Passive Learning of Regular Data Languages in Polynomial Time and Data

Mrudula Balachander
Université libre de Bruxelles, Belgium

Emmanuel Filiot
Université libre de Bruxelles, Belgium

Raffaella Gentilini
Universitá degli Studi di Perugia, Italy

Abstract

A regular data language is a language over an infinite alphabet recognized by a deterministic register automaton (DRA), as defined by Benedikt, Ley and Puppis. The later model, which is expressively equivalent to the deterministic finite-memory automata introduced earlier by Francez and Kaminsky, enjoys unique minimal automata (up to isomorphism), based on a Myhill-Nerode theorem.

In this paper, we introduce a polynomial time passive learning algorithm for regular data languages from positive and negative samples. Following Gold's model for learning languages, we prove that our algorithm can identify in the limit any regular data language L, i.e. it returns a minimal DRA recognizing L if a characteristic sample set for L is provided as input. We prove that there exist characteristic sample sets of polynomial size with respect to the size of the minimal DRA recognizing L. To the best of our knowledge, it is the first passive learning algorithm for data languages, and the first learning algorithm which is fully polynomial, both with respect to time complexity and size of the characteristic sample set.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Automata over infinite objects

Keywords and phrases Register automata, passive learning, automata over infinite alphabets

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.10

Funding *Mrudula Balachander*: PhD funded by F.R.S.-FNRS.
Emmanuel Filiot: Senior research associate of the F.R.S.-FNRS. His contribution to this work was partly funded by the FNRS MIS project F451019F and the FNRS PDR project T.0117.24.

Acknowledgements We thank Marie Tcheng for spotting some issue in the characterization of completability in a preliminary version of this paper.

1 Introduction

Finite-memory automata (FMA) have been introduced by Francez and Kaminsky in the 90s [27], as an extension of finite automata to infinite alphabets of data values, which can be stored and compared using a finite set of registers. Since the introduction of FMA, a rich literature on automata for languages over infinite alphabets has emerged, e.g. see [12, 6, 35, 13, 11, 31, 24]. Automata for data languages have many applications in computer science, for instance in verification of concurrent systems [13, 14, 2, 1], of programs with dynamic allocation [24] as well as in reactive system synthesis [28, 20, 19]. Unlike in the finite alphabet setting, non-determinism and determinism often yield expressively inequivalent models, such as for FMA. In this paper, we consider languages defined by deterministic FMA (DFMA), which we call regular data languages. Regular data languages form a robust class of languages. They are for instance captured by many variations on the

data storage mechanism of DFMA: allowing registers to be emptied during computation, or requiring that they must hold distinct values, or allowing their contents to be reassigned [33], or finally requiring that they must follow a *last appearance record (LAR)* policy [6].

Register automata with last appearance record. The latter model of [6], which we simply call *deterministic register automata* (DRA), is the chosen model in this paper for regular data languages. As shown in [6], for any regular data language L, there exists a unique DRA for L which is both minimal with respect to the number of states and number of registers. This is particularly relevant for learning regular data languages. In this model, data values are stored in a list whose length can vary during execution but is of bounded length k. Moreover, data occurring in the list are distinct. When a new data d is read, it is compared to the list α of stored values. The data d, if ever stored, has to be stored at the end of α, resulting in a new list αd. Values in α can be erased, and must be if d already occurs in α (to keep distinct values in memory), or if the list exceeds length k. E.g., consider the data language $L_{\neq}$ of words of the form $(d_1 d_2)^n$ for all $n > 0$ and all integers $d_1 \neq d_2$. A DRA recognizing $L_{\neq}$ would start with an empty memory, then store the 1st data d_1, then the 2nd data d_2 (after checking $d_1 \neq d_2$). At this point its memory is the list $d_1 d_2$. Then it would alternate between (1) checking whether the next data is d_1 and update the memory to $d_2 d_1$, and (2) checking whether the next data is d_2 and update the memory to $d_1 d_2$.

Passive learning of languages. Language inference has a long history, with applications in (but not limited to) verification [29]. The (passive) inference of regular languages (over a finite alphabet) from positive and negative samples has been studied since the 60s, see e.g. [30] for a survey. Given a sample set $S = (I^+, I^-)$ consisting of a finite set I^+ (resp. I^-) of positive (resp. negative) samples, the goal is to infer a regular language as a DFA consistent with S, i.e. it accepts all words in I^+ and rejects those in I^-. Gold proposed *identification in the limit* of a regular language as a formal notion to characterize the learning capability of inference algorithms [23, 16]. The algorithm RPNI (standing for Regular Positive and Negative Inference) [34] is one of the reference algorithms in passive learning: given a sample set S, it returns in polynomial-time a DFA consistent with S, and for any regular language L, if S is characteristic enough for L, then RPNI returns $\mathcal{A}_L$. Moreover, there is always a characteristic sample of polysize in the size of $\mathcal{A}_L$. In other words, RPNI identifies regular languages in polynomial time and data. Since then, there have been many variants of RPNI, for example for regular tree languages [22], regular queries in trees [15] or ω-regular languages [8, 9]. Recently, RPNI was used in combination with reactive synthesis methods to automatically generate reactive systems from LTL specifications and examples [5]. The extension of [5] to the synthesis of data-processing reactive systems is our main motivation for the present work, as the former heavily relies on passive learning.

Despite a rich literature on passive learning for regular languages over *finite alphabets*, and on automata and logics for words over *infinite alphabets*, only few is yet known about passive learning for regular languages over infinite alphabets, to the best of our knowledge. Passive learning for symbolic automata has been investigated in [21]. This is an orthogonal model to register automata: symbolic automata have complex tests over data, but no storage mechanism, while register automata have simple data tests, but registers to store data. Handling data that need to be memorized in a passive learning context is one of the main difficulties of our paper. Let us also mention that they are many contributions in *active* learning for regular data languages, in particular extensions of L* algorithm [7, 25, 26, 32]. All these extensions however run in exponential time.

Contributions. We prove that any regular data language L can be identified in polynomial time from polynomial data, in the size of the minimal DRA for L. To do so, we design an RPNI-like algorithm, which takes as input a finite sample set $S = (I^+, I^-)$ of positive and negative data words over an infinite domain of data values, and prove that it returns, in polynomial time, a DRA consistent with S, assuming data equality can be tested in polynomial time. We then show that for any regular language L, there exists a characteristic sample S_L, such that for any sample S containing S_L, our algorithm returns the minimal DRA recognizing L (as defined in [31]). To the best of our knowledge, it is the first fully polynomial passive learning algorithm for regular data languages.

Our RPNI algorithm is based on an alternative presentation of RPNI for DFA, compared to the original formulation of [34]. This alternative presentation is similar to the one of [8] for ω-regular languages. We believe this presentation is simpler than the original one, as it does not involve operations such as deterministic state-merging, and is easier to analyse. The idea is to start from a single-state automaton, and to incrementally extend it by adding new transitions, allowing more sample prefixes to be readable by the automaton (taken in length lexicographic order). To generalize the samples, when adding a new transition, existing states are prioritized as a target of the new transition, over creating a fresh new state. A transition can be added if the resulting automaton can still be completed into an automaton consistent with S. We call this S-completability and show it can be checked in PTime. In Section 3, we first present this alternative RPNI algorithm for DFA. It is of independent interest, but also helpful to understand its extension to DRA, which is done in Section 4. Briefly, when adding a transition, the algorithm now has to decide whether the incoming data must be stored and which of the stored data can be erased from memory. To do so, and to generalize the sample as much as possible, it tries both to reuse existing states for the target of the new transition, and to erase as many registers as possible, while preserving S-completability.

In Section 5, we prove that this algorithm identifies any regular data language L from a characteristic sample set of size polynomial in the size of the minimal DRA for L. To prove that the characteristic sample is polynomial, classical automata-theoretic arguments (such as intersection closure) do not apply because they involve exponential blow-up. Instead, we adapt group-theoretic techniques for checking bisimulation of register automata as defined in [33], which exploits symmetries underlying sets of configurations of register automata. As a byproduct of our techniques, we obtain that equivalence of DRA can be checked in CoNP.

2 Preliminaries

Data Words and Languages. In this paper, a *data domain* is an infinite countable set $\mathcal{D}$ equipped with equality, whose elements are called *data*. In this paper, we assume an arbitrary well-ordering $<_{\mathcal{D}}$ over $\mathcal{D}$, so that every subset has a minimal element. It is also assumed that any data has an effective representation, and that data comparison $<_{\mathcal{D}}$ can be tested in polynomial time.

A *(data) word* w (or sometimes just word) is a finite sequence of data from $\mathcal{D}$. Given a word w having length $|w| = n$, we denote by $w[i]$ the ith data of w for $1 \leq i \leq |w|$. We define the length-lexicographic order over $\mathcal{D}^*$: $u <_{llex} v$ if $|u| < |v|$ or $|u| = |v|$, $u = wdu'$, $v = wd'v'$ for some $d <_{\mathcal{D}} d'$ and w, u', v'. For a set $X \subseteq \mathcal{D}^*$, we let $\mathsf{Prefs}(X)$ be the set of words u such that u is a prefix (not necessarily strict) of some word in X.

Any (total) function $\mu : \mathcal{D} \to \mathcal{D}$ can be morphically extended to $\mu : \mathcal{D}^* \to \mathcal{D}^*$, where $\mu(w)[i] = \mu(w[i])$ for all $w \in \mathcal{D}^*$ and $1 \leq i \leq n$. The function μ is called a *data permutation* if μ is bijective. Two data words $w_1, w_2 \in \mathcal{D}^*$ are said to be data-equivalent[1], written as

[1] In the terminology of orbit-finite sets, w_1 and w_2 are said to be in the same orbit [10].

$w_1 \simeq w_2$, if $w_1 = \mu(w_2)$ for some data permutation μ. Note that it implies that $|w_1| = |w_2|$. We denote by $[w]_\simeq$ the $\simeq$-class of any data word w. The following result is immediate (under the previous assumption that data equality is in PTime):

▶ **Proposition 1.** *For any data domain $\mathcal{D}$, $\simeq$ is decidable in PTime.*

A *(data) language* L over $\mathcal{D}$ is a set of data words over $\mathcal{D}$. It is *equivariant* if for all $w \in L$, $[w]_\simeq \subseteq L$. For $w_1, w_2 \in \mathcal{D}^*$, we write $w_1 =_L w_2$ when $w_1 \in L \leftrightarrow w_2 \in L$ holds.

