can be found in [30], which also introduces optimized versions of the MSCC-based algorithm of [29] and the NDFS-based algorithm of [78]. At this point these two algorithms seem to be the best of their types. These algorithms are implemented in the model-checking library *SPOT* [32], which is publicly available at `spot.lip6.fr`. For a survey of distributed algorithms for explicit model checking, see [6].

BDD-Based Algorithms In the symbolic approach, we do not construct the state graphs of the system and property automata explicitly. Rather, these graphs are described in a logical language, cf. [5]. The model-checking algorithms can then work directly on the symbolic representation. BDD-based model checkers such as SMV use propositional formulas as the user-level representation formalism. The tool then translates these formulas into Reduced Ordered Binary Decision Diagrams (BDDs) [12] and the nonemptiness algorithm works directly on these BDDs [14]. BDD-based algorithms are set based (the algorithms manipulate sets of states) and cannot directly implement depth-first search. In the symbolic approach, the property automaton also has to be represented symbolically [20, 14]; in fact, that representation captures directly the structure of the automaton described in [84]. While the explicit representation of the automaton can be exponentially large with respect to the LTL formula it represents, the symbolic representation is linear in the size of the formula.

A set-based algorithm for fair reachability was described in [34], based on a fixpoint characterization of fair reachability [33]. The algorithm performs a nested fixpoint computation, which implies that it uses, in the worst case, a quadratic number of symbolic operations (with respect to the number of nodes in the state graph). This should be contrasted with explicit nonemptiness algorithms, which run in linear time.

The algorithm of [34], referred to as the *EL algorithm*, is the one implemented on available symbolic model checkers [17, 63]. A heuristic optimization of the EL algorithm called *CTY*, was proposed in [49]. An improvement of CTY, called *OWCTY*, was proposed in [36], where it was argued that it is preferred to the standard EL algorithm, but this conclusion was disputed in [77].

Both CTY and OWCTY retain the structure of a nested fixpoint computation with a quadratic number of symbolic operations. In contrast, the algorithm presented in [9] uses only $n \log n$ symbolic operations. Disappointingly, this algorithm does not perform better in practice than EL and its variants [69]. Further improvement was provided in [42], which described an algorithm with a *linear* number of symbolic steps. Unfortunately, there is no experimental information on the performance of that algorithm in practice.

Another approach to the fair-reachability problem is to reduce it to a simple reachability problem [7]. This replaces the nested fixpoint computation by a simple fixpoint computation, at the cost of doubling the number of BDD variables. Practical performance of this algorithm has been disappointing [7]. On the other hand, it was shown in [10] that for a certain class of LTL formulas the nested fixpoint algorithms can be replaced by a simple fixpoint algorithm with no increase in the number of variables, resulting in significant performance improvement. (A general characterization of LTL formulas for which model checking can be performed without nested fixpoints is provided in [56]. That characterization, however, does not yield a practical algorithm.)

A *hybrid* approach to LTL symbolic model checking, that is, an approach that uses an explicit representation of the property automaton, whose state space is often quite manageable, and a symbolic representation of the system, whose state space is typically exceedingly large, was studied in [74] (following initial study in [72]): They compared the effects of using: (i) a purely symbolic representation of the property automaton, (ii) a symbolic representation, using binary encoding[2], of an explicitly compiled property automaton, and (iii) a partitioning of the symbolic state space according to an explicitly translated property automaton. This comparison was applied to three model-checking algorithms: the nested fixpoint algorithm of [34], the reduction of fair reachability to reachability of [7], and the simple fixpoint algorithm of [10]. The emerging picture from this comparison is quite clear; the hybrid approach outperforms pure symbolic model checking, while partitioning outperforms binary encoding. The conclusion is that the hybrid approaches benefits from state-of-the-art techniques in explicit compilation of LTL formulas. Also, partitioning gains from the fact that symbolic operations are applied to smaller sets of states.

