Efficient Learning of Weak Deterministic Büchi Automata

Mona Alluwaym[a], Yong Li[b,1], Sven Schewe[a] and Qiyi Tang[a,1]

[a]University of Liverpool, Liverpool, UK
[b]Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
ORCID (Yong Li): https://orcid.org/0000-0002-7301-9234, ORCID (Sven Schewe): https://orcid.org/0000-0002-9093-9518, ORCID (Qiyi Tang): https://orcid.org/0000-0002-9265-3011

Abstract. We present an efficient Angluin-style learning algorithm for weak deterministic Büchi automata (wDBAs). Different to ordinary deterministic Büchi and co-Büchi automata, wDBAs have a minimal normal form, and we show that we can learn this minimal normal form efficiently. We provide an improved result on the number of queries required and show on benchmarks that this theoretical advantage translates into significantly fewer queries: while previous approaches require a quintic number of queries, we only require quadratically many queries in the size of the canonic wDBA that recognises the target language.

1 Introduction

This paper examines the L^* learning framework introduced by Angluin [2], commonly referred to as Angluin-style learning. This framework allows a learner to infer an automaton that represents an unknown system by interacting with a teacher through two types of queries: membership queries and equivalence queries. Membership queries ask whether a specific word u belongs to the target language L, while equivalence queries ask whether a proposed automaton recognises L. After a sufficient number of membership queries, the learner constructs a conjectured automaton and submits it via an equivalence query. The teacher either confirms that the conjecture accurately recognises L or provides a counterexample, which the learner uses to locate the error and further refine the automaton. This process continues until the learner successfully converges to the correct automaton.

The Angluin-style learning framework has been applied in a variety of domains, including learning assumptions for compositional verification [7], software testing [8], detecting errors in network protocol implementations [9], regular model checking [6], extracting automata models for recurrent neural networks [27], verifying binarized neural networks [25], and understanding machine learning models [22].

The foundation of the Angluin-style learning framework relies on the existence of a canonical form for the target automaton representation. For regular languages over *finite* words, this is guaranteed by the Myhill-Nerode theorem [19, 21], which ensures that every regular language has a canonical minimal deterministic finite automaton (DFA). However, the situation is more complex for ω-regular languages over *infinite* words. It remains unknown whether deterministic Büchi and co-Büchi automata have minimal canonical forms, and the question is even more challenging for more powerful classes of ω-automata, such as deterministic Rabin automata. As a result, learning automata over infinite words typically involves first learning a family of DFAs (FDFAs) [3, 14] and then converting the FDFA to the desired ω-automaton [13, 15]. This approach works because FDFAs have canonical forms for every ω-regular language [3, 14]. However, such learning algorithms may suffer from exponential blow-up, as the canonical FDFA can be exponentially larger than the target ω-automaton [15][2]. Nonetheless, a subclass of ω-automata, known as *weak* deterministic Büchi automata (DBAs), has canonical forms and can be directly learned [17].

Weak DBAs (wDBAs) are an intriguing class of ω-automata because they recognise weak languages—languages that can be recognised by both deterministic Büchi automata (DBAs) and deterministic co-Büchi automata (DCAs), or equivalently, languages that can express any Boolean combination of safety properties (ensuring that "nothing bad happens") and reachability properties (ensuring that "good things will happen"). For example, wDBAs have been used to express sets of assignments defined by formulas in additive or linear arithmetic over the reals and integers, i.e., $\langle \mathbf{R}, \mathbf{Z}, +, \leq \rangle$ [4]. For example, the real 3.5 can be encoded as an ω-word $0^+11 \cdot 10^\omega$ using binary representation. Vectors of reals can similarly be encoded over the alphabet $\{0, 1, \cdot\}^n$. The set of satisfying assignments of a formula corresponds to a weak language [4]. This makes weak DBAs a highly expressive class of deterministic automata with a canonical normal form. Minimising wDBAs into this canonical form is tractable [16], whereas minimising even the next larger classes of automata, such as DBAs and DCAs, is already NP-hard [24].

The current state-of-the-art learning algorithm for wDBAs remains the classic L^ω approach [17]. One might wonder whether it is possible to learn wDBAs using the efficient FDFA normal forms described in [3] and [14]. However, these normal FDFAs are quadratic in size with respect to the minimal wDBA [14]; utilising them for learning is thus unlikely to improve over the state-of-the-art.

In this paper, we present a simple and novel wDBA learning al-

[1] Corresponding author {liyong@ios.ac.cn, qiyi.tang@liverpool.ac.uk}

[2] A direct learning algorithm for ω-automata exists [18], which uses extra loop-index queries and thus does not fit into the Angluin-style learning framework.

gorithm that directly learns the canonical form, similar to L^ω. Our main observation is that, by using an *extra function* that stores a loop word for each state during the learning procedure, we can then easily determine whether a state is accepting and thus significantly reduce the queries needed. This is in sharp contrast to the classic algorithm L^ω [17], which lacks this insight and has to compensate for it by looking for the loop words in the observation table each time. This causes asking redundant queries when building the conjectured wDBA. Notably, our algorithm offers several further improvements over the L^ω approach: it reduces the theoretical number of equivalence queries from quadratic to *linear*, matching the theoretical bound in the finite-word setting. Additionally, we improve the theoretical number of membership queries from quintic to *quadratic* in the size of the canonical wDBA. Unlike L^ω, which heavily relies on the observation table to store query results, our algorithm is formulated independently of the data structure used for learning. This flexibility allows us to use observation tables from classic algorithms like L^* [2] as well as classification trees [11], and it will allow us to benefit from future advances in data structures.

Our experimental results demonstrate that our approach requires significantly fewer queries compared to L^ω.

Related work. Learning algorithms for DBAs and DCAs are proposed in [15], where a DBA is learned via a limit family of DFAs (FDFA). An FDFA consists of a *leading* transition system to process the finite prefix u and for each state of the transition system, there is a *progress* DFA to accept the valid finite loops v. This FDFA accepts ω-words uv^ω by accepting the pair (u, v). The target limit FDFA can be quadratically larger than the minimal wDBA [14], making their learned DBA potentially *quadratically larger* than ours. Thus, our algorithm is at least quadratically more efficient in theory. In practice, [15] also requires significantly more queries than our algorithm on the selected benchmarks.

We use the binary analysis algorithm from [23] to analyse the counterexamples for the conjectured wDBA $\mathcal{B}$. The classification tree data structure employed in this paper was originally proposed by [11]. A more advanced tree-based data structure called *discrimination trees* [10] offers additional efficiency. Integrating discrimination trees into our algorithm is a direction of future work.

Compared to L^ω [17], which relies heavily on observation tables to determine the acceptance of states, our approach introduces several key improvements. First, we define an auxiliary function g that maps each state to a loop word, enabling precise localisation of conflicts when a counterexample (CEX) is returned. Without this, L^ω must add all possible suffixes of the CEX to the observation table, resulting in a lot of membership queries to fill the entries of the table. Our algorithm, by contrast, guarantees the addition of a new state after each equivalence query, whereas L^ω provides no such guarantee, with a worst-case bound of n^2 equivalence queries. This gap is not just theoretical—our experiments confirm the practical efficiency of our method. Moreover, our algorithm is presented independently of specific data structures, allowing future adaptability. We further employ global bounds on local parameters to reduce the number of iterations. Together, these design choices result in significantly fewer queries both theoretically and empirically.

2 Preliminaries

In the whole paper, we fix a finite *alphabet* Σ. A *word* is a finite or infinite sequence of letters in Σ; ε denotes the empty word. Let Σ^* and Σ^ω denote the set of all finite and infinite words (or ω-words), respectively. In particular, we let $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. Let ρ be a sequence; we denote by $\rho[i]$ the i-th element of ρ and by $\rho[i..k]$ the subsequence of ρ starting at the i-th element and ending at the $(k-1)$-th element when $0 \leq i < k$, and the empty sequence ε when $i \geq k$. We denote by $\rho[i\ldots]$ the subsequence of ρ starting at the i-th element when $i < |\rho|$, and the empty sequence ε when $i \geq |\rho|$. Given a finite word u and a word w, we denote by $u \cdot w$ (or uw, for short) the concatenation of u and w.

Transition system. A transition system (TS) is a tuple $\mathcal{T} = (Q, q_0, \delta)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, and $\delta : Q \times \Sigma \to Q$ is a transition function. We also extend δ to words in a usual way, by letting $\delta(q, \varepsilon) = q$ and $\delta(q, u \cdot a) = \delta(\delta(q, u), a)$, where $u \in \Sigma^*$ and $a \in \Sigma$. For simplicity, we denote by $\mathcal{T}(u)$ the state $\delta(q_0, u)$ without mentioning its transition function δ and the initial state q_0.

