

Constructing Deterministic Parity Automata from Positive and Negative Examples

León Bohn ✉ 

RWTH Aachen University, Ahornstr. 55, 52074 Aachen, Germany

Christof Löding ✉ 

RWTH Aachen University, Ahornstr. 55, 52074 Aachen, Germany

Abstract

We present a polynomial time algorithm that constructs a deterministic parity automaton (DPA) from a given set of positive and negative ultimately periodic example words. We show that this algorithm is complete for the class of ω -regular languages, that is, it can learn a DPA for each regular ω -language. For use in the algorithm, we give a definition of a DPA, that we call the precise DPA of a language, and show that it can be constructed from the syntactic family of right congruences for that language (introduced by Maler and Staiger in 1997). Depending on the structure of the language, the precise DPA can be of exponential size compared to a minimal DPA, but it can also be a minimal DPA. The upper bound that we obtain on the number of examples required for our algorithm to find a DPA for L is therefore exponential in the size of a minimal DPA, in general. However we identify two parameters of regular ω -languages such that fixing these parameters makes the bound polynomial.

2012 ACM Subject Classification Theory of computation \rightarrow Automata over infinite objects

Keywords and phrases deterministic parity automata, learning from examples, learning in the limit

Funding *León Bohn*: Supported by DFG grant LO 1174/7-1

1 Introduction

Construction of deterministic finite automata (DFA) from labeled example words is usually referred to as passive learning of automata, and has been studied since the 1970s [7, 35, 19], see [25] for a survey. A passive learning algorithm (passive learner for short) receives a sample $S = (S_+, S_-)$ of positive and negative example words as input, and should return a DFA that accepts all words in S_+ and rejects all words in S_- . In order to find passive learners that are robust and generalize from the sample, Gold proposed the notion of “learning in the limit” [18]. Given a class \mathcal{C} of regular languages, a learner is said to learn every language in \mathcal{C} in the limit, if for each language $L \in \mathcal{C}$ there is a DFA \mathcal{A} and a characteristic sample S^L that is consistent with L , such that the learner returns \mathcal{A} for each sample that is consistent with L and contains all examples from S^L . In this case, we also say that the learner is complete for the class \mathcal{C} .

In active DFA learning, the learning algorithm (referred to as active learner) has access to an oracle for a regular target language L . Angluin proposed an active learner that finds the minimal DFA for a target language L in polynomial time based on membership and equivalence queries [1]. A membership query asks for a specific word if it is in the target language, and the oracle answers “yes” or “no”. An equivalence query asks if a hypothesis DFA accepts the target language, and the oracle provides a counterexample if it does not.

Starting from these first works on DFA learning, many variations of the basic algorithms have been developed and were implemented in recent years, e.g., in the framework learnlib [21] and the library flexfringe [37].

The key property that is used in most learning algorithms for regular languages is the characterization of regular languages by the Myhill/Nerode congruence: For a language L ,

two words u, v are equivalent if for each word w , either both uw, vw are in L , or both are not in L . It is a basic result from automata theory that L is regular iff the Myhill/Nerode congruence has finitely many classes, and that these classes can be used as state set of the minimal DFA for L (see basic textbooks on automata theory, e.g. [20]). In particular, this means that two words can lead to the same state if they cannot be separated by a common suffix. Based on this observation, the transition structure of a minimal n state DFA can be fully characterized by a set of examples (words together with the information whether they are in L or not) whose size is polynomial in n .

In this paper, we consider the problem of constructing deterministic automata on infinite words, so called ω -automata, from examples. Automata on infinite words have been studied since the early 1960s as a tool for solving decision problems in logic [11] (see also [32]), and are nowadays used in procedures for formal verification and synthesis of reactive systems (see, e.g., [5, 34, 28] for surveys and recent work). But in contrast to standard finite automata, very little is known about learning deterministic ω -automata.

The first problem, how to represent infinite example words, is easily solved by considering ultimately periodic words. An infinite word is called periodic if it is of the form v^ω for a finite word v , and ultimately periodic if it is of the form uv^ω for finite words u, v (where v must be non-empty). It is a classical result that a regular ω -language is uniquely determined by the ultimately periodic words that it contains [11].

The main obstacle in learning deterministic ω -automata is that there is no Myhill/Nerode-style characterization of deterministic ω -automata. It is still true that two finite words u, v that are separated by L with a common suffix w (which is an infinite word in this case) have to lead to different states in any deterministic ω -automaton for L . But it is not true anymore that all words that are not separated can lead to the same state. Currently, there are no active or passive polynomial time learners that can infer a deterministic ω -automaton for each regular ω -language. The existing algorithms either learn different representations, use information about the target automaton, or can only learn subclasses of the regular ω -languages (see related work at the end of this introduction).

In this paper, we focus on passive learning of deterministic parity automata (DPA). We show that there is a polynomial time passive learner that can learn a DPA for every regular ω -language in the limit. To the best of our knowledge, this is the first polynomial time passive learner for deterministic ω -automata that can learn every regular ω -language in the limit. In more detail, our contributions can be summarized as follows:

1. We introduce a DPA for a regular language L that we call the precise DPA. The priority assignment computed by this DPA corresponds to a priority assignment that is obtained in a natural way by analyzing the periodic parts of words in the language.
2. We show how the precise DPA for L can be constructed from the syntactic family of right congruences (FORC) of L [27]. The construction proposed in [2] first builds a nondeterministic Büchi automaton whose size is polynomial in the size of the FORC, and then determinizes this Büchi automaton. While it is known that the conversion from the syntactic FORC into a DPA must be exponential [2], our construction is only exponential in the number of required priorities. In particular, it is polynomial (in the size of the syntactic FORC) if the number of priorities is fixed.
3. We present a polynomial time algorithm for constructing a consistent DPA from given sets of positive and negative examples of ultimately periodic words. The algorithm can be seen as an extension of the algorithm from [9], which is for deterministic Büchi automata. However, this extension has completely new parts that are based on the insights on the precise DPA of a language and its construction from the syntactic FORC. The main steps

of the algorithm are:

- a. Infer a FORC from the examples using a state merging technique as in [8, 9].
- b. Compute a priority assignment on the prefixes of the example words, based on the construction of the precise DPA from the syntactic FORC mentioned in contribution 2. We show that, although the construction might yield an exponential result, if it is only applied to the example words, then it induces a priority assignment of polynomial size.
- c. Use (as black box) an active learning algorithm for Mealy machines (on finite words) to infer a DPA that is consistent with the sample, using the priority assignment from the previous step for answering queries of the active learning algorithm.

We show that this algorithm, in the limit, infers a DPA for each regular ω -language L (either the precise DPA or a smaller one). In general, our upper bound for the number of examples that is required for inferring a DPA for L is exponential in the size of a minimal DPA for L . However, we identify two parameters of regular ω -languages such that the required data is polynomial if we fix these parameters. This generalizes the currently known results for passive learning of deterministic ω -automata [4, 8, 9] to the class of all regular ω -languages.

The paper is structured as follows. In Section 2 we introduce required notation and results. In Section 3 we introduce the notion of precise DPA and prove some results on this class of DPAs. In Section 4 we show how to construct the precise DPA of a language from its syntactic FORC. In Section 5 we present the learning algorithm, and in Section 6 we conclude.

Related Work

The first paper explicitly dealing with construction of ω -automata from examples that we are aware of is [15], where example words w are finite and can be of one of the following four types: all ω -words with w as prefix are in the language, at least one such ω -word is in the language, all of them are outside the language, or at least one is outside the language. It is shown in [15] that from such examples, an adaption of a state merging technique for DFA can learn all safety languages in the limit and runs in polynomial time on a sample (the class of safety languages corresponds to the regular ω -languages that can be characterized by forbidden prefixes, and is quite restricted).

The construction of nondeterministic Büchi automata (NBAs) from examples is considered in [6] by a reduction to SAT. This construction is used in order to reduce the size of a given NBA. Roughly, the algorithm collects some ultimately periodic example words of the form uv^ω from the given NBA, and then checks if there is a smaller NBA consistent with these words (using a SAT instance). If it finds one, then it checks whether it is equivalent to the given NBA. Otherwise, it adds a new example obtained from the equivalence test and continues. Since the algorithm uses a reduction to SAT, it is clearly not a polynomial time algorithm.

The construction of deterministic parity automata from examples is considered in [4] for the class of IRC(parity) languages, which are the ω -languages that can be accepted by a parity automaton that uses the Myhill/Nerode congruence as transitions structure. The transition structure can hence be inferred from the examples in the same way as for DFA. It is shown in [4] that the algorithm can infer every such language with polynomial time and data by using a decomposition of parity automata that is known from [13] where it was used for the minimization of the number of required priorities.

The well-known RPNI algorithm [30] that infers DFA from examples by a state merging technique has been adapted to deterministic ω -automata in [8], resulting in a polynomial time

passive learner that can infer all IRC(parity) languages in the limit from polynomial data (the same is shown for other acceptance conditions like Büchi, generalized Büchi, and Rabin). The algorithm can also infer automata for languages that are not in this class, however it is also known that there are regular ω -languages for which it cannot infer a correct automaton.

Finally, [9] presents a polynomial time passive learner for deterministic Büchi automata (DBA) that can infer a DBA for every DBA-recognizable language in the limit. The best known upper bound for the number of examples and the size of the resulting DBA is, however, exponential (in the size of a minimal DBA for the language). For the class of IRC(Büchi) languages this algorithm only requires polynomial data.

There are also a few active learning algorithms for ω -languages. We are focusing on passive learning here, but as shown in [8, Proposition 13], a polynomial time active learner can be turned into a polynomial time passive learner through simulation, given that the class of target automata satisfies certain properties. We briefly summarize active learning algorithms for ω -languages in this context. If not mentioned otherwise, the active learners use membership and equivalence queries.

The first active learning algorithm for ω -languages can learn deterministic weak Büchi automata in polynomial time [26]. This algorithm and the class of target automata satisfy the properties from [8] and thus can be used to build a passive learner for deterministic weak Büchi automata, which define a subclass of IRC(Büchi) languages.

The first active learner for the full class of regular ω -languages was proposed in [17]. It can learn an NBA for each regular ω -language L by learning a representation of L using finite words called $L_\$$ that was proposed in [12]. This representation contains all finite words of the form $u\$v$ (for a fresh symbol $\$$) such that the ultimately periodic ω -word uv^ω is in L . A DFA for this language can be learned using known active learning algorithms for DFAs. The main difficulty is that the oracle expects an ω -language on equivalence queries, and that an intermediate DFA in the learning process might be inconsistent in the sense that it accepts $u\$v$ and rejects $u'\$v'$, although $uv^\omega = u'(v')^\omega$. The algorithm in [17] transforms the DFAs into NBAs such that progress in the DFA learning algorithm can be ensured. The resulting active learner can learn an NBA for every regular ω -language L in time polynomial in the minimal DFA for $L_\$$ and in the length of the shortest counterexample returned by the teacher. Note, however, that the size of a minimal DFA for $L_\$$ may be exponential in the size of a minimal NBA for L .

A similar idea is used in [24] that also presents an active learner for NBAs. Instead of $L_\$$, the algorithm learns a representation of the target ω -language that is called family of DFAs (FDDFA for short). This formalism has been introduced in [22] based on the notion of family of right congruences (FORC for short) from [27] (the technical report of [27] dates back to the same year as [22]). FDDFAs are very similar to the $L_\$$ -representation by DFAs. In FDDFAs, the pairs are represented by several DFAs, one (actually just a deterministic transition system without accepting states) for the first component, called the leading automaton, and one DFA for each state q of the leading DFA, called the progress automaton for q . A pair (u, v) is accepted if v is accepted from the progress DFA of state q that is reached via u in the leading automaton. In contrast to a DFA for $L_\$$, an FDDFA needs only to correctly accept/reject pairs (u, v) for which u and uv reach the same state in the leading automaton (other pairs are don't cares). For this reason, the (syntactic) FDDFA can be exponentially more succinct than a minimal DFA for $L_\$$ [3, Theorem 2] (the statement in [3] is for the periodic FDDFA, which is almost the same as a minimal DFA for $L_\$$).

The algorithm in [24] uses an active FDDFA learning algorithm as presented in [3]. For FDDFAs there is the same difficulty as for the $L_\$$ representation: An FDDFA might accept

some decompositions of an ultimately periodic word and reject others. In [3], the assumption is that the teacher can take a hypothesis in form of an FDFA without transforming the representation into an ω -regular language first. So the outcome of the learning algorithm from [3] is an FDFA that represents an ω -language but the intermediate FDFAs submitted in equivalence queries might not. For turning this into a learning algorithm with a teacher for regular ω -languages, [24] proposes two alternative constructions for transforming FDFAs into NBAs.

Since these two active learners produce NBAs, they cannot be used to build a passive learner for a deterministic automaton model.

It is shown in [2] that saturated FDFAs have many good closure and algorithmic properties similar to deterministic automata. An FDFA is saturated if it accepts all relevant decompositions of an ultimately periodic word, or rejects all of them. The active FDFA learning algorithm from [3] learns saturated FDFAs for a regular ω -language (since the teacher is assumed to return a counterexample as long as the hypothesis FDFA is not saturated). However, when using this algorithm in a passive setting by implementing the oracle based on the given examples (as generically described in [8]), the run of the active learner might terminate with an FDFA that only has the saturation property on the given example words, but is not saturated in general. Since the best known algorithm for checking whether an FDFA is saturated is in PSPACE [2], one cannot build, based on the currently available results, a polynomial time passive learner for saturated FDFAs based on the active FDFA learner from [3].

Finally, there is the active learning algorithm for deterministic parity automata presented in [29]. This algorithm does not purely use membership and equivalence queries, but additionally so called loop-index queries, which given an ω -word w return the number of symbols after which the run of the target automaton on w enters the looping part. In order to answer such queries, one needs knowledge about the target automaton (and not just the language). So this active learner cannot be used to directly obtain a passive learner through simulation.

2 Preliminaries

We use standard definitions and terminology from the theory of finite automata and ω -automata, and assume some familiarity with these concepts (see, e.g., [33, 36, 38] for some background). An alphabet Σ is a non-empty, finite set of symbols. We use the standard notations Σ^* , Σ^ω for the sets of all finite words and all ω -words, respectively, and let $\Sigma^+ := \Sigma^* \setminus \{\varepsilon\}$, where ε is the empty word, and $\Sigma^\infty := \Sigma^* \cup \Sigma^\omega$. For $u \in \Sigma^*$ we write $|u|$ for its length. A word $u \in \Sigma^*$ is called *prefix* of $v \in \Sigma^\infty$ if $ux = v$ for some $x \in \Sigma^\infty$ and we write $u \sqsubseteq v$ in that case. For a word $w \in \Sigma^\infty$, we use $\text{Prf}(w)$ to refer to the set of all prefixes of w , and for $X \subseteq \Sigma^\infty$, we write $\text{Prf}(X)$ for the union of all $\text{Prf}(x)$ for $x \in X$. For $u \in \Sigma^*$ and $X \subseteq \Sigma^\infty$, we let $u^{-1}X := \{w \in \Sigma^\infty \mid uw \in X\}$. We use the *length-lexicographic (llex) ordering* on finite words that is based on some underlying ordering of the alphabet, and first compares words by length, and words of same length in the lexicographic ordering.

We call $\sim \subseteq \Sigma^* \times \Sigma^*$ a *right congruence (RC)* if \sim is an equivalence relation and $u \sim v$ implies $ua \sim va$ for all $a \in \Sigma$. For a word $u \in \Sigma^*$, we use $[u]_\sim$ (just $[u]$ if \sim is clear from the context) to denote the *class* of u in \sim and write $[\sim]$ for the set of all classes in \sim . We say that \sim_1 *refines* \sim_2 , written as $\sim_1 \leq \sim_2$ if $u \sim_1 v$ implies $u \sim_2 v$. We often use families indexed by classes of an RC. In such cases we also use words as indices representing their class. For example, if we have a family $(\gamma_c)_{c \in [\sim]}$, then we often write γ_u to refer to $\gamma_{[u]_\sim}$.

The index or size, denoted $|\sim|$, of an RC \sim is the number of its classes. In the following, we tacitly assume that all considered RCs are of finite index, and not always explicitly mention this.

A (deterministic) transition system (TS) $\mathcal{T} = (Q, \Sigma, \iota, \delta)$ over the finite alphabet Σ consists of a finite, non-empty set of states Q , a transition function $\delta : Q \times \Sigma \rightarrow Q$ and an initial state $\iota \in Q$. We define $\delta^* : Q \times \Sigma^* \rightarrow Q$ inductively by $\delta^*(q, \varepsilon) = q$ and $\delta^*(q, aw) = \delta^*(\delta(q, a), w)$, and write $\delta^*(u)$ or $\mathcal{T}(u)$ for $\delta^*(\iota, u)$. The *run* of \mathcal{T} on $w = a_0a_1 \dots \in \Sigma^\omega$ from $q_0 \in Q$ is a (possibly infinite) sequence $q_0q_1 \dots$ with $q_{i+1} = \delta(q_i, a_i)$. We write $\text{Inf}_{\mathcal{T}}(w)$ for the set of states that occur infinitely often in the run of \mathcal{T} on $w \in \Sigma^\omega$, and $\text{Inf}_{\mathcal{T}}(X)$ for the union of $\text{Inf}_{\mathcal{T}}(w)$ for $w \in X$. A *strongly connected component (SCC)* of \mathcal{T} is a maximal strongly connected set (with the usual definition). We write $\text{SCC}_{\mathcal{T}}(q)$ for the SCC of \mathcal{T} that contains q . A *partial TS* is a TS in which δ is a partial function.

An RC \sim induces a TS $\mathcal{T}_\sim = ([\sim], \Sigma, [\varepsilon], \delta_\sim)$, where $\delta_\sim(c, a) = [ca]$ with $[ca]$ being the class of ua for an arbitrary $u \in c$. Vice versa, a transition system \mathcal{T} gives rise to the congruence $\sim_{\mathcal{T}}$, where $u \sim_{\mathcal{T}} v$ iff $\mathcal{T}(u) = \mathcal{T}(v)$. So we interchange these objects and often just speak of the TS \sim , meaning the TS \mathcal{T}_\sim .

For an RC \sim and a class $c \in [\sim]$, we define $E_c^\sim = \{x \in \Sigma^+ \mid cx \sim c\}$ as the set of non-empty words that loop on c . Often, we omit the superscript \sim if it is clear from the context.

A *language* (over Σ) is a subset of Σ^* . A *deterministic finite automaton (DFA)* \mathcal{D} consists of a transition system and a subset $F \subseteq Q$ of final or accepting states. The accepted language is $L(\mathcal{D})$ with the standard definition, i.e. the set of all words on which \mathcal{D} has a run from its initial state to some final state. An ω -*language* is a set $L \subseteq \Sigma^\omega$ of ω -words. There are different automaton models for defining the class of ω -regular languages (see [33, 36, 38]). We are interested in (transition-based) deterministic parity automata defined below.

