TheoretiCS

Constructing Deterministic Parity Automata from Positive and Negative Examples

León Bohn[a] ✉ ⓘ
Christof Löding[a] ✉ ⓘ

a 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.

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, 36, 18], 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" [19].

Given a class C of regular languages, a learner is said to learn every language in C in the limit, if for each language $L \in C$ 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 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 [38].

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 if and only if 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 [33]), 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, see e.g. [11] or [12, Fact 1].

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). A passive learner for DPAs receives a sample $S = (S_+, S_-)$ of positive and negative ultimately periodic example words as input, and returns a DPA that is consistent with the sample, that is, it accepts all words from S_+ and rejects all words from S_-. We are interested in such a passive learner that can learn a DPA for every regular ω-language in the limit. Without further requirements, such an algorithm is fairly easy to obtain by simple enumeration: Iterate through all DPAs by increasing size and some lexicographic ordering for DPAs of same size, and output the first DPA that is consistent with S. This algorithm is easily seen to infer a smallest DPA for each regular ω-language L in the limit: A characteristic sample for L only needs to contain example words that separate L from each language L' that is accepted by a DPA that precedes the first DPA for L in the enumeration used by the algorithm. However, it is fairly obvious that the running time of such an algorithm is exponential. And it is known that already for DFAs, the minimum automaton identification problem is NP-hard [18] (follows also from [31]). This easily transfers to DPAs.

So, as already proposed by Gold [18], the time and data requirements of a passive learner are of interest. More precisely, the time complexity of a passive learner is its running time measured in the size of the input sample. A polynomial time passive learner constructs a DPA that is consistent with the input sample S in time polynomial in the size of S. Note that this property is independent of the learning-in-the-limit property, so it is not related to the size of characteristic samples for identifying a language. This is what the data requirement is about: A passive learner for DPAs identifies a class C of regular ω-languages from polynomial data if for each $L \in C$ there is a characteristic sample S^L for L whose size is polynomial in the size of a smallest DPA for L. (Where, as for DFAs, S^L is called characteristic for the passive learner and L if the passive learner returns the same DPA for L for each sample S that is consistent with L and contains all examples from S^L.)

Currently, no passive learner is known that runs in polynomial time and learns every regular ω-language in the limit from polynomial data. In fact, all passive learners for deterministic ω-automata that have been presented so far only learn subclasses of the regular ω-languages in the limit (see related work at the end of this introduction for more details). We present, to the best of our knowledge, the first polynomial time passive learner for deterministic ω-automata that can learn every regular ω-language in the limit. Furthermore, although we can only show an exponential upper bound on the data requirement of our algorithm in general, the data

requirement of our algorithm is polynomial for the subclasses of the regular ω-languages for which learning in the limit from polynomial data is already known (see Corollary 5.8).

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. This analysis yields what we call the precise family of weak priority mapping (precise FWPM), which is a family of mappings that assign priorities to finite words. We give a construction that joins this family of mappings into a single one. The minimal Mealy machine computing this join mapping corresponds to the precise DPA.

2. We show how the precise DPA for L can be constructed from the syntactic family of right congruences (FORC) of L [27] by showing that a family of Mealy machines for the precise FWPM can be obtained in polynomial time by assigning priorities to the classes of the syntactic FORC. From the Mealy machines for the precise FWPM one then obtains a construction of the precise DPA that 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. This improves the known upper bound for the construction of a DPA for a language from its syntactic FORC that was known from [2], which first builds a nondeterministic Büchi automaton whose size is polynomial in the size of the FORC, and then determinizes this Büchi automaton. Because of the last step, this construction is exponential in the size of the FORC.

3. We present a polynomial time passive learner for DPAs (an 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 [10], 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, 10]. For this purpose, we propose a generic algorithm GLeRC for inferring right congruences from samples. This algorithm is parameterized by a consistency function that is invoked in order to check if a merge (an inserted transition) causes an inconsistency with the sample. This algorithm can be instantiated in different settings and allows us to give a uniform presentation of passive learners for different right congruences and their completeness proofs for the learning in the limit property.

   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 of the precise DPA from the FORC is exponential, if it is only applied to the example words, then it induces a priority

           assignment of polynomial size. This is important because we cannot simply apply an exponential construction if we want to obtain a polynomial time passive learner.

      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. This step extends the priority mapping that is computed in the previous step on the prefixes of the example words to the set of all finite words. Again, this step is done in order to bypass the direct construction of the DPA from the FORC, which is exponential.

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, generalizing the currently known results on passive learning of DPAs with polynomial time and data from [4, 8, 10].

The paper is structured as follows. We continue this introduction with a discussion on related work. 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 speaking, the algorithm collects some ultimately periodic example words of the form uv^ω from the given NBA, and subsequently 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 its 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 a 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, [10] 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 (which is the number of states) 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 (FDFA 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]). FDFAs are very similar to the $L_\$$-representation by DFAs. In FDFAs, 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 FDFA 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) FDFA can be exponentially more succinct than a minimal DFA for $L_\$$ [3, Theorem 2] (the statement in [3] is for the periodic FDFA, which is almost the same as a minimal DFA for $L_\$$).

However, for FDFAs there is the same difficulty as for the $L_\$$ representation: An FDFA might accept some decompositions of an ultimately periodic word and reject others. An FDFA is called saturated if for each ultimately periodic word it accepts all relevant decompositions or rejects all of them.

The algorithm in [24] uses an active FDFA learning algorithm as presented in [3]. But [3] requires a teacher that can take a hypothesis in form of a general FDFA (i.e. one that is not necessarily saturated)[1]. 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 from [17, 24] 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. We are not aware of any passive learners for saturated FDFAs. And the generic use of an active learner in a passive setting as described in [8] does not work for the active FDFA learning algorithm from [3] because it works with general FDFAs, not only saturated ones, and the best known algorithm for checking whether an FDFA is saturated is in PSPACE [2]. So 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 addition-

---

1   The presentation of [3] contains an error because it assumes that the FDFAs used in the learning algorithm are saturated and hence define an ω-language, which is not correct. So the algorithm only works with a teacher for FDFAs instead of ω-languages.

ally 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., [35, 37, 39] 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 $\mathrm{Prf}(w)$ to refer to the set of all prefixes of w, and for $X \subseteq \Sigma^\infty$, we write $\mathrm{Prf}(X)$ for the union of all $\mathrm{Prf}(x)$ 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 $\sim_1 \le \sim_2$ 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 a right congruence $\sim$ is the number of its classes. In the following, we tacitly assume that all considered right congruences 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 $\delta : Q \times \Sigma \to Q$ and an initial state $\iota \in Q$. We define $\delta^* : Q \times \Sigma^* \to Q$ 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 $w = a_0 a_1 \ldots \in \Sigma^\infty$ from $q_0 \in Q$ is a (possibly infinite) sequence $q_0 q_1 \ldots$ with $q_{i+1} = \delta(q_i, a_i)$. We write $\mathrm{Inf}_\mathcal{T}(w)$ for the set of states that occur infinitely often in the run of $\mathcal{T}$ on $w \in \Sigma^\omega$, and $\mathrm{Inf}_\mathcal{T}(X)$ for the union of $\mathrm{Inf}_\mathcal{T}(w)$ for $w \in X$. A *strongly connected component (SCC)* of $\mathcal{T}$ is a maximal strongly connected set (with the usual definition). We write $\mathrm{SCC}_\mathcal{T}(q)$ for the SCC of $\mathcal{T}$ that contains q. A *partial TS* is a TS in which δ is a partial function.

A right congruence $\sim$ 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 $u \sim_\mathcal{T} v$ if and only if $\mathcal{T}(u) = \mathcal{T}(v)$. So we interchange these objects and often just speak of the TS $\sim$, meaning the TS $\mathcal{T}_\sim$.

For a right congruence $\sim$ 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 [35, 37, 39]). We are interested in (transition-based) deterministic parity automata defined below.

A *priority mapping* is a function $\pi : \Sigma^+ \to \{0, \dots, k-1\}$ 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 $\overline{\pi} = (\pi_c)_{c \in [\sim]}$ 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 $\kappa : Q \times \Sigma \to \{0, \dots, k-1\}$ mapping transitions in $\mathcal{A}$ to priorities. The size of $\mathcal{A}$ is the number of states, and we denote it by $|\mathcal{A}|$. We overload notation and denote the priority mapping induced by a DPA as $\mathcal{A} : \Sigma^+ \to \{0, \dots, k-1\}$ which is defined as $\mathcal{A}(ua) := \kappa(\delta^*(u), a)$ 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 right congruence 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 $L \subseteq \Sigma^\circ$ for $\circ \in \{*, \omega\}$ is the right congruence 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. If $\sim$ is a right congruence 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 $\mathrm{pc}(L)$, 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 $\kappa' : Q \times \Sigma \to \mathbb{N}$ 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 2.1. *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 some word $v \in \Sigma^*$ with $\delta^*(q, uv) = q$ and $\mathcal{A}_{\min}(q, uv) = i$, or $i = 0$ and $SCC_{\mathcal{A}}(\delta^*(q, u)) \cap SCC_{\mathcal{A}}(q) = \emptyset$.*
> *(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 $SCC_{\mathcal{A}'}(q)$ and $\delta(q_1, a)$ is not in $SCC_{\mathcal{A}'}(q)$. 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), $SCC_{\mathcal{A}''}(q)$ contains a loop and thus at least one transition. If $SCC_{\mathcal{A}''}(q)$ 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 $SCC_{\mathcal{A}''}(q)$ by 2. Since $i \geq 2$ and no transition in $SCC_{\mathcal{A}''}(q)$ 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 $X \cap SCC_{\mathcal{A}''}(q) = \emptyset$, the smallest priority in X is the same in $\mathcal{A}$ and $\mathcal{B}$. If $X \subseteq SCC_{\mathcal{A}''}(q)$, then the minimal priority in X differs by exactly 2 in $\mathcal{A}$ and $\mathcal{B}$. Otherwise, X contains a transition that exits $SCC_{\mathcal{A}''}(q)$, and hence has priority $\leq i - 2$. Since all transitions inside $SCC_{\mathcal{A}''}(q)$ 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 $\mathcal{A}$ if and only if 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 [32] 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, Fact 1]. 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 C of regular ω-languages we say that f can learn every language in C in the limit *from polynomial data* if the characteristic samples for the languages in 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 3.18). 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 with regard to $\sim$ (up to Lemma 3.12). Then we show how to combine them into a single priority mapping (Definition 3.14, Lemma 3.16), and how to build a DPA for this combined priority mapping (Lemma 3.17). We finish the section with an observation on precise DPAs, Lemma 3.21, which later on plays a crucial role in achieving polynomial runtime of our learner.