SAT-Based Algorithms In *bounded* model checking we check whether there exists a counterexample trace of bounded size (that is, both the prefix and the cycles have to be of bounded size). As is shown [8], this can be expressed as a propositional formula whose satisfiability implies the existence of a counterexample. While propositional satisfiability is NP-complete, today's satisfiability-solving tools, known as *SAT solvers*, can solve instances with up to hundreds of thousands of variables [89]. It turned out that SAT-based bounded model checkers can handle designs that are order-of-magnitude larger than those handled by BDD-based model checkers, making this technology quite popular in the industry, cf. [26].

In spite of several papers on symbolic translation of LTL in the context of SAT-based model checking [8,19,59], we are far from having reached a solid understanding on the relative merits of the different approaches. These papers study various propositional encodings of LTL extended with past temporal connectives. These encodings compare favorably with what is referred to as "automata-theoretic encoding". The latter refers to a binary encoding of automata generated by some LTL translator. This encoding ignores the inner structure of automata states. In automata generated from LTL formulas, the states are sets of subformulas; a reasonable automata-theoretic encoding should then take the inner structure of states into account, rather than use an arbitrary binary encoding of states. Also, the reduction of fair reachability to reachability [7] has yet to be evaluated in the context of SAT-based model checking.

[2] That is, we encode n states by $\log n$ Boolean variables.

3.4 Büchi Properties

The property (specification) as a Büchi automaton

As mentioned earlier, in some cases it is desirable to specify properties directly in terms of Büchi automata, rather in terms of a temporal logic. In this case the automata-theoretic approach requires complementation of the property automaton. Note that while it is easy to complement properties given in terms of formulas in temporal logic, complementation of properties given in terms of nondeterministic automata is not simple. Indeed, a word w is rejected by a nondeterministic automaton A if all the runs of A on w rejects the word. Thus, the complementary automaton has to consider all possible runs, and complementation has the flavor of determinization.

For Büchi automata on infinite words, which are required for the modeling of liveness properties, optimal complementation constructions are quite complicated, as the subset construction is not sufficient. Due to the lack of a simple complementation construction, the user is typically required to specify the property by a deterministic Büchi automaton [57] (it is easy to complement a deterministic Büchi automaton), or to supply the automaton for the negation of the property [50]. Thus, an effective algorithm for the complementation of Büchi automata would be of significant practical value.

Efforts to develop simple complementation constructions for nondeterministic automata started early in the 1960s, motivated by decision problems for second-order logics. Büchi suggested a complementation construction for nondeterministic Büchi automata that involved a complicated combinatorial argument and a doubly-exponential blow-up in the state space [13]. Thus, complementing an automaton with n states resulted in an automaton with $2^{2^{O(n)}}$ states. In [75], Sistla et al. suggested an improved implementation of Büchi's construction, with only $2^{O(n^2)}$ states, which is still, however, not optimal. Only in [70], Safra introduced a determinization construction, which also enabled a $2^{O(n \log n)}$ complementation construction, matching a lower bound described by Michel [64] (cf. [62]). Thus, from a theoretical point of view, some considered the problem solved since 1988.

A careful analysis, however, of the exact blow-up in Safra's and Michel's bounds reveals an exponential gap in the constants hiding in the $O()$ notations: while the upper bound on the number of states in the complementary automaton constructed by Safra is n^{2n}, Michel's lower bound involves only an $n!$ blow up, which is roughly $(n/e)^n$. Recent efforts focused on narrowing the gap between the upper and lower bounds. A new complementation construction, which avoids determinization, was introduced in [54], and then tightened in [37] to yield an upper bound of $(0.97n)^n$. On the other hand, Michel's bound was improved in [88] to yield a lower bound to $(0.76n)^n$. Thus, the gap between the lower and upper bound has narrowed, but it is still exponentially wide. (For a study of the relationship between complementation and the OWCTY fair-reachability algorithm, see [55].)

The construction of [54] has been implemented with many added optimizations [47]. This optimized construction proved to be highly effective on Büchi automata obtained from LTL formulas. It is shown in [47] that the automaton obtained by complementing A_φ, for a random LTL formula φ, is not much larger than the automaton $A_{\neg\varphi}$. This, however, does not imply that the construction is equally effective when applied to generic Büchi properties. So far no tool supports model checking Büchi properties.

