Automata Equipped with Auxiliary Data Structures and Regular Realizability Problems

Alexander Rubtsov* Mikhail Vyalyi†

October 11, 2022

Abstract

We consider general computational models: one-way and two-way finite automata, and logarithmic space Turing machines, all equipped with an auxiliary data structure (ADS). The definition of an ADS is based on the language of protocols of work with the ADS. We describe the connection of automata-based models with "Balloon automata" that are another general formalization of automata equipped with an ADS presented by Hopcroft and Ullman in 1967. This definition establishes the connection between the non-emptiness problem for one-way automata with ADS, languages recognizable by nondeterministic log-space Turing machines equipped with the same ADS, and a regular realizability problem (NRR) for the language of ADS' protocols. The NRR problem is to verify whether the regular language on the input has a non-empty intersection with the language of protocols. The computational complexity of these problems (and languages) is the same up to log-space reductions.

Keywords: Finite automata; Balloon automata; Auxiliary data structures

1 Introduction

Many computational models are derived from (one-way) finite automata (FAs) via equipping them with an auxiliary data structure (ADS). The best-known model of this kind is pushdown automata (PDAs), the deterministic version of which is widely used in compilers. Other examples are k-counter automata, (k, r)-reversal-bounded counter automata (equipped with k counters each of which can switch between increasing and decreasing modes at most r times), stack automata, nested stack automata, bag automata [5], set automata (SAs) [8] and their another variant [9]; more examples can be found in [7].

---

*Faculty of Computer Science, National Research University Higher School of Economics, Pokrovsky boulevard 11, Moscow, 109028, Russia, `rubtsov99@gmail.com`

†Faculty of Computer Science, National Research University Higher School of Economics, Pokrovsky boulevard 11, Moscow, 109028, Russia, `vyalyi@gmail.com`

During the investigation of balloon automata (BAs) [7], Hopcroft and Ullman connected the decidability of the membership and the emptiness problems for one-way and two-way models; we denote them as M-$xyBA$ and E-$xyBA$ respectively, where $x = 1$ denotes one-way and $x = 2$ denotes two-way models, and $y \in \{D, N\}$ stands for determinism or nondeterminism respectively. Eq. (1) summarizes results on decidability questions from [7], where $\leq_T$ is a *Turing-reduction* and $\{A, B\}$ means that $A \leq_T B$ and $B \leq_T A$.

$$\{\text{M-1DBA}, \text{M-2DBA}\} \leq_T \{\text{E-1DBA}, \text{E-1NBA}, \text{M-1NBA}, \text{M-2NBA}\} \leq_T$$
$$\leq_T \text{E-2DBA} \leq_T \text{E-2NBA}. \tag{1}$$

We remark that the relation E-1NBA $\leq_T$ E-1DBA was proved for the case of at least a two-letter input alphabet.

While a lot of models can be described as BA, it is hard to invent such a model with good computational properties. One of the reasons is that the equipment of finite automata with a complex data structure (or with several simple data structures) often leads to a universal computational model. For example, FAs equipped with two pushdown stores are equivalent to Turing machines (TMs), as well as FAs equipped with two non-restricted counters.

In this paper, we investigate the computational power of FAs equipped with an ADS. We describe the model using the language of correct protocols of work with the ADS. We provide a general approach to analyze the complexity of the emptiness problem and prove the following non-trivial result. If FAs are equipped with an ADS and nondeterministic logarithmic space TMs (log-TMs, see the definition in [16]) are equipped with the same ADS, then the FAs' non-emptines problem and the TMs-recognizable languages are of the same complexity (up to log-space reductions). Our key tool is the regular realizability problem (see Definition 1 below).

1.1 Our Contribution

BAs were initially defined as automata with access to additional storage of unspecified structure—*the balloon*. A rather general axioms were imposed for the balloon and the interaction of the balloon and the automaton (see Definition 4 below). In this paper, we propose another definition based on a language of the ADS' protocols that we denote as P, so we refer to the ADS as B_P. We prove that languages recognizable by $1NB_PA$ form not just a rational cone as in the case of 1NBA [7], but a principal rational cone generated by P (we provide the definition in Section 2.2).

This reformulation guarantees good structural properties, some of them follow from the connection with BA (Section 4), and provides the relation between E-$1NB_PA$ and the *nondeterministic regular realizability problem*.

Definition 1. Fix a formal language F called a *filter*, the parameter of *regular realizability problems* $\text{DRR}(F)$ and $\text{NRR}(F)$ that are the problems of verifying non-emptiness of the intersection of the filter F with a regular language $L(\mathcal{A})$

described via the DFA or NFA $\mathcal{A}$ respectively. Formally,

$$\mathrm{NRR}(F) = \{\mathcal{A} \mid \mathcal{A} \text{ is an NFA and } L(\mathcal{A}) \cap F \neq \varnothing\},$$
$$\mathrm{DRR}(F) = \{\mathcal{A} \mid \mathcal{A} \text{ is a DFA and } L(\mathcal{A}) \cap F \neq \varnothing\}.$$

RR problems have independently been studied under the name regular intersection emptiness problems [21, 22]. A restricted version of RR problem (for context-free filters only) is a well-known CFL-reachability problem, which is related to problems in interprocedural program analysis [3, 4, 6, 10, 11, 23].

In this paper we focus on the computational complexity, so we use the weakest reduction suitable for our needs, the deterministic log-space reduction that we denote as $\leq_{\log}$. If $A \leq_{\log} B$ and $B \leq_{\log} A$ we write $A \sim_{\log} B$ and say that A and B are *log-space equivalent*. Note that in our constructions, emptiness and membership problems are the sets of instances' descriptions with positive answers, i.e., E-xyB$_\mathsf{P}$A $= \{\langle M\rangle \mid L(M) = \varnothing\}$, M-$xy$B$_\mathsf{P}$A $= \{\langle M, w\rangle \mid w \in L(M)\}$, where M is a xyB$_\mathsf{P}$A and $\langle x\rangle$ is the description of x. So, $\overline{\text{E-}xy\text{B}_\mathsf{P}\text{A}} = \{\langle M\rangle \mid L(M) \neq \varnothing\}$. We prove that $\overline{\text{E-1NB}_\mathsf{P}\text{A}} \sim_{\log} \mathrm{NRR}(\mathsf{P})$. Based on this result, we establish computational universality of $\overline{\text{E-1NB}_\mathsf{P}\text{A}}$ (see Theorem 34 below). Note that in the universality result we need Turing reductions in polynomial time instead of log-space reductions.

We equip with ADS not only FAs but also log-TMs. We denote deterministic and nondeterministic log-TMs equipped with an ADS B$_\mathsf{P}$ as DB$_\mathsf{P}$log-TM and NB$_\mathsf{P}$log-TM respectively. We prove that

$$\mathrm{NRR}(\mathsf{P}) \sim_{\log} \mathscr{L}(\mathrm{NB}_\mathsf{P}\mathrm{log\text{-}TM}) = \{L \mid L \leq_{\log} \mathrm{NRR}(\mathsf{P})\}, \tag{2}$$

hereinafter $\mathscr{L}(\mathrm{model})$ is the class of languages recognizable by the model. If P is a problem (formal language) and S is a set of problems (class of formal languages) the reductions mean as follows. $P \leq S$ means that $\exists P' \in S : P \leq P'$ and $S \leq P$ means that $\forall P' \in S : P' \leq P$; $S \sim P$ means $(P \leq S) \wedge (S \leq P)$.

It is easy to verify that in the original proofs in [7], Turing reductions in (1) can be replaced by the log-space reductions provided we replace the emptiness problems with non-emptiness ones. So, we obtain

$$\{\text{M-1DB}_\mathsf{P}\text{A}, \text{M-2DB}_\mathsf{P}\text{A}\} \leq_{\log} \{\overline{\text{E-1DB}_\mathsf{P}\text{A}}, \overline{\text{E-1NB}_\mathsf{P}\text{A}}, \text{M-1NB}_\mathsf{P}\text{A}, \text{M-2NB}_\mathsf{P}\text{A},$$
$$\mathrm{NRR}(\mathsf{P}), \mathscr{L}(\mathrm{NB}_\mathsf{P}\mathrm{log\text{-}TM})\} \leq_{\log} \overline{\text{E-2DBA}} \leq_{\log} \overline{\text{E-2NBA}} \leq_{\log} \overline{\text{E-NB}_\mathsf{P}\mathrm{log\text{-}TM}}. \tag{3}$$

We also prove the reduction

$$\text{M-1DB}_\mathsf{P}\text{A} \leq_{\log} \mathrm{DRR}(\mathsf{P}). \tag{4}$$

Results (3) combined with known facts imply assertions (5-8), where S is the set data structure as in SA, S_1 is the set data structure that supports the insertion of at most one word, that cannot be removed further but can be tested if a query-word in the set. In $S_{1,|\Gamma|=1}$ the word in the set is over an unary

alphabet, **PSPACE-c** and **NP-c** are subclasses of complete languages.

$$\mathbf{P} = \mathscr{L}(\text{NPDlog-TM}), \text{ where PD is Pushdown store}, \tag{5}$$

$$\mathbf{PSPACE} \supseteq \mathscr{L}(\text{NSlog-TM}), \exists L \in \mathscr{L}(\text{NSlog-TM}) : L \in \mathbf{PSPACE\text{-}c}, \tag{6}$$

$$\mathbf{PSPACE} \supseteq \mathscr{L}(\text{NS}_1\text{log-TM}), \exists L \in \mathscr{L}(\text{NS}_1\text{log-TM}) : L \in \mathbf{PSPACE\text{-}c}, \tag{7}$$

$$\mathbf{NP} \supseteq \mathscr{L}(\text{NS}_{1,|\Gamma|=1}\text{log-TM}), , \exists L \in \mathscr{L}(\text{NS}_{1,|\Gamma|=1}\text{log-TM}) : L \in \mathbf{NP\text{-}c}. \tag{8}$$

Assertion (5) is a well-known fact. Our technique here just shows a new connection: (5) directly follows from the fact that the emptiness problem for PDA is **P**-complete. Assertions (6-8) are new results to the best of our knowledge, we prove them in Section 6. Assertions (7-8) lead to (3) for the corresponding classes of automata. For (6), we have already obtained the result in [14] in the same way and present in this paper the generalized technique.