Register Automata. Register automata have first been introduced by Kaminsky and Francez under the name *finite memory automata* [27]. In this paper, we use an equi-expressive model defined in [6], which enjoys a canonical, state-minimal and register-minimal form. In this model, the number of available registers may vary over time, but depends on the state, i.e. any configuration in state p has the same number $\lambda(p)$ of stored data. Moreover, registers follow a fixed policy in the way they are used, called *last appearance record*. Thanks to this policy, it is not necessary to give names to registers, and the set of stored data in a configuration is just a data word (of bounded length). Before giving the formal definition, let us informally explain how this model works. Any transition is of the form $t = (p, \alpha, E, q)$ where p, q are states, α is a data word of length $\lambda(p) + 1$, E is a subset of $\{1, \dots, \lambda(p) + 1\}$. The word α is seen as a test of the new data read as input against the registers, via $\simeq$-equivalence. There can be two types of α: *disequality tests*, which require all the data of α to be pairwise different, and *equality tests*, which require all data to be pairwise different but the last one, which repeats exactly once in α. E.g. over $\mathcal{D} = \mathbb{N}$, $\alpha = 1323$ is an equality test, $\alpha = 2341$ is a disequality test and $\alpha = 222$ is not a valid test. Any configuration in state p is of the form $(p, d_1 \dots d_{\lambda(p)})$ where $d_1 \dots d_{\lambda(p)} \in \mathcal{D}^*$ is the memory content. Reading a new incoming data d, transition t above can be triggered only if $d_1 \dots d_{\lambda(p)} d \simeq \alpha$, and in that case the automaton moves to state q. Some of the data in memory can be dropped, and E specifies which one, by their positions in $\{1, \dots, \lambda(p) + 1\}$. The automaton maintains the following invariant: no data is stored multiple times in memory, and the size of the memory only depends on the state. To maintain the first invariant, if α is an equality test $\alpha_1 \dots \alpha_{\lambda(p)+1}$ with $\alpha_j = \alpha_{\lambda(p)+1}$ for some $j \leq \lambda(p)$, then $j \in E$: this implies that only the *last* occurence of d_j is kept. For the second invariant, we must have $\lambda(p) + 1 - |E| = \lambda(q)$.

▶ **Definition 2.** *A (deterministic)* register automaton *(DRA) over $\mathcal{D}$ is a tuple $\mathcal{A} = \langle Q, k, \lambda, T, q_0, F \rangle$, where:*

- *Q is the set of states, q_0 is the initial state with $\lambda(q_0) = 0$ and $F \subseteq Q$ are the final states.*
- *$k \in \mathbb{N}$ is the maximum number of stored values, and $\lambda : Q \to \{0, \dots, k\}$ is called an availability function;*
- *T is a finite set of transitions of the form (p, α, E, q), where $p, q \in Q$, $\alpha \in \mathcal{D}^{\lambda(p)+1}$ is either an equality or a disequality test and $E \subseteq \{1, \dots, \lambda(p) + 1\}$. It is required that: (i) $\lambda(p) + 1 - |E| = \lambda(q)$, (ii) if $\alpha[i] = \alpha[\lambda(p) + 1]$ for some $i \leq \lambda(p)$, then $i \in E$, and (iii) T is deterministic, i.e. for all states p and $\simeq$-equivalence class c, there exists at most one transition (p, α, E, q) such that $\alpha \in c$.*

For all $0 \leq i \leq k$, we let $Q_i = \lambda^{-1}(i)$ be the set of states with i available registers. Given this notation, we may freely denote a DRA as a tuple $\langle Q_0, \dots, Q_k, T, I, F \rangle$. A *configuration* is a pair $\tau = (q, \sigma)$, where $q \in Q$ and $\sigma \in \mathcal{D}^{\lambda(q)}$. Given two configurations (p, σ) and (q, σ') and a data $a \in \mathcal{D}$, we write $(p, \sigma) \xrightarrow{a}_{\mathcal{A}} (q, \sigma')$ whenever there exists a transition (p, α, E, q) such that $\sigma.d \simeq \alpha$, and $\sigma' = \mathsf{drop}_E(\sigma.d)$ where $\mathsf{drop}_E(.)$ replaces, for all $i \in E$, the ith data of $\sigma.d$ by ϵ, and for all $i \in \{1, \dots, k\} \setminus E$, keeps the ith data.

A run of $\mathcal{A}$ over a data word w is a sequence of configurations $(p_1, \sigma_1) \dots (p_n, \sigma_n)$ such that $n = |w| + 1$, and for all $1 \leq i < n$, $(p_i, \sigma_i) \xrightarrow{w[i]}_{\mathcal{A}} (p_{i+1}, \sigma_{i+1})$. We write $(p, \sigma_1) \xrightarrow{w}_{\mathcal{A}} (p_n, \sigma_n)$ to denote the existence of such a run. The language recognized by the register automaton $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of words w such that $(q_0, \epsilon) \xrightarrow{w}_{\mathcal{A}} (p, \sigma)$ for some $p \in F$.

▶ **Example 3.** Consider the language of data words $L = \{d_1 d'_1 d_2 d'_2 \dots \ d_n d'_n \in \mathbb{N}^* \mid \forall 1 \leq i \leq n, d_i = d'_i\}$. The data language L is recognizable by the DRA $\mathcal{A} = \langle Q, k, \lambda, T, q_0, F\rangle$, where $k = 2$, $Q = \{q_0, q_1\}$, $\lambda(q_0) = 0$, $\lambda(q_1) = 1$, $F = \{q_0\}$ and $T = \{(q_0, 1, \emptyset, q_1), (q_1, 11, \{1, 2\}, q_0)\}$.

The language $L_{\neq} = \{(d_1 d_2)^n \mid n, d_1, d_2 \in \mathbb{N}, n > 0, d_1 \neq d_2\}$ of Introduction is recognizable by $\mathcal{A}' = \langle \{p_0, \dots, p_3\}, 2, \lambda', T', p_0, \{p_2\}\rangle$ where $\lambda'(p_0) = 0$, $\lambda'(p_1) = 1$, $\lambda'(p_2) = \lambda'(p_3) = 2$, and $T' = \{(p_0, 1, \varnothing, p_1), (p_1, 12, \varnothing, p_2), (p_2, 121, \{1\}, p_3), (p_3, 121, \{1\}, p_2)\}$.

We say that a data language L is *regular* if it is recognized by a DRA $\mathcal{A}$, i.e. $\mathcal{L}(\mathcal{A}) = L$. Note that regular data languages are equivariant, i.e. stable under data renaming, because the property of ρ to be a run on some data word w only depends on the data equalities in w.

3 Warm Up: Passive Learning of DFA

In this section, we recall how passive learning of DFA is achieved by the RPNI algorithm [34]. However we provide an alternative presentation, inspired by a similar presentation in the context of ω-regular languages [8]. RPNI first constructs a tree-like DFA accepting exactly the set of positive samples, and to generalize them, it merges the states of the initial DFA as much as possible and in a specific order, while preserving the rejection of negative samples. In contrast, our approach starts from a single-state DFA and adds transitions incrementally. When adding a new transition, to generalize the positive samples, it tries to reuse an existing state while rejecting negative samples, otherwise it creates a new state. This approach, which we believe is slightly simpler than the original RPNI algorithm, and offers the same guarantees, is easier to generalize to register automata, as it is not based on state-merging. It is indeed not clear how to define such an operation on register automata.

Some useful notions on DFA. Before defining the algorithm, we introduce some notions for DFA. We denote a DFA over an alphabet Σ as a tuple $\mathcal{A} = \langle Q, q_0, F, \delta\rangle$, where Q is the set of states with initial state q_0, $F \subseteq Q$ are the final states and $\delta : Q \times \Sigma \to Q$ is the transition function (which might be partial). The language accepted by $\mathcal{A}$ is denoted by $L(\mathcal{A})$. We also denote (if it exists) by $\delta^*(p, w)$ the state reached by $\mathcal{A}$ when reading a word w from a state p. In particular, $\delta^*(p, \epsilon) = p$. Given a set $S \subseteq \Sigma^*$, a word $w \in \Sigma^*$, we define its *residual language* $w^{-1}S = \{w' \in \Sigma^* \mid w.w' \in S\}$.

We now define a relation on DFA, which characterizes when a DFA is a subgraph of another one. Given two DFAs $\mathcal{A}_i = \langle Q_i, q_0^i, F_i, \delta_i\rangle$, $i = 1, 2$, we say that $\mathcal{A}_2$ *completes* $\mathcal{A}_1$, written $\mathcal{A}_1 \preceq \mathcal{A}_2$, if there exists an injective morphism $\Phi : Q_1 \to Q_2$, i.e. a mapping which preserves the initial state, final states and the transitions: $\Phi(q_0^1) = q_0^2$, $\Phi(F_1) \subseteq F_2$ and for all transitions $(p, \sigma, q) \in \delta_1$, we have $(\Phi(p), \sigma, \Phi(q)) \in \delta_2$. Given a sample set $S = (I^+, I^-)$ where $I^+, I^- \subseteq \Sigma^*$, we say that a DFA $\mathcal{A}$ is *S-consistent* if $I^+ \subseteq L(\mathcal{A})$ and $I^- \cap L(\mathcal{A}) = \varnothing$. We say that $\mathcal{A}$ is *S-completable* if there exists $\mathcal{A}'$ such that $\mathcal{A} \preceq \mathcal{A}'$ and $\mathcal{A}'$ is S-consistent.

▶ **Proposition 4.** *A DFA $\mathcal{A}$ is S-completable iff $I^- \cap L(\mathcal{A}) = \varnothing$ and there do* not *exist $u_1, u_2, z \in \Sigma^*$ such that $u_1 z \in I^+$, $u_2 z \in I^-$ and $\delta^*(q_0, u_1) = \delta^*(q_0, u_2)$.*

As a corollary, DFA completability can be checked in PTime, where we define the size $||\mathcal{A}||$ of a DFA $\mathcal{A}$ as its number of states plus number of transitions.

▶ **Corollary 5.** *Given a DFA $\mathcal{A}$ and a (finite) sample set S, it can be checked in time $O(||\mathcal{A}||.||S||)$ whether $\mathcal{A}$ is S-completable.*

Algorithm description. The algorithm is given as Algorithm 1. It takes as input a finite sample set $S = (I^+, I^-)$ of consistent positive and negative samples ($I^+ \cap I^- = \varnothing$), and returns an S-consistent DFA $\mathcal{A}$. It starts with a single state DFA $\mathcal{A}$ and keeps on extending it with new transitions (possibly creating new states), to be able to read more and more prefixes of words from S. To do so, it adds all the prefixes of words from $I^+ \cup I^-$ (but ϵ) to a waiting list *ToRead*, processed in length-lexicographic order $<_{llex}$. The invariants of the algorithm (preserved by the while-loop at line 3) are:

- **(i)** any word w in *ToRead* cannot be read fully by $\mathcal{A}$, i.e. $\delta^*(q_0, w)$ is undefined.
- **(ii)** $\mathcal{A}$ is S-completable.
- **(iii)** $\mathcal{A}$ is $(I^+ \setminus \mathit{ToRead}, I^-)$-consistent.

It is easily seen that those invariants are initially true, because the sample set is consistent. At any iteration the algorithm picks a $<_{llex}$-minimal word $w = ua$ in *ToRead*, with $a \in \Sigma$. Since w is length-minimal, $\delta^*(q_0, u) = p$ exists. The algorithm then adds a new transition from p on reading a. To do so, it calls a function SET_TRANSITION which first tries to reuse an existing state q such that adding transition $p \xrightarrow{a} q$ preserves S-completability (reusing existing states is how the algorithm *generalizes* the samples). If no such state exists, it creates the fresh state pa and adds the transition $p \xrightarrow{a} pa$. After the transition is added, all words in *ToRead* which can now be read are removed from *ToRead* (thus preserving invariant (i)), and if additionally they are positive, the state they reached are set to be accepting. Invariant (ii) is preserved because SET_TRANSITION makes sure the new transition can be added w/o breaking S-completability. If invariant (iii) is not preserved, since line 11 makes sure all words in $I^+ \setminus \mathit{ToRead}$ are accepted (for the new set *ToRead*), there are some $w \in I^+$ and $w' \in I^-$ which both reach the same accepting state, contradicting S-completability and (ii).

The algorithm terminates in a polynomial number of steps with respect to the size of S, and returns an S-consistent DFA, according to invariant (iii). Note that it is not specified in which order states are enumerated at line 15. Different orders may yield different DFA, but this has no influence if the sample is rich enough, as explained in the next paragraph.