A *priority mapping* is a function $\pi : \Sigma^+ \rightarrow \{0, \dots, k-1\}$ for some $k > 0$. We overload notation and write $\pi(w)$ for $w \in \Sigma^\omega$ to denote the least i such that $\pi(x) = i$ for infinitely many prefixes x of w . The language $L(\pi)$ defined by π is the set of all ω -words w such that $\pi(w)$ is even. We call π *weak* if $\pi(xy) \leq \pi(x)$ for all $x \in \Sigma^+$ and $y \in \Sigma^+$. If \sim is a right congruence and π_c for each c is a weak priority mapping, then $\bar{\pi} = (\pi_c)_{c \in [\sim]}$ is called a *family of weak priority mappings (FWPM)*.

A *deterministic parity automaton (DPA)* is of the form $\mathcal{A} = (Q, \Sigma, \iota, \delta, \kappa)$ with a TS $\mathcal{T}_{\mathcal{A}} = (Q, \Sigma, \iota, \delta)$ and a function $\kappa : Q \times \Sigma \rightarrow \{0, \dots, k-1\}$ mapping transitions in \mathcal{A} to priorities. It induces a priority mapping $\mathcal{A} : \Sigma^+ \rightarrow \{0, \dots, k\}$ by $\mathcal{A}(ua) := \kappa(\delta^*(u), a)$ for $u \in \Sigma^*$ and $a \in \Sigma$. The ω -language $L(\mathcal{A})$ accepted by \mathcal{A} is the language of the induced priority mapping, i.e. the set of all ω -words w such that the least priority seen infinitely often during the run of \mathcal{A} on w is even.

For an automaton \mathcal{A} , we use $\sim_{\mathcal{A}}$ to refer to the RC of its transition system. We call an ω -language regular if it is accepted by a DPA. The Myhill/Nerode congruence \sim_L of a language $L \subseteq \Sigma^\omega$ for $\circ \in \{*, \omega\}$ is the RC defined by $u \sim_L v$ if $\forall w \in \Sigma^\circ : (uw \in L \Leftrightarrow vw \in L)$. If L is regular, then \sim_L is of finite index. For an RC \sim that refines \sim_L and $c \in [\sim]$ we let $L_c := u^{-1}L$, where u is an arbitrary word in c .

For $u \in \Sigma^+$ and $q \in Q$ we use $\mathcal{A}_{\min}(q, u)$ to refer to the minimal priority in the run of \mathcal{A} on u from q . For a right congruence \sim that is refined by $\sim_{\mathcal{A}}$, and a class c of \sim , we call $q \in Q$ a c -state if the words that lead to q in \mathcal{A} from the initial state are in c .

The *parity complexity* of a regular ω -language L , denoted $\text{pc}(L)$, is the least k such that L is accepted by a DPA with priorities $\{0, \dots, k-1\}$ (one may distinguish also whether the smallest priority required is 0 or 1, but we omit this for simplicity). We call a DPA

$\mathcal{A} = (Q, \Sigma, \iota, \delta, \kappa)$ *normalized* if κ is minimal in the following sense: For every $\kappa' : Q \times \Sigma \rightarrow \mathbb{N}$ with $L(\mathcal{T}_{\mathcal{A}}, \kappa') = L(\mathcal{A})$, we have $\kappa(q, a) \leq \kappa'(q, a)$ for all $q \in Q$ and $a \in \Sigma$. This unique minimal priority function on $\mathcal{T}_{\mathcal{A}}$ can be computed in polynomial time [13] (see also [16] for an adaption to transition-based DPAs). We refer to this as *normalization of \mathcal{A}* .

In some proofs in Section 3 we need some basic facts about normalized DPAs that are somehow known, but we cannot give a concrete reference. So we state and prove them in the following lemma. The intuition behind the lemma is the following: If a normalized DPA reads a word u from some state q and visits minimal priority i on the way, then it is possible to complete a loop on q that starts with u and has i as minimal priority (or the run changed the SCC and then $i = 0$). Furthermore, if it is possible to have a loop with priority i on q , then also with priority $i - 1$ (for $i \geq 2$).

► **Lemma 1.** *Let $\mathcal{A} = (Q, \Sigma, \iota, \delta, \kappa)$ be a normalized DPA, $q \in Q$ and $u \in \Sigma^+$ be such that $\mathcal{A}_{\min}(q, u) = i$.*

(a) *Either there exists $v \in \Sigma^*$ with $\delta^*(q, uv) = q$ and $\mathcal{A}_{\min}(q, uv) = i$, or $i = 0$ and $\text{SCC}_{\mathcal{A}}(\delta^*(q, u)) \cap \text{SCC}_{\mathcal{A}}(q) = \emptyset$.*

(b) *If $i \geq 2$, then there is $y \in \Sigma^+$ with $\delta^*(q, y) = q$ and $\mathcal{A}_{\min}(q, y) = i - 1$.*

Proof. For (a), remove from \mathcal{A} all transitions with priority $< i$ and call the resulting partial DPA \mathcal{A}' . Let $q' := \delta^*(q, u)$. Since $\mathcal{A}_{\min}(q, u) = i$, the u -path from q to q' still exists in \mathcal{A}' . If there is a path from q' to q in \mathcal{A}' , then let v be a word labeling such a path. Then $\mathcal{A}_{\min}(q, uv) = i$ and $\delta^*(q, uv) = q$.

If there is no path from q' to q in \mathcal{A}' , let $u = u_1 a u_2$ with $u_1, u_2 \in \Sigma^*$ and $a \in \Sigma$ such that $q_1 := \delta^*(q, u_1)$ is in $\text{SCC}_{\mathcal{A}'}(q)$ and $\delta(q_1, a)$ is not in $\text{SCC}_{\mathcal{A}'}(q)$. Then every run in \mathcal{A} that takes the a -transition from q_1 infinitely often, contains a transition of priority $< i$ infinitely often. So, if $i > 0$, we can change the priority of the a -transition from q_1 to $i - 1$ without changing the language of \mathcal{A} , contradicting the fact that \mathcal{A} is normalized (since the transition exists in \mathcal{A}' , its priority is at least i in \mathcal{A}). Thus, $i = 0$ and $\mathcal{A}' = \mathcal{A}$. This means that q and q' are in different SCCs and thus there is no path back from q' to q .

For (b), remove all transitions of priority $< i - 1$ from \mathcal{A} , obtaining \mathcal{A}'' . By (a), $\text{SCC}_{\mathcal{A}''}(q)$ contains a loop and thus at least one transition. If $\text{SCC}_{\mathcal{A}''}(q)$ contains a transition with priority $i - 1$, we are done (take as y the label of a path in \mathcal{A}'' that starts in q , takes this $(i - 1)$ -transition, and goes back to q).

Otherwise, we show that \mathcal{A} cannot be normalized: Lower in \mathcal{A} the priority of all transitions that are in $\text{SCC}_{\mathcal{A}''}(q)$ by 2. Since $i \geq 2$ and no transition in $\text{SCC}_{\mathcal{A}''}(q)$ has priority $i - 1$, this is possible. Call the resulting DPA \mathcal{B} . Consider the infinity set X of transitions of a run (since \mathcal{A} and \mathcal{B} have the same transition structure, it is the same infinity set in both). If $X \cap \text{SCC}_{\mathcal{A}''}(q) = \emptyset$, the smallest priority in X is the same in \mathcal{A} and \mathcal{B} . If $X \subseteq \text{SCC}_{\mathcal{A}''}(q)$, then the minimal priority in X differs by exactly 2 in \mathcal{A} and \mathcal{B} . Otherwise, X contains a transition that exits $\text{SCC}_{\mathcal{A}''}(q)$, and hence has priority $\leq i - 2$. Since all transitions inside $\text{SCC}_{\mathcal{A}''}(q)$ have priority $\geq i - 2$ in \mathcal{A} and \mathcal{B} , the minimal priority in X is the same in \mathcal{A} and \mathcal{B} . So in all cases, X is accepting in \mathcal{A} iff it is in \mathcal{B} , contradicting the fact that \mathcal{A} is normalized. ◀

Deterministic parity automata can also be viewed as Mealy machines (with the priorities as output alphabet). Given a DPA \mathcal{A} , there is thus a unique minimal DPA that induces the same priority mapping (when minimizing \mathcal{A} as Mealy machine). Active learning algorithms for DFAs (as described in the introduction) can be adapted to Mealy machines. In Section 5 we use the fact that there are polynomial time active learning algorithms for Mealy machines

that produce growing sequences of hypotheses (so their hypotheses are always bounded by the target Mealy machine). See [31] for a detailed introduction to Mealy machines, their minimization and active learning.

For $v \in \Sigma^+$ we write v^ω for the *periodic* ω -word $vvv\cdots$, and call an ω -word *ultimately periodic* if it is of the form uv^ω for $u \in \Sigma^*$ and $v \in \Sigma^+$. It follows from [11] that two regular ω -languages are the same if they contain the same ultimately periodic words, see also [12]. We use ω -samples of the form $S = (S_+, S_-)$, where S_σ for $\sigma \in \{+, -\}$ is a finite set of ultimately periodic words. For simplicity, we do not explicitly distinguish ultimately periodic words and their representations, and $uv^\omega \in S_\sigma$ means that some representation of that word is in S_σ . We sometimes write $uv^\omega \in S$ for $uv^\omega \in S_+ \cup S_-$. A sample S is consistent with L if $L \cap S_- = \emptyset$ and $S_+ \subseteq L$.

A *passive learner* (for DPAs) is a function f that maps ω -samples to DPAs. f is called a polynomial-time learner if f can be computed in polynomial time. A learner f is *consistent* if it constructs from each ω -sample $S = (S_+, S_-)$ a DPA \mathcal{A} such that $L(\mathcal{A})$ is consistent with S . We say that f can *learn every regular ω -language in the limit* if for each such language L there is a *characteristic sample* S^L such that $L(f(S^L)) = L$ and $f(S^L) = f(S)$ is the same DPA for all samples S that are consistent with L and contain S^L . For a class \mathcal{C} of regular ω -languages we say that f can learn every language in \mathcal{C} in the limit *from polynomial data* if the characteristic samples for the languages in \mathcal{C} are of polynomial size (in a smallest DPA for the corresponding language).

3 Precise DPA of a language

In this section we introduce the precise DPA for a regular ω -language L (Definition 17). The definition we give is parameterized by a right congruence \sim of finite index that refines \sim_L , meaning there is a precise DPA for L and each such \sim . The precise DPA of an ω -language L is then the one where \sim is the same as \sim_L . For defining the precise DPA, we first introduce families of weak priority mappings that capture the periodic part of the language L wrt. \sim (up to Lemma 12). Then we show how to combine them into a single priority mapping (Definition 13, Lemma 15), and how to build a DPA for this combined priority mapping (Lemma 16). We finish the section with an observation on precise DPAs, Lemma 20, which later on plays a crucial role in achieving polynomial runtime of our learner.

Let L be a regular ω -language and \sim be a right congruence that refines \sim_L . The following definition of the sets $P_{c,i}^\sim$, is the basis for the priority assignment computed by the precise DPA for L and \sim . Consider a finite word u that is in class c of \sim . We want to assign a priority to each finite word v , assuming that the input word is of the form $u(vx)^\omega$ for some x such that vx loops on the class c , that is $u \sim uvx$ or equivalently $x \in v^{-1}E_c$. If these words are all in L (for each such x), then v is in the set $P_{c,0}^\sim$, meaning that we can assign priority 0 to v in this context of loops on the class c . The definition then proceeds inductively for higher priorities. For $i = 1$, the definition is such that a word v is in $P_{c,1}^\sim$ if for all x as above, $u(vx)^\omega$ is not in L , or $(vx)^\omega$ has a prefix that is in $P_{c,0}^\sim$. The intuition being that a word v that is in $P_{c,1}^\sim$ but not in $P_{c,0}^\sim$ has extensions x such that $u(vx)^\omega$ is not in L (hence the odd priority), and for all extension x such that $u(vx)^\omega$ is in L , there is a prefix of $(vx)^\omega$ that is assigned priority 0. Hence it is “safe” to assign priority 1 to v (in the context of loops on c).

The definition has some similarities with the definition of *natural color of an infinite word* from [16], however we define a priority for each finite word in the context of each class of \sim , whereas [16] defines one priority for each infinite word.

► **Definition 2.** Let L be a regular ω -language and \sim be a right congruence that refines \sim_L . For each class c of \sim , we define

$$P_{c,0}^\sim(L) = \{v \in \Sigma^+ \mid \forall x \in v^{-1}E_c : (vx)^\omega \in L_c\}$$

$$P_{c,i}^\sim(L) = \{v \in \Sigma^+ \mid \forall x \in v^{-1}E_c : ((vx)^\omega \in P_{c,i-1}^\sim(L)\Sigma^\omega \text{ or } ((vx)^\omega \in L_c \text{ iff } i \text{ is even}))\}.$$

► **Example 3.** Let $L \subseteq \{a, b\}^\omega$ consist of all ω -words with finitely many occurrences of b , or infinitely many occurrences of the infix aba . As \sim we take \sim_L , which has only one class (adding or removing finite prefixes is irrelevant for membership in L). Then clearly all finite words that contain the infix aba are in $P_{\varepsilon,0}^\sim(L)$. If a finite word contains b , then its iteration contains infinitely many b , so it is not in L except if its iteration also contains infinitely many aba , which means it has a prefix in $P_{\varepsilon,0}^\sim(L)$. If a word does not contain b , then its iteration is in the language because it contains finitely many b . The languages $P_{\varepsilon,i}^\sim(L)$ for $i \leq 3$ we thus obtain are formally given in the second column of the table in Figure 1 in the last two columns.

i	$P_{\varepsilon,i}^\sim(L)$	$P_{\varepsilon,i}^\sim(L')$	$P_{a,i}^\sim(L')$
0	$\Sigma^*aba\Sigma^*$	$\Sigma^*aa\Sigma^*$	
1	$\Sigma^*b\Sigma^*$	$\Sigma^*a\Sigma^*$	$\Sigma^*(a + b\Sigma^*c + c\Sigma^*b)\Sigma^*$
2	Σ^+	$\Sigma^*(a + b\Sigma^*c + c\Sigma^*b)\Sigma^*$	Σ^+
3	Σ^+	Σ^+	Σ^+

■ **Figure 1** An overview over languages arising from Definition 2 for the languages L and L' from Example 3 and Example 4, respectively.

► **Example 4.** Let $\Sigma = \{a, b, c\}$ and consider the language

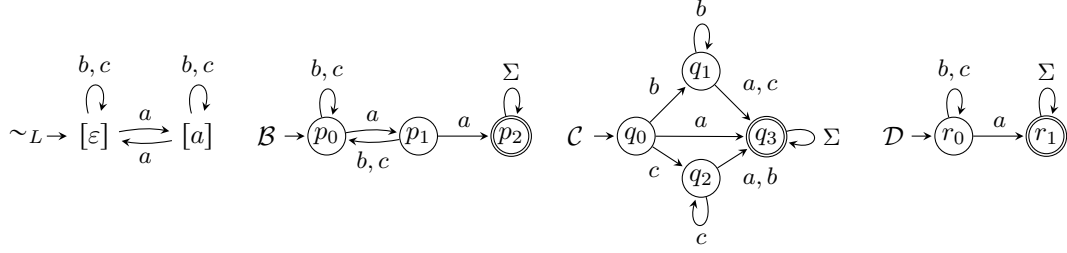
$$L' = \{w \in \Sigma^\omega \mid \text{there are infinitely many infixes } aa \text{ in } w\}$$

$$\cup \{w \in \Sigma^\omega \mid |w|_a \text{ is finite and (even iff } w \text{ contains infinitely many } b \text{ and } c)\}.$$

As \sim , we use $\sim_{L'}$, which has two classes, $[\varepsilon]$ and $[a]$, which are reached when reading words containing an even respectively odd number of a . A depiction of \sim can be found in Figure 2 alongside DFAs recognizing the $P_{c,i}^\sim(L')$ for $i \leq 3$ and $c \in [\sim]$. Regardless of the class $c \in [\sim]$, all words that have an infix aa are in $P_{c,0}^\sim(L')$ and words containing an a are included in $P_{c,1}^\sim(L')$. Consider now a non-empty, finite word u containing b and c but not a , then $u^\omega \in L'$. As $(ua)^\omega \in (P_{c,1}^\sim(L')\Sigma^\omega \setminus L')$ for $c \in [\sim]$, it follows that $u \in P_{\varepsilon,2}^\sim(L')$. On the other hand $a \cdot (ux)^\omega \notin L'$ whenever x does not contain an infix aa , we have $u \in P_{a,1}^\sim(L')$. An overview of the sets $P_{c,i}^\sim(L')$ is shown in Figure 1.

Note that all sets $P_{c,i}$ arising from the given examples are regular. In Example 3 one can directly see that the indices 0, 1, 2 naturally correspond to the priorities that a DPA would use to accept the language (emit priority 0 whenever an infix aba is detected, and priority 1 for every occurrence of b , and priority 2 otherwise). We show that this is not a coincidence by establishing a tight connection between the sets $P_{c,i}^\sim(L)$ and the priorities visited by a normalized DPA \mathcal{A} for L , namely, $P_{c,i}^\sim(L)$ contains precisely those words u that are guaranteed to take a transition of priority at most i from every state in class c (provided that $\sim_{\mathcal{A}}$ refines \sim as otherwise it makes no sense to speak of c -states).

► **Lemma 5.** Let \mathcal{A} be a normalized DPA with priorities $\{0, \dots, k-1\}$ recognizing the language $L \subseteq \Sigma^\omega$ and \sim be a right congruence with $\sim_{\mathcal{A}} \leq \sim \leq \sim_L$. Then for each class c of \sim and each $i \geq 0$ holds $P_{c,i}^\sim(L) = P_{c,i}^\sim(\mathcal{A}) := \{u \in \Sigma^+ \mid \max\{\mathcal{A}_{\min}(q, u) \mid q \text{ is } c\text{-state}\} \leq i\}$.



■ **Figure 2** The leading right congruence underlying the language L from Example 4 is depicted on the far left. We have $L(\mathcal{B}) = P_{0,\varepsilon}^{\sim L}(L) = P_{0,a}^{\sim L}(L)$, $L(\mathcal{C}) = P_{2,\varepsilon}^{\sim L}(L) = P_{1,a}^{\sim L}(L)$ and $L(\mathcal{D}) = P_{1,\varepsilon}^{\sim L}(L)$. We omit the depiction of a DFA for $P_{3,\varepsilon}^{\sim L}(L) = P_{2,a}^{\sim L}(L) = \Sigma^+$.