2 Definitions

2.1 Notation on binary relations

We associate with a binary relation $R \subseteq A \times B$ the corresponding mappings $A \to 2^B$ and $2^A \to 2^B$ that are denoted by the same letter R, so $R(a) = \{b : aRb\}$ and $R(S) = \cup_{a \in S} R(a)$. A relation R is the *composition* of the relations $P \subseteq A \times C$ and $Q \subseteq C \times B$ if $R = \{(a, b) \mid \exists c : aPc \wedge cQb\}$; we denote the composition as $Q \circ P$. In the case of a set $S \subseteq C$ we treat S as a binary relation $S \subseteq C \times \{0, 1\}$ in the composition $S \circ P = S'$ that returns the set $S' \subseteq A$. We denote the reflexive and transitive closure of $R \subseteq A \times A$ by R^*; the symbol $*$ can also be placed above the relation, e.g., $\overset{*}{\vdash}$. We denote by $R^{-1} \subseteq B \times A$ the inverse relation, i.e., $aRb \iff bR^{-1}a$.

2.2 Rational Transductions

Our technique is based on the connection of NRR problems with rational cones. We recall the definitions borrowing them from the book [2]. A *finite state transducer* (FST) is a nondeterministic finite automaton with an output tape, and DFST is the deterministic version of FST. For the deterministic version, it is important that a transducer can write a word (but not only a single symbol) on the output tape on processing a letter from the input tape. Let T be an FST; we also denote by T the corresponding relation, i.e., uTv if there exists a run of T on the input u from the initial state to a final state such that at the end of the run the word v is written on the output tape. The *rational dominance* relation $A \leq_{\text{rat}} B$ holds if there exists an FST T such that $A = T(B)$, here A and B are languages. The relations computable by FSTs are known as *rational relations*. The following lemmata are algorithmic versions of well-known facts (see [2], Chapter III), the first one is the algorithmic version of the Elgot-Mezei theorem. The log-space algorithms follow from straight-forward constructions.

Lemma 2. *For FSTs T_1 and T_2 such that $T_1 \subseteq \Sigma^* \times \Delta^*$, $T_2 \subseteq \Delta^* \times \Gamma^*$, and FA $\mathcal{A}$ such that $L(\mathcal{A}) \subseteq \Delta^*$, there exists an FST T such that $T = T_2 \circ T_1 \subseteq \Sigma^* \times \Gamma^*$, and NFA $\mathcal{B}$ recognizing the language $T_1^{-1} L(\mathcal{A})$. So, the relation $\leq_{\mathrm{rat}}$ is transitive. Moreover, T and $\mathcal{B}$ are constructible in logarithmic space. We denote FST T and NFA $\mathcal{B}$ as $T_2 \circ T_1$ and $\mathcal{A} \circ T_1$ respectively.*

Lemma 3. *For each FST T there exists an FST T^{-1} that computes the inverse relation of the relation T. FST T^{-1} is log-space constructible by FST T.*

A *rational cone* is a family of languages **C** that is closed under the rational dominance relation: $A \leq_{\mathrm{rat}} B$ and $B \in \mathbf{C}$ imply $A \in \mathbf{C}$. If there exists a language $F \in \mathbf{C}$ such that $L \leq_{\mathrm{rat}} F$ for any $L \in \mathbf{C}$, then **C** is a *principal* rational cone generated by F; we denote it as $\mathbf{C} = \mathcal{T}(F)$.

Rational transductions for context-free languages were thoroughly investigated in the 1970s, particularly by the French school. The main results of this research were published in Berstel's book [2]. As described in [2], it follows from the Chomsky-Schützenberger theorem that CFL is a principal rational cone: $\mathsf{CFL} = \mathcal{T}(D_2)$, where D_2 is the Dyck language on two types of brackets.

2.3 Computational Models

Firstly, we define BA. We provide the definition that is equivalent to the original definition from [7] but has technical differences, for the sake of convenience. Then we provide the definitions of other models: the refined definition of Balloon automata in terms of protocols and computational models based on log-TM that are connected with NRR-problem as well as with 1NB$_{\mathsf{P}}$A.

As it said, the balloon is a storage medium of unspecified structure. Thus its states are represented by (a subset of) positive integers. A BA can get limited information about the state of the balloon (the balloon information function in the definition below) and can modify the states of the balloon (the balloon control function). Here we need 1BAs only. So we give the definition for them. The definitions for 2BAs are similar, they are provided in [7].

Definition 4. A 1-way balloon automaton (1BA) is defined by a tuple

$$\langle S, \Sigma_{\rhd\lhd}, B_S, B_I, \mathsf{get}_{\mathsf{B_I}}, \mathsf{upd}_{\mathsf{B_S}}, F, s_0, \delta \rangle, \quad \text{where}$$

- S is the finite set of automaton states.

- $\Sigma_{\rhd\lhd} = \Sigma \cup \{\rhd, \lhd\}$, where Σ is the finite input alphabet and $\rhd, \lhd$ are the endmarkers. The input has the form $\rhd w \lhd$, $w \in \Sigma^*$.

- $B_S \subseteq \mathbb{Z}_{>0}$ is the set of the balloon states.

- B_I is the finite set of the balloon information states.

- $\mathsf{get}_{\mathsf{B_I}} : B_S \to B_I$ is a total computable function (balloon information function).

5

- $\mathsf{upd}_{\mathsf{B_S}}$ is a partially computable function from $S \times B_S$ to B_S (balloon control function).

- $F \subsetneq S$ is the set of the final states.

- $s_0 \in S \setminus F$ is the initial state.

- δ is the transition relation (a partial function for deterministic automata) defined as $\delta \subseteq (S \times \Sigma_{\triangleright\triangleleft,\varepsilon} \times B_I) \times S$; hereinafter $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$ for any alphabet Γ.

Definition 5. A *configuration* of a 1BA is a triple $(q, u, i) \in S \times \Sigma_{\triangleright\triangleleft}^* \times B_S$, where u is the unprocessed part of the input w so u is either $\triangleright w \triangleleft$ or a suffix of $w \triangleleft$. The *initial configuration* of 1BA is $(s_0, \triangleright w \triangleleft, 1)$, a *move* of 1BA is defined by the *relation* $\vdash$ on configurations as follows: $(q, \sigma u, i) \vdash (p, u, j)$, where $\sigma \in \Sigma_{\triangleright\triangleleft,\varepsilon}$ if $j = \mathsf{upd}_{\mathsf{B_S}}(p, i)$, $p \in \delta(q, \sigma, \mathsf{get}_{\mathsf{B_I}}(i))$. A 1BA accepts the input w if there exists a sequence of moves *(computational path)* such that after processing of $\triangleright w \triangleleft$ the final state is reached, i.e., $(s_0, \triangleright w \triangleleft, 1) \vdash^* (q_f, \varepsilon, i)$, where $q_f \in F$, $i \in B_S$.

It is not easy to define classes of balloon automata (like PDAs or SAs) since one needs to define valid families of functions $\mathsf{get}_{\mathsf{B_I}}$ and $\mathsf{upd}_{\mathsf{B_S}}$. One can see an example of PDAs definition in terms of BA in [7]. We suggest another approach for the definition of BA classes in Section 4. The approach simplifies the definitions since it is only needed to define a language of correct protocols to define an ADS.

We define a protocol as a sequence of triples $p_i = u_i \mathsf{q}_i \mathsf{r}_i$ of the query-word u_i, the query q_i and the response r_i on the query. Numerous extra conditions are listed in the following formal definition.

Definition 6. Let $\Gamma_{\mathsf{write}}, \Gamma_{\mathsf{query}}, \Gamma_{\mathsf{resp}}$ be finite disjoint alphabets such that $\Gamma_{\mathsf{query}} \neq \varnothing, \Gamma_{\mathsf{resp}} \neq \varnothing$. Let $\mathsf{valid} \subseteq \Gamma_{\mathsf{query}} \times \Gamma_{\mathsf{resp}}$ be a relation that provides the correspondence between queries and possible responses. A *protocol* is a word p such that $p = p_1 \cdots p_n$, where $n \geq 0$, $p_i = u_i \mathsf{q}_i \mathsf{r}_i$, $u_i \in \Gamma_{\mathsf{write}}^*$, $q_i \in \Gamma_{\mathsf{query}}$, $r_i \in \Gamma_{\mathsf{resp}}$, and $\mathsf{r}_i \in \mathsf{valid}(\mathsf{q}_i)$. We call a word p_i a *query block*. We say that a language $\mathsf{P} \subseteq (\Gamma_{\mathsf{write}}^* \Gamma_{\mathsf{query}} \Gamma_{\mathsf{resp}})^*$ is *a language of correct protocols* if the axioms (i-v) hold:

(i) $\varepsilon \in \mathsf{P}$;

(ii) $\forall p \in \mathsf{P} : p$ is a protocol;

(iii) $\forall p \in \mathsf{P} :$ if $p = p_1 p_2$ and p_1 is a protocol, then $p_1 \in \mathsf{P}$;

(iv) $\forall p \in \mathsf{P} \ \forall u \in \Gamma_{\mathsf{write}}^* \ \forall \mathsf{q} \in \Gamma_{\mathsf{query}} \ \exists \mathsf{r} \in \Gamma_{\mathsf{resp}} : pu\mathsf{q}\mathsf{r} \in \mathsf{P}$;

(v) $\forall pu\mathsf{q}\mathsf{r} \in \mathsf{P} :$ if $p' \in \mathsf{P}$ and $p' = pu\mathsf{q}\mathsf{r}'s$, then $\mathsf{r}' = \mathsf{r}$;

(vi) $\exists \mathsf{q} \in \Gamma_{\mathsf{query}}, \mathsf{r} \in \Gamma_{\mathsf{resp}} \ \forall p_1, p_2 \in \mathsf{P} : p_1 \mathsf{q}\mathsf{r} p_2 \in \mathsf{P}$.

Axiom (vi) does not hold in the general case, e.g., for SAs and counter automata without zero tests. It is needed to describe the connection of automata with an ADS with BAs in Section 4.

A language of correct protocols P generates the corresponding class of languages, the principal rational cone $\mathcal{T}(\mathsf{P})$. All examples of BAs languages classes in [7] can be presented as $\mathcal{T}(\mathsf{P})$. We provide here only two examples.

Example 7. It is well-known [2] that $\mathsf{CFL} = \mathcal{T}(D_2)$, where D_2 is the Dyck language with two types of parentheses. It is also well-known that a Dyck word is a protocol of the stack. We transform the language D_2 into a language of protocols $\mathsf{D_2\text{-}PROT}$ as follows.