Completeness. A way to formalize how well RPNI generalizes the samples is given by a completeness result: for any regular language L, the minimal (not necessarily complete) DFA $\mathcal{A}_L$ recognizing L is output by RPNI, if a small (polynomial size in the size of $\mathcal{A}_L$) but characteristic enough sample set is provided as input. We obtain the same result for our modified RPNI algorithm. We do not formally prove it because it is a particular case of the same result on DRA, fully proven in Section 5, and the purpose of this section is to convey intuitions easing the comprehension of the DRA setting. We now define a characteristic sample set S_L of polynomial size in the size of $\mathcal{A}_L$, such that Algo. 1, on input S_L, returns a DFA isomorphic to $\mathcal{A}_L$. We give the intuitions on how S_L influences the execution of Algorithm 1. For the algorithm to create as many states and transitions as $\mathcal{A}_L$, *ToRead* needs to contain at least one word per state and transition. Then, when it tries to add a new transition, negative samples are needed to control the target state: by an adequate choice of negative samples, exactly one target state (either existing or fresh) will be picked.

It is convenient to view $\mathcal{A}_L$ as the quotient DFA obtained from the Myhill-Nerode congruence $\equiv_L$, defined by $u \equiv_L v$ if for all $w \in \Sigma^*$, $uw \in L \leftrightarrow vw \in L$ holds. Let $[u]$ be the class of any word u. For any two different classes c, c', there exists $w_{c,c'} \in \Sigma^*$ which distinguishes c and c', in the sense that $u_c w \in L \not\leftrightarrow u_{c'} w \in L$. Then, for u, v and $a \in \Sigma$

Algorithm 1 Alternative formulation of RPNI algorithm for DFA.

Input: A (finite) sample set $S = (I^+, I^-)$ of positive and negative samples over an alphabet Σ: $I^+, I^- \subseteq \Sigma^*$ such that $I^+ \cap I^- = \varnothing$.
Output: An S-consistent DFA $\mathcal{A} = \langle Q, q_0, F, \delta \rangle$

$q_0 \leftarrow \epsilon$; $Q \leftarrow \{\epsilon\}$; $\delta \leftarrow \emptyset$; **if** $\epsilon \in I^+$ **then** $F \leftarrow \{q_0\}$ **else** $F \leftarrow \emptyset$
$ToRead \leftarrow \mathsf{Prefs}(I^+ \cup I^-) \setminus \{\epsilon\}$
while $ToRead \neq \emptyset$ **do**
  $u \cdot a \leftarrow$ length-lexicographic minimal word in $ToRead$
  $p \leftarrow \delta^*(q_0, u)$ `// guaranteed to exist`
  $(p, a, q) \leftarrow \mathsf{SET\_TRANSITION}(\mathcal{A}, p, a, S)$
  $\delta \leftarrow \delta \cup \{(p, a, q)\}$; $Q \leftarrow Q \cup \{q\}$
  foreach $w \in ToRead$ **do**
    if $\delta^*(q_0, w)$ *exists* **then**
      $ToRead \leftarrow ToRead \setminus \{w\}$
      if $w \in I^+$ **then** $F \leftarrow F \cup \{\delta^*(q_0, w)\}$;
return $\mathcal{A} = \langle Q, q_0, F, \delta \rangle$

Function $\mathsf{SET\_TRANSITION}(\mathcal{A}, p, a, S)$:
  Input: A DFA $\mathcal{A} = \langle Q, q_0, F, \delta \rangle$, state $p \in Q$ and character $a \in \Sigma$ such that $\delta(p, a)$ is undefined and $\mathcal{A}$ is S-completable
  Output: A state q (either in Q if it exists, otherwise fresh) such that $\langle Q, q_0, F, \delta \cup \{p \xrightarrow{a} q\} \rangle$ is S-completable
  foreach $q \in Q$ **do**
    if $\mathsf{COMPLETABLE}(\langle Q, q_0, F, \delta \cup \{p \xrightarrow{a} q\} \rangle, S)$ **then return** (p, a, q)
  `// check` S`-completability, see Cor.5`
  return (p, a, pa)

such that $[ua] \neq [v]$, samples are needed to forbid the learning algorithm to create the transition $[u] \xrightarrow{a} [v]$. We use the word $w_{[ua],[v]}$ to do so. Formally, a sample set $S = (I^+, I^-)$ is *characteristic for* L if it is L-consistent and there exist $St, Tr, D \subseteq \Sigma^*$ such that:

- St contains for each class $c \in \Sigma^*/_{\equiv_L}$, a $<_{llex}$-minimal representative u, i.e. $[u] = c$,
- for all $u \in St$ and $a \in \Sigma$, $ua \in Tr$,
- for all $u, v \in St$, $a \in \Sigma$, if $[ua] \neq [v]$ then $uaw_{[ua],[v]}, vw_{[ua],[v]} \in D$,

and such that $(St \cup Tr \cup D) \cap L \subseteq I^+$ and $(St \cup Tr \cup D) \cap \overline{L} \subseteq I^-$.

Given a characteristic sample set S_L, the invariant maintained by Algorithm 1 on the while-loop is that $\mathcal{A}_L$ completes the DFA $\mathcal{A}$ constructed so far. This is ensured by the fact that $\mathcal{A}$ is constructed incrementally by adding new transitions, and by the samples in D which prevents some wrong transitions to be created. Since $ToRead$ initially contains at least one representative sample per states and transitions of $\mathcal{A}_L$, the final automaton $\mathcal{A}$ returned by Algorithm 1 has the same number of states and transitions as $\mathcal{A}_L$, and by the invariant, we get that it is isomorphic to $\mathcal{A}_L$. Moreover, there exists a characteristic sample set of polynomial size in the size of $\mathcal{A}_L$: there is a representative u_c of any class c of linear length in the number of states of $\mathcal{A}_L$, while the distinguishing words $w_{c,c'}$ are bounded, using classical pumping arguments, quadratically in the number of states of $\mathcal{A}_L$. This can even be improved to a linear upper bound [18].

4 Learning Register Automata from Positive and Negative Samples

Let $\mathcal{D}$ be a data domain. A (finite) *sample set* is a pair $S = (I^+, I^-)$ such that $I^+, I^- \subseteq \mathcal{D}^*$ are finite sets of respectively positive and negative samples. S is *consistent* whenever for all $u \in I^+, v \in I^-$, we have $u \not\simeq v$. In this section we introduce an RPNI-like passive learning algorithm for DRA from consistent sample sets. We start by giving some intuition on the key ingredients underlying the algorithmic process. It follows the same structure as the DFA learning algorithm presented before.

The algorithm initially constructs a DRA $\mathcal{A}$ being able just to read the empty word ϵ (and accept it if $\epsilon \in I^+$). Moreover, the non-empty samples prefixes are collected into a set, called $ToRead$ (just as for the DFA case of Section 3): namely, $ToRead$ maintains the sample prefixes not yet readable by the DRA $\mathcal{A}$ under construction. The algorithm keeps on adding transitions to be able to read such samples prefixes, until $ToRead$ is empty (iterating over $ToRead$ by increasing samples in $<_{llex}$ order). The following crucial invariants are maintained during the overall execution:

- **(i)** $\mathcal{A}$ is a DRA able to read each word in $\mathsf{Prefs}(I^+ \cup I^-) \setminus ToRead$
- **(ii)** $\mathcal{A}$ accepts (resp. rejects) each sample in $I^+ \setminus ToRead$ (resp. I^-)
- **(iii)** It is possible to "complete" $\mathcal{A}$ (adding new states and transitions) into a DRA accepting any sample of I^+ and rejecting all samples of I^- (the latter DRA is said to be S-consistent).

Those invariants are clearly met by the initial single-state DRA. We now formalize the notion of being S-completable for DRA. As for the DFA case, one needs a notion of embedding of DRA into another. For $i = 1, 2$, let $\mathcal{A}_i = \langle Q_i, k_i, \lambda_i, T_i, q_0^i, F_i \rangle$ be a DRA. We say that $\mathcal{A}_2$ completes $\mathcal{A}_1$, written $\mathcal{A}_1 \preceq \mathcal{A}_2$, whenever there exists an injective mapping $\Phi : Q_1 \to Q_2$ which preserves the initial state, the availability function, the final states, and the transitions, as follows: $\lambda_1 = \lambda_2 \circ \Phi$, $\Phi(F_1) \subseteq F_2$, $\Phi(q_0^1) = q_0^2$, and for all transitions $(p, \alpha, E, q) \in T_1$, there exists $\beta \simeq \alpha$ such that $(\Phi(p), \beta, E, \Phi(q)) \in T_2$. In particular, observe that $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$. We say that $\mathcal{A}_1$ is S-completable whenever there exists an S-consistent DRA $\mathcal{A}_2$ completing $\mathcal{A}_1$. This notion is decidable in PTime. To prove it, we first give a characterization of S-completability, which can be proved in the same line as the DFA case:

▶ **Proposition 6.** *A DRA $\mathcal{A}$ is S-completable for a sample set $S = (I^+, I^-)$, iff $L(\mathcal{A}) \cap I^- = \varnothing$ and there do not exist words $w \in I^+$, $z \in I^-$, state q and words σ, σ' such that: $w = w_1 w_2 \wedge z = z_1 z_2 \wedge (q_0, \epsilon) \xrightarrow{w_1}_{\mathcal{A}} (q, \sigma) \wedge (q_0, \epsilon) \xrightarrow{z_1}_{\mathcal{A}} (q, \sigma') \wedge \sigma w_2 \simeq \sigma' z_2$.*

By inspecting I^+, I^- and computing the states reached by their prefixes in $\mathcal{A}$, it is not difficult to decide the latter characterization in PTime:

▶ **Corollary 7.** *It is decidable in PTime whether for a given DRA $\mathcal{A}$ is S-completable.*

The prefixes in $ToRead$ are considered in $<_{llex}$-increasing order and used to add a transition to $\mathcal{A}$ allowing their processing without violating the three invariants above. This can be done since $\mathcal{A}$ is S-completable. Therefore, by the end of the algorithm $\mathcal{A}$ is an S-consistent DRA (due to the first two invariants and the fact that *toRead* is empty).

With this intuition in mind, we go in more details introducing the pseudocode of our DRA passive learning algorithm. A careful reader may notice a clear symmetry between the DFA and DRA learning processes presented, and may wonder how we take care of the registers when creating new transitions. This will be made clear in the pseudocode presentation below.

Pseudocode of the DRA Passive Learning Algorithm. The pseudocode in Algorithm 2 illustrates the procedure DRA_PASSIVE_LEARN(S), which takes as input a consistent finite sample set $S = (I^+, I^-)$ and returns an S-consistent DRA $\mathcal{A}$. A full execution of the algorithm is provided as Example 8.