Automata. An automaton on finite words is called a *deterministic finite automaton* (DFA). A DFA $\mathcal{A}$ is formally defined as a tuple $(\mathcal{T}, F)$, where $\mathcal{T}$ is a TS and $F \subseteq Q$ is a set of *accepting* states. An automaton on ω-words is called a *deterministic Büchi automaton* (DBA). A DBA $\mathcal{B}$ is also represented as a tuple $(\mathcal{T}, F)$ like DFA.

The *run* of a DFA $\mathcal{A}$ on a finite word u of length $n \geq 0$ is a sequence of states $\rho = q_0 q_1 \cdots q_n \in Q^+$ such that, for every $0 \leq i < n$, $q_{i+1} = \delta(q_i, u[i])$. A finite word $u \in \Sigma^*$ is *accepted* by a DFA $\mathcal{A}$ if its run $q_0 \cdots q_n$ over u ends in a final state $q_n \in F$. Similarly, the *ω-run* of $\mathcal{A}$ on an ω-word w is an infinite sequence of states $\rho = q_0 q_1 \cdots$ such that, for every $i \geq 0$, $q_{i+1} = \delta(q_i, w[i])$. Let $\inf(\rho)$ be the set of states that occur infinitely often in ρ. An ω-word $w \in \Sigma^\omega$ is *accepted* by a DBA $\mathcal{A}$ if its ω-run ρ of $\mathcal{A}$ over w satisfies that $\inf(\rho) \cap F \neq \emptyset$. The language recognised by a DFA $\mathcal{A}$, denoted $\mathcal{L}_*(\mathcal{A})$, is the set of finite words accepted by $\mathcal{A}$. Similarly, we denote by $\mathcal{L}(\mathcal{A})$ the language recognised by a DBA $\mathcal{A}$.

A set $C \subseteq Q$ is said to be a *strongly connected component* (SCC) if, for every two different states $p, q \in C$, we have $q \in \delta(p, u)$ and $p \in \delta(q, v)$ for some $u, v \in \Sigma^+$. A DBA is called *weak* iff every SCC contains either only accepting states or only rejecting states. Therefore, every ω-run ρ of a weak DBA (wDBA) will eventually get trapped within either an accepting SCC or a rejecting SCC, i.e., $\inf(\rho) \subseteq F$ or $\inf(\rho) \cap F = \emptyset$. We say $\inf(\rho)$ is an accepting loop of ρ if $\inf(\rho) \subseteq F$ and a rejecting loop if $\inf(\rho) \cap F = \emptyset$.

3 Learning wDBA using RCs

Right congruences (RCs) are at the core of the Angluin-style learning framework to distinguish finite word representatives of the states in the target automaton. An RC relation is an equivalence relation $\backsim$ over Σ^* such that $x \backsim y$ implies $xv \backsim yv$ for all $v \in \Sigma^*$. For a word $x \in \Sigma^*$, we denote by $[x]_\backsim$ the equivalence class of $\backsim$ that x resides in. We denote by $\Sigma^*/_\backsim$ the set of equivalence classes of Σ^* under $\backsim$.

An RC $\backsim$ defines a TS $\mathcal{T}$ where each state of $\mathcal{T}$ corresponds to an equivalence class in $\Sigma^*/_\backsim$. The TS $\mathcal{T}[\backsim]$ is defined as follows.

Definition 1 ([19, 21])**.** *Let $\backsim$ be an RC with finite number of equivalence classes. The TS $\mathcal{T}[\backsim]$ induced by $\backsim$ is a tuple (S, s_0, δ) where $S = \Sigma^*/_\backsim$, $s_0 = [\varepsilon]_\backsim$, and for each $u \in \Sigma^*$ and $a \in \Sigma$, $\delta([u]_\backsim, a) = [ua]_\backsim$.*

The RC relation $\backsim_R$ of a regular language R is defined as: $u_1 \backsim_R u_2$ iff for all $v \in \Sigma^*$, we have $u_1 v \in R \Leftrightarrow u_2 v \in R$, where $u_1, u_2 \in \Sigma^*$. In fact, Angluin [2] uses $\backsim_R$ to discover the word representative for each state in the TS $\mathcal{T}[\backsim_R]$ and put all equivalence classes $[u]_{\backsim_R}$ with $u \in R$ to the final state set F. In this way, every

equivalence class of $\backsim_R$ corresponds to a state in the minimal DFA of R.

For ω-regular languages, however, the situation is more involved. The first challenge is how to represent the infinite words as some word, like $ab^1ab^2\cdots ab^k\cdots$ with k increasing to ∞, cannot be stored with finite memory. The observation is that when dealing with an ω-regular language L, we do not need to consider all types of ω-words but a type of ω-words called *ultimately periodic* (UP) words w that are in the form uv^ω, where $u \in \Sigma^*$ and $v \in \Sigma^+$ [5].

Naturally, a UP-word $w = uv^\omega$ can be represented as a pair of finite words (u, v), called a *decomposition* of w. For an ω-language L, let UP$(L) = \{\, uv^\omega \in L \mid u \in \Sigma^* \wedge v \in \Sigma^+ \,\}$ denote the set of all UP-words in L.

Theorem 1 ([5])**.** *Let L and L' be two ω-regular languages. Then $L = L'$ holds if, and only if, UP(L) = UP(L').*

Therefore, every ω-regular language is uniquely characterised by its UP-words. Consequently, to learn an automaton of an ω-regular language L, we only need to learn an automaton whose UP-words is UP(L).

Moreover, unlike other ω-automata, we have a canonical form of wDBAs for each wDBA language L [26]. There is also an RC $\backsim_L$ that induces the TS of the minimal wDBA of L, where the RC $\backsim_L$ is defined as: for $u_1, u_2 \in \Sigma^*$,

$$u_1 \backsim_L u_2 \text{ iff } \forall(x \in \Sigma^*, y \in \Sigma^+).u_1xy^\omega \in L \Leftrightarrow u_2xy^\omega \in L.$$

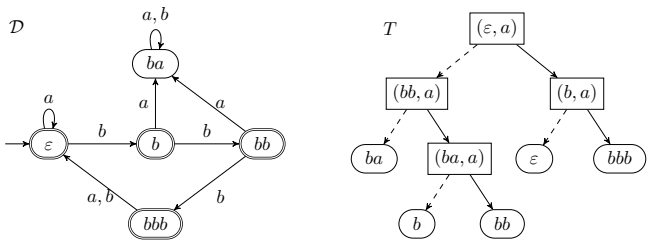

Figure 1. An example wDBA $\mathcal{D}$ and a classification tree T.

Example 1. *Figure 1 gives an example wDBA $\mathcal{D}$, which accepts the language $(a^* \cdot bbb \cdot (a|b))^\omega + (a^* \cdot bbb \cdot (a|b))^* a^\omega$. The states marked with double circles indicate that they are accepting states. On the right hand side, we depict a classification tree T where each leaf node corresponds to an equivalence class of $\backsim_L$ with respect to $\mathcal{L}(\mathcal{D})$. All internal nodes in the tree represent distinguishing words such that a pair (x, y) represents an ω-word xy^ω; they are used to distinguish the word representatives in leaf nodes. Dashed arrows indicate a membership query result of $\bot$, while solid arrows represent a result of $\top$. For example, ϵ and bbb can be distinguished by the node (b, a). The transition from (b, a) to ϵ is represented by a dashed arrow, indicating that $\epsilon \cdot ba^\omega \notin \mathcal{L}(\mathcal{D})$. Conversely, the transition from (b, a) to bbb is shown with a solid arrow, indicating that $bbb \cdot ba^\omega \in \mathcal{L}(\mathcal{D})$.*

According to [17], the canonical equivalence relation $\backsim_L$ can be used to learn the TS of the minimal wDBA for a language L. Furthermore, the set of accepting states is determined by leveraging the samples stored in a data structure known as the observation table [2]. As an initial step, the experiment word c^ω is added to the columns for each letter $c \in \Sigma$. Moreover, upon receiving a counterexample w from the oracle, their algorithm adds all suffixes of w to the columns of the observation table. A suffix of a UP-word uv^ω includes all words of the form xv^ω and v'^ω, where x is a prefix of u and v' is a rotation of v. For instance, the set of suffixes of $ad \cdot (bc)^\omega$ includes $ad \cdot (bc)^\omega$, $d \cdot (bc)^\omega$, $(bc)^\omega$, and $(cb)^\omega$. Figure 3 illustrates an observation table used by [17] to learn the wDBA $\mathcal{D}$ shown in Figure 1. The entry for a row u and column (x, y) corresponds to the membership query result for the decomposition (ux, y). In the observation table $\mathcal{TB}$, the columns include the words a^ω and b^ω, as $\Sigma = \{a, b\}$. For the counterexample $w = (bbbaa, a)$ returned by the teacher, all suffixes of w are added, including $(bbbaa, a)$, $(bbaa, a)$, (baa, a), and $(aa, a) = (\epsilon, a)$. The classification tree T offers a more compact representation for storing membership query results compared to $\mathcal{TB}$. However, due to the suffix-adding operation, their algorithm heavily relies on observation tables, as it remains unclear how to apply this operation directly to classification trees [11]. In contrast, one advantage of our algorithm is that our algorithm can be implemented using either observation tables or classification trees.

