Learning to Complement Büchi Automata

Yong Li[1,2], Andrea Turrini[1], Lijun Zhang[1,2], and Sven Schewe[3]

[1] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China
[3] University of Liverpool, Liverpool, UK

Abstract. Complementing Büchi automata is an intriguing and intensively studied problem. Complementation suffers from a theoretical super-exponential complexity. From an applied point of view, however, there is no reason to assume that the target language is more complex than the source language. The chance that the smallest representation of a complement language is (much) smaller or (much) larger than the representation of its source should be the same; after all, complementing twice is an empty operation. With this insight, we study the use of learning for complementation. We use a recent learning approach for FDFAs, families of DFAs, that can be used to represent ω-regular languages, as a basis for our complementation technique. As a surprising result, it has proven beneficial not to learn an FDFA that represents the complement language of a Büchi automaton (or the language itself, as complementing FDFAs is cheap), but to use it as an intermediate construction in the learning cycle. While the FDFA is refined in every step, the target is an associated Büchi automaton that underestimates the language of a conjecture FDFA. We have implemented our approach and compared it on benchmarks against the algorithms provided in GOAL. The complement automata we produce for large Büchi automata are generally smaller, which makes them more valuable for applications like model checking. Our approach has also been faster in 98% of the cases. Finally we compare the advantages we gain by the novel techniques with advantages provided by the high level optimisations implemented in the state-of-the-art tool SPOT.

1 Introduction

The complementation of Büchi automata [15] is a classic problem that has been extensively studied for more than half a century; see [56] for a survey. The classic line of research on complementation has started with a proof on the existence of complementation algorithms [38,40] and continued to home in on the complexity of Büchi complementation, finally leading to matching upper [47] and lower [57] bounds ($\approx (0.76n)^n$) for complementing Büchi automata. This line of research has been extended to more general classes of automata, notably parity [49] and generalised Büchi [48] automata.

The complementation of Büchi automata is a valuable tool in formal verification (cf. [34]), in particular when a property that all runs of a model shall have is provided as a Büchi automaton (one tests if the automaton that recognises the runs of a system has an empty intersection with the automaton that recognises the complement of the property language) and when studying language inclusion problems of ω-regular languages [3, 4, 52].

With the growing understanding of the worst case complexity, the practical cost of complementing Büchi automata has become a second line of research. In particular the GOAL tool suite [54] provides a platform for comparing the behaviour of different complementation techniques on various benchmarks [53].

Traditional complementation techniques use the automaton they seek to complement as a starting point for complex state space transformations. These transformations may lead to a super-exponential growth in the size. While this is generally unavoidable [57], we believe that there is no inherent reason to assume that the complement language is harder than the initial language; after all, complementing twice does not change the language[1]. This begs to ask the question, if we can—and if we should—avoid or reduce the dependency on the syntactic representation of the language we want to complement by a Büchi automaton.

This puts the focus on learning based approaches. The classic DFA learning algorithm L* has been proposed by Angluin in [6]. Based on L*, improvements and extensions have been made in [12,32,44]. They have been successfully applied in formal verification, for instance in compositional reasoning [17,21,23], system synthesis [2, 5, 13], and error localisation [20, 22]. Recently, Angluin's learning algorithm has been extended to ω-regular languages [8, 25, 35].

Families of DFAs [8, 35] (FDFAs), introduced in [7], have emerged as an excellent tool to represent ω-regular languages based on the representation of ultimately periodic words uv^ω as pairs (u, v). Based on the experience that DFAs tend not to be much larger than NFAs in practice, there is reasonable hope that FDFAs relate similarly to Büchi automata. Indeed, we have observed that, when we complement Büchi automata using existing determinisation techniques, it is often the case that their corresponding complement Büchi automata are much larger than themselves, while their complements by learning corresponding FDFAs have similar size to them, see Table 2 in Section 4. Moreover, FDFAs have proven to be well suited for learning [8, 35], which makes them an ideal starting point for developing a learning based automata complementation approach.

In a surprising twist, we found that FDFAs do not have to be learned to exploit them in a learning approach. Instead, we use candidate FDFAs $\mathcal{F}$ that are produced during the learning to infer Büchi automata $\underline{B}(\mathcal{F})$ that accept a subset of the ultimately periodic words represented by $\mathcal{F}$. Thus, while our learning algorithm is driven by a core that tries to learn a corresponding FDFA $\mathcal{F}$, it often terminates well before such an FDFA is found. This is possible, because the correspoinding FDFA is only a tool in the complementation algorithm. Broadly speaking, the algorithm uses a candidate $\mathcal{F}$, its complement $\mathcal{F}^{\mathsf{c}}$, and un-

[1] The typical model checking approach to complement the specification first also assumes that the translation into a Büchi automaton is equally efficient for the formula and its negation.

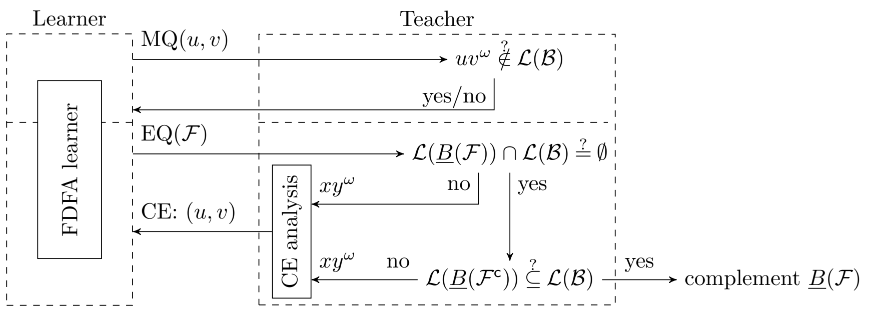

Learner
Teacher
MQ(u, v)
$uv^\omega \overset{?}{\notin} \mathcal{L}(\mathcal{B})$
yes/no
FDFA learner
EQ$(\mathcal{F})$
$\mathcal{L}(\underline{B}(\mathcal{F})) \cap \mathcal{L}(\mathcal{B}) \overset{?}{=} \emptyset$
CE analysis
xy^ω
no
yes
CE: (u, v)
xy^ω
no
$\mathcal{L}(\underline{B}(\mathcal{F}^c)) \overset{?}{\subseteq} \mathcal{L}(\mathcal{B})$
yes
complement $\underline{B}(\mathcal{F})$

Fig. 1. The learning framework for complementing a Büchi automaton $\mathcal{B}$. The learner makes membership queries MQ(u, v), followed by the teacher revealing whether uv^ω is in the complement of $\mathcal{L}(\mathcal{B})$, and equivalence queries EQ$(\mathcal{F})$, upon which the teacher either replies that $\underline{B}(\mathcal{F})$ complements $\mathcal{B}$, or produces a counterexample CE: (u, v), such that the learner can refine $\mathcal{F}$ by either removing (u, v) from the language of $\mathcal{F}$ (if $uv^\omega \in \mathcal{L}(\mathcal{B})$), or by adding it to the language of $\mathcal{F}$ (otherwise).

derapproximations $\underline{B}(\mathcal{F})$ and $\underline{B}(\mathcal{F}^c)$ of their respective ω-languages as its main components, and it can stop as soon as $\underline{B}(\mathcal{F}^c)$ complements the given NBA.

This is feasible because complementing an FDFA $\mathcal{F}$ into an FDFA $\mathcal{F}^c$ is trivial (see Definition 5), and, while a Büchi automaton $\underline{B}(\mathcal{F})$ accepts only a subset of the ultimately periodic words defined by the FDFA $\mathcal{F}$ it under-approximates, we observe that the union of $\underline{B}(\mathcal{F})$ and $\underline{B}(\mathcal{F}^c)$ accepts all infinite words, which is justified by Proposition 2 in Section 3.3.

On first glance, this may sound as if this means that $\underline{B}(\mathcal{F})$ precisely captures the language represented by $\mathcal{F}$, but this is not always the case: an ultimately periodic word uv^ω has many representations as pairs, including e.g. (u, v), (uv, v), and (u, v^7), and it can happen that some are accepted by an FDFA $\mathcal{F}$, while others are accepted by its complement $\mathcal{F}^c$. In this case, we show that uv^ω will be accepted by $\underline{B}(\mathcal{F})$ *or* $\underline{B}(\mathcal{F}^c)$—and possibly by both of them (Proposition 2 in Section 3.3).

We use a variation of Angluin's classic DFA learning algorithm [6] to learn $\mathcal{F}$. Our learning approach, outlined in Figure 1, uses membership queries for $\mathcal{F}$ until a consistent automaton is created. It then turns to equivalence queries. For the membership queries, we use—cheap—standard queries [8, 35]. The novelty lies in a careful design of equivalence queries that make use of cheap operations whenever possible.

These equivalence queries are *not* executed with the FDFA $\mathcal{F}$ and its complement $\mathcal{F}^c$, but with the Büchi automata $\underline{B}(\mathcal{F})$ and $\underline{B}(\mathcal{F}^c)$ that underestimate them. We first check if $\underline{B}(\mathcal{F})$ has an empty language intersection with the automaton $\mathcal{B}$ we want to complement. This step is cheap, and if the answer is negative, then we get an ultimately periodic word uv^ω in the language of $\mathcal{B}$, where at least some representations of uv^ω are accepted by $\mathcal{F}$. We remove the representative provided by the teacher from the language of $\mathcal{F}$ and continue.

We then check if the language of $\underline{B}(\mathcal{F}^{\mathsf{c}})$ is included in the language of $\mathcal{B}$. This is an interesting twist, as language inclusion is one of the traditional justifications for complementing Büchi automata. But while the problem is PSPACE complete, it can usually be handled well by using efficient tools like RABIT [3, 4, 52]. Non-inclusion comes with a witness in the form of an ultimately periodic word uv^ω accepted by $\underline{B}(\mathcal{F}^{\mathsf{c}})$, but not by $\mathcal{B}$. Thus, some representations of uv^ω are (incorrectly) rejected by $\mathcal{F}$. We add them to the language of $\mathcal{F}$ and continue. Otherwise we have $\mathcal{L}(\underline{B}(\mathcal{F}^{\mathsf{c}})) \subseteq \mathcal{L}(\mathcal{B})$. We then conclude that $\mathcal{L}(\underline{B}(\mathcal{F})) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{B})$ and terminate the algorithm with $\underline{B}(\mathcal{F})$ as the complement of $\mathcal{B}$, which is justified by Theorem 2.

