Constructing Deterministic ω-Automata from Examples by an Extension of the RPNI Algorithm

León Bohn
RWTH Aachen University, Germany

Christof Löding
RWTH Aachen University, Germany

Abstract

The RPNI algorithm (Oncina, Garcia 1992) constructs deterministic finite automata from finite sets of negative and positive example words. We propose and analyze an extension of this algorithm to deterministic ω-automata with different types of acceptance conditions. In order to obtain this generalization of RPNI, we develop algorithms for the standard acceptance conditions of ω-automata that check for a given set of example words and a deterministic transition system, whether these example words can be accepted in the transition system with a corresponding acceptance condition. Based on these algorithms, we can define the extension of RPNI to infinite words. We prove that it can learn all deterministic ω-automata with an informative right congruence in the limit with polynomial time and data. We also show that the algorithm, while it can learn some automata that do not have an informative right congruence, cannot learn deterministic ω-automata for all regular ω-languages in the limit. Finally, we also prove that active learning with membership and equivalence queries is not easier for automata with an informative right congruence than for general deterministic ω-automata.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases deterministic omega-automata, learning from examples, learning in the limit, constructing acceptance conditions, active learning

Digital Object Identifier 10.4230/LIPIcs.MFCS.2021.20

Related Version *Full Version*: https://arxiv.org/abs/2108.03735

1 Introduction

In this paper we consider learning problems for automata on infinite words, also referred to as ω-automata, which have been studied since the early 1960s as a tool for solving decision problems in logic [7] (see also [26]), and are nowadays used in procedures for formal verification and synthesis of reactive systems (see, e.g., [5, 27, 18] for surveys and recent work). Syntactically ω-automata are very similar to NFA resp. DFA (standard nondeterministic resp. deterministic finite automata on finite words), and they also share many closure and algorithmic properties. However, many algorithms and constructions are much more involved for ω-automata, one prominent such example being determinization [23, 22, 24, 15], and another one the minimization of deterministic ω-automata [25], which is hard for most of the acceptance conditions of ω-automata. The underlying reason is that regular languages of finite words have a simple characterization in terms of the Myhill/Nerode congruence, and the unique minimal DFA for a regular language can be constructed by merging language equivalent states (see [13]). In contrast, deterministic ω-automata need, in general, different language equivalent states for accepting a given regular ω-language.

The characterization of minimal DFA in terms of the Myhill/Nerode congruence is also an important property that is used by learning algorithms for DFA. In automaton learning one usually distinguishes the two settings of passive and active learning. We are mainly concerned with passive learning in this paper, where the task is to construct an automaton

from a sample, a given finite set of words together with a classification if they are in the language or not. The RPNI algorithm [21] is a passive learning algorithm that constructs a DFA from a given sample of positive and negative examples (words that are in the language and words that are not in the language, respectively). It starts with the prefix tree acceptor, a tree shaped DFA that accepts precisely the positive examples and subsequently it tries to merge pairs of states in the canonical order of words (each state is associated with the word reaching it in the prefix tree acceptor). If a merge results in a DFA that accepts a negative example, the merge is discarded. Otherwise the merge is kept and the algorithm continues with this DFA. RPNI can learn the minimal DFA for each regular language in the limit with polynomial time and data. This means that RPNI runs in polynomial time in the size of the given sample, and for each regular language L there is a characteristic sample S_L of polynomial size, such that RPNI produces the minimal DFA for L for each sample that is consistent with L and contains S_L [21]. The RPNI algorithm is a simple algorithm that also produces useful results if the sample does not include the characteristic sample of any language L. Therefore its principle of state merging has been used for other automaton models, e.g., probabilistic automata [8, 17] and sequential transducers [20].

In this paper we propose and analyze an extension of RPNI to ω-automata. In the setting of infinite words, one uses ultimately periodic words of the form uv^ω for finite words u, v. These are infinite words with a finite representation, and each regular ω-language is uniquely determined by the set of ultimately periodic words that it contains (see [26]). There are two main obstacles that one has to overcome for a generalization of RPNI. First, it is not clear how to generalize the prefix tree acceptor to infinite words, since a tree shaped acceptor for a set of infinite words necessarily needs to be infinite. We therefore propose a formulation of the algorithm that inserts transitions instead of merging states, and creates new states in case none of the existing states can be used as a target of the transition. In the setting of finite words, this method of inserting transitions produces the same result as RPNI, and it can easily be used for infinite words as well. Because this algorithm produces growing transition system, we call it Sprout.

The second problem arises in the test whether a merge (in our formulation an inserted transition) should be kept or discarded. In the case of finite words, one can simply check whether there are a positive and a negative example that reach the same state, which obviously is not possible in a DFA that is consistent with the sample. For ω-automata the situation is a bit more involved, because acceptance of a word is not determined by a single state, but rather the set of states that is reached infinitely often. And furthermore, there are various acceptance conditions using different ways of classifying these infinity sets into accepting and rejecting. To solve this problem, we propose polynomial time algorithms for checking whether a deterministic transition system admits an acceptance condition of a given type (Büchi, generalized Büchi, parity, or Rabin) that turns the transition system into a deterministic ω-automaton that is consistent with the sample. These consistency algorithms are then used in Sprout in order to check whether a merge (inserted transition) produces a transition system that can still be consistent with the sample (for the acceptance condition under consideration). However, we believe that these consistency algorithms are of interest on their own and might also be useful in other contexts. We also show that bounding the size of the acceptance condition can make the problem hard: consistency with a Rabin condition with three pairs or generalized Büchi condition with three sets is NP-hard.

Our analysis of Sprout reveals that it can learn every ω-regular language with an informative right congruence (IRC) in the limit from polynomial time and data. A deterministic ω-automaton has an informative right congruence if it has only one state for each My-

hill/Nerode equivalence class of the language that it defines [3]. Recently, another algorithm that can learn every ω-regular language with an IRC in the limit from polynomial time and data has been proposed [4]. This algorithm is an extension of the approach from [12] from finite to infinite words. However, the algorithm from [4] has explicitly been developed for automata with an IRC, and it can only produce such automata (it defaults to an automaton accepting precisely the positive examples in case the sample does not completely characterize the target automaton). In contrast, Sprout is not specifically designed for IRC languages, it can also construct automata that do not have an IRC. But on the negative side we also show that Sprout cannot learn a deterministic ω-automaton for every regular ω-language.

The positive results for passive learning of IRC languages raise the question whether this class is also simpler for active learning than general deterministic ω-automata. The standard model for active learning of automata uses membership and equivalence queries, and DFA can be learned in polynomial time in this model [1]. This approach has been extended to the class of weak deterministic Büchi automata [16], whose minimal automata can also be defined using the standard right congruence. For general regular ω-languages, the only known algorithms either learn a different representation based on DFA [2], or add another query about the loop structure of the target automaton [19]. Since the characterization of the minimal automata by a right congruence is a crucial point in many active learning algorithms, it is tempting to believe that the algorithms can be extended to the classes of languages with an IRC. We prove that this is not the case by showing that a polynomial time active learning algorithm for deterministic ω-automata with an IRC can be turned into a polynomial time learning algorithm for general deterministic ω-automata.

Finally, we also make the observation that polynomial time active learning (with membership and equivalence queries) is at least as hard as learning in the limit with polynomial time and data.

The paper is structured as follows. In Section 2 we give basic definitions. In Section 3 we present the consistency algorithms, and in Section 4 we describe our extension of RPNI to ω-automata. In Section 5 we show that the property of an IRC does not help for polynomial time active learning, and in Section 6 we conclude.

2 Preliminaries

For a finite alphabet Σ we use Σ^* and Σ^ω to refer to the set of finite and infinite words respectively. The empty word is denoted by ε, and $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. A deterministic transition system (TS) is defined by a tuple $\mathcal{T} = (Q, \Sigma, q_0, \delta)$ where Q is a finite set of states, Σ a finite alphabet, $q_0 \in Q$ the initial state and $\delta : Q \times \Sigma \to Q$ is the transition function. We use $\delta(q, a) = \bot$ to indicate that a transition $(q, a) \in Q \times \Sigma$ is not defined in $\mathcal{T}$. Further we extend δ to $\delta^* : Q \times \Sigma^* \to Q$ defined as $\delta^*(q, \varepsilon) = q$ and $\delta^*(q, aw) = \delta^*(\delta(q, a), w)$ for $q \in Q, a \in \Sigma$ and $w \in \Sigma^*$. Unless otherwise specified $\mathcal{T}$ will be used to refer to a transition system with components as above. The unique run of $\mathcal{T}$ on $w \in \Sigma^\omega$ is a sequence of transitions $\rho = q_0 w_0 q_1 w_1 \dots$ with $q_{i+1} = \delta(q_i, w_i)$. For an infinite run ρ we denote by $\mathsf{inf}(\rho)$ the *infinity set* of ρ, consisting of all state-symbol pairs that occur infinitely often in ρ. A set of states $\emptyset \neq C \subseteq Q$ is called *strongly connected* if for all $p, q \in C$ we have $\delta^*(p, w) = q$ for some $w \in \Sigma^+$. The $\subseteq$-maximal strongly connected sets of $\mathcal{T}$ are called strongly connected components (SCCs) and for a set R we use $\mathsf{SCC}(R)$ to refer to the set of all SCCs $S \subseteq R$.