$\mathcal{TB}$

	(ε, a)	(ε, b)	$(bbbaa, a)$	$(bbaa, a)$	(baa, a)
ε	$\top$	$\top$	$\top$	$\bot$	$\bot$
b	$\bot$	$\top$	$\top$	$\top$	$\bot$
ba	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
bb	$\bot$	$\top$	$\bot$	$\top$	$\top$
bbb	$\top$	$\top$	$\bot$	$\bot$	$\top$
a	$\top$	$\top$	$\top$	$\bot$	$\bot$
baa	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
bab	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
bba	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
$bbba$	$\top$	$\top$	$\top$	$\bot$	$\bot$
$bbbb$	$\top$	$\top$	$\top$	$\bot$	$\bot$

Figure 2. The observation table $\mathcal{TB}$ for the wDBA $\mathcal{D}$. We omit the columns for the suffixes (aa, a) and (a, a) because the entry values are the same as column (ε, a).

Let $\mathcal{T}$ be the TS of a conjecture wDBA $\mathcal{B}$. We say a decomposition (u, v) is *normalised* if $\mathcal{T}(u) = \mathcal{T}(uv)$. That is, over the word v, $\mathcal{T}$ goes back to the state $\mathcal{T}(u)$. Moreover, we denote by $\mathcal{T}_u(v)$ the state $\mathcal{T}(uv)$.

4 Our wDBA learning algorithm

We first give an overview of our wDBA learning algorithm and then present more details afterwards. Assume that we have a wDBA oracle who knows L and can answer membership queries about L and equivalence queries about whether a given wDBA recognises L.

Our main idea is to use a TS learner to first learn the TS $\mathcal{T}$ of the target wDBA of L and then use membership queries to decide whether a state in the TS belongs to the accepting state set F or not. Then, we obtain our wDBA $\mathcal{B} = (\mathcal{T}, F)$. To this end, our wDBA learner is comprised of three components: the TS learner that learns a TS $\mathcal{T}$ (cf. section 4.1), the acceptance marking component that decides whether a state belongs to F (cf. section 4.2), and a counterexample (CEX) analysis component (cf. section 4.3).

Our learning algorithm starts by using a TS learner to learn a TS $\mathcal{T}$ via membership queries. However, a challenge arises because the learned TS $\mathcal{T}$ lacks an acceptance mechanism F, which will then be addressed by our acceptance marking component. To decide whether a state u in $\mathcal{T}$ belongs to F or not, we can just find a loop word

$v \in \Sigma^+$ such that (u, v) is a *normalised* decomposition in $\mathcal{T}$. We mark the state u as accepting or rejecting, i.e., put u in F or not, depending on the result of the membership query $\mathsf{MQ}(u, v)$. The challenge here is that it is possible to find two different states within the same SCC marked with different acceptance labels. This violates the definition of wDBAs, which requires all states within the same SCC to have a single acceptance label. In this case, we say that we find a *conflict* in $\mathcal{T}$. To build a conjecture wDBA $\mathcal{B}$, we need to first make sure that $\mathcal{T}$ is *conflict-free*. We use the idea presented in [17] to resolve the conflicts. Once no further conflicts have been found, we are able to construct a conjecture wDBA $\mathcal{B} = (\mathcal{T}, F)$ and ask an equivalence query about $\mathcal{B}$.

If the oracle returns "yes" to our equivalence query, it means that our learning task has completed and we can output $\mathcal{B}$ as the correct wDBA of L. In case that we receive the answer "no" together with a counterexample $w \in \mathcal{L}(\mathcal{B}) \ominus L$, the symmetric difference of $\mathcal{L}(\mathcal{B})$ and L, we will call the counterexample analysis component to obtain a valid counterexample for refining current conjecture $\mathcal{B}$. In contrast to that the algorithm by Maler and Pnueli [17] does *not* guarantee to increment the number of states in the conjecture by one after each counterexample guided refinement, our algorithm makes sure of that. This then allows our algorithm to use at most n equivalence queries instead of $\mathcal{O}(n^2)$ in [17], where n is the number of states in the target wDBA $\mathcal{D}$.

We will describe each component of the wDBA learner separately with more details in subsequent sections.

4.1 The TS Learner

We will describe in this section the learner of the TS $\mathcal{T}$.

In [17], Maler and Pnueli utilise an *observation table* [2] to store the membership queries during the learning procedure. To make our algorithm more general, we do not use a concrete data structure such as an observation table, a classification tree [11] or other possible data structures to store membership queries. Instead, we only assume that our learner has a function $f : S \times E \cup S \cdot \Sigma \times E \to \{\top, \bot\}$, where $S \subseteq \Sigma^*$ is a set of word representatives of the equivalence classes (i.e., the state names of the constructed TS), $E \subseteq \Sigma^* \times \Sigma^+$ is a set of experiments (UP-words) used to distinguish the words in S and $\{\top, \bot\}$ is the function codomain. Intuitively, for two different states/representatives $s_1, s_2 \in S$, we must have an experiment word $e \in E$ to distinguish s_1 and s_2, i.e., $f(s_1, e) \neq f(s_2, e)$ according to the RC $\backsim$ we are using.

We let $\mathsf{MQ}(x, y)$ be the result of the membership query of $x \cdot y^\omega$ returned by the oracle. The learning procedure begins by asking membership queries to define the function f over its domain and then constructing a conjecture automaton for asking an equivalence query. Note that $\varepsilon \in S$ initially.

By the definition of $\backsim_L$, for every $u \in S, (x, y) \in E$, we define $f(u, (x, y)) = \mathsf{MQ}(u \cdot x, y)$, i.e., the membership result of $u \cdot xy^\omega$. In particular, for two different word representatives $u_1, u_2 \in S$, there must exist $(x, y) \in E$ such that $f(u_1, (x, y)) \neq f(u_2, (x, y))$, which means that $x \cdot y^\omega$ distinguishes the finite words u_1 and u_2 according to $\backsim_L$. When constructing $\mathcal{T}$, computing the a-successor of a representative u reduces to finding a word representative $u' \in S$ such that $f(u', (x, y)) = f(ua, (x, y))$ for all $(x, y) \in E$. Such a representative u' will be guaranteed to exist during construction. Note that $f(ua, (x, y))$ is also defined since the set $S \cdot \Sigma \times E$ is also part of the domain of f. The initial state will always be the state ε. This way, we obtain the TS $\mathcal{T}$.

Note that, for each $u \in S$, we assume that $u = \mathcal{T}(u)$ in the whole paper. This generally holds for observation tables but *not* for classification trees, However, we can achieve it by extra checks and refinements later given in experiments section.

Now we show that, if the CEX returned to the TS learner is valid (cf. Definition 2), we can refine the current conjecture wDBA. By analysing a valid CEX, we can add a new *experiment* e to the corresponding E in order to distinguish two words $x \cdot a$ and x' that are currently classified as equivalent, i.e., for $x, x' \in S$ and $x \cdot a \in S \times \Sigma$, we have currently $f(x \cdot a) = f(x')$.

Definition 2. *Let $\mathcal{B} = (\mathcal{T}, F)$ be the current conjecture wDBA and (u, v) a decomposition. We say (u, v) is a* valid *CEX of $\mathcal{B}$ if it satisfies that $\mathsf{MQ}(u, v) \neq \mathsf{MQ}(\tilde{u}, v)$ where $\tilde{u} = \mathcal{T}(u)$.*

In the following, we show that as long as (u, v) is a valid CEX, we can use it to refine the current conjecture wDBA $\mathcal{B}$.