In a final bid for optimisation, we observe that this learning procedure can only terminate if $\underline{B}(\mathcal{F})$ and $\underline{B}(\mathcal{F}^{\mathsf{c}})$ are disjoint, which is justified by Corollary 2 in Section 3.4. If they are not, each ultimately periodic word uv^ω in their intersection will, in the final check, be a witness for language non-inclusion. It is, however, much cheaper to find. We therefore suggest that we check disjointness first and proceed to the more expensive language inclusion test only when the disjointness test fails.

Remark. We have also experimented with checking universality of $\mathcal{L}(\mathcal{B}) \cup \mathcal{L}(\underline{B}(\mathcal{F}))$ instead of checking language inclusion of $\underline{B}(\mathcal{F}^{\mathsf{c}})$ in $\mathcal{B}$ in the framework since this is a simple and more intuitive algorithm for complementing Büchi automata based on our learning framework. It has proven to be slower than the algorithm depicted in Figure 1 which confirms that our handling with the equivalence queries is more practical.

Contribution. The complementation of Büchi automata is a heavily researched field. However, to the best of our knowledge, all methods applied to it so far have been automata based. While this focus is natural, it is an important conceptional contribution to consider methods that do *not* focus on manipulating the automata we seek to complement.

Technically, publications about L*-style algorithms can be divided into two main classes: extensions of the L* family to new classes of automata [1,8,12,25,35, 36] and the works that provide suitable—and usually well-performing—teachers for a class of learning problems (e.g. [2,5,12,13,17–23,26,27,31,32,35,41,43,44]). This paper belongs to the latter class of contributions: we propose a simple and practical learning algorithm for complementing Büchi automata.

The performance of learning algorithms depends heavily on the implementation of the teacher. In line with other applications of L*-style algorithms, our contribution is the careful design of an FDFA teacher. In our context, membership queries are straight forward, and the challenge is exclusively in the equivalence queries. The PSPACE equivalence queries the teacher has to answer look like a show stopper. Adding the theoretical super-exponential blow-up incurred by complementing a Büchi automaton to the mix, it is like having the cards stacked against you.

Looking more closely at the challenges posed by equivalence queries, however, reveals that the high costs of equivalence checking can often be avoided. First and foremost, we can check if the candidate language intersects with the language of the automaton we want to complement by a cheap emptiness query ($\mathcal{L}(\underline{B}(\mathcal{F})) \cap \mathcal{L}(\mathcal{B}) \overset{?}{=} \emptyset$). When the emptiness holds, we check if $\mathcal{L}(\underline{B}(\mathcal{F}))$ and $\mathcal{L}(\underline{B}(\mathcal{F}^{\mathsf{c}}))$ intersect, using a second cheap emptiness query ($\mathcal{L}(\underline{B}(\mathcal{F})) \cap \mathcal{L}(\underline{B}(\mathcal{F}^{\mathsf{c}})) \overset{?}{=} \emptyset$). The teacher only uses PSPACE-hard language inclusion queries ($\mathcal{L}(\underline{B}(\mathcal{F}^{\mathsf{c}})) \overset{?}{\subseteq} \mathcal{L}(\mathcal{B})$) once both previous queries are passed. These are usually few queries, and we found that, in spite of the theoretical complexity, existing tools can check inclusion sufficiently fast for this approach to be efficient.

We have implemented the learning-based approach in the tool Buechic based on ROLL [35]. Although we do not improve the theoretical complexity of complementing Büchi automata, our careful design of the FDFA teacher makes learning complement Büchi automata work reasonably well in practice. This is confirmed by the experiments we have performed on the roughly 500 Büchi automata from Büchi Store [55], the generated Büchi automata by SPOT for formulas in [50] and NCSB-Complementation [11].

In the performance evaluation, we were particularly interested in a comparison with GOAL [54]—considering the time to generate the complement automata and their size—as GOAL provides a comprehensive collection of the state-of-the-art techniques as well as a collection of benchmarks. It is therefore well suited for serving as a point of comparison with our novel technique.

In order to give a complete picture of the Büchi automata complementation state of the art, we have also compared Buechic against SPOT [24]. Differently from GOAL, SPOT only implements the most successful technique, and is a highly engineered state-of-the-art tool that has used the insight from GOAL and other automata manipulation techniques to obtain powerful heuristics for state space reduction on top of the principle techniques. While we consider the comparison with GOAL to be fair, comparing with SPOT is over-stretching what our tool can achieve—a bit like comparing a prototype for a new model checking approach with NuSMV. Moreover, SPOT takes advantage of a symbolic representation of the automata, by means of *Ordered Binary Decision Diagrams* (OBDDs) [14], while both Buechic and GOAL use an explicit graph data structure to represent the automata. This means that SPOT can work on multiple states and transitions simultaneously while Buechic and GOAL can only work on a single state/transition at a time. This is another reason why we consider the comparison of Buechic with GOAL to be fairer than with SPOT. Since SPOT does not provide a complementation function for general automata, but only for deterministic ones, we have derived one based on the implemented techniques (determinisation, complementation of deterministic automata, transformation to Büchi) to compare the advancement obtained by our technique with the advancement obtained by using symbolic encoding, states reduction, powerful heuristics, and performance optimisation.

The complement automata we produce are generally smaller for large Büchi automata than those generated by GOAL and SPOT, which makes them more valuable for applications like model checking. Moreover, Buechic has also been faster in 98% of the cases when compared to GOAL, though SPOT is often considerably faster due to its maturity and use of symbolic data structures.

Related Work. Current algorithms [15, 28–30, 33, 38–40, 42, 45–47, 51, 53, 54, 56, 57] for the complementation of Büchi automata are based on a *direct* complementation approach, which is quite different from learning. For a given Büchi automaton $\mathcal{B}$, these approaches use the *structure* of $\mathcal{B}$ as a base to construct a new Büchi automaton that recognises the complement language $\Sigma^\omega \setminus \mathcal{L}(\mathcal{B})$.

We use the learning algorithm instead to directly obtain an automaton that recognises $\Sigma^\omega \setminus \mathcal{L}(\mathcal{B})$. It relies mainly on the *language* of $\mathcal{B}$ instead of on its structure. This allows for obtaining a small automaton for $\Sigma^\omega \setminus \mathcal{L}(\mathcal{B})$, even one that is much smaller than $\mathcal{B}$.

Regarding the use of learning algorithms, there is a vast literature about regular languages (see, e.g., [2, 5, 6, 12, 13, 17, 20–23, 32, 44]); learning ω-regular languages [8, 25, 35] is a young and emerging field. In [25], they learn a Büchi automaton for an ω-regular language $\mathcal{L}$ by learning a DFA defined in [16]. The work proposed in [8] sets the general framework for learning ω-regular languages by means of FDFAs while [35] proposes a practical implementable framework by providing the appropriate FDFA teacher: it assumes that there exists an oracle for the target ω-regular language $\mathcal{L}$ and constructs an automaton accepting $\mathcal{L}$. In this paper we design the oracle for the FDFA teacher used in [35]; the oracle knows the complement of the language of $\mathcal{B}$ and is able to produce the appropriate counterexamples that are then analysed and returned to the learner.

Organisation of the Paper. After starting with some background and notation in Section 2, we describe our learning based complementation technique in Section 3. In Section 4, we evaluate our technique on standard complementation benchmarks and against the competitor algorithms from the GOAL suite and SPOT, before concluding the paper with Section 5.

2 Preliminaries

Let X and Y be two sets; we denote by $X \ominus Y$ their *symmetric difference*, i.e. the set $(X \setminus Y) \cup (Y \setminus X)$.

Let Σ be a finite set of *letters* called *alphabet*. A finite sequence of letters is called a (finite) *word*. An infinite sequence of letters is called an *ω-word*. We use $|\alpha|$ to denote the length of the finite word α and we denote by $\mathrm{last}(\alpha)$ the last letter of α. We use ε to represent an empty word. The set of all finite words on Σ is denoted by Σ^*, and the set of all ω-words is denoted by Σ^ω. Moreover, we also denote by Σ^+ the set $\Sigma^* \setminus \{\varepsilon\}$. Given a finite word $\alpha = a_0 a_1 \dots$ and $i, k < |\alpha|$, we denote by $\alpha(i)$ the letter a_i and we use $\alpha[i : k]$ to denote the subword $\alpha' = a_i \dots a_k$ of α, when $i \leq k$, and the empty word ε when $i > k$.

Definition 1. *A* nondeterministic Büchi automaton *(NBA) is a tuple* $\mathcal{B} = (\Sigma, Q, I, \mathrm{T}, \mathrm{F})$*, consisting of a finite alphabet* Σ *of input letters, a finite set* Q *of states with a non-empty subset* $I \subseteq Q$ *of initial states, a set* $\mathrm{T} \subseteq Q \times \Sigma \times Q$ *of transitions, and a set* $\mathrm{F} \subseteq Q$ *of accepting states.*

We denote the generic elements of an NBA $\mathcal{B}$ by Σ, Q, I, T, F, and we propagate primes and indices when necessary. Thus, for example, the NBA $\mathcal{B}'_i$ has states Q'_i, initial states I'_i, input letters Σ'_i, transition set T'_i, and accepting states F'_i; we use a similar notation for the other automata we introduce later.

An *run* of an NBA $\mathcal{B}$ over an ω-word $\alpha = a_0 a_1 a_2 \cdots \in \Sigma^\omega$ is an infinite sequence of states $\rho = q_0 q_1 q_2 \cdots \in Q^\omega$ such that $q_0 \in I$ and, for each $i \geq 0$, $\big(\rho(i), a_i, \rho(i{+}1)\big) \in \mathrm{T}$ where $\rho(i) = q_i$. A run ρ is *accepting* if it contains infinitely many accepting states, i.e. $\mathrm{Inf}(\rho) \cap \mathrm{F} \neq \emptyset$, where $\mathrm{Inf}(\rho) = \{\, q \in Q \mid \forall i \in \mathbb{N}. \exists j > i : \rho(j) = q \,\}$. A ω-word α is *accepted* by $\mathcal{B}$ if $\mathcal{B}$ has an accepting run on α, and the set of words $\mathcal{L}(\mathcal{B}) = \{\, \alpha \in \Sigma^\omega \mid \alpha \text{ is accepted by } \mathcal{B} \,\}$ accepted by $\mathcal{B}$ is called its *language*.