We define the alphabets $\Gamma_{\mathsf{write}} = \varnothing$, $\Gamma_{\mathsf{query}} = \{\mathsf{push}_(, \mathsf{push}_[, \mathsf{pop}\}$, $\Gamma_{\mathsf{resp}} = \{(,),[,]\}$, $\mathsf{valid} = \{(\mathsf{push}_[,[),(\mathsf{push}_(,(),(\mathsf{pop},]),(\mathsf{pop},))\}$. To define correct protocols we use an FST T that erases all symbols from Γ_{query} of the input. So,

$$\mathsf{D_2\text{-}PROT} = \{p \mid T(p) \in D_2\}.$$

By the definition $D_2 \leq_{\mathrm{rat}} \mathsf{D_2\text{-}PROT}$, so we have that $\mathcal{T}(D_2) \subseteq \mathcal{T}(\mathsf{D_2\text{-}PROT})$. It is also easy to show that $\mathsf{D_2\text{-}PROT} \leq_{\mathrm{rat}} D_2$, so $\mathcal{T}(\mathsf{D_2\text{-}PROT}) = \mathcal{T}(D_2) = \mathsf{CFL}$.

Note that we set here $\Gamma_{\mathsf{write}} = \varnothing$ for the sake of simplicity. One can use another variant: $\Gamma_{\mathsf{write}} = \{(,[\}$, $\Gamma_{\mathsf{query}} = \{\mathsf{multipush}, \mathsf{pop}\}$, $\Gamma_{\mathsf{resp}} = \{\mathsf{pushed},),]\}$. $\square$

The following example is a starting point for the generalization presented in this paper.

Example 8. The data structure Set consists of the set $\mathbb{S}$ which is initially empty. Set supports the following operations: $\mathsf{in}(x) : \mathbb{S} \to \mathbb{S} \cup \{x\}$, $\mathsf{out}(x) : \mathbb{S} \to \mathbb{S} \setminus \{x\}$, $\mathsf{test}(x) : x \overset{?}{\in} \mathbb{S}$. We define the protocol language $\mathsf{SA\text{-}PROT}$ consistently with [13, 14], so the elements of alphabets below are individual symbols while they are words in [13, 14]. $\Gamma_{\mathsf{write}} = \{a, b\}$, $\Gamma_{\mathsf{query}} = \{\#\mathsf{in}, \#\mathsf{out}, \#\mathsf{test}\}$, $\Gamma_{\mathsf{resp}} = \{\#, +\#, -\#\}$, $\mathsf{valid} = \{(\#\mathsf{in}, \#), (\#\mathsf{out}, \#), (\#\mathsf{test}, +\#), (\#\mathsf{test}, -\#)\}$.

It was proved in [13] that $\mathscr{L}(1\mathrm{NSA}) = \mathcal{T}(\mathsf{SA\text{-}PROT})$. $\square$

Definition 9. Fix a language of correct protocols P. An *automaton equipped with auxiliary data structure* $\mathsf{B_P}$ (defined by P) is defined by a tuple

$$\langle S, \Sigma_{\triangleright\triangleleft}, \Gamma_{\mathsf{write}}, \Gamma_{\mathsf{query}}, \Gamma_{\mathsf{resp}}, F, s_0, \delta \rangle, \text{ where}$$

- S, $\Sigma_{\triangleright\triangleleft}$, F, s_0 are the same as in Definition 4, so as $\Sigma_{\triangleright\triangleleft,\varepsilon}$.

- $S = S_{\mathsf{write}} \cup S_{\mathsf{query}}$, $S_{\mathsf{write}} \cap S_{\mathsf{query}} = \varnothing$.

- $\mathsf{P} \subseteq (\Gamma_{\mathsf{write}}^* \Gamma_{\mathsf{query}} \Gamma_{\mathsf{resp}})^*$.

- δ is the transition relation defined as

  $$\delta \subseteq ([S_{\mathsf{write}} \times \Sigma_{\triangleright\triangleleft,\varepsilon}] \times [\Gamma_{\mathsf{write}}^* \times S]) \cup (S_{\mathsf{query}} \times \Gamma_{\mathsf{query}} \times \Gamma_{\mathsf{resp}} \times S_{\mathsf{write}}).$$

The automaton has a one-way write-only query tape. During the processing of the input, it writes query-words $u_i \in \Gamma^*_{\mathsf{write}}$ on the query tape, performs queries q_i, and receives responses r_i such that $u_1 \mathsf{q}_1 \mathsf{r}_1 \cdots u_n \mathsf{q}_n \mathsf{r}_n \in \mathsf{P}$. After each query, the query tape is erased.

A *configuration* of an ADS-automaton is a tuple

$$(s, v, u, p) \in S \times \Sigma^*_{\triangleright\triangleleft} \times \Gamma^*_{\mathsf{write}} \times (\Gamma^*_{\mathsf{write}}\Gamma_{\mathsf{query}}\Gamma_{\mathsf{resp}})^*,$$

where v is the unprocessed part of the input w, i.e., v is the suffix of $\triangleright w \triangleleft$, u is the content of the work tape, and p is the protocol of the automaton operating with the data structure. A move of an automaton is defined via the relation $\vdash$ on configurations which is defined as follows:

$$(s, av, u, p) \vdash (s', v, ux, p), \qquad \text{if } s \in S_{\mathsf{write}}, \; (s, a, x, s') \in \delta, \tag{9}$$

$$(s, v, u, p) \vdash (s', v, \varepsilon, pu\mathsf{q}\mathsf{r}), \qquad \text{if } s \in S_{\mathsf{query}}, \; (s, \mathsf{q}, \mathsf{r}, s') \in \delta, \; pu\mathsf{q}\mathsf{r} \in \mathsf{P}. \tag{10}$$

A configuration is *initial* if it has the form $(s_0, \triangleright w \triangleleft, \varepsilon, \varepsilon)$, a configuration is *accepting* if it has the form $(s_f, \varepsilon, \varepsilon, p)$, where $s_f \in F, p \in \mathsf{P}$. A word w is *accepted* by an automaton with ADS if $(s_0, \triangleright w \triangleleft, \varepsilon, \varepsilon) \vdash^* (s_f, \varepsilon, \varepsilon, p)$. An automaton is *deterministic* if for all configurations c, c_1, c_2 from $c \vdash c_1$ and $c \vdash c_2$ follows $c_1 = c_2$.

For the next two models, we provide the definitions on the implementation level only.

Definition 10. A DB$_{\mathsf{P}}$log-TM (NB$_{\mathsf{P}}$log-TM) is a deterministic (nondeterministic) log-TM M equipped with an ADS defined by the language of correct protocols P. I.e., M is equipped with an additional write-only one-way query tape that is used to write down a query word u_i and perform a query. After a query q_i is performed, the tape is erased and the finite state control of M receives the result r_i of the query q_i. The query results are consistent with P, i.e., $p_1 \cdots p_n \in \mathsf{P}$, $p_i = u_i \mathsf{q}_i \mathsf{r}_i$.

A *configuration* of B$_{\mathsf{P}}$log-TM is a triple (c, u, p) where c is the configuration of log-TM-part, u is the word written on the query tape, and $p \in \mathsf{P}$ is the protocol that is the result of all the performed queries. A B$_{\mathsf{P}}$log-TM M accepts a word w if $(c_0(w), \varepsilon, \varepsilon) \vdash^* (c_f, \varepsilon, p)$, where $c_0(w)$ is the initial configuration of the log-TM-part of M, c_f is the accepting configuration of log-TM-part of M, and $p \in \mathsf{P}$, the relation $\vdash$ corresponds to the M's moves.

Definition 11. Let F be an arbitrary formal language (filter). A DA$_F$log-TM (NA$_F$log-TM) is a deterministic (non-deterministic) log-space TM equipped with a read-only one-way infinite tape called *advice tape*. At the beginning of the computation, the advice tape contains a word $y\Lambda^\infty$, where $y \in F$ and Λ is a symbol that indicates empty cells.

A *configuration* of an A$_F$log-TM M is a pair (c, u) where c is the configuration of the log-TM-part of M, u is the unprocessed part of y. M accepts a word x if there exists $y \in F$ such that $(c_0(x), y) \vdash^* (c_f, \varepsilon)$, where $c_0(x)$ is the initial configuration of the log-TM-part of M, c_f is the accepting configuration of the log-TM-part of M.

8

An equivalent model to $\mathrm{DA}_F\log$-TMs appeared in [17] and its journal version [19] under the name "models of generalized nondeterminism (GNA)" and lead to the appearance of the $\mathrm{DRR}(F)$ problem. In this paper we repeat the steps of [17, 19] to establish the connection between $\mathrm{NA}_F\log$-TM and $\mathrm{NRR}(F)$ problem in Section 5 to prove one of the main results of the paper Eq. (2). We also show the equivalence between $\mathrm{DA}_F\log$-TMs and GNA. The difference is that in GNA it is allowed not to process the advice till the end of the word, so it was demanded for F to be a prefix-closed language in [17, 19].

3 Principal Rational Cones and the NRR-Problem

In this section, we provide the core of our technique. We prove that $\mathscr{L}(1\mathrm{NB}_\mathsf{P}\mathrm{A})$ is a principal rational cone generated by the language of correct protocols P, i.e., $\mathscr{L}(1\mathrm{NB}_\mathsf{P}\mathrm{A}) = \mathcal{T}(\mathsf{P})$; it is the first main result of the section. This fact yields structural results about the family $\mathscr{L}(1\mathrm{NB}_\mathsf{P}\mathrm{A})$, as well as the results on the complexity of the emptiness problem. We focus in this section on the connection between the non-emptiness problem and the $\mathrm{NRR}(\mathsf{P})$ problem. We prove that these problems are equivalent under log-space reductions, it is the second main result of the section. It leads us to the main results of the paper in Section 5. We provide in this section structural results that naturally arise in the proofs. Other structural results are discussed in Section 4 since their relation to [7].

Most of the results of this section directly generalize the results of [13, Section 3] (see the full journal version [15]). In most cases, to get a generalized result, one can substitute SA protocols (see Example 8) with general protocols as defined in Definition 6. So, our general approach comes from the generalization of the technique that was developed for SAs. One can also find in [15] more technically detailed proofs.

Lemma 12. *There exists a* $1\mathrm{NB}_\mathsf{P}\mathrm{A}$ M_P *recognizing* P.