Refinement of $\mathcal{B}$. Since $\mathsf{MQ}(u, v) \neq \mathsf{MQ}(\tilde{u}, v)$, it then holds that $u \not\sim \tilde{u}$ because v^ω can be used to distinguish them. We can find a new experiment and a new state as follows. Let $k = |u|$ and $x_i = \mathcal{T}(u[0 \cdots i])$ be the state that $\mathcal{B}$ arrives in after reading the first i letters of u. Specially, $x_0 = \varepsilon$ and $x_k = \tilde{u}$. We construct the following sequence via membership queries: $\mathsf{MQ}(x_0 \cdot u[0 \ldots k], v), \cdots, \mathsf{MQ}(x_i \cdot u[i \ldots k], v), \cdots, \mathsf{MQ}(\tilde{u} \cdot \varepsilon, v)$. Since $\mathsf{MQ}(x_0 \cdot u, v) \neq \mathsf{MQ}(\tilde{u}, v)$ by assumption, this sequence must differ at some indices $0 \leq \ell, \ell + 1 \leq k$.

That is, there must exist the smallest $\ell \in [0, k)$ such that $\mathsf{MQ}(x_\ell \cdot u[\ell] \cdot u[\ell + 1 \ldots k], v) \neq \mathsf{MQ}(x_{\ell+1} \cdot u[\ell + 1 \ldots k], v)$. Then, since $x_{\ell+1} = \mathcal{T}(x_\ell \cdot u[\ell])$, we can use the experiment $e = (u[\ell + 1 \ldots k], v)$ to distinguish $x_\ell \cdot u[\ell]$ and $x_{\ell+1}$. Then, we add $x_\ell \cdot u[\ell]$ to S and e to E.

Note that we use linear search to find ℓ above but we can also use binary search to further reduce the number of membership queries. Therefore, upon receiving a valid CEX, the current conjectured wDBA will be refined.

Lemma 2. *The number of states in $\mathcal{T}$ will increase by one after a valid CEX (u, v) is used to refine the current conjecture.*

4.2 Acceptance marking

In this section, we will explain how to construct a weak DBA $\mathcal{B} = (\mathcal{T}, F)$ from the TS $\mathcal{T}$ by deciding the acceptance label for each state. Since a run of a weak DBA eventually becomes trapped in either an accepting loop or a rejecting loop, it is quite natural to mark a state accepting if it belongs to the loop run of an accepting word and rejecting if belongs to the loop run of a rejecting word.

To this end, we call a state u *transient* if there exists no word $v \in \Sigma^+$ such that $u = \mathcal{T}(uv)$. In other words, u belongs to a trivial SCC. Conversely, if a state u is in a non-trivial SCC, referred to as an *SCC state*, there always exists a *loop* word $v \in \Sigma^+$ such that $u = \mathcal{T}(uv)$.

We will use a function $g : S \to \Sigma^+$ to store a loop word for each state during the learning procedure. This function g is the core part of our algorithm as it allows us to easily analyse the counterexample received from the oracle as described in Section 4.3. More importantly, this function g allows the counterexample analysis component to identify a valid counterexample for $\mathcal{B}$. Consequently, our learning algorithm can increment the number of states in the conjectured wDBA after each refinement guided by the counterexample.

Marking acceptance. Each *transient* state u is directly marked as *rejecting*. For an *SCC state* u, we first identify a loop word $v \in \Sigma^+$ for u, set $g(u) = v$, and then mark u as *accepting* if $\mathsf{MQ}(u, v) = \top$, or as *rejecting* otherwise.

Ideally, after the marking procedure, the resulting wDBA would have all states within the same SCC marked with the same acceptance label. In practice, however, as noted in [17], conflicts may arise during the marking process if $\mathcal{B}$ is not yet the final automaton. One such conflict, referred to as C1, occurs when two states u_1 and u_2 within the same SCC have loop words x and y, respectively, such that $u_1 x^\omega \in L$ while $u_2 y^\omega \notin L$.

Since all states within the same SCC must be either entirely accepting or entirely rejecting, conflict C1 must be resolved before constructing the conjectured wDBA. To address this, we first introduce a second type of conflict, labelled C2, to which conflict C1 may be reduced: there is a state u such that for two of its loop words $x, y \in \Sigma^+$, it holds that $ux^\omega \in L$ but $uy^\omega \notin L$.

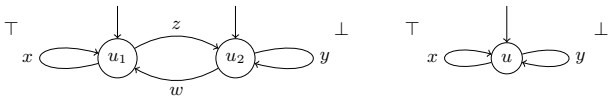

Figure 3. Two conflict cases C1 (left) and C2 (right).

Figure 3 illustrates the two conflict cases. Our approach to resolving these conflicts is inspired by [17]. Specifically, we provide a concrete implementation of the conflict resolution framework proposed in [17] for cases C1 and C2. To resolve conflict C2, we invoke the function resolveConflict(u, x, y), as described in Algorithm 1.

In order to make the explanation more general, we assume that $u = \mathcal{T}(ux) = \mathcal{T}(uy), ux^\omega \in L$ and $uy^\omega \notin L$. We first explain the soundness of resolveConflict(u, x, y), i.e., the returned counterexample is valid. Since $u = \mathcal{T}(ux) = \mathcal{T}(uy)$, for any $h, k > 0$ and for $z = y^k \cdot x^k$, it follows that $u = \mathcal{T}(u \cdot z^{h-1} \cdot y^k) = \mathcal{T}(u \cdot z^h)$. In other words, both $u \cdot z^{h-1} \cdot y^k$ and $u \cdot z^h$ are identified as equivalent to u.

If $\mathsf{MQ}(u \cdot z^{h-1} \cdot y^k, x) = \bot$, it means that we can use x^ω to distinguish $u \cdot z^{h-1} \cdot y^k$ and u since $ux^\omega \in L$. Similarly, if $\mathsf{MQ}(u \cdot z^h, y) = \top$, we can then use y^ω to distinguish $u \cdot z^h$ and u since $uy^\omega \notin L$.

For completeness, we show that the algorithm will terminate. Assume that $\mathcal{D}$ is the minimal wDBA of the target language L with n states. Since the number of SCCs in $\mathcal{D}$ is bounded by n, the word $m := u \cdot z^k$ for any $k \geq n$ may cause $\mathcal{D}$ to alternate between rejecting and accepting SCCs, but at most n alternations. Beyond a certain point (i.e., as $k \geq n$), the words $u \cdot (y^n \cdot x^n)^k \cdot y^n$ and $u \cdot (y^n \cdot x^n)^k \cdot y^n \cdot x^n$ will eventually lead the run in $\mathcal{D}$ to be trapped in the same SCC, whether it is accepting or rejecting.

If the SCC is accepting, then $\mathsf{MQ}(m \cdot y^n \cdot x^n, y) = \top$, the algorithm will return on Line 10, otherwise the algorithm returns on Line 7. Since n is not known a priori, we increment k in the outmost while loop, ensuring that k will reach n in the worst case.

We now discuss how to resolve conflict C1. The idea is that we first find a word z from state u_1 to u_2 and a word w from u_2 to u_1. This is possible since u_1 and u_2 are within the same SCC. Clearly, $u_1 = \mathcal{T}_{u_1}(zw)$ and $u_2 = \mathcal{T}_{u_2}(wz)$.

We first check whether $\mathsf{MQ}(u_1, zw) = \bot$. If so, there is a conflict C2 because $u_1 = \mathcal{T}_{u_1}(zw) = \mathcal{T}_{u_1}(x), ux^\omega \in L$ and $u_1 \cdot (zw)^\omega \notin L$. We can then call resolveConflict(u_1, x, zw) to obtain a valid CEX. Similarly, if $\mathsf{MQ}(u_2, wz) = \top$, we find a conflict C2 because $u_2 = \mathcal{T}_{u_2}(wz) = \mathcal{T}_{u_2}(y), u_2(wz)^\omega \in L$ and $u_2 \cdot (y)^\omega \notin L$. We can also call resolveConflict(u_2, wz, y) to obtain a valid CEX. In the case of $\mathsf{MQ}(u_2, wz) = \bot$ and $\mathsf{MQ}(u_1, zw) = \top$ (or equivalently $\mathsf{MQ}(u_1 z, wz) = \top$), it then follows that $(wz)^\omega$ can distinguish u_2 and $u_1 z$ since $u_2 = \mathcal{T}(u_1 z)$. Hence, $(u_1 z, wz)$ is a valid CEX.