We call a subset of Σ^ω is an *ω-language* and the language of an NBA an *ω-regular language*. Words of the form uv^ω are called *ultimately periodic* words. We use a pair of finite words (u, v) to denote the ultimately periodic word $w = uv^\omega$. We also call (u, v) a *decomposition* of w. For an ω-language L, let $\mathrm{UP}(L) = \{\, uv^\omega \in L \mid u \in \Sigma^*, v \in \Sigma^+ \,\}$ be the set of all ultimately periodic words in L.

Theorem 1 (Ultimately Periodic Words [15]). *Let L, L' be two ω-regular languages. Then $L = L'$ if, and only if, $UP(L) = UP(L')$.*

An immediate consequence of the above theorem is that, for any two ω-regular languages L_1 and L_2, if $L_1 \neq L_2$ then there is an ultimately periodic word $xy^\omega \in \mathrm{UP}(L_1) \ominus \mathrm{UP}(L_2)$.

Definition 2. *A* deterministic finite automaton *(DFA) is a tuple $A = (\Sigma, Q, \bar{q}, \mathrm{T}, F)$, consisting of a finite alphabet Σ of input letters, a finite set Q of states with an initial state $\bar{q} \in Q$, a total transition function* $\mathrm{T} \colon Q \times \Sigma \to Q$, *and a set $F \subseteq Q$ of accepting (final) states.*

The complement A^{c} *of a DFA $A = (\Sigma, Q, \bar{q}, \mathrm{T}, F)$ is the DFA $A^{\mathsf{c}} = (\Sigma, Q, \bar{q}, \mathrm{T}, Q \setminus F)$.*

Given a DFA A and two states s and f, let $A^s_f = (\Sigma, Q, s, \mathrm{T}, \{f\})$ be the DFA obtained from A by setting its initial and accepting states to s and $\{f\}$, respectively.

A run of a DFA A over a word $\alpha = a_0 \cdots a_k \in \Sigma^*$ is a finite sequence of states $\rho = q_0 \cdots q_{k+1} \in Q^*$ such that $q_0 = \bar{q}$ and for every $0 \leq i \leq k$, $q_{i+1} = \mathrm{T}(q_i, a_i)$ where $k \geq 0$. The run ρ of A on α is accepting if $q_{k+1} \in F$. We denote by $\mathcal{L}(A)$ the language of A, i.e., the set of all words whose corresponding runs are accepted by A. We call the language of a DFA a *regular language*. Given an input word $\alpha \in \Sigma^*$ and the run ρ of A on α, we denote by $A(\alpha)$ the last reached state $\mathrm{last}(\rho)$. Given a DFA A with alphabet Σ, it holds that $\mathcal{L}(A^{\mathsf{c}}) = \Sigma^* \setminus \mathcal{L}(A)$.

Note that we require T to be total so to simplify the definitions in the remainder of the paper. Each DFA A with a non-total transition function can be transformed to a DFA A' as by Definition 2 such that $\mathcal{L}(A') = \mathcal{L}(A)$ by adding a fresh non-final sink state, and by letting T' agree with T where T is defined and mapping to this fresh sink state otherwise.

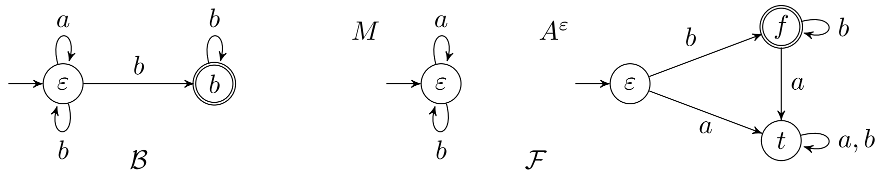


Fig. 2. An NBA $\mathcal{B}$ and an FDFA $\mathcal{F} = (M, \{A^\varepsilon\})$ recognising the same language $L = \Sigma^* \cdot b^\omega$.

Learning regular languages via DFAs was first proposed in [6], and the *right congruence* is the theoretical foundation for it to discover states in a regular language. A right congruence is an equivalence relation $\sim$ on Σ^* such that $x \sim y$ implies $xv \sim yv$ for every $x, y, v \in \Sigma^*$. We denote by $|\sim|$ the index of $\sim$, i.e. the number of equivalence classes of $\sim$. We use $\Sigma^*/_\sim$ to denote the equivalence classes of the right congruence $\sim$. A *finite right congruence* is a right congruence with a finite index. For a word $u \in \Sigma^*$, we denote by $[u]_\sim$ the class of $\sim$ in which u resides.

The main obstacle to learn ω-regular languages via Büchi automata is that there is a lack of right congruence for Büchi automata. Inspired by the work of Arnold [9], Maler and Stager [37] proposed the notion of *family of right-congruences*. Based on this, Angluin and Fisman [8] further proposed to learn ω-regular languages via a formalism called *family of DFAs*, in which every DFA corresponds to a right congruence.

Definition 3 (Family of DFAs [8]). *A* family of DFAs *(FDFA) over an alphabet Σ is a pair $\mathcal{F} = (M, \{A^q\})$ consisting of a leading DFA $M = (\Sigma, Q, \bar{q}, \mathrm{T}, \emptyset)$ and of a progress DFA $A^q = (\Sigma, Q^q, \bar{q}^q, \mathrm{T}^q, F^q)$ for each $q \in Q$.*

In the remainder of the paper we may just write $M = (\Sigma, Q, \bar{q}, \mathrm{T})$ for a leading DFA. We say that a decomposition (u, v) is accepted by an FDFA $\mathcal{F}$ if $M(u) = M(uv)$ and $A^q(v) \in F^q$ where $q = M(u)$. An ultimately periodic word $\alpha \in \Sigma^\omega$ is accepted by an FDFA $\mathcal{F}$ if there exists a decomposition (u, v) of α that is accepted by $\mathcal{F}$. Then we define $\mathrm{UP}(\mathcal{F}) = \{\, \alpha \in \Sigma^\omega \mid \alpha \text{ is accepted by } \mathcal{F} \,\}$. As an example of FDFAs, consider the FDFA $\mathcal{F}$ shown in Figure 2: the leading DFA M has only one state, ε, and the corresponding progress DFA for state ε is A^ε. The word ab^ω is accepted by $\mathcal{F}$ since there exists the decomposition (a, b) of ab^ω being accepted by $\mathcal{F}$. It is easy to see that $\mathrm{UP}(\mathcal{F}) = \Sigma^* \cdot b^\omega$, which is also recognised by the NBA $\mathcal{B}$ depicted in Figure 2.

In [8], Angluin and Fisman propose to use three canonical FDFAs to recognise ω-regular languages, namely periodic FDFAs, syntactic FDFAs, and recurrent FDFAs. In this paper, we only use syntactic FDFAs since they can be exponentially smaller than their periodic counterpart [8] and have proved to be well suited for converting to Büchi automata [35]. The right congruence $\sim_L$ of a given ω-regular language L is defined such that $x \sim_L y$ if for each $w \in \Sigma^\omega$, it holds that $xw \in L$ if and only if $yw \in L$.

Definition 4 (Syntactic FDFA [8]). *Given an ω-regular language L, the syntactic FDFA $\mathcal{F} = (M, \{A^u\})$ for L is defined as follows. The leading DFA M is the tuple $M = (\Sigma, \Sigma^*/_{\sim_L}, [\varepsilon]_{\sim_L}, \mathrm{T}, \emptyset)$, where $\mathrm{T}([u]_{\sim_L}, a) = [ua]_{\sim_L}$ for all $u \in \Sigma^*$ and $a \in \Sigma$.*

The right congruence $\approx^u_S$ for a progress DFA A^u of the syntactic FDFA is defined as follows.

$$x \approx^u_S y \text{ iff } ux \sim_L uy \wedge \forall v \in \Sigma^*. uxv \sim_L u \implies (u(xv)^\omega \in L \iff u(yv)^\omega \in L).$$

The progress DFA A^u is the tuple $(\Sigma, \Sigma^*/_{\approx^u_S}, [\varepsilon]_{\approx^u_S}, \mathrm{T}_S, F_S)$, where, for each $v \in \Sigma^*$ and $a \in \Sigma$, $\mathrm{T}_S([v]_{\approx^u_S}, a) = [va]_{\approx^u_S}$. The set of accepting states F_S is the set of equivalence classes $[v]_{\approx^u_S}$, for which $uv \sim_L u$ and $uv^\omega \in L$ hold.

Given an ω-regular language L, the corresponding syntactic FDFA for L has finite states [8].

Lemma 1 (cf. [8]). *Let $\mathcal{F} = (M, \{A^q\})$ be a syntactic FDFA recognising the ω-regular language L. Then we have $UP(\mathcal{F}) = UP(L)$ and if $xy^\omega \in L$, then every decomposition (u, v) of xy^ω with $M(u) = M(uv)$ is accepted by $\mathcal{F}$.*

An example of syntactic FDFAs is the FDFA $\mathcal{F}$ shown in Figure 2. This FDFA $\mathcal{F}$ recognises the ω-regular language $\Sigma^* \cdot b^\omega$. Since ab^ω is accepted by $\mathcal{F}$, we have that every decomposition of ab^ω is accepted by $\mathcal{F}$.

Definition 5 (Complement of FDFA [7]). *Given an FDFA $\mathcal{F} = (M, \{A^q\})$, the* complement $\mathcal{F}^\mathsf{c}$ *of $\mathcal{F}$ is the FDFA $\mathcal{F}^\mathsf{c} = (M, \{A^{q\mathsf{c}}\})$.*

In contrast to [7], we consider general FDFAs instead of only canonical FDFAs in this paper. As a consequence, though we call $\mathcal{F}^\mathsf{c}$ the complement of $\mathcal{F}$, actually it is possible to have $\mathrm{UP}(\mathcal{F}) \cap \mathrm{UP}(\mathcal{F}^\mathsf{c}) \neq \emptyset$. This complicates the use of $\mathcal{F}^\mathsf{c}$. More details will be given in Section 3.