Proof. Let us assume that M_P has on the input the word of the form $p_1 \cdots p_n$, where $p_i = u_i\mathsf{q}_i\mathsf{r}_i$ (since it is a regular condition). M_P writes a word u_i on the query tape, performs the query q_i and tests that the responce is r_i. If all tests are correct than p is accepted; otherwise, it is rejected. $\square$

Lemma 13. *For each language of correct protocols* $\mathsf{P} \subseteq (\Gamma^*_{write}\Gamma_{\mathsf{query}}\Gamma_{\mathsf{resp}})^*$ *there exists a language of correct protocols* $\mathsf{P}_{\{a,b\}} \subseteq (\{a,b\}^*\Gamma_{\mathsf{query}}\Gamma_{\mathsf{resp}})^*$, *provided* $(\Gamma_{\mathsf{query}} \cup \Gamma_{\mathsf{resp}}) \cap \{a,b\} = \varnothing$, *such that the following properties hold*

- $\mathscr{L}(1\mathrm{NB}_\mathsf{P}\mathrm{A}) = \mathscr{L}(1\mathrm{NB}_{\mathsf{P}_{\{a,b\}}}\mathrm{A})$,

- $\mathscr{L}(1\mathrm{DB}_\mathsf{P}\mathrm{A}) = \mathscr{L}(1\mathrm{DB}_{\mathsf{P}_{\{a,b\}}}\mathrm{A})$,

- *There exists a DFST* T *such that* $T(\mathsf{P}) = \mathsf{P}_{\{a,b\}}$ *and* T^{-1} *is a DFST,*

9

- *For each $1x\mathsf{B_P}$A M there exists an equivalent $1x\mathsf{B_{P_{\{a,b\}}}}$A $M_{\{a,b\}}$ such that $M_{\{a,b\}}$ is log-space constructible by M and vice versa ($x \in \{N, D\}$).*

Proof. Enumerate all letters from Γ_{write} and encode the i-th letter as ab^ia. Such encoding is computable by a DFST T and the inverse encoding is computed by T^{-1} that is a DFST as well. So, $\mathsf{P}_{\{a,b\}} = T(\mathsf{P})$ (we assume that T preserves letters from $\Gamma_{\mathsf{query}} \cup \Gamma_{\mathsf{resp}}$). Now we show that for each $1\mathrm{NB_P}$A M there exists an equivalent $1\mathrm{NB_{P_{\{a,b\}}}}$A $M_{\{a,b\}}$.

By our construction, $M_{\{a,b\}}$ simulates M, i.e., $M_{\{a,b\}}$ has states of the form (s, aux) where s is a state of M and aux is an auxiliary information needed for simulation; so for each configuration $((s, \mathsf{aux}), v, u', p')$ of $M_{\{a,b\}}$ there is a corresponding configuration (s, v, u, p) of M, where $p' = T(p)$ and u' is a prefix of $T(u)$. $M_{\{a,b\}}$ computes $T(u)$ by simulation of T via finite control and information of this simulation is stored in aux; the other part of finite control simulates M's transitions.

Each $1\mathrm{NB_{P_{\{a,b\}}}}$A $M_{\{a,b\}}$ can be simulated by a $1\mathrm{NB_P}$A M in the same way, one shall use T^{-1} instead of T. Note that described simulations preserve determinism and the transformations between M and $M_{\{a,b\}}$ are log-space computable. $\square$

Lemma 14. *Let T be an FST with the input alphabet Δ and the output alphabet Σ and M be a $1\mathrm{NB_P}$A over the alphabet Σ. There exists a $1\mathrm{NB_P}$A $M' = M \circ T$ recognizing the language $T^{-1}(L(M))$. If T is a DFST and M is a $1\mathrm{DB_P}$A then M' is a $1\mathrm{DB_P}$A as well.*

Proof. The simulation is performed in a straight-forward way. M' guesses an image $w \in T(w')$ of the input word w' such that $w \in L(M)$ if $T(w') \cap L(M) \neq \varnothing$, computes w by simulation of T and simulates M on the input w. M' has configurations of the form $((s, \mathsf{aux}), v', u, p)$ that correspond to configurations (s, v, u, p) of M. As in the proof of Lemma 13, the aux information is used to simulate T via finite state control. The construction preserves determinism of $1\mathrm{DB_P}$A if T is a DFST. $\square$

Lemma 15. *Let M be a $1\mathrm{NB_P}$A. There exists an FST T_M such that $w \in L(M)$ iff $T_M(w) \cap \mathsf{P} \neq \varnothing$. Moreover, $p \in T_M(w)$ iff M has a run on w such that $(s_0, w, \varepsilon, \varepsilon) \vdash^* (s_f, \varepsilon, \varepsilon, p)$.*

We denote by $s \xrightarrow[x]{a} s'$ the move of T_M from the state s to the state s' on which it reads a from the input tape and writes x on the output tape.

Proof. One can construct T_M by M as follows. T_M has the same states as M (and the same initial state and set of accepting states). In the case of move (9), T_M has the move $s \xrightarrow[x]{a} s'$, and in the case of move (10), T_M has moves $s \xrightarrow[\mathsf{qr'}]{\varepsilon} s'$ for all $\mathsf{r'}$ such that $(s, \mathsf{q}, \mathsf{r'}, s') \in \delta_M$.

Assertion $p \in T_M(w) \cap \mathsf{P}$ implies that M has the corresponding run by axiom (v) in Definition 6. So, if $T_M(w)$ contains a correct protocol p then M has the run $(s_0, w, \varepsilon, \varepsilon) \vdash^* (s_f, \varepsilon, \varepsilon, p)$. The implication in the other direction directly follows from the construction of T_M. $\square$

Definition 16. An FST T_M from Lemma 15 called *extractor* (of protocols).

Theorem 17. $\mathscr{L}(1\text{NB}_\mathsf{P}\text{A}) = \mathcal{T}(\mathsf{P})$.

Proof. Lemma 15 implies that for each $1\text{NB}_\mathsf{P}\text{A}$ M there exists an extractor T_M such that $L(M) = T_M^{-1}(\mathsf{P})$, and by Lemma 3 there exists FST $T = T_M^{-1}$ such that $L(M) = T(\mathsf{P})$, so $L(M) \leq_{\mathrm{rat}} \mathsf{P}$ and therefore $\mathscr{L}(1\text{NB}_\mathsf{P}\text{A}) \subseteq \mathcal{T}(\mathsf{P})$.

The inclusion $\mathcal{T}(\mathsf{P}) \subseteq \mathscr{L}(1\text{NB}_\mathsf{P}\text{A})$ follows from Lemmata 12 and 14: for each $L = T'(\mathsf{P})$ we take an FST $T = T'^{-1}$ and apply the lemmata. $\square$

Theorem 18. $\overline{\text{E-1NB}_\mathsf{P}\text{A}} \leq_{\log} \text{NRR}(\mathsf{P}) \leq_{\log} \overline{\text{E-1NB}_\mathsf{P}\text{A}}$.

Proof. Let M be the input of the non-emptiness problem $\overline{\text{E-1NB}_\mathsf{P}\text{A}}$ and T_M be the corresponding extractor. By Lemma 15, $w \in L(M) \iff T_M(w) \cap \mathsf{P} \neq \varnothing$. So, $L(M) \neq \varnothing \iff T_M(\Sigma^*) \cap \mathsf{P} \neq \varnothing$. Construct an NFA $\mathcal{A}$ recognizing $T_M(\Sigma^*)$ by Lemma 2 in log space. So,

$$L(M) \neq \varnothing \iff L(\mathcal{A}) \cap \mathsf{P} \neq \varnothing \overset{\text{Def. 1}}{\iff} \mathcal{A} \in \text{NRR}(\mathsf{P}).$$

So we have proved $\overline{\text{E-1NB}_\mathsf{P}\text{A}} \leq_{\log} \text{NRR}(\mathsf{P})$.

The reduction $\text{NRR}(\mathsf{P}) \leq_{\log} \overline{\text{E-1NB}_\mathsf{P}\text{A}}$ follows from Lemmata 12 and 14. We construct by $\mathcal{A}$ on the input of $\text{NRR}(\mathsf{P})$ the automaton $M = M_\mathsf{P} \circ T$, where $xTy \iff (x = y) \wedge (x \in L(\mathcal{A}))$. $\square$

Theorem 19. $\text{M-1DB}_\mathsf{P}\text{A} \leq_{\log} \text{DRR}(\mathsf{P})$.

Proof. We construct a DFA $\mathcal{A}$ on the input of $\text{DRR}(\mathsf{P})$ by (w, M) on the input of $\text{M-1DB}_\mathsf{P}\text{A}$ via a log-space transducer. The idea is that $\mathcal{A}$ simulates M's run on the input w and checks the correctness of the protocol by reading the input word, that is a protocol $p \in \mathsf{P}$. The protocol p is accepted iff p is the protocol of M on processing of w and M accepts w.

A state of $\mathcal{A}$ is a tuple (s, i, aux) where $s \in S_M$, i is the index of the letter w_i over the M's head and aux is the auxiliary information needed for the simulation. To simulate a transition of M, the following actions are performed by $\mathcal{A}$. If M writes a word v (a subword of the future query word) to the query tape, $\mathcal{A}$ stores v in the finite memory (a part of aux component of its states) and checks whether the unprocessed part of its input begins with v (if not, the input word p is rejected). If M performs a query q, $\mathcal{A}$ verifies that the unprocessed part of its input begins with qr and performs the transition that M does after receiving r as a response. $\mathcal{A}$ accepts the input p if it was not rejected during the simulation, $i = |w| + 1$ (i.e., M's head is over $\triangleleft$) and M is in accepting state.

It follows from the construction that $\mathcal{A}$ accepts p iff p is the protocol of M processing the input w. Note that this protocol is unique since M is a $1\text{DB}_\mathsf{P}\text{A}$. Also since M is $1\text{DB}_\mathsf{P}\text{A}$, $\mathcal{A}$ is log-space constructible. Finally, $L(\mathcal{A}) \cap \mathsf{P} \neq \varnothing \iff w \in L(M)$, so $\text{M-1DB}_\mathsf{P}\text{A} \leq_{\log} \text{DRR}(\mathsf{P})$. $\square$

4 Connection with Balloon Automata

We provide a high-level description of classes $\mathscr{M}_B$ of BAs. The definition in a more formal style could be found in [7].

Definition 20. A subset of BAs $\mathscr{M}_B$ is a *class of BAs* if the following conditions hold.

(I) $\mathscr{M}_B$ contains all automata with $\mathsf{get}_{\mathsf{B_I}}$ such that, for each state s, $\mathsf{upd}_{\mathsf{B_S}}(s,i)$ is either i for all i or $\mathsf{upd}_{\mathsf{B_S}}(s,i) = j$ for all i and some constant j.