Algorithm 1: resolveConflict(u, x, y) where $u = \mathcal{T}_u(x) = \mathcal{T}_u(y), u \cdot x^\omega \in L$ and $u \cdot y^\omega \notin L$

Input: $u \in \Sigma^*$ and $x, y \in \Sigma^+$
Output: A valid counterexample
1 global k initialised to 1 at the beginning;
2 **while** *true* **do**
3 $h := 1$;
4 **while** $h \leq k$ **do**
5 $z := y^k \cdot x^k$;
6 **if** $\mathsf{MQ}(u \cdot z^{h-1} \cdot y^k, x) = \bot$ **then**
7 return $(u \cdot z^{h-1} \cdot y^k, x)$ as a valid CEX;
8 **end**
9 **if** $\mathsf{MQ}(u \cdot z^h, y) = \top$ **then**
10 return $(u \cdot z^h, y)$ as a valid CEX;
11 **end**
12 $h := h + 1$;
13 **end**
14 $k := k + 1$;
15 **end**

Algorithm 2: resolveConflictState(u_1, u_2, x, y) where $u_1 = \mathcal{T}_{u_1}(x), u_2 = \mathcal{T}_{u_2}(y), u_1 \cdot x^\omega \in L$ and $u_2 \cdot y^\omega \notin L$

Input: $u_1, u_2 \in \Sigma^*, x, y \in \Sigma^+$
Output: A valid counterexample
1 Let $z \in \Sigma^+$ such that $u_2 = \mathcal{T}_{u_1}(z)$;
2 Let $w \in \Sigma^+$ such that $u_1 = \mathcal{T}_{u_2}(w)$;
3 **if** $\mathsf{MQ}(u_1, zw) = \bot$ **then**
4 return resolveConflict(u_1, x, zw) ;
5 **end**
6 **if** $\mathsf{MQ}(u_2, wz) = \top$ **then**
7 return resolveConflict(u_2, wz, y) ;
8 **end**
9 return $(u_1 z, wz)$ as a valid CEX;

This then completes the construction of the conjectured wDBA $\mathcal{B}$ and we can ask an equivalence query $\mathsf{EQ}(\mathcal{B})$.

We remark that the acceptance marking procedure in [17] works by traversing all rows u and columns (x, y) in the observation table and mark the states in the loop run over uxy^ω according to its entry value. It may happen that some SCCs are not marked because they are not covered by the words in the observation table. These SCCs are marked arbitrarily in their algorithm and must be fixed by counterexamples returned from the oracle, while our marking algorithm guarantees that every SCC state will be marked according to their real loop words.

4.3 Counterexample analysis

When the conjectured wDBA $\mathcal{B}$ is not correct, the teacher will reply with the answer "no" alongside a CEX $w \in \mathrm{UP}(L) \ominus \mathcal{L}(\mathcal{B})$, represented by a decomposition. In this section, we will make use of the CEX w to refine the current conjecture $\mathcal{B}$. We say the CEX w is *negative* if $w \in \mathcal{L}(\mathcal{B}) \setminus L$ and *positive* if $w \in L \setminus \mathcal{L}(\mathcal{B})$. As usual, since $\mathcal{B} = (\mathcal{T}, F)$ has a finite number of states, we can easily obtain a normalised decomposition (x, y) of w such that $\mathcal{T}(x) = \mathcal{T}(xy)$. Let $u = \mathcal{T}(x) = \mathcal{T}_u(y)$.

We first analyse the case when w is a negative counterexample, i.e., $w \in \mathcal{L}(\mathcal{B}) \setminus L$. This means that the run ρ over w gets trapped in

an accepting SCC C of $\mathcal{B}$, but it should not be accepted. Since u is in an accepting SCC and $\mathcal{T}$ is a conflict-free TS, we can find its loop word $v = g(u)$ such that $u = \mathcal{T}_u(v)$ and $uv^\omega \in L$. Recall that in the construction of wDBA, we use function g to record the loop word for each SCC state. We first test whether $\mathsf{MQ}(u, y) = \top$, i.e., whether uy^ω is in L or not. If $uy^\omega \in L$, we can use y^ω to distinguish x and u since $xy^\omega \notin L$ and x is currently classified as being equivalent to u. We then return (x, y) as a valid CEX to refine $\mathcal{T}$. Otherwise $uy^\omega \notin L$, together with the fact that $u = \mathcal{T}_u(v), uv^\omega \in L, u = \mathcal{T}_u(y)$ and $uy^\omega \notin L$, we find a conflict type C2. We can then call $\mathsf{resolveConflict}(u, v, y)$ to obtain a valid counterexample.

Now we can analyse the case when w is a positive counterexample, i.e., $w \in L \setminus \mathcal{L}(\mathcal{B})$. Again, this indicates that the run ρ over w is trapped in a rejecting SCC C but it should be accepted. Since C is rejecting, we can find the loop word $v = g(u)$ such that $u = \mathcal{T}_u(v)$ and $uv^\omega \notin L$. We can test whether $\mathsf{MQ}(u, y) = \top$, i.e., whether uy^ω is in L or not. Again, if $uy^\omega \notin L$, we can then use y^ω to distinguish u and x because $u = \mathcal{T}(x)$, indicating that x is currently equivalent to u. Then, we can return (x, y) as a valid CEX to refine $\mathcal{B}$. Otherwise $uy^\omega \in L$, this then leads to a conflict type C2 because $u = \mathcal{T}_u(y), u = \mathcal{T}_u(v), uy^\omega \in L$ and $uv^\omega \notin L$. Hence, we can call $\mathsf{resolveConflict}(u, y, v)$ to obtain a valid CEX to refine $\mathcal{B}$. Lemma 3 then follows.

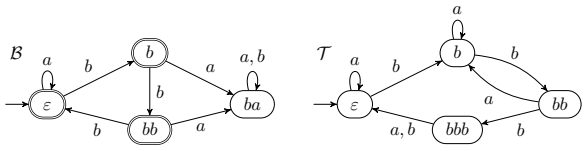

Figure 4. (a) Left: A wDBA $\mathcal{B}$ constructed from the conflict-free TS $\mathcal{T}$, with the mapping $g = \{\varepsilon \mapsto a, b \mapsto bbb, ba \mapsto a, bb \mapsto bbb\}$. (b) Right: A TS $\mathcal{T}$ with the mapping $g = \{\varepsilon \mapsto a, b \mapsto a, bb \mapsto ab, bbb \mapsto abbb\}$, exhibiting conflicts.

Lemma 3. *A valid CEX will be constructed to refine the current wDBA once a counterexample is returned by the oracle.*

Example 2. *(a)* **Conflict-free TS** *Figure 4(a) depicts a TS $\mathcal{T}$ with the function g and its corresponding wDBA $\mathcal{B} = (\mathcal{T}, F)$ where $F = \{\varepsilon, b, bb\}$ during the learning procedure of $\mathcal{D}$ in Figure 1. We can see that for every accepting state $u \in F$, we have $u \cdot (g(u))^\omega \in \mathcal{L}(\mathcal{D})$, while for the rejecting state ba, it holds that $ba \cdot (g(ba))^\omega = ba \cdot a^\omega \notin L$. In this case, $\mathcal{T}$ is a conflict-free TS with respect to the function g.*

Thus, we pose the equivalence query $\mathsf{EQ}(\mathcal{B})$. Since $\mathcal{L}(\mathcal{D}) \neq \mathcal{L}(\mathcal{B})$, the oracle responds with the answer "no" along with a counterexample $w \in \mathcal{L}(\mathcal{D}) \ominus \mathcal{L}(\mathcal{B})$. Suppose $w = bbbbaa^\omega$, which is a positive counterexample such that $w \in \mathcal{L}(\mathcal{D})$ and $w \notin \mathcal{L}(\mathcal{B})$. We then obtain a normalized decomposition $(x, y) = (bbbba, a)$ of w. Here, $\mathcal{T}(x) = \mathcal{T}(xy)$, and we have $\mathcal{T}(bbbba) = ba$. Since $ba \cdot y^\omega \notin \mathcal{L}(\mathcal{D})$ but $x \cdot y^\omega \in \mathcal{L}(\mathcal{D})$, we can use $y^\omega = a^\omega$ to distinguish ba and $x = bbbba$. Consequently, $(bbbba, a)$ is returned as a valid CEX to the TS learner to refine $\mathcal{B}$.

To refine $\mathcal{B}$, we examine the state $ba = \mathcal{T}(bbbba)$ and the corresponding sequence of membership queries:

$$\mathsf{MQ}(\epsilon \cdot bbbba, a), \mathsf{MQ}(b \cdot bbba, a), \mathsf{MQ}(bb \cdot bba, a),$$
$$\mathsf{MQ}(\epsilon \cdot ba, a), \mathsf{MQ}(b \cdot a, a), \mathsf{MQ}(ba \cdot \epsilon, a).$$

The first position at which the query results differ is $\mathsf{MQ}(bb \cdot bba, a) = \top$ and $\mathsf{MQ}(\epsilon \cdot ba, a) = \bot$. Based on this discrepancy, we identify the experiment $e = (ba, a)$, which serves to distinguish between $bb \cdot b$ and ϵ. The word $bb \cdot b$ is then added to S. This refinement ensures the correct TS for $\mathcal{D}$, and with the correct acceptance labelling of ba, the wDBA is accurately constructed.