Augmenting a transition system $\mathcal{T}$ with an acceptance condition $\mathcal{C}$ yields an ω-automaton $\langle \mathcal{T}, \mathcal{C} \rangle = (Q, \Sigma, q_0, \delta, \mathcal{C})$. We now introduce different types of acceptance conditions (based on the survey [26]), give a notion of their size $|\mathcal{C}|$ and define which sets $X \subseteq Q \times \Sigma$ satisfy

them. Note that while acceptance is often defined based on *states* that occur infinitely often, we opt for transition-based acceptance due to its succinctness (state-based acceptance can be turned into transition-based acceptance without changing the transition system, while the transformation in the other direction requires a blow-up of the transition system depending on the acceptance condition).

A *Büchi condition* $F \subseteq Q \times \Sigma$ is satisfied if $X \cap F \neq \emptyset$, whereas a *generalized Büchi condition* $\mathcal{B} = \{F_1, \dots, F_k\}$ with $F_i \subseteq Q \times \Sigma$ is satisfied if $X \cap F_i \neq \emptyset$ for all $i \in [1, k] \subseteq \mathbb{N}$. The set X satisfies a *parity condition* $\kappa : (Q \times \Sigma) \to C$ for a finite $C \subseteq \mathbb{N}$ if $\min(\kappa(X))$ is even where $\kappa(X) = \{\kappa(q, a) : (q, a) \in X\}$. We call $\mathcal{R} = \{(E_1, F_1), \dots, (E_k, F_k)\}$ with $E_i, F_i \subseteq (Q \times \Sigma)$ a *Rabin condition* and it is satisfied if $E_i \cap X = \emptyset$ and $F_i \cap X \neq \emptyset$ for some $i \in [1, k] \subseteq \mathbb{N}$. Finally a *Muller condition* $\mathcal{F} \subseteq 2^{Q \times \Sigma}$ is satisfied if $X \in \mathcal{F}$. For an acceptance condition $\mathcal{C}$ of type $\Omega \in \{\text{Parity}, \text{generalized Büchi}, \text{Rabin}\}$ we use $|\mathcal{C}|$ to refer to the number of priorities/recurring sets/Rabin pairs respectively. We use abbreviations (g)DBA, DPA, DRA to refer to deterministic (generalized) Büchi, Parity and Rabin automata and introduce a set Acc containing these acceptance types. An automaton $\mathcal{A} = \langle \mathcal{T}, \mathcal{C} \rangle$ accepts $w \in \Sigma^\omega$ if $\mathsf{inf}(\rho)$ satisfies $\mathcal{C}$, where ρ refers to the unique run of $\mathcal{T}$ on w. The set of all words that are accepted by $\mathcal{A}$ is the language accepted by $\mathcal{A}$, denoted by $L(\mathcal{A})$.

Let $\sim$ be an equivalence relation over Σ^*. We refer to the equivalence class of x under $\sim$ as $[x]_\sim = \{y \in \Sigma^* : x \sim y\}$ and call $\sim$ a (right) *congruence* if $u \sim v$ implies $ua \sim va$ for all $a \in \Sigma$. A regular language $L \subseteq \Sigma^\omega$ induces the *canonical right congruence* $\sim_L$ in which $u \sim_L v$ holds if and only if $u^{-1}L = v^{-1}L$ with $u^{-1}L = \{w \in \Sigma^\omega : uw \in L\}$. Using the terminology of [3], we say that an automaton $\mathcal{A}$ has an *informative right congruence* (IRC) if $u \sim_{L(\mathcal{A})} v$ implies that $\mathcal{A}$ reaches the same state from q_0 when reading u or v. A language L has an Ω-IRC for $\Omega \in \mathsf{Acc}$ if an Ω-automaton with an IRC which recognizes L exists and we denote by $\mathsf{ind}(L)$ the number of equivalence classes of $\sim_L$.

A word $w \in \Sigma^\omega$ is called *ultimately periodic* if $w = uv^\omega$ with $u \in \Sigma^*, v \in \Sigma^+$. We denote by UP_Σ the set of all ultimately periodic words in Σ^ω and note that two regular languages $K, L \subseteq \Sigma^\omega$ are equal if and only if $K \cap \mathsf{UP}_\Sigma = L \cap \mathsf{UP}_\Sigma$ [7]. Note that there always exists a reduced form $w = uv^\omega$ in which u and v are as short as possible. We call a pair $S = (S_+, S_-)$ with $S_+, S_- \subseteq \mathsf{UP}_\Sigma$ and $S_+ \cap S_- = \emptyset$ a *sample* and say that S is *in reduced form* if each $uv^\omega \in S$ is in a reduced form where $w \in S$ is used as a shorthand for $w \in S_+ \cup S_-$. For $L \subseteq \Sigma^\omega$ we say that S is consistent with L if $S_+ \subseteq L$ and $S_- \cap L = \emptyset$. Similarly an automaton $\mathcal{A}$ is consistent with S if $S_+ \subseteq L(\mathcal{A})$ and $S_- \cap L(\mathcal{A}) = \emptyset$.

For $\Omega \in \mathsf{Acc}$ we call a function f that maps a sample to an Ω-automaton a *passive learner*. f is called *consistent* if for any sample S the constructed automaton $f(S)$ is consistent with S. A sample S_L is *characteristic* for L and f if for any sample S that is consistent with L and that contains S_L, the learner produces an automaton $f(S)$ recognizing L. For a class of representations of languages $\mathbb{C}$ (in our case deterministic ω-automata) we use $\mathcal{L}(\mathbb{C})$ to refer to the represented languages and define the size of $L \in \mathcal{L}(\mathbb{C})$ to be the size of the minimal representation of L in $\mathbb{C}$. Based on the definition in [10] we say $\mathbb{C}$ is *learnable in the limit using polynomial time and data* if there exists a learner f that runs in polynomial time for any input sample, and for each $L \in \mathcal{L}(\mathbb{C})$ there exists a characteristic sample whose size is polynomial in the size of L.

We call $w \in \Sigma^\omega$ *escaping* from $p \in Q$ with $a \in \Sigma$ in $\mathcal{T}$ if there exists a decomposition $w = uav$ with $v \in \Sigma^\omega$ such that $\delta^*(q_0, u) = p$ and $\delta(p, a) = \bot$. We refer to ua as the *escape-prefix* and call av the *exit string* of w. Two escaping words w_1, w_2 are *indistinguishable* if they escape $\mathcal{T}$ from the same state and their exit strings coincide. We call $\mathcal{T}$ Ω-*consistent* with a sample S if there exists an Ω-acceptance condition $\mathcal{C}$ such that $\{w \in S_+ : w \text{ not escaping in } \mathcal{T}\} \subseteq$

$L(\mathcal{T},\mathcal{C}), S_- \cap L(\mathcal{T},\mathcal{C}) = \emptyset$ and no pair of sample words from $S_+ \times S_-$ is indistinguishable. Note that Ω-consistency with a transition system does not require all words from S_+ to have an infinite run in the transition system. It just means that $\mathcal{T}$ does not produce any conflicts between words in S_+ and in S_-. In contrast, for an automaton to be considered consistent with S it is required that all words from S_+ are accepted.

3 Consistency Algorithms

The algorithm for learning ω-automata that we describe in Section 4 constructs a transition system and then tests whether an acceptance condition can be found such that all sample words are accepted and rejected accordingly. In this section we develop algorithms for this test, so we assume that a transition system $\mathcal{T} = (Q, \Sigma, q_0, \delta)$ is given. We do not work with the sample directly in this section, and rather work with the infinity sets induced by the sample words. This leads to the notion of a partial condition, which we define below. Then we investigate how different types of acceptance conditions that are consistent with such a partial condition can be constructed.