(II) If $\mathcal{A}, \mathcal{B} \in \mathscr{M}_B$, $\mathsf{upd}_{\mathsf{B_S}}^{\mathcal{A}}, \mathsf{get}_{\mathsf{B_I}}^{\mathcal{B}}, \mathsf{upd}_{\mathsf{B_S}}^{\mathcal{A}}, \mathsf{upd}_{\mathsf{B_S}}^{\mathcal{B}}$ are the corresponding functions of $\mathcal{A}$ and $\mathcal{B}$, then $\mathscr{M}_B$ includes each automaton $\mathcal{C}$ such that $\mathsf{get}_{\mathsf{B_I}}^{\mathcal{C}}$ and $\mathsf{upd}_{\mathsf{B_S}}^{\mathcal{C}}$ are the functions that are obtained from the functions of $\mathcal{A}$ and $\mathcal{B}$ via finite control, i.e., for each state $s \in S_{\mathcal{C}}$ $\mathsf{get}_{\mathsf{B_I}}^{\mathcal{C}}(s,i)$ equals to either $\mathsf{get}_{\mathsf{B_I}}^{\mathcal{A}}(s,i)$ or $\mathsf{get}_{\mathsf{B_I}}^{\mathcal{B}}(s,i)$ for all i, for each i,j if $\mathsf{upd}_{\mathsf{B_S}}^{\mathcal{C}}(i) \neq \mathsf{upd}_{\mathsf{B_S}}^{\mathcal{C}}(j)$ then either $\mathsf{upd}_{\mathsf{B_S}}^{\mathcal{A}}(i) \neq \mathsf{upd}_{\mathsf{B_S}}^{\mathcal{A}}(j)$ or $\mathsf{upd}_{\mathsf{B_S}}^{\mathcal{B}}(i) \neq \mathsf{upd}_{\mathsf{B_S}}^{\mathcal{A}}(j)$.

Property (I) implies that $\mathscr{M}_B$ contains automata that can reset any state i of the balloon to the initial state 1 (or to some fixed state j as well). Together with Property (II) it implies that the balloon has a reset operation that sets the balloon's state to the initial state. This property does not hold for SAs, so there is no direct correspondence between classes of languages of BAs and automata with an ADS in the general case.

Theorem 21. *For each ADS $\mathsf{B_P}$ there exists a balloon B and a subset of BAs $\mathscr{M}_B$ such that the corresponding classes of languages coincide, i.e. $\mathscr{L}(xy\mathsf{B_P}\mathrm{A}) = \mathscr{L}(xy\mathrm{BA})$ and Property (II) holds. If P has the reset operation* (vi)*, Property (I) also holds, i.e $\mathscr{M}_B$ is a class (in terms of Definition 20).*

Proof. We begin with the construction of the balloon B and the bijection from $xy\mathsf{B_P}\mathrm{A}$ to $xy\mathrm{BA}$ such that $xy\mathrm{BA}$ form the set $\mathscr{M}_B$ satisfying Property (II). A state of the balloon B is an integer that is the encoding of pairs of words (p,u), where p is the current protocol, i.e., the protocol of all previous operations before the upcoming move, and u is the word on the query-tape. We enumerate all $p \in \mathsf{P}$ and all $u \in \Sigma^*$ and use the standard enumeration of pairs of integers.

Firstly, we define functions $\mathsf{upd}_{\mathsf{B_S}}$ and $\mathsf{get}_{\mathsf{B_I}}$ for the BA M_{P}^B recognizing P. Recall that for any language of correct protocols P there exists 1DB$_\mathsf{P}$A M_P recognizing P by Lemma 12. The function $\mathsf{upd}_{\mathsf{B_S}}$ simulates write operations and queries: it just updates the ballon's state according to the encoding. The function $\mathsf{get}_{\mathsf{B_I}} : \mathbb{Z}_{>0} \to \Gamma_{\mathsf{resp}} \cup \{\perp\}$ returns responses or $\perp$ if there were no query. For an arbitrary $xy\mathsf{B_P}\mathrm{A}$ M the $xy\mathrm{BA}$ M^B is constructed as follows. The function $\mathsf{get}_{\mathsf{B_I}}^M$ is the same as for M_P^B for any M^B. The function $\mathsf{upd}_{\mathsf{B_S}}^M$ is a modification of the function for M_P^B according to the finite state control of M. We define the class $\mathscr{F}(\mathsf{upd}_{\mathsf{B_S}})$ more formally below to show that Property (II) holds.

As the result, the states of the ballon B just encode the part of $xy\mathsf{B_P}A$ that describes the data structures, and $\mathsf{upd_{B_S}}$ and $\mathsf{get_{B_I}}$ simulate the work with the data structure defined by P. So we provided the bijection between $xy\mathsf{B_P}A$ and $xy\mathsf{B}A$.

We move to a formal definition of the class $\mathscr{F}(\mathsf{upd_{B_S}})$. At first, we define $\mathscr{F}_1(\mathsf{upd_{B_S}})$ satisfying Property (II). Assume that M_P writes at most one letter to the query tape per move and it also has a state s_ε in which it neither writes nor performs query. So, $\mathsf{upd_{B_S}}(s_\varepsilon, i) = i$ for all i (hereinafter in M_P^B). We mark a state s as s_a if M_P writes a on the output tape and mark a state s as s_q if M performs query q. From definition follows that $\mathsf{upd_{B_S}}(s_m, i) = \mathsf{upd_{B_S}}(s'_m, i)$ for all states s and s' marked by the same mark (a, ε or q). So we define a function $\mathsf{upd_{B_S}^P} : (\Gamma_{\mathsf{write},\varepsilon} \cup \Gamma_{\mathsf{query}}) \times \mathbb{Z}_{>0} \to \mathbb{Z}_{>0}$ so that $\mathsf{upd_{B_S}^P}(m, i) = \mathsf{upd_{B_S}}(s_m, i)$. So for any $xy\mathsf{B}A$ M the function $\mathsf{upd_{B_S}^M}$ defined as follows. The states of M are marked by symbols from $\Gamma_{\mathsf{write},\varepsilon} \cup \Gamma_{\mathsf{query}}$ and $\mathsf{upd_{B_S}^M}(s_m, i) = \mathsf{upd_{B_S}^P}(m, i)$. It is easy to see that from our definition of $\mathscr{F}_1(\mathsf{upd_{B_S}})$ follows the bijection between $xy\mathsf{B_P}A$ and $xy\mathsf{B}A$ and Property (II) holds as well.

If P has the reset operation (vi), then we shall modify the interpretation of B since $\mathscr{F}(\mathsf{upd_{B_S}})$ does not satisfy our definition anymore. Firstly we describe the interpretation of $\mathsf{upd_{B_S}}$ functions for $xy\mathsf{B}A$ M^B from Property (I). If $\mathsf{upd_{B_S}}(s, i) = j$, i and j encode pairs (p_i, u_i) and (p_j, u_j) respectively and (p_j, u_j) is not obtained from (p_i, u_i) by a single move of M_P^B, then we interpretate the state change $i \to j$ as follows. The corresponding to M^B $xy\mathsf{B_P}A$ M performs the reset operation and then performs sequence of queries that move the configuration from $(\varepsilon, \varepsilon)$ to (p_j, u_j) during ε moves. Note that by the definition of Property (I) M^B has finitely many j's in the range of $\mathsf{upd_{B_S}}$ so M is well-defined. Denote $\mathsf{upd_{B_S}}$ functions for automata from Property (I) as $\mathscr{F}_I(\mathsf{upd_{B_S}})$. So $\mathscr{F}(\mathsf{upd_{B_S}})$ is a closure of $\mathscr{F}_I(\mathsf{upd_{B_S}})$ and $\mathscr{F}_1(\mathsf{upd_{B_S}})$ in terms of Property (II). From Property (II) and our construction of $xy\mathsf{B_P}A$'s for Property (I) follows the construction of $xy\mathsf{B_P}A$ for any of $xy\mathsf{B}A$ from the closure in terms of Property (II). So, we have proved the second part of the theorem. $\quad\square$

So all the results from [7] that do not rely on (I) hold for B_P-automata. We are most interested in (1) and its complexity analogue (3). Many structural results from [7] follow from the fact that $\mathscr{L}(1\mathrm{NB_P}A)$ is a principal cone (Theorem 17), namely, closure of $\mathscr{L}(1\mathrm{NB_P}A)$ over union and rational transductions[1]. We shall also mention the closure over gsm inverse mappings proved in [7] for all $xy\mathsf{B}A$ that implies the same closure for all $xy\mathsf{B_P}A$.

Lemma 22. *If B_P contains the reset operation then $\mathscr{L}(1\mathrm{NB_P}A)$ is closed over concatenation and iteration.*

Proof. We construct $1\mathrm{NB_P}A$'s M^2 and M^* by $1\mathrm{NB_P}A$ M recognizing $L(M) \cdot L(M)$ and $L(M)^*$ respectively in a straight-forward way. M^2 simulates M and nondeterministically guess the split of the input uv such that $u, v \in L(M)$ at the

---

[1] Intersection and quotient with regular languages, gsm forward mapping are the partial cases of rational transductions.

end of the u. If after processing of u, M is in an accepting state, M^2 performs the reset operation and simulates M on v. M^* guesses the split of the input into $u_1 u_2 \ldots u_m, u_i \in L(M)$ and acts in the similar way. □

The standard technique from [2] implies the following lemma.

Lemma 23. *If* $\mathsf{P}\#\mathsf{P} \leq_{\mathrm{rat}} \mathsf{P}$, $\# \notin \Gamma$, *then* $\mathscr{L}(1\mathrm{NB_PA})$ *is closed over concatenation. If* $(\mathsf{P}\#)^* \leq_{\mathrm{rat}} \mathsf{P}$, $\# \notin \Gamma$, *then* $\mathscr{L}(1\mathrm{NB_PA})$ *is closed over iteration.*

Proof idea. The construction is similar to the one from the proof of Lemma 22. FSTs T_{L^2} and T_{L^*} uses marks $\#$ to split the input and simulate an FST T_L corresponding to the language L, i.e., $L = T_L(\mathsf{P})$. □

Remark 24. We leave open the question of the reduction in the opposite direction. I.e., does for each class of BAs exist a language of correct protocols P such that BAs recognize the same class of languages as $\mathsf{B_P}$ automata? The essence of the problem is as follows. If axioms (I-II) for the class of BAs are satisfied, does it imply that there exists a "universal" BA M_U such that $\mathscr{L}(1\mathrm{NBA}) = \mathcal{T}(L(M_U))$ and for each $M \in 1\mathrm{NBA}$ there exists an FST T such that $L(M) = L(M_U \circ T)$?