Proof. For every class c of \sim we show $P_{c,i}^{\sim}(L) = P_{c,i}^{\sim}(\mathcal{A})$ by induction on i . Note that by setting $P_{c,-1}^{\sim}(L) = \emptyset$, we obtain a uniform definition of all $P_{c,i}^{\sim}(L)$ for $i \geq 0$. We can start the induction at -1 . Since \mathcal{A} only uses priorities ≥ 0 , we obtain $P_{c,-1}^{\sim}(\mathcal{A}) = \emptyset$. So let $i \geq 0$ and assume by induction that $P_{c,i-1}^{\sim}(L) = P_{c,i-1}^{\sim}(\mathcal{A})$.

We first show $P_{c,i}^{\sim}(L) \subseteq P_{c,i}^{\sim}(\mathcal{A})$ by establishing that for all $u \in P_{c,i}^{\sim}(L) \setminus P_{c,i-1}^{\sim}(L)$ holds $u \in P_{c,i}^{\sim}(\mathcal{A})$. Let q be a c -state, and let $i' := \mathcal{A}_{\min}(q, u)$. We have to show that $i' \leq i$, so assume towards a contradiction that $i' > i$. Let x be such that $\mathcal{A} : q \xrightarrow{ux} q$ with minimal priority i' on this loop (according to Lemma 1). Then no prefix of $(ux)^\omega$ is in $P_{c,i}^{\sim}(\mathcal{A})$ because the minimal priority seen from q on any prefix of $(ux)^\omega$ is at least $i' > i$. Since $P_{c,i}^{\sim}(\mathcal{A}) \supseteq P_{c,i-1}^{\sim}(\mathcal{A})$, also no prefix of $(ux)^\omega$ is in $P_{c,i-1}^{\sim}(\mathcal{A})$. By induction, it follows that no prefix of $(ux)^\omega$ is in $P_{c,i-1}^{\sim}(L)$, and thus $(ux)^\omega \in L_c$ iff i is even. Since $(ux)^\omega$ is accepted from the c -state q iff i' is even, we obtain that i is even iff i' is even, and hence $i' - 1 > i$ because $i' > i$.

Let $y \in \Sigma^*$ be such that $\mathcal{A} : q \xrightarrow{y} q$ with minimal priority $i' - 1$ (according to Lemma 1). Then $(uxy)^\omega$ is accepted from q iff $i' - 1$ is even iff i is odd. Since u is in $P_{c,i}^{\sim}(L)$, we obtain that $(uxy)^\omega$ has a prefix in $P_{c,i-1}^{\sim}(L)$. This implies by induction that $(uxy)^\omega$ has a prefix in $P_{c,i-1}^{\sim}(\mathcal{A})$, contradicting the fact that the minimal priority that is seen from q on $(uxy)^\omega$ is $i' - 1 > i$.

To establish the other inclusion $P_{c,i}^{\sim}(\mathcal{A}) \subseteq P_{c,i}^{\sim}(L)$, we show for all $u \in P_{c,i}^{\sim}(\mathcal{A}) \setminus P_{c,i-1}^{\sim}(\mathcal{A})$ that $u \in P_{c,i}^{\sim}(L)$. So let $x \in \Sigma^*$ with $ux \in E_c$, and let $n > |\mathcal{A}|$. Since $u \in P_{c,i}^{\sim}(\mathcal{A})$, also $(ux)^n \in P_{c,i}^{\sim}(\mathcal{A})$.

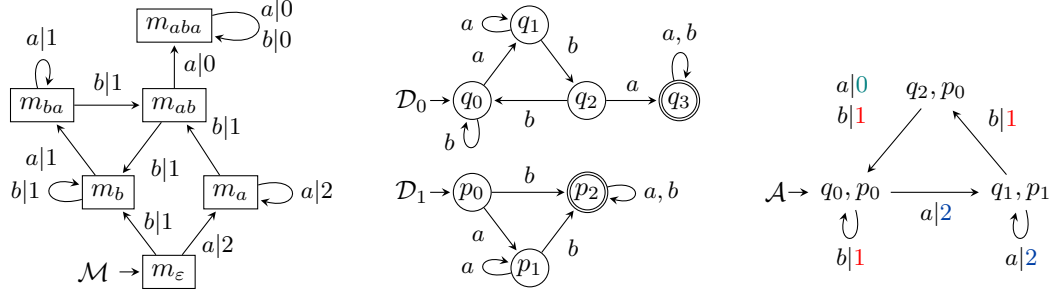
If $(ux)^n \in P_{c,i-1}^{\sim}(\mathcal{A})$, then by induction $(ux)^\omega$ has a prefix in $P_{c,i-1}^{\sim}(L)$.

If $(ux)^n \in P_{c,i}^{\sim}(\mathcal{A}) \setminus P_{c,i-1}^{\sim}(\mathcal{A})$, then let q be a c -state such that $\mathcal{A}_{\min}(q, (ux)^n) = i$. Since $n > |\mathcal{A}|$ and $ux \in E_c$, there is a c -state q' and n_1, n_2 with $n_1 + n_2 \leq n$ such that $\mathcal{A} : q \xrightarrow{(ux)^{n_1}} q' \xrightarrow{(ux)^{n_2}} q'$. Since $u \in P_{c,i}^{\sim}(\mathcal{A})$, the minimal priority on the loop $q' \xrightarrow{(ux)^{n_2}} q'$ is at most i . And by choice of q , the minimal priority on $\mathcal{A} : q \xrightarrow{(ux)^{n_1}} q' \xrightarrow{(ux)^{n_2}} q'$ is i . Hence, the minimal priority on the loop $q' \xrightarrow{(ux)^{n_2}} q'$ is i , and thus $(ux)^\omega \in L_c$ iff i is even.

We conclude that $u \in P_{c,i}^{\sim}(L)$. ◀

As consequence of Lemma 5, if L is of parity complexity k , we obtain an inclusion chain $P_{c,0}^{\sim}(L) \subseteq \dots \subseteq P_{c,k-1}^{\sim}(L) = \Sigma^+$ of regular sets, implying that the priority mappings from the next definition are totally defined, weak, and can be computed by Mealy machines.

► **Definition 6.** With the terminology from Definition 2 and L of parity complexity k , we represent the sets $P_{c,i}^{\sim}(L)$ for each class by a priority mapping $\pi_c^{\sim} : \Sigma^+ \rightarrow \{0, \dots, k-1\}$ defined by $\pi_c^{\sim}(u) = i$ for the minimal i such that $u \in P_{c,i}^{\sim}(L)$. We write $\overline{\pi}^{\sim} = (\pi_c^{\sim})_{c \in [\sim]}$ for the family of these priority mappings, and refer to it as the precise FWPM for L and \sim .



■ **Figure 3** On the left-hand side a Mealy machine \mathcal{M} that computes π_ε^\sim for L and \sim from Example 3 is shown. The DFAs \mathcal{D}_i for $i \in \{0, 1\}$ in the middle accept precisely the words that are assigned priority $\leq i$ by \mathcal{M} . On the right-hand side the precise DPA that is obtained from the \mathcal{D}_i is depicted.

With L and \sim as in Example 3, we obtain that $\pi_\varepsilon^\sim(u) = 0$ if u contains aba , $\pi_\varepsilon^\sim(u) = 1$ if u contains b but not aba , and $\pi_\varepsilon^\sim(u) = 2$ otherwise. The left-hand side of Figure 3 shows a Mealy machine that computes $\pi_\varepsilon^\sim(u)$. We give a bound on the size of Mealy machines for computing the precise FWPM, which can be derived from Lemma 5.

► **Theorem 7.** *Let \mathcal{A} be a normalized DPA with k priorities that recognizes L , and \sim be a right congruence with $\sim_{\mathcal{A}} \leq \sim \leq \sim_L$. Then for each class c of \sim , the mapping π_c^\sim can be computed by a Mealy machine \mathcal{M}_c with at most $m(dk)^d$ states, where m is the number of classes of \sim , and each class of \sim -equivalent states has at most d many states in \mathcal{A} .*

Proof. According to Lemma 5, a Mealy machine only needs to keep track of the minimal priority that is visited from each c -state of \mathcal{A} (and it then outputs the maximum of those numbers).

Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, \kappa)$ and Q_c be the set of c -states in \mathcal{A} . The states of \mathcal{M}_c are pairs (τ, μ) of functions $\tau : Q_c \rightarrow Q$ and $\mu : Q_c \rightarrow \{0, \dots, k-1\}$, where $\tau(q_1) \sim \tau(q_2)$ for all $q_1, q_2 \in Q_c$. There are $m(dk)^d$ such pairs of functions (not all of which need to be reachable in \mathcal{M}).

The initial state is (τ_0, μ_0) with $\tau_0(q) = q$ and $\mu_0(q) = k-1$ for all $q \in Q_c$. For (τ, μ) and an input $a \in \Sigma$, let the a -successor of (τ, μ) be (τ', μ') with $\tau'(q) = \delta(\tau(q), a)$ and $\mu'(q) = \min(\mu(q), \kappa(\tau(q), a))$. The output of \mathcal{M}_c on this transition is $\max_{q \in Q_c} (\mu'(q))$.

Then \mathcal{M}_c outputs on u the least i such that $u \in P_{c,i}(\mathcal{A})$, which corresponds to $\pi_c^\sim(u)$ according to Lemma 5 and Definition 6. ◀

To justify the name “precise” as introduced in Definition 6, we show that $\overline{\pi^\sim}$ is the least weak priority mapping that describes the loops on the classes of \sim wrt. the language L .

► **Definition 8.** *We say (with the terminology from Definition 2) that an FWPM $(\gamma_c)_{c \in [\sim]}$ captures the periodic part of (L, \sim) , if for all classes $c \in [\sim]$ and $v \in E_c$ holds that $v^\omega \in L_c$ iff $v^\omega \in L(\gamma_c)$. Further, we partially order FWPMs by point-wise comparison, that is, for two FWPMs $\bar{\gamma} = (\gamma_c)_{c \in [\sim]}$ and $\bar{\gamma}' = (\gamma'_c)_{c \in [\sim]}$ we let $\bar{\gamma} \leq \bar{\gamma}'$ if for all classes c of \sim and $u \in \Sigma^+$ holds $\gamma_c(u) \leq \gamma'_c(u)$.*

► **Lemma 9.** *Let L be a regular ω -language and \sim refine \sim_L . The family $\overline{\pi^\sim} = (\pi_c^\sim)_{c \in [\sim]}$ captures the periodic part of (L, \sim) . Furthermore, it is the unique least such FWPM.*

Proof. We first show that $\overline{\pi^\sim}$ captures the periodic part of (L, \sim) . Let c be a class of \sim , $v \in E_c$ and set $i := \pi_c^\sim(v^\omega)$. We show that i is even if and only if $v^\omega \in L_c$. Pick an n such

that $\pi_c^\sim(v^n x) = i$ for all $x \sqsubseteq v^\omega$. This is possible since π_c^\sim is weak. By definition, $\pi_c^\sim(v^n) = i$ implies that either $(v^n)^\omega$ has a prefix in $P_{c,i-1}^\sim(L)$ or $((v^n)^\omega \in L_c$ iff i is even). In the former case, there would exist some $x \sqsubseteq v^\omega$ with $\pi_c^\sim(v^n x) < i$, contradicting our choice of i . Hence, the second case must be true, which is what we set out to show.

We now show that $\overline{\pi^\sim}$ is the least such FWPM. The proof makes use of Lemma 10 stated below, whose proof uses the fact that $\overline{\pi^\sim}$ captures the periodic part of (L, \sim) , which we have already shown.

Let $\bar{\gamma}$ be an FWPM which captures the periodic part of (L, \sim) . We show $\overline{\pi^\sim} \leq \bar{\gamma}$. Let $v \in \Sigma^+$ be an arbitrary word with $\gamma_c(v) = i$. We show by induction on i that $\pi_c^\sim(v) \leq i$. For the base, let $i = 0$. Then $\gamma_c((vx)^\omega) = 0$ for all $x \in v^{-1}E_c$ since γ_c is weak. This means for all $x \in \Sigma^*$, if $vx \in E_c$, then $(vx)^\omega \in L_c$ since $\bar{\gamma}$ captures the periodic part of (L, \sim) . Thus, $v \in P_{c,0}^\sim$ and therefore $\pi_c^\sim(v) = 0$.

For an $i > 0$, we assume towards a contradiction that $\pi_c^\sim(v) = j > i$, implying $j \geq 2$. By Lemma 10, there exists some $x \in \Sigma^*$ such that $vx \in E_c$ and $\pi_c^\sim((vx)^\omega) = j$. Further, it must be that $\gamma_c((vx)^\omega) = i$, as otherwise if $\gamma_c((vx)^\omega) < i$, it would follow by induction that $\pi_c^\sim((vx)^\omega) < i$. This means j is even iff i is even and thus $j - 1 > i$.

Using Lemma 10 once more, we pick some $y \in \Sigma^*$ with $vxy \in E_c$ and $\pi_c^\sim((vxy)^\omega) = j - 1$. Again, by induction $\gamma_c((vxy)^\omega) = i$ and hence $(vxy)^\omega \in L(\gamma_c)$ iff $(vxy)^\omega \notin L(\pi_c^\sim)$. This contradicts our assumption that both $\bar{\gamma}$ and $\overline{\pi^\sim}$ capture the periodic part of (L, \sim) . Hence, $\pi_c^\sim(v) \leq i$, concluding the proof. \blacktriangleleft

► **Lemma 10.** *Let $\overline{\pi^\sim}$ be as in Definition 6 and $v \in \Sigma^+$. If $\pi_c^\sim(v) = i$ then there exists an $x \in \Sigma^*$ such that $vx \in E_c$ and $\pi_c^\sim((vx)^\omega) = i$. Moreover, if $i \geq 2$, then there exists $y \in \Sigma^*$ with $vy \in E_c$ and $\pi_c^\sim((vy)^\omega) = i - 1$.*

Proof. We show this claim by induction on i . To simplify the argument, we define $P_{c,-1}^\sim(L) := \emptyset$. The base for $i = 0$ is trivial since π_c^\sim is weak and thus $\pi_c^\sim(vx) = 0$ for all $x \in \Sigma^*$, which directly implies the claim.

Let $i > 0$. Since $v \notin P_{c,i-1}^\sim(L)$, there exists an $x \in \Sigma^*$ such that $vx \in E_c$, $((vx)^\omega \in L_c$ iff $i - 1$ is odd), and $\text{Prf}((vx)^\omega) \cap P_{c,i-2}^\sim(L) = \emptyset$ (this corresponds to the negation of the condition for membership in $P_{c,i-1}^\sim(L)$). Further, $(vx)^\omega$ also has no prefix in $P_{c,i-1}^\sim(L)$. Otherwise, $\pi_c^\sim((vx)^\omega) = i - 1$, contradicting the fact that by Lemma 9, $\overline{\pi^\sim}$ captures the periodic part of L . Thus, it must be that $\pi_c^\sim((vx)^\omega) = i$.

For the second part, assume towards a contradiction that $i \geq 2$ and $\pi_c^\sim((vy)^\omega) \neq i - 1$ for all $y \in \Sigma^*$ with $vy \in E_c$. Then for each such y , either $\pi_c^\sim((vy)^\omega) \in \{i, i - 2\}$ or $\pi_c^\sim((vy)^\omega) < i - 2$. The former implies $(vy)^\omega \in L_c$ iff $i - 2$ is even, while the latter means that $(vy)^\omega$ has some prefix in $P_{c,i-3}^\sim(L)$. In either case, $v \in P_{c,i-2}^\sim(L)$, contradicting $\pi_c^\sim(v) = i$. \blacktriangleleft

The fact that $\overline{\pi^\sim}$ is the least FWPM that captures the periodic part of (L, \sim) somehow means that each $\pi_c^\sim(u)$ gives the most precise information wrt. L for words that loop on c and start with u . Hence, the name “precise FWPM”. The precise DPA for L and \sim that we introduce below is derived from this precise FWPM for L and \sim . In general, for the construction of a DPA from an FWPM that captures the periodic part of (L, \sim) , it is important that the individual priority mappings are compatible with each other. Specifically, adding a prefix to a word should only lead to a smaller (i.e. more significant) priority. This is expressed formally in the following definition.

► **Definition 11.** *We call an FWPM $\bar{\gamma} = (\gamma_c)_{c \in [\sim]}$ monotonic if $\gamma_u(vx) \leq \gamma_{uv}(x)$ for all $u, v \in \Sigma^*, x \in \Sigma^+$.*

► **Lemma 12.** *Let L be a regular ω -language, and \sim refine \sim_L . The precise FWPM $\overline{\pi^\sim}$ of (L, \sim) is monotonic.*

Proof. Let $u, v \in \Sigma^*$, $x \in \Sigma^+$, and let $c = [u]_\sim$ and $c' = [uv]_\sim$. We use Lemma 5, so let $\mathcal{A} = (Q, \Sigma, q_0, \delta, \kappa)$ be a normalized DPA for L . Denote by Q_c and $Q_{c'}$ the sets of c -states and c' -states in \mathcal{A} , respectively. By Lemma 5, $\pi_u^\sim(vx) = \max\{\mathcal{A}_{\min}(q, vx) \mid q \in Q_c\}$. Further, for each $q \in Q_c$ we have $\mathcal{A}_{\min}(q, vx) \leq \mathcal{A}_{\min}(\delta^*(q, v), x)$ because $\mathcal{A}_{\min}(\delta^*(q, v), x)$ takes the minimal priority of a run that is the suffix of the run considered in $\mathcal{A}_{\min}(q, vx)$. We conclude that

$$\begin{aligned} \pi_u^\sim(vx) &= \max\{\mathcal{A}_{\min}(q, vx) \mid q \in Q_c\} \\ &\leq \max\{\mathcal{A}_{\min}(\delta^*(q, v), x) \mid q \in Q_c\} \\ &\leq \max\{\mathcal{A}_{\min}(q', x) \mid q' \in Q_{c'}\} \\ &= \pi_{uv}^\sim(x) \end{aligned}$$

where the second inequality follows from the fact that $\delta^*(q, v)$ is a c' -state for a c -state q . ◀

The Precise DPA. Our goal is now to construct, from a monotonic FWPM $\overline{\gamma} = (\gamma_c)_{c \in [\sim]}$ that captures the periodic part of (L, \sim) , a combined priority mapping γ_{\bowtie} that defines L . We refer to γ_{\bowtie} as the *join* of $\overline{\gamma}$.

For an ultimately periodic word xy^ω with $x \sim xy$, we know that $\gamma_x(y^\omega)$ is an even priority iff $xy^\omega \in L$ (from the fact that $\overline{\gamma}$ captures the periodic part of (L, \sim)). The problem is that γ_{\bowtie} does not “know” when the periodic part of an ultimately periodic word starts. So the priority $\gamma_{\bowtie}(u)$ for a finite word u should take into account all possible prefixes of u as potential starting points of the periodic part. In other words for every possible division $u = xy$, the value $\gamma_x(y)$ has to be taken into account. We capture this with the following definitions (where we refer to partial priority mappings because we are going to define γ_{\bowtie} inductively, and need to refer to properties of the partially defined mapping). While the definition is quite natural, it is a bit technical to write, so we encourage the reader to look at the example after the definition.