Recall that a Muller condition $\mathcal{F} \subseteq 2^{Q\times\Sigma}$ is satisfied by an infinity set $X \subseteq Q \times \Sigma$ if and only if $X \in \mathcal{F}$. Instead of specifying such a Muller condition based solely on the infinity sets that satisfy it, we can also define it as a partition $\mathcal{F} = (\mathcal{F}_0, \mathcal{F}_1)$ of $2^{Q\times\Sigma}$ into accepting and rejecting sets, in the following also referred to as positive and negative sets respectively. In other words such a condition assigns to each possible set $X \subseteq Q\times\Sigma$ a *classification* $\sigma \in \{0,1\}$, which we denote as $\mathcal{F}(X) = \sigma$ for $X \in \mathcal{F}_\sigma$. Note that any acceptance condition $\mathcal{C}$ can be viewed as a Muller condition $(\mathcal{F}_0^{\mathcal{C}}, \mathcal{F}_1^{\mathcal{C}})$ by assigning to $\mathcal{F}_0^{\mathcal{C}}$ exactly those sets $X \subseteq Q \times \Sigma$ that satisfy $\mathcal{C}$ and defining $\mathcal{F}_1^{\mathcal{C}}$ to contain all others.

To incorporate the fact that the infinity sets induced by sample words might not classify all subsets of $Q \times \Sigma$, we introduce the concept of a *partial condition* $\mathcal{H} = (\mathcal{H}_0, \mathcal{H}_1)$ with $\mathcal{H}_0, \mathcal{H}_1 \subseteq 2^{Q\times\Sigma}$ in which only a subset of all elements $X \subseteq Q \times \Sigma$ receives a classification $\mathcal{H}(X) \in \{0,1\}$. We use $X \in \mathcal{H}$ to denote $X \in \mathcal{H}_0 \cup \mathcal{H}_1$ and call a partial condition $\mathcal{H} = (\mathcal{H}_0, \mathcal{H}_1)$ *consistent* if $\mathcal{H}_0 \cap \mathcal{H}_1 = \emptyset$. A component $\mathcal{H}_\sigma$ of a partial condition is called *union-closed* if for any finite collection $X_1, \ldots, X_n \in \mathcal{H}_\sigma$ we have $X_1 \cup \ldots \cup X_n \notin \mathcal{H}_{1-\sigma}$ or in other words the union of positive sets is not negative and vice versa. We call an acceptance condition $\mathcal{C}$ *consistent with* a partial condition $\mathcal{H} = (\mathcal{H}_0, \mathcal{H}_1)$ if $\mathcal{H}_0 \subseteq \mathcal{F}_0^{\mathcal{C}}$ and $\mathcal{H}_1 \subseteq \mathcal{F}_1^{\mathcal{C}}$.

For each $\Omega \in \mathsf{Acc}$ we can now define the decision problem Ω-Consistency: Given a transition system $\mathcal{T} = (Q, \Sigma, q_0, \delta)$ and a partial condition $\mathcal{H} = (\mathcal{H}_0, \mathcal{H}_1)$ with $\mathcal{H}_0, \mathcal{H}_1 \subseteq 2^{Q\times\Sigma}$, the question is whether there exists an acceptance condition $\mathcal{C}$ of type Ω over $Q \times \Sigma$ that is consistent with $\mathcal{H}$. In the following we provide algorithms that decide Ω-consistency for the various acceptance types we introduced and investigate their complexity.

Büchi and generalized Büchi conditions. For a Büchi condition $F \subseteq Q \times \Sigma$ we know that every superset of some $X \subseteq Q \times \Sigma$ with $X \cap F \neq \emptyset$ clearly has a non-empty intersection with F. Based on this observation we can define an algorithm that computes for a given partial condition $\mathcal{H} = (\mathcal{H}_0, \mathcal{H}_1)$ a Büchi condition F which is consistent with $\mathcal{H}$. We forego a formal definition of the algorithm itself and instead define the partial function it computes, where a result of $\bot$ is used to indicate that no Büchi condition exists that is consistent with $\mathcal{H}$.

$$\mathsf{BuchiCons}(\mathcal{H}_0, \mathcal{H}_1) = \begin{cases} \texttt{return } \bot & \text{if } P \in \mathcal{H}_0 \text{ exists with } P \subseteq \bigcup \mathcal{H}_1 \\ \texttt{return } (Q \times \Sigma) \setminus \bigcup \mathcal{H}_1 & \text{otherwise} \end{cases}$$

It is easily verified that BuchiCons is computable in polynomial time and a formal proof for the correctness of this algorithm can be found in the full version of this paper.

With generalized Büchi conditions it is no longer guaranteed that the union of two negative sets $N, N' \in \mathcal{H}_1$ is also negative. Consider a generalized Büchi condition $\mathcal{B} = \{F, F'\}$ such that N has a non-empty intersection with F but not with F', whereas $N' \cap F = \emptyset$ and $N' \cap F' \neq \emptyset$. Then their union $F \cup F'$ has a non-empty intersection with both F and F' and hence satisfies $\mathcal{B}$. Therefore we first isolate the $\subseteq$-maximal sets $N_1, \ldots, N_k$ in $\mathcal{H}_1$. As before we give a function

$$\mathsf{genBuchiCons}(\mathcal{H}_0, \mathcal{H}_1) = \begin{cases} \texttt{return } \bot & \text{if } P \in \mathcal{H}_0 \text{ with } P \subseteq N_i \text{ exists} \\ \texttt{return } \{(Q \times \Sigma) \setminus N_i : i \leq k\} & \text{otherwise} \end{cases}$$

which maps a partial condition to a generalized Büchi condition that is consistent with $\mathcal{H}$ or $\bot$ if no such condition exists. It is again not difficult to see that an algorithm can compute genBuchiCons in polynomial time. A formal proof of the correctness of genBuchiCons can be found in the full version of this paper.

▶ **Theorem 1.** *The algorithm (gen)BuchiCons decides the (generalized) Büchi-*Consistency *problem in polynomial time and returns a corresponding acceptance condition if one exists.*

Parity Conditions. It is a well-known observation that for a given Muller condition $(\mathcal{F}_0, \mathcal{F}_1)$ there exists an equivalent parity condition κ if and only if $\mathcal{F}_0$ and $\mathcal{F}_1$ are union-closed [28]. We show an analogous statement for partial conditions, starting with the following lemma which establishes that if the union of positive and negative elements coincide, then no equivalent parity condition can be found.

▶ **Lemma 2.** *Let $\mathcal{H} = (\mathcal{H}_0, \mathcal{H}_1)$ be a consistent partial condition. If we have $P = N$ for $P = P_1 \cup \ldots \cup P_k$ and $N = N_1 \cup \ldots \cup N_l$ with $P_i \in \mathcal{H}_0$ and $N_j \in \mathcal{H}_1$ then there exists no parity condition that is consistent with $\mathcal{H}$*

It turns out that the opposite direction also holds, meaning if no such unions of positive and negative sets can be found, then an equivalent parity condition must exist. This implication arises as a consequence of the ParityCons algorithm we present later together with the proofs of its correctness. For a given partial condition ParityCons (see Algorithm 1) attempts to construct a chain of sets of transitions $Z_0 \supseteq Z_1 \supseteq \ldots \supseteq Z_{n-1}$ with alternating classifications $\sigma_i \in \{0, 1\}$, i.e., $\sigma_{i+1} = 1 - \sigma_i$ for $i < n - 1$. We refer to this as a *Zielonka path* because it corresponds to the Split or Zielonka tree representation of a parity condition [28, 11].

From such a Zielonka path one obtains a parity condition κ where $\kappa(q, a) = \sigma_0 + i$ for the maximal i such that $(q, a) \in Z_i$. On the other hand every parity condition κ with priorities C determines a chain $Z_0 \supseteq Z_1 \supseteq \ldots \supseteq Z_{|C|-1}$ and alternating classifications σ_i where $\sigma_0 = \min(C) \bmod 2$, $\sigma_{i+1} = 1 - \sigma_i$ and Z_i contains all state-symbol pairs whose color is greater or equal to $\sigma_0 + i$. To guarantee the existence of such an alternating chain, we assume that κ is optimal and contains no gaps, which can be ensured in polynomial time [9].

▶ **Example 3.** As an example consider a partial condition $\mathcal{H} = (\mathcal{H}_0, \mathcal{H}_1)$ with set inclusion diagram as shown on the left of Figure 1, where $\mathcal{H}_0$ contains the transition sets drawn with rounded border, and $\mathcal{H}_1$ those with rectangular border (the leaves of the tree, in this example). We assume an underlying transition system in which the transition sets in $\mathcal{H}$ are strongly connected. It is easily verified that $\mathcal{H}$ does not satisfy the condition of Lemma 2. Since we claimed the converse of Lemma 2 to be true, a parity condition that is consistent with $\mathcal{H}$ should exist. It turns out that such a parity condition requires 6 distinct priorities (the corresponding Zielonka path is shown on the right of Figure 1) even though there is at most one alternation between positive and negative sets along inclusion chains in $\mathcal{H}$. This is due to the fact that more alternations are introduced by unions of positive and negative sets.