Transforming FDFAs to Büchi Automata. According to [35, Section 6], an FDFA $\mathcal{F}$ does not necessarily recognise an ω-regular language. Thus one cannot construct an NBA $\mathcal{B}$ for an arbitrary FDFA $\mathcal{F}$ such that $\mathrm{UP}(\mathcal{F}) = \mathrm{UP}(\mathcal{L}(\mathcal{B}))$. To overcome this obstacle, the authors of [35] propose two methods to approximate $\mathrm{UP}(\mathcal{F})$ by means of two Büchi automata $\underline{B}(\mathcal{F})$ and $\overline{B}(\mathcal{F})$ that accept an under- and an over-approximation, respectively, of $\mathrm{UP}(\mathcal{F})$. We use the under-approximation method, because this ensures that $\mathrm{UP}(\mathcal{L}(\underline{B}(\mathcal{F}))) = \mathrm{UP}(\mathcal{F})$ holds whenever $\mathcal{F}$ is a canonical FDFA (cf. [35, Lemma 3]). No such property has been established for the over-approximation method.

We now present the idea underlying the construction of the under-approximation $\underline{B}(\mathcal{F})$ proposed in [35], to which we refer for details. Recall that A^s_f denotes the DFA A where s is the initial state and f the only accepting state; recall that an FDFA $\mathcal{F} = (M, \{A^q\})$ consists of a leading DFA $M = (\Sigma, Q, \bar{q}, \mathrm{T}, \emptyset)$ and of a progress DFA $A^q = (\Sigma, Q^q, \bar{q}^q, \mathrm{T}^q, F^q)$ for each $q \in Q$; recall also that $\mathrm{UP}(\mathcal{F}) = \{\, \alpha \in \Sigma^\omega \mid \alpha \text{ is accepted by } \mathcal{F} \,\}$, where α is accepted if there exists a decomposition (u, v) of α, such that $uv^\omega = \alpha$, $M(u) = M(uv)$, and $A^q(v) \in F^q$, where $q = M(u)$. This implies that every word α in $\mathrm{UP}(\mathcal{F})$ can be decomposed into two parts u and v, such that u is consumed by a run of M and v by a run of A^q. Note that, if we consider $M_q^{\bar{q}}$, then we have that uv^ω is accepted by $\mathcal{F}$ if $M_q^{\bar{q}}(u) = M_q^{\bar{q}}(uv)$, $u \in \mathcal{L}(M_q^{\bar{q}})$, and $A^q(v) \in F^q$, where $q = M(u)$. This means that we can write $\mathrm{UP}(\mathcal{F})$ as $\mathrm{UP}(\mathcal{F}) = \bigcup_{q \in Q, f \in F^q} \mathcal{L}(M_q^{\bar{q}}) \cdot N_{(q,f)}$ where $N_{(q,f)} = \{\, v^\omega \in \Sigma^\omega \mid v \in \Sigma^+ \wedge q = M_q^q(v) \wedge v \in \mathcal{L}((A^q)_f^{\bar{q}^q}) \,\}$ is the set of all infinite repetitions of the finite words v accepted by A_f^q.

In order to under-approximate $\mathrm{UP}(\mathcal{F})$, it is enough to match exactly $\mathcal{L}(M_q^{\bar{q}})$ and to under-approximate $N_{(q,f)}$. The former is trivial, since we already have $M_q^{\bar{q}}$; for the latter, consider the DFA $\underline{P}_{(q,f)} = M_q^q \times (A^q)_f^{\bar{q}^q} \times (A^q)_f^f$, where $\times$ stands for the standard intersection product between DFAs: the DFA $M_q^q \times (A^q)_f^{\bar{q}^q}$ ensures that for any $v \in \mathcal{L}(M_q^q \times (A^q)_f^{\bar{q}^q})$ and $u \in \mathcal{L}(M_q^{\bar{q}})$, we have $q = M(u) = M(uv)$ while $(A^q)_f^f$ guarantees that $v, v^2 \in \mathcal{L}((A^q)_f^{\bar{q}^q})$. Then, by the construction in [35, Definition 4], it is possible to construct an NBA $\underline{B}(\mathcal{F})$ such that $\mathcal{L}(\underline{B}(\mathcal{F})) = \bigcup_{q \in Q, f \in F^q} \mathcal{L}(M_q^{\bar{q}}) \cdot \underline{N}_{(q,f)}$ where $\underline{N}_{(q,f)} = \mathcal{L}(\underline{P}_{(q,f)})^\omega$. $\underline{B}(\mathcal{F})$ under-approximates the language of $\mathcal{F}$:

Lemma 2 ([35, Lemma 3]). *For every FDFA $\mathcal{F}$, $UP(\mathcal{L}(\underline{B}(\mathcal{F}))) \subseteq UP(\mathcal{F})$ holds. If $\mathcal{F}$ is canonical, then $UP(\mathcal{L}(\underline{B}(\mathcal{F}))) = UP(\mathcal{F})$ holds.*

3 Learning to Complement Büchi Automata

In this section we present the details of our learning framework, depicted in Figure 1, to learn the complement language $L = \Sigma^\omega \setminus \mathcal{L}(\mathcal{B})$ of a given NBA $\mathcal{B}$. We first outline the general framework. We then continue with the technical part, where we first show that the counterexamples are correct in Section 3.2, and then establish termination and correctness in Section 3.3, before we finally discuss an optimisation in Section 3.4.

3.1 The Learning Framework

We begin with an introduction of the learning framework for L, depicted in Figure 1. The framework consists of two components, namely the learner and the teacher for complementing Büchi automaton.

The learner is a standard FDFA learner (see, e.g. [8,35]). He tries to learn an FDFA that recognises an ω-regular language L by means of two types of queries: membership queries of the form $\mathrm{MQ}(u, v)$ that provide him with information about whether the word uv^ω has to be included in L; and equivalence queries of the form $\mathrm{EQ}(\mathcal{F})$, aimed to find differences between the current conjecture $\mathcal{F}$ and the language he shall learn. The learner is oblivious of the fact that the NBA $\underline{B}(\mathcal{F})$ is sought after, not $\mathcal{F}$ itself.

The teacher provides answers to these queries based on a definition of the complement of L by an NBA $\mathcal{B}$. Answering a membership query $\mathrm{MQ}(u, v)$ is easy: it reduces to checking whether $uv^\omega \in L$, i.e. whether $uv^\omega \notin \mathcal{L}(\mathcal{B})$.

The innovation is in the way the equivalence queries $\mathrm{EQ}(\mathcal{F})$ are answered. For checking equivalence, the teacher works with two NBAs $\underline{B}(\mathcal{F})$ and $\underline{B}(\mathcal{F}^{\mathsf{c}})$ that underestimate the ω-languages recognised by $\mathcal{F}$ and $\mathcal{F}^{\mathsf{c}}$, respectively. She reports equivalence to the learner, when she is satisfied that $\mathcal{L}(\underline{B}(\mathcal{F})) = L$ holds. For algorithmic reasons, this is the case when $\mathcal{L}(\underline{B}(\mathcal{F}^{\mathsf{c}})) = \mathcal{L}(\mathcal{B})$ holds, too.

In her first step in answering an equivalence query $\mathrm{EQ}(\mathcal{F})$, she constructs the NBA $\underline{B}(\mathcal{F})$ from the conjecture $\mathcal{F}$ and then checks whether $\mathcal{L}(\underline{B}(\mathcal{F})) \cap \mathcal{L}(\mathcal{B}) = \emptyset$ holds. If this is not the case, then a witness $xy^{\omega} \in \mathcal{L}(\underline{B}(\mathcal{F})) \cap \mathcal{L}(\mathcal{B})$ is constructed. Since $\mathrm{UP}(\mathcal{L}(\underline{B}(\mathcal{F}))) \subseteq \mathrm{UP}(\mathcal{F})$ is established in Lemma 2, this implies $xy^{\omega} \in \mathrm{UP}(\mathcal{F}) \cap \mathrm{UP}(\mathcal{L}(\mathcal{B}))$.

She then analyses the witness xy^{ω} to get a decomposition (u, v) of xy^{ω} that is accepted by $\mathcal{F}$. She then returns (u, v) to the learner as a counterexample (that matches Definition 6), for him to remove (u, v) from the current FDFA $\mathcal{F}$, since $uv^{\omega} \in \mathcal{L}(\mathcal{B})$.

When the first check $\mathcal{L}(\underline{B}(\mathcal{F})) \cap \mathcal{L}(\mathcal{B}) = \emptyset$ has been passed successfully, the teacher constructs $\underline{B}(\mathcal{F}^{\mathsf{c}})$ and checks whether $\mathcal{L}(\underline{B}(\mathcal{F}^{\mathsf{c}})) \subseteq \mathcal{L}(\mathcal{B})$ holds. This language inclusion test is delegated to the off-the-shelf tool RABIT [3, 4, 52]. Note that RABIT does not complement either of the two input languages. If language inclusion holds, we exploit $\mathcal{L}(\underline{B}(\mathcal{F})) \cup \mathcal{L}(\underline{B}(\mathcal{F}^{\mathsf{c}})) = \Sigma^{\omega}$ (a property we establish in Proposition 2) to infer $\mathcal{L}(\underline{B}(\mathcal{F})) \cup \mathcal{L}(\mathcal{B}) = \Sigma^{\omega}$. Since we know that $\mathcal{L}(\underline{B}(\mathcal{F})) \cap \mathcal{L}(\mathcal{B}) = \emptyset$ holds from the first check, this implies that $\underline{B}(\mathcal{F})$ complements $\mathcal{B}$.

If the second check fails, the teacher gets a witness $xy^{\omega} \in \mathcal{L}(\underline{B}(\mathcal{F}^{\mathsf{c}})) \setminus \mathcal{L}(\mathcal{B})$, such that $\mathrm{UP}(\mathcal{L}(\underline{B}(\mathcal{F}^{\mathsf{c}}))) \subseteq \mathrm{UP}(\mathcal{F}^{\mathsf{c}})$ (Lemma 2) implies $xy^{\omega} \in \mathrm{UP}(\mathcal{F}^{\mathsf{c}}) \setminus \mathrm{UP}(\mathcal{L}(\mathcal{B}))$. She then analyses the witness xy^{ω} to derive a decomposition (u, v) of xy^{ω} that is accepted by $\mathcal{F}^{\mathsf{c}}$. She then returns (u, v) to the learner as a counterexample (that matches Definition 6), for him to add (u, v) to the current FDFA $\mathcal{F}$, since $uv^{\omega} \notin \mathcal{L}(\mathcal{B})$ and (u, v) is not accepted by $\mathcal{F}$.