► **Definition 13.** *Let $u \in \Sigma^+$ and γ be a partial priority mapping that is defined for all non-empty prefixes of u . Let y be a non-empty suffix of u , that is $u = xy$ for an $x \in \Sigma^*$. We say that γ covers $\overline{\gamma}$ on the suffix y of u if $\gamma(xz) \leq \gamma_x(y)$ for some non-empty prefix z of y .*

The priority mapping γ_{\bowtie} is defined inductively as follows. Assume that γ_{\bowtie} has been defined for all non-empty prefixes of $u \in \Sigma^$, and let $a \in \Sigma$. Then $\gamma_{\bowtie}(ua)$ is the maximal value such that γ_{\bowtie} covers $\overline{\gamma}$ on all non-empty suffixes of ua .*

Consider the FWPM $\overline{\pi^\sim}$ for L and \sim from Example 3, where \sim has a single class c , and $\pi_c^\sim(u)$ is 0 if u contains aba , 1 if u contains b but not aba , and 2 otherwise. Now take the word $u = ababa$. The table on the right shows the π_c^\sim values for the different suffixes of u , and the resulting π_{\bowtie}^\sim -values. For finding $\pi_{\bowtie}^\sim(u)$, we have to consider the π_c^\sim -values of all suffixes of u , which corresponds to the last column of the π_c^\sim -values. The top three entries in this column are 0, corresponding to $\pi_c^\sim(ababa)$, $\pi_c^\sim(baba)$, $\pi_c^\sim(aba)$. All of these suffixes are already covered by $\pi_{\bowtie}^\sim(aba) = 0$. Furthermore, $\pi_c^\sim(ba) = 1$ (corresponding to the fourth entry in the last column) is already covered by $\pi_{\bowtie}^\sim(abab) = 1$. Hence, it only remains to cover the suffix a with the value $\pi_c^\sim(a) = 2$. The maximal value for covering this is 2 itself, so we set $\pi_{\bowtie}^\sim(ababa) = 2$.

We first show that γ_{\bowtie} behaves correctly on ultimately periodic words in an auxiliary lemma. From this we can easily deduce that γ_{\bowtie} defines the correct language.

$u :$	a	b	a	b	a
$\pi_c^\sim :$	2	1	0	0	0
		1	1	1	0
			2	1	0
				1	1
					2
$\pi_{\bowtie}^\sim :$	2	1	0	1	2

► **Lemma 14.** *Let $\bar{\gamma} = (\gamma_c)_{c \in [\sim]}$ be a monotonic FWPM, $u \in \Sigma^*$ and $v \in \Sigma^+$. If $u \sim uv$, then $\gamma_{\bowtie}(uv^\omega) = \gamma_u(v^\omega)$.*

Proof. First note that if u' and v' are such that $u'(v')^\omega = uv^\omega$ and $u' \sim u'v'$, then $\gamma_u(v^\omega) = \gamma_{u'}((v')^\omega)$. This is implied by monotonicity of $\bar{\gamma}$ (Definition 11): Assume w.l.o.g. that u is a prefix of u' . Then by monotonicity, we have $\gamma_u(v^\omega) \leq \gamma_{u'}((v')^\omega)$. Now choose n such that u' is a prefix of uv^n . Then again by monotonicity, we have $\gamma_{u'}((v')^\omega) \leq \gamma_{uv^n}(v^\omega) = \gamma_u(v^\omega)$.

This means that we can choose any representation of the given ultimately periodic word for proving the claim.

We now turn to the claim of the lemma. Let $\gamma_{\bowtie}(uv^\omega) = i$, then the least color assigned to infinitely many prefixes of uv^ω is i . Thus, we can rewrite uv^ω such that $\gamma_{\bowtie}(u) = i$ and $\gamma_{\bowtie}(ux) \geq i$ for all non-empty prefixes $x \sqsubset v^\omega$. Further, we assume that u is picked to be the shortest prefix with this property.

Consider any non-empty $x \sqsubseteq v^\omega$ with $\gamma_{\bowtie}(ux) = i$. By definition of γ_{\bowtie} , there is a suffix z of ux that would not be covered if we had $\gamma_{\bowtie}(ux) > i$. Since $\gamma_{\bowtie}(u) = i$, we conclude that z is a suffix of x (otherwise it would be covered by the value of $\gamma_{\bowtie}(u)$). Thus, we can write $x = yz$ with $\gamma_{uy}(z) = i$. Since $\bar{\gamma}$ is monotonic, it follows that $\gamma_u(x) = \gamma_u(yz) \leq \gamma_{uy}(z) = i$. This inequality cannot be strict because otherwise the suffix x of ux would not be covered by γ_{\bowtie} since all γ_{\bowtie} -values after u are $\geq i$. We conclude that $\gamma_u(v^\omega) = i$ as desired. ◀

► **Lemma 15.** *Let L be an ω -regular language and \sim refine \sim_L . If $\bar{\gamma}$ is a monotonic FWPM that captures the periodic part of (L, \sim) , then $L(\gamma_{\bowtie}) = L$. In particular, $L(\pi_{\bowtie}^\sim) = L$.*

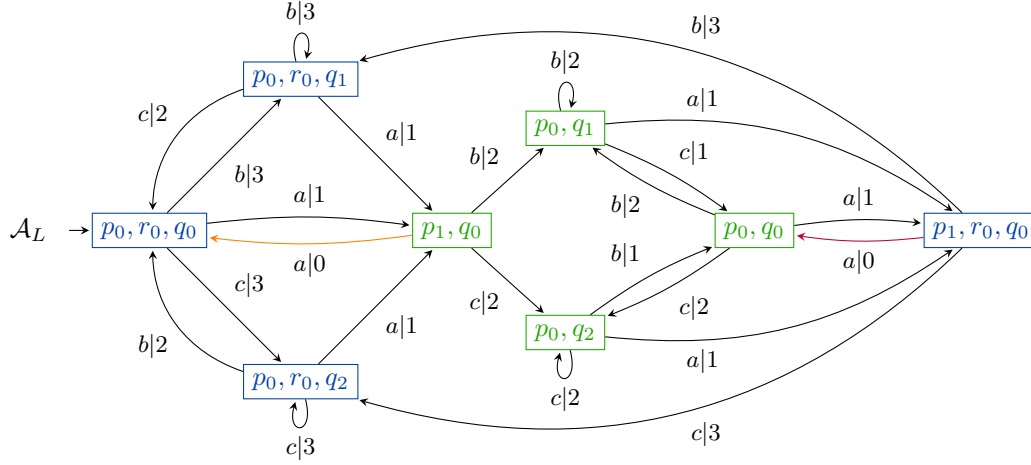
Proof. It follows from [11] that if $K \cap \mathbb{UP} = L \cap \mathbb{UP}$ for two regular ω -languages K and L , then $K = L$. Thus, it suffices to verify that for all $u \in \Sigma^*, v \in \Sigma^+$ holds $uv^\omega \in L$ iff $uv^\omega \in L(\gamma_{\bowtie})$. Consider the sequence of classes $[u], [uv], [uvv], \dots$. Because \sim has only finitely many classes, there must exist $i < j \leq |\sim| + 1$ such that $[uv^i] = [uv^j]$. Thus, we can pick x and y such that $uv^\omega = xy^\omega$ and $xy \sim x$.

By definition, we have $xy^\omega \in L$ iff $y^\omega \in L_x$. Since $y \in E_x$ and $\bar{\gamma}$ captures the periodic part of L , it further holds that $y^\omega \in L_x$ iff $y^\omega \in L(\gamma_x)$. Applying Lemma 14 now gives us $\gamma_{\bowtie}(xy^\omega) = \gamma_x(y^\omega)$, from which we immediately conclude that $xy^\omega \in L(\gamma_{\bowtie})$ iff $xy^\omega \in L$.

The claim $L(\pi_{\bowtie}^\sim) = L$ directly follows from Lemma 9 and Lemma 12. ◀

We now explain how to construct a DPA that computes the priority mapping γ_{\bowtie} if $\bar{\gamma}$ is a monotonic FWPM given by a family $\bar{\mathcal{M}} = (\mathcal{M}_c)_{c \in [\sim]}$ of Mealy machines. This DPA, which we denote by $\mathcal{A}_{\bowtie}(\bar{\mathcal{M}})$, has one component for tracking \sim , and one component for each priority i that checks if there is a suffix of the input read so far that has to be covered and produces priority i . The minimal such i is emitted as priority, and all components for priorities $\geq i$ are reset to start tracking suffixes from this point. Note that the construction can be applied even if $\bar{\gamma}$ is not monotonic (but then we do not have any guarantees on the behavior of the resulting DPA). We formalize this below.

Assume that the range of the mappings in $\bar{\gamma}$ is $\{0, \dots, k-1\}$. First, we extract for each priority $i \in \{0, \dots, k-1\}$ from each Mealy machine \mathcal{M}_c a DFA $\mathcal{D}_{c,i} = (Q_{c,i}, \Sigma, \iota_{c,i}, \delta_{c,i}, F_{c,i})$ with $L(\mathcal{D}_{c,i}) = \{u \in \Sigma^+ \mid \gamma_c(u) \leq i\}$. This is achieved by taking the transition structure of \mathcal{M}_c and redirecting all transitions with output $\leq i$ into an accepting sink state (recall that γ_c is weak), and then minimizing the resulting DFA. This step is illustrated in Figure 3, where the class index is omitted because there is only one class. Note that the DFA $\mathcal{D}_{c,k-1}$ always accepts every non-empty word, so it is omitted in the example. However, it is convenient to keep it in the formal construction.



■ **Figure 4** The precise DPA for the language L from example Example 4. The DPA is built using the right congruence \sim_L and DFAs for the sets $P_{c,i}^{\sim_L}(L)$ for $i \leq 3$ and $c \in [\sim_L]$, which are depicted in Figure 2. The states of \mathcal{A}_L are tuples, where the i -th position from the front is a state belonging to the component for priority i , i.e. a DFA which keeps track of level i of the parity decomposition. We omit a component if the corresponding DFA accepts Σ^+ and also do not include the last component, instead depicting states belonging to class $[\varepsilon]_{\sim_L}$ in blue and those belonging to $[a]_{\sim_L}$ in green. Note that reading a from the initial state, the first component is not reset, as the corresponding DFA still waits on completing an infix aa . The DFA \mathcal{C} from Figure 2 tracks priority 2 from $[\varepsilon]_{\sim_L}$ and priority 1 from $[a]_{\sim_L}$. Hence its states appear in two different positions in the tuples. The DFA \mathcal{B} tracks priority 0 for both classes, so the component for priority 0 is reset to p_0 for both classes on transitions with priority 0 (the orange and purple a -transitions).

To simplify notation, we assume that the $Q_{c,i}$ are pairwise disjoint, allowing us to write $\delta(q, a)$ and $q \in F$ to refer to $\delta_{c,i}(q, a)$ resp. $q \in F_{c,i}$ for the unique $c \in [\sim]$ and $i < k$ such that $q \in Q_{c,i}$. We now define the DPA $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}}) = (Q_{\bowtie}, \Sigma, \iota_{\bowtie}, \delta_{\bowtie}, \kappa_{\bowtie})$ with

$$Q_{\bowtie} = Q_0 \times \dots \times Q_{k-1} \times Q_{\sim}, \text{ where } Q_i = \bigcup_{c \in [\sim]} (Q_{c,i} \setminus F_{c,i}) \text{ and } \iota_{\bowtie} = (\iota_{[\varepsilon],0}, \dots, \iota_{[\varepsilon],k-1}, [\varepsilon]).$$

In this product, we refer to the i th component as the *component for priority i* for $i \in \{0, \dots, k-1\}$. For each transition from some state $\tilde{q} = (q_0, \dots, q_{k-1}, c) \in Q$ on a symbol $a \in \Sigma$, we begin by computing the reset point $r < k$, which corresponds to the least index of a DFA that reaches a final state, meaning formally $r = \min\{i < k \mid \delta(q_i, a) \in F\}$. Note that r is always defined because the DFA for priority $k-1$ accepts everything.

The reset point directly determines the priority of the transition, meaning $\kappa_{\bowtie}(\tilde{q}, a) = r$. For computing the target of the transition, we first advance each q_j for $j < r$ by a in the respective DFA and then reset all other states to the initial state of the appropriate DFA, i.e.

$$\delta_{\bowtie}((q_0, \dots, q_{k-1}, c), a) = (\delta(q_0, a), \dots, \delta(q_{r-1}, a), \iota_{c',r}, \dots, \iota_{c',k-1}, c') \text{ where } c' = [ca]_{\sim}.$$

In Figure 3 the resulting DPA for Example 3 is shown on the right-hand side. We omit the component for the \sim -class and the component for priority 2, because they both only consists of one state. The precise DPA for Example 4 is shown in Figure 4.

The following lemma establishes that $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$ indeed computes the join.

► **Lemma 16.** *If $\overline{\mathcal{M}}$ is a family of Mealy machines representing a monotonic FWPM $\overline{\gamma}$, then $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})(u) = \gamma_{\bowtie}(u)$ for each $u \in \Sigma^+$. In particular, $L(\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})) = L(\gamma_{\bowtie})$.*

Proof. We write \mathcal{A}_{\bowtie} for $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$. Further, for avoiding case distinctions, we let $\gamma_{\bowtie}(\varepsilon) := \mathcal{A}_{\bowtie}(\varepsilon) := 0$. Note that this has no influence on the claim of the lemma because the claim is only on non-empty words.

We first prove the following characterization of γ_{\bowtie} , and then show that this is precisely what $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$ computes:

Claim: For $u \in \Sigma^+$, we have that $\gamma_{\bowtie}(u)$ is the minimal i such that there are $x \in \Sigma^*$, $y \in \Sigma^+$ with $u = xy$ and

- (1) $\gamma_x(y) = i$,
- (2) $\forall z \in \Sigma^+ : z \subsetneq y \Rightarrow \gamma_{\bowtie}(xz) > i$, and
- (3) $\gamma_{\bowtie}(x) \leq i$.

(Where $z \subsetneq y$ means that z is a prefix of y but not equal to y .)

From the definition of γ_{\bowtie} it follows that $\gamma_{\bowtie}(u)$ is the minimal i for which $u = xy$ with properties (1) and (2) exist, because these are precisely the suffixes of u that need to be covered by the value $\gamma_{\bowtie}(u)$. So let i be minimal such that there are $x \in \Sigma^*$, $y \in \Sigma^+$ with $u = xy$, $\gamma_x(y) = i$, and $\gamma_{\bowtie}(xz) > i$ for all strict non-empty prefixes of y . Furthermore, choose x and y such that y has maximal length with these properties. We now show that (3) is also satisfied, thus proving the claim.

Assume for contradiction that $\gamma_{\bowtie}(x) = j > i$. Then x is non-empty because we defined $\gamma_{\bowtie}(\varepsilon) = 0$. Let $x = x'a$ with $a \in \Sigma$. Since $\bar{\gamma}$ is monotonic, $i' := \gamma_{x'}(ay) \leq \gamma_x(y) = i$. Now let z be a nonempty strict prefix of ay . If $z = a$, then $\gamma_{\bowtie}(x'z) = \gamma_{\bowtie}(x) = j > i$. If $z = az'$ with nonempty z' , then $\gamma_{\bowtie}(x'z) = \gamma_{\bowtie}(xz') > i$ because x, y and i satisfy property (2). Overall, we obtain that x', ay and i' satisfy properties (1) and (2). If $i' < i$, this contradicts the choice of i as the minimal such value, and if $i' = i$, this contradicts the choice of y as longest suffix of u such that x, y and i have properties (1) and (2).

Now let us show that the priority of \mathcal{A}_{\bowtie} after reading u is the one characterized in the claim. We assume inductively that \mathcal{A}_{\bowtie} has produced the same priorities as γ_{\bowtie} on all strict non-empty prefixes of u . Let $\mathcal{A}_{\bowtie}(u) =: i$. Then the component for priority i in \mathcal{A}_{\bowtie} has reached a final state of the respective DFA that is currently running in that component, and none of the DFAs in the components of smaller priorities has reached a final state. Let x be the longest strict prefix of u such that $\mathcal{A}_{\bowtie}(x) \leq i$ (by our convention $\mathcal{A}_{\bowtie}(\varepsilon) = 0$, such an x always exists), and let $y \in \Sigma^+$ be such that $u = xy$.

By definition of \mathcal{A}_{\bowtie} , the component for priority i is reset to the initial state of $\mathcal{D}_{x,i}$ after reading x , and since x is the longest prefix on which priority $\leq i$ is produced, none of the components for priorities $\leq i$ is reset after x . We had seen above that the component for priority i in \mathcal{A}_{\bowtie} has reached a final state, which means that $\gamma_x(y) \leq i$ by definition of $\mathcal{D}_{x,i}$. Since the components for priorities smaller than i have not reached a final state, we obtain by monotonicity that $\gamma_x(y) = i$. Hence, x, y and i satisfy the properties (1)–(3) of the claim, and thus $\mathcal{A}_{\bowtie}(u) = i \geq \gamma_{\bowtie}(u)$.

Now assume that there is $j < i$ that satisfies (1)–(3) for some appropriate x and y . Then, along the same lines as above, the component for priority j is reset after that x , and reaches a final state after reading y . This means that \mathcal{A}_{\bowtie} would output a priority $\leq j$. \blacktriangleleft

We know from Theorem 7 that the precise FWPM $\overline{\pi^\sim}$ for (L, \sim) can be computed by a family of Mealy machines. This enables us now to define the central object of this section.

► **Definition 17.** Let L be a regular ω -language, and let \sim be a right congruence that refines \sim_L . The precise DPA $\mathcal{A}_{L, \sim}$ for (L, \sim) is the minimal DPA whose priority mapping is the join π_{\bowtie}^\sim of the precise FWPM $\overline{\pi^\sim}$ for (L, \sim) . The precise DPA \mathcal{A}_L for L is \mathcal{A}_{L, \sim_L} .

The DPA on the right-hand side of Figure 3 is minimal as Mealy machine, so it is the precise DPA for the language from Example 3. Similarly, the DPA in Figure 4 is the precise DPA for the language from Example 4.

Note that $L(\mathcal{A}_{L,\sim}) = L$ because the precise FWPM $\overline{\pi}^\sim$ for L and \sim captures the periodic part of (L, \sim) according to Lemma 9, and hence $L(\pi_{\boxtimes}^\sim) = L$ by Lemma 15. Since, by definition, the priority mapping of $\mathcal{A}_{L,\sim}$ is π_{\boxtimes}^\sim , we obtain that $L(\mathcal{A}_{L,\sim}) = L$. Furthermore, given an arbitrary DPA \mathcal{A} for L , and \sim given as transition system, one can construct $\mathcal{A}_{L,\sim}$. For use in the Section 5, we give an upper bound on the size of the precise DPA that is constructed from another DPA.