To get an intuition for the family of priority mappings based on which we later define the precise DPA for a language, we fix a regular ω-language L and a right congruence $\sim$ that refines $\sim_L$. Our goal is to define a priority mapping for each $\sim$-class c which assigns priorities to all finite non-empty words, such that it correctly classifies all words v that loop on c, in the sense that the priority assigned to v should be even if and only if $v^\omega \in L_c$. Additionally, we want the mapping to be weak, that is, non-increasing with growing length of the words. The intuition for this is that the priority of a finite word v should reflect the information that is contained in v with respect to periods that loop on c and start with v. In Example 3.2 below that illustrates the formal definition, the language L contains all words with finitely many b or infinitely many aba. Clearly, if v contains the pattern aba, then this is the "maximal" possible information: all periodic words that start with v are in the language. So the priority that is assigned to such v is 0. If v contains b but not aba, then all periodic words that start with v are either not in the language or contain aba. We assign priority 1 to such words, corresponding to the information that a periodic word starting with v might be not in L, or it has a prefix that already contains the maximal information, i.e. it is assigned priority 0. Finally, if v does not contain b, then a periodic word that starts with v either is in L because it contains b finitely often, or it has a prefix that is already assigned a priority 0 or 1. We assign 2 to such v.

More formally, to determine the priority for a word $v \in \Sigma^+$ in the context of a class c of $\sim$, we proceed inductively as follows (recall that L_c denotes the residual of L from class c, and E_c denotes the set of all finite words that loop on c). If there exists no x such that $vx \in E_c$, the word v is assigned priority 0 since no extension of v can ever loop on class c and hence the priority of v is irrelevant in the context of c. Thus, in the following, we only need to consider words v that actually have some extension x such that vx loops on c, and we write $v^{-1}E_c$ to denote the set of all such extensions. Now if for all $x \in v^{-1}E_c$ holds that $(vx)^\omega \in L_c$, then we assign 0 to v because regardless of the suffix we append to v, the ω-iteration is in L_c. Then, priority 1 is assigned to words v such that v is not assigned 0, and for all extensions $x \in v^{-1}E_c$ holds that either $(vx)^\omega \notin L_c$, or $(vx)^\omega$ has a prefix that is assigned priority 0. From here on for priorities $i > 1$, we proceed inductively until a priority has been identified for each word.

The definition we obtain from this procedure 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. We will now give the formal definition of a family of sets $P_{c,i}^{\sim}(L)$ indexed by a class c of $\sim$ and a natural number i. The set $P_{c,i}^{\sim}(L)$ contains all words which are assigned priority i or smaller in the priority mapping for class c. The priority mapping itself, which assigns to v the smallest i such that v is in $P_{c,i}$ is then extracted in Definition 3.5.

DEFINITION 3.1. 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

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

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

One easily verifies that for a fixed class c, the sets $P_{c,i}^{\sim}(L)$ form an inclusion chain where a larger index i leads to a larger set. This is because the first condition in the definition of $P_{c,i}^{\sim}(L)$ is trivially satisfied by all words belonging to $P_{c,i-1}^{\sim}(L)$. For a class c, we can now define a priority mapping that assigns to a finite word v the least priority such that $v \in P_{c,i}^{\sim}(L)$. A formal definition is deferred to Definition 3.5 below, until after we have showed that the obtained inclusion chain is guaranteed to be finite whenever L is a regular ω-language and $\sim$ refines $\sim_L$.

EXAMPLE 3.2. 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, i.e.

$$L = \{w \in \{a,b\}^{\omega} \mid w \text{ contains } b \text{ finitely often or } w \text{ contains } aba \text{ infinitely often}\}.$$

As $\sim$ we take $\sim_L$, which has only one class (adding or removing finite prefixes is irrelevant for membership in L). This also implies that the universally quantified x in Definition 3.1 ranges over all finite words. So a word v belongs to $P_{\varepsilon,0}^{\sim}(L)$ if for any possible extension $x \in \Sigma^*$ holds that $(vx)^{\omega} \in L_{\varepsilon}$. This is satisfied precisely by those words that contain the infix aba.

Consider now a word v that contains the letter b and let $x \in \Sigma^*$, then clearly $(vx)^{\omega}$ is guaranteed to contain infinitely many b. If $(vx)^{\omega}$ also contains infinitely many aba, then it certainly has a prefix in $P_{\varepsilon,0}^{\sim}(L)$, which means it satisfies the first condition. Otherwise, $(vx)^{\omega} \notin L_{\varepsilon}$ and the second condition is satisfied. So overall, $P_{\varepsilon,1}^{\sim}(L)$ consists of all words that have an occurrence of b.

Finally, $P_{\varepsilon,2}^{\sim}(L)$ consists of all finite, non-empty words over Σ. Indeed, either $(vx)^{\omega}$ contains b infinitely often, in which case $(vx)^{\omega}$ has a prefix in $P_{\varepsilon,1}^{\sim}(L)$, or there are only finitely many occurrences of b, implying $(vx)^{\omega} \in L_{\varepsilon}$. The languages $P_{\varepsilon,i}^{\sim}(L)$ for $i \leq 3$ are given in the second column of the table in Table 1.   ◆

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

Table 1. An overview over languages arising from Definition 3.1 for the languages L and L' from Example 3.2 and Example 3.3, respectively.
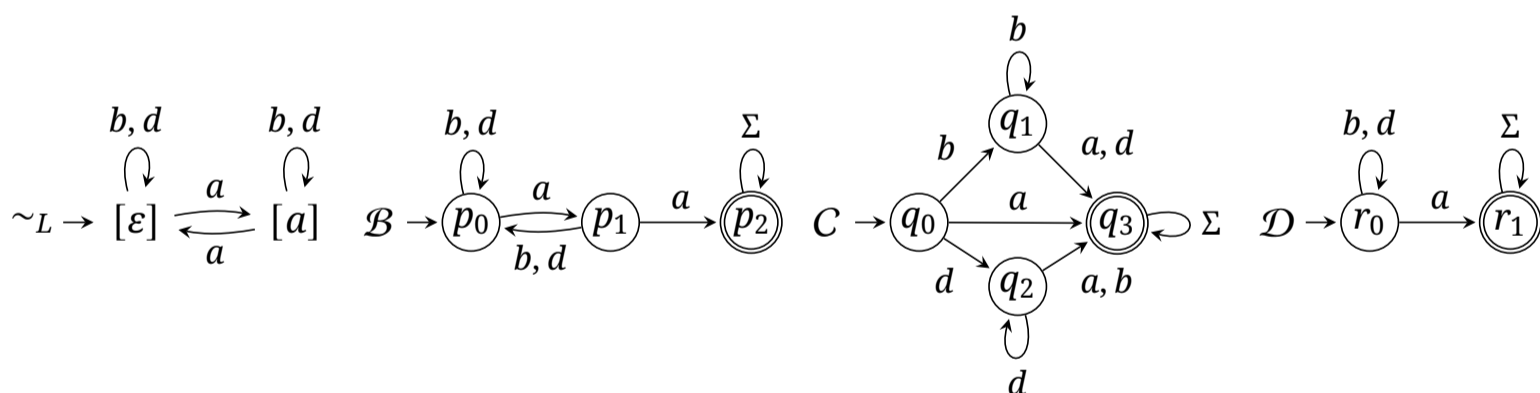
EXAMPLE 3.3. Let $\Sigma = \{a, b, d\}$ and consider the language

$$L' = \{w \in \Sigma^\omega \mid \text{there are infinitely many infixes } aa \text{ in } w\}$$
$$\cup \{w \in \Sigma^\omega \mid |w|_a \text{ is finite and } (\text{even iff } w \text{ contains infinitely many } b \text{ and } d)\}.$$

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 1 alongside DFAs recognizing the sets $P^{\sim}_{\varepsilon,i}(L')$ and $P^{\sim}_{a,i}(L')$ for $i \leq 3$.

Regardless of the class c we consider, if a word $v \in \Sigma^+$ contains an infix aa, then for every possible $x \in v^{-1}E_c$, we have $(vx)^\omega \in L'_c$ because any such ω-iteration must contain infinitely many occurrences of aa. Hence, the sets $P^{\sim}_{\varepsilon,0}(L')$ and $P^{\sim}_{a,0}(L')$ are equal and contain precisely those words that have an infix aa. Similarly, we see that if v contains a, then either $(vx)^\omega$ contains the infix aa infinitely often or $(vx)^\omega$ does not belong to L_c. Thus, $P^{\sim}_{\varepsilon,1}(L')$ and $P^{\sim}_{a,1}(L')$ contain all words with an infix a.

Consider now a word v that contains both b and d, but not a. Such words appear at different indices depending on the class c. We have $v \in P^{\sim}_{a,1}(L')$ since for each $x \in v^{-1}E_a$, the ω-iteration $(vx)^\omega$ is not in L_a unless $(vx)^\omega$ contains aa infinitely often. However, $v \notin P^{\sim}_{\varepsilon,1}(L')$ since already v^ω has no prefix in $P^{\sim}_{\varepsilon,1}(L')$ and $v^\omega \in L_\varepsilon$. So v is in $P^{\sim}_{\varepsilon,2}(L')$. Finally, the sets $P^{\sim}_{\varepsilon,3}(L')$, $P^{\sim}_{a,2}(L')$ and $P^{\sim}_{a,3}(L')$ coincide and contain all non-empty finite words over Σ. The sequences of sets for both classes are depicted in the last two columns of the table in Table 1.   ◆

Figure 1.   The leading right congruence underlying the language L from Example 3.3 is depicted on the far left. We have $L(\mathcal{B}) = P^{\sim_L}_{0,\varepsilon}(L) = P^{\sim_L}_{0,a}(L)$, $L(\mathcal{C}) = P^{\sim_L}_{2,\varepsilon}(L) = P^{\sim_L}_{1,a}(L)$ and $L(\mathcal{D}) = P^{\sim_L}_{1,\varepsilon}(L)$. We omit the depiction of a DFA for $P^{\sim_L}_{3,\varepsilon}(L) = P^{\sim_L}_{2,a}(L) = \Sigma^+$.