5 RR Problems and log-TM Models

5.1 $\mathrm{A}_F$log-TM models

In [17, 19] it was shown that $\mathrm{DRR}(\mathrm{Pref}(F\Lambda^*))$ is a complete problem in the class of languages recognizable by GNA with advices from F (this model corresponds to DA_Flog-TM, $\mathrm{Pref}(L)$ is the set of all prefixes of L). We prove that $\mathrm{DRR}(\mathrm{Pref}(F\Lambda^*))$ and $\mathrm{DRR}(F)$ are complete problems in the class $\mathscr{L}(\mathrm{DA}_F\log\text{-}\mathrm{TM})$. We begin with the proof of similar result for $\mathrm{NRR}(F)$ and $\mathscr{L}(\mathrm{NA}_F\log\text{-}\mathrm{TM})$. We introduce the following auxiliary lemma for the sake of the proof.

Lemma 25 ([12]). $F_1 \leq_{\mathrm{rat}} F_2 \Rightarrow \mathrm{NRR}(F_1) \leq_{\log} \mathrm{NRR}(F_2)$.

Lemma 26. $\mathscr{L}(\mathrm{NA}_F\log\text{-}\mathrm{TM}) \leq_{\log} \mathrm{NRR}(F)$.

Proof. An $\mathrm{NA}_F\log\text{-}\mathrm{TM}$ M takes on the input a word x and also takes $y \in F$ on the advice tape. $x \in L(M) \iff \exists y \in F : M(x, y) = 1$, where $M(x, y) = 1$ if M halts in an accepting state, $M(x, y) = 0$ if M halts in a rejecting state.

A surface-configuration of a $\mathrm{NA}_F\log\text{-}\mathrm{TM}$ is a tuple (q, mem, i, j) of the state q, log-space memory configuration mem and the positions i of the head on the input tape and j of the head on the advice (one-way) tape. The tuples (q, mem, i) are the states of finite automata $\mathcal{A}$ on the input of $\mathrm{NRR}(F)$ problem constructed by M and x. The initial state is $(q_0, \varnothing, 0)$, where q_0 is the initial state of the $\mathrm{NA}_F\log\text{-}\mathrm{TM}$, accepting states are states of the form (q_f, mem, i) where q_f is an accepting state of the $\mathrm{NA}_F\log\text{-}\mathrm{TM}$. The transitions between the states are determined by letters of y, i.e. $(q, \mathsf{mem}, i, j) \vdash (q', \mathsf{mem}', i', j')$, $i' \in \{i-1, i, i+1\}$, $j' \in \{j, j+1\}$ if $(q', \mathsf{mem}', i') \in \delta_{\mathcal{A}}((q, \mathsf{mem}, i, j), y_j)$. The

list of the $\mathcal{A}$'s transitions $\delta_{\mathcal{A}}$ is log-space computable so as the set of the $\mathcal{A}$'s states as well.

Without loss of generality, we assume that M always processes y till the end, i.e. till mits Λ on the advice tape. So, by the construction of $\mathcal{A}$, we obtain

$$x \in L(M) \iff \exists y \in F : M(x,y) \iff \exists y \in F, k \geq 0 : y\Lambda^k \in L(\mathcal{A}) \iff$$

$$\iff \mathcal{A} \in \mathrm{NRR}(F\Lambda^*) \stackrel{\text{Lemma 25}}{\iff} \mathcal{A}' \in \mathrm{NRR}(F),$$

where $\mathcal{A}'$ is constructed by $\mathcal{A}$ due to the reduction in Lemma 25. Since M is fixed, we get that

$$x \stackrel{?}{\in} L(M) \leq_{\log} \mathcal{A} \stackrel{?}{\in} \mathrm{NRR}(F\Lambda^*) \leq_{\log} \mathcal{A}' \stackrel{?}{\in} \mathrm{NRR}(F),$$

and we obtain that $\mathscr{L}(\mathrm{NA}_F\log\text{-}\mathrm{TM}) \leq_{\log} \mathrm{NRR}(F)$ by the transitivity of the log-space reductions. $\qquad\square$

Remark 27. Note that $F \sim_{\mathrm{rat}} F\Lambda^* \sim_{\mathrm{rat}} \mathrm{Pref}(F\Lambda^*)$ so the NRR problems for these filters are equivalent (up to $\leq_{\log}$ reductions). The equivalence holds since nondeterministic FST's can have several images for the same word, particularly write many Λ's at the end of the word. It does not hold for deterministic FSTs, so $F\Lambda^* \not\sim_{\mathrm{drat}} F$. So to obtain the corresponding lemma for $\mathrm{DA}_F\log\text{-}\mathrm{TM}$ we need to modify the proof of Lemma 26.

Lemma 28. $\mathscr{L}(\mathrm{DA}_F\log\text{-}\mathrm{TM}) \leq_{\log} \mathrm{DRR}(F)$.

Proof. We repeat the steps of the proof of Lemma 26. Note that $\mathcal{A}$ is a DFA, since M is a deterministic machine. To construct $\mathcal{A}'$ we construct an auxiliary DFA $\mathcal{A}''$ by $\mathcal{A}$ as follows. $\mathcal{A}''$ has the states $(q, \not\Lambda)$ and (q, Λ) for each state q of $\mathcal{A}$. The auxiliary bit of a state indicates whether $\mathcal{A}''$ met Λ. So for each transition $q \stackrel{\Lambda}{\to} p$ of $\mathcal{A}$ there are two corresponding transitions $(q, b) \stackrel{\Lambda}{\to} (p, \Lambda)$, $b \in \{\Lambda, \not\Lambda\}$. For states (q, Λ) $\mathcal{A}'$ has only transitions by Λ. For the transitions $q \stackrel{a}{\to} p$, $a \neq \Lambda$, $\mathcal{A}''$ has transitions $(q, \not\Lambda) \stackrel{a}{\to} (p, \not\Lambda)$. A state (q, b) is an accepting if q is an accepting state of $\mathcal{A}$.

Now we construct $\mathcal{A}'$. It is obtained from $\mathcal{A}$ by removing all Λ-transitions. Each $\mathcal{A}$'s accepting state is an accepting for $\mathcal{A}'$ and $\mathcal{A}'$ also has accepting states determined as follows. If $(q, \not\Lambda)$ has Λ-path to an accepting state (q_f, Λ) in $\mathcal{A}''$, then q is an accepting state in $\mathcal{A}'$. It is easy to see that $y \in L(\mathcal{A}') \iff \exists k \geq 0 \ y\Lambda^k \in L(\mathcal{A})$ and $\mathcal{A}''$ is log-space computable from $\mathcal{A}$ and $\mathcal{A}'$ is log-space computable from $\mathcal{A}$ and $\mathcal{A}''$. $\qquad\square$

Lemma 29. $\mathrm{NRR}(F) \leq_{\log} \mathscr{L}(\mathrm{NA}_F\log\text{-}\mathrm{TM})$ *and* $\mathrm{DRR}(F) \leq_{\log} \mathscr{L}(\mathrm{DA}_F\log\text{-}\mathrm{TM})$. *Moreover, there exist an* $\mathrm{NA}_F\log\text{-}\mathrm{TM}$ M_{NRR} *that recognizes the problem* $\mathrm{NRR}(F)$ *and* M_{DRR} *that recognizes* $\mathrm{DRR}(F)$ *as well.*

Proof. The proof is straightforward. M_{NRR} gets on the input an NFA $\mathcal{A}$ and verifies whether $\mathcal{A}$ accepts the word $y \in F$ written on the advice tape. If $y \in L(\mathcal{A})$, M_{NRR} nondeterministically guesses the $\mathcal{A}$'s run on y. So, by Definition 11, $\mathcal{A} \in L(M_{\mathrm{NRR}})$ iff $\exists y \in F : y \in L(\mathcal{A}) \iff \mathcal{A} \in \mathrm{NRR}(F)$.

The construction for M_{DRR} is the same. $\qquad\square$

Theorem 30.

$$\mathscr{L}(\mathrm{NA}_F\text{log-TM}) = \{L \mid L \leq_{\log} \mathrm{NRR}(F)\},$$
$$\mathscr{L}(\mathrm{DA}_F\text{log-TM}) = \{L \mid L \leq_{\log} \mathrm{DRR}(F)\}.$$

Proof. By the definition of $\leq_{\log}$, $L \leq_{\log} \mathrm{NRR}(F)$ iff there exists a log-TM transducer T that maps the input x of the problem $x \overset{?}{\in} L$ to the input $T(x)$ of the problem $\mathrm{NRR}(F)$. We construct an NA_Flog-TM M recognizing L via the composition of T and NA_Flog-TM M_{NRR} from Lemma 29.

So $\{L \mid L \leq_{\log} \mathrm{NRR}(F)\} \subseteq \mathscr{L}(\mathrm{NA}_F\text{log-TM})$; the opposite inclusion follows from Lemma 26. We repeat the same arguments for the deterministic case and apply Lemma 28 for the opposite inclusion. $\qquad\blacksquare$

5.2   B$_\mathsf{P}$log-TM models

In Section 3 we exploited the following idea. In the case of a nondeterministic model, performing queries one by one and proceeding the computation depending on the queries' results computationally equivalent to guessing all the queries results and verifying whether all the results were correct in the end (by testing whether obtained protocol was correct). In fact, this idea works even in the case of a deterministic model in Theorem 19. Now we exploit it again for log-TM-based models.

Lemma 31.

M-NA$_\mathsf{P}$log-TM $\sim_{\log}$ M-NB$_\mathsf{P}$log-TM *and* M-DB$_\mathsf{P}$log-TM $\leq_{\log}$ M-DA$_\mathsf{P}$log-TM.

We provide only the proof idea since the proof follows our general technique that we have applied above a lot.

Proof idea. Let M_A be a NA$_\mathsf{P}$log-TM, M_B be a NB$_\mathsf{P}$log-TM, and x be an input word. Since both kinds of log-TMs are nondeterministic, M_A can guess and verify M_B's successful run on x provided that M_B's protocol is written on the advice tape; M_B can guess $y \in \mathsf{P}$ and a successful run of M_A on (x, y), and verify it: the transitions on configurations are simulated on log space and the fact $y \in \mathsf{P}$ is verified by performing subsequently the queries from the sequence y.

The case of deterministic models is similar to Theorem 19. M_A just simulates M_B and tests whether the query words on the advice tape, queries an the results are the same as M_B has during processing of the input. $\qquad\blacksquare$

Combining all together, we obtain the main theorem of the section.