► **Theorem 18.** *Let \mathcal{A} be a DPA with priorities in $\{0, \dots, k-1\}$ accepting some language L , and \sim be a right congruence with $\sim_{\mathcal{A}} \leq \sim \leq \sim_L$. If \mathcal{A} has at most d many c -states for each class c of \sim_L , then $|\mathcal{A}_{L,\sim}| \leq m(2^d - 1)^{k-1}$, where m is the number of \sim classes.*

Proof. We can assume that $\mathcal{A} = (Q, \Sigma, \iota, \delta, \kappa)$ is normalized (this does not change the transition structure and does not increase the number of priorities). The precise DPA $\mathcal{A}_{L,\sim}$ can be constructed from the family $\overline{\mathcal{M}}$ of Mealy machines for $\overline{\pi}^\sim$ by building $\mathcal{A}_{\boxtimes}(\overline{\mathcal{M}})$. In the construction of $\mathcal{A}_{\boxtimes}(\overline{\mathcal{M}})$, the first step is to extract the DFAs $\mathcal{D}_{c,i}$ for each class c and each priority i from the Mealy machines in $\overline{\mathcal{M}}$. The DFA $\mathcal{D}_{c,i}$ accepts a word w if $\pi_c^\sim(w) \leq i$. From Lemma 5 we obtain that $\pi_c^\sim(w) \leq i$ if from every c -state q in \mathcal{A} , the run on u has visited a priority $\leq i$. We can directly construct $\mathcal{D}_{c,i}$ from \mathcal{A} without going through the Mealy machines as follows. The states of $\mathcal{D}_{c,i}$ are sets $P \subseteq Q$ such that all states in P are pairwise \sim -equivalent. The initial state is the set Q_c of c -states in \mathcal{A} . For a state P of $\mathcal{D}_{c,i}$, the a -successor of P is $\{\delta(p, a) \mid p \in P \text{ and } \kappa(p, a) > i\}$. Since $\sim_{\mathcal{A}}$ refines \sim , the property that all states in P are \sim -equivalent is preserved by the transitions. The only accepting state of $\mathcal{D}_{c,i}$ is \emptyset .

The states of $\mathcal{A}_{\boxtimes}(\overline{\mathcal{M}})$ are tuples (P_0, \dots, P_{k-1}, c) where each P_i is a non-accepting state of some $\mathcal{D}_{c',i}$, and c is the \sim -class of the input read so far. By construction, all \mathcal{A} -states in P_i are in Q_c . Further, in $\mathcal{D}_{c,k-1}$, every transition from the initial state directly leads to the accepting state. So for each c there are at most $(2^d - 1)^{k-1}$ many states in $\mathcal{A}_{\boxtimes}(\overline{\mathcal{M}})$, which shows the claimed bound. ◀

We now present a family of examples showing that the precise DPA can be exponential in the size of a minimal DPA. This is already witnessed by Büchi automata, so DPAs with priorities $\{0, 1\}$. The example is taken from [9, Proposition 14].

For a number $d \geq 1$, consider the alphabet $\Sigma_d = \{a_0, \dots, a_{d-1}\}$ and let $L_d \subseteq \Sigma_d^\omega$ be the language consisting of all ω -words that contain each symbol from Σ_d infinitely often. One easily verifies that \sim_{L_d} has only one class.

In a first approach for constructing a DPA for L_d , one would build an automaton that tracks precisely which symbols it has seen. This means that the states are subsets of Σ_d , with \emptyset as initial state, and transitions adding the processed symbol to the current state. Whenever a transition would result in a state for the full set Σ_d , the DPA emits priority 0, and resets the state to \emptyset . All other transitions have priority 1.

Indeed, this is what the precise DPA does: The precise FWPM for L_d and \sim_{L_d} (consisting only of one weak priority mapping since there is only one class), maps a finite word to priority 0 if it contains all symbols from Σ_d , and to 1 otherwise. The precise DPA resulting from that precise FWPM then behaves exactly as described above. So it has $2^d - 1$ states.

However, one can build a DPA with only d states that waits for the next symbol from Σ_d for some fixed order on Σ_d , and resets to the initial state with priority 0 if it has reached the last symbol in this order. For the standard order on $\{0, \dots, d-1\}$ and state set $\{q_0, \dots, q_{d-1}\}$,

the resulting transition function is $\delta(q_h, a_h) = a_{h+1 \bmod d}$ with priority 0 if $h = d - 1$ and 1 otherwise, and $\delta(q_h, a_{h'}) = q_h$ if $h \neq h'$.

This family of examples might also convey some further intuition why we chose the name “precise DPA”: It emits the accepting priority 0 precisely when all symbols have occurred. The small DPA with only d states emits priority 0 if all symbols have occurred in a specific order, so it would not emit priority 0 on the finite word $a_2a_1a_0$ (for $d = 3$) but on $a_0a_1a_2$. It seems hard to define such a behavior just from the language definition in a canonical way.

The construction of the precise DPA for L sets out a rough road map for the construction of a DPA from a collection S of positive and negative example words: Extract a right congruence \sim from S as candidate for \sim_L (this is known how to do it). Then extract Mealy machines $\overline{\mathcal{M}}$ that capture the periodic part of S and \sim . Construct $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$.

In Section 4 we present some results that enable us to realize the extraction of Mealy machines. The problem with the last step is that the size of $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$ is of order $|\overline{\mathcal{M}}|^k$, where k is the number of priorities. In a polynomial time procedure we thus cannot simply construct $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$ from the Mealy machines because this would result in an exponential step. To overcome this, we show in the following that the size of the representation of the color sequence produced by $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$ on an ultimately periodic word uv^ω is polynomial in $|uv|$ and $|\overline{\mathcal{M}}|$. This allows us to construct a DPA of polynomial size, which approximates $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$ in the sense that it emits the correct color sequence for a set of ultimately periodic words (the details how it is used are given in Section 5). In the following, we first state an auxiliary lemma.

► **Lemma 19.** *Let $v \in \Sigma^+$. A DFA \mathcal{D} with n states accepts a prefix v^ω from a state q if, and only if, there exists a non-empty $x \in \text{Prf}(v^\omega) \cap L(\mathcal{D})$ with $|x| \leq n \cdot |v|$.*

Proof. Consider the run $q = q_0 \xrightarrow{v} q_1 \xrightarrow{v} q_2 \xrightarrow{v} \dots$ of \mathcal{D} on v^ω . As \mathcal{D} has n states, there must be positions $i < j \leq n$ at which a state repeats, i.e. $q_i = q_j$. Either a final state is visited before position j , or the DFA loops on v^ω without visiting a final state at all. ◀

Note that the FWPM in the following lemma is not required to be monotonic.

► **Lemma 20.** *Let $u \in \Sigma^*$, $v \in \Sigma^+$ and $\overline{\mathcal{M}}$ be a family of mealy machines computing an FWPM $\overline{\gamma}$. The sequence of priorities produced by $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$ on uv^ω can be written as an ultimately periodic word rs^ω where $|rs|$ is polynomial in $|uv|$ and $|\overline{\mathcal{M}}|$. Furthermore, r and s can be computed in polynomial time.*

Proof. To simplify notation, we let $\mathcal{A} := \mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$ and use γ to refer to the priority mapping computed by \mathcal{A} . Further, we assume that $uv \sim u$. Otherwise, we can rewrite the representation of uv^ω accordingly with only a polynomial blow-up of the representation.

We decompose uv^ω into a sequence of non-empty finite factors $x_0, x_1 \dots$ by making the next cut for each x_h right after the first time \mathcal{A} emits the smallest priority i_h it will emit on the remaining suffix w_h . Formally, we define the sequence inductively with $y_0 := \varepsilon$, $w_0 := uv^\omega$, and if y_h, w_h with $y_h w_h = uv^\omega$ are given, we let x_h be the shortest prefix of w_h such that $\gamma(y_h x_h) \leq \gamma(y_h x)$ for all non-empty prefixes x of w_h . Then let $i_h := \gamma(y_h x_h)$, $y_{h+1} := y_h x_h$ and w_{h+1} such that $w_h = x_h w_{h+1}$.

We first explain why i_h, x_h can be computed in polynomial time, and why the length of each x_h is polynomial in $|uv|$ and $|\overline{\mathcal{M}}|$.

By definition, the states of \mathcal{A} are tuples of the form (p_0, \dots, p_{k-1}, c) , where c is a class of \sim and the p_i are states of DFAs whose size is bounded by the size of the Mealy machines in $\overline{\mathcal{M}}$. We already know that \mathcal{A} only emits priorities $\geq i_{h-1}$ on the remaining suffix w_h (letting $i_{-1} = 0$ for $h = 0$). And after the prefix y_h , all components for priorities $j \geq i_{h-1}$ are set to

the initial state of the DFAs for the current \sim -class c_h . To determine i_h , we can check in increasing order starting from $j = i_{h-1}$, whether $\mathcal{D}_{c_h,j}$ accepts a prefix of w_h . By Lemma 19 this can be done in polynomial time, and if there is such a prefix, its length is bounded by $|u| + |v| |\mathcal{D}_{c_h,j}|$.

We now explain why we only need to compute a polynomial number of the factors.

Since each i_h is the smallest priority that appears on the remaining suffix, we have $i_0 \leq i_1 \leq \dots$. So for $h \geq k-1$, all i_h are the same, say i . Since \mathcal{A} resets all components for priorities $\geq i$ after emitting i , we get for all $h, h' \geq k$ with $w_h = w_{h'}$ that $x_h = x_{h'}$. Furthermore, the sequence of priorities emitted on the factors x_h and $x_{h'}$ is also the same because it only depends on the states in the components for priorities $\geq i$, which are reset each time that i is emitted.

Since the w_h are suffixes of uv^ω , such a repetition of a suffix happens after at most $|u| + |v|$ steps. But if $w_h = w_{h'}$ and $x_h = x_{h'}$, then also $w_{h+1} = w_{h'+1}$ and thus the sequence of priorities becomes periodic at this point.

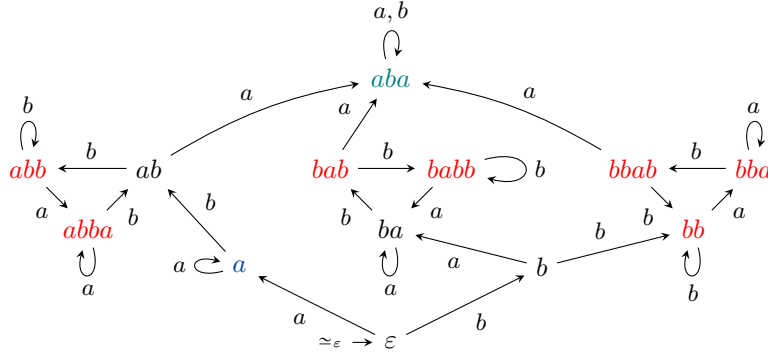
Thus, in order to compute an ultimately periodic representation of the priority sequence of \mathcal{A} on uv^ω , it suffices to compute the sequence of the w_h, y_h, x_h, i_h up to a point h_1 such that there is $h_0 < h_1$ with $w_{h_1} = w_{h_0}$ and $i_{h_1} = i_{h_0}$. The resulting prefix $x_0 x_1 \dots x_{h_1-1}$ is of polynomial length in $|uv|$ and $|\overline{\mathcal{M}}|$ according to the above explanations. Then we let r be the priority sequence of \mathcal{A} on $x_0 \dots x_{h_0-1}$ and s the priority sequence of \mathcal{A} on $x_{h_0} \dots x_{h_1-1}$. These can be computed on the fly without fully constructing \mathcal{A} , so their computation is polynomial in $|uv|$ and $|\overline{\mathcal{M}}|$. \blacktriangleleft

4 Families of Right Congruences

In Section 3 we have seen that we can construct a DPA for L from \sim_L or any right congruence \sim that refines \sim_L , and from Mealy machines for the colorings $\bar{\pi}^\sim = (\pi_c^\sim)_{c \in [\sim]}$. Our goal is to learn such Mealy machines from given examples. For this purpose, we explain in this section how to define the Mealy machines using the formalism of families of right congruences (FORCs) [27, Definition 5]. We start by giving some intuition, why and how we use this formalism, and then go into the formal details.

The usual way to obtain a method that can construct a class of transition systems in the limit from examples, is to characterize them by a right congruence on finite words, such that non-equivalence of two words is witnessed by a finite set of positive and negative examples for L . For example, the non-equivalence of x, y for \sim_L is witnessed by two examples $xuv^\omega \in L \Leftrightarrow yuv^\omega \notin L$. Then an appropriate method can extract the desired transition system if enough non-equivalences are specified by the examples. We did not manage to characterize the transition structure of a minimal Mealy machine for a coloring π_c^\sim in such a way. To illustrate this, consider the case that $\sim = \sim_L$ has only one class c , and that only the priorities 0 and 1 are used. Write P_0 for $P_{c,0}^\sim$ and similarly for P_1 . A minimal Mealy machine for π_c^\sim consists of an initial part that assigns 1 to the words in $P_1(L)$, and a sink state that is reached for all words in $P_0(L)$. So two words x, y lead to different states in this Mealy machine iff there is an extension z with $(xz \in P_0(L) \Leftrightarrow yz \notin P_0(L))$. Substituting the definition of $P_0(L)$, we obtain that x and y lead to a different state iff $\exists z \in \Sigma^* : (\forall z' \in \Sigma^* : (xzz')^\omega \in L) \Leftrightarrow (\exists z' \in \Sigma^* : (yzz')^\omega \notin L)$.

This characterization of non-equivalence has a universal quantifier, and thus cannot be witnessed by a finite set of examples. However, we can use a condition that is implied by the above one, which means that we characterize a possibly larger transition system. If x and y satisfy the above non-equivalence condition, then, in particular, there are $z, z' \in \Sigma^*$ with



■ **Figure 5** The PRC \simeq_ε of the syntactic FORC for the language from Example 3. The colors code the idempotent classes that are positive (aba and a) and negative.

$(xzz')^\omega \in L \Leftrightarrow (yzz')^\omega \notin L$. Rewriting this with a single word z instead of zz' , we get the condition $\exists z \in \Sigma^* : (xz)^\omega \in L \Leftrightarrow (yz)^\omega \notin L$ for non-equivalence of x and y , which is implied by the first condition. Generalizing this to a right congruence \sim with several classes, directly leads to the notion of a canonical FORC for \sim , as defined below.

A *family of right congruences (FORC)* [27] is a tuple $\mathcal{F} = (\sim, (\approx_c)_{c \in [\sim]})$ such that \sim is a right congruence over Σ^* , each \approx_c for $c \in [\sim]$ is a right congruence over Σ^+ , and for all $x, y \in \Sigma^*$ and $c \in [\sim]$ it holds that $x \approx_c y$ implies $ux \sim uy$, for an arbitrary element $u \in c$. Instead of writing \approx_c for a class c , we sometimes also write \approx_u for an arbitrary word $u \in c$. We extend \approx_c from Σ^+ to Σ^* by adding a new class that contains only ε .

Adopting the terminology of [22, 3], we call \sim the *leading right congruence (LRC)* of \mathcal{F} , and each \approx_c the *progress right congruence (PRC)* for c . FORCs can be used as acceptors for ω -languages (see [27, Definition 6], [22, Definition of LFORC], [3, Definition 6]) but since this is not relevant for our results, we do not go into the details of it. We are interested in the *canonical FORC* for $L \subseteq \Sigma^\omega$ and an LRC \sim that refines \sim_L . For $u \in \Sigma^*$, we define the *canonical PRC* \simeq_u for L and \sim by

$$x \simeq_u y \text{ iff } ux \sim uy \text{ and } \forall z \in \Sigma^* : uxz \sim u \Rightarrow (u(xz)^\omega \in L \Leftrightarrow u(yz)^\omega \in L).$$

The *syntactic FORC* of L [27, Definition 9] is the canonical FORC for L and \sim_L . As a running example, consider the language L from Example 3. Then \sim_L has only one class, and the PRC for this class is shown in Figure 5. The transition structure of the Mealy machine in Figure 3 is obtained by merging some of the classes in the PRC.

The syntactic FORC is a canonical object for representing regular ω -languages: In [27, Theorem 22 and Theorem 24] it is shown that $L \subseteq \Sigma^\omega$ is regular if and only if its syntactic FORC is finite, and the syntactic FORC is the coarsest FORC with \sim_L as leading right congruence that recognizes L .

We define the size of a FORC as the sum of the sizes of the LRC and the PRC. Note that if the syntactic FORC of L is finite, then also the canonical PRCs for each \sim that refines L are finite, as a direct consequence of the definition of the canonical FORC for \sim .

We now explain how we can use the PRCs \simeq_c of the canonical FORC for L and \sim in order to obtain a Mealy machine for the priority mappings π_c^\sim . So in the following, let $L \subseteq \Sigma^\omega$ be regular with parity complexity k , let \sim be a refinement of \sim_L , and let \simeq_c be the canonical PRC for each class c of \sim . We call a class $[x]_{\simeq_c}$ with $x \in E_c$ *positive* if $x^\omega \in L_c$, and *negative* otherwise (by the definition of \simeq_c this is independent of the chosen word from the class). In the example from Figure 5, the positive classes are the ones of a , ab , ba , and aba .

We define a coloring κ_c on the classes of \simeq_c (so on the states of the transition system defined by \simeq_c) such that the Mealy machine that outputs the color of its target state on each transition defines π_c^\sim (see Lemma 22). For this coloring, idempotent classes of \simeq_c play a central role. We call a word $x \in \Sigma^+$ *idempotent* in \simeq_c if $x \in E_c$ and $x \simeq_c xx$. The following lemma states two useful properties on idempotent words in \simeq_c .

► **Lemma 21.** *If $x \in \Sigma^+$ is idempotent in \simeq_c , then each $y \in \Sigma^+$ with $y \simeq_c x$ is idempotent in \simeq_c . Furthermore, for each $x \in E_c$, there is a number $n \geq 1$ such that x^n is idempotent in \simeq_c .*

Proof. We begin with the first claim. Let $u \in c$. First note that $y \in E_c$ because $x \in E_c$ and $y \simeq_c x$ implies $ux \sim uy$. For showing $y \simeq_c yy$, consider $z \in \Sigma^+$ with $uyz \sim u$. Then

$$\begin{aligned}
 u(yz)^\omega \in L &\Leftrightarrow u(xz)^\omega \in L && \text{because } y \simeq_c x \\
 &\Leftrightarrow u(xxz)^\omega \in L && \text{as } x \in E_c \text{ is idempotent} \\
 &\Leftrightarrow u(yxz)^\omega \in L && y \simeq_c x \\
 &\Leftrightarrow uy(xzy)^\omega \in L && uy \sim u \text{ and } \sim \text{ refines } \sim_L \\
 &\Leftrightarrow uy(yzy)^\omega \in L && y \simeq_c x \\
 &\Leftrightarrow u(yyz)^\omega \in L && \text{as } uy \sim u \text{ and } \sim \text{ refines } \sim_L.
 \end{aligned}$$

We now show the second claim. Since \simeq_c has finitely many classes, there are $n_1, n_2 \geq 1$ such that $x^{n_1} \simeq_c x^{n_1+n_2}$. Then it is not hard to see that for $n := n_1 n_2$ we have $x^n \simeq_c x^{2n}$. ◀

Accordingly, we call a class of \simeq_c idempotent if one (and hence all) of its members are idempotent. In the example PRC in Figure 5, the only non-idempotent classes are the classes of b , ab and ba .