A while-loop follows the initialization phase in lines (1)-(2), controlled by a guard checking whether the set of samples prefixes $ToRead$ is not empty. In that case, the DRA under construction is not yet capable to process all the words in S. Hence, a $<_{llex}$-minimal word ua is extracted from $ToRead$ at line (4). Invariant (i) and the minimal length of ua ensures that $\mathcal{A}$ can read u but not ua. As $\mathcal{A}$ is deterministic, it admits a unique run $\rho : \langle q_0, \epsilon\rangle \xrightarrow{u}_{\mathcal{A}} \langle q, \sigma\rangle$ processing u. The S-completability of $\mathcal{A}$ ensures that it is possible to add a transition from state q such that ua is now readable, while preserving the invariants. To generalize the samples, the choice of that transition is done in such a way that it tries to keep the least amount of data in memory and tries to reuse existing states. Such a choice is delegated to the procedure SET_TRANSITION at line (6), which returns a new transition $t = (q, \sigma a, E, p)$. Before describing this procedure, we finish the overview of Algorithm 2. Line (7) augments $\mathcal{A}$ by adding to T the new transition t. If the target state q' is new, line (8) updates consistently the set of states of the DRA under construction and the availability function. Finally, the for-loop at line (9) updates $ToRead$ by removing the prefixes that can now be read, and updates the final states consistently based on positive samples which can now be read by $\mathcal{A}$.

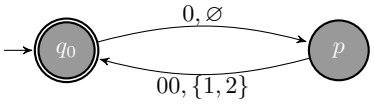
We now describe the procedure SET_TRANSITION whose pseudocode is given in Algorithm 3. It takes as input a consistent sample set $S = (I^+, I^-)$, an S-completable DRA $\mathcal{A}$, a state q, and a word $\sigma \cdot a \in \mathcal{D}^{\lambda(q)+1}$ such that $\mathcal{A}$ has no transition be able to read a from configuration (q, σ). It computes a set of registers E to be erased, and a new target state p, and returns transition $(q, \sigma a, E, p)$, with the guarantee that adding this new transition to $\mathcal{A}$ preserves S-completability, while trying to generalize the samples. To compute E, it first tries to determine if some stored values can be dropped, while preserving S-completability. Then it attempts to reuse an existing state p as the target of a transition $(q, \sigma a, E, p)$, otherwise creates a fresh new state. When computing E, it tries to erase as many stored values as possible, which informally allows for more generalization of the samples. Another fundamental reason for doing so is because it is needed to be able to prove that our algorithm identifies regular data languages in the limit. Since the characteristic sample somehow encodes the behaviours of the minimal automaton, which stores the least amount of data in its configurations, the algorithm as well has to follow the same policy, to be able to output the minimal canonical automaton for the language. We now give more details.

If a is registered in the word σ at position j (i.e. $\sigma(j) = a$) then the set of erasable registers E is initialized to $\{j\}$ otherwise to the empty set. Line (3) initializes the set R (of registers to check for erasability) to $\{1, \ldots, \lambda(q) + 1\} \setminus E$. The loop at line (4) (taken $|R|$ times) extracts one-by-one the registers in R and checks whether their values can be safely erased. It relies on a function Choose(R) which picks a register from R. In each iteration the current register h chosen from R is recognized as erasable only if the DRA obtained by adding the transition $t = (q, \sigma a, E \cup \{h\}, \mathsf{f})$ towards a fresh new state f does not compromise S-completability. In that case h is added to E (and never removed). The task of checking if adjoining the transition t to $\mathcal{A}$ leads to an S-completable DRA $\mathcal{A}'$ is performed through a call to a function COMPLETABLE($\mathcal{A}', I^+, I^-$), which exists by Corollary 7.

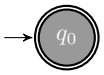
Once the set of erasable registers E for the new transition is defined, the function SET_TRANSITION proceeds with a a loop trough the states $p \in Q$ such that $\lambda(p) = \lambda(q) + 1) - |E|$ at line (8), aiming at identifying the target of the new transition among existing states. To check that p can be reused, the algorithm calls COMPLETABLE($\mathcal{A}', I^+, I^-$)

to determine whether the DRA $\mathcal{A}'$ obtained by adding the transition $(q, \sigma a, E, q')$ to $\mathcal{A}$ is S-completable. In that case the function returns the transition $(q, \sigma a, E, p)$. If no vertex is identified as a safe target for the new transition within the loop at line (8), a fresh state $\mathsf{f} \notin Q$ is created and the function returns the transition $(q, \sigma a, E, \mathsf{f})$ at line (11).

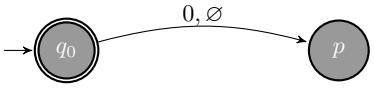
▶ **Example 8** (RPNI Execution for DRA)**.** Consider the language $L = \{\epsilon\} \cup \{d_1 d'_1 d_2 d'_2 \dots d_n d'_n \in \mathbb{N}^* \mid \forall 1 \leq i \leq n, d_i = d'_i\}$ recognizable by the the DRA $\mathcal{A} = \langle Q, k, \lambda, T, q_0, F \rangle$, where $k = 2$, $Q = \{q_0, p\}$, $\lambda(q_0) = 0$, $\lambda(p) = 1$, $F = \{q_0\}$ and $T = \{(q_0, 0, \emptyset, p), (p, 00, \{1, 2\}, q_0)\}$, as depicted below:



We provide a complete example of execution of the proposed DRA learner, given the sample-set $S = (I^+ = \{\epsilon, 00, 11\}, I^- = \{0, 011, 01\})$, which eventually returns the latter DRA. The RPNI algorithm builds an initial DRA $\mathcal{A}$ composed by the only initial state q_0, which is inserted in the final states since $\epsilon \in I^+$. Then, the set $\mathit{ToRead} = \{0, 1, 00, 01, 11, 011\}$ of non-empty sample-prefixes is constructed. The initial DRA is:



The first execution of the main loop extracts the $<_{llex}$-minimum sample $ua = 0$ from ToRead and calls SET_TRANSITION$(\mathcal{A}, q_0, \sigma a = 0, I^+, I^-)$ to build a transition t out from q_0 (reached by $\mathcal{A}$ upon reading $u = \epsilon$) processing the value $a = 0$. SET_TRANSITION$(\mathcal{A}, q_0, \sigma a = 0, I^+, I^-)$ detects that a needs to be remembered, given that the DRA $\mathcal{A}'$ obtained by adding to $\mathcal{A}$ the transition $(q_0, a, \emptyset, p)$ (toward the fresh state p) is not completable since $00 \in I^+, 01 \in I^-$. Therefore, the set of erasable registers for the transition $(q_0, a, E, \_)$ is set to the empty set (i.e. $E = \emptyset$). There is no state in $\mathcal{A}$ with $(\lambda(q_0) + 1) - |E|) = (0 + 1) - 0 = 1$ registers. Hence, the second loop at line 8 does not perform any iteration and SET_TRANSITION$(\mathcal{A}, , q_0, \sigma a = 0, I^+, I^-)$ returns the transition $t = (q_0, a, \emptyset, p)$ toward the fresh state p. The DRA under construction is then updated to $\mathcal{A} = \langle \{q_0, p\}, 1, \lambda, \{t\}, \{q_0\} \rangle$, where $\lambda(q_0) = 0, \lambda(p) = 1$. The final for-loop at line 9 extracts from ToRead the sample 0 and the sample 1 that are now readable by $\mathcal{A}$ and does not label p as final because there is no positive sample readable by $\mathcal{A}$ leading to p. Therefore, at the beginning of the second iteration of the main loop at line 3, we have that $\mathcal{A} = \langle \{q_0, p\}, 1, \lambda, \{t\}, \{q_0\} \rangle$ and $\mathit{ToRead} = \{00, 01, 11, 011\}$.



The second execution of the main loop extracts the $<_{llex}$-minimum sample $ua = 00$ from ToRead and calls SET_TRANSITION$(\mathcal{A}, p, \sigma a = 00, I^+, I^-)$ to build a transition t out from p (reached by $\mathcal{A}$ upon reading $u = 0$) processing the value $a = 0$. Line 5 of SET_TRANSITION inserts the index 1 in E since $\sigma(1) = a = 0$ (avoiding the storage of duplicates). The automaton $\mathcal{A}'$ obtained adding to $\mathcal{A}$ the transition $(p, 00, E = \{1, 2\}, s)$, where s is a fresh state is completable. Therefore E is set to $\{1, 2\}$ at the end of the first (and only) iteration of the loop at line 4. Since $\mathcal{A}$ contains only one state with $(\lambda(p)+1)-|E| = (1+1)-2 = 0$ registers, also the loop at line 8 iterates only once, attempting

at redirecting $(p, 00, E = \{1, 2\}, s)$ toward q_0. Such a redirection succeeds since the DRA $\mathcal{A}'$ obtained by adding to $\mathcal{A}$ the transition $(p, 00, E = \{1, 2\}, q_0)$ is completable. Hence SET_TRANSITION$(\mathcal{A}, p, \sigma a = 00, I^+, I^-)$ returns $t' = (p, 00, \{1, 2\}, q_0)$. The DRA under construction is then updated to $\mathcal{A} = \langle \{q_0, p\}, 1, \lambda, \{t, t'\}, \{q_0\} \rangle$, where $\lambda(q_0) = 0, \lambda(p) = 1$. Since $\mathcal{A}$ is able to read all the samples provided as input, accepting them if they belong to I^+, the final for-loop at line 9 extracts all the samples from *ToRead* and does not modify the set of final states. The algorithm then terminates and returns a DRA recognizing L.

▶ Remark 9. In the algorithm SET_TRANSITIONS, there are several places where choices can be made: at line 5 when a register is chosen, and later at line 8 when target states p are enumerated. Different implementations of those choice and enumeration functions might result in differing DRA in general. However, our learning algorithm's guarantees are independent of those implementations, namely: 1) the algorithm returns an S-consistent DRA, and 2) for a regular language L, given a characteristic sample it returns a minimal DRA for L.

Algorithm 2 Algorithm DRA_PASSIVE_LEARN(S) for learning a deterministic register automaton from a set of positive and negative samples $S = (I^+, I^-)$.

Input: A finite consistent sample set $S = (I^+, I^-)$.
Output: An S-consistent DRA $\mathcal{A} = \langle Q, k, \lambda, T, q_0, F \rangle$, i.e. $I^+ \subseteq L(\mathcal{A})$ and $I^- \cap L(\mathcal{A}) = \varnothing$.

/* define an initial S-completable DRA $\mathcal{A}$ */
1 $Q \leftarrow \{\epsilon\}; \lambda(\epsilon) = 0; T \leftarrow \emptyset; q_0 \leftarrow \epsilon; F \leftarrow \emptyset$;**if** $\epsilon \in I^+$ **then** $F \leftarrow \{q_0\}$
2 *ToRead* $\leftarrow$ (prefs$(I^+ \cup I^-)) \setminus \{\epsilon\}$
3 **while** *ToRead is not empty* **do**
4 $\quad w = ua \leftarrow <_{llex}$-minimal word in *ToRead*
5 $\quad \langle q, \sigma \rangle \leftarrow$ configuration such that $\langle q_0, \epsilon \rangle \xrightarrow{u}_{\mathcal{A}} \langle q, \sigma \rangle$
6 $\quad (q, \sigma a, E, p) \leftarrow$ SET_TRANSITION$(\mathcal{A}, q, \sigma a, I^+, I^-)$
7 $\quad T \leftarrow T \cup \{(q, \sigma a, E, p)\}$
8 $\quad$ **if** $p \notin Q$ **then** $Q \leftarrow Q \cup \{p\}; \lambda(p) \leftarrow \lambda(q) + 1 - |E|$
9 $\quad$ **for** $w \in$ *ToRead* **do**
10 $\quad\quad$ **if** $\langle q_0, \epsilon \rangle \xrightarrow{w}_{\mathcal{A}} \langle s, \sigma \rangle$ **then**
11 $\quad\quad\quad$ *ToRead* $\leftarrow$ *ToRead* $\setminus \{w\}$
12 $\quad\quad\quad$ **if** $\exists w' \in I^+, w' \simeq w$ **then** $F \leftarrow F \cup \{s\}$
13 **return** $\mathcal{A} = \langle Q, k = \max_{q \in Q} \lambda(q), \lambda, T, q_0, F \rangle$

We now prove the correctness of our passive DRA learning algorithm. Lemma 10 below proves the validity of the crucial invariants briefly introduced in the previous section.