(b) **TS with conflicts** *If the TS $\mathcal{T}$ is not conflict-free—for instance, as in the TS depicted in Figure 4(b)—we first resolve the conflict before posing an equivalence query. In this case, $\epsilon \cdot (g(\epsilon))^\omega \in L$, whereas $b \cdot (g(b))^\omega = b \cdot a^\omega \notin L$. To address this, we invoke $\mathsf{resolveConflictState}(\epsilon, b, a, a)$.*

We set $z = b$ and $w = bba$, where $\epsilon = \mathcal{T}_b(w)$ and $b = \mathcal{T}_\epsilon(z)$. Following Algorithm 2, we first check $\mathsf{MQ}(\epsilon, zw) = \mathsf{MQ}(\epsilon, bbba) = \bot$. The response is positive since $\epsilon \cdot (bbba)^\omega \in \mathcal{L}(\mathcal{D})$. We then test $\mathsf{MQ}(b, wz) = \mathsf{MQ}(b, bba \cdot b) = \top$, which is also confirmed.

At this point, we call $\mathsf{resolveConflict}(b, bbab, a)$ as per Algorithm 1. The algorithm terminates at $k = 1$ and $h = 1$ at line 7 because $\mathsf{MQ}(m \cdot y, x) = \bot$, where $m = u = b$, $y = a$, and $x = bbab$. This produces a valid counterexample $(ba, bbab)$ for the TS learner.

The TS learner uses this counterexample to identify an experiment $(\epsilon, bbab)$, distinguishing b from ba. Consequently, ba is added to S, refining the TS of $\mathcal{D}$. With the correct acceptance labelling of ba, this process yields the correct wDBA.

4.4 Correctness and Complexity

Let $\mathcal{D}$ be the target minimal wDBA of L and n the number of states in $\mathcal{D}$. Three aspects of correctness and complexity are easy to establish:

1. the states in the successively built function f have pairwise different right languages – this entails that we can produce at most n states;
2. one state is added after every failed equivalence query – thus, the number of equivalence queries is bound by n;
3. we only add one experiment per failed equivalence query – the size of function f, and the number of membership queries used to build it, is therefore in $\mathcal{O}(n^2 \cdot |\Sigma|)$.

The remaining part of complexity is the cost of identifying the experiments added to the table. They fall into failed and successful inner loops of Algorithm 1, both of which occur at most n times and require $\mathcal{O}(n^2)$ many membership queries; a further $\mathcal{O}(n^2)$ membership queries to determine acceptance marking for the SCCs, and finally the membership queries to identify the exact experiments for refining $\mathcal{T}$.

The latter depend on the length ℓ of a counterexample returned by a failed equivalence query; more precisely, the length of the words that we have to base the search on can be up to $\mathcal{O}(n^3 + n^2\ell)$. However, we note that this word can be traversed with a logarithmic search as described in [23], so that only $\mathcal{O}(\log(n^3 + n^2\ell))$ membership queries of this type are needed for each of the up to n failed equivalence queries. Thus, we require $\mathcal{O}(\max(n^2 \cdot |\Sigma|, n \cdot \log \ell))$ membership queries, as $\mathcal{O}(n \cdot \log(n^2 \cdot \ell + n^3)) = \mathcal{O}(n \cdot \log(n^2 \cdot (\ell + n))) = \mathcal{O}(n \cdot \log \ell)$ holds when $\ell \in 2^{\mathcal{O}(n \cdot |\Sigma|)}$; see the appendix for more details.

Theorem 4. *Our wDBA learning algorithm needs at most n equivalence queries and $\mathcal{O}(\max\{n^2 \cdot |\Sigma|, n \cdot \log \ell\})$ membership queries.*

Our wDBA learning algorithm is correct and runs in time polynomial in n.

In [17], the guarantee provided is $\mathcal{O}(n^3 \cdot \ell \cdot |\Sigma|)$; with the additional assumption that the counterexamples provided are minimal, and thus $\ell \in \mathcal{O}(n^2)$, this results in $\mathcal{O}(n^5 \cdot |\Sigma|)$; note that we get our $\mathcal{O}(n^2 \cdot |\Sigma|)$

guarantee under the assumption that $\ell \in 2^{\mathcal{O}(n \cdot |\Sigma|)}$, a much milder assumption than minimality.

We note that as all our conjecture automata are weak by construction, the algorithm would not terminate if the target language is not weak—just as L^* [2] would not terminate if learning, say, a pushdown language.

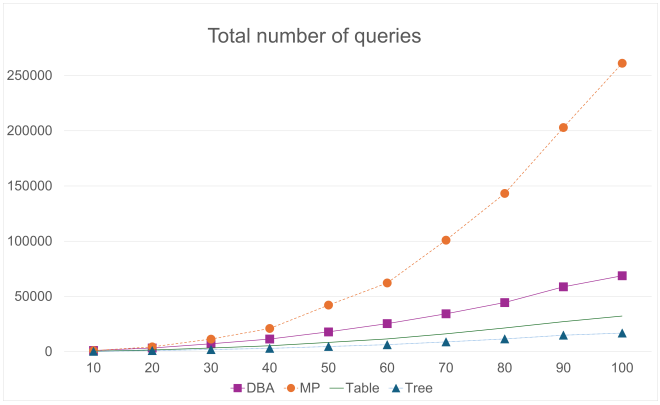

Figure 5. Total number of queries, the sum of equivalence queries and membership queries, for learning wDBAs.

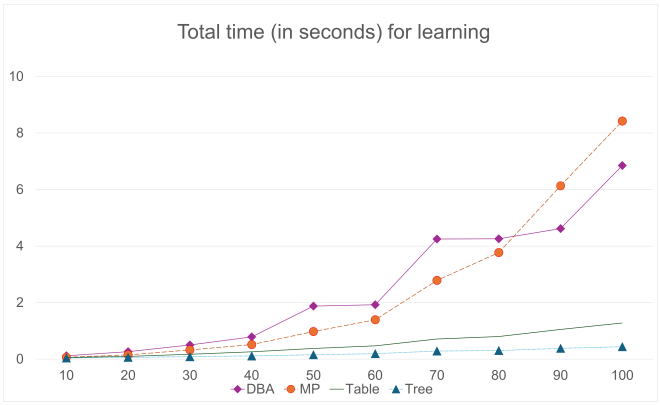

Figure 6. Total time (in seconds) for learning wDBAs.

5 Experiments

To demonstrate the efficiency of our wDBA learning algorithm, we compare it with the algorithm proposed by [17], referred to as MP and the DBA learning algorithm from [15], referred to as DBA. Recall that a DBA is learned in [15] through learning limit FDFAs. Our algorithm is implemented using two data structures: observation tables (Table) and classification trees (Tree). Note that, for the tree-based approach, we may have that for a state $u \in S$, $u \neq \mathcal{T}(u)$. Let $u' = \mathcal{T}(u)$. Since u and u' are distinguished by some experiment $e = vy^\omega \in E$, we can return the example (uv, y) to refine the TS $\mathcal{T}$. This is because $\mathsf{MQ}(uv, y) \neq \mathsf{MQ}(u'v, y)$, so there will be different membership query results between the first $\mathsf{MQ}(x_0 \cdot uv, y)$ and a middle $\mathsf{MQ}(x_{|u|} \cdot v, y)$ where $x_0 = \varepsilon$ and $x_{|u|} = u'$ are two state representatives in the sequence when refining $\mathcal{T}$. We note that there are more advanced techniques to make sure that $u = \mathcal{T}(u)$ holds, see e.g. [10].

We have randomly generated 500 minimal wDBAs with state sizes ranging from 10 to 100, increasing in steps of 10. For each state size, we have generated 50 wDBAs. These wDBAs are non-trivial, each containing between 2 and 10 non-trivial SCCs. A non-trivial SCC is one with at least two states. We would like to point out that passing the equivalence query guarantees that the learned wDBA and the target wDBA are language equivalent. This implies that all our learned automata are correct.

To evaluate the performance of each algorithm, we measured the average number of queries required to learn the target wDBA and the average running time. As shown in Figure 5, Tree consistently requires the fewest queries among all algorithms, followed by Table, DBA, and, lastly, MP. The advantage of our algorithms over MP and DBA becomes more pronounced as the size of the wDBA increases. Both the average number of queries and the time taken to learn a wDBA (cf. Figure 6) grow significantly more slowly with automaton size for Table and Tree compared to MP and DBA. All algorithms are available for use in the learning library ROLL[3] [12]. We provide an artifact [1] to reproduce all experimental results reported in this paper.