The algorithm for computing κ_c considers positive and negative idempotent classes (the classification of non-idempotent classes as positive or negative does not play any role). In the example from Figure 5, the positive idempotent classes are the ones of a and aba . The other two positive classes ab and ba are not idempotent.

Now let $\kappa_c : [\simeq_c] \rightarrow \{0, \dots, k\}$ be defined by

- $\kappa_c([x]_{\simeq_c}) = 0$ if there is no z such that $[xz]_{\simeq_c}$ is a negative idempotent class. In other words, if no negative idempotent class is reachable from x .
- Let $i \geq 1$ and assume that κ_c has already been defined for all values $\{0, \dots, i-1\}$. Then $\kappa_c([x]_{\simeq_c}) = i$ for $i \geq 1$ if each idempotent class of the form $[xz]_{\simeq_c}$
 - either already has a κ_c value less than i ,
 - or is positive iff i is even.

In the example from Figure 5, the class of aba is assigned 0. Then all remaining classes of words that contain b are assigned value 1, and finally, the class of a is assigned 2.

The following lemma shows that this indeed gives the desired coloring of Σ^+ . In particular, all classes of \simeq_c are assigned a value by the above procedure.

► **Lemma 22.** *For all $x \in \Sigma^+$: $\kappa_c(x) = i$ iff $\pi_c^\sim(x) = i$.*

Proof. The proof goes by induction on i . For the base case, note that $x \in P_{c,0}^\sim(L)$ iff $(xz)^\omega \in L_c$ for all $z \in \Sigma^*$ with $xz \in E_c$. In particular, this means that all idempotent classes of \simeq_c that are reachable from the class of x , and hence contain a word of the form xz , must be positive. So κ_c assigns 0 to all classes $[x]_{\simeq_c}$ with $\pi_c^\sim(x) = 0$. Vice versa, if no negative idempotent classes are reachable from the class of x , then $(xz)^\omega \in L_c$ for all $z \in \Sigma^*$ with $xz \in E_c$ (if $(xz)^\omega \notin L_c$, then it has an idempotent $(xz)^n$ as prefix by Lemma 21, which would then be negative).

Now let $i \geq 1$ and assume by induction that κ_c and π_c^\sim coincide for all values less than i . The proof is a simple reasoning using the definitions of κ_c and π_c^\sim .

Let $x \in \Sigma^+$. If $\pi_c^\sim(x) = i$, then we already know by induction that $\kappa_c(x) \geq i$. We show that $\kappa_c(x) = i$. Consider an idempotent class that is reachable from the class of x . This class is of the form $[xz]_{\simeq_c}$ for some z with $xz \in E_c$. By definition, $\pi_c^\sim(x) = i$ implies that for each $z \in \Sigma^*$ with $xz \in E_c$, either $(xz)^\omega$ has a prefix in $P_{c,i-1}^\sim(L)$, or $((xz)^\omega \in L_c$ iff i is even). In the first case, if $(xz)^\omega$ has a prefix in $P_{c,i-1}^\sim(L)$, then there is an n such that $\pi_c^\sim((xz)^n) \leq i-1$ (because π_c^\sim is weak). Then by induction also $\kappa_c((xz)^n) \leq i-1$. Since xz is idempotent, we have $xz \simeq_c (xz)^n$, and thus also $\kappa_c(xz) \leq i-1$. In the second case, $((xz)^\omega \in L_c$ iff i is even), the class of xz is positive iff i is even. Hence, each idempotent class that is reachable from x either has a κ_c value smaller than i or is positive iff i is even. Hence $\kappa_c(x) = i$.

For the other direction, assume that $\kappa_c(x) = i$ and show that $\pi_c^\sim(x) = i$. By induction, $\pi_c^\sim(x) \geq i$, which means that $x \notin P_{c,i-1}^\sim$. Let $z \in \Sigma^*$ be such that $xz \in E_c$. Let n be such that the class of $(xz)^n$ is idempotent (by Lemma 21). Since $\kappa_c(x) = i$, the class of $(xz)^n$ has a κ_c -value less than i , or it is positive iff i is even. In the first case, we get by induction that $(xz)^n \in P_{c,i-1}^\sim$. In the second case, $((xz)^n)^\omega = (xz)^\omega \in L_c$ iff i is even. So $x \in P_{c,i}^\sim$ and hence $\pi_c^\sim(x) = i$. \blacktriangleleft

If there are a negative and a positive idempotent class in the same SCC of a PRC, then none of them will satisfy the conditions for being assigned value i by κ_c . This contradicts the fact that every class is assigned a value as a consequence of Lemma 22. We use this structural property of the PRCs in Section 5, and hence state it as a lemma.

► **Lemma 23.** *If two idempotent classes in a PRC of a canonical FORC are in the same SCC, then both are positive or both are negative. Furthermore, two classes in the same SCC of \simeq_c have the same κ_c -value.*

As a consequence of Lemma 22, we obtain a construction from canonical FORCs to DPAs that is only exponential in the parity complexity of the language, improving the upper bound from [2, Proposition 5.9] that is exponential in the size of the FORC (the statement in [2] is about saturated families of DFAs, which are basically refinements of canonical FORCs up to some small details).

► **Theorem 24.** *Let L be a regular ω -language of parity complexity k , \sim be a right congruence that refines \sim_L , and $(\sim, (\simeq_c)_{c \in [\sim]})$ be the canonical FORC for L and \sim . Then the precise DPA \mathcal{A}_L^\sim for L and \sim has at most n^k states, where n is the size of the FORC.*

Proof. The Mealy machine obtained from the PRC \simeq_c by using the transition structure of \simeq_c and assigning the κ_c -value of the target state to a transition computes π_c^\sim according to Lemma 22. Denote the resulting family of Mealy machines by $\overline{\mathcal{M}}$. Then $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$ computes the priority mapping π_{\bowtie}^\sim and thus accepts L by Lemma 15. Since each \simeq_c refines \sim by definition, the component for \sim in the construction of $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$ is not required (and even if it is used it does not blow up the state space). The states used for the component of priority i in $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$ is the sum of the states of the $\mathcal{D}_{c,i}$, each of which has size bounded by the size of \simeq_c . So the number of states used for the component of priority i is at most n . Hence, the size of $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$ is bounded by n^k . \blacktriangleleft

Our completeness result for the learning algorithm uses the fact that we can infer the syntactic FORC of a language if the sample contains enough information. For bounding the number of required examples, we give an upper bound on the size of the syntactic FORC for L in the size of a DPA for L .

Algorithm 1 GLeRC

Input: A function `cons` taking a partial TS as input and returning a boolean.
 A complete TS \mathcal{T}_0 with `cons`(\mathcal{T}_0) = `true`, serving as default.
Output: A complete TS \mathcal{T} such that `cons`(\mathcal{T}) = `true` and $|\mathcal{T}| \leq |\mathcal{T}_0|$.
 $Q \leftarrow \{\varepsilon\}, \delta \leftarrow \emptyset, \mathcal{T} \leftarrow (Q, \Sigma, \delta, \varepsilon)$
while \mathcal{T} is not complete **do**
 if $|\mathcal{T}| > |\mathcal{T}_0|$ **then return** \mathcal{T}_0
 $pa \leftarrow \text{llex. minimal word with } p \in Q, a \in \Sigma \text{ such that } \delta(p, a) \text{ is undefined}$
 foreach $q \in Q$ **in length-lexicographic order do**
 $\delta' \leftarrow \delta \cup \{p \xrightarrow{a} q\}$
 if `cons`(($Q, \Sigma, \varepsilon, \delta'$)) **returns true then**
 $\delta \leftarrow \delta'$ **and continue** with next iteration of while loop
 $Q \leftarrow Q \cup \{pa\}, \delta \leftarrow \delta \cup \{p \xrightarrow{a} pa\}$
return \mathcal{T}

► **Proposition 25.** *Let \mathcal{A} be a normalized DPA with priorities $\{0, \dots, k-1\}$ recognizing the language $L \subseteq \Sigma^\omega$ and \sim be a right congruence with $\sim_{\mathcal{A}} \leq \sim \leq \sim_L$. Let m be the number of \sim -classes, and d such that for each class c of \sim there are at most d many c -states in \mathcal{A} . Then for the canonical FORC of L with LRC \sim , the size of each PRC is at most $m(dk)^d$.*

Proof. Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, \kappa)$, let c be a class of \sim , and let \simeq_c be the canonical PRC for c . Let $Q_c \subseteq Q$ be the set of c -states of \mathcal{A} . Then by assumption $|Q_c| \leq d$. For $x \in \Sigma^*$ define the function $\Delta_{c,x} : Q_c \rightarrow Q \times \{0, \dots, k-1\}$ by induction on the length of x by $\Delta_{c,\varepsilon}(q) = (q, k-1)$ and for $x = x'a$ with $\Delta_{c,x'}(q) = (q', i)$ let $\Delta_{c,x'a}(q) = (\delta(q', a), \min(i, \kappa(q, a)))$.

If two words $x, y \in \Sigma^+$ with $ux \sim uy$ define the same function $\Delta_{c,x} = \Delta_{c,y}$, then $x \simeq_c y$. To see this, let $u \in c$, and $z \in \Sigma^*$ such that $uxz \sim u$. For verifying $ux \sim uy$, let $q := \delta^*(q_0, u) \in Q_c$. By assumption $\Delta_{c,x}(q) = \Delta_{c,y}(q) =: (q', i)$. Thus, $\delta^*(q_0, ux) = \delta^*(q_0, uy) = q'$, and hence $ux \sim uy$.

For verifying $(u(xz)^\omega \in L \Leftrightarrow u(yz)^\omega \in L)$, consider the runs of \mathcal{A} on the two words. On u they are the same. On x and y they might differ, but they reach the same state and see the same minimal priority on the way (because $\Delta_{c,x} = \Delta_{c,y}$). On z they agree again, so they are in the same state after uxz and uyz . Since $uxz \sim u$, this state is in Q_c . We can now repeat this argument, so the two runs only differ on the parts resulting from the x , respectively y , segments of the word. But on these segments, the same minimal priority is visited. Hence, the minimal priority that appears infinitely often is the same in the two runs.

Since $\sim_{\mathcal{A}}$ refines \sim , $\Delta_{c,x}(q_1)$ and $\Delta_{c,x}(q_2)$ are \sim -equivalent for all $q_1, q_2 \in Q_c$. Hence, all the states in the image of $\Delta_{c,x}$ are \sim -equivalent. This implies that there are at most $m(dk)^d$ different possible functions for $\Delta_{c,x}$. Hence, the number of classes in \simeq_c on Σ^+ is bounded by $m(dk)^d$. ◀

5 DPA learner

In this section, we introduce the passive learner **DPAInf** (for “DPA inference”), which constructs in polynomial time a DPA that is consistent with a given input sample. Throughout, we assume $S = (S_+, S_-)$ is a finite ω -sample of ultimately periodic words. The overall idea is to infer a FORC from S and color it according to the procedure outlined in Section 4. We later show that if S contains enough information from L , then the syntactic FORC and the precise coloring will be inferred.

For learning the FORC, we use the procedure *Greedy Learn Right Congruence* (GLeRC), which can be seen in Algorithm 1. It receives as input a consistency function cons and incrementally constructs a TS \mathcal{T} by adding missing transitions, first trying existing states as targets in canonical order. After inserting a transition to a potential target, GLeRC verifies whether cons is satisfied. If yes, GLeRC keeps the transition and proceeds to the next iteration. Otherwise, if no existing state works, GLeRC adds a new state as target for the transition. To ensure termination, GLeRC has a second input, which is a fallback transition system \mathcal{T}_0 . If the constructed TS \mathcal{T} exceeds \mathcal{T}_0 in size at any point, the algorithm terminates and returns \mathcal{T}_0 . This guarantees runtime that is polynomial in the execution time of cons and in the size of \mathcal{T}_0 , since GLeRC attempts to insert at most polynomially many transitions.

► **Proposition 26.** *For a consistency function cons and a complete transition system \mathcal{T}_0 with $\text{cons}(\mathcal{T}_0) = \text{true}$, the algorithm GLeRC computes a complete TS \mathcal{T} such that $\text{cons}(\mathcal{T}) = \text{true}$ in time polynomial in the runtime of cons and the size of \mathcal{T}_0 .*

The cons functions that we pass to GLeRC, verify that the constructed TS satisfies certain notions of consistency with the sample, which we outline below. We use this specific algorithm, since it can be ensured that GLeRC infers the appropriate transition system in the limit. In the following, we introduce the different notions of consistency, which relativize the definitions of the canonical FORC and its coloring from Section 4 to a sample S .

► **Definition 27.** *Let $S = (S_+, S_-)$ be an ω -sample and \mathcal{T} be a partial transition system. Two words $x, y \in \Sigma^*$ are separated by \mathcal{T} , if $\mathcal{T}(x)$ or $\mathcal{T}(y)$ is undefined, or if $\mathcal{T}(x) \neq \mathcal{T}(y)$.*

- We call \mathcal{T} MN-consistent with S , if for all $xuv^\omega \in S_+, yuv^\omega \in S_-$, the prefixes x and y are separated by \mathcal{T} , and
- say that \mathcal{T} is iteration consistent with (S, \sim, c) , where \sim represents a complete transition system that is MN-consistent with S and c is a class of \sim , if for all $xz, yz \in E_c^\sim$ with $(xz)^\omega \in S_+, (yz)^\omega \in S_-$ holds that x and y are separated by \mathcal{T}

A check for MN-consistency is part of the consistency checks in the learning algorithms from [8]. For making the paper self-contained, we nevertheless describe a possible algorithm below. For showing that MN- and iteration consistency can be verified in polynomial time, we reduce the problems to a unified setting, which can be solved by performing a polynomial number of reachability analyses. Specifically, we construct two DFAs $\mathcal{A}_1, \mathcal{A}_2$ and a conflict relation $C \subseteq Q_1 \times Q_2$ from the sample, to encode the pairs of words x, y which are not allowed to lead to the same state in a consistent transition system \mathcal{T} . Now we can check for consistency of \mathcal{T} by verifying for all $x, y \in \Sigma^*$, that $(\delta_1^*(\iota_1, x), \delta_2^*(\iota_2, y)) \in C$ implies that x and y do not lead to the same state in \mathcal{T} . In other words \mathcal{T} is not consistent iff there is some q in \mathcal{T} such that (q, q_1) and (q, q_2) are reachable in $\mathcal{T} \times \mathcal{A}_1$ and $\mathcal{T} \times \mathcal{A}_2$, respectively, and $(q_1, q_2) \in C$. This can clearly be checked in time polynomial in the size of $|\mathcal{T}|, |\mathcal{A}_1|$ and $|\mathcal{A}_2|$. In the proof of the following two lemmas, we thus only need to show that suitable DFAs $\mathcal{A}_1, \mathcal{A}_2$ and a conflict relation C can be constructed in polynomial time.

► **Lemma 28.** *It can be verified in polynomial time whether a (partial) transition system \mathcal{T} is MN-consistent with an ω -sample S .*

Proof. We construct DFAs $\mathcal{A}_1, \mathcal{A}_2$ such that \mathcal{A}_1 and \mathcal{A}_2 accept all prefixes of S_+ and S_- , respectively. This can easily be done by using the prefix trees for S_+ , respectively S_- , and attaching a loop for the periodic part once the prefix uniquely identifies the example. All states of the resulting transition system are accepting. Missing transitions are directed into a rejecting sink state.

A pair (q_1, q_2) of states with $q_i \in \mathcal{A}_i$ for $i \in \{1, 2\}$ is in the conflict relation C , if there exists an infinite run in the product automaton $\mathcal{A}_1 \times \mathcal{A}_2$, which starts in (q_1, q_2) and stays in $F_1 \times F_2$. Checking for the existence of such an infinite run starting in a pair (q_1, q_2) can be done by restricting the product to $F_1 \times F_2$ and searching for a loop. As the product $\mathcal{A}_1 \times \mathcal{A}_2$ is polynomial in S , the construction of C is clearly possible in polynomial time.

To see that the construction is correct, let q be a state of \mathcal{T} such that (q, q_1) and (q, q_2) are reachable in $\mathcal{T} \times \mathcal{A}_1$ and $\mathcal{T} \times \mathcal{A}_2$, respectively, and $(q_1, q_2) \in C$. Further, let $r_1, r_2 \in \Sigma^*$ be such that $\mathcal{T} \times \mathcal{A}_i$ reaches (q, q_i) when reading r_i from the initial state. As $(q_1, q_2) \in C$, there exists an infinite run from (q_1, q_2) in $\mathcal{A}_1 \times \mathcal{A}_2$, which stays in $F_1 \times F_2$. This means for some $x \in \Sigma^*, y \in \Sigma^+$ we have $(q_1, q_2) \xrightarrow{x} (p_1, p_2) \xrightarrow{y} (p_1, p_2)$ while only visiting states from $F_1 \times F_2$. Then $r_1xy^\omega \in S_+$ and $r_2xy^\omega \in S_-$ and because r_1 and r_2 reach the same state in \mathcal{T} , \mathcal{T} cannot be MN-consistent with S .

For the other direction, assume that \mathcal{T} is not MN-consistent with S . This means there exist $xw, yw \in S$ with $xw \in S_+$ and $yw \in S_-$, but x and y lead to the same state q in \mathcal{T} . The pair $(q_1, q_2) := (\delta_1^*(\iota_1, x), \delta_2^*(\iota_2, y))$ is in C , because for each prefix z of w , $xz, yz \in \text{Prf } S$, guaranteeing that there exists an infinite run that starts in (q_1, q_2) and remains in $F_1 \times F_2$.

Overall, \mathcal{T} is *not* MN-consistent with S , if and only if there is some q in \mathcal{T} such that (q, q_1) and (q, q_2) are reachable in $\mathcal{T} \times \mathcal{A}_1$ and $\mathcal{T} \times \mathcal{A}_2$, respectively, and $(q_1, q_2) \in C$. As the sizes of $\mathcal{A}_1, \mathcal{A}_2$ and C are polynomial in $|S|$, it follows from the remarks preceding this proof that MN-consistency can be checked in polynomial time. \blacktriangleleft

► **Lemma 29.** *For a given finite ω -sample S , a right congruence \sim whose TS is MN-consistent with S , a class c of \sim , and a partial TS \mathcal{T} , it can be checked in polynomial time whether \mathcal{T} is iteration consistent with (S, \sim, c) .*

Proof. The proof of this lemma is analogous to the preceding Lemma 28, but differs in the construction of the $\mathcal{A}_1, \mathcal{A}_2$ and the conflict relation. We build the DFAs such that

- \mathcal{A}_1 accepts a word x if $x \in E_c^\sim$ and $x^\omega \in S_+$, whereas
- \mathcal{A}_2 accepts x if $x \in E_c^\sim$ and $x^\omega \in S_-$.