Theorem 32.

$$\mathrm{NRR}(\mathsf{P}) \sim_{\log} \mathscr{L}(\mathrm{NB}_\mathsf{P}\text{log-TM}) = \{L \mid L \leq_{\log} \mathrm{NRR}(\mathsf{P})\}, \tag{11}$$
$$\mathscr{L}(\mathrm{DB}_\mathsf{P}\text{log-TM}) \leq_{\log} \mathrm{DRR}(\mathsf{P}), \tag{12}$$
$$\mathscr{L}(\mathrm{DB}_\mathsf{P}\text{log-TM}) \subseteq \{L \mid L \leq_{\log} \mathrm{DRR}(\mathsf{P})\}. \tag{13}$$

Proof. We begin with the proof of (11). By Lemma 31

$$\text{M-NB}_\text{P}\text{log-TM} \sim_{\log} \text{M-NA}_\text{P}\text{log-TM}. \tag{14}$$

By Lemmata 26 and 29

$$\mathscr{L}(\text{NA}_\text{P}\text{log-TM}) \sim_{\log} \text{NRR}(\text{P}). \tag{15}$$

Eqs. (14) and (15) imply

$$\mathscr{L}(\text{NB}_\text{P}\text{log-TM}) \sim_{\log} \text{NRR}(\text{P}). \tag{16}$$

By Theorem 30

$$\mathscr{L}(\text{NA}_\text{P}\text{log-TM}) = \{L \mid L \leq_{\log} \text{NRR}(\text{P})\}. \tag{17}$$

Eqs. (14) and (17) imply

$$\mathscr{L}(\text{NB}_\text{P}\text{log-TM}) = \{L \mid L \leq_{\log} \text{NRR}(\text{P})\}. \tag{18}$$

Eqs. (16) and (18) form (11).

Now we move to the proof of (12) and (13). By Lemma 28

$$\mathscr{L}(\text{DA}_\text{P}\text{log-TM}) \leq_{\log} \text{DRR}(\text{P}). \tag{19}$$

By Lemma 31

$$\text{M-DB}_\text{P}\text{log-TM} \leq_{\log} \text{M-DA}_\text{P}\text{log-TM}. \tag{20}$$

Eqs. (19) and (20) imply (12). By Theorem 30

$$\mathscr{L}(\text{DA}_\text{P}\text{log-TM}) = \{L \mid L \leq_{\log} \text{DRR}(\text{P})\}. \tag{21}$$

Eqs. (20) and (21) imply (13). $\square$

6 Applications

In this section we prove the applications (5-8) described in Section 1.1.

Theorem 33. *Assertions (5-8) hold.*

Proof. SA-PROT was defined in Example 8. It was proved in [14] that the problems $\overline{\text{E-1NSA}} \sim_{\log} \text{NRR}(\text{SA-PROT})$ are **PSPACE**-complete. So we obtain (6) by applying Theorem 32. We prove (5) in the same way by combining the facts $\text{D}_2\text{-PROT} \sim_{\text{rat}} D_2$ (Example 7) and $\text{NRR}(D_2)$ is P-complete [12], and apply Lemma 25 and Theorem 32.

To prove (7-8) we use facts about the filters $\text{Per}_k = \{(w\#)^k \mid w \in \Sigma_k\}$, where Σ_k is a k-letter alphabet. The problem $\text{NRR}(\text{Per}_1)$ is **NP**-complete and $\text{NRR}(\text{Per}_k)$, $k > 1$, is **PSPACE**-complete [1, 18]. We construct set-protocols based on these languages as follows. Let $\Gamma_\text{write} = \Sigma_k$, $\Gamma_\text{query} = \{\text{in}, \text{test}\}$, $\Gamma_\text{resp} = \{+, -\}$. The response to the in-query is positive only for the first query,

test-queries are the same as in Example 8. We denote the language of correct protocols with $\Gamma_{\mathsf{write}} = \Sigma_k$ as $\mathsf{S_{1,k}PROT}$. It is easy to see that $\mathrm{Per}_k \leq_{\mathrm{rat}} \mathsf{S_{1,k}PROT}$: an FST T maps words of the form $w\mathsf{in} + w\mathsf{test} + \cdots w\mathsf{test}+$ to $w\#w\#\cdots w\#$ by replacing queries and responses by $\#$; the sequence of queries with responses $\mathsf{in}+, \mathsf{test}+, \ldots, \mathsf{test}+$ is verifiable via a finite state control (the inputs with invalid sequence are rejected by the FST), so $\mathrm{NRR}(\mathrm{Per}_k) \leq_{\log} \mathrm{NRR}(\mathsf{S_{1,k}PROT})$ by Lemma 25.

Now we prove that $\mathsf{S_{1,k}PROT} \leq_{\mathrm{rat}} \mathrm{Per}_k$. The FST T takes on the input a word $(w\#)^n$ and acts as follows. While translating a block $w\#$ to the output, it has the following options: (i) change at least one letter, (ii) erase at least one letter and maybe change others, (iii) add at least one letter and maybe change others, (iv) do not change w. Until T has not write in, it replaces $\#$ by $\mathsf{test}-$ in the cases (i-iii), and either by $\mathsf{test}-$ or by $\mathsf{in}+$ in the case (iv). After T wrote $\mathsf{in}+$, it replaces $\#$ by $\mathsf{test}+$ in the case (iv) and either by $\mathsf{test}-$ or by $\mathsf{in}-$ in the cases (i-iii). It is easy to see that $T((w\#)^n)$ consists of all correct protocols with either w first in-query or without in-queries at all, and exactly n queries. So $T(\mathrm{Per}_k) = \mathsf{S_{1,k}PROT}$ and assertions (7-8) follows from Lemma 25 and Theorem 32. □

7 On computational complexity of correct protocol languages

Theorem 18 essentially says that the computational complexity of the non-emptiness problem for ADS-automata is the same as the computational complexity of the NRR problem for the corresponding correct protocols languages. It can be used to answer the question about the range of complexities of the non-emptiness problems for ADS-automata. It extends the known results about the complexity of RR problems [20]. It appears that these complexities are almost universal. It means that for any nonempty language X there exists a language of correct protocols P such that X is reducible to $\overline{\text{E-1NB}_\mathsf{P}\text{A}}$. The reductions in the two directions differ. In one direction it is a log-space m-reduction. In the other, we present the proof only for Turing reductions in polynomial time.

Theorem 34. *For any nonempty language $X \subseteq \{0,1\}^*$ there exists a language of correct protocols P such that*

$$X \leq_{\log} \overline{\text{E-1NB}_\mathsf{P}\text{A}} \overset{\text{Th. 18}}{\sim_{\log}} \mathrm{NRR}(\mathsf{P}) \leq^{\mathrm{P}}_{\mathrm{T}} X.$$

In the proof of Theorem 34 we use the language of protocols defined as follows. Set $\Gamma_{\mathsf{write}} = \{0,1\}$, $\Gamma_{\mathsf{query}} = \{\#, r\}$, $\Gamma_{\mathsf{resp}} = \{+, -, r\}$. The relation valid is defined as follows: $\mathsf{valid}(\#) = \{+, -\}$, $\mathsf{valid}(r) = \{r\}$. The language of correct protocols P consists of protocols such that, for every query block $u_i\mathsf{q}_i\mathsf{r}_i$, either $\mathsf{q}_i = \mathsf{r}_i = r$ and $u_i = \varepsilon$, or $\mathsf{q}_i = \#$, $\mathsf{r}_i = +$ and $u_i \in L$, or $\mathsf{q}_i = \#$, $\mathsf{r}_i = -$ and $u_i \notin L$. Here $L \subseteq \{0,1\}^*$ is a language depending on X in the statement of the theorem.

The exact choice of L is complicated. So we start with the presentation of basic ideas behind the proof of Theorem 34. We encode binary words using a log-space computable injective map sq: $\{0,1\}^* \to \{0,1\}^*$ such that $\mathrm{sq}(X) \subseteq L$ and $\mathrm{sq}(\bar{X}) \subseteq \bar{L}$. It suffices for the first reduction in the theorem, $X \leq_{\log} \mathrm{NRR}(\mathsf{P})$, since the protocol $\mathrm{sq}(x)\#+$ is correct iff $x \in X$.

For the second reduction, i.e., $\mathrm{NRR}(\mathsf{P}) \leq_{\mathrm{T}}^{\mathrm{P}} X$, we need much more requirements. Let $\mathcal{A}$ be an input automaton for $\mathrm{NRR}(\mathsf{P})$ and S be its state set. We are going to decide $L(\mathcal{A}) \cap \mathsf{P} \neq \varnothing$ in polynomial time using oracle calls of the oracle X. For this purpose we reduce the question $L(\mathcal{A}) \cap \mathsf{P} \neq \varnothing$ to the question $R \neq \varnothing$ for some regular language $R \in \{y, n, \#, +, -, r\}^*$. By definition, $w \in R$ if there exists an accepting run of $\mathcal{A}$ that processes a correct protocol p such that p is obtained from w by substitutions of letters y and n with words of L and $\bar{L}$ respectively (different words may be used for different occurrences of the letters). To check the correctness of the run processing the protocol, we need to compute, for all pairs $s', s'' \in S$, all possible transitions from s' to s'' by processing words from L only as well as all possible transitions from s' to s'' by processing words from $\bar{L}$ only.

Thus, the main part of the reduction consists of solving NRR problems $L(\mathcal{A}_{s's''}) \cap L \neq \varnothing$ and $L(\mathcal{A}_{s's''}) \cap \bar{L} \neq \varnothing$ for all pairs $s', s'' \in S$. Here $\mathcal{A}_{s's''}$ are auxiliary automata. The states and the transitions of $\mathcal{A}_{s's''}$ coincide with the states and the transitions of $\mathcal{A}$. The initial state of $\mathcal{A}_{s's''}$ is s' and the only accepting state is s''.

Note that $L(\mathcal{A}_{s's''})$ may be infinite and it causes the first difficulty: one need to consider arbitrary long words in the protocol language. To avoid this difficulty we require that any infinite regular language intersects both L and $\bar{L}$. Therefore $L(\mathcal{A}_{s's''}) \cap L \neq \varnothing$ and $L(\mathcal{A}_{s's''}) \cap \bar{L} \neq \varnothing$ if $L(\mathcal{A}_{s's''})$ is infinite.

If $L(\mathcal{A}_{s's''})$ is finite, it means that the transition graph is DAG (after removing states that are not reachable and coreachable in $\mathcal{A}_{s's''}$). The second difficulty: it might be exponentially many words in $L(\mathcal{A}_{s's''})$. Again, to overcome it, we pose specific requirements on L to guarantee that that verifying $L(\mathcal{A}_{s's''}) \cap L \neq \varnothing$ and $L(\mathcal{A}_{s's''}) \cap \bar{L} \neq \varnothing$ requires polynomially many oracle calls of the oracle X.