We are aware of a collected benchmark in [20][4], which contains DFAs, Moore machines, Mealy machines, interface automata and register automata. For our experiments, we focus on the DFAs. The benchmark provides two types of DFAs: one with 1,000 states and the other with 2,000 states. Each DFA contains a single non-trivial SCC. To construct wDBAs, we applied a simple transformation by making all states within the SCC as accepting. The resulting wDBAs preserve the original structure, yet their minimal equivalents reduce to a single-state automaton. Our algorithms, Table and Tree, consistently require significantly fewer queries than MP to learn these wDBAs. Additional details regarding the results can be found in the appendix. We do not see an easy way to use the other benchmarks in our experiments and thus do not include them in the evaluation.

6 Conclusion

We have introduced a novel learner for weak languages. It improves over the state-of-the-art in terms of complexity, where we reduce the number of queries from quintic to quadratic, with milder assumptions on the length of the counterexamples provided by equivalence queries – where the old quintic result relies on them being quadratic in the size of the automaton, while we can allow for it to be exponential. Further, we have formulated our framework such that it works more flexibly with other data structures, like the classification trees we have implemented. Our experimental results show that, irrespective of the data structure chosen, we always outperform the state-of-the-art algorithm from [17].

From a practical point of view, the advantage our algorithms holds over [17] seems to grow with the size of the automaton. Even for the small 100 state automata, the largest automata we used for evaluation, the advantage our table-based implementation holds over [17] is almost an order of magnitude, while moving to the more modern classification trees as the underpinning data structure further halves the number of queries. With better efficiency, our work would further benefit the applications of learning algorithms in various fields, including verification, testing, and understanding of ML models.

[3] Available at https://github.com/iscas-tis/roll-library
[4] Available at https://automata.cs.ru.nl/

Acknowledgements. This work was supported in part by the CAS Project for Young Scientists in Basic Research (Grant No. YSBR-040), ISCAS Basic Research (Grant Nos. ISCAS-JCZD-202406, ISCAS-JCZD-202302), ISCAS New Cultivation Project ISCAS-PYFX-202201, and by the EPSRC through grants EP/X03688X/1 and EP/X042596/1.

References

[1] M. Alluwaym, Y. Li, S. Schewe, and Q. Tang. wDBA Learning, 2025. Software archived at Zenodo, DOI:10.5281/zenodo.16905843.

[2] D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987. doi: 10.1016/0890-5401(87)90052-6. URL https://doi.org/10.1016/0890-5401(87)90052-6.

[3] D. Angluin and D. Fisman. Learning regular omega languages. *Theor. Comput. Sci.*, 650:57–72, 2016. doi: 10.1016/J.TCS.2016.07.031. URL https://doi.org/10.1016/j.tcs.2016.07.031.

[4] B. Boigelot, S. Jodogne, and P. Wolper. An effective decision procedure for linear arithmetic over the integers and reals. *ACM Trans. Comput. Log.*, 6(3):614–633, 2005. doi: 10.1145/1071596.1071601. URL https://doi.org/10.1145/1071596.1071601.

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

[6] Y. Chen, C. Hong, A. W. Lin, and P. Rümmer. Learning to prove safety over parameterised concurrent systems. In D. Stewart and G. Weissenbacher, editors, *2017 Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, October 2-6, 2017*, pages 76–83. IEEE, 2017. doi: 10.23919/FMCAD.2017.8102244. URL https://doi.org/10.23919/FMCAD.2017.8102244.

[7] J. M. Cobleigh, D. Giannakopoulou, and C. S. Pasareanu. Learning assumptions for compositional verification. In H. Garavel and J. Hatcliff, editors, *TACAS*, volume 2619 of *Lecture Notes in Computer Science*, pages 331–346. Springer, 2003. doi: 10.1007/3-540-36577-X_24. URL https://doi.org/10.1007/3-540-36577-X_24.

[8] M. X. Czerny. Learning-based software testing: Evaluation of angluin's l^* algorithm and adaptations in practice. 2014. URL https://api.semanticscholar.org/CorpusID:55771319.

[9] J. de Ruiter and E. Poll. Protocol state fuzzing of TLS implementations. In J. Jung and T. Holz, editors, *USENIX*, pages 193–206. USENIX Association, 2015. URL https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/de-ruiter.

[10] M. Isberner, F. Howar, and B. Steffen. The TTT algorithm: A redundancy-free approach to active automata learning. In B. Bonakdarpour and S. A. Smolka, editors, *Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings*, volume 8734 of *Lecture Notes in Computer Science*, pages 307–322. Springer, 2014. doi: 10.1007/978-3-319-11164-3_26. URL https://doi.org/10.1007/978-3-319-11164-3_26.

[11] M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994. ISBN 978-0-262-11193-5. URL https://mitpress.mit.edu/books/introduction-computational-learning-theory.

[12] Y. Li, X. Sun, A. Turrini, Y. Chen, and J. Xu. ROLL 1.0: ω-regular Language Learning Library. In *Tools and Algorithms for the Construction and Analysis of Systems – 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I*, volume 11427 of *Lecture Notes in Computer Science*, pages 365–371. Springer, 2019. doi: 10.1007/978-3-030-17462-0_23. URL https://doi.org/10.1007/978-3-030-17462-0_23.

[13] Y. Li, Y. Chen, L. Zhang, and D. Liu. A novel learning algorithm for Büchi automata based on family of DFAs and classification trees. *Inf. Comput.*, 281:104678, 2021. doi: 10.1016/J.IC.2020.104678. URL https://doi.org/10.1016/j.ic.2020.104678.

[14] Y. Li, S. Schewe, and Q. Tang. A novel family of finite automata for recognizing and learning ω-regular languages. In É. André and J. Sun, editors, *ATVA*, volume 14215 of *Lecture Notes in Computer Science*, pages 53–73. Springer, 2023. doi: 10.1007/978-3-031-45329-8_3. URL https://doi.org/10.1007/978-3-031-45329-8_3.

[15] Y. Li, S. Schewe, and Q. Tang. Angluin-style learning of deterministic Büchi and co-Büchi automata. In *Proceedings of the Thirty-third International Joint Conference on Artificial Intelligence, IJCAI 2024*. ijcai.org, 2024.

[16] C. Löding. Efficient minimization of deterministic weak omega-automata. *Inf. Process. Lett.*, 79(3):105–109, 2001. doi: 10.1016/S0020-0190(00)00183-6. URL https://doi.org/10.1016/S0020-0190(00)00183-6.

[17] O. Maler and A. Pnueli. On the learnability of infinitary regular sets. *Inf. Comput.*, 118(2):316–326, 1995. doi: 10.1006/inco.1995.1070. URL https://doi.org/10.1006/inco.1995.1070.

[18] J. Michaliszyn and J. Otop. Learning infinite-word automata with loop-index queries. *Artif. Intell.*, 307:103710, 2022. doi: 10.1016/J.ARTINT.2022.103710. URL https://doi.org/10.1016/j.artint.2022.103710.

[19] J. Myhill. Finite automata and the representation of events. In *Technical Report WADD TR-57-624*, page 112–137, 1957.

[20] D. Neider, R. Smetsers, F. W. Vaandrager, and H. Kuppens. Benchmarks for automata learning and conformance testing. In T. Margaria, S. Graf, and K. G. Larsen, editors, *Models, Mindsets, Meta: The What, the How, and the Why Not? - Essays Dedicated to Bernhard Steffen on the Occasion of His 60th Birthday*, volume 11200 of *Lecture Notes in Computer Science*, pages 390–416. Springer, 2018. doi: 10.1007/978-3-030-22348-9_23. URL https://doi.org/10.1007/978-3-030-22348-9_23.

[21] A. Nerode. Linear automaton transformations. In *American Mathematical Society*, page 541–544, 1958.

[22] A. Ozaki. Actively learning from machine learning models with queries and counterexamples (extended abstract). In U. Endriss, F. S. Melo, K. Bach, A. J. B. Diz, J. M. Alonso-Moral, S. Barro, and F. Heintz, editors, *ECAI 2024 - 27th European Conference on Artificial Intelligence, 19-24 October 2024, Santiago de Compostela, Spain - Including 13th Conference on Prestigious Applications of Intelligent Systems (PAIS 2024)*, volume 392 of *Frontiers in Artificial Intelligence and Applications*, pages 25–26. IOS Press, 2024. doi: 10.3233/FAIA240462. URL https://doi.org/10.3233/FAIA240462.