▶ **Lemma 10.** *The following invariants hold at line* (3) *(entrance of the main loop) of the algorithm* DRA_PASSIVE_LEARN(S)*, where* $S = (I^+, I^-)$*:*

- Inv1*:* $\mathcal{A}$ *is a DRA accepting (resp. rejecting) all words in* $I^+ \setminus$ *ToRead (resp. in* I^-*)*
- Inv2*:* $\mathcal{A}$ *can read all words in* Prefs$(I^+ \cup I^-) \setminus$ *ToRead*
- Inv3*:* $\mathcal{A}$ *is* S*-completable*

Sketch. The invariants are clearly true before the while-loop has been entered for the first time (for the 3rd invariant, it is because S is consistent, so we can always built an DRA which accepts exactly I^+ and its $\simeq$-equivalent words, and rejects all words from I^- and their $\simeq$-equivalent words).

Algorithm 3 Function SET_TRANSITION$(\mathcal{A}, q, \sigma a, I^+, I^-)$ used by DRA learner to complete the new transition $(q, \sigma \cdot a, \_, \_)$ defining the erasable registers and the target node.

```
1  Function SET_TRANSITION(A, q, σ·a, I⁺, I⁻):
       Input: S-completable DRA A = ⟨Q, k, λ, T, q₀, F⟩, state q ∈ Q, and word
              σ·a ∈ D^{λ(q)+1} such that T does not contain any transition (q, σ·a, _, _)
       Output: New t = (q, σa, E, p) s.t. ⟨Q, k, λ, T ∪ {t}, q₀, F⟩ is S-completable
     /* Compute some set E ⊆ {1, ..., λ(q)+1} of erasable registers          */
2  E ← ∅; i ← λ(q);
3  if ∃j ≤ i : σ(j) = a then E ← {j}; R ← {1...i+1} \ {j} else R ← {1...i+1};
4  while R ≠ ∅ do
5      h ← Choose(R); R ← R \ {h}; E ← E ∪ {h} ;
6      A' ← ⟨Q ∪ {f}, k, λ ∪ {f ↦ i+1−|E|}, T ∪ {(q, σa, E, f)}, q₀, F⟩ for f fresh ;
7      if ¬(COMPLETABLE(A', I⁺, I⁻)) then E ← E \ {h}  ;
     /* Pick some p ∈ Q, if it exists, as target of the new transition       */
8  foreach p ∈ Q such that λ(p) = λ(q) + 1 − |E| do
9      A' ← ⟨Q, k, λ, T ∪ {(q, σa, E, p)}, q₀, F⟩;
10     if COMPLETABLE(A', I⁺, I⁻) then return (q, σa, E, p);
     /* Otherwise create a fresh new state as target                         */
11 return (q, σa, E, f) for f a fresh state ;
```

The inductive step is rather simple. For Inv2, this is precisely the essence of our algorithm: it picks any word $ua \in ToRead$ which cannot be read fully (while u can) and completes $\mathcal{A}$ with one transition allowing to read ua. Inv3 is clear because calls to COMPLETABLE in SET_TRANSITION make sure that S-completability is preserved. For Inv1, the loop at line (9) updates the accepting states and so it makes sure that all words of I^+ which can be read are accepted. The only possible issue is that it could now accept words from I^-. It is not possible since SET_TRANSITION ensures that adding the new transition preserves S-completability (given an S-completable DRA, which is ensured by the induction hypothesis). If a word of I^- is now accepted, it means that some accepting state is now reachable by a positive word and a negative word, contradicting S-completability, ensured by Inv3. ◀

On the ground of Lemma 10, we are ready to prove the correctness and complexity of our DRA passive learning algorithm.

▶ **Theorem 11.** *The algorithm* DRA_PASSIVE_LEARN(S) *returns a DRA* $\mathcal{A}$ *consistent with* $S = (I^+, I^-)$ *(i.e.* $L(\mathcal{A}) \supseteq I^+ \wedge L(\mathcal{A}) \cap I^- = \emptyset$*) in time polynomial w.r.t. the size of* S*.*

Proof. Correctness is trivial by Inv1 and the fact that $ToRead$ is eventually empty when the algorithm terminates. For complexity, note that the algorithm takes as input a consistent sample set. Consistency can be checked in PTime by Lemma 1. Note that all the loops in DRA_PASSIVE_LEARN and SET_TRANSITION are iterated only a polynomial number of times in the size of their inputs. Moreover, COMPLETABLE runs in PTime by Cor. 7. ◀

▶ Remark 12. Note that correctness of SET_TRANSITION is trivial since it calls COMPLETABLE to make sure that the returned transition can be added while preserving S-completability. The fact that SET_TRANSITION tries to erase some registers and to reuse existing states is important for *generalizing* the samples, not for correctness, and important for proving that our learning algorithm identifies regular data languages in the limit. Moreover, $<_{llex}$-minimality at line 4 could be replaced by just length minimality while preserving correctness, but it is important to prove completeness.

5 Identification in the Limit of the Class of Regular Data Languages

In this section we prove that the proposed RPNI algorithm for DRA identifies in the limit the class of regular data languages, according to the definition of [23, 16, 8]. Namely, given a regular data language L, we show that there exists a characteristic sample S_L such that DRA_PASSIVE_LEARN(S) learns a DRA recognizing L whenever $S_L \subseteq S$. We then show that it can be chosen of polynomial size in the size of the minimal DRA recognizing L. As for the DFA case, we rely on the existence of a unique minimal DRA for L, as shown in [6]. We recall this notion here.

Memorable data and canonical register automata. Any regular data language L admits a unique minimal DRA, up to DRA isomorphism[2], as shown in [6]. Minimality is both for the number of states, as well as the number of stored data, in the sense that when the canonical automaton reads a prefix, it stores the least amount of data. The key notion towards canonicity and minimality is based on the notion of *memorable data* with respect to L. Given a word $w \in \mathcal{D}^*$ and two data $a, b \in \mathcal{D}$, we let $w[a/b] = \mu_{a/b}(w)$ where $\mu_{a/b}(a) = b$ and $\mu_{a/b}(d) = d$ for all $d \neq a$. Intuitively, a data a in a word w is memorable if for some continuation u, renaming data a in u by some data b which does not occur in wu has an influence on the membership into L.

▶ **Definition 13** (Memorable Values [6]). *A data $a \in \mathcal{D}$ in a word w is L-*memorable *for a language L if there exist $u \in \mathcal{D}^*$ and $b \in \mathcal{D}$ such that: $wu \simeq (wu)[a/b]$ and $wu \not\equiv_L w(u[a/b])$.*

Given a word w and a language L on $\mathcal{D}$, we let $mem_L(w)$ denote the finite sequence of all L-memorable data of w ordered according to the positions of their last occurrences in w: a occurs before b in $mem_L(w)$ if they are both memorable in w, and the last occurrence of a in w is before the last occurrence of b in w. Intuitively, $mem_L(w)$ represents the data that necessarily need to be stored by any DRA recognizing L. We now define an equivalence $\equiv_L \subseteq \mathcal{D}^* \times \mathcal{D}^*$ which is of finite index iff L is DRA-recognizable. We let $w \equiv_L w'$ if $|mem_L(w)| = |mem_L(w')|$ and for all words u, u', if $mem_L(w)u \simeq mem_L(w')u'$ then $wu =_L w'u'$. When $\equiv_L$ is of finite index, it is used to define a DRA recognizing L (the canonical DRA $\mathcal{A}_L$) that is minimal amongst all DRA recognizing L [6].

▶ **Theorem 14** ([6]). *For any DRA-recognizable language L, there exists a unique DRA $\mathcal{A}_L$ (up to isomorphism) recognizing L, which is both state-minimal and register-minimal (in the sense that the number of stored values in any configuration is minimal).*

We recall the construction of $\mathcal{A}_L = \langle Q, k, \lambda, T, q_0, F\rangle$. The set of states is $Q = \mathcal{D}/_{\equiv_L}$ with for all $u \in \mathcal{D}^*$, $\lambda([u]) = |mem_L(u)|$ and $k = max_{q\in Q}\lambda(q)$. The initial state is $q_0 = [\epsilon]$ and $F = \{[u] \mid u \in L\}$. Finally, for all classes $c \in \mathcal{D}/_{\equiv_L}$, pick a representative u such that $[u] = c$. For all $d \in \mathcal{D}$, pick a word[3] α such that $mem_L(u)d \simeq \alpha$, then set $([u], \alpha, E, [ud]) \in T$ where E is defined such that $mem_L(ud) = \mathsf{drop}_E(mem_L(u)d)$. The later automaton is well-defined [6]. Even though there are infinitely many minimal automata up to isomorphism, in the sequel we refer to $\mathcal{A}_L$ as a representative of one of them.

Characteristic samples. As for the DFA case, one needs samples to account for the states and transitions of the minimal automaton, and to distinguish equivalence classes. In addition, samples are also needed to witness memorable data. A sample set $S = (I^+, I^-)$ is said to be L-consistent if $I^+ \subseteq L$ and $I^- \cap L = \varnothing$.

[2] Two DRA $\mathcal{A}_1, \mathcal{A}_2$ are isomorphic if $\mathcal{A}_1 \preceq \mathcal{A}_2$ and $\mathcal{A}_2 \preceq \mathcal{A}_1$, see Section 4
[3] Different choices for the representative u and α yield isomorphic DRA.

▶ **Definition 15** (Characteristic sample). *Let L be a regular data language. A sample set $S = (I^+, I^-)$ is* characteristic for L *if it is L-consistent and there exist $St, Tr, Mem, D \subseteq \mathcal{D}^*$ such that:*

- *St contains, for each class $c \in \mathcal{D}/_{\equiv_L}$, a $<_{llex}$-minimal representative u, i.e. $[u] = c$,*
- *for all $w \in St, a \in mem_L(w), d \notin mem_L(w)$ s.t. d is $<_{\mathcal{D}}$-minimal: $wa, wd \in Tr$,*
- *for all $w \in Tr$, all $a \in mem_L(w)$, there exists $b \in \mathcal{D}$ and $u \in \mathcal{D}^*$ such that $wu \simeq (wu)[a/b]$, $wu \neq_L w(u[a/b])$ and $wu, w(u[a/b]) \in Mem$,*
- *for all $w \in St, z \in Tr$ such that $|mem_L(w)| = |mem_L(z)|$ and $w \not\equiv_L z$, there exist w', z' such that $mem_L(w)w' \simeq mem_L(z)z'$, $ww' \neq_L zz'$ and $ww', zz' \in D$,*

and $(St \cup Tr \cup Mem \cup D) \cap L \subseteq I^+$ and $(St \cup Tr \cup Mem \cup D) \cap \overline{L} \subseteq I^-$.

We now establish the following theorem:

▶ **Theorem 16.** *Let L be a regular data language and let S_L be a characteristic sample for L. Then* DRA_PASSIVE_LEARN(S_L) *returns a DRA isomorphic to the minimal DRA $\mathcal{A}_L$.*

Sketch. The main arguments are however rather intuitive and similar to the DFA case, except that additional samples are needed to account for memorable data. In the definition of characteristic sample, St and Tr allow the learner to identify the states and the transitions of $\mathcal{A}_L$. Instead, the learner uses the samples in Mem within the first loop of SET_TRANSITION to avoid dropping necessary registers. Finally, the learner uses the samples in D within the second loop of SET_TRANSITION to set correctly the target of new transitions avoiding wrong redirections toward old states.