Now we provide formal arguments for the above plan of proof. We encode binary words by the injective map

$$\mathrm{sq}\colon x \mapsto \beta(x)11\beta(x)11, \qquad (22)$$

where $\beta\colon \{0,1\}^* \to \{0,1\}^*$ is the morphism defined on the symbols as $\beta(0) = 01$, $\beta(1) = 10$.

For an NFA $\mathcal{A}$ with the state set S we define a relation

$$s' \overset{u}{\hookrightarrow} s''$$

that holds if $\mathcal{A}$ can reach s'' on processing u starting from the state s'.

Now we list the requirements on the language L.

1. As it mentioned before, $\mathrm{sq}(X) \subseteq L$ and $\mathrm{sq}(\bar{X}) \subseteq \bar{L}$.

2. There exists a language $W \subseteq \{0,1\}^*$ such that both $W \cap L$ and $W \cap \bar{L}$ are recognized in polynomial time, and, for any NFA $\mathcal{A}$ over the alphabet $\{0,1\}$ and any pair of its states s_1, s_2, either $L(\mathcal{A}_{s_1 s_2})$ is finite, or there exist $w_1 \in L \cap W$, $w_2 \in \bar{L} \cap W$ such that $w_1 \in L(\mathcal{A}_{s_1 s_2})$ and $w_2 \in L(\mathcal{A}_{s_1 s_2})$.

3. The language W is sparse: $|W \cap \{0,1\}^{\leq n}| = \mathrm{poly}(n)$. Moreover, the lists of words in $L \cap W \cap \{0,1\}^{\leq n}$ and, respectively, in $\bar{L} \cap W \cap \{0,1\}^{\leq n}$ can be generated in polynomial time.

4. If $|u| = |v|$, $u \neq v$, and $uv \notin W$ then $uv \in L$ iff $u \prec v$, where $\prec$ is the lexicographical order.

5. If $|w|$ is odd and $w \notin W$ then $w \in L$. If $w = xx$ and $w \notin \mathrm{sq}(\{0,1\}^*) \cup W$ then $w \in L$.

The sets of L-transitions and $\bar{L}$-transitions are defined as follows:

$$\delta_{\mathcal{A}}^{L}(s) = \left\{ s' \in S : \exists u \ s \xrightarrow{u} s', \ u \in L \right\}, \quad \delta_{\mathcal{A}}^{\bar{L}}(s) = \left\{ s' \in S : \exists u \ s \xrightarrow{u} s', \ u \in \bar{L} \right\}.$$

The main part of the proof of Theorem 34 is the following lemma.

Lemma 35. *Let L be a language satisfying Requirements 1–5. Then there exists a polynomial time algorithm with the oracle X that outputs the sets $\delta_{\mathcal{A}}^{L}(s)$, $\delta_{\mathcal{A}}^{\bar{L}}(s)$, where $\mathcal{A}$ is an input of $\mathrm{NRR}(\mathsf{P})$ and s is its state.*

Before presenting the algorithm from the lemma, we analyze the most difficult case separately.

Proposition 36. *Let L be a language satisfying Requirements 1–5 and $\mathcal{A}$ be an NFA with the initial state s_0 and the unique accepting state s_f such that $L(\mathcal{A})$ is finite, $L(\mathcal{A}) \cap W = \varnothing$, and each word in $L(\mathcal{A})$ has an even length. Then conditions $s_f \in \delta_{\mathcal{A}}^{L}(s_0)$ and $s_f \in \delta_{\mathcal{A}}^{\bar{L}}(s_0)$ can be verified by a polynomial time algorithm with the oracle X.*

Proof. By solving the reachability problem, one can detect the set of reachable and coreachable states of $\mathcal{A}$. All other states can be deleted without affecting $L(\mathcal{A})$. From now on, we assume that all the states $s \in S$ are reachable and coreachable.

Since $L(\mathcal{A})$ is finite, the transition graph of $\mathcal{A}$ is a DAG as well as all its subgraphs. For each pair of states $s_1, s_2 \in Q$, let $\ell(s_1, s_2)$ be the set

$$\left\{ k : \exists u \ s_1 \xrightarrow{u} s_2 \text{ and } |u| = k \right\}.$$

Using topological sorting, one can construct all the sets $\ell(s_1, s_2)$ in polynomial time by the backward induction based on the relation

$$\ell(s_1, s_2) = \bigcup_{s \in N(s_1)} \left(1 + \ell(s, s_2) \right),$$

where $N(s_1)$ is the set of states that are reachable from s in one move and $1 + X = \{y : y = 1 + x, \ x \in X\}$.

For a positive integer ℓ and a state s of $\mathcal{A}$, we define

$$\text{Left}(s, \ell) = \big\{ u : s_0 \overset{u}{\hookrightarrow} s, \ |u| = \ell \big\}, \quad \text{Right}(s, \ell) = \big\{ u : s \overset{u}{\hookrightarrow} s_f, \ |u| = \ell \big\}.$$

We order the sets $\text{Left}(s, \ell)$ and $\text{Right}(s, \ell)$ in the lexicographical order. Let $\min_0(s, \ell)$ be the minimal word in $\text{Left}(s, \ell)$, and $\max_0(s, \ell)$ be the maximal word in $\text{Left}(s, \ell)$, and $\min_1(s, \ell)$ be the minimal word in $\text{Right}(s, \ell)$, and $\max_1(s, \ell)$ be the maximal word in $\text{Right}(s, \ell)$.

There exists an inductive procedure that computes $\min_0(s, \ell)$, $\max_0(s, \ell)$, $\min_1(s, \ell)$, and $\max_1(s, \ell)$ in polynomial time. The procedure also verifies the conditions $\text{Left}(s, \ell) \neq \varnothing$, $\text{Right}(s, \ell) \neq \varnothing$. We describe computation of $\min_0(s, \ell)$, the other words are computed similarly.

Suppose that u is the prefix of $\min_0(s, \ell)$ of the length $0 \leq k < \ell$ (if $k = \ell$, then the procedure returns u and stops). Let $S_k = \{s' : s_0 \overset{u}{\hookrightarrow} s'\}$. This set can be computed in polynomial time. If there exists $s' \in S_k$ and $s'' \in S$ such that $s'' \in \delta_{\mathcal{A}}(s', 0)$ and $\ell - k - 1 \in \ell(s'', s)$, then $u0$ is a prefix of $\min_0(s, \ell)$ of the length $k + 1$. Otherwise, if there exists $s' \in S_k$ and $s'' \in S$ such that $s'' \in \delta_{\mathcal{A}}(s', 1)$ and $\ell - k - 1 \in \ell(s'', s)$, then $u1$ is a prefix of $\min_0(s, \ell)$. If both conditions are not satisfied, then $\text{Left}(s) = \varnothing$.

According to Requirement 4 on L, if there exist a state s and an integer ℓ such that $\min_0(s, \ell) \prec \max_1(s, \ell)$, then $s_f \in \delta_{\mathcal{A}}^L(s_0)$. Otherwise, $\max_1(s, \ell) \preceq \min_0(s, \ell)$ for all s, ℓ. Since $L(\mathcal{A}) \cap W = \varnothing$, in this case $s_f \in \delta_{\mathcal{A}}^L(s_0)$ if and only if there exist a state s and an integer ℓ such that $\min_0(s, \ell) = \max_1(s, \ell)$ and either $\min_0(s, \ell) = \beta(x)11$ and $x \in X$ or $\min_0(s, \ell) \max_1(s, \ell) \notin \text{sq}(\{0, 1\}^*)$ due to Requirements 1, 5. The condition $x \in X$ can be verified by an oracle call, the rest of conditions can be verified in polynomial time.

A similar check can be done for the condition $s_f \in \delta_{\mathcal{A}}^{\bar{L}}(s_0)$. It is equivalent to the following: there exist a state s and an integer ℓ such that either $\min_1(s, \ell) \prec \max_0(s, \ell)$, or $\min_1(s, \ell) = \max_0(s, \ell) = \beta(x)11$ and $x \notin X$. $\qquad \blacksquare$

Proof of Lemma 35. The algorithm maintains the sets $S^+ \subseteq S$, $S^- \subseteq S$. Initially, $S^+ = S^- = \varnothing$. We will prove that at the end $S^+ = \delta_{\mathcal{A}}^L(s)$, $S^- = \delta_{\mathcal{A}}^{\bar{L}}(s)$. The algorithm analyzes states $s' \in S$ one by one and adds s' to the sets S^+, S^- according to the following rules.

In the first step, the algorithm decides whether $L(\mathcal{A}_{ss'})$ is infinite. It can be done in polynomial time. If the answer is 'yes', then the algorithm adds s' to both sets S^+, S^- and continues with the next state. The correctness of this step is guaranteed by Requirement 2.

If the answer at the first step is 'no', the lengths of words in $L(\mathcal{A}_{ss'})$ do not exceed $|S|$ (otherwise, there exists a run of $\mathcal{A}$ from s to s' containing a cycle, which implies that $L(\mathcal{A}_{ss'})$ is infinite). In the second step, the algorithm checks whether $L(\mathcal{A}_{ss'}) \cap L \cap W \neq \varnothing$ and, respectively, whether $L(\mathcal{A}_{ss'}) \cap \bar{L} \cap W \neq \varnothing$. It can be done in polynomial time due to Requirement 3. If the first condition holds, then the algorithm adds s' to S^+. If the second condition holds, then the algorithm adds s' to S^-.

In the third step, the algorithm constructs an NFA $\mathcal{A}'$ recognizing $L(\mathcal{A}_{ss'}) \setminus W$. Let P be the set of prefixes of all words in $W \cap \{0,1\}^{\leq |S|}$. Due to Requirement 3, $|P| = \mathrm{poly}(|S|)$ and P can be constructed in polynomial time. The states of $\mathcal{A}'$ are the pairs $(\tilde{s}, p)$, $\tilde{s} \in S$, $p \in P \cup \{\bot\}$. The set of transitions $\delta_{\mathcal{A}'}((s, p), a)$ consists of pairs $(\tilde{s}, p')$ such that $\tilde{s} \in \delta_{\mathcal{A}}(s, a)$ and $p' = pa \in P$, and pairs $(\tilde{s}, \bot)$ such that $\tilde{s} \in \delta_{\mathcal{A}}(s, a)$ and $p' = pa \notin P$. The set of transitions $\delta_{\mathcal{A}'}((s, \bot), a)$ consists of pairs $(\tilde{s}, \bot)$ such