[23] R. L. Rivest and R. E. Schapire. Inference of finite automata using homing sequences. *Inf. Comput.*, 103(2):299–347, 1993. doi: 10.1006/INCO.1993.1021. URL https://doi.org/10.1006/inco.1993.1021.

[24] S. Schewe. Beyond hyper-minimisation—minimising DBAs and DPAs is NP-complete. In K. Lodaya and M. Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, volume 8 of *LIPIcs*, pages 400–411. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010. doi: 10.4230/LIPICS.FSTTCS.2010.400. URL https://doi.org/10.4230/LIPIcs.FSTTCS.2010.400.

[25] A. Shih, A. Darwiche, and A. Choi. Verifying binarized neural networks by angluin-style learning. In M. Janota and I. Lynce, editors, *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 354–370. Springer, 2019. doi: 10.1007/978-3-030-24258-9_25. URL https://doi.org/10.1007/978-3-030-24258-9_25.

[26] L. Staiger. Finite-state omega-languages. *J. Comput. Syst. Sci.*, 27(3):434–448, 1983. doi: 10.1016/0022-0000(83)90051-X. URL https://doi.org/10.1016/0022-0000(83)90051-X.

[27] G. Weiss, Y. Goldberg, and E. Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In J. G. Dy and A. Krause, editors, *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 5244–5253. PMLR, 2018. URL http://proceedings.mlr.press/v80/weiss18a.html.

A Correctness and Complexity

Let $\mathcal{D}$ be the target minimal wDBA of L and n the number of states in $\mathcal{D}$. Three aspects of correctness and complexity are easy to establish:

1. the states in the successively built function f have pairwise different right languages – this entails that we can produce at most n states;
2. one state is added after every failed equivalence query – thus, the number of equivalence queries is bound by n;
3. we only add one experiment per failed equivalence query – the size of the function f, and the number of membership queries used to build it, is therefore in $\mathcal{O}(n^2|\Sigma|)$.

The remaining part of complexity is the cost of identifying the experiments added to the table.

Since in Algorithm 1, the variable k is a global variable and only increases to at most to n during learning; these increases happen when the inner loop of Algorithm 1 was unsuccessful in identifying a counterexample. The number of membership queries resulting from these unsuccessful attempts is $\mathcal{O}(n^2)$ and thus dominated by the $\mathcal{O}(n^2 \cdot |\Sigma|)$ queries from (3).

When the inner loop is successful in identifying a counterexample, a new state is added, and this happens only $\leq n$ times, so that the number of membership queries used in successful iterations of the inner loop is bounded by $2n^2$.

For each acceptance marking, we only use t membership queries for a TS with t states to decide the acceptance of each state. The number of membership queries is $\sum_{t=1}^{n} t \in \mathcal{O}(n^2)$. In the whole learning procedure, Algorithm 2 uses $2n$ membership queries because it can only be called at most n times.

Now we discuss about the number of membership queries used in refining $\mathcal{T}$. For a valid CEX (u', v') for $\mathcal{T}$, the number of membership queries used is $\mathcal{O}(|u'|)$ if we use linear search and $\mathcal{O}(\log |u'|)$ when we use binary search. So we only need to estimate the length of u'.

First, when refining the TS $\mathcal{T}$, a new state representative $u \cdot a$ is obtained from an existing state u and a letter extension a. Hence, the state representatives in the correct $\mathcal{T}$ has length at most n since the first state representative ε has length 0 and there are only n state representatives.

There are two sources of counterexamples for refining $\mathcal{T}$: one from acceptance marking and the other from CEX analysis component. For the first, the input words u, x and y have length $\mathcal{O}(n)$, where x and y are two different loop words of state u. So, the length of the prefix u' returned from two algorithms will have length $\mathcal{O}(n^3)$.

Now we discuss the length of prefixes of counterexamples generated by the counterexample analysis component when a CEX (x', y') is returned from the oracle. Let $\ell = |x'| + |y'|$ be the length of the *longest* CEX returned.

The length of the prefix u' of a valid counterexample $u \cdot [y^k \cdot x^k]^h$ or $u \cdot [y^k \cdot x^k]^{h-1} \cdot y^k$ returned by resolveConflict is $\mathcal{O}(n^2 \cdot \ell + n^3)$, where $|u|$ is bounded by n and, for $|x|$ and $|y|$, one is bounded by ℓ and the other by n in the worst case.

Therefore, we only need $\mathcal{O}(n^2 \cdot \ell + n^3)$ membership queries for refining conjecture TS $\mathcal{T}$ every time if we use linear search. But we only need $\mathcal{O}(\log(n^2 \cdot \ell + n^3))$ membership queries with binary search. Since we need to refine $\mathcal{T}$ for at most n times, then the number of membership queries needed there is $\mathcal{O}(n \cdot \log(n^2 \cdot \ell + n^3))$. It then follows that the number of membership queries consumed by our algorithm is $\mathcal{O}(\max(n^2 \cdot |\Sigma|, n \cdot \log(n^2 \cdot \ell + n^3))$.

We note that $\mathcal{O}(n \cdot \log(n^2 \cdot \ell + n^3)) = \mathcal{O}(n \cdot \log(n^2 \cdot (\ell + n))) = \mathcal{O}(n \cdot (\log n^2 + \log(\ell + n))) = \mathcal{O}(n \cdot \log \ell)$ holds if the second term dominates and we assume that $\ell \in 2^{\mathcal{O}(n \cdot |\Sigma|)}$. So in summary, we obtain our main result.

Theorem 4. *Our wDBA learning algorithm needs at most n equivalence queries and $\mathcal{O}(\max\{n^2 \cdot |\Sigma|, n \cdot \log \ell\})$ membership queries.*

Our wDBA learning algorithm is correct and runs in time polynomial in n.

In [17], the guarantee provided is $\mathcal{O}(n^3 \cdot \ell \cdot |\Sigma|)$; with the additional assumption that the counterexamples provided are minimal, and thus $\ell \in \mathcal{O}(n^2)$, this results in $\mathcal{O}(n^5 \cdot |\Sigma|)$; note that we get our $\mathcal{O}(n^2 \cdot |\Sigma|)$ guarantee under the assumption that $\ell \in 2^{\mathcal{O}(n \cdot |\Sigma|)}$, a much milder assumption than minimality.

B Experiments

DBA Learning Algorithm We present the complete experimental results on the randomly generated wDBAs with the learning algorithm for DBAs proposed in [15], which is referred to as DBA. The table below reports the average number of queries required to learn the target wDBAs by the four algorithms.

Size	DBA	MP	Table	Tree
10	1022.70	960.64	393.22	254.46
20	3327.54	4283.12	1454.60	900.94
30	7100.54	11337.86	3262.88	1886.54
40	11418.94	20932.16	5265.00	2936.94
50	17869.08	42094.92	8317.58	4630.58
60	25349.06	62179.50	11510.66	6276.00
70	34230.96	100934.16	16083.08	8810.94
80	44437.68	143230.18	21410.68	11513.10
90	58640.84	202896.64	27128.64	14841.84
100	68700.80	261111.70	32286.96	16813.64

The table below presents the average running time required by the four algorithms to learn the target wDBAs.

Size	DBA	MP	Table	Tree
10	0.11780	0.06510	0.05724	0.04030
20	0.26428	0.14988	0.09938	0.06324
30	0.50000	0.32686	0.17346	0.08648
40	0.78704	0.51622	0.26170	0.11286
50	1.87838	0.97790	0.37732	0.15364
60	1.92722	1.39432	0.46960	0.19368
70	4.24870	2.78708	0.71104	0.28710
80	4.25622	3.76858	0.80084	0.30426
90	4.61958	6.13236	1.04892	0.38436
100	6.85212	8.42328	1.27996	0.44038

DFA Benchmarks We present experimental results on the DFA benchmarks introduced in [20]. After transformation and minimisation, all DFAs with 1,000 states result in trivial wDBAs—each comprising a single state with the same number of transitions. The same outcome is observed for DFAs with 2,000 states. Thus, although there are 3,600 DFAs of each type, the number of queries required by each algorithm remains consistent across all instances of the same type. For both sets of wDBAs (prior to minimisation), our algorithms consistently outperform MP, while DBA fails to handle these cases due to stack overflows, as shown in the table below.

Method	EQs	MQs	Total	EQs	MQs	Total
	1,000 states			2,000 states		
DBA	-	-	-	-	-	-
MP	1	420	421	1	110	111
Table	1	22	23	1	12	13
Tree	1	1	2	1	1	2