3.2 Correctness of the Counterexample Analysis

One important task of the teacher in the learning framework depicted in Figure 1 is the construction of the appropriate counterexample (u, v) in case the equivalence query $\mathrm{EQ}(\mathcal{F})$ has to be answered negatively. Note that this is the only step in our learning loop that depends on the representation of the complement language by $\mathcal{B}$—a much looser connection than for the off-the-shelf complementation algorithms implemented in GOAL [54] and SPOT [24]. The counterexample we receive is an ultimately periodic word xy^{ω}. We cannot, however, simply return (x, y) but we have to infer an appropriate counterexample (u, v) such that $uv^{\omega} = xy^{\omega}$. For this, we first recall the notion of counterexamples for FDFA learners.

Definition 6 (Counterexample for the FDFA learner [35]). *Given a conjectured FDFA $\mathcal{F} = (M, \{A^q\})$ and the target language L, we say that a counterexample (u, v) is*

– *positive if $M(u) = M(uv)$, $uv^{\omega} \in UP(L)$, and (u, v) is not accepted by $\mathcal{F}$,*

– *negative if $M(u) = M(uv)$, $uv^\omega \notin UP(L)$, and (u, v) is accepted by $\mathcal{F}$.*

Note that, when a pair (u, v) is accepted by $\mathcal{F}$, then $M(u) = M(uv)$ holds. The FDFA learner underlying the Büchi automaton complementation learner can use the counterexample for the FDFA learner to refine the conjecture $\mathcal{F}$ for the target language L. Intuitively, if a counterexample (u, v) is positive, then $\mathcal{F}$ should accept it, while $\mathcal{F}$ should reject it when it is negative. Our goal is to infer a valid decomposition (u, v) from xy^ω, which matches the cases in Definition 6, to be able to refine $\mathcal{F}$. Proposition 1 guarantees that, if there exists xy^ω violating the checks performed in our learning framework, then we can always construct a decomposition (u, v) from xy^ω—that satisfies $uv^\omega = xy^\omega$—to refine $\mathcal{F}$.

Proposition 1. *Given an NBA $\mathcal{B}$ with alphabet Σ, let $L = \Sigma^\omega \setminus \mathcal{L}(\mathcal{B})$ be its complement language and target ω-regular language. Suppose $\mathcal{F}$ is the current FDFA conjecture. Whenever the teacher returns (u, v) as answer to an equivalence query* $\mathrm{EQ}(\mathcal{F})$*, then (u, v) is either a positive or negative counterexample.*

3.3 Termination and Correctness of the Learning Algorithm

Based on Proposition 1, the learner can refine the current FDFA $\mathcal{F}$ with the returned counterexample (u, v) from the teacher. Since the learner is the same as the FDFA learner proposed in [8, 35], in the worst case, we have to get the canonical FDFA that recognises L in order to complete the learning task. Moreover, the number of membership queries and equivalence queries are polynomial in the size of the canonical periodic FDFA [8, 35].

In order to establish the correctness of our learning algorithm, we first introduce a result that, while being used for proving the correctness of the algorithm, is of interest in its own right: we establish in Proposition 2 that, for a (not necessarily canonical) FDFA $\mathcal{F}$, the NBAs $\underline{B}(\mathcal{F})$ and $\underline{B}(\mathcal{F}^\mathsf{c})$ that underapproximate the languages of $\mathcal{F}$ and its complement $\mathcal{F}^\mathsf{c}$, respectively, cover the whole Σ^ω. This generalises a simpler result for canonical FDFAs from [7].

Proposition 2. *Given an FDFA $\mathcal{F}$ with alphabet Σ, it is $\mathcal{L}(\underline{B}(\mathcal{F})) \cup \mathcal{L}(\underline{B}(\mathcal{F}^\mathsf{c})) = \Sigma^\omega$.*

Proof. First one can show that for each pair of ω-regular languages L_1 and L_2, we have that $\mathrm{UP}(L_1 \cup L_2) = \mathrm{UP}(L_1) \cup \mathrm{UP}(L_2)$. By Theorem 1, it suffices to prove that $\mathrm{UP}(\mathcal{L}(\underline{B}(\mathcal{F}))) \cup \mathrm{UP}(\mathcal{L}(\underline{B}(\mathcal{F}^\mathsf{c}))) = \mathrm{UP}(\Sigma^\omega) = \{\, uv^\omega \in \Sigma^\omega \mid u \in \Sigma^*, v \in \Sigma^+ \,\}$ holds in order to show that $\mathcal{L}(\underline{B}(\mathcal{F})) \cup \mathcal{L}(\underline{B}(\mathcal{F}^\mathsf{c})) = \Sigma^\omega$ holds. That is, we need to show that, for all finite words $u \in \Sigma^*$ and $v \in \Sigma^+$, $uv^\omega \in \mathrm{UP}(\mathcal{L}(\underline{B}(\mathcal{F})))$ or $uv^\omega \in \mathrm{UP}(\mathcal{L}(\underline{B}(\mathcal{F}^\mathsf{c})))$.

Given an FDFA $\mathcal{F} = (M, \{A^q\})$, for any $u \in \Sigma^*$ and $v \in \Sigma^+$, by [8] we can always find a *normalised* decomposition (x, y) of uv^ω such that $q = M(x) = M(xy)$ and $xy^\omega = uv^\omega$ since M is a complete DFA with a finite set of states. Then, one can show that there exists some $j \geq 1$ such that y^j is either accepted by A^q or $A^{q\mathsf{c}}$. Therefore, we can conclude that (x, y^j) is either accepted by $\mathcal{F}$ or $\mathcal{F}^\mathsf{c}$. Consequently, we get that $xy^\omega = x(y^j)^\omega \in \mathrm{UP}(\mathcal{L}(\underline{B}(\mathcal{F})))$ or that $xy^\omega = x(y^j)^\omega \in \mathrm{UP}(\mathcal{L}(\underline{B}(\mathcal{F}^\mathsf{c})))$, as required. □

The following theorem guarantees the main result about the termination and correctness of the proposed framework. That is, the learning algorithm always returns an NBA that accepts the complement language of the given $\mathcal{B}$.

Theorem 2. *Given an NBA $\mathcal{B}$ with alphabet Σ, the learning algorithm depicted in Figure 1 terminates and returns an NBA $\underline{B}(\mathcal{F})$ such that $\mathcal{L}(\underline{B}(\mathcal{F})) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{B})$.*

Note that the algorithm can terminate before we have learned the canonical FDFA that represents $\Sigma^\omega \setminus \mathcal{L}(\mathcal{B})$: on termination $\mathcal{L}(\underline{B}(\mathcal{F})) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{B})$ is guaranteed since the conjecture $\mathcal{F}$ satisfies $\mathcal{L}(\underline{B}(\mathcal{F})) \cap \mathcal{L}(\mathcal{B}) = \emptyset$ and $\mathcal{L}(\underline{B}(\mathcal{F}^{\mathsf{c}})) \subseteq \mathcal{L}(\mathcal{B})$. When a conjectured $\mathcal{F}$ does not satisfy $\mathcal{L}(\underline{B}(\mathcal{F})) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{B})$, then it is easy to conclude, together with $\mathcal{L}(\underline{B}(\mathcal{F})) \cup \mathcal{L}(\underline{B}(\mathcal{F}^{\mathsf{c}})) = \Sigma^\omega$ by Proposition 2, that $\mathcal{L}(\underline{B}(\mathcal{F})) \cap \mathcal{L}(\mathcal{B}) \neq \emptyset$ or $\mathcal{L}(\underline{B}(\mathcal{F}^{\mathsf{c}})) \not\subseteq \mathcal{L}(\mathcal{B})$ holds.

Corollary 1. *The learning algorithm terminates with $\mathcal{L}(\underline{B}(\mathcal{F}^{\mathsf{c}})) = \mathcal{L}(\mathcal{B}) = \Sigma^\omega \setminus \mathcal{L}(\underline{B}(\mathcal{F}))$.*

From Corollary 1, we can get a Büchi automaton $\underline{B}(\mathcal{F}^{\mathsf{c}})$ accepting the same language of $\mathcal{B}$ as a for-free by-product of the complementing algorithm. This means that we have also provided an alternative oracle that can be used to learn the language of $\mathcal{B}$, which can be another method to reduce the size of $\mathcal{B}$. Therefore, our learning based complementation algorithm has proven beneficial not to learn an FDFA that represents the complement language of a Büchi automaton (or the language itself, as complementing FDFAs is cheap), but to use it as an intermediate construction in the learning cycle.

3.4 An Improved Algorithm

Once the learning algorithm terminates we have that $\mathcal{L}(\underline{B}(\mathcal{F})) \cap \mathcal{L}(\mathcal{B}) = \emptyset$ and $\mathcal{L}(\underline{B}(\mathcal{F}^{\mathsf{c}})) \subseteq \mathcal{L}(\mathcal{B})$. It trivially follows that $\mathcal{L}(\underline{B}(\mathcal{F})) \cap \mathcal{L}(\underline{B}(\mathcal{F}^{\mathsf{c}})) = \emptyset$ holds.

Corollary 2. *The learning algorithm terminates with $\mathcal{L}(\underline{B}(\mathcal{F})) \cap \mathcal{L}(\underline{B}(\mathcal{F}^{\mathsf{c}})) = \emptyset$.*

Therefore, $\mathcal{L}(\underline{B}(\mathcal{F})) \cap \mathcal{L}(\underline{B}(\mathcal{F}^{\mathsf{c}})) = \emptyset$ is a necessary condition for the termination of the learning framework. Since the most expensive step is checking language inclusion between $\mathcal{L}(\underline{B}(\mathcal{F}^{\mathsf{c}}))$ and $\mathcal{L}(\mathcal{B})$, we should avoid this check whenever possible. To do so, we can simply check whether $\mathcal{L}(\underline{B}(\mathcal{F})) \cap \mathcal{L}(\underline{B}(\mathcal{F}^{\mathsf{c}})) = \emptyset$ holds right before checking the language inclusion.