For the construction of the \mathcal{A}_i , let \check{R}_σ for $\sigma \in \{+, -\}$ be the set consisting of all y such that y is a shortest word with $y^\omega \in S_\sigma$. For computing the sets \check{R}_σ , one can consider each uv^ω in S , check whether it is periodic, and if yes, compute the shortest y with $y^\omega = uv^\omega$. This can be done, for example, by building a DFA that accepts precisely the prefixes of uv^ω (using the prefixes of uv as states, identifying the states for u and uv). This DFA can be minimized, and then uv^ω is periodic iff the minimal DFA consists of a loop of accepting states (with one rejecting sink). The label sequence of this loop gives the shortest word y . One can also use other methods for computing \check{R}_σ , for example the characterization of minimal representations of ultimately periodic words from [23, Proposition 4.42].

Now it is not hard to see that the words x with $x^\omega \in S_\sigma$ are precisely those of the form y^n with $y \in \check{R}_\sigma$. To see that, note that $x^\omega \in S_\sigma$ implies that $x^\omega = y^\omega$ for some $y \in \check{R}_\sigma$. So x must be of the form $y^n z$ with z a strict prefix of y . If $z \neq \varepsilon$, this would imply that $z^\omega = y^\omega$, contradicting the fact that y is the shortest word representing y^ω .

With these observations, it is then easy to build the DFAs $\mathcal{A}_1, \mathcal{A}_2$: The states are prefixes of y^ω for $y \in \check{R}_\sigma$ (with $\sigma = +$ for \mathcal{A}_1 , and $\sigma = -$ for \mathcal{A}_2). For the least k such that y^k is not a prefix of another $(y')^\omega$ anymore, start looping on y . Finally, we intersect the resulting automata with DFAs for E_c^\sim , which can be directly obtained from the transition structure of \sim by using c as initial and final state (since the DFAs that we built from \check{R}_σ only accept non-empty words, we do not need to exclude the empty word in the DFA for E_c^\sim). Overall, the size of the resulting automata is clearly polynomial in $|S|$.

The conflict relation is the least relation C over $Q_1 \times Q_2$ containing $F_1 \times F_2$ and verifying that $(\delta_1(q_1, a), \delta_2(q_2, a)) \in C$ implies $(q_1, q_2) \in C$. The relation C can be obtained by a fixpoint iteration, ensuring that the number of iterations is bounded by $|Q_1 \times Q_2|$. Thus, the computation of C is possible in polynomial time.

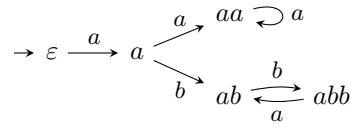
For the correctness of the construction, let q be a state of a partial TS \mathcal{T} which is MN-consistent with S . Further, let (q, q_1) and (q, q_2) with $(q_1, q_2) \in C$ be reachable on r_1 in $\mathcal{T} \times \mathcal{A}_1$ and on r_2 in $\mathcal{T} \times \mathcal{A}_2$, respectively. As $(q_1, q_2) \in C$, there must exist some word $x \in \Sigma^*$ such that $(p_1, p_2) := (\delta_1^*(q_1, x), \delta_2^*(q_2, x)) \in F_1 \times F_2$. This means $r_1 x, r_2 x \in E_c^\sim$ and $(r_1 x)^\omega, (r_2 x)^\omega \in S$ with $((r_1 x)^\omega \in S_+ \Leftrightarrow (r_2 x)^\omega \in S_-)$. But then because r_1 and r_2 reach the same state in \mathcal{T} , it follows that \mathcal{T} is not iteration consistent with (S, \sim, c) .

For the other direction, assume that \mathcal{T} is MN-consistent with S but not iteration consistent with (S, \sim, c) . This means we can find $xz, yz \in E_c^\sim$ such that $(xz)^\omega \in S_+$, $(yz)^\omega \in S_-$ and x, y lead to the same state p in \mathcal{T} . Clearly it holds that $(q_1, q_2) := (\delta_1^*(xz), \delta_2^*(yz)) \in F_1 \times F_2 \subseteq C$. Moreover, our definition of C ensures that removing the common suffix z from both words retains membership in C , meaning $(\delta_1^*(x), \delta_2^*(y)) \in C$.

Overall, we obtain that \mathcal{T} is *not* iteration consistent with (S, \sim, c) , if and only if there is some q in \mathcal{T} such that (q, q_1) and (q, q_2) are reachable in $\mathcal{T} \times \mathcal{A}_1$ and $\mathcal{T} \times \mathcal{A}_2$, respectively, and $(q_1, q_2) \in C$. As $\mathcal{A}_1, \mathcal{A}_2$ and C can be constructed in polynomial time, checking for iteration consistency effectively reduces to computing the reachable states in the two products $\mathcal{T} \times \mathcal{A}_1$ and $\mathcal{T} \times \mathcal{A}_2$ and checking whether a pair with the properties outlined above exists. This is clearly possible in time polynomial in the size of $|\mathcal{T}|$, $|\mathcal{A}_1|$ and $|\mathcal{A}_2|$, thus concluding the proof. \blacktriangleleft

We now combine the notions of MN- and iteration consistency to define the conditions under which a FORC is consistent with a sample.

- **Definition 30.** A FORC $(\sim, (\approx_c)_{c \in [\sim]})$ is consistent with S if
- \sim is MN-consistent with S and
 - for each $c \in [\sim]$, \approx_c is MN-consistent with S_c and iteration consistent with (S_c, \sim, c) , where $S_c = (S_{c,+}, S_{c,-})$ and $S_{c,\sigma} = \{uv^\omega \mid \exists x \in c \text{ with } xuv^\omega \in S_\sigma\}$ for $\sigma \in \{+, -\}$.

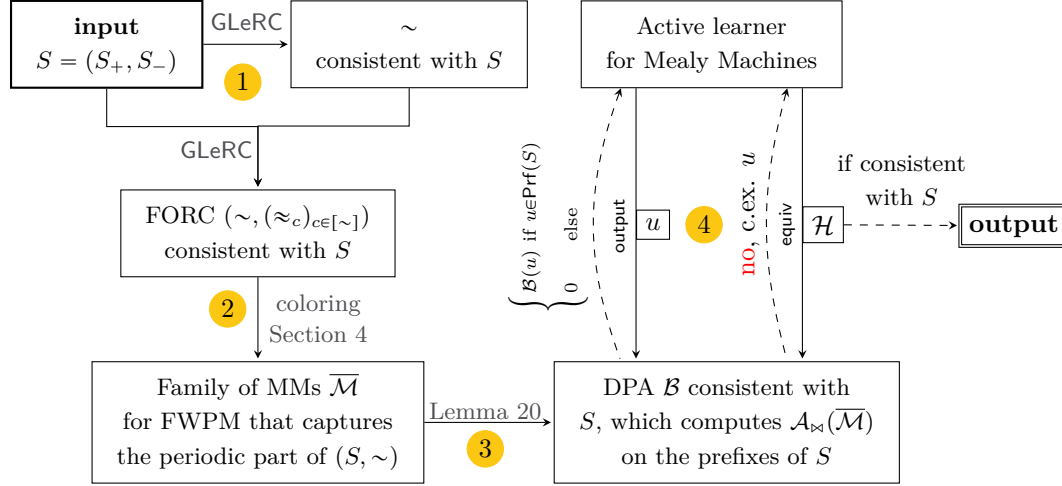


■ **Figure 6** An example for the construction of a default structure \sim_S for the sample S containing only the examples a^ω and $ab(ba)^\omega$. Note, that the classification of these words is not relevant for the definition of the default structure \sim_S , which is why we omit it from this description.

In the following, we outline how default structures that are MN- and iteration consistent with a given sample S can be constructed. An illustration for this construction can be seen in Figure 6. For a given sample S , we can construct default structures that are MN- and iteration consistent as follows. Consider the prefix tree of S , to which we add one sink state for words that are not prefixes of S , and loops on the shortest words which are prefix of exactly one example word. Formally, we define \sim_S as $x \sim_S y$ if $x, y \notin \text{Prf}(S)$ or we have $x^{-1}S = y^{-1}S$ and $|x^{-1}S| = 1$. It is not hard to see that the size of \sim_S is polynomial in $|S|$, see for example [10, Appendix B].

Formal description of the learner

The learner that we propose consists of several steps. An overview of the algorithm can be found in Figure 7. Roughly speaking, it extracts a FORC from the given sample, colors it using the algorithm from Section 4, builds a family of Mealy machines for weak priority mappings capturing the periodic part of the sample, and then uses an active learner for Mealy machines for joining the learned FWPM. In the following, we explain each step of the learner DPAlnf in more detail.



■ **Figure 7** Schematic view of the DPAlnf algorithm, details for each step are in the text.

Step 1 In the first step, DPAlnf learns a FORC that is consistent with the sample S using GLeRC. The procedures for checking MN-consistency and iteration consistency are based on Lemma 28 and Lemma 29. For the LRC, DPAlnf calls GLeRC with \sim_S as default and using LRC-cons as constraint function, where $\text{LRC-cons}(T)$ returns true iff T is MN-consistent with S (see Lemma 28). We denote the resulting right congruence with \sim . For each $c \in [\sim]$ and $\sigma \in \{+, -\}$, the learner now computes the sets

$$S_{c,\sigma} = \{uv^\omega \mid \exists x \in c \text{ with } xuv^\omega \in S_\sigma\} \quad R_{c,\sigma} = \{v^\omega \mid \exists x \in c \text{ with } xv^\omega \in S_\sigma \text{ and } xv \sim x\}.$$

It then executes GLeRC with the product $(\sim_{S_c} \times \sim)$ as default and the constraint function $\text{PRC-cons}(T)$, which returns true if

- \mathcal{T} is MN-consistent and iteration consistent with S_c
- $\text{SCC}_{\mathcal{T}}(q) \cap \text{SCC}_{\mathcal{T}}(q') = \emptyset$ for all $q \in \text{Inf}_{\mathcal{T}}(R_{c,+})$, $q' \in \text{Inf}_{\mathcal{T}}(R_{c,-})$.

Thereby, a FORC $\mathcal{F} = (\sim, (\approx_c)_{c \in [\sim]})$ that is consistent with S is obtained. The second condition ensures that \mathcal{F} meets the properties of Lemma 23. This makes it possible to compute a coloring analogous to κ_c , in the next step.

Step 2 In the following, we use $\mathcal{F} = (\sim, (\approx_c)_{c \in [\sim]})$ to refer to the FORC that is constructed in step 1. The algorithm now builds a family of Mealy machines $\bar{\mathcal{M}} = (\mathcal{M}_c)_{c \in [\sim]}$ computing a weak priority mapping $\bar{\gamma} = (\gamma_c)_{c \in [\sim]}$. This is done in a way which mimics the definition of the coloring κ_c in Section 4 and ensures that $\bar{\gamma}$ is the precise FWPM of L , provided the syntactic FORC was learned in step 1 (see Lemma 22). For a class $c \in [\sim]$, we proceed as follows:

- Label all states with σ in \mathcal{T}_{\approx_c} , the TS that represents \approx_c , that appear in $\text{Inf}_{\mathcal{T}}(R_{c,\sigma})$ with $\sigma \in \{+, -\}$.
- For each state q , compute the set P_q containing all states that are reachable from q and have a classification.
- Starting with $i = 0$ and increasing i after every iteration, assign i to those states q such that each $p \in P_q$ either already has a priority, or (p has classification $+$ iff i is even).

As the construction of \mathcal{F} in step 1 ensures that no SCCs with positive and negative looping sample words exist, each class of a PRC \approx_c is assigned a priority. For each class c of \sim , we obtain a Mealy machine \mathcal{M}_c on the transition system of \approx_c , which uses the priority (wrt. the computed coloring) of the target state for each transition. Overall, this procedure returns in polynomial time a family of Mealy machines $\overline{\mathcal{M}} = (\mathcal{M}_c)_{c \in [\sim]}$ whose size is polynomial in \mathcal{F} .

Step 3 In the following, we use \mathcal{A}_{\bowtie} to refer to the DPA $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$ from Section 3. Note that we cannot simply construct $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$ because its size can be exponential in $\overline{\mathcal{M}}$. Instead, DPALnf builds a DPA \mathcal{B} which computes the priority mapping

$$\mathcal{B} : u \mapsto \mathcal{A}_{\bowtie}(u) \text{ if } u \in \text{Prf}(S) \text{ and } u \mapsto 0 \text{ otherwise.}$$

It can be shown by using Lemma 20 that constructing \mathcal{B} is possible in polynomial time (see the proof of Theorem 31 below). Then, it is verified that \mathcal{B} is consistent with S , i.e. that $S_+ \subseteq L(\mathcal{B})$ and $S_- \cap L(\mathcal{B}) = \emptyset$. If yes, the algorithm proceeds to step 4. Otherwise, we redefine \mathcal{B} as a fallback DPA that outputs 1 on infixes of negative loops and 0 otherwise. Formally, the fallback DPA computes the priority mapping $\mathcal{B} : \Sigma^+ \rightarrow \{0, 1\}$ with $\mathcal{B}(u) \mapsto 1$ iff $u \in \text{Prf}(\bigcup_{c \in [\sim]} R_{c,-})$.

Step 4 DPALnf now runs a polynomial time active learner MMAL for Mealy machines as black box. The oracle provided to MMAL uses the DPA \mathcal{B} from step 3 and reacts to queries as follows:

$$\begin{aligned} \text{output}(u) &\rightsquigarrow \text{answer } \mathcal{B}(u) \\ \text{equiv}(\mathcal{H}) &\rightsquigarrow \begin{cases} \text{terminate and return } \mathcal{H} & \text{if } \mathcal{H} \text{ is consistent with } S \\ \text{answer } \min_{\text{lex}} \{x \in \text{Prf}(S) \mid \mathcal{B}(x) \neq \mathcal{H}(x)\} & \text{otherwise.} \end{cases} \end{aligned}$$

This ends the description of the algorithm, and we now state the main results of this section.

► **Theorem 31.** *DPALnf computes in polynomial time a DPA that is consistent with the ω -sample S it receives as input.*

Proof. Let $S = (S_+, S_-)$ be the sample on which DPALnf is called. If step 4 terminates, then by definition of the procedure, it does so with a DPA \mathcal{B} that is consistent with S . Thus, it remains to show that DPALnf always terminates in polynomial time. In the following, we consider each step of the algorithm individually:

- In Lemma 28 and Lemma 29, it was established that the two constraint functions LRC-cons and PRC-cons, which are passed to GLeRC, run in polynomial time. Thus by Proposition 26, it follows that the first step terminates in polynomial time.
- The second step begins by computing the infinity sets of all sample words from R_c in the respective PRC \approx_c for a $c \in [\sim]$, which is possible in time polynomial in $|\approx_c|$ and S . Further, the number of distinct priorities is bounded for each class $c \in [\sim]$ by the number of SCCs in \approx_c . Therefore the computation of each κ_c terminates after at most polynomially many iterations. Since each iteration consists of a reachability analysis and some elementary operations, the construction of $\overline{\mathcal{M}}$ is clearly possible in polynomial time.

- We use \mathcal{A}_{\bowtie} to denote $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$. For the computation of \mathcal{B} , we first build what we call a *colored sample* \hat{S} as follows. For each sample word $uv^\omega \in S$, we compute the sequence of colors produced by \mathcal{A}_{\bowtie} on uv^ω . By Lemma 20, we can write this sequence as rs^ω , where $|rs|$ is polynomial in $|uv|$ and $|\overline{\mathcal{M}}|$. The product of uv^ω with rs^ω (which is a sequence of letter-priority pairs), is then polynomial in $|S|$. Overall, the size of the colored sample \hat{S} is polynomial in $|S|$.

\mathcal{B} then basically is the prefix tree of \hat{S} starting to loop on the periodic part if the prefix uniquely identifies the word in \hat{S} , and a sink state q_\perp for all non-prefixes of S . More formally, for each colored example $\hat{u}\hat{v}^\omega$, there is n , polynomial in the size of \hat{S} , such that $\hat{u}\hat{v}^n$ is not a prefix of any other example in \hat{S} . As states of \mathcal{B} we take all the prefixes of $\hat{u}\hat{v}^{n+1}$ (for all the examples from \hat{S}). For some prefix \hat{x} and a letter a , we define the transitions $\delta_{\mathcal{B}}(\hat{x}, a)$ as follows. If for all i , $\hat{x}(a, i)$ is not a prefix of \hat{S} , then $\delta(\hat{x}, a) = (q_\perp, 0)$. Otherwise, $\hat{x}(a, i)$ is a prefix of \hat{S} for a unique i . If $\hat{x}(a, i)$ is a state of \mathcal{B} , then $\delta_{\mathcal{B}}(\hat{x}, a) = (\hat{x}(a, i), i)$. Otherwise, \hat{x} is of the form $\hat{u}\hat{v}^{n+1}$, and $\hat{u}\hat{v}^{n+1}(a, i)$ is prefix of $\hat{u}\hat{v}^\omega$, and only of that word. We then set $\delta_{\mathcal{B}}(\hat{x}, a) = (\hat{u}\hat{v}^n(a, i), i)$, defining the loop for the periodic part of $\hat{u}\hat{v}^\omega$. This construction results in a polynomial size DPA \mathcal{B} that computes the priority mapping defined in in Step 3.

It can be checked in polynomial time whether \mathcal{B} is consistent with the sample (actually, this can already be checked on \hat{S} by looking at the minimal priorities on the loops of the colored examples). In case \mathcal{B} is not consistent with S , DPALnf constructs a fallback DPA. The construction of such a fallback DPA also works in polynomial time along the same lines as the construction explained above (take infixes of the periodic parts of negative example words as states, entering a loop once the infix uniquely identifies the periodic part of a negative example).

- Finally in step 4, all answers provided to MMAL are consistent with the DPA \mathcal{B} from step 3, which by the previous considerations is polynomial in $|S|$. Since \mathcal{B} itself is consistent with S , it follows from the properties of MMAL, that it terminates at the latest with the DPA \mathcal{B} , or it terminates earlier with a strictly smaller DPA that is consistent with S . To conclude that the execution of MMAL indeed terminates in polynomial time, we need to verify that each counterexample provided by the teacher is polynomial in $|S|$. So let \mathcal{H} be the hypothesis posed by MMAL, which is not consistent with S . This means \mathcal{H} and \mathcal{B} viewed as Mealy machines compute different priority mappings and a counterexample corresponds to a prefix x of S witnessing that $\mathcal{H}(x) \neq \mathcal{B}(x)$. As all hypotheses given by MMAL in an equivalence query have at most as many states as \mathcal{B} , the shortest such x is clearly polynomial in $|S|$.