Note that all sets $P_{c,i}$ arising from the given examples are regular. In Example 3.2 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 $P^\sim_{c,i}(L)$ and the priorities visited by a normalized DPA $\mathcal{A}$ for L, namely, $P^\sim_{c,i}(L)$ 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 3.4. *Let $\mathcal{A}$ be a normalized DPA with priorities $\{0, \ldots, k-1\}$ recognizing the language $L \subseteq \Sigma^\omega$ and $\sim$ be a right congruence with $\sim_{\mathcal{A}} \le\ \sim\ \le\ \sim_L$. Then for each class c of $\sim$ and each $i \ge 0$ holds $P^\sim_{c,i}(L) = P^\sim_{c,i}(\mathcal{A}) := \{u \in \Sigma^+ \mid \max\{\mathcal{A}_{\min}(q, u) \mid q \text{ is } c\text{-state}\} \le i\}$.*

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

We first show $P^\sim_{c,i}(L) \subseteq P^\sim_{c,i}(\mathcal{A})$ by establishing that for all $u \in P^\sim_{c,i}(L) \setminus P^\sim_{c,i-1}(L)$ holds $u \in P^\sim_{c,i}(\mathcal{A})$. Let q be a c-state, and let $i' := \mathcal{A}_{\min}(q, u)$. We have to show that $i' \le i$, 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 2.1). Then no prefix of $(ux)^\omega$ is in $P^\sim_{c,i}(\mathcal{A})$ because the minimal priority seen from q on any prefix of $(ux)^\omega$ is a least $i' > i$. Since $P^\sim_{c,i}(\mathcal{A}) \supseteq P^\sim_{c,i-1}(\mathcal{A})$, also no prefix of $(ux)^\omega$ is in $P^\sim_{c,i-1}(\mathcal{A})$. By induction, it follows that no prefix of $(ux)^\omega$ is in $P^\sim_{c,i-1}(L)$, and thus $(ux)^\omega \in L_c$ if and only if i is even. Since $(ux)^\omega$ is accepted from the c-state q if and only if i' is even, we obtain that i is even if and only if 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 2.1). Then $(uxy)^\omega$ is accepted from q if and only if $i' - 1$ is even if and only if i is odd. Since u is in $P^\sim_{c,i}(L)$, we obtain that $(uxy)^\omega$ has a prefix in $P^\sim_{c,i-1}(L)$. This implies by induction that $(uxy)^\omega$ has a prefix in $P^\sim_{c,i-1}(\mathcal{A})$, contradicting the fact that the minimal priority that is seen from q on $(uxy)^\omega$ is $i' - 1 > i$.
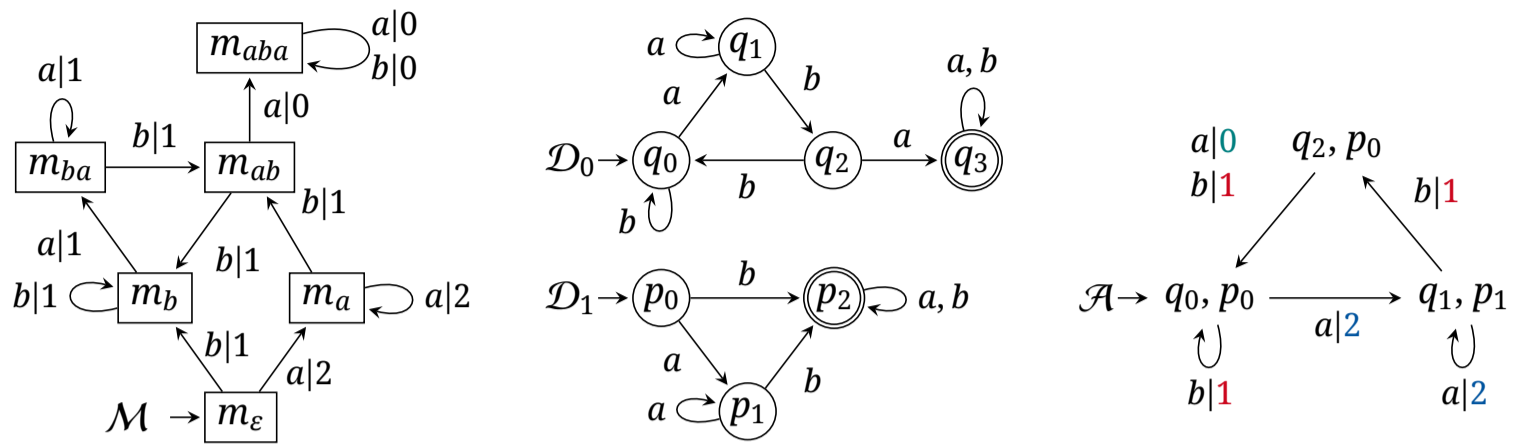
To establish the other inclusion $P^\sim_{c,i}(\mathcal{A}) \subseteq P^\sim_{c,i}(L)$, we show for all $u \in P^\sim_{c,i}(\mathcal{A}) \setminus P^\sim_{c,i-1}(\mathcal{A})$ that $u \in P^\sim_{c,i}(L)$. So let $x \in \Sigma^*$ with $ux \in E_c$, and let $n > |\mathcal{A}|$. Since $u \in P^\sim_{c,i}(\mathcal{A})$, also $(ux)^n \in P^\sim_{c,i}(\mathcal{A})$.

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

If $(ux)^n \in P^\sim_{c,i}(\mathcal{A}) \setminus P^\sim_{c,i-1}(\mathcal{A})$, 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 $n_1 + n_2 \le n$ such that $\mathcal{A} : q \xrightarrow{(ux)^{n_1}} q' \xrightarrow{(ux)^{n_2}} q'$. Since $u \in P^\sim_{c,i}(\mathcal{A})$, 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 $(ux)^\omega \in L_c$ if and only if i is even.

We conclude that $u \in P^\sim_{c,i}(L)$. ∎

We already noted above that the sets $P^\sim_{c,i}(L)$ form an inclusion chain. As consequence of Lemma 3.4, we obtain that this inclusion chain is of length k whenever L is a regular ω-

Figure 2. On the left-hand side a Mealy machine $\mathcal{M}$ that computes $\pi_\varepsilon^\sim$ for L and $\sim$ from Example 3.2 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.

language of parity complexity k (the minimal number of priorities a DPA for L needs is k). So $P_{c,0}^\sim(L) \subseteq \cdots \subseteq P_{c,k-1}^\sim(L) = \Sigma^+$ and each set along the chain is regular. This implies that the priority mappings we obtain from the following definition are totally defined, weak, and can be computed by Mealy machines.

DEFINITION 3.5. With the terminology from Definition 3.1 and L of parity complexity k, we represent the sets $P_{c,i}^\sim(L)$ for each class by a priority mapping $\pi_c^\sim : \Sigma^+ \to \{0, \ldots, k-1\}$ 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 $\sim$*.

With L and $\sim$ as in Example 3.2, 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 2 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 3.4.

THEOREM 3.6. *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 $\pi_c^\sim$ 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 3.4, 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 $\tau : Q_c \to Q$ and $\mu : Q_c \to \{0, \ldots, k-1\}$, 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 3.4 and Definition 3.5. ∎

The following definition expresses what it means for an FWPM to correctly capture the words that loop on the classes of the underlying right congruence $\sim$.

DEFINITION 3.7. We say (with the terminology from Definition 3.1) 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 $v^\omega \in L_c$ if and only if $v^\omega \in L(\gamma_c)$.

LEMMA 3.8. *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 $(L, \sim)$.*

PROOF. 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 $\pi_c^\sim$ 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 $((v^n)^\omega \in L_c$ if and only if 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 n. Hence, the second case must be true, which is what we set out to show. ∎

The following technical lemma is only used to establish the result in Lemma 3.10 that gives justification for the name "precise FWPM". As it is not required for the other results of this paper, readers may safely skip to after Lemma 3.10.

LEMMA 3.9. *Let $\overline{\pi^\sim}$ be as in Definition 3.5, c be a class of $\sim$ and $v \in \Sigma^+$. If $\pi_c^\sim(v) = i \geq 1$, then there exists an $x \in \Sigma^*$ such that $vx \in E_c$ and $\pi_c^\sim((vx)^\omega) = i$. Moreover, if $\pi_c^\sim(v) = i \geq 2$, then there exists $y \in \Sigma^*$ with $vy \in E_c$ and $\pi_c^\sim((vy)^\omega) = i - 1$.*

PROOF. We show the first part of the claim. Let c be a class of $\sim$, $v \in \Sigma^+$ and $\pi_c^\sim(v) = i \geq 1$. Since $v \notin P_{c,i-1}^\sim(L)$, there exists an $x \in \Sigma^*$ such that $vx \in E_c$, $((vx)^\omega \in L_c$ if and only if $i - 1$ is odd), and $\mathrm{Prf}((vx)^\omega) \cap P_{c,i-2}^\sim(L) = \emptyset$ (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 3.8, $\overline{\pi^\sim}$ 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 $(vy)^\omega \in L_c$ if and only if $i - 2$ 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 $\pi_c^\sim(v) = i$. ∎

We now give the result that can be seen as justification for the name "precise FWPM". It shows that in a certain sense, each mapping $\pi_{\tilde{c}}$ of the precise FWPM provides the most accurate information with regard to words that loop on c. For this, we partially order FWPMs by point-wise comparison, that is, for two FWPMs $\overline{\gamma} = (\gamma_c)_{c \in [\sim]}$ and $\overline{\gamma}' = (\gamma'_c)_{c \in [\sim]}$ we let $\overline{\gamma} \leq \overline{\gamma}'$ if for all classes c of $\sim$ and $u \in \Sigma^+$ holds $\gamma_c(u) \leq \gamma'_c(u)$. The following lemma now establishes that with regard to this order, no smaller FWPM than the precise FPWM correctly captures the periodic part of $(L, \sim)$.