The proof in Theorem 16 uses an inductive argument on the number of iterations of the main loop in RA_Passive_Learning to establish that, whenever the guard-condition at line (3) is checked, the DRA $\mathcal{A}$ under construction (given a characteristic sample for L) is isomorphic to a portion of the minimal DRA $\mathcal{A}_L$. In other words $\mathcal{A}_L$ completes $\mathcal{A}$, i.e. $\mathcal{A} \preceq \mathcal{A}_L$. When processing a new prefix $w = ua$ in $ToRead$, where w leads to a configuration (q, σ) of the so far constructed DRA, we know by IH that $\sigma = mem_L(u)$. The call to SET_TRANSITION adds a new transition $(q, \sigma a, E, p)$, and we have to prove that $(q, \sigma a, E, p)$ can be embedded into a transition $(q^{\circledast}, \alpha^{\circledast}, E^{\circledast}, p^{\circledast})$ of $\mathcal{A}_L$. By IH and the definition of $\mathcal{A}_L$, we know that u leads to configuration $(q^{\circledast}, \sigma = mem_L(u))$ by $\mathcal{A}_L$. Therefore, $\alpha^{\circledast} \simeq mem_L(u)a$. Again by definition of $\mathcal{A}_L$, $E^{\circledast}$ drops only unmemorable data of ua. By definition of characteristic samples, there are samples in Mem that prevent dropping memorable data of ua, so the first phase of SET_TRANSITIONS will keep only the necessary data, i.e. $E = E^{\circledast}$. Finally, the samples in D ensures that $(q, \sigma a, E)$ can be directed towards a unique state p. For any other state, calls to COMPLETABLE are failing. This state p is then embedded into state $p^{\circledast}$ of $\mathcal{A}_L$ to show that $\mathcal{A}$ extended with the new transition $(q, \sigma a, E, p)$ is completable by $\mathcal{A}_L$.

When DRA_PASSIVE_LEARN(S_L) returns, it yields a DRA $\mathcal{A}$ where $\mathcal{A} \preceq \mathcal{A}_L$. Moreover, since in S, there is at least one representative sample per state and transition of $\mathcal{A}_L$, $\mathcal{A}$ has as many states and transitions as $\mathcal{A}_L$, and therefore $\mathcal{A}_L \preceq \mathcal{A}$, i.e., $\mathcal{A}$ and $\mathcal{A}_L$ are isomorphic. ◀

Characteristic samples of polynomial size. We conclude this section by showing that given a regular data language L, there exists a characteristic sample S_L (see Definition 15) of polynomial size in the size of the minimal DRA $\mathcal{A}_L$ for L. Let us give a brief overview of the proof. For the samples representing the states and transitions of $\mathcal{A}(L)$ (sets St and Tr), it is rather easy, via pumping arguments, to show that samples of lengths polynomial in the number of states of $\mathcal{A}_L$ are sufficient. It relies on the next lemma, proved by similar arguments as in the case of finite memory automata [27].

▶ **Lemma 17.** *Let $\mathcal{A}$ be a DRA with n states such that $L(\mathcal{A}) \neq \varnothing$. Then there exists w such that $w \in L(\mathcal{A})$ and $|w|$ is bounded by n.*

Given a data word u, we let $u^{-1}L = \{v \mid uv \in L\}$ be the residual language of L by u. Bounding Mem is similar to bounding D. We recall that for Mem, we need two samples of the form wu and $w(u[a/b])$ for any $w \in St$ and $a \in mem_L(w)$, such that $wu \not\simeq_L w(u[a/b])$. We show that $wu \not\simeq_L (w[a/b])u$ since b is fresh in wu. Since samples $w \in St$ have been bounded already, it remains to bound u, i.e. show that the inequality $w^{-1}L \neq (w[a/b])^{-1}L$ is witnessed by a polysize word u. Similarly, for D, given $w \in St$ and $z \in Tr$, we have to bound w', z' such that $mem_L(w)w' \simeq mem_L(z)z'$ and $ww' \not\simeq_L zz'$. We prove that the latter is equivalent to the existence of a data permutation τ such that $z' = \tau(w')$ and $\tau(w)z' \not\simeq_L zz'$. So, we only need to bound z' satisfying $\tau(w)z' \not\simeq_L zz'$, i.e. show that the inequality $(\tau(w))^{-1}L \neq z^{-1}L$ is witnessed by a polysize word z'. In general, for a DRA $\mathcal{A}$ recognizing a language K and a configuration κ, we let $\kappa^{-1}K$ be the set of words on which $\mathcal{A}$ has an accepting run starting in κ. The polysize of Mem and D is then a consequence of:

▶ **Theorem 18.** *Given a DRA $\mathcal{A}$ recognizing a language K and two configurations κ_1, κ_2, if $\kappa_1^{-1}K \neq \kappa_2^{-1}K$, then there is $w \in \mathcal{D}^*$ of polynomial length such that $w \in \kappa_1^{-1}K \not\Leftrightarrow w \in \kappa_2^{-1}K$.*

A low-hanging consequence of the latter theorem is that DRA equivalence is solvable in coNP, which is new for this model of DRA. Note that for DFA, proving a result like Theorem 18 is easy by using Boolean operations on DFA and short witness for DFA non-emptiness. However in the data setting, intersection of DRA involves an exponential blow-up, for instance to maintain that registers should hold distinct values[4]. Instead, to prove Theorem 18, we show that given two configurations of a DRA, if they are not *bisimilar*, then this is witnessed by a polynomial word. We rely on group-theoretic methods as developed for fresh register automata in [33], which we adapt to the DRA model defined in this paper. Unfortunately, it does not seem that DRA can be transformed in PTime into any of the models of [33], thus preventing us from directly applying those results. We reprove them with a slight adaptation to our setting. We now give an overview of the procedure.

The space of configurations of a DRA form a labelled transition system (where the data are labels of the transitions). Bisimulation is defined classically, see e.g. [4], and is seen as an Attacker/Defender game: Attacker picks a data and transition, Defender replies by a transition on the same data, etc. In the deterministic case, a strategy for Attacker is just a data word. We prove that if Attacker has a strategy, i.e. a word, it has a small strategy (of poly-length in the size of the DRA). To show the latter, the concrete bisimulation relation, which is infinite, is finitely but completely abstracted, based on the observation that the data themselves are not important, only equalities between data matter. Based on this, any pair of configurations $(q_1, \alpha_1), (q_2, \alpha_2) \in Q \times \mathcal{D}^\ell$ is abstracted as a triple (q_1, π, q_2) where $\pi : [1, \ell] \hookrightarrow [1, \ell]$ is a partial permutation s.t. $\pi(i) = j$ iff $\alpha_1[i] = \alpha_2[j]$. A relation is then defined over such triples, called *symbolic bisimulation*, which reflects in an abstract way the moves of Attacker and Defender in the concrete bisimulation game. It is then shown that two configurations are bisimilar iff their abstraction as a triple is in the symbolic bisimulation (in other words, the abstraction is sound and complete).

The symbolic bisimulation relation can be computed as the greatest fixpoint of a monotonic function F, generating a decreasing chain of intermediate relations $\mathcal{U}_0 \supsetneq \cdots \supsetneq \mathcal{U}_n = \mathcal{U}_{n+1}$ converging to the symbolic bisimulation relation in n steps. From this chain, it is possible to

[4] We conjecture the exponential intersection blow-up is unavoidable.

extract for any non-bisimilar configurations, a word strategy for Attacker of length at most n. The group-theoretic arguments come into play to bound polynomially n, as introduced in [33]. All the sets $\mathcal{U}_i$ bear a lot of symmetries: for example if $(q_1, \pi, q_2) \in \mathcal{U}_i$ for some i, then $(q_2, \pi^{-1}, q_1) \in \mathcal{U}_i$ (closure by inverse). In fact, this is precisely where the fact that registers hold distinct data is crucial: it implies that π is a permutation and not an arbitrary relation, and so it can be inverted. The sets $\mathcal{U}_i$ enjoy other closures (such as composition) allowing one to show that some characteristic subset of permutations of $\mathcal{U}_i$ form a subgroup of the symmetric group, ordered by the subgroup relation for increasing i. The final argument follows from [3]: chains of subgroups of the symmetric group are linearly bounded.

6 Future work

While we have shown that our learning algorithm runs in PTime, there are interesting questions related to implementation and in particular efficient data structures, for instance to deal with sample prefixes, and to quickly check for completability. As mentioned in Remark 9, it is worth investigating heuristics for choosing target states of new created transitions when there are multiple candidates, for example adaptations of evidence-based heuristics [17] to the context of data languages. The same question arises when erasing registers: there might be several possible sets E.

An interesting future direction, related to the implementation of our algorithm, is to make it incremental, in the sense that it does not have to restart the whole procedure from scratch whenever a new example is added to the set of samples, in the spirit of [18].

Finally, a natural continuation is to investigate active learning for the register automata model of this paper. As mentioned in the introduction, all known results about active learning for regular data languages have exponential time complexity, but this model has not been investigated yet, to the best of our knowledge.

References

1 Parosh Aziz Abdulla, C. Aiswarya, and Mohamed Faouzi Atig. Data communicating processes with unreliable channels. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 166–175. ACM, 2016. doi:10.1145/2933575.2934535.

2 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Ahmet Kara, and Othmane Rezine. Verification of dynamic register automata. In Venkatesh Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, volume 29 of *LIPIcs*, pages 653–665. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.653.

3 László Babai. On the length of subgroup chains in the symmetric group. *Communications in Algebra*, 14(9):1729–1736, 1986. doi:10.1080/00927878608823393.

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

5 Mrudula Balachander, Emmanuel Filiot, and Jean-François Raskin. LTL reactive synthesis with a few hints. In Sriram Sankaranarayanan and Natasha Sharygina, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part II*, volume 13994 of *Lecture Notes in Computer Science*, pages 309–328. Springer, 2023. doi:10.1007/978-3-031-30820-8_20.

6 Michael Benedikt, Clemens Ley, and Gabriele Puppis. What you must remember when processing data words. In Alberto H. F. Laender and Laks V. S. Lakshmanan, editors, *Proceedings of the 4th Alberto Mendelzon International Workshop on Foundations of Data Management, Buenos Aires, Argentina, May 17-20, 2010*, volume 619 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010. URL: https://ceur-ws.org/Vol-619/paper11.pdf.

7 Therese Berg, Bengt Jonsson, and Harald Raffelt. Regular inference for state machines using domains with equality tests. In José Luiz Fiadeiro and Paola Inverardi, editors, *Fundamental Approaches to Software Engineering, 11th International Conference, FASE 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4961 of *Lecture Notes in Computer Science*, pages 317–331. Springer, 2008. doi:10.1007/978-3-540-78743-3_24.

8 León Bohn and Christof Löding. Constructing deterministic ω-automata from examples by an extension of the RPNI algorithm. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27, 2021, Tallinn, Estonia*, volume 202 of *LIPIcs*, pages 20:1–20:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.MFCS.2021.20.

9 León Bohn and Christof Löding. Passive learning of deterministic büchi automata by combinations of dfas. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 114:1–114:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.114.