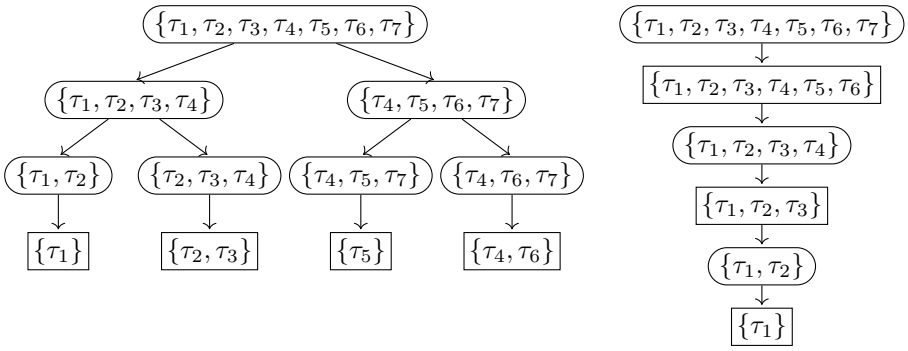


Figure 1 On the left an inclusion graph for the partial condition $\mathcal{H}$ from Example 3 can be seen in which positive elements are depicted with rounded and negative ones with rectangular borders. The path depicted on the right corresponds to a priority function κ with domain $\{0, 1, 2, 3, 4, 5\}$ such that $\tau_7 \mapsto 0, \tau_6 \mapsto 1, \tau_5 \mapsto 1, \tau_4 \mapsto 2, \tau_3 \mapsto 3, \tau_2 \mapsto 4, \tau_1 \mapsto 5$ which is the minimal parity condition that is consistent with $\mathcal{H}$.

We now present an algorithm that given a consistent partial condition $\mathcal{H}$ over $Q \times \Sigma$ constructs an equivalent parity condition with the least number of distinct priorities if one exists. As a simplification we assume that the set $Q \times \Sigma$ of all transitions is classified by $\mathcal{H}$, which enables us to use $Q \times \Sigma$ as the first set Z_0 of the chain that is constructed. We describe later how partial conditions that do not satisfy this assumption can be dealt with.

Algorithm 1 ParityCons.

Input: A consistent partial condition $\mathcal{H} = (\mathcal{H}_0, \mathcal{H}_1)$ with $Q \times \Sigma \in \mathcal{H}$
Output: A Zielonka path $(Z_0, \sigma_0), (Z_1, \sigma_1), \ldots, (Z_{n-1}, \sigma_{n-1})$
$Z_0 \leftarrow Q \times \Sigma$, $\sigma_0 \leftarrow \mathcal{H}(Q \times \Sigma)$, $i \leftarrow 0$
repeat
  $i \leftarrow i + 1$
  $Z \leftarrow \bigcup\{X \subseteq Z_{i-1} : \mathcal{H}(X) = 1 - \sigma_{i-1}\}$
  if $Z = Z_{i-1}$ **then**
    return *No consistent parity condition exists.*
  $Z_i \leftarrow Z$, $\sigma_i \leftarrow 1 - \sigma_{i-1}$
until $Z = \emptyset$
return $(Z_0, \sigma_0), \ldots, (Z_{i-1}, \sigma_{i-1})$

After Z_0 and its corresponding classification $\sigma_0 = \mathcal{H}(Z_0)$ have been determined, the algorithm computes Z_1 as the union of all $1 - \sigma_0$ subsets of Z_0. If this union coincides with Z_0 then the conditions for Lemma 2 are met and the algorithm terminates prematurely as no equivalent parity condition can exist. Otherwise this construction ensures that every strict superset of Z_1 receives the same classification as Z_0 from the constructed parity condition. This process is then repeated for Z_1 with $\sigma_1 = 1 - \sigma_0$, Z_2 with $\sigma_2 = 1 - \sigma_1$ and so on until no subsets of opposite classification remain. At this point the algorithm terminates and returns the constructed chain of sets of transitions together with their corresponding classification.

Proving the correctness of this approach forms the opposite direction of Lemma 2 as it entails that if no union of positive and negative sets as in Lemma 2 is found, an equivalent parity condition can be constructed. One restriction on the partial conditions that can be passed to ParityCons is that the set of all transitions, $Q \times \Sigma$, must be present in either $\mathcal{H}_0$ or $\mathcal{H}_1$. As these partial conditions arise from the infinity sets that words from a finite sample

induce, however, it is easily conceivable that there are many scenarios - for example when the automaton that we want to learn is made up of multiple SCCs - in which no word inducing $Q \times \Sigma$ exists. In this case we can simply define two extended partial conditions $\mathcal{H}^p$ and $\mathcal{H}^n$ in which $Q \times \Sigma$ is added as a positive or negative set respectively and execute ParityCons separately for each of them. If only one computation results in a Zielonka path we are done, otherwise the two resulting paths are compared with regard to their length and the longer one is discarded.

▶ **Theorem 4.** ParityCons *decides Parity-*Consistency *in polynomial time and returns a corresponding parity condition with a minimal number of priorities if one exists.*

Proof (sketch). We proceed in two steps and first show that the classification obtained by the Zielonka path computed by ParityCons are indeed consistent with the original partial condition. Subsequently one shows that if the computation exits prematurely, then there exist positive and negative sets whose unions coincide, which by Lemma 2 means that no equivalent parity condition exists. ◀

Rabin Conditions. We now turn towards computing an equivalent Rabin condition based on a given partial condition, for which we again utilize an observation about union-closedness. Specifically, a Muller condition is equivalent to a Rabin condition if and only if $\mathcal{F}_1$ is union-closed [28]. The algorithm RabinCons (see Algorithm 2) computes for each positive set P in $\mathcal{H}$ a separate Rabin pair (E_P, F_P) in which each transition that is not part of P belongs to E_P and every transition which does not occur in a negative subloop of P belongs to F_P. In case a positive loop is equal to the union of its maximal negative subloops, no equivalent Rabin condition can be found as the condition on union-closedness outlined above is violated.

Algorithm 2 RabinCons.

Input: A consistent partial condition $\mathcal{H} = (\mathcal{H}_0, \mathcal{H}_1)$
Output: A Rabin condition $\mathcal{R}$ consistent with $\mathcal{H}$
$\mathcal{R} \leftarrow \emptyset$
foreach $P \in \mathcal{H}_0$ **do**
  $N_1, \ldots, N_k \leftarrow$ maximal sets in $\mathcal{P}(P) \cap \mathcal{H}_1$
  $E_P \leftarrow (Q \times \Sigma) \setminus P$
  $F_P \leftarrow P \setminus (N_1 \cup \ldots \cup N_k)$
  if $F_P = \emptyset$ **then**
    return *No consistent Rabin condition exists*
  $\mathcal{R} \leftarrow \mathcal{R} \cup \{(E_P, F_P)\}$
return $\mathcal{R}$

▶ **Theorem 5.** *The algorithm* RabinCons *decides Rabin-*Consistency *in polynomial time and returns a corresponding Rabin condition if one exists.*

A Rabin condition produced by RabinCons has $|\mathcal{H}_0|$ pairs and is not guaranteed to have the minimal number of pairs. Even though it is possible to find optimizations which might make use of the underlying structure with regard to strongly connected components and subset relations between positive and negative loops, we now illustrate why the computation of an optimal Rabin condition (with a minimal number of pairs) is NP-hard.

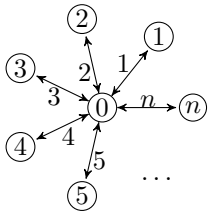


Figure 2 This figure contains a depiction of the transition system $\mathcal{T}_\mathcal{G}$, which can be used to show NP-completeness of k-generalized Büchi-Consistency and k-Rabin-Consistency.

Fixed-size consistency. For each acceptance type $\Omega \in \{\text{generalized Büchi}, \text{Parity}, \text{Rabin}\}$ and every natural number $k \in \mathbb{N}$ we define the decision problem k-Ω-Consistency: Given a transition system $\mathcal{T}$ and a consistent partial condition $\mathcal{H}$ the question is whether there is an acceptance condition $\mathcal{C}$ of type Ω in $\mathcal{T}$ which is consistent with $\mathcal{H}$ such that $|\mathcal{C}| \leq k$. The algorithm ParityCons we provided earlier decides k-Parity-Consistency in polynomial time, however finding a Rabin or generalized Büchi condition of bounded size turns out to be much more difficult.

Intuitively, the difficulty in finding an optimal generalized Büchi condition with at most k components arises from the fact that the union of two negative sets is not necessarily guaranteed to also be negative. As there are in general exponentially many possible ways of partitioning the transitions into k sets, a procedure for constructing an optimal generalized Büchi condition would need to consider all of them. In the following we establish that the fixed-size consistency problem for generalized Büchi conditions is already NP-complete when $k = 3$. This is done by giving a reduction from 3-Coloring for directed graphs, which is known to be NP-complete [14].