LEMMA 3.10. *For a regular ω-language L and a right congruence $\sim$ that refines $\sim_L$, the family $\overline{\pi^{\sim}} = (\pi_{\tilde{c}})_{c \in [\sim]}$ is the unique least FWPM that captures the periodic part of $(L, \sim)$.*

PROOF. Let $\overline{\gamma}$ be an FWPM which captures the periodic part of $(L, \sim)$. We show $\overline{\pi^{\sim}} \leq \overline{\gamma}$. Let $v \in \Sigma^+$ be an arbitrary word with $\gamma_c(v) = i$. We show by induction on i that $\pi_{\tilde{c}}(v) \leq i$. 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 $\overline{\gamma}$ captures the periodic part of $(L, \sim)$. Thus, $v \in P_{c,0}^{\sim}$ and therefore $\pi_{\tilde{c}}(v) = 0$.

For an $i > 0$, we assume towards a contradiction that $\pi_{\tilde{c}}(v) = j > i$, implying $j \geq 2$. By Lemma 3.9, there exists some $x \in \Sigma^*$ such that $vx \in E_c$ and $\pi_{\tilde{c}}((vx)^{\omega}) = j$. Further, it must be that $\gamma_c((vx)^{\omega}) = i$, as otherwise if $\gamma_c((vx)^{\omega}) < i$, it would follow by induction that $\pi_{\tilde{c}}((vx)^{\omega}) < i$. This means j is even if and only if i is even and thus $j - 1 > i$.
Using Lemma 3.9 once more, we pick some $y \in \Sigma^*$ with $vxy \in E_c$ and $\pi_{\tilde{c}}((vxy)^{\omega}) = j - 1$. Again, by induction $\gamma_c((vxy)^{\omega}) = i$ and hence $(vxy)^{\omega} \in L(\gamma_c)$ if and only if $(vxy)^{\omega} \notin L(\pi_{\tilde{c}})$. This contradicts our assumption that both $\overline{\gamma}$ and $\overline{\pi^{\sim}}$ capture the periodic part of $(L, \sim)$. Hence, $\pi_{\tilde{c}}(v) \leq i$, concluding the proof. ∎

The precise DPA for L and $\sim$ that we introduce below is derived from the 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 3.11. We call an FWPM $\overline{\gamma} = (\gamma_c)_{c \in [\sim]}$ *monotonic* if $\gamma_u(vx) \leq \gamma_{uv}(x)$ for all $u, v \in \Sigma^*, x \in \Sigma^+$.

LEMMA 3.12. *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 3.4, 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 3.4, $\pi_{\tilde{u}}(vx) = \max\{\mathcal{A}_{\min}(q, vx) \mid q \in Q_c\}$. 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

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

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 $\gamma_{\bowtie}$ that defines L. We refer to $\gamma_{\bowtie}$ as the *join of $\overline{\gamma}$*.

Recall that it is sufficient if $\gamma_{\bowtie}$ works correctly on ultimately periodic words. Since $\overline{\gamma}$ captures the periodic part of L, for all uv^ω such that $uv \sim u$ we have that $\gamma_c(v^\omega)$ is even if and only if $v^\omega \in L_c$, where c is the $\sim$-class reached by u. The main problem when defining $\gamma_{\bowtie}$ for an input $p \in \Sigma^+$ is that we only know the prefix p but not the ultimately periodic word uv^ω that it is a prefix of.

Intuitively, the idea for computing the join for an input p is to consider all possible factorizations $uv = p$ and their associated priorities $\gamma_u(v)$. Of those, the most significant one (i.e. the smallest) then determines the priority of $\gamma_{\bowtie}(p)$. Naively defining the join based on this, however, would just lead to the weak priority mapping for the initial class, since $\overline{\gamma}$ is monotonic. Instead, the join ensures that all these values $\gamma_u(v)$ are *covered* by the priority sequence on p, where $\gamma_u(v)$ is covered if $\gamma_{\bowtie}$ emits a priority less than or equal to $\gamma_u(v)$ on the v-part of the decomposition $p = uv$. Obviously, with this definition, it is possible to always emit priority 0. This clearly will cover all values $\gamma_u(v)$, but it will certainly not lead to a correct priority mapping for L. Instead, the join always picks the least significant priority that is necessary to ensure that all values are covered. Assume that the whole input is $w = uv^\omega$ with $uv \sim u$ and $\gamma_u(v^\omega) = i$ where priority i is already assumed after one iteration of v, so $\gamma_u(v) = i$. This means that $i = \gamma_u(v) = \gamma_{uv}(v) = \gamma_{uvv}(v) \cdots$. Then the join will infinitely often emit a priority $\leq i$ in order to cover all these values. Based on the monotonicity of $\overline{\gamma}$ we can show in Lemma 3.15 that indeed i will be the dominating priority that is emitted by $\gamma_{\bowtie}$.

In the following example, we further try to illustrate the idea of the join, before subsequently formalizing it in Definition 3.14.

EXAMPLE 3.13. Consider the FWPM $(\pi_c)_{c \in [\sim]}$ from Example 3.2 which is displayed on the left of Figure 3: $\tilde{\pi_\varepsilon}$ assigns 0 to a word $v \in \Sigma^+$ precisely if v contains the infix aba, 1 if v contains a b but no aba and 2 otherwise. Further, ε is the only class. The table on the right of Figure 3 visualizes the computation of the join for all prefixes of the word $p = abaaba$. The i-th letter of this word is denoted by p_i. As explained before, to compute $\tilde{\pi_{\bowtie}}$ for some prefix $p_0 \cdots p_i$ of p, we have to consider all values $\tilde{\pi}_{p_0 \ldots p_{j-1}}(p_j \ldots p_i)$, which are given in the column for the

$$\pi_{\tilde{\varepsilon}}(u) = \begin{cases} 0 & \text{if } u \text{ contains } aba \\ 1 & \text{if } u \text{ contains } b \text{ but not } aba \\ 2 & \text{otherwise} \end{cases}$$

i	0	1	2	3	4	5
p_i	a	b	a	a	b	a
$\pi_{\tilde{\varepsilon}}(p_0 \dots p_i)$	$\underline{2}$	$\underline{1}$	$\underline{0}$	0	0	0
$\pi_{\tilde{p_0}}(p_1 \dots p_i)$		$\underline{1}$	1	1	1	0
$\pi_{\tilde{p_0 \dots p_1}}(p_2 \dots p_i)$			$\underline{2}$	2	1	0
$\pi_{\tilde{p_0 \dots p_2}}(p_3 \dots p_i)$				$\underline{2}$	$\underline{1}$	$\underline{0}$
$\pi_{\tilde{p_0 \dots p_3}}(p_4 \dots p_i)$					$\underline{1}$	0
$\pi_{\tilde{p_0 \dots p_4}}(p_5 \dots p_i)$						$\underline{2}$
$\pi_{\tilde{\bowtie}}(p_0 \dots p_i)$	2	1	0	2	1	0

Figure 3. On the left, the priority mapping $\pi_{\tilde{\varepsilon}}$ associated with the FWPM $(\pi_c)_{c \in [\sim]}$ from Example 3.2 is shown. The table on the right displays the values of $\pi_{\tilde{p_0 \dots p_{j-1}}}(p_j \dots p_i)$ where i is the length of a prefix of p and j is a possible first position inside the looping part of the guessed ultimately periodic word. In each column, we mark the value(s) that are not yet covered by underlining them.

---

corresponding value of i. Note that the subscript $p_0 \dots p_j$, which denotes the class, does not play a role in this example, because there is only one class. We nevertheless spell it out in the example in order to match the formal definition.

For finding $\pi_{\tilde{\bowtie}}(p_0 \cdots p_i)$, we have to ensure that all the values from the column for the corresponding i are covered. The values that are not covered by a previous value of $\pi_{\tilde{\bowtie}}$ are underlined. For $i = 0$, no priority has been emitted by $\pi_{\tilde{\bowtie}}$, so we just take the maximal value that covers 2, which is 2 itself. For $i = 1$, the column contains two times priority 1, and since up to now only 2 has been emitted by $\pi_{\tilde{\bowtie}}$, we obtain $\pi_{\tilde{\bowtie}}(ab) = 1$. Similarly, because priority 0 has to be covered in the next step, we obtain $\pi_{\tilde{\bowtie}}(aba) = 0$.

Now, for $i = 3$, the values that need to be covered are the four values in the column for $i = 3$. The value 0 is coming from the decomposition $(\varepsilon, abaa)$ in which the periodic part is the whole word $abaa$. This value 0 is already covered by $\pi_{\tilde{\bowtie}}(aba) = 0$. Similarly, $\pi_{\tilde{\bowtie}}(aba) = 0$ covers the values $\pi_{\tilde{a}}(baa) = 1$ and $\pi_{\tilde{ab}}(aa) = 2$. The only value from the column for $i = 3$ that has not yet been covered, is the value $\pi_{\tilde{aba}}(a) = 2$, because $\pi_{\tilde{\bowtie}}$ did not yet emit anything on this periodic part, which only includes the last letter. Hence, $\pi_{\tilde{\bowtie}}(abaa) = 2$.

Let us jump to $i = 6$. For finding $\pi_{\tilde{\bowtie}}(abaaba)$, all values in the last column are considered. The first three 0s are covered by $\pi_{\tilde{\bowtie}}(aba) = 0$ because they correspond to decompositions whose periodic part contains position $i = 2$. The other values have not yet been covered, and hence $\pi_{\tilde{\bowtie}}(abaaba) = 0$. ◆

We are now ready to give the formal definition of the join. Note that since it is defined inductively, we need to reference values of the join that have already been computed for prefixes

of the input. So while the definition speaks of partial priority mappings, the resulting mapping is guaranteed to be total.

DEFINITION 3.14. 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 $\gamma_{\bowtie}$ is defined inductively as follows. Assume that $\gamma_{\bowtie}$ 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 $\gamma_{\bowtie}$ covers $\overline{\gamma}$ on all non-empty suffixes of ua.