10 Mikolaj Bojanczyk. Orbit-finite sets and their algorithms (invited talk). In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 1:1–1:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.1.

11 Mikolaj Bojanczyk, Bartek Klin, and Slawomir Lasota. Automata theory in nominal sets. *Log. Methods Comput. Sci.*, 10(3), 2014. doi:10.2168/LMCS-10(3:4)2014.

12 Mikolaj Bojanczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, and Claire David. Two-variable logic on words with data. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 7–16. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.51.

13 Benedikt Bollig. An automaton over data words that captures EMSO logic. In Joost-Pieter Katoen and Barbara König, editors, *CONCUR 2011 - Concurrency Theory - 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011. Proceedings*, volume 6901 of *Lecture Notes in Computer Science*, pages 171–186. Springer, 2011. doi:10.1007/978-3-642-23217-6_12.

14 Benedikt Bollig, Aiswarya Cyriac, Paul Gastin, and K. Narayan Kumar. Model checking languages of data words. In Lars Birkedal, editor, *Foundations of Software Science and Computational Structures - 15th International Conference, FOSSACS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7213 of *Lecture Notes in Computer Science*, pages 391–405. Springer, 2012. doi:10.1007/978-3-642-28729-9_26.

15 Julien Carme, Rémi Gilleron, Aurélien Lemay, and Joachim Niehren. Interactive learning of node selecting tree transducer. *Mach. Learn.*, 66(1):33–67, 2007. doi:10.1007/s10994-006-9613-8.

16 Colin de la Higuera. Characteristic sets for polynomial grammatical inference. *Mach. Learn.*, 27(2):125–138, 1997. doi:10.1023/A:1007353007695.

17 Colin de la Higuera, José Oncina, and Enrique Vidal. Identification of DFA: data-dependent vs data-independent algorithms. In Laurent Miclet and Colin de la Higuera, editors, *Grammatical Inference: Learning Syntax from Sentences, 3rd International Colloquium, ICGI-96, Montpellier, France, September 25-27, 1996, Proceedings*, volume 1147 of *Lecture Notes in Computer Science*, pages 313–325. Springer, 1996. doi:10.1007/BFb0033365.

18 Pierre Dupont. Incremental regular inference. In Laurent Miclet and Colin de la Higuera, editors, *Grammatical Interference: Learning Syntax from Sentences*, pages 222–237, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. doi:10.1007/BFb0033357.

19 Léo Exibard, Emmanuel Filiot, and Ayrat Khalimov. Church synthesis on register automata over linearly ordered data domains. *Formal Methods Syst. Des.*, 61(2):290–337, 2022. doi:10.1007/s10703-023-00435-w.

20 Léo Exibard, Emmanuel Filiot, and Pierre-Alain Reynier. Synthesis of data word transducers. *Log. Methods Comput. Sci.*, 17(1), 2021. URL: https://lmcs.episciences.org/7279.

21 Dana Fisman, Hadar Frenkel, and Sandra Zilles. Inferring symbolic automata. *Log. Methods Comput. Sci.*, 19(2), 2023. doi:10.46298/lmcs-19(2:5)2023.

22 Pedro García and Jose Oncina. Inference of recognizable tree sets. *Tech. rep., Departamento de Sistemas Informáticos y Computación, Universidad de Alicante. DSIC-II/47/93*, 1993.

23 E. Mark Gold. Language identification in the limit. *Inf. Control.*, 10(5):447–474, 1967. doi:10.1016/S0019-9958(67)91165-5.

24 Radu Grigore and Nikos Tzevelekos. History-register automata. *Log. Methods Comput. Sci.*, 12(1), 2016. doi:10.2168/LMCS-12(1:7)2016.

25 Falk Howar, Bernhard Steffen, Bengt Jonsson, and Sofia Cassel. Inferring canonical register automata. In Viktor Kuncak and Andrey Rybalchenko, editors, *Verification, Model Checking, and Abstract Interpretation - 13th International Conference, VMCAI 2012, Philadelphia, PA, USA, January 22-24, 2012. Proceedings*, volume 7148 of *Lecture Notes in Computer Science*, pages 251–266. Springer, 2012. doi:10.1007/978-3-642-27940-9_17.

26 Malte Isberner, Falk Howar, and Bernhard Steffen. Learning register automata: from languages to program structures. *Mach. Learn.*, 96(1-2):65–98, 2014. doi:10.1007/s10994-013-5419-7.

27 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9.

28 Ayrat Khalimov and Orna Kupferman. Register-bounded synthesis. In Wan J. Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPIcs*, pages 25:1–25:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CONCUR.2019.25.

29 Martin Leucker. Learning meets verification. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, editors, *Formal Methods for Components and Objects, 5th International Symposium, FMCO 2006, Amsterdam, The Netherlands, November 7-10, 2006, Revised Lectures*, volume 4709 of *Lecture Notes in Computer Science*, pages 127–151. Springer, 2006. doi:10.1007/978-3-540-74792-5_6.

30 Damián López and Pedro García. On the inference of finite state automata from positive and negative data. *Topics in Grammatical Inference*, pages 73–112, 2016.

31 Amaldev Manuel, Anca Muscholl, and Gabriele Puppis. Walking on data words. *Theory Comput. Syst.*, 59(2):180–208, 2016. doi:10.1007/s00224-014-9603-3.

32 Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin, and Michal Szynwelski. Learning nominal automata. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 613–625. ACM, 2017. doi:10.1145/3009837.3009879.

33 Andrzej S. Murawski, Steven J. Ramsay, and Nikos Tzevelekos. Bisimilarity in fresh-register automata. In *Proceedings of the 2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, LICS '15, pages 156–167, USA, 2015. IEEE Computer Society. doi:10.1109/LICS.2015.24.

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

35 Nikos Tzevelekos. Fresh-register automata. In Thomas Ball and Mooly Sagiv, editors, *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 295–306. ACM, 2011. doi:10.1145/1926385.1926420.

A Completability of DFA

Before proving Proposition 4, we define the notion of prefix-tree acceptor of a finite set S, as a tree-shaped DFA accepting exactly S. For any finite set $S \subseteq \Sigma^*$, $\mathsf{PTA}(S) = (Q = \mathsf{Prefs}(S), q_0 = \varepsilon, \delta, F = S)$, where $\forall wa \in \mathsf{Prefs}(S) : \delta(w, a) = wa$. Clearly, $L(\mathsf{PTA}(S)) = S$.

▶ **Proposition 4.** *A DFA $\mathcal{A}$ is S-completable iff $I^- \cap L(\mathcal{A}) = \varnothing$ and there do* not *exist $u_1, u_2, z \in \Sigma^*$ such that $u_1 z \in I^+$, $u_2 z \in I^-$ and $\delta^*(q_0, u_1) = \delta^*(q_0, u_2)$.*

Proof. ($\Longrightarrow$) Clearly, if $I^- \cap L(\mathcal{A}) \neq \varnothing$, then $\mathcal{A}$ is not completable, since by definition of completability, final states have to be preserved.

Now, let $w_1 = u_1 z \in I^+$ and $w_2 = u_2 z \in I^-$ such that $\delta^*_{\mathcal{A}}(q_0, u_1) = \delta^*_{\mathcal{A}}(q_0, u_2)$, then any S-consistent extension $\mathcal{A}^{\mathsf{ext}}$ of $\mathcal{A}$ have the following property: $\delta^*_{\mathcal{A}^*}(q_0, w_1) = \delta^*_{\mathcal{A}^*}(q_0, w_2)$ and therefore, $w_1 \in L(\mathcal{A}^{\mathsf{ext}})$ iff $w_2 \in \mathcal{A}^{\mathsf{ext}}$, hence $\mathcal{A}^{\mathsf{ext}}$ is not S-consistent. Contradiction.

($\Longleftarrow$) We construct an S-consistent extension $\mathcal{A}^{\mathsf{ext}}$ of $\mathcal{A}$. For every state q of $\mathcal{A}$, let L_q be the set of words w such that $q_0 \xrightarrow{w}_{\mathcal{A}} q$. For all letter $a \in \Sigma$ such that $\delta(q, a)$ is undefined, let $I^+_{q,a} = (L_q a)^{-1} I^+$ be the set of words v such that $uav \in I^+$ for some $u \in L_q$. We construct $T_{q,a} = \mathsf{PTA}(I^+_{q,a})$ with root node t_ε and add a transition from q to the initial state of $T_{q,a}$ on reading a. The set of accepting states is set to be the accepting states of $\mathcal{A}$ plus the states reached by words from I^+. We make such construction for any pair (q, a) such that $\delta(q, a)$ is undefined. This results in an extension $\mathcal{A}^{\mathsf{ext}}$ of $\mathcal{A}$, which is now able to read any word of I^+.

We now show A^{ext} is S-consistent. The inclusion $I^+ \subseteq \mathcal{A}^{\mathsf{ext}}$ is immediate by construction. Let us show $I^- \cap L(\mathcal{A}^{\mathsf{ext}}) = \varnothing$. Suppose, for the sake of contradiction, that there is some $w' \in I^- \cap L(\mathcal{A}^{\mathsf{ext}})$. Let $p = \delta^*_{\mathsf{ext}}(q_0, w')$. We consider two cases:

1. $p \notin Q_{\mathcal{A}}$, i.e. p is a state of some added PTA $T_{q,a}$. Hence there is some $w \in I^+$ such that $\delta^*_{\mathcal{A}^{\mathsf{ext}}}(q_0, w) = p$. By construction of $\mathcal{A}^{\mathsf{ext}}$ (an in particular because $T_{q,a}$ is tree-shaped), it implies that w and w' can be factorized as $w = u_1 z$ and $w' = u_2 z$ for some $u_1, u_2 \in \Sigma^*$, such that $\delta^*_{\mathcal{A}}(q_0, u_1) = \delta^*_{\mathcal{A}}(q_0, u_2) = q$ exists and in $\mathcal{A}^{\mathsf{ext}}$, there exists a transition from q to the initial state of $T_{q,a}$ where a is the first letter of z. This contradicts our assumption, since $w = u_1 z \in I^+$ while $w' = u_2 z \in I^-$.
2. $p \in Q_{\mathcal{A}}$: since $I^- \cap L(\mathcal{A}) = \varnothing$, then p was not accepting in $\mathcal{A}$. Hence there must be some positive example $w \in I^+$ such that $q_0 \xrightarrow{w}_{\mathcal{A}} p$. We then immediately get a contradiction, by taking $u_1 = w$, $u_2 = w'$ and $z = \epsilon$. ◀

▶ **Corollary 5.** *Given a DFA $\mathcal{A}$ and a (finite) sample set S, it can be checked in time $O(||\mathcal{A}||.||S||)$ whether $\mathcal{A}$ is S-completable.*

Proof. We check the characterization given by Proposition 4. Checking $I^- \cap L(\mathcal{A}) = \varnothing$ is done in $O(||I^-||.||\mathcal{A}||)$. For the second condition, we check the non-existence of u_1, u_2, z as in Proposition 4. Let $\mathsf{Suff}(S)$ be the set of suffixes of words of S. For any word $z \in \mathsf{Suff}(S)$, the objective is to compute two sets P^+_z, P^-_z, where for $s \in \{+, -\}$, $P^s_z = \{\delta^*(q_0, u) \mid u \in \Sigma^* \wedge uz \in I^s\}$ (this set can be empty). Then, it suffices to check that no node $z \in \mathsf{Suff}(S)$ satisfies $P^+_z \cap P^-_z \neq \varnothing$.