If there exists some $xy^\omega \in \mathcal{L}(\underline{B}(\mathcal{F})) \cap \mathcal{L}(\underline{B}(\mathcal{F}^{\mathsf{c}}))$, then we have in particular that some decomposition (u, v) of xy^ω is accepted by $\mathcal{F}^{\mathsf{c}}$, as well as $xy^\omega \in \mathcal{L}(\underline{B}(\mathcal{F}))$. The latter implies with $\mathcal{L}(\underline{B}(\mathcal{F})) \cap \mathcal{L}(\mathcal{B}) = \emptyset$ (recall that this is checked first) that $xy^\omega \in L$ (since $\mathcal{L}(\underline{B}(\mathcal{F})) \subseteq L$ was shown). We can therefore return the decomposition (u, v) as a positive counterexample for the FDFA learner to refine $\mathcal{F}$. Otherwise, we just proceed to check the language inclusion.

This optimisation preserves the correctness of the algorithm, and we apply it by default.

Table 1. Comparison between GOAL, SPOT, and Buechic on complementing Büchi Store. The average number of letters in each alphabet is about 9.

Block	Experiments		GOAL				Buechic	SPOT
	(States, Transitions)		Ramsey	Determinisation	Rank	Slice		
1	287 NBAs (928, 2071)	$\|Q\|$	21610	3919	21769	4537	2428	**1629**
		$\|T\|$	964105	87033	179983	125155	35392	**13623**
		t_c	992	300	203	204	105	**6**
2	5 NBAs (55, 304)	$\|Q\|$	–to–	926	38172	1541	**165**	495
		$\|T\|$		21845	384378	50689	5768	**4263**
		t_c		28	42	12	474	**<1**
3	2 NBAs (20, 80)	$\|Q\|$	–to–	–to–	27372	11734	**96**	2210
		$\|T\|$			622071	1391424	**6260**	102180
		t_c			56	152	7	**1**

4 Experimental Evaluation

We have implemented a prototype, Buechic, of our learning approach based on the ROLL learning library [35]. We use RABIT [3,4,52] to perform the inclusion check $\mathcal{L}(\underline{B}(\mathcal{F}^c)) \subseteq \mathcal{L}(\mathcal{B})$ that occurs in the evaluation of the equivalence query $\mathrm{EQ}(\mathcal{F})$ (cf. Figure 1). The machine we used for the experiments is a 3.6 GHz Intel i7-4790 with 16 GB of RAM, of which 8 GB were assigned to the tool. The timeout has been set to 300 seconds in this section. In the experiments, we compare our Büchi complementation algorithm with two tools. The first tool is GOAL (the latest version 2015-10-18) [54], which is a mature and well-known tool for manipulating Büchi automata. We consider four different complementing algorithms implemented in GOAL, see [54] for more details.

We have used SPOT (the stable version 2.3.5) [24] as a second point of comparison. SPOT is the state-of-the-art platform for manipulating ω-automata, including Büchi automata. Recall that SPOT does not provide a complementation function for generic Büchi automata directly, thus we first use SPOT to get a deterministic automaton from the given Büchi automaton, then complement the resulting deterministic automaton (for parity automata this simply means adding 1 to all priorities), and finally transform the resulting complement automaton to an equivalent Büchi automaton. (This follows one of the classic approaches for complementing Büchi automata.)

The automata we used in this section for the experiments are taken from the benchmark sets provided by Büchi Store [55] and the Büchi automata generated by SPOT from the formulas in [50]. The former contains 295 NBAs with 1 to 17 states and with 0 to 123 transitions; the latter comprises 90 NBAs with 1 to 165 states and with 0 to 493 transitions. We then considered 300 randomly generated Büchi automata generated by SPOT. All automata are represented in the Hanoi Omega-Automata (HOA) format [10].

4.1 Complementation for Büchi Store

Büchi Store provides 295 nondeterministic Büchi automata; however, since one of such automata has only one state without transitions and GOAL fails in recognising it as a Büchi automaton, we decided to exclude it from the experiments and consider only the remaining 294 cases. In practice, such an automaton accepts the empty language, so its complement accepts the whole Σ^ω. Our tool learns a complement automaton with 3 states and 12 transitions in just 0.16 seconds, so it mildly contributes to demonstrate the efficiency of Buechic. SPOT can also output a complement automaton with 1 state and 1 transition in just 0.02 seconds, which is the smallest Büchi automaton recognising Σ^ω.

The experiments shown in Table 1 are organised by blocks of rows; each block reports the experiments it represents together with the total number of states and transitions of the considered input NBAs and comprises three rows, marked with $|Q|$, $|\mathrm{T}|$, t_c, reporting the overall number of states and transitions, and the total time in seconds, respectively, spent by the different tools for computing the complement automata. For each row, we mark in bold the minimum value among all entries.

By inspecting the entries in Table 1 we can see that our learning based complementation method always outperforms the complementation methods offered by GOAL when we consider the number of states and transitions. If we compare Buechic with SPOT, we can find that for 287 out of the 294 tasks, SPOT produces smaller complement automata than other competitors. Moreover, SPOT is generally faster than the other competitors on all tasks. The results are not surprising since SPOT has implemented a lot of optimisations to reduce the size of the automata and it makes use of very efficient data structure called OBDDs. We note that for Block 2 and Block 3, our complementation method produces much smaller automata than the other tools. We explain later why this happens.

Block 1 reports the results relative to 287 NBAs which can be solved by all algorithms. For those automata, the complement NBAs learned by Buechic have much fewer states and transitions than the automata constructed by the algorithms from GOAL. Moreover, our learning algorithm spent less time than the four complementation algorithms from GOAL. Since on average only 7 equivalence queries are needed for the learning procedure for each NBA and the size of the corresponding FDFA is small, our learning based complementation algorithms perform well for those cases. Nevertheless, SPOT is faster than our learning algorithms and even produces smaller automata. This is because that on average there are only 3.2 states in each Büchi automaton and the optimisations in SPOT work quite well in reducing the size of their deterministic automata as well as their complement Büchi automata.

Block 2 refers to five NBAs on which only the Ramsey-based complementation approach fails. The NBAs in this block induce quite large complement automata, as we can see from the other GOAL solvers, thus quite some work is required for constructing them; this means that a failure can be expected also because the approach is rather slow compared with the other GOAL approaches. This is justified by the fact that, as mentioned in [11], the Ramsey-based complementation is the first complementation method proposed by Büchi [15] and was later improved in [51]. Our approach is much slower than GOAL and SPOT since on average 56 equivalence queries for the learning algorithm are posed before obtaining the appropriate conjecture $\mathcal{F}$. However, the complement automata we learned have much fewer states than all approaches implemented in GOAL—and even SPOT—since the corresponding FDFAs we learned are small. It is worth mentioning that the reduction optimisations in SPOT are less effective here since the constructed automata by SPOT are relatively large. In our experiments, more states in an automaton usually go along with more transitions. The constructed automata by SPOT have fewer transitions since SPOT merges all transitions which have the same source state and target state as one transition, which is different from GOAL and Buechic.

Block 3 contains two NBAs on which both Ramsey- and determinisation-based complementation fail. For one NBA, the determinisation method can complete in 430 seconds and returns a Büchi automaton with 243 states. Regarding the other NBA, the determinisation method cannot terminate in 600 seconds. The bottleneck in this case is the transformation of the NBA to a deterministic parity automaton. In this block, our learning algorithm learns much smaller automata than its competitors since the corresponding FDFAs are very small.

For the given automata of Block 2 and Block 3, which are larger than the automata in Block 1, our algorithm can learn much smaller complement automata than its competitors. This is particularly important when the complementation is used by a model checker to check a system against a property that has been provided as a Büchi automaton or as an ω-regular language, since it helps in limiting the state-explosion problem the model checking algorithms are subject to.

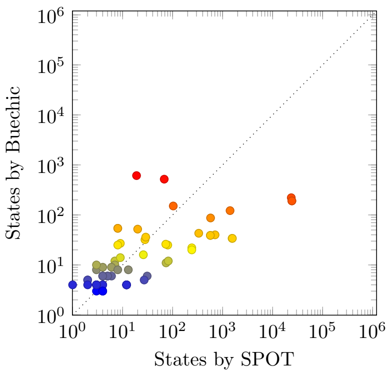
4.2 Complementation for Büchi Automata Generated from Formulas

In order to compare our algorithms with GOAL and SPOT on larger Büchi automata than those in Büchi Store, we consider the Büchi automata generated by SPOT from the formulas in paper [50]. Table 2 gives the complementation results for the Büchi automata of 18 formulas that are explicitly given in [50]. From Table 2, we can conclude that our algorithm can learn much smaller automata than GOAL and SPOT on the large Büchi automata except for the formula pattern f(0, k) where $k \in \{0, 2, 4\}$.

We have also considered 72 further Büchi automata generated from 72 formulas from [50]. In summary, Ramsey-based, Determinisation-based, Rank-based and Slice-based GOAL approaches solve 49, 58, 61, and 62 complementation tasks, respectively, within the time limit, while SPOT solves 66 tasks and Buechic solves 65 tasks. The results are similar as those in Table 2; we thus only discuss the comparison between SPOT and Buechic as best performing tools. Note that there are 64 tasks solved by both SPOT and Buechic and those tasks solved only by SPOT and Buechic separately are disjoint, which implies that our algorithm complements existing complementation approaches very well. Due to the large number of cases, in order to present the experimental results in a more intuitive and compact way for all generated automata, we provide here the scatter plots of Buechic and SPOT in Figure 3 for the 64 commonly solved tasks.

Table 2. Comparison between GOAL, SPOT, and Buechic on complementing generated Büchi automata. The average number of letters in each alphabet is about 29.