4 Concluding Remarks

Since its introduction in 1981, model checking has proved to be a highly successful technology. The automata-theoretic approach offers a uniform algorithmic framework for model checking linear-time properties. As we saw, recent progress has increased our knowledge, but also opened many questions, regarding the translation of temporal properties to automata, algorithms for fair reachability, and complementation of Büchi properties. We hope to see many of these questions answered in the coming years. Equally important, we hope to see software tools implementing new algorithmic developments in this area.

Main algorithmic challenges for
automata-theoretic model checking:
- translation from temporal logic formulas
  to automata
- algorithms for fair reachability
- complementation

It should be noted, however, that for many of the computational tasks involved in model checking there are several possible algorithms. Often, we do not have a solid understanding of the relative merits of these algorithms. Furthermore, it is not clear than we can alsways select a "best" algorithm from the pallette of possible algorithms. Perhaps it is time to abandon the "winner-takes-all" mentality that seem to pervade algorithmic research. In this, we can be inspired by recent research in Artificial Intelligence, which advocates a portfolio approach to algorithm selection [60]. In this approach, one matches algorithms to problem instances. Thus, rather then try to identify a "best-overall" algorithm, the focus shifts to attempting to identify a good algorithm to the problem instance at hand. As an example along these lines, we can mention the work reported at [10], which adapts the nonemptiness algorithm used to the LTL property at hand.

References

1. K. Albin et al. Property specification language reference manual. Technical Report Version 1.1, Accellera, 2004.
2. R. Armoni, S. Egorov, R. Fraer, D. Korchemny, and M.Y. Vardi. Efficient LTL compilation for SAT-based model checking. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 877–884, 2005.
3. 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 International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2280 of *Lecture Notes in Computer Science*, pages 296–211, Grenoble, France, April 2002. Springer-Verlag.
4. R. Armoni, D. Korchemny, A. Tiemeyer, and M.Y. Vardi Y. Zbar. Deterministic dynamic monitors for linear-time assertions. In *Proc. Workshop on Formal Approaches to Testing and Runtime Verification*, volume 4262 of *Lecture Notes in Computer Science*. Springer, 2006.
5. J. L. Balcázar and A. Lozano. The complexity of graph problems for succinctly represented graphs. In *Proc. Graph-Theoretic Concepts in Computer Science*, Lecture Notes in Computer Science 411, pages 277–285. Springer, 1989.
6. J. Barnat, L. Brim, and I. Cerna. Cluster-based LTL model checking of large systems. In *Proc. 4th Int'l Symp. on Formal Methods for Components and Objects*, volume 4111 of *Lecture Notes in Computer Science*, pages 259–279. Springer, 2006.
7. A. Biere, C. Artho, and V. Schuppan. Liveness checking as safety checking. In *Proc. 7th Int'l Workshop on Formal Methods for Industrial Critical Systems*, volume 66:2 of *Electr. Notes Theor. Comput. Sci.*, 2002.

8. 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.
9. 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.
10. R. Bloem, K. Ravi, and F. Somenzi. Efficient decision procedures for model checking of linear time logic properties. In *Computer Aided Verification, Proc. 11th International Conference*, volume 1633 of *Lecture Notes in Computer Science*, pages 222–235. Springer-Verlag, 1999.
11. R.K. Brayton, G.D. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S.-T. Cheng, S. Edwards, S. Khatri, T. Kukimoto, A. Pardo, S. Qadeer, R.K. Ranjan, S. Sarwary, T.R. Shiple, G. Swamy, and T. Villa. VIS: a system for verification and synthesis. In *Computer Aided Verification, Proc. 8th International Conference*, volume 1102 of *Lecture Notes in Computer Science*, pages 428–432. Springer-Verlag, 1996.
12. R.E. Bryant. Graph-based algorithms for boolean-function manipulation. *IEEE Trans. on Computers*, C-35(8), 1986.
13. J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. International Congress on Logic, Method, and Philosophy of Science. 1960*, pages 1–12, Stanford, 1962. Stanford University Press.
14. 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, June 1992.
15. D. Bustan, D. Fisman, and John Havlicek. Automata construction for psl. Technical report, The Weizmann Institute of Science, May 2005. Technical Report MCS05-04.
16. Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *Journal of Computer and System Sciences*, 8:117–141, 1974.
17. A. Cimatti, E.M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In *Proc. 14th Int'l Conf. on Computer Aided Verification*, Lecture Notes in Computer Science 2404, pages 359–364. Springer-Verlag, 2002.
18. A. Cimatti, M. Roveri, S. Semprini, and S. Tonetta. From psl to nba: A modular symbolic encoding. In *Proc. 6th Int'l Conf. on Formal Methods in Computer-Aided design*, 2006.
19. A. Cimatti, M. Roveri, and D. Sheridan. Bounded verification of past LTL. In *Proc. 5th Int'l Conf. on Formal Methods in Computer-Aided Design*, volume 3312 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 2004.
20. E. M. Clarke, O. Grumberg, and K. Hamaguchi. Another look at ltl model checking. *Formal Methods in System Design*, 10(1):47–71, 1997.
21. E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, January 1986.
22. E.M. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. In *Computer Aided Verification, Proc. 6th International Conference*, pages 415 – 427, Stanford, California, June 1994. Lecture Notes in Computer Science, Springer-Verlag.
23. E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
24. E.M. Clarke and R.P. Kurshan. Computer aided verification. *IEEE Spectrum*, 33:61–67, 1986.
25. E.M. Clarke and J.M. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 28:626–643, 1996.
26. F. Copty, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M.Y. Vardi. Benefits of bounded model checking at an industrial setting. In *Computer Aided Verification, Proc.*