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

LEMMA 3.15. *Let $\overline{\gamma} = (\gamma_c)_{c \in [\sim]}$ 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 $\overline{\gamma}$ (Definition 3.11): 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 $\gamma_{\bowtie}$, 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 $\overline{\gamma}$ 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 $\gamma_{\bowtie}$ since all $\gamma_{\bowtie}$-values after u are $\geq i$. We conclude that $\gamma_u(v^\omega) = i$ as desired. ∎

LEMMA 3.16. *Let L be an ω-regular language and $\sim$ refine $\sim_L$. If $\overline{\gamma}$ 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. If 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 $uv^\omega \in L$ if and only if $uv^\omega \in L(\gamma_{\bowtie})$. Consider the sequence of classes $[u], [uv], [uvv], \ldots$. 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 $xy^\omega \in L$ if and only if $y^\omega \in L_x$. Since $y \in E_x$ and $\overline{\gamma}$ captures the periodic part of L, it further holds that $y^\omega \in L_x$ iff $y^\omega \in L(\gamma_x)$. Applying Lemma 3.15 now gives us $\gamma_{\bowtie}(xy^\omega) = \gamma_x(y^\omega)$, from which we immediately conclude that $xy^\omega \in L(\gamma_{\bowtie})$ if and only if $xy^\omega \in L$.

The claim $L(\pi_{\bowtie}^{\smile}) = L$ directly follows from Lemma 3.8 and Lemma 3.12.   ∎

We now explain how to construct a DPA that computes the priority mapping $\gamma_{\bowtie}$ if $\overline{\gamma}$ is a monotonic FWPM given by a family $\overline{\mathcal{M}} = (\mathcal{M}_c)_{c \in [\sim]}$ of Mealy machines. This DPA, which we denote by $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$, 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 $\overline{\gamma}$ 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 $\overline{\gamma}$ is $\{0, \ldots, k - 1\}$. First, we extract for each priority $i \in \{0, \ldots, k - 1\}$ 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 2, 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.
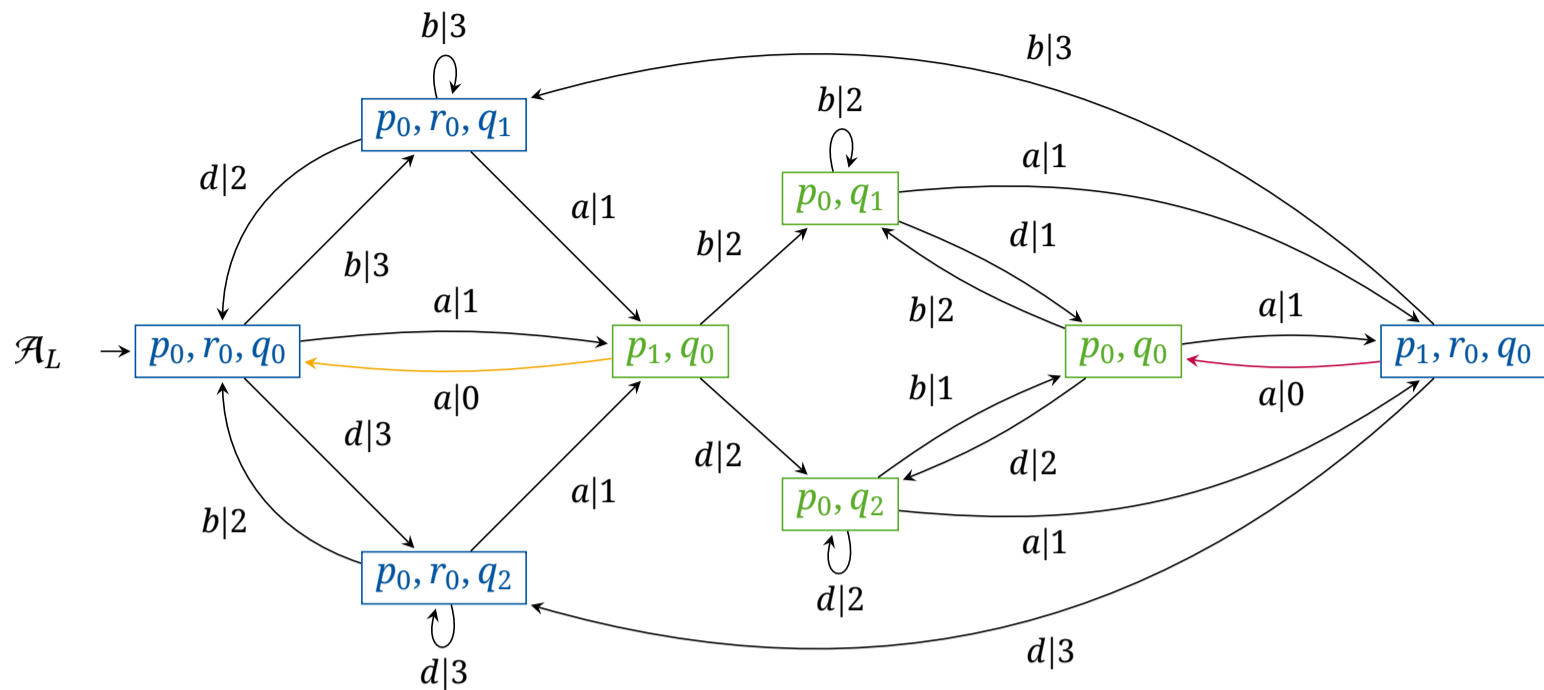
To simplify notation, we assume that the $Q_{c,i}$ are pairwise disjoint, allowing us to write $\delta(q, a)$ and $q \in F$ to refer to $\delta_{c,i}(q, a)$ resp. $q \in F_{c,i}$ for the unique $c \in [\sim]$ and $i < k$ such that $q \in Q_{c,i}$. We now define the DPA $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}}) = (Q_{\bowtie}, \Sigma, \iota_{\bowtie}, \delta_{\bowtie}, \kappa_{\bowtie})$ with

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

In this product, we refer to the i-th component as the *component for priority i* for $i \in \{0, \ldots, k-1\}$. For each transition from some state $\tilde{q} = (q_0, \ldots, q_{k-1}, c) \in Q$ 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.

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

Figure 4. The precise DPA for the language L from example Example 3.3. The DPA is built using the right congruence $\sim_L$ and DFAs for the sets $P^{\sim_L}_{c,i}(L)$ for $i \le 3$ and $d \in [\sim_L]$, which are depicted in Figure 1. 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 1 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).

In Figure 2 the resulting DPA for Example 3.2 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 3.3 is shown in Figure 4.

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

LEMMA 3.17. *If $\overline{M}$ is a family of Mealy machines representing a monotonic FWPM $\overline{\gamma}$, then $\mathcal{A}_{\bowtie}(\overline{M})(u) = \gamma_{\bowtie}(u)$ for each $u \in \Sigma^+$. In particular, $L(\mathcal{A}_{\bowtie}(\overline{M})) = L(\gamma_{\bowtie})$.*

PROOF. We write $\mathcal{A}_{\bowtie}$ for $\mathcal{A}_{\bowtie}(\overline{M})$. 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 $\gamma_{\bowtie}$, and then show that this is precisely what $\mathcal{A}_{\bowtie}(\overline{M})$ computes:

<u>Claim:</u> 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) $\forall z \in \Sigma^+ : z \subsetneqq y \implies \gamma_{\bowtie}(xz) > i$, and

(3) $\gamma_{\bowtie}(x) \leq i$.

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

From the definition of $\gamma_{\bowtie}$ 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 $\overline{\gamma}$ 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 $\gamma_{\bowtie}$ 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 $\leq j$.   ■

We know from Theorem 3.6 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 3.18. 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 $\pi^{\sim}_{\bowtie}$ 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 2 is minimal as a Mealy machine, so it is the precise DPA for the language from Example 3.2. Similarly, the DPA in Figure 4 is the precise DPA for the language from Example 3.3.

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 3.8, and hence $L(\pi^{\sim}_{\bowtie}) = L$ by Lemma 3.16. Since, by definition, the priority mapping of $\mathcal{A}_{L,\sim}$ is $\pi^{\sim}_{\bowtie}$, 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 3.19. *Let $\mathcal{A}$ be a DPA with priorities in $\{0, \ldots, k-1\}$ 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 $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$. In the construction of $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$, 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 $\pi^{\sim}_c(w) \leq i$. From Lemma 3.4 we obtain that $\pi^{\sim}_c(w) \leq i$ 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 $\{\delta(p, a) \mid p \in P$ 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 $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$ are tuples $(P_0, \ldots, P_{k-1}, c)$ 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 $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$, 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 **[10, Proposition 14]**.

For a number $d \geq 1$, consider the alphabet $\Sigma_d = \{a_0, \ldots, a_{d-1}\}$ 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 $\{0, \ldots, d-1\}$ and state set $\{q_0, \ldots, q_{d-1}\}$, the resulting transition function is $\delta(q_h, a_h) = a_{h+1 \mod d}$ 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 $a_2 a_1 a_0$ (for $d = 3$) but on $a_0 a_1 a_2$. 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 therefore cannot simply construct $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$ from the Mealy machines because this would result in an exponential step. To overcome this issue, 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 3.20. *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 $x \in \mathsf{Prf}(v^\omega) \cap L(\mathcal{D})$ with $|x| \leq n \cdot |v|$.*