Block	Experiments		GOAL				Buechic	SPOT
	(States, Transitions)		Ramsey	Determinisation	Rank	Slice		
1	gf (j=1)	$\lvert Q\rvert$	17	26	34	30	**4**	12
	(4, 8)	|T|	75	95	167	156	**16**	40
		t_c	**<1**	**<1**	**<1**	**<1**	**<1**	**<1**
2	gf (j=2)	$\lvert Q\rvert$	275	405	808	467	**12**	82
	(6,14)	|T|	4609	5881	11408	9440	**158**	736
		t_c	10	6	2	3	**<1**	**<1**
3	gf (j=3)	$\lvert Q\rvert$			25248	15691	**34**	1547
	(8,21)	|T|	–to–	–to–	1097400	1270016	**1512**	60973
		t_c			99	175	**2**	**2**
4	gf (j=4)	$\lvert Q\rvert$					**95**	
	(10,29)	|T|	–to–	–to–	–to–	–to–	**14922**	–to–
		t_c					**45**	
5	gffg1 (j=2)	$\lvert Q\rvert$	559	1497	5773	1333	**22**	242
	(9,22)	|T|	9027	17386	44277	25760	**210**	2613
		t_c	48	19	6	6	**<1**	**<1**
6	gffg2 (j=3)	$\lvert Q\rvert$					**118**	20558
	(21,59)	|T|	–to–	–to–	–to–	–to–	**2662**	806312
		t_c					**6**	9
7	phi1	$\lvert Q\rvert$	17	5	5	5	**4**	5
	(4,10)	|T|	77	15	22	22	16	**11**
		t_c	**<1**	**<1**	**<1**	**<1**	**<1**	**<1**
8	phi2	$\lvert Q\rvert$	31	21	32	19	**3**	8
	(4,10)	|T|	149	70	126	84	**9**	22
		t_c	**<1**	**<1**	**<1**	**<1**	**<1**	**<1**
9	phi3	$\lvert Q\rvert$	39	6	9	8	**4**	6
	(6,14)	|T|	165	19	38	36	16	**13**
		t_c	1	**<1**	**<1**	**<1**	**<1**	**<1**
10	f(0, 0)	$\lvert Q\rvert$	29	**5**	7	7	8	**5**
	(5,9)	|T|	135	20	32	32	36	**9**
		t_c	**<1**	**<1**	**<1**	**<1**	**<1**	**<1**
11	f(0, 2)	$\lvert Q\rvert$	214	13	141	56	10	**7**
	(9,13)	|T|	1164	51	538	268	44	**11**
		t_c	37	**<1**	**<1**	**<1**	**<1**	**<1**
12	f(0, 4)	$\lvert Q\rvert$		15	234	101	12	**9**
	(13,17)	|T|	–to–	59	854	456	52	**13**
		t_c		**<1**	**<1**	**<1**	**<1**	**<1**
13	f(1, 0)	$\lvert Q\rvert$		105	8121	581	**10**	100
	(12,35)	|T|	–to–	534	36685	5096	**69**	563
		t_c		1	5	2	2	**<1**
14	f(1, 2)	$\lvert Q\rvert$			83050	8413	**16**	175
	(31,88)	|T|	–to–	–to–	367699	82832	**109**	1034
		t_c			86	25	**<1**	**<1**
15	f(1, 4)	$\lvert Q\rvert$					**18**	2723
	(107, 306)	|T|	–to–	–to–	–to–	–to–	**125**	20442
		t_c					**2**	3
16	f(2, 0)	$\lvert Q\rvert$				31281	**10**	840
	(18,56)	|T|	–to–	–to–	–to–	529216	**133**	6460
		t_c				103	26	**<1**
17	f(2, 2)	$\lvert Q\rvert$					**9**	1161
	(47,141)	|T|	–to–	–to–	–to–	–to–	**144**	10630
		t_c					2	**<1**
18	f(2, 4)	$\lvert Q\rvert$						
	(165, 493)	|T|	–to–	–to–	–to–	–to–	–to–	–to–
		t_c						



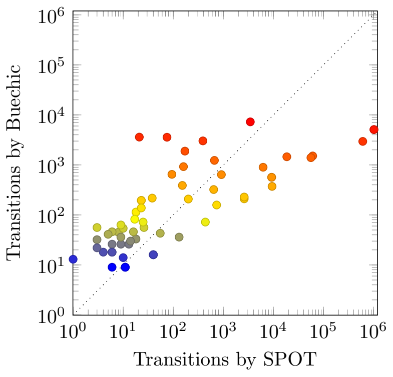


Fig. 3. Comparison between the number of states and transitions of automata generated by SPOT and Buechic on 72 automata corresponding to formulas from [50]. The average number of letters in each alphabet is about 301.

In Figure 3, the coordinate values of the y axis and x axis are the corresponding number of states (resp. transitions) in the complement automata of Buechic and SPOT. All points below the dotted diagonal indicate that the complement automata learned by our algorithm have smaller values than the complement automata constructed by SPOT, which is the case for almost all large examples. We recall that SPOT merges transitions that share the same source state and target state as one transition, so in the right scatter plot of Figure 3, many points are above the diagonal line. Nevertheless, we can learn from the plots that only SPOT produces those automata with more than 10^4 transitions, which indicates that the reduction optimisations of SPOT do not work well on large automata and our algorithm performs much better on large automata. Figure 4 is similar to Figure 3 but it refers to 300 randomly generated Büchi automata with size ranging between 1 and 69 states and between 0 and 263 transitions. The behaviour of SPOT on these automata is similar to the one shown in Figure 3.

In order to show how the growing trend of the number of states in the complement automata of the complementation algorithms behaves when we increase the size of the given Büchi automata in some cases, we take the generated Büchi automata of the formula pattern $\bigwedge_{i=1}^{k}(\mathsf{GF}a_i) \to \mathsf{GF}b$. The growing trend of the number of states in the complement automata for the approaches in GOAL, SPOT, and Buechic are plotted in Figure 5. The number of states in the complement automaton constructed by GOAL and SPOT is growing exponentially with respect to the parameter k, while the number of states in the complement automaton learned by our learning algorithm grows much more slowly than others. The experimental results show that the performance of our algorithm can be much more stable for some automata with their growth of the states. Thus an advantage of our learning approach is that it has potentially better performance on large automata compared to classic complementation techniques.

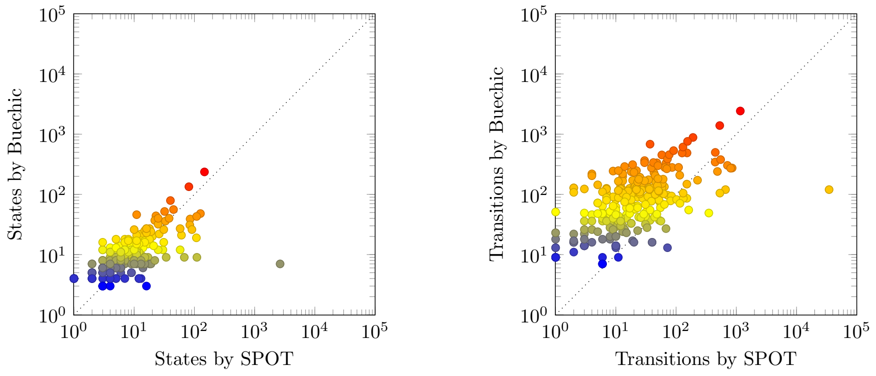


Fig. 4. Comparison between the number of states and transitions of automata constructed by SPOT and Buechic on 300 randomly generated automata. The average number of letters in each alphabet is about 7.

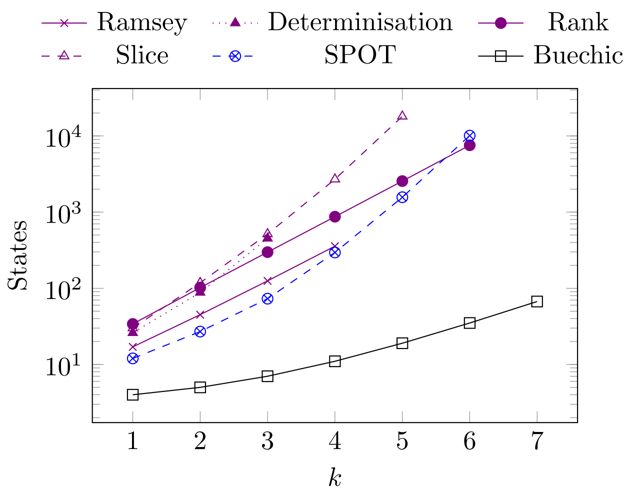


Fig. 5. States comparison of GOAL, SPOT, and Buechic on the formula pattern $\bigwedge_{i=1}^{k}(\mathsf{GF}a_i) \to \mathsf{GF}b$. The number of letters in the alphabet is 2^{k+1} for case k.

4.3 Further Experimental Results

We have conducted further experiments. We have considered *double* complementation on the automata from Büchi Store and generated automata by SPOT. We define double complementation as first using a complementation algorithm to complement the input NBA $\mathcal{B}$ to get the complement $\mathcal{B}^{\mathsf{c}}$ of $\mathcal{B}$; and then complementing $\mathcal{B}^{\mathsf{c}}$ using the same algorithm. It is actually an empty operation. From the experiments, in particular where the complement automata were large, we gained advantage over the competitor algorithms. As another set of benchmarks, we have also considered the complementation of semi-deterministic automata (sometimes called limit-deterministic automata). We considered all 106 SDBAs from [11], and additionally compared them with the NCSB method from [11], which is implemented in GOAL. Note that this is a specialist method and we compete on its soil. This becomes quite clear when comparing with the other general complementation techniques. The experimental evaluation shows that we are competitive with the specialised method from [11] and the highly optimised tool SPOT. Finally, we considered a variation of our learning algorithm, that is, we experimented with checking completeness of $\mathcal{L}(\underline{B}(\mathcal{F})) \cup \mathcal{L}(\mathcal{B})$ instead of testing language inclusion $\mathcal{L}(\underline{B}(\mathcal{F}^{\mathsf{c}})) \subseteq \mathcal{L}(\mathcal{B})$, as proposed in Figure 1. The universality check for $\mathcal{L}(\underline{B}(\mathcal{F})) \cup \mathcal{L}(\mathcal{B})$ is only invoked after the disjointness test for $\mathcal{L}(\underline{B}(\mathcal{F}))$ and $\mathcal{L}(\mathcal{B})$ is passed. According to the experimental results, our handling with the equivalence queries in Figure 1 is more practical.

5 Conclusion

We have introduced a learning based approach for the complementation of Büchi automata. We expected that learning based approaches provide small complementations, that they are less perceptive of the initial representation of the ω-regular language to complement, and that they tend to be fast. In short: that they are practical.