13th International Conference, volume 2102 of *Lecture Notes in Computer Science*, pages 436–453. Springer-Verlag, 2001.

27. T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 1990.
28. 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.
29. J-M. Couvreur. On-the-fly verification of linear temporal logic. In *Proc. World Congress on Formal Methods*, pages 253–271, 1999.
30. J.M. Couvreur, A. Duret-Lutz, and D. Poitrenaud. On-the-fly emptiness checks for generalized Büchi automata. In *Proc. 12th Int'l SPIN Workshop on Model Checking of Software*, volume 3639 of *Lecture Notes in Computer Science*, pages 143–158. Springer, 2005.
31. N. Daniele, F. Guinchiglia, and M.Y. Vardi. Improved automata generation for linear temporal logic. In *Computer Aided Verification, Proc. 11th International Conference*, volume 1633 of *Lecture Notes in Computer Science*, pages 249–260. Springer-Verlag, 1999.
32. A. Duret-Lutz and Denis Poitrenaud. Spot: An extensible model checking library using transition-based generalized büchi automata. In *Proc. 12th Int'l Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 76–83. IEEE Computer Society, 2004.
33. E.A. Emerson and E.M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proc. 7th InternationalColloq. on Automata, Languages and Programming*, pages 169–181, 1980.
34. 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.
35. K. Etessami and G.J. Holzmann. Optimizing Büchi automata. In *Proc. 11th Int'l Conf. on Concurrency Theory*, Lecture Notes in Computer Science 1877, pages 153–167. Springer-Verlag, 2000.
36. K. Fisler, R. Fraer, G. Kamhi, M.Y. Vardi, and Z. Yang. Is there a best symbolic cycle-detection algorithm? In *7th International Conference on Tools and algorithms for the construction and analysis of systems*, number 2031 in Lecture Notes in Computer Science, pages 420–434. Springer-Verlag, 2001.
37. E. Friedgut, O. Kupferman, and M.Y. Vardi. Büchi complementation made tighter. *Int'l J. of Foundations of Computer Science*, 17(4):851–867, 2006.
38. C. Fritz. Constructing Büchi automata from linear temporal logic using simulation relations for alternating bchi automata. In *Proc. 8th Intl. Conference on Implementation and Application of Automata*, number 2759 in Lecture Notes in Computer Science, pages 35–48. Springer-Verlag, 2003.
39. C. Fritz. Concepts of automata construction from LTL. In *Proc. 12th Int'l Conf. on Logic for Programming, Artificial Intelligence, and Reasoning*, Lecture Notes in Computer Science 3835, pages 728–742. Springer-Verlag, 2005.
40. P. Gastin and D. Oddoux. Fast LTL to büchi automata translation. In *Computer Aided Verification, Proc. 13th International Conference*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65. Springer-Verlag, 2001.
41. J. Geldenhuys and A. Valmari. Tarjan's algorithm makes on-the-fly LTL verification more efficient. In *Proc. 10th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science 2988, pages 205–219. Springer-Verlag, 2004.
42. R. Gentilini, C. Piazza, and A. Policriti. Computing strongly connected components in a linear number of symbolic steps. In *14th ACM-SIAM Symposium on Discrete Algorithms*, pages 573–582, Baltimore, Maryland, 2003.