▶ **Lemma 6.** *3-generalized Büchi-*Consistency *is NP-complete.*

Proof. Let $\mathcal{G} = (V, E)$ be a finite directed graph with $V = \{v_1, v_2, \ldots, v_n\}$. We define the deterministic partial transition system

$$\mathcal{T}_\mathcal{G} = (\{0, 1, 2, \ldots, n\}, \{1, 2, \ldots, n\}, 0, \delta) \text{ with } \delta(q, a) = \begin{cases} a \text{ if } q = 0 \\ 0 \text{ if } q = a \\ \bot \text{ otherwise} \end{cases}$$

which is depicted in Figure 2. Note that it is possible to construct an equivalent transition system over a binary alphabet $\Sigma' = \{a, b\}$ by encoding $i \in \Sigma$ as $a^i b$. Thus our choice of Σ depending on the size of the graph merely serves to simplify notation in the following. We define a sample $S_\mathcal{G} = (P_\mathcal{G}, N_\mathcal{G})$ with

$$P_\mathcal{G} = \{p_{ij} : (v_i, v_j) \in E\} \text{ and } N_\mathcal{G} = \{n_i : 0 < i \leq n\} \text{ where } p_{ij} = (iijj)^\omega, n_i = i^\omega$$

In the following we use $\bar{p_{ij}}$ and $\bar{n_i}$ to refer to the infinity set of the unique run of $\mathcal{T}_\mathcal{G}$ on p_{ij} and n_i respectively. Let $c : V \to \{1, 2, 3\}$ be a 3-coloring for V such that $c(v_i) \neq c(v_j)$ for all $(v_i, v_j) \in E$. We construct a generalized Büchi condition $\mathcal{B}_\mathcal{G} = (F_1, F_2, F_3)$ with $F_k = \{i : c(v_i) \neq k\}$, witnessing membership in 3-generalized Büchi-Consistency. For all $i \leq n$ we have for $k = c(v_i)$ that $\bar{n_i} \cap F_k = \{0, i\} \cap F_k = \emptyset$ and thus $n_i \notin L(\mathcal{T}, \mathcal{B}_\mathcal{G})$. On the other hand $\bar{p_{ij}} \cap F_k = \{0, i, j\} \cap F_k \neq \emptyset$ for all k as $c(v_i) \neq c(v_j)$ is guaranteed for all $(v_i, v_j) \in E$ by the coloring function c. Hence $p_{ij} \in L(\mathcal{T}, \mathcal{B}_\mathcal{G})$ and the constructed condition is indeed consistent with the sample.

For the other direction assume that there exists a generalized Büchi condition $\mathcal{B} = (F_1, F_2, F_3)$ such that $\langle \mathcal{T}, \mathcal{B} \rangle$ is consistent with S. Clearly it must hold that $F_1 \cap F_2 \cap F_3 = \emptyset$ as otherwise there would exist some word $n_i \in N_\mathcal{G}$ with $\bar{n_i} \cap F_k = \{0, i\} \cap F_k \neq \emptyset$ for all k, which would contradict consistency with S. We can now define a coloring $c : V \to \{1, 2, 3\}$ with $c(v_i) = \min\{k : i \notin F_k\}$. For any $v_i, v_j \in V$ with $c(v_i) = c(v_j) = k$ we have $(v_i, v_j) \notin E$. If not then there would exist a word $p_{ij} \in P_\mathcal{G}$ for which consistency guarantees that $\bar{p_{ij}} \cap F_k = \{0, i, j\} \cap F_k \neq \emptyset$, which can only hold if v_i and v_j are assigned different colors. Thus c is indeed a valid 3-coloring, which concludes the reduction proof.

Membership in NP holds as it is possible to verify for a guessed generalized Büchi condition $\mathcal{B}$ of size 3 whether $\langle \mathcal{T}, \mathcal{B} \rangle$ is consistent with S in polynomial time by iterating over all $w \in P_\mathcal{G} \cup N_\mathcal{G}$ and verifying adequate acceptance/rejection by $\langle \mathcal{T}, \mathcal{B} \rangle$. ◀

A similar reduction can be used to show the NP-hardness of k-Rabin-Consistency as well. This leads to the following theorem, which establishes the complexity of all fixed-size consistency decision problems we defined above.

▶ **Theorem 7.** *k-Parity-Consistency is solvable in polynomial time. For $k > 2$ both k-generalized Büchi-Consistency and k-Rabin-Consistency are NP-complete.*

4 Passive learning

Our procedure for the construction of a deterministic partial transition system is inspired by the well known regular positive negative inference (RPNI) algorithm through which deterministic finite automata can be constructed [21]. RPNI first constructs a prefix tree automaton which accepts precisely the positive sample words from S_+ and subsequently attempts to merge states of this automaton in canonical order. If a merge introduces an inconsistency with the sample (i.e. the resulting automaton accepts a word in S_-) it is reverted. Otherwise the algorithm continues with the resulting automaton until no further merges are possible at which point it terminates.

When attempting to transfer this principle to infinite words, it is difficult to find a suitable counterpart for the prefix tree automaton. If we simply attached disjoint loops to the prefix tree at a certain depth, the resulting transition system could certainly be equipped with an acceptance condition such that it accepts precisely S_+. However, through the introduction of loops with a fixed length that cannot be resolved during the execution, we already determine parts of the structure of the resulting automaton. Instead, we start with a transition system consisting of a single initial state and attempt to introduce new transitions in a specific order (which is reminiscent of the algorithm presented in [6]).

The resulting algorithm Sprout is shown in Algorithm 3. In each iteration we begin by computing Escapes$(S_+, \mathcal{T})$, the set of all prefixes of words in S_+ which are escaping in $\mathcal{T}$. From this set we now determine the word with the minimal escape-prefix ua in length-lexicographic order. The existing states are then tested as a target for the missing transition in canonical order and if the resulting transition system is Ω-consistent with the sample, we continue with the next escaping word. Checking for consistency is done by using the results from Section 3 and ensuring that no pair of indistinguishable words in $S_+ \times S_-$ exists, both of which are possible in polynomial time. If no suitable target can be found, a new state is introduced instead. See Figure 3 for an illustration. Note that the order in which states are checked as a potential transition target coincides with the order in which merges are attempted in RPNI.

Algorithm 3 Sprout.

Input: A Sample $S = (S_+, S_-)$ over the alphabet Σ and an acceptance type $\Omega \in \mathsf{Acc}$
Output: The deterministic Ω-automaton $\mathcal{A} = (Q, \Sigma, q_0, \delta, \mathcal{C})$ consistent with S
$Q \leftarrow \{\varepsilon\},\ \delta \leftarrow \emptyset,\ \mathcal{T} \leftarrow (Q, \Sigma, \varepsilon, \delta)$
while $\mathsf{Escapes}(S_+, \mathcal{T}) \neq \emptyset$ **do**
  $ua \leftarrow$ length-lexicographic minimal escape-prefix of a word in S_+
  if $|u| > \mathsf{Thres}(S, \mathcal{T})$ **then**
    return $\mathsf{Aut}(\mathsf{Extend}(Q, \Sigma, \varepsilon, \delta, S_+, S_-), S, \Omega)$
  forall $q \in Q$ *in canonical order* **do**
    $\delta' \leftarrow \delta \cup \{\hat{u} \xrightarrow{a} q\}$ for the $\hat{u} \in Q$ with $\delta^*(\varepsilon, u) = \hat{u}$
    if $(Q, \Sigma, \varepsilon, \delta')$ *is* Ω*-consistent with* S **then**
      $\delta \leftarrow \delta'$ and **continue** with the next escpaing word
  $Q \leftarrow Q \cup \{\hat{u}a\},\ \delta \leftarrow \delta \cup \{\hat{u} \xrightarrow{a} \hat{u}a\}$ for the $\hat{u} \in Q$ with $\delta^*(\varepsilon, u) = \hat{u}$
return $\mathsf{Aut}(\mathcal{T}, S, \Omega)$

Unfortunately there exist samples for which this approach of introducing transitions does not terminate. When executed on $S = (\{(baa)^\omega\}, \{(ab)^\omega, (ba)^\omega, (babaa)^\omega\})$ for example, the algorithm would not terminate and instead construct an infinite b-chain with a-loops on each state. We therefore introduce a threshold on the maximal length of escape-prefixes that are considered in the algorithm. Once this threshold is exceeded, the algorithm terminates. We have choosen the threshold such that we can show completeness for IRC, which works for $\mathsf{Thres}(S, \mathcal{T}) = l_b + l_e^2 + 1$, where l_e and l_b denote the maximal length of u and v for any sample word $uv^\omega \in S$. Intuitively, this value is sufficient to obtain completeness for IRC as any two sample words must have already differed in at least one position once it is exceeded.