PROOF. Consider the run $q = q_0 \xrightarrow{v} q_1 \xrightarrow{v} q_2 \xrightarrow{v} \cdots$ 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 3.21. *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 $x_0, x_1 \cdots$ 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 $(p_0, \ldots, p_{k-1}, c)$, 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 3.20 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 $i_0 \leq i_1 \leq \cdots$. 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 $x_0 x_1 \cdots x_{h_1-1}$ 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 $x_0 \cdots x_{h_0-1}$ and s the priority sequence of $\mathcal{A}$ on $x_{h_0} \cdots x_{h_1-1}$. These can be computed on the fly without fully constructing $\mathcal{A}$, so their computation is polynomial in $|uv|$ and $|\overline{\mathcal{M}}|$. ∎
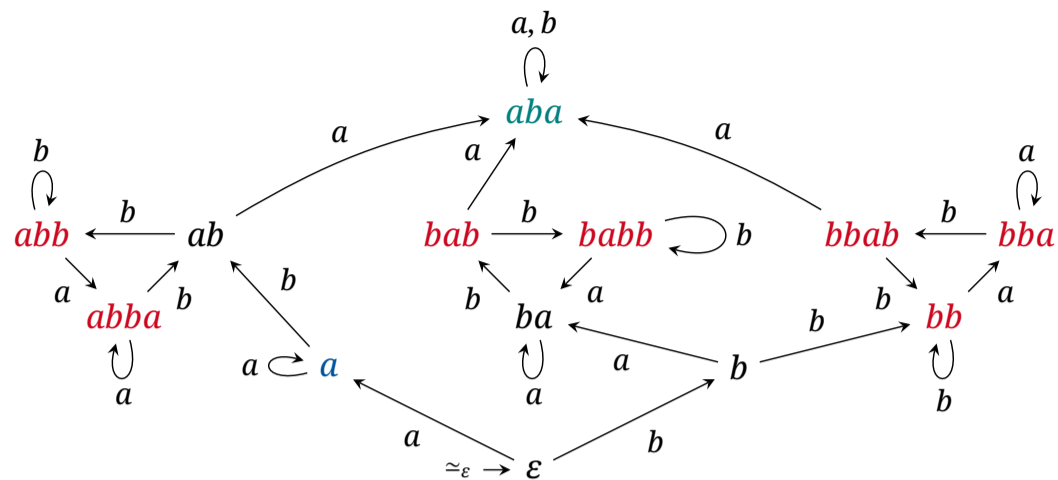
4.   Families of Right Congruences

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 $\overline{\pi}^{\sim} = (\pi_c^{\sim})_{c \in [\sim]}$. 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 $\pi_c^{\sim}$ 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 $\pi_c^{\sim}$ consists of an initial part that assigns 1 to the words in $P_1(L)$, and a sink state that that is reached for all words in $P_0(L)$. So two words x, y lead to different states in this Mealy machine if and only if 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 if and only if $\exists z \in \Sigma^* : (\forall z' \in \Sigma^* : (xzz')^{\omega} \in L) \Leftrightarrow (\exists z' \in \Sigma^* : (yzz')^{\omega} \notin L)$.

  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 $(xzz')^{\omega} \in L \Leftrightarrow (yzz')^{\omega} \notin L$. 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

Figure 5. The PRC $\simeq_\varepsilon$ of the syntactic FORC for the language from Example 3.2. The colors code the idempotent classes that are positive (aba and a) and negative.

$\approx_c$ 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 3.2. 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 2 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 $\pi_c^{\sim}$. 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 $\pi_c^{\sim}$ (see Lemma 4.2). 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 4.1. *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

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

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 $x^n \simeq x^{2n}$. ∎

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 $\kappa_c : [\simeq_c] \to \{0, \ldots, k\}$ 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 $\{0, \ldots, i-1\}$. 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 4.2. *For all $x \in \Sigma^+$: $\kappa_c(x) = i$ if and only if $\tilde{\pi_c}(x) = i$.*

PROOF. The proof goes by induction on i. For the base case, note that $x \in \tilde{P}_{c,0}(L)$ if and only if $(xz)^\omega \in L_c$ 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 $\tilde{\pi_c}(x) = 0$. 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 4.1, which would then be negative).

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

Let $x \in \Sigma^+$. If $\tilde{\pi_c}(x) = i$, 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, $\tilde{\pi_c}(x) = i$ implies that for each $z \in \Sigma^*$ with $xz \in E_c$, either $(xz)^\omega$ has a prefix in $\tilde{P}_{c,i-1}(L)$, or $((xz)^\omega \in L_c$ if and only if i is even). In the first case, if $(xz)^\omega$ has a prefix in $\tilde{P}_{c,i-1}(L)$, then there is an n such that $\tilde{\pi_c}((xz)^n) \leq i-1$ (because $\tilde{\pi_c}$ 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, $((xz)^\omega \in L_c$ if and only if i is even), the class of xz is positive if and only if i is even. Hence, each idempotent class that is reachable from x either has a κ_c value smaller than i or is positive if and only if i is even. Hence $\kappa_c(x) = i$.

For the other direction, assume that $\kappa_c(x) = i$ and show that $\tilde{\pi_c}(x) = i$. By induction, $\tilde{\pi_c}(x) \geq i$, which means that $x \notin \tilde{P}_{c,i-1}$. 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 4.1). Since $\kappa_c(x) = i$, the class of $(xz)^n$ has a κ_c-value less than i, or it is positive if and only if i is even. In the first case, we get by induction that $(xz)^n \in \tilde{P}_{c,i-1}$. In the second case, $((xz)^n)^\omega = (xz)^\omega \in L_c$ if and only if i is even. So $x \in \tilde{P}_{c,i}$ and hence $\tilde{\pi_c}(x) = i$. ∎

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 4.2. We use this structural property of the PRCs in Section 5, and hence state it as a lemma.

LEMMA 4.3. *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 4.2, 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 4.4. *Let L be a regular ω-language of parity complexity k, $\sim$ be a right congruence that refines $\sim_L$, and $(\sim, (\approx_c)_{c \in [\sim]})$ 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 $\approx_c$ by using the transition structure of $\approx_c$ and assigning the κ_c-value of the target state to a transition computes $\pi_c^{\sim}$ according to Lemma 4.2. Denote the resulting family of Mealy machines by $\overline{\mathcal{M}}$. Then $\mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$ computes the priority mapping $\pi_{\bowtie}^{\sim}$ and thus accepts L by Lemma 3.16. Since each $\approx_c$ 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 $\approx_c$. 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.   ■

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.

PROPOSITION 4.5. *Let $\mathcal{A}$ be a normalized DPA with priorities $\{0, \ldots, k-1\}$ 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 $\approx_c$ 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 $\Delta_{c,x} : Q_c \to Q \times \{0, \ldots, k-1\}$ 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 $x \approx_c y$. 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. Furthermore, the learner is designed in such a way that it can infer a DPA for every regular ω-language L, given that the sample contains enough information on L (which is made precise in the proof of Theorem 5.7).

Throughout this section, we assume $S = (S_+, S_-)$ is a finite ω-sample of ultimately periodic words. The overall idea of the algorithm is to first infer a FORC from S (step 1), color it according to the procedure outlined in Section 4 (step 2), and then construct the precise DPA from it as in Theorem 4.4. However, the upper bound on the size of the precise DPA is exponential (in the number of priorities) compared to the size of the FORC, and further, even if the precise DPA is small, the construction might produce a large intermediate result before minimizing the DPA as a Mealy machine that emits priorities. For this reason, the step that transforms the coloring on the FORC into a DPA is split up into two parts (steps 3 and 4 of the algorithm). In step 3, the coloring of the precise DPA on prefixes of the sample is computed. This can be done in polynomial time thanks to Lemma 3.21. In step 4 an active learner for Mealy machines is used, where the oracle answers queries based on the coloring computed in step 3. This ensures that no large intermediate result is produced and that the computation remains polynomial in the size of the sample.

Note that these descriptions are given under the assumption that the sample contains enough information from an underlying language L for inferring the correct objects in each step. However, the algorithm has to produce some DPA that is consistent with the sample in polynomial time, even if the sample does not fully characterize the objects required by our algorithm. In such situations, it might happen that intermediate results are inconsistent with the sample. For example, in step 1 the algorithm might not be able to infer a FORC that is consistent with the sample, or in step 3, the coloring on the prefixes of the sample might not be consistent with the sample. In such cases, the algorithm defaults, for that step, to a simple structure that is guaranteed to be consistent with the sample, such that it can proceed with the next step. In such situations, one could also immediately return some default DPA that is consistent with the sample. We decided to return default structures for the corresponding step because then the algorithm still might generalize from the sample in the following steps.

Input: A function cons and a complete TS $\mathcal{T}_0$, where
   cons maps a partial TS to a boolean
   $\mathcal{T}_0$ serves as default and verifies cons($\mathcal{T}_0$) = true.
Output: A complete TS $\mathcal{T}$ with cons($\mathcal{T}$) = true and $|\mathcal{T}| \leq |\mathcal{T}_0|$.

1:   $Q \leftarrow \{\varepsilon\}, \delta \leftarrow \emptyset, \mathcal{T} \leftarrow (Q, \Sigma, \delta, \varepsilon)$
2:   **while** $\mathcal{T}$ is not complete **do**
3:       **if** $|\mathcal{T}| > |\mathcal{T}_0|$ **then**   **return** $\mathcal{T}_0$
4:       $pa \leftarrow$ llex. minimal word in $Q\Sigma$ such that $\delta(p, a)$ is undefined
5:       **foreach** $q \in Q$ in length-lexicographic order **do**
6:           $\delta' \leftarrow \delta \cup \{p \xrightarrow{a} q\}$
7:           **if** cons($(Q, \Sigma, \varepsilon, \delta')$) returns true **then**
8:               $\delta \leftarrow \delta'$
9:               **continue** with next iteration of while loop
10:      $Q \leftarrow Q \cup \{pa\}, \delta \leftarrow \delta \cup \{p \xrightarrow{a} pa\}$
11: **return** $\mathcal{T}$

Algorithm 1. GLeRC

We later show that if S contains enough information on the language L, then the syntactic FORC and thus the precise coloring will be inferred, and that the construction of the DPA from the coloring will result in a correct DPA for L whose size is bounded by the precise DPA (Theorem 5.7).

The theoretical foundations for the steps 2,3,4 have been developed in the previous sections. We now give more details on the first step, and then describe the learning algorithm.

For learning the FORC, we use the procedure *G*reedily *Le*arn *R*ight *C*ongruence (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 5.1. *For a consistency function* cons *and a complete transition system $\mathcal{T}_0$ with* cons$(\mathcal{T}_0)$ = true*, the algorithm* GLeRC *computes a complete TS $\mathcal{T}$ such that* cons$(\mathcal{T})$ = true *in time polynomial in the runtime of* cons *and the size of $\mathcal{T}_0$.*

Before we go into the details of how we instantiate GLeRC in our setting for learning the congruences of a FORC, let us briefly comment on the default transition system that is used for ensuring termination. The algorithm GLeRC has emerged as an abstract version of the learning algorithm Sprout for deterministic ω-automata presented in [8], which itself can be seen as an extension of the RPNI algorithm for learning DFAs from samples of finite words [30]. The instantiation of GLeRC that corresponds to the RPNI algorithm for samples of finite words does not require a default transition system, because after polynomially many steps all finite words from the sample do have a path that completely lies inside the constructed transition system, and after that point no new states will be created by GLeRC. In the context of ω-words, however, this is not necessarily the case: It was established in [8, Proposition 10] that there exist samples for which the Sprout algorithm does not terminate without the use of a default transition system. While this example does not directly apply to the instantiations of GLeRC that we use below, there are no easy termination arguments in the setting of ω-words as it is the case for finite words. For this reason, we use the default transition system. This allows us to cover existing algorithms in a unified form, and it gives an easy termination argument.