43. 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, August 1995.
44. D. Giannakopoulou and F. Lerda. From states to transitions: Improving translation of LTL formulae to büchi automata. In *Proc. 22nd IFIP Int'l Conf. on Formal Techniques for Networked and Distributed Systems*, pages 308–326, 2002.
45. P. Godefroid and G.J. Holzmann. On the verification of temporal properties. In *Proc. 13th Int. Conf on Protocol Specification Testing and Verification*, pages 109–124, 1993.
46. S. Gurumurthy, R. Bloem, and F. Somenzi. Fair simulation minimization. In *Computer Aided Verification, Proc. 14th International Conference*, volume 2404 of *Lecture Notes in Computer Science*, pages 610–623. Springer-Verlag, 2002.
47. S. Gurumurthy, O. Kupferman, F. Somenzi, and M.Y. Vardi. On complementing nondeterministic Büchi automata. In *12th Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, volume 2860 of *Lecture Notes in Computer Science*, pages 96–110. Springer-Verlag, 2003.
48. R.H. Hardin, Z. Har'el, and R.P. Kurshan. COSPAN. In *Computer Aided Verification, Proc. 8th International Conference*, volume 1102 of *Lecture Notes in Computer Science*, pages 423–427. Springer-Verlag, 1996.
49. R.H. Hardin, R.P. Kurshan, S.K. Shukla, and M.Y. Vardi. A new heuristic for bad cycle detection using BDDs. *Formal Methods in System Design*, 18:131–140, 2001.
50. G.J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Engineering*, 23(5):279–295, May 1997. Special issue on Formal Methods in Software Practice.
51. G.J. Holzmann, D. Peled, and M. Yannakakis. On nested depth-first search. In *Proc. 2nd Spin Workshop on The Spin Verification System*, pages 23–32. American Math. Soc., 1996.
52. O. Kupferman and R. Lampert. On the construction of fine automata for safety properties. In *4th International Symposium on Automated Technology for Verification and Analysis*, volume 4218 of *Lecture Notes in Computer Science*, pages 110–124. Springer-Verlag, 2006.
53. O. Kupferman and M.Y. Vardi. Model checking of safety properties. *Formal methods in System Design*, 19(3):291–314, November 2001.
54. O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. on Computational Logic*, 2(2):408–429, July 2001.
55. O. Kupferman and M.Y. Vardi. From complementation to certification. *Theoretical Computer Science*, 305:591–606, 2005.
56. O. Kupferman and M.Y. Vardi. From linear time to branching time. *ACM Trans. on Computational Logic*, 6(2):273–294, April 2005.
57. R.P. Kurshan. *Computer Aided Verification of Coordinating Processes*. Princeton Univ. Press, 1994.
58. R.P. Kurshan. Formal verification in a commercial setting. In *Proc. Conf. on Design Automation (DAC'97)*, volume 34, pages 258–262, 1997.
59. T. Latvala, A. Biere, K. Heljanko, and T. A. Junttila. Simple is better: Efficient bounded model checking for past ltl. In *Proc. 6th Int'l Conf. on Verification, Model Checking, and Abstract Interpretation*, volume 3385 of *Lecture Notes in Computer Science*, pages 380–395. Springer, 2005.
60. K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham. A portfolio approach to algorithm selection. In *Proc. 18th Int'l Joint Conf. on Artificial Intelligence*, pages 1542–1543, 2003.
61. 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, New Orleans, January 1985.