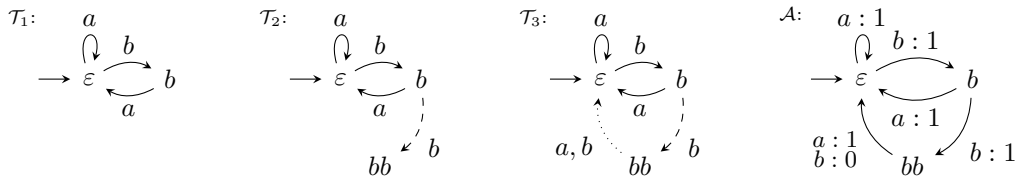
If the threshold is exceeded before a transition system is found that is consistent with the sample and has no escaping words from S_+, the transition system is extended with disjoint loops that guarantee acceptance of the remaining words in S_+ through the function Extend, which we describe in the following. Assume that the algorithm has constructed a transition system $\mathcal{T} = (Q, \Sigma, \varepsilon, \delta)$ for which it then encounters an escape-prefix exceeding the defined threshold. For each state $q \in Q$ we collect all exit strings that leave $\mathcal{T}$ from q in a set E_q. Note that since the shortest escape-prefix in $\mathcal{T}$ exceeded the threshold, each word in E_q must be of the form u^ω for some $u \in \Sigma^+$ and we can write $E_q = \{u_1^\omega, \dots, u_k^\omega\}$.

For each state q such that $E_q \neq \emptyset$ we now construct the transition system $\mathcal{T}_{E_q}^{\circlearrowleft}$ in which exactly those words that belong to E_q induce loops. To prevent any unintended words from being accepted, we additionally ensure that the initial state of $\mathcal{T}_{E_q}^{\circlearrowleft}$ is transient (meaning it cannot be reached from any state within $\mathcal{T}_{E_q}^{\circlearrowleft}$). In the following we use $\mathsf{Prf}(u)$ for a word $u \in \Sigma^*$ to denote the set of all prefixes of u. Formally we define $\mathcal{T}_{E_q}^{\circlearrowleft} = (Q_{E_q}^{\circlearrowleft}, \Sigma, q_0, \delta_{E_q}^{\circlearrowleft})$ with

$$Q_{E_q}^{\circlearrowleft} = \{q_0\} \cup \bigcup_{u^\omega \in E_q} \mathsf{Prf}(u)$$

$$\delta_{E_q}^{\circlearrowleft}(w, a) = \begin{cases} a & \text{if } w = q_0 \text{ and } a \in \Sigma \cap Q_{E_q}^{\circlearrowleft} \\ \varepsilon & \text{if } (wa)^\omega \in E_q \\ wa & \text{if } wa \in Q_{E_q}^{\circlearrowleft} \text{ and } (wa)^\omega \notin E_q \\ \bot & \text{otherwise} \end{cases}$$

It is easy to see that q_0 is indeed transient in $\mathcal{T}_{E_q}^{\circlearrowleft}$ and we can clearly find a Büchi (and thus also a generalized Büchi, Rabin and Parity) condition such that every word in E_q induces an accepting run in $\mathcal{T}_{E_q}^{\circlearrowleft}$. By attaching the corresponding $\mathcal{T}_{E_q}^{\circlearrowleft}$ to each state q for which E_q is non-empty, we obtain a transition system in which no word from S_+ is escaping.



$S_+ = \{b^\omega, (bbbabbaba)^\omega, (abbb)^\omega, (babb)^\omega, (bbab)^\omega, (bbba)^\omega\}$ and $S_- = \{a^\omega, (ba)^\omega, (bba)^\omega\}$

Figure 3 In this figure three transition systems that arise during the execution of Sprout on the sample $S = (S_+, S_-)$ are depicted. The dashed transition cannot lead to ε as otherwise the union of the infinity sets of $(ba)^\omega$ and $(bba)^\omega$ would coincide with that of $(bbbabbaba)^\omega$ and thus no consistent parity condition exists. Similarly a self-loop on b would mean that the infinity sets induced by $(babb)^\omega$ and $(bba)^\omega$ would coincide. Thus the b-transition must lead to a new state bb. On the right we can see the DPA obtained by augmenting $\mathcal{T}_3$ with the parity function computed by ParityCons on the partial condition induced by S.

Once the main loop terminates, the function Aut is called, which uses the results from Section 3 to compute an automaton that is Ω-consistent with S, which is then returned.

▶ **Proposition 8.** *For a given sample S and an acceptance type $\Omega \in$ Acc the algorithm* Sprout *computes in polynomial time an automaton of type Ω that is consistent with S.*

While Sprout cannot learn all regular ω-languages in the limit (see Proposition 10), we can show completeness for languages with an IRC.

▶ **Theorem 9.** *The algorithm* Sprout *learns every Ω-IRC language L for $\Omega \in$ Acc in the limit with polynomial time and data.*

Proof (sketch). We describe the properties that a sample $S = (S_+, S_-)$ has to satisfy in order to be *characteristic* for an Ω-IRC language L:

- The set of prefixes of S_+ has to contain for each $\sim_L$ equivalence class the minimal word in length-lexicographic order on which it is reached.
- Further the sample needs to contain words with which all pairs of equivalence classes can be separated.
- Finally S needs to contain sufficient information about the acceptance condition of an automaton recognizing L.

The first two requirements can be satisfied in a similar way as for the original RPNI algorithm [21]. For parity conditions this has already been investigated in [4]. Below we give a description for Rabin conditions. Detailed definitions for the remaining types of acceptance conditions we introduced can be found in the full version of this paper.

A sample $S_\mathcal{R} = (S_+, S_-)$ capturing a Rabin condition $\mathcal{R}$ can be obtained as follows: For each pair (E_i, F_i) we remove all transitions in E_i from the transition system that $\mathcal{R}$ is defined in, decompose the result into its SCCs $C_1, \ldots, C_k$ and compute sets K_i consisting of all transitions in C_i. If the set of all transitions K_i in such an SCC satisfies $\mathcal{R}$ we add a word w_i inducing K_i to S_+, otherwise w_i is added to S_-. For each accepting K_i we then remove all transitions in an F_j for which $K_i \cap E_j = \emptyset$, and decompose the resulting transition system into its SCCs $D_1, \ldots, D_l$. These are the maximal negative subloops of K_i and for each D_j a word visiting all transitions in D_j is added to S_-. ◀

While every Ω-IRC language can be learned through a characteristic sample, the same does not hold for arbitrary ω-regular languages as the following proposition establishes.

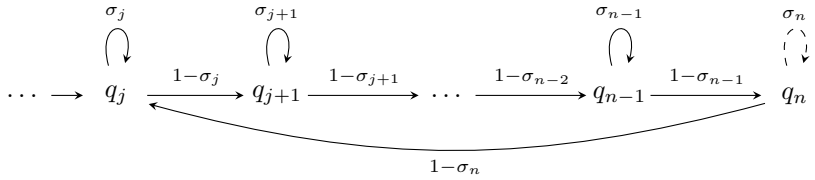


Figure 4 In this figure an excerpt of the transition system for the proof of Proposition 10 is depicted. The transition from q_n to q_j forms a closed loop and words $w_1 \in L_\vee, w_2 \notin L_\vee$ which induce the same infinity set can be found. Based on the existence of these words we can conclude that Sprout constructs a chain with self-loops on each state when attempting to learn an automaton recognizing $L_\vee$.

▶ **Proposition 10.** *The language $L_\vee = \{w \in \{a,b\}^\omega : aaaa$ occurrs infinitely often in w or $bbbb$ occurrs infinitely often in $w\}$ cannot be learned by* Sprout.

Proof. To simplify notation, we exchange the alphabet and use $\Sigma = \{0,1\}$ instead, as it allows arithmetic on the symbols in Σ. We prove this claim by showing through induction that the transition system constructed by Sprout must be a chain with loops on each state. Specifically we show that every intermediate transition system $\mathcal{T} = (Q, \Sigma, q_0, \delta)$ with $Q = \{q_0, q_1, \ldots, q_n\}$ created by Sprout before the threshold is exceeded is either not Ω-consistent with $L_\vee$ for any $\Omega \in$ Acc or the following holds:

- for each $i < n$ there exists a symbol $\sigma \in \Sigma$ such that $\delta(q_i, \sigma) = q_i$ and $\delta(q_i, 1-\sigma) = q_{i+1}$
- if q_n has an outgoing transition on some $\sigma \in \Sigma$ then $\delta(q_n, \sigma) = q_n$ and $\delta(q_n, 1-\sigma) = \bot$

The initial transition system is clearly Ω-consistent with $L_\vee$ for all $\Omega \in$ Acc. Further it trivially satisfies the two outlined conditions as it has only one state, for which no outgoing transitions exist. For the induction step assume that Sprout has constructed a transition system $\mathcal{T} = (Q, \Sigma, q_0, \delta)$ with $Q = \{q_0, q_1, \ldots, q_n\}$ for which the claim holds. We now show that the next inserted transition either introduces an inconsistency with $L_\vee$ or it leads to a transition system that also satisfies the two conditions.