Let us now describe how we instantiate GLeRC for learning the congruences of a FORC. The cons functions that we pass to GLeRC, verify that the constructed TS satisfies certain notions of consistency with the sample, which relativize the definitions of the canonical FORC and its coloring from Section 4 to a sample S (Definition 5.2). In Lemma 5.3 and Lemma 5.4 we show that these consistency checks can be performed in polynomial time. We then combine these two consistency notions into a notion of consistency of a sample with a FORC (Definition 5.5), followed by a short description of the default transition system that we use in combination with the two consistency functions. This allows us to subsequently use the GLeRC algorithms with different cons functions for inferring the leading and progress right congruences in step 1 of the learning algorithm. As we see later in the proof of Theorem 5.7, for these consistency functions there are simple arguments for showing that the appropriate transition systems are inferred in the limit.

DEFINITION 5.2. 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 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 $(q_1, q_2) \in C$. 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 5.3. *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 $r_1 x y^\omega \in S_+$ and $r_2 x y^\omega \in S_-$ 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, $xz, yz \in \mathrm{Prf}\, S$, 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. ∎

LEMMA 5.4. *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 5.3, 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

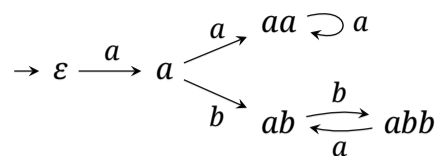— $\mathcal{A}_2$ accepts x if $x \in E_c^\sim$ and $x^\omega \in S_-$.

For the construction of the $\mathcal{A}_i$, let $\check{R}_\sigma$ 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 $\check{R}_\sigma$, 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 if and only if 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 $\check{R}_\sigma$, 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 $\check{R}_\sigma$ 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 $\mathcal{T} \times \mathcal{A}_1$

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.

---

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 $(\delta_1^*(x), \delta_2^*(y)) \in C$.
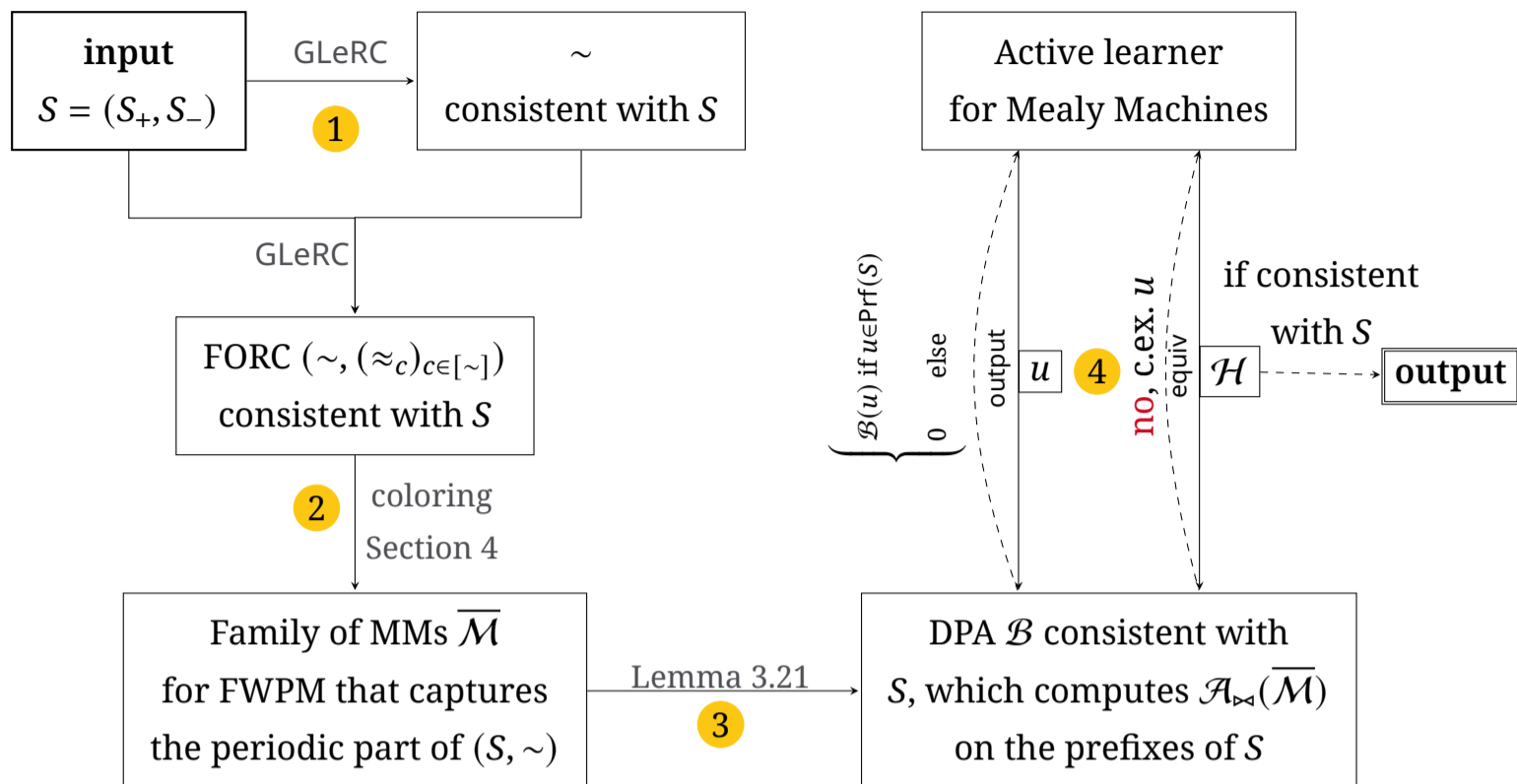
Overall, we obtain that $\mathcal{T}$ is *not* iteration consistent with $(S, \sim, c)$, if and only if there is some q in $\mathcal{T}$ such that (q, q_1) and (q, q_2) are reachable in $\mathcal{T} \times \mathcal{A}_1$ and $\mathcal{T} \times \mathcal{A}_2$, respectively, and $(q_1, q_2) \in C$. As $\mathcal{A}_1, \mathcal{A}_2$ 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. ■

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

DEFINITION 5.5. 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 $\sigma \in \{+, -\}$.

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 $x, y \notin \mathrm{Prf}(S)$ or we have $x^{-1}S = y^{-1}S$ and $|x^{-1}S| = 1$. It

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

is not hard to see that the size of $\sim_S$ is polynomial in $|S|$, see for example [9, Appendix B]. Further, $\sim_S$ is MN- and iteration consistent with S because it separates all words x, y that are prefixes of different words in S, which directly implies that the conditions from Definition 5.2 are satisfied.

Formal description of the learner The learner that we propose consists of several steps (which are sketched at beginning of this Section 5). 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 DPAInf in more detail. The full arguments why this results in a polynomial time consistent learner that can learn every regular ω-language in the limit is given in the proofs of Theorem 5.6 and Theorem 5.7. In the description of the steps we only indicate some of these arguments in order to ease the understanding of the algorithm.

Step 1 In the first step, DPAInf 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 5.3 and Lemma 5.4. For the LRC, DPAInf calls GLeRC with $\sim_S$ as default and using LRC-cons as consistency function, where LRC-cons(T) returns true if and only if T is MN-consistent with S (see Lemma 5.3). 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 consistency function PRC-cons$(\mathcal{T})$, which returns true if

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

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 4.3. 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 $\overline{\mathcal{M}} = (\mathcal{M}_c)_{c \in [\sim]}$ computing a weak priority mapping $\overline{\gamma} = (\gamma_c)_{c \in [\sim]}$. This is done in a way which mimics the definition of the coloring κ_c in Section 4 and ensures that $\overline{\gamma}$ is the precise FWPM of L, provided the syntactic FORC was learned in step 1 (see Lemma 4.2). 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 $\mathsf{Inf}_{\mathcal{T}}(R_{c,\sigma})$ 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 $+$ if and only if 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 (with regard to 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 (or the size of an intermediate result) can be exponential in $\overline{\mathcal{M}}$. Instead, DPAInf builds a DPA $\mathcal{B}$ which computes the priority mapping

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

It can be shown by using Lemma 3.21 that constructing $\mathcal{B}$ is possible in polynomial time (see the proof of Theorem 5.6 below). Then, our algorithm checks if $\mathcal{B}$ 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 $\mathcal{B} : \Sigma^+ \to \{0, 1\}$ with $\mathcal{B}(u) \mapsto 1$

iff $u \in \mathsf{Prf}\left(\bigcup_{c \in [\sim]} R_{c,-}\right)$. Note that this fallback DPA is consistent with S since it outputs only priority 1 on all negative examples, and finally only priority 0 on all other ω-words. Hence, the DPA $\mathcal{B}$ that is passed to the next step is consistent with S.

Step 4 The purpose of this step is to compute a small DPA that is consistent with the sample and minimal viewed as a Mealy machine. To achieve this, step 4 runs a polynomial time active learner MMAL for Mealy machines as a black box, answering the queries based on the coloring computed in step 3. If the sample is complete for a language L, then step 3 terminates with a coloring on the prefixes of the sample that corresponds to the one computed by the precise DPA for L. If all queries posed by MMAL can be answered using prefixes of the sample, we can prove that the precise DPA for L (or a smaller one) will be learned.

The oracle that is provided to MMAL uses the DPA $\mathcal{B}$ from step 3 and reacts to the posed queries as follows:

$$\mathsf{output}(u) \rightsquigarrow \text{ answer } \mathcal{B}(u)$$

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

Note that, by definition, if this step terminates, then the resulting DPA is consistent with the sample. Further, this step is guaranteed to terminate because the coloring from step 3 is consistent with the sample (see proof of Theorem 5.6).