As each step of DPALnf runs in polynomial time and we have shown that the DPA it constructs must be consistent with S , the statement follows. ◀

The core idea for constructing a characteristic sample is to simulate a run DPALnf and whenever the algorithm would make a mistake in producing the canonical object we are interested in, we add words preventing it, which leads to the following theorem.

► **Theorem 32.** *DPALnf can learn a DPA \mathcal{A} for every ω -regular language L in the limit. Moreover, the size of \mathcal{A} is bounded by the size of the precise DPA \mathcal{A}_L for L , and there exists a characteristic sample S^L for L that is polynomial in the maximum of the size of \mathcal{A}_L and the size of the syntactic FORC for L .*

Proof. Let L be a regular ω -language over some alphabet Σ . Recall that our goal is to construct a characteristic sample $S^L = (S_+, S_-)$ for L , meaning that DPALnf infers a DPA

for L from any sample S' which contains S^L and is consistent with L . In the following, we illustrate how the sample S^L can be built in such a way that it guarantees that

- the syntactic FORC $\mathcal{F}_L = (\sim_L, (\simeq_c)_{c \in [\sim]})$ for L is learned in the first step,
- a family of Mealy machines $\overline{\mathcal{M}} = (\mathcal{M}_c)_{c \in [\sim_L]}$ for the precise coloring $\overline{\pi^{\sim_L}} = (\pi_c)_{c \in [\sim_L]}$ is learned in step 2,
- for all $\text{output}(u)$ queries posed by MMAL in step 4, $u \in \text{Prf}(S^L)$ and
- it contains the necessary counterexamples to ensure that MMAL terminates with a DPA for L .

For convenience, we do not describe S^L directly. Instead, we build a set W of ultimately periodic words and subsequently define $S^L = (W \cap L, W \setminus L)$.

The core idea for ensuring that the syntactic FORC \mathcal{F}_L is learned in the first step, is to simulate the necessary executions of GLeRC step by step, and to extend W whenever GLeRC would otherwise insert a transition that is not present in the target RC. Let \mathcal{T} be the partial TS that GLeRC has constructed up to the point at which we want to prevent the insertion of a transition. For a state q of \mathcal{T} , we use $\text{mr}(q)$ to denote the length-lexicographically minimal word that reaches q . Then if a transition $\delta(q, a) = p$ has to be prevented, it must be that $xa \not\sim y$ for $x = \text{mr}(q), y = \text{mr}(p)$ because up to now all transitions have been inserted correctly. Depending on the kind of RC we want to learn, we add the following words to W :

- If \sim is the LRC of \mathcal{F}_L and $xa \not\sim_L y$:
There exist $u \in \Sigma^*, v \in \Sigma^+$ with $xauv^\omega \in L$ iff $yuv^\omega \notin L$, i.e. uv^ω separates xa and y in \sim_L . We pick the length-lexicographically minimal such u and v and add the words $xauv^\omega$ and yuv^ω to W .
- If \simeq_c is the PRC for some class $c \in [\sim_L]$ and $xa \not\simeq_c y$:
Let $u = \text{mr}(c)$. If not $uxa \sim_L uy$, then we proceed as for the LRC. Otherwise, there exists some $z \in \Sigma^*$ with $uxaz \sim_L u$ and $(u(xaz)^\omega \in L \Leftrightarrow u(yz)^\omega \notin L)$. Again, we choose the length-lexicographically minimal such z and add $u(xaz)^\omega$ and $u(yz)^\omega$ to W .

For computing the coloring, DPALnf considers all words in $R_{c,\sigma}$ for $\sigma \in \{+, -\}$. To ensure that the second step of DPALnf terminates with the precise coloring $\overline{\pi^{\sim_L}}$ for (L, \sim_L) we add to W words which guarantee that all idempotent classes of \simeq_c are in the infinity set of some word in R_c . Let $c \in [\sim_L]$ be a class and $u \in c$. From each idempotent class of \simeq_c , we pick the length-lexicographically least representative x and add ux^ω to W . Note that since x is idempotent in \simeq_c , it must also be in E_c and hence x^ω will be in the set R_c that is computed in the beginning of step 2. This guarantees that if the syntactic FORC is learned in the first step, every idempotent class of \simeq_c obtains a correct classification in step 2 of DPALnf. As the subsequent computation of the mapping mirrors that of κ_c from Section 4, it is guaranteed that the family of Mealy machines $\overline{\mathcal{M}}$ returned by step 2 computes the precise FWPM $\overline{\pi^{\sim_L}}$.

Let $\mathcal{A}_\boxtimes := \mathcal{A}_\boxtimes(\overline{\mathcal{M}})$ and \mathcal{B} be as constructed in step 3. Further, let $S = (S_+, S_-) = (W \cap L, W \setminus L)$ be the obtained from the set W constructed so far. Since $\overline{\mathcal{M}}$ computes the precise FWPM $\overline{\pi^{\sim_L}}$, it follows from Lemma 16 that \mathcal{A}_\boxtimes computes the priority mapping of \mathcal{A}_L . By definition, \mathcal{B} behaves like \mathcal{A}_\boxtimes on prefixes from S , which guarantees that \mathcal{B} must be consistent with S and the algorithm does not default to the fallback DPA. However in the last step, there might be some $\text{output}(u)$ queries posed by MMAL, which are (falsely) answered with 0 purely because u is not a prefix of S . Thus, to ensure that all queries are answered correctly, we simulate an execution of MMAL with a teacher for \mathcal{A}_\boxtimes and whenever MMAL poses an $\text{output}(u)$ query for a word that is not yet a prefix of W , we add u^ω to W . Additionally, for each equivalence query $\text{equiv}(\mathcal{H})$ with $L(\mathcal{H}) \neq L$, we add the least counterexample to W that witnesses the inequality. The overall construction of S^L is

guaranteed to terminate, because all answers given to MMAL are consistent with \mathcal{A}_L , which by the properties of MMAL guarantees that it terminates at the latest with the DPA \mathcal{A}_L .

In the end, $S^L = (S_+, S_-) = (W \cap L, W \setminus L)$ is the characteristic sample for L . Our construction ensures that if called on a sample which is consistent with L and contains S^L , DPALnf infers the syntactic FORC of L in step 1, and then computes the precise FWPM $\overline{\pi^{\sim L}}$ in step 2. The output queries of MMAL are all on prefixes of S^L , and as long as MMAL does not propose a hypothesis that is consistent with L , the sample contains an example witnessing the difference. Hence DPALnf terminates with a DPA \mathcal{A} for L .

For an upper bound on the size of S^L , we consider the words which are added in each step. The number of insertions that have to be prevented in step 1 and the number of idempotent classes that need to be considered in step two are clearly polynomial in the syntactic FORC \mathcal{F}_L . The shortest words for witnessing the inequivalences that prevent the insertions are polynomial in the size of \mathcal{F}_L . In step four, the number of added words (because of output or equivalence queries) and their lengths are polynomial in the size of the precise DPA \mathcal{A}_L because the hypotheses are growing and bounded by the size of \mathcal{A}_L , and the algorithm runs in time polynomial in the resulting Mealy machine and the length of the longest counterexample. In summary, the size of S^L is polynomial in the maximum of $|\mathcal{F}_L|$ and $|\mathcal{A}_L|$. \blacktriangleleft

By combining Theorem 32 with Proposition 25 and Theorem 18, we can recover and extend the currently known results on passive learning of deterministic ω -automata from polynomial data. For that purpose, we consider the subclasses $\text{IRC}(\text{DPA})$ and $d\text{-IRC}(k\text{-DPA})$ of the ω -regular languages. The first one contains all languages L that can be accepted by a DPA that has exactly one state for each \sim_L -class. Polynomial time passive learners that can learn each language from this class in the limit from polynomial data are presented in [4, 8]. The class $d\text{-IRC}(k\text{-DPA})$ contains all languages L that can be recognized by a DPA with priorities in $\{0, \dots, k-1\}$ and at most d many c -states for each \sim_L class c . A polynomial time passive learner that, for fixed d , can learn each language in $d\text{-IRC}(2\text{-DPA})$ in the limit from polynomial data is presented in [9].

The languages in $\text{IRC}(\text{DPA})$ are those for which $d = 1$ in Proposition 25 and Theorem 18. In this case, the term in Theorem 18 reduces to m and the one in Proposition 25 to mk . If d and k are both fixed, both expressions become linear in m . This implies the following.

► **Corollary 33.** *The algorithm DPALnf can learn a DPA for every language in $\text{IRC}(\text{DPA})$ from polynomial data. Furthermore, there is a fixed polynomial g such that for every k and d , DPALnf can learn a DPA for every language in $d\text{-IRC}(k\text{-DPA})$ in the limit from polynomial data with characteristic samples of size $\mathcal{O}(g)$.*

6 Conclusion

We have presented a passive learner for deterministic parity automata that runs in polynomial time and can learn a DPA for every regular ω -language in the limit. Our upper bound for the size of complete samples is, in general, exponential in the size of a minimal DPA for the language. However, for fixed number of priorities and fixed maximal number of pairwise language equivalent states, this bound becomes polynomial. The learning algorithm is based on the precise DPA of a language that we introduced in this paper, and that can be constructed from the syntactic FORC of the language.

We see two natural main directions of future research based on the results presented here. First, we proposed a basic version of the algorithm that is complete for the class of regular

ω -languages and runs in polynomial time. But there are many parts of the algorithm that allow for optimizations and variations without losing these properties. These variations should then be implemented and compared in an empirical study. For example, all the progress congruences of the FORC are learned independently, while there are strong dependencies between the progress congruences of the syntactic FORC. One can explore techniques for learning these congruences while respecting mutual dependencies. And one can also try other types of passive learning algorithms from finite automata adapted to learning FORCs. Also, variations of the active learner used in the last step of the algorithm can have an impact on the running time and the result produced by the overall procedure.

Second, the precise DPA for a language deserves further study. An understanding which structural properties of a language can cause the precise DPA to be much larger than a minimal DPA for the language might give insights to minimization problems for DPAs. Furthermore, our construction of the precise DPA from the syntactic FORC has similarities with a construction that starts from the syntactic semigroup (see specifically Lemma 22 in [14]). And the construction of the precise FWPM from the syntactic FORC using idempotent classes suggests a variation of FDFAs with “idempotent acceptance”, which could lead to smaller representations of ω -languages by FDFAs. There is also a connection to the canonical representation by good-for-games automata from [16, Definition 4], in the sense that the precise DPA computes the natural color in the limit. Finally, one can show that the class of precise DPAs for a language subsumes the class of normalized DPAs in the sense that a DPA \mathcal{A} is normalized and minimal as a Mealy machine if it is the precise DPA for $(L(\mathcal{A}), \sim_{\mathcal{A}})$. So it seems that studying precise DPAs and their construction in more detail can lead to further insights on connections between representations of ω -languages.

References

- 1 Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987. URL: <https://www.sciencedirect.com/science/article/pii/0890540187900526>, doi:10.1016/0890-5401(87)90052-6.
- 2 Dana Angluin, Udi Boker, and Dana Fisman. Families of DFAs as acceptors of omega-regular languages. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016*, volume 58 of *LIPIcs*, pages 11:1–11:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. URL: <http://www.dagstuhl.de/dagpub/978-3-95977-016-3>.
- 3 Dana Angluin and Dana Fisman. Learning regular omega languages. *Theor. Comput. Sci.*, 650:57–72, 2016. doi:10.1016/j.tcs.2016.07.031.
- 4 Dana Angluin, Dana Fisman, and Yaara Shoval. Polynomial identification of omega-automata. In *Tools and Algorithms for the Construction and Analysis of Systems - 26th International Conference, TACAS 2020*, volume 12079 of *Lecture Notes in Computer Science*, pages 325–343. Springer, 2020. doi:10.1007/978-3-030-45237-7_20.
- 5 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
- 6 Stephan Barth and Martin Hofmann. Learn with SAT to minimize büchi automata. In *Proceedings Third International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2012*, volume 96 of *EPTCS*, pages 71–84, 2012. doi:10.4204/EPTCS.96.
- 7 Alan W. Biermann and Jerome A. Feldman. On the synthesis of finite-state machines from samples of their behavior. *IEEE Trans. Computers*, 21(6):592–597, 1972. doi:10.1109/TC.1972.5009015.
- 8 León Bohn and Christof Löding. Constructing deterministic ω -automata from examples by an extension of the RPNI algorithm. In *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021*, volume 202 of *LIPIcs*, pages 20:1–20:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.MFCS.2021.20.

- 9 León Bohn and Christof Löding. Passive learning of deterministic büchi automata by combinations of DFAs. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 114:1–114:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.114.
- 10 León Bohn and Christof Löding. Constructing deterministic ω -automata from examples by an extension of the rpni algorithm, 2021. URL: <https://arxiv.org/abs/2108.03735>, doi:10.48550/ARXIV.2108.03735.
- 11 J Richard Büchi. On a decision method in restricted second order arithmetic, logic, methodology and philosophy of science (proc. 1960 internat. congr.), 1962.
- 12 Hugues Calbrix, Maurice Nivat, and Andreas Podelski. Ultimately periodic words of rational ω -languages. In Stephen Brookes, Michael Main, Austin Melton, Michael Mislove, and David Schmidt, editors, *Mathematical Foundations of Programming Semantics*, pages 554–566, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.
- 13 Olivier Carton and Ramón Maceiras. Computing the rabin index of a parity automaton. *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications*, 33(6):495–505, 1999. URL: http://www.numdam.org/item/ITA_1999__33_6_495_0/.
- 14 Thomas Colcombet. Green’s relations and their use in automata theory. In *Language and Automata Theory and Applications - 5th International Conference, LATA 2011*, volume 6638 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2011. doi:10.1007/978-3-642-21254-3_1.
- 15 Colin de la Higuera and Jean-Christophe Janodet. Inference of $[\omega]$ -languages from prefixes. *Theor. Comput. Sci.*, 313(2):295–312, 2004. doi:10.1016/j.tcs.2003.11.009.
- 16 Rüdiger Ehlers and Sven Schewe. Natural colors of infinite words. In *Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2022*, volume 250 of *LIPIcs*, pages 36:1–36:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.FSTTCS.2022.36.
- 17 Azadeh Farzan, Yu-Fang Chen, Edmund M. Clarke, Yih-Kuen Tsay, and Bow-Yaw Wang. Extending automated compositional verification to the full class of omega-regular languages. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 2–17. Springer, 2008. doi:10.1007/978-3-540-78800-3_2.
- 18 E Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967. URL: <https://www.sciencedirect.com/science/article/pii/S001995867911655>, doi:10.1016/S0019-9958(67)91165-5.
- 19 E. Mark Gold. Complexity of automaton identification from given data. *Inf. Control.*, 37(3):302–320, 1978. doi:10.1016/S0019-9958(78)90562-4.
- 20 John E. Hopcroft and Jeffrey D. Ullman. *Formal Languages and their Relation to Automata*. Addison-Wesley, 1969.
- 21 Malte Isberner, Falk Howar, and Bernhard Steffen. The open-source learnlib - A framework for active automata learning. In *Computer Aided Verification - 27th International Conference, CAV 2015*, volume 9206 of *Lecture Notes in Computer Science*, pages 487–495. Springer, 2015. doi:10.1007/978-3-319-21690-4_32.
- 22 Nils Klarlund. A homomorphism concepts for omega-regularity. In *Computer Science Logic, 8th International Workshop, CSL ’94, Kazimierz, Poland, September 25-30, 1994, Selected Papers*, volume 933 of *Lecture Notes in Computer Science*, pages 471–485. Springer, 1994. doi:10.1007/BFb0022276.
- 23 Patrick Landwehr. *Tree automata with constraints on infinite trees*. Dissertation, RWTH Aachen University, Aachen, 2021. URL: <https://publications.rwth-aachen.de/record/837422>, doi:10.18154/RWTH-2021-12010.

- 24 Yong Li, Yu-Fang Chen, Lijun Zhang, and Depeng Liu. A novel learning algorithm for büchi automata based on family of dfas and classification trees. In *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I*, volume 10205 of *Lecture Notes in Computer Science*, pages 208–226, 2017. doi:10.1007/978-3-662-54577-5_12.
- 25 Damian López and Pedro García. On the inference of finite state automata from positive and negative data. In Sempere J. In: Heinz J., editor, *Topics in Grammatical Inference*. Springer, 2016. doi:10.1007/978-3-662-48395-4_4.
- 26 Oded Maler and Amir Pnueli. On the learnability of infinitary regular sets. *Inf. Comput.*, 118(2):316–326, 1995. doi:10.1006/inco.1995.1070.
- 27 Oded Maler and Ludwig Staiger. On syntactic congruences for ω -languages. *Theoretical Computer Science*, 183(1):93–112, 1997.
- 28 Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. Strix: Explicit reactive synthesis strikes back! In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, pages 578–586, 2018. doi:10.1007/978-3-319-96145-3_31.
- 29 Jakub Michaliszyn and Jan Otop. Learning deterministic automata on infinite words. In *ECAI 2020 - 24th European Conference on Artificial Intelligence*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 2370–2377. IOS Press, 2020. doi:10.3233/FAIA200367.
- 30 Jose Oncina and Pedro García. Inferring regular languages in polynomial update time. *World Scientific*, 01 1992. doi:10.1142/9789812797902_0004.
- 31 Bernhard Steffen, Falk Howar, and Maik Merten. Introduction to active automata learning from a practical perspective. In *Formal Methods for Eternal Networked Software Systems - 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011, Bertinoro, Italy, June 13-18, 2011. Advanced Lectures*, volume 6659 of *Lecture Notes in Computer Science*, pages 256–296. Springer, 2011. doi:10.1007/978-3-642-21455-4_8.
- 32 Wolfgang Thomas. *Automata on Infinite Objects*, page 133–191. MIT Press, Cambridge, MA, USA, 1991.
- 33 Wolfgang Thomas. Languages, automata, and logic. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Volume 3: Beyond Words*, pages 389–455. Springer, 1997. doi:10.1007/978-3-642-59126-6_7.
- 34 Wolfgang Thomas. Facets of synthesis: Revisiting Church’s problem. In *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures, FOSSACS 2009*, volume 5504 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2009.
- 35 Boris A. Trakhtenbrot and Y.M. Barzdin. *Finite Automata: Behavior and Synthesis*. North-Holland Publishing Company, Amsterdam, 1973.
- 36 Moshe Y. Vardi and Thomas Wilke. Automata: from logics to algorithms. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 629–736. Amsterdam University Press, 2008.
- 37 Sicco Verwer and Christian A. Hammerschmidt. flexfringe: A passive automaton learning package. In *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017*, pages 638–642. IEEE Computer Society, 2017. URL: <http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=8090480>.
- 38 Thomas Wilke. ω -automata. *CoRR*, abs/1609.03062, 2016. URL: <http://arxiv.org/abs/1609.03062>, arXiv:1609.03062.