If a transition from q_n to some q_j with $j < n$ were inserted, then a closed cycle is formed. As q_j is reachable there must exist some word $u \in \Sigma^*$ such that $\delta^*(q_0, u) = q_j$. Consider now the word $v \in \Sigma^*$ such that $\delta^*(q_j, v) = q_j$ and the letters in v are such that they alternate between taking the self-loop and moving to the next state along the cycle. If the loop on q_n does not exist, then v just transitions back to q_j at this point. As can be seen in Figure 4, no alphabet symbol can occur more than once in a row in v if the dashed self-loop on q_n is present. Otherwise at most three consecutive occurrences of the same symbol can appear in v and we clearly have that $w_1 = uv^\omega \notin L_\vee$. Consider now a word w_2 which takes each self-loop on the cycle four times before moving to the next state. This means $w_2 \in L_\vee$ but because the infinity sets induced by w_1 and w_2 coincide (as both words take all possible transitions infinitely often), an automaton containing such a closed cycle cannot be consistent with $L_\vee$.

We have thus shown that no transition can lead from q_n back to a state q_j with $j < n$. If q_n has no outgoing transitions, then a self-loop on the currently escaping symbol is inserted as it clearly does not introduce an inconsistency. On the other hand if q_n already has a self-loop on some symbol $\sigma \in \Sigma$, then the transition on $1-\sigma$ must lead to a new state q_{n+1} as otherwise $(ab)^\omega \notin L_\vee$ and $(aaab)^\omega \in L_\vee$ would induce the same infinity set. Thus the Sprout algorithm indeed constructs a chain with self-loops until it eventually exceeds the threshold. Once this happens, the transition system is extended such that it accepts precisely the positive sample words. As the sample is finite, the resulting automaton cannot recognize $L_\vee$ since there will always be some word $w \in L_\vee$ that is not present in the sample. ◀

However on the other hand Sprout is not limited to learning automata for languages with IRC of some type. In the following proposition we give an infinite family of languages which are not in Ω-IRC for any $\Omega \in \mathsf{Acc}$, and have polynomial size characteristic samples for Sprout.

▶ **Proposition 11.** *For $i > 1$, consider $L_i = (\Sigma^* b^i)^\omega$ and the sample $S^i = (S^i_+, S^i_-)$ with $S^i_+ = \{b^\omega, (b^i a b^{i-1} a \dots b^2 a b^1 a)^\omega\} \cup \{(b^j a b^k)^\omega : j + k = i\}$ and $S^i_- = \{(b^j a)^\omega : j < i\}$. Then S^i is a characteristic sample for L_i and the learner* Sprout *with parity as target condition. (The sample for $i = 3$ is used in the example in Figure 3.)*

Proof. In the following we show that Sprout constructs a DPA for the language L_i from the characteristic sample S^i. Note first that the exit-strings of any two sample words are distinct for every transition system constructed by Sprout, since all words in S^i consist of only a periodic part. Further in every word $v^\omega \in S^i_+$ the infix b^i occurs, which means that an infinite run on any positive sample word is only possible in a transition system that permits i consecutive transitions on the symbol b.

Initially, the algorithm inserts a self-loop on a as no sample words prevent this. Subsequently the b-transition cannot be a self-loop as otherwise the infinity sets induced by positive and negative sample words would coincide. Thus a new state is added to which the b-transition from ε leads. We now proceed inductively to show that a b-chain of length $i-1$ with a-transitions leading back to the initial state is created. We will identify each state on this chain with the minimal word of the form b^j that reaches it.

Formally such a chain satisfies that for all $j < i$ we have $\delta^*(\varepsilon, b^j) = b^j$ and $\delta^*(\varepsilon, b^k a) = \varepsilon$ for all $k < j$. The base case for $j = 1$ has already been described above so assume now that the statement holds for $j - 1$ and consider the two transitions that Sprout inserts for the state b^{j-1}. We see that inserting an a-transition from b^{j-1} to ε does not introduce an inconsistency. This is because as outlined above no positive sample word induces an infinite run and the exit string of any two sample words must be distinct.

It remains to be shown that the b-transition from b^{j-1} must lead to a new state b^j. To see this assume to the contrary that the introduction of a b-transition from b^{j-1} to some b^l with $l < j$ leads to a transition system $\mathcal{T}'$ which is Parity-consistent with S^i. It is not hard to see that the infinity set P induced by the positive sample word $(b^i a b^{i-1} \dots b^1 a)^\omega$ contains all transitions in $\mathcal{T}'$. Now let $N_0, N_1, \dots, N_j$ be the infinity sets induced by the negative sample words $a^\omega, (ba)^\omega, \dots, (b^j a)^\omega$. It is easily verified that $P = N_0 \cup N_1 \cup \dots \cup N_j$, thus satisfying the conditions for Lemma 2. This means that $\mathcal{T}'$ cannot be Parity-consistent with S^i and hence no b-transition from b^{j-1} to any b^l with $l < j$ is kept.

Once this b-chain of length $i-1$ is constructed, we simply need to verify that inserting both the a- and b-transition from b^{i-1} to ε does not lead to an inconsistent transition system. Since only positive sample words contain i consecutive occurrences of b, the b-transition from b^{i-1} to ε occurs exclusively in the infinity set induced by positive but not negative words. Thus a consistent parity condition exists and Sprout constructs a DPA recognizing L_i. ◀

5 Active Learning

We consider the standard minimal adequate teacher (MAT) active learning scenario [1], in which the learning algorithm has access to a teacher that can answer membership queries and equivalence queries for the target language, and returns a counterexample if the automaton for an equivalence query is not correct. A natural extension to ω-automata considers membership queries for ultimately periodic words and equivalence queries with ultimately periodic words as counterexamples (see [16]).

Since there is a polynomial time active learning algorithm for deterministic weak automata [16], a natural next candidate for polynomial time active learning are deterministic automata with an informative right congruence. However, the theorem below basically shows that this class is as hard for active learning as general regular ω-languages.

▶ **Theorem 12.** *Let $\Omega \in \mathsf{Acc}$ be an acceptance type, and consider the active learning setting with membership and equivalence queries for ultimately periodic words. There is a polynomial time active learning algorithm for deterministic automata of type Ω with informative right congruence if, and only if, there is a polynomial time active learning algorithm for general deterministic automata of type Ω.*

Proof (sketch). Assume that AL_{IRC} is an active learning algorithm for automata with informative right congruence of type Ω. The arguments used below work for all acceptance types $\Omega \in \mathsf{Acc}$. For simplicity we use the parity condition in the following.

Our goal is to use AL_{IRC} in order to define an active learning algorithm AL for general DPA that runs in polynomial time if AL_{IRC} does. The rough idea is as follows: We have to learn an automaton $\mathcal{A}$ for a target language $L \subseteq \Sigma^\omega$ that does not have an IRC, in general. Such an automaton $\mathcal{A}$ can be turned into an automaton with IRC by adding new letters to the alphabet, and then extending the automaton such that from each state a different word over these new letters is accepted. Restricted to the original alphabet, this extended automaton still accepts the same language as before. Since the new automaton has an IRC, we can use AL_{IRC} to learn it. The only problem with this approach is that we do not know the target automaton $\mathcal{A}$, so we cannot simply extend it and let AL_{IRC} learn the extension. However, we can simulate a teacher for AL_{IRC} that answers queries of AL_{IRC} such that these answers are consistent with such an extension of $\mathcal{A}$. We give the answers such that they only reveal information on the original target language L. Hence, AL_{IRC} first has to learn, in some sense, an automaton for L in order to obtain information on the newly added letters in the extension.