To compute the sets P^+_z and P^-_z for all $z \in \mathsf{Suff}(S)$ in $O(||\mathcal{A}||.||S||)$, the algorithm first builds a suffix tree based on S, whose set of nodes is $\mathsf{Suff}(S)$. The root is ϵ, and if $u \in \Sigma^*$ and $\sigma u \in \Sigma^*$ are two nodes of the tree, there is an edge from u to σu. Therefore, if a node z is a common ancestor of two nodes u_1 and u_2, then z is a common suffix of u_1 and u_2. Constructing the tree is done in time $O(||S||)$.

The tree is then processed bottom-up, exploiting the following claim:

▷ Claim. For all $z \in \mathsf{Suff}(S)$, all $s \in \{+,-\}$, $P^s_z = \{\delta(q,\sigma) \mid \sigma z \in \mathsf{Suff}(S) \wedge q \in P^s_{\sigma z}\} \cup X$ where $X = \{q_0\}$ if $z \in I^s$, otherwise $X = \varnothing$.

Proof of the claim. Let denote by Q^s_z the rhs of the above equality. We show that $P^s_z = Q^s_z$. Let $p \in P^s_z$. Then $p = \delta^*(q_0, u)$ for some $uz \in I^s$. If $u = \epsilon$, then $p = q_0$ and $z \in I^s$, so $p \in X$ and $p \in Q^s_z$. If $u = u'\sigma$ for some u', then $u'\sigma z \in I^s$ and $\sigma z \in \mathsf{Suff}(S)$. Let $q = \delta^*(q_0, u')$. Then $p = \delta(q,\sigma)$ and we can conclude since $q \in P^s_{\sigma z}$ by definition of $P^s_{\sigma z}$.

Conversely, let $p \in Q^s_z$. There are two cases: either $p = q_0$ and $z \in I^s$, we get immediately that $p \in P^s_z$, or $p \in \{\delta(q,\sigma) \mid \sigma z \in \mathsf{Suff}(S) \wedge q \in P^s_{\sigma z}\}$. Since $q \in P^s_{\sigma z}$, $q = \delta^*(q_0, u)$ for some u such that $u\sigma z \in I^s$. As $p = \delta(q,\sigma)$, we get that $p = \delta^*(q_0, u\sigma)$ and since $u\sigma z \in I^s$, we can conclude that $p \in P^s_z$. ◁

The algorithm is then immediate by processing the tree bottom-up, applying δ on the sets computed for the children of any node currently processed. The overall computation is in time $O(||\mathcal{A}||.||S||)$. ◀

B Completability for DRA

▶ **Proposition 6.** *A DRA $\mathcal{A}$ is S-completable for a sample set $S = (I^+, I^-)$, iff $L(\mathcal{A}) \cap I^- = \varnothing$ and there do not exist words $w \in I^+$, $z \in I^-$, state q and words σ, σ' such that: $w = w_1w_2 \wedge z = z_1z_2 \wedge (q_0,\epsilon) \xrightarrow{w_1}_{\mathcal{A}} (q,\sigma) \wedge (q_0,\epsilon) \xrightarrow{z_1}_{\mathcal{A}} (q,\sigma') \wedge \sigma w_2 \simeq \sigma' z_2$.*

Proof. ($\Leftarrow$) Let $W = \{s_1, \ldots, s_n\} = I^+ \cup I^-$ and for all $0 \leq i \leq n$, let $W_i = \{s_1, \ldots, s_i\}$, $I^+_i = I^+ \cap W_i$, $I^-_i = I^- \cap W_i$ and $S_i = (I^+_i, I^-_i)$. Note that $S_0 = (\varnothing, I^-)$ and $S_n = S$. We show by induction on i how to build a sequence of DRA $\mathcal{A}_0 = \mathcal{A} \preceq \mathcal{A}_1 \preceq \cdots \preceq \mathcal{A}_n$ such for each $0 \leq i \leq n$, A_i is S_i-consistent and there does not exists $w \in I^+, z \in I^-$ and state q and words σ, σ' such that:

$$w = w_1w_2 \wedge z = z_1z_2 \wedge (q_0,\epsilon) \xrightarrow{w_1}_{\mathcal{A}_i} (q,\sigma) \wedge (q_0,\epsilon) \xrightarrow{z_1}_{\mathcal{A}_i} (q,\sigma') \wedge \sigma w_2 \simeq \sigma' z_2 \tag{1}$$

The base case $i = 0$ is clear by hypothesis, since $\mathcal{A}_0 = \mathcal{A}$. Let $i > 1$ and assume by IH that $\mathcal{A}_{i-1}$ is S_{i-1}-consistent. The DRA $\mathcal{A}_i = \langle Q_i, k_i, \lambda_i, T_i, q^i_0, F_i\rangle$ is obtained from $\mathcal{A}_{i-1} = \langle Q_{i-1}, k_{i-1}, \lambda_{i-1}, T_{i-1}, q^{i-1}_0, F_{i-1}\rangle$ as follows. We consider two cases:

(1) If $\mathcal{A}_{i-1}$ reads s_i reaching a configuration (q,σ) then let q be a final state iff $s_i \in I^+$. By contradiction suppose that $\mathcal{A}_i$ is not S_i-consistent. Then, there exists $s_{j<i}$ such that both reading s_i and reading s_j, $\mathcal{A}_{i-1}$ gets to a configuration where the first component is q, say $(q,\sigma), (q,\delta)$, however only one between s_i, s_j is a positive sample. This contradicts the fact that $\mathcal{A}_{i-1}$ respects condition (1), since $\sigma \simeq \delta$ (by definition of DRA).

(2) Otherwise, let decompose s_i into $s_i = s^1_i s^2_i$ where s^1_i is the longest prefix of s_i which can be read by $\mathcal{A}_{i-1}$, and let (q,σ) be the configuration reached by $\mathcal{A}_{i-1}$ upon reading s^1_i. To obtain $\mathcal{A}_i$, we complete $\mathcal{A}_{i-1}$ with new transitions from state q, so that $\mathcal{A}_i$ can only read s^2_i from q (and all $\simeq$-equivalent words). Let $a_1 \ldots a_m = s^2_i$, $\sigma_0 = \sigma$, and for all $1 \leq k \leq m$, $\sigma_k = \mathsf{drop}_{E_k}(\sigma_{k-1}a_k)$ where $E_k = \{x\}$ such that $a_k = \sigma_{k-1}[x]$ if it exists, otherwise $E_k = \varnothing$. We now construct $\mathcal{A}_i$. Let $p_1, \ldots, p_m \notin Q_i$ be fresh new states. Let $Q_i = Q_{i-1} \cup \{p_1 \ldots p_m\}$ and $p_0 = q$, $T_i = T_{i-1} \cup \{(p_k, \sigma_k \cdot a_{k+1}, E_{k+1}, p_{k+1}) \mid 0 \leq j < m\}$, $q^i_0 = q^{i-1}_0$, $F_i = F_{i-1} \cup \{p_m\}$ if $s_i \in I^+$, otherwise $F_i = F_{i-1}$.

We first show that $\mathcal{A}_i$ meets Condition (1). If it is not the case, then, there exists a positive sample $u_1 = w_1z_1$ and a negative sample $u_2 = w_2z_2$ and runs $(q^i_0,\epsilon) \xrightarrow{w_1}_{\mathcal{A}_i} (t,\lambda_1)$ and $(q^i_0,\epsilon) \xrightarrow{w_2}_{\mathcal{A}_i} (t,\lambda_2)$ such that $\lambda_1z_1 \simeq \lambda_2z_2$. SInce $\mathcal{A}_{i-1}$ satisfies Condition (1), necessarily, state t is a newly added state, so one of $\{p_1, \ldots, p_m\}$, say $t = p_j$ for some j. So, w_1 and w_2 can be further decomposed into $w'_1w''_1$ and $w'_2w''_2$ with $|w''_1| = |w''_2|$, and the runs into:

$$(q_0^i, \epsilon) \xrightarrow{w_1'}_{\mathcal{A}_{i-1}} (q, \lambda_1') \xrightarrow{w_1''}_{\mathcal{A}_i} (p_j, \lambda_1)$$
$$(q_0^i, \epsilon) \xrightarrow{w_2'}_{\mathcal{A}_{i-1}} (q, \lambda_2') \xrightarrow{w_2''}_{\mathcal{A}_i} (p_j, \lambda_2)$$

By definition of $\mathcal{A}_i$, $\sigma a_1 \dots a_j \simeq \lambda_1' w_1''$ and $\sigma a_1 \dots a_j \simeq \lambda_2' w_2''$ (where σ, $a_1, \dots, a_m$ have been used before to define $\mathcal{A}_i$), hence $\lambda_1' w_1'' \simeq \lambda_2' w_2''$. Now, again by definition of $\mathcal{A}_i$, λ_1 (resp. λ_2) is made of exactly one occurrence of each data appearing in $\lambda_1' w_1''$ (resp. $\lambda_2' w_2''$). From this fact, the fact that $\lambda_1 z_1 \simeq \lambda_2 z_2$ and $\lambda_1' w_1'' \simeq \lambda_2' w_2''$, we get that $\lambda_1' w_1'' z_1 \simeq \lambda_2' w_2'' z_2$, contradicting the fact that $\mathcal{A}_{i-1}$ respects Condition (1).

It remains to show that $\mathcal{A}_i$ is S_i-consistent. By construction, $\mathcal{A}_i$ accepts all words in I_i^+, because $\mathcal{A}_i$ accepts all words in I_{i-1}^+, by IH, and we have not removed accepting states, only added potentially one accepting state to accept s_i. We show that $\mathcal{A}_i$ rejects all words in I_i^-. If it were not the case, then let $s \in I_i^-$ accepted by $\mathcal{A}_i$. As $\mathcal{A}_{i-1}$ is S_{i-1}-consistent, it is necessarily the case that s has an accepting run towards a new accepting state, and this new accepting state can only be p_m, and $s_i \in I^+$. From this we immediately get that Condition (1) is not satisfied, which is a contradiction.

($\Rightarrow$) Let $\mathcal{A}' = (Q', k', \lambda', T', q_0', F')$ be the S-consistent DRA that completes $\mathcal{A}$ and let $\Phi : Q \to Q'$ the injective function witnessing it. If $L(\mathcal{A}) \cap I^- \neq \varnothing$, since Φ must preserve accepting states, then $L(\mathcal{A}') \cap I^- \neq \varnothing$, which contradicts that $\mathcal{A}'$ is S-consistent. On the other hand, suppose that there exist $w = w_1 w_2 \in I^+, z = z_1 z_2 \in I^-$ such that $(q_0, \epsilon) \xrightarrow{w_1}_{\mathcal{A}} (q, \sigma) \wedge (q_0, \epsilon) \xrightarrow{z_1}_{\mathcal{A}} (q, \sigma') \wedge \sigma w_2 \simeq \sigma' z_2$. Then, $\mathcal{A}'$ admits runs $(\Phi(q_0), \epsilon) \xrightarrow{w_1}_{\mathcal{A}'} (\Phi(q), \sigma) \xrightarrow{w_2}_{\mathcal{A}'} (p, \delta) \wedge (\Phi(q_0), \epsilon) \xrightarrow{z_1}_{\mathcal{A}'} (\Phi(q), \sigma') \xrightarrow{z_2}_{\mathcal{A}'} (p, \delta')$. If $p \in F'$, then $\mathcal{A}'$ accepts both w and z, otherwise it rejects them, contradicting its S-consistency. ◀