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

THEOREM 5.6. DPAInf *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 DPAInf is called. By definition, step 4 always terminates with a hypothesis $\mathcal{H}$ that is consistent with S. Thus, it remains to show that DPAInf always terminates in polynomial time. In the following, we consider each step of the algorithm individually:

— In Lemma 5.3 and Lemma 5.4, it was established that the two consistency functions LRC-cons and PRC-cons, which are passed to GLeRC, run in polynomial time. Thus, by Proposition 5.1, 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 3.21, 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 a 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 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, DPAInf 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 DPAInf 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 DPAInf 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 5.7. DPAInf *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 DPAInf 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 output(u) queries posed by MMAL in step 4, $u \in \mathsf{Prf}(S^L)$ 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 $\mathsf{mr}(q)$ 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 $xa \nsim y$ for $x = \mathsf{mr}(q), y = \mathsf{mr}(p)$ because up to now all transitions have been inserted correctly. Depending on the kind of right congruence we want to learn, we add the following words to W:

— If $\sim$ is the LRC of $\mathcal{F}_L$ and $xa \nsim_L y$:
   There exist $u \in \Sigma^*, v \in \Sigma^+$ with $xauv^\omega \in L$ if and only if $yuv^\omega \notin L$, 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 $xa \nsimeq_c y$:
   Let $u = \mathsf{mr}(c)$. 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, DPAInf considers all words in $R_{c,\sigma}$ for $\sigma \in \{+, -\}$. To ensure that the second step of DPAInf 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 DPAInf. 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 $\mathcal{A}_{\bowtie} := \mathcal{A}_{\bowtie}(\overline{\mathcal{M}})$ 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 3.17 that $\mathcal{A}_{\bowtie}$ computes the priority mapping of $\mathcal{A}_L$. By definition, $\mathcal{B}$ behaves like $\mathcal{A}_{\bowtie}$ 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 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 $\mathcal{A}_{\bowtie}$ and whenever MMAL poses an 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 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, DPAInf 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, DPAInf 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 $|\mathcal{A}_L|$.   ∎

By combining Theorem 5.7 with Proposition 4.5 and Theorem 3.19, 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 $\mathbb{IRC}(\mathbb{DPA})$ and $d\text{-}\mathbb{IRC}(k\text{-}\mathbb{DPA})$ 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 $d\text{-}\mathbb{IRC}(k\text{-}\mathbb{DPA})$ contains all languages L that can be recognized by a DPA with priorities in $\{0, \ldots, k-1\}$ 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 $d\text{-}\mathbb{IRC}(2\text{-}\mathbb{DPA})$ in the limit from polynomial data is presented in [10]

The languages in $\mathbb{IRC}(\mathbb{DPA})$ are those for which $d = 1$ in Proposition 4.5 and Theorem 3.19. In this case, the term in Theorem 3.19 reduces to m and the one in Proposition 4.5 to mk. If d and k are both fixed, both expressions become linear in m. This implies the following.

COROLLARY 5.8. *The algorithm* DPAInf *can learn a DPA for every language in* $\mathbb{IRC}(\mathbb{DPA})$ *from polynomial data. Furthermore, there is a fixed polynomial g such that for every k and d,* DPAInf *can learn a DPA for every language in* $d\text{-}\mathbb{IRC}(k\text{-}\mathbb{DPA})$ *in the limit from polynomial data with characteristic samples of size* $O(g)$.

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. DOI (2)

[2] **Dana Angluin**, **Udi Boker**, and **Dana Fisman**. Families of DFAs as acceptors of ω-regular languages. *Logical Methods in Computer Science*, 14, 2018. DOI (4, 7, 32)

[3] **Dana Angluin** and **Dana Fisman**. Learning regular omega languages. *Theoretical Computer Science*, 650:57–72, 2016. DOI (7, 29)

[4] **Dana Angluin**, **Dana Fisman**, and **Yaara Shoval**. Polynomial identification of ω-automata. *Proceedings of the 26th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2020)*, pages 325–343. DOI (5, 6, 45)

[5] **Christel Baier** and **Joost-Pieter Katoen**. Principles of model checking. MIT Press, 2008. (2)

[6] **Stephan Barth** and **Martin Hofmann**. Learn with SAT to minimize Büchi automata. *Proceedings of the 3rd International Symposium on Games, Automata, Logics and Formal Verification (GandALF 2012)*, pages 71–84. DOI (5)

[7] **Alan W. Biermann** and **Jerome A. Feldman**. On the synthesis of finite-state machines from samples of their behavior. *IEEE Transactions on Computers*, 21(6):592–597, 1972. DOI (1)

[8] **León Bohn** and **Christof Löding**. Constructing deterministic ω-automata from examples by an extension of the RPNI algorithm. *Proceedings of the 46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021)*, 20:1–20:18. DOI (4–7, 35, 36, 45)

[9] **León Bohn** and **Christof Löding**. Constructing deterministic ω-automata from examples by an extension of the RPNI algorithm. *arXiv e-prints*:arXiv:2108.03735, August 2021. DOI (39)

[10] **León Bohn** and **Christof Löding**. Passive learning of deterministic Büchi automata by combinations of DFAs. *Proceedings of the 49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, 114:1–114:20. DOI (4–6, 25, 45)

[11] **J Richard Büchi**. Symposium on decision problems: on a decision method in restricted second order arithmetic, *Studies in Logic and the Foundations of Mathematics*. Volume 44, pages 1–11. 1966. (2, 11, 21)

[12] **Hugues Calbrix**, **Maurice Nivat**, and **Andreas Podelski**. Ultimately periodic words of rational ω-languages. *Mathematical Foundations of Programming Semantics*, pages 554–566, Berlin, Heidelberg, 1994. DOI (2, 6, 11)

[13] **Olivier Carton** and **Ramón Maceiras**. Computing the Rabin index of a parity automaton. *RAIRO - Theoretical Informatics and Applications*, 33(6):495–505, 1999. DOI (6, 9)

[14] **Thomas Colcombet**. Green's relations and their use in automata theory. *Proceedings of the 5th International Conference on Language and Automata Theory and Applications (LATA 2011)*, pages 1–21. DOI (46)

[15] **Colin de la Higuera** and **Jean-Christophe Janodet**. Inference of ω-languages from prefixes. *Theoretical Computer Science*, 313(2):295–312, 2004. DOI (5)

[16] **Rüdiger Ehlers and Sven Schewe**. Natural Colors of Infinite Words. *Proceedings of the 42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022)*, 36:1–36:17. DOI (9, 12, 46)

[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 ω-regular languages. *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008)*, pages 2–17. DOI (6, 7)

[18] **E. Mark Gold**. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978. DOI (1, 3)

[19] **E. Mark Gold**. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967. DOI (1)

[20] **John E. Hopcroft and Jeffrey D. Ullman**. Formal Languages and their Relation to Automata. Addison-Wesley, 1969. (2)

[21] **Malte Isberner, Falk Howar, and Bernhard Steffen**. The open-source learnlib - A framework for active automata learning. *Proceedings of the 27th International Conference of Computer Aided Verification (CAV 2015)*, pages 487–495. DOI (2)

[22] **Nils Klarlund**. A homomorphism concept for ω-regularity. *Proceedings of the 8th International Workshop on Computer Science Logic (CSL 1994)*, volume 933, pages 471–485. DOI (7, 29)

[23] **Patrick Landwehr**. Tree automata with constraints on infinite trees. Dissertation, RWTH Aachen University, Aachen, 2021. DOI (37)

[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. *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2017)*, pages 208–226. DOI (7)

[25] **Damian López and Pedro García**. On the inference of finite state automata from positive and negative data, *Topics in Grammatical Inference*. Springer, 2016. DOI (1)

[26] **Oded Maler and Amir Pnueli**. On the learnability of infinitary regular sets. *Information and Computation*, 118(2):316–326, 1995. DOI (6)

[27] **Oded Maler and Ludwig Staiger**. On syntactic congruences for ω-languages. *Theoretical Computer Science*, 183(1):93–112, 1997. DOI (4, 7, 28, 29)

[28] **Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger**. Strix: explicit reactive synthesis strikes back! *Proceedings of the 30th International Conference on Computer Aided Verification (CAV 2018)*, pages 578–586. DOI (2)

[29] **Jakub Michaliszyn and Jan Otop**. Learning infinite-word automata with loop-index queries. *Artificial Intelligence*, 307:103710, 2022. DOI (7)

[30] **Jose Oncina and Pedro García**. Inferring regular languages in polynomial update time. *World Scientific*, January 1992. DOI (6, 35)

[31] **Charles P. Pfleeger**. State reduction in incompletely specified finite-state machines. *IEEE Transactions on Computers*, C-22(12):87–106, 1973. DOI (3)

[32] **Bernhard Steffen, Falk Howar, and Maik Merten**. Introduction to active automata learning from a practical perspective. *Proceedings of the 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM 2011)*, pages 256–296. DOI (11)

[33] **Wolfgang Thomas**. Automata on infinite objects. *Handbook of Theoretical Computer Science (Vol. B): Formal Models and Semantics*. Cambridge, MA, USA, 1991., pages 133–191. DOI (2)

[34] **Wolfgang Thomas**. Facets of synthesis: revisiting Church's problem. *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures (FOSSACS 2009)*, pages 1–14. DOI (2)

[35] **Wolfgang Thomas**. Languages, automata, and logic, *Handbook of Formal Languages, Volume 3: Beyond Words*, pages 389–455. 1997. DOI (8, 9)

[36] **BA Trakhtenbrot and Ya M Barzdin**. Finite automata—behaviour and synthesis, vol. 1 of. *Fundamental Studies in Computer Science*, 1973. (1)

[37] **Moshe Y Vardi and Thomas Wilke**. Automata: from logics to algorithms. *Logic and automata*, 2:629–736, 2008. (8, 9)

[38] **Sicco Verwer and Christian A. Hammerschmidt**. Flexfringe: A passive automaton learning package. *Proceedings of the 33rd IEEE International Conference on Software Maintenance and Evolution (ICSME 2017)*, pages 638–642. URL (2)

[39] **Thomas Wilke**. ω-automata. *arXiv preprint arXiv:1609.03062*, 2016. (8, 9)