62. C. Löding. Optimal bounds for the transformation of omega-automata. In *Proc. 19th Conference on the Foundations of Software Technology and Theoretical Computer Science*, volume 1738 of *Lecture Notes in Computer Science*, pages 97–109, December 1999.
63. K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
64. M. Michel. Complementation is more difficult with automata on infinite words. CNET, Paris, 1988.
65. A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symp. on Foundation of Computer Science*, pages 46–57, 1977.
66. A. Pnueli and A. Zaks. Psl model checking and run-time verification via testers. In *Proc. 14th Int'l Symp. on Formal Methods*, volume 4085 of *Lecture Notes in Computer Science*, pages 573–586. Springer, 2006.
67. J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 5th International Symp. on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer-Verlag, 1982.
68. M.O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:115–125, 1959.
69. K. Ravi, R. Bloem, and F. Somenzi. A comparative study of symbolic algorithms for the computation of fair cycles. In *Proc. Intl. Conference on Formal Methods in Computer-Aided Design*, number 1954 in Lecture Notes in Computer Science, pages 143–160. Springer-Verlag, 2000.
70. S. Safra. On the complexity of ω-automata. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 319–327, White Plains, October 1988.
71. S. Schwoon and J. Esparza. A note on on-the-fly verification algorithms. In *Proc. 11th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science 3440, pages 174–190. Springer-Verlag, 2005.
72. R. Sebastiani, E. Singerman, S. Tonetta, and M.Y. Vardi. GSTE is partitioned model checking. In *Proc. 16th Int'l Conf. on Computer Aided Verification*, Lecture Notes in Computer Science 3114, pages 229–241, 2004.
73. R. Sebastiani and S. Tonetta. "more deterministic" vs. "smaller" büchi automata for efficient ltl model checking. In *12th Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, volume 2860 of *Lecture Notes in Computer Science*, pages 126–140. Springer-Verlag, 2003.
74. R. Sebastiani, S. Tonetta, and M.Y. Vardi. Symbolic systems, explicit properties: on hybrid approaches for LTL symbolic model checking. In *Proc. 17th Int'l Conf. on Computer Aided Verification*, Lecture Notes in Computer Science 3576, pages 350–373. Springer-Verlag, 2005.
75. A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.
76. F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *Computer Aided Verification, Proc. 12th International Conference*, volume 1855 of *Lecture Notes in Computer Science*, pages 248–263. Springer-Verlag, 2000.
77. F. Somenzi, K. Ravi, and R. Bloem. Analysis of symbolic scc hull algorithms. In *Proc. 4th Int'l Conf. on Formal Methods in Computer-Aided Design*, volume 2517 of *Lecture Notes in Computer Science*, pages 88–105. Springer, 2002.
78. Heikki Tauriainen. Nested emptiness search for generalized Büchi automata. *Fundamenta Informaticae*, 70(1–2):127–154, 2006.
79. X. Thirioux. Simple and efficient translation from LTL formulas to Büchi automata. *Electr. Notes Theor. Comput. Sci.*, 66(2), 2002.
80. M.Y. Vardi. Nontraditional applications of automata theory. In *Proc. International Symp. on Theoretical Aspects of Computer Software*, volume 789 of *Lecture Notes in Computer Science*, pages 575–597. Springer-Verlag, 1994.

81. 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.
82. M.Y. Vardi. Branching vs. linear time: Final showdown. In *Proc. 7th International Conference on Tools and algorithms for the construction and analysis of systems*, volume 2031 of *Lecture Notes in Computer Science*, pages 1–22. Springer-Verlag, 2001.
83. M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st Symp. on Logic in Computer Science*, pages 332–344, Cambridge, June 1986.
84. M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.
85. P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1–2):72–99, 1983.
86. P. Wolper. The tableau method for temporal logic: An overview. *Logique et Analyse*, 110–111:119–136, 1985.
87. 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, Tucson, 1983.
88. Q. Yan. Lower bounds for complementation of ω-automata via the full automata technique. In *Proc. 33rd Intl. Colloq. on Automata, Languages and Pr ogramming*, volume 4052 of *Lecture Notes in Computer Science*, pages 589–600. Springer-Verlag, 2006.
89. L. Zhang and S. Malik. The quest for efficient Boolean satisfiability solvers. In A. Voronkov, editor, *Proc. 18th Int'l Conf. on Automated Deduction (CADE'02)*, Lecture Notes in Computer Science 2392, pages 295–313. Springer-Verlag, 2002.