More formally, define an extended alphabet $\Sigma_\star = \Sigma \dot\cup \{\star, 0, 1\}$ with new letters $\star, 0, 1$ that do not occur in Σ. Now let $L \subseteq \Sigma^\omega$ be a target language which we want to learn. Our algorithm AL simulates AL_{IRC} over the alphabet $\Sigma_\star$. Note that AL has access to a teacher T that answers queries for the language L over the alphabet Σ. We define a teacher T_{IRC} that answers queries that are asked by AL_{IRC} during its simulation as follows:

- *Membership query for a word $w = uv^\omega$:* If none of the newly introduced symbols occur in w, i.e. $w \in \Sigma^\omega$ then we simply copy the answer $T(w)$. Otherwise w must contain $0, 1$ or $\star$ in which case T_{IRC} always gives a negative answer.
- *Equivalence query for an automaton $\mathcal{A}$:* We construct a new automaton $\mathcal{B}$ by removing from $\mathcal{A}$ all transitions on symbols 0, 1 or $\star$ and pruning any unreachable states. $\mathcal{B}$ is then given to T for an equivalence query. If $T(\mathcal{B})$ returns a counterexample w, then this is used as the result of $T_{IRC}(\mathcal{A})$.
  Otherwise the automaton $\mathcal{B}$ must recognize the target language L. In this case, the simulation of AL_{IRC} is stopped, and our algorithm AL returns $\mathcal{B}$.

It can be shown that this algorithm AL learns the target language L in polynomial time if AL_{IRC} is a polynomial time algorithm. ◀

So the property of an IRC does not help for active learning, while for passive learning in the limit it seems to make the problem simpler. We finish this section with the observation that polynomial time active learning is at least as hard as learning in the limit with polynomial time and data, given that the class $\mathcal{K}$ of target automata satisfies the following properties (which are satisfied by standard classes of deterministic automata):

- (P1) It is decidable in polynomial time if a given word is accepted by a given automaton from $\mathcal{K}$.
- (P2) For a given sample S, one can construct in polynomial time an automaton from $\mathcal{K}$ that is consistent with S.
- (P3) If two automata from $\mathcal{K}$ are not equivalent, then there exists a word of polynomial size witnessing the difference.

▶ **Proposition 13.** *Consider a class $\mathcal{K}$ of finite automata for which properties (P1)–(P3) are satisfied. If there is a polynomial time active learning algorithm for $\mathcal{K}$, then $\mathcal{K}$ can be learned in the limit with polynomial time and data.*

Proof (sketch). Assume that there is a polynomial time active learning algorithm $AL_{\mathcal{K}}$ for target automata from $\mathcal{K}$. A passive learner can simulate an execution of $AL_{\mathcal{K}}$ in which equivalence queries are always answered with the smallest counterexample. A characteristic sample can be constructed from all the words that are used in such an execution of $AL_{\mathcal{K}}$. ◀

6 Conclusion

We have presented polynomial time algorithms for checking the consistency of a (partial) deterministic transition system with a set of positive and negative ultimately periodic words for the acceptance conditions Büchi, generalized Büchi, parity, and Rabin. Since co-Büchi and Streett conditions are dual to Büchi and Rabin conditions, respectively, one also obtains algorithms for these conditions by flipping negative and positive examples.

The consistency algorithms allow us to extend the principle of the RPNI algorithm from finite to infinite words, leading to the polynomial time algorithm Sprout that constructs a deterministic ω-automaton from given ultimately periodic examples. We have shown that Sprout can learn deterministic automata for languages with an IRC in the limit with polynomial time and data. While Sprout is not restricted to IRC languages, there are regular ω-langauges which it cannot learn. It is obviously an interesting open question whether there is an algorithm that learns deterministic automata for general regular ω-languages with polynomial time and data. Our results in Section 5 show that finding such an algorithm is not more difficult than finding an active learning algorithm that learns deterministic automata for IRC languages from membership and equivalence queries.

References

1 Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987. doi:10.1016/0890-5401(87)90052-6.

2 Dana Angluin and Dana Fisman. Learning regular omega languages. *Theor. Comput. Sci.*, 650:57–72, 2016. doi:10.1016/j.tcs.2016.07.031.

3 Dana Angluin and Dana Fisman. Regular omega-languages with an informative right congruence. In *Proceedings Ninth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2018, Saarbrücken, Germany, 26-28th September 2018*, volume 277 of *EPTCS*, pages 265–279, 2018. doi:10.4204/EPTCS.277.19.

4 Dana Angluin, Dana Fisman, and Yaara Shoval. Polynomial identification of ømega-automata. In Armin Biere and David Parker, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Part II*, volume 12079 of *Lecture Notes in Computer Science*, pages 325–343. Springer, 2020. doi:10.1007/978-3-030-45237-7_20.

5 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

6 Andreas Birkendorf, Andreas Böker, and Hans Simon. Learning deterministic finite automata from smallest counterexamples. *SIAM J. Discrete Math.*, 13:465–491, January 2000. doi:10.1137/S0895480198340943.

7 J Richard Büchi. On a decision method in restricted second order arithmetic, logic, methodology and philosophy of science (proc. 1960 internat. congr.), 1962.

8 Rafael C. Carrasco and José Oncina. Learning stochastic regular grammars by means of a state merging method. In *Grammatical Inference and Applications, Second International Colloquium, ICGI-94, Alicante, Spain, September 21-23, 1994, Proceedings*, volume 862 of *Lecture Notes in Computer Science*, pages 139–152. Springer, 1994. doi:10.1007/3-540-58473-0_144.

9 Olivier Carton and Ramón Maceiras. Computing the rabin index of a parity automaton. *RAIRO – Theoretical Informatics and Applications – Informatique Théorique et Applications*, 33(6):495–505, 1999. URL: http://www.numdam.org/item/ITA_1999__33_6_495_0/.

10 Colin De La Higuera. Characteristic sets for polynomial grammatical inference. In Laurent Miclet and Colin de la Higuera, editors, *Grammatical Interference: Learning Syntax from Sentences*, pages 59–71, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

11 Stefan Dziembowski, Marcin Jurdziński, and Igor Walukiewicz. How much memory is needed to win infinite games? In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science, LICS '97*, pages 99–110, Los Alamitos, California, 1997. IEEE Computer Society Press. doi:10.1109/lics.1997.614939.

12 E. Mark Gold. Complexity of automaton identification from given data. *Inf. Control.*, 37(3):302–320, 1978. doi:10.1016/S0019-9958(78)90562-4.

13 John E. Hopcroft and Jeffrey D. Ullman. *Formal Languages and their Relation to Automata*. Addison-Wesley, 1969.

14 Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972. doi:10.1007/978-1-4684-2001-2_9.

15 Christof Löding and Anton Pirogov. Determinization of Büchi automata: Unifying the approaches of Safra and Muller-Schupp. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019*, volume 132 of *LIPIcs*, pages 120:1–120:13. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.120.

16 Oded Maler and Amir Pnueli. On the learnability of infinitary regular sets. *Inf. Comput.*, 118(2):316–326, 1995. doi:10.1006/inco.1995.1070.

17 Hua Mao, Yingke Chen, Manfred Jaeger, Thomas D. Nielsen, Kim G. Larsen, and Brian Nielsen. Learning probabilistic automata for model checking. In *Eighth International Conference on Quantitative Evaluation of Systems, QEST 2011, Aachen, Germany, 5-8 September, 2011*, pages 111–120. IEEE Computer Society, 2011. doi:10.1109/QEST.2011.21.

18 Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. Strix: Explicit reactive synthesis strikes back! In *Computer Aided Verification – 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, pages 578–586, 2018. doi:10.1007/978-3-319-96145-3_31.

19 Jakub Michaliszyn and Jan Otop. Learning deterministic automata on infinite words. In *ECAI 2020 – 24th European Conference on Artificial Intelligence*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 2370–2377. IOS Press, 2020. doi:10.3233/FAIA200367.

20 José Oncina, Pedro García, and Enrique Vidal. Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(5):448–458, 1993. doi:10.1109/34.211465.

21 Jose Oncina and Pedro García. Inferring regular languages in polynomial update time. *World Scientific*, January 1992. doi:10.1142/9789812797902_0004.

22 Nir Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Proceedings of the 21st IEEE Symposium on Logic in Computer Science (LICS 2006)*, pages 255–264. IEEE Computer Society, 2006. doi:10.2168/LMCS-3(3:5)2007.

23 Shmuel Safra. On the complexity of omega-automata. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science, FoCS '88*, pages 319–327, Los Alamitos, California, 1988. IEEE Computer Society Press. doi:10.1109/SFCS.1988.21948.

24 Sven Schewe. Tighter bounds for the determinisation of Büchi automata. In *Proceedings of Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009*, volume 5504 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2009. doi:10.1007/978-3-642-00596-1_13.

25 Sven Schewe. Beyond Hyper-Minimisation – Minimising DBAs and DPAs is NP-Complete. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 400–411, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSTTCS.2010.400.

26 Wolfgang Thomas. *Automata on Infinite Objects*, page 133–191. MIT Press, Cambridge, MA, USA, 1991.

27 Wolfgang Thomas. Facets of synthesis: Revisiting Church's problem. In *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures, FOSSACS 2009*, volume 5504 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2009. doi:10.1007/978-3-642-00596-1_1.

28 Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1):135–183, 1998. doi:10.1016/S0304-3975(98)00009-7.