Our experimental evaluation has confirmed our expectation that learning based complementation usually provides smaller complements. More surprisingly, the language inclusion checks in the loop are usually quite fast. As a result, the running time displayed by Buechic is competitive. We have also seen that, while we did gain a clear advantage over the basic techniques as implemented in GOAL, the comparison with SPOT shows that this advantage is not quite in the same league as the advantages one can obtain by high level optimisations implemented in SPOT. We expect that, after the pure technique has proven to be a very strong competitor, many improvements will follow. One improvement is to make the approach symbolic since learning algorithms usually become slow when dealing with large alphabets. This needs a symbolic learning algorithm for FDFAs, which is an interesting future work.

Acknowledgement. This work has been supported by the National Natural Science Foundation of China (Grants 61532019, 61472473, 61650410658), the CAS/SAFEA International Partnership Program for Creative Research Teams, the CAS Fellowship for International Young Scientists, the Sino-German CDZ project CAP (GZ 1023), the EPSRC grants EP/M027287/1 and EP/P020909/1.

References

1. F. Aarts, P. Fiterau-Brostean, H. Kuppens, and F. W. Vaandrager. Learning register automata with fresh value generation. In *ICTAC*, volume 9399 of *LNCS*, pages 165–183, 2015.

2. F. Aarts, B. Jonsson, J. Uijen, and F. Vaandrager. Generating models of infinite-state communication protocols using regular inference with abstraction. *Formal Methods in System Design*, 46(1):1–41, 2015.
3. P. A. Abdulla, Y.-F. Chen, L. Clemente, L. Holík, C. Hong, R. Mayr, and T. Vojnar. Simulation subsumption in Ramsey-based Büchi automata universality and inclusion testing. In *CAV*, volume 6174 of *LNCS*, pages 132–147, 2010.
4. P. A. Abdulla, Y.-F. Chen, L. Clemente, L. Holík, C. Hong, R. Mayr, and T. Vojnar. Advanced Ramsey-based Büchi automata inclusion testing. In *CONCUR*, volume 6901 of *LNCS*, pages 187–202, 2011.
5. R. Alur, P. Černỳ, P. Madhusudan, and W. Nam. Synthesis of interface specifications for Java classes. In *POPL*, pages 98–109, 2005.
6. D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.
7. D. Angluin, U. Boker, and D. Fisman. Families of DFAs as acceptors of omega-regular languages. In *MFCS*, volume 58 of *LIPIcs*, pages 11:1–11:14, 2016.
8. D. Angluin and D. Fisman. Learning regular omega languages. *Theoretical Computer Science*, 650:57–72, 2016.
9. A. Arnold. A syntactic congruence for rational ω-languages. *Theoretical Computer Science*, 39:333–335, 1985.
10. T. Babiak, F. Blahoudek, A. Duret-Lutz, J. Klein, J. Kretínský, D. Müller, D. Parker, and J. Strejcek. The Hanoi omega-automata format. In *CAV*, volume 9206 of *LNCS*, pages 479–486, 2015.
11. F. Blahoudek, M. Heizmann, S. Schewe, J. Strejček, and M.-H. Tsai. Complementing semi-deterministic Büchi automata. In *TACAS*, volume 9636 of *LNCS*, pages 770–787, 2016.
12. B. Bollig, P. Habermehl, C. Kern, and M. Leucker. Angluin-style learning of NFA. In *IJCAI*, pages 1004–1009, 2009.
13. M. Botincan and D. Babic. Sigma*: symbolic learning of input-output specifications. In *POPL*, pages 443–456, 2013.
14. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE TRANSACTIONS ON COMPUTERS*, 1986.
15. J. R. Büchi. On a decision method in restricted second order arithmetic. In *Int. Congress on Logic, Methodology and Philosophy of Science*, pages 1–11, 1962.
16. H. Calbrix, M. Nivat, and A. Podelski. Ultimately periodic words of rational ω-languages. In *MFPS*, volume 802 of *LNCS*, pages 554–566, 1993.
17. S. Chaki, E. M. Clarke, N. Sinha, and P. Thati. Automated assume-guarantee reasoning for simulation conformance. In *CAV*, volume 3576 of *LNCS*, pages 534–547, 2005.
18. S. Chaki and A. Gurfinkel. Automated assume-guarantee reasoning for omega-regular systems and specifications. *Innovations in Systems and Software Engineering*, 7:131–139, 2011.
19. S. Chaki and O. Strichman. Optimized L*-based assume-guarantee reasoning. In *TACAS*, volume 4424 of *LNCS*, pages 276–291, 2007.
20. M. Chapman, H. Chockler, P. Kesseli, D. Kroening, O. Strichman, and M. Tautschnig. Learning the language of error. In *ATVA*, volume 9364 of *LNCS*, pages 114–130, 2015.
21. Y.-F. Chen, A. Farzan, E. M. Clarke, Y.-K. Tsay, and B.-Y. Wang. Learning minimal separating DFAs for compositional verification. In *TACAS*, volume 5505 of *LNCS*, pages 31–45, 2009.

22. Y.-F. Chen, C. Hsieh, O. Lengál, T.-J. Lii, M.-H. Tsai, B.-Y. Wang, and F. Wang. PAC learning-based verification and model synthesis. In *ICSE*, pages 714–724, 2016.
23. J. M. Cobleigh, D. Giannakopoulou, and C. S. Păsăreanu. Learning assumptions for compositional verification. In *TACAS*, volume 2619 of *LNCS*, pages 331–346, 2003.
24. A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *ATVA*, volume 9938 of *LNCS*, pages 122–129. Springer, 2016.
25. A. Farzan, Y.-F. Chen, E. M. Clarke, Y.-K. Tsay, and B.-Y. Wang. Extending automated compositional verification to the full class of omega-regular languages. In *TACAS*, volume 4963 of *LNCS*, pages 2–17, 2008.
26. L. Feng, M. Kwiatkowska, and D. Parker. Compositional verification of probabilistic systems using learning. In *QEST*, pages 133–142, 2010.
27. L. Feng, M. Kwiatkowska, and D. Parker. Automated learning of probabilistic assumptions for compositional reasoning. In *FASE*, volume 6603 of *LNCS*, pages 2–17, 2011.
28. S. Fogarty, O. Kupferman, T. Wilke, and M. Y. Vardi. Unifying Büchi complementation constructions. *Logical Methods in Computer Science*, 9(1), 2013.
29. E. Friedgut, O. Kupferman, and M. Y. Vardi. Büchi complementation made tighter. *International Journal of Foundations of Computer Science*, 17(4):851–868, 2006.
30. S. Gurumurthy, O. Kupferman, F. Somenzi, and M. Y. Vardi. On complementing nondeterministic Büchi automata. In *CHARME*, volume 2860 of *LNCS*, pages 96–110, 2003.
31. F. He, X. Gao, B.-Y. Wang, and L. Zhang. Leveraging weighted automata in compositional reasoning about concurrent probabilistic systems. In *POPL*, pages 503–514, 2015.
32. M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory.* MIT Press, Cambridge, MA, USA, 1994.
33. O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. *ACM Transactions on Computational Logic*, 2(2):408–429, July 2001.
34. R. P. Kurshan. *Computer-aided verification of coordinating processes: the automata-theoretic approach.* Princeton University Press, 1994.
35. Y. Li, Y.-F. Chen, L. Zhang, and D. Liu. A novel learning algorithm for Büchi automata based on family of DFAs and classification trees. In *TACAS*, volume 10205 of *LNCS*, pages 208–226, 2017.
36. O. Maler and A. Pnueli. On the learnability of infinitary regular sets. *Information and Computation*, 118(2):316–326, 1995.
37. O. Maler and L. Staiger. On syntactic congruences for omega-languages. In *STACS*, volume 665 of *LNCS*, pages 586–594, 1993.
38. R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9(5):521–530, 1966.
39. M. Michel. Complementation is more difficult with automata on infinite words. Technical report, CNET, Paris (Manuscript), 1988.
40. D. E. Muller. Infinite sequences and finite machines. In *FOCS*, pages 3–16, 1963.
41. C. S. Păsăreanu, D. Giannakopoulou, M. G. Bobaru, J. M. Cobleigh, and H. Barringer. Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning. *Formal Methods in System Design*, 32(3):175–205, 2008.
42. J.-P. Pécuchet. On the complementation of Büchi automata. *Theoretical Computer Science*, 47(3):95–98, 1986.

43. D. Peled, M. Y. Vardi, and M. Yannakakis. Black box checking. *Journal of Automata, Languages and Combinatorics*, 7(2):225–246, 2001.
44. R. L. Rivest and R. E. Schapire. Inference of finite automata using homing sequences. In *STOC*, pages 411–420, 1989.
45. S. Safra. On the complexity of omega-automata. In *FOCS*, pages 319–327, 1988.
46. W. J. Sakoda and M. Sipser. Non-determinism and the size of two-way automata. In *STOC*, pages 274–286, 1978.
47. S. Schewe. Büchi complementation made tight. In *STACS*, volume 3 of *LIPIcs*, pages 661–672, 2009.
48. S. Schewe and T. Varghese. Tight bounds for the determinisation and complementation of generalised Büchi automata. In *ATVA*, volume 7561 of *LNCS*, pages 42–56, 2012.
49. S. Schewe and T. Varghese. Tight bounds for complementing parity automata. In *MFCS*, volume 8634 of *LNCS*, pages 499–510, 2014.
50. S. Sickert, J. Esparza, S. Jaax, and J. Křetínský. Limit-deterministic Büchi automata for linear temporal logic. In *CAV*, volume 9780 of *LNCS*, pages 312–332, 2016.
51. A. P. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49(3):217–239, 1987.
52. R. tool. http://languageinclusion.org/doku.php?id=tools, 2016.
53. M. Tsai, S. Fogarty, M. Y. Vardi, and Y. Tsay. State of Büchi complementation. *Logical Methods in Computer Science*, 10(4), 2014.
54. M.-H. Tsai, Y.-K. Tsay, and Y.-S. Hwang. GOAL for games, omega-automata, and logics. In *CAV*, volume 8044 of *LNCS*, pages 883–889, 2013.
55. Y.-K. Tsay, M.-H. Tsai, J.-S. Chang, and Y.-W. Chang. Büchi store: An open repository of Büchi automata. In *TACAS*, volume 6605 of *LNCS*, pages 262–266, 2011.
56. M. Y. Vardi. The Büchi complementation saga. In *STACS*, volume 4393 of *LNCS*, pages 12–22, 2007.
57. Q. Yan. Lower bounds for complementation of ω-automata via the full automata technique. *Logical Methods in Computer Science*, 4(1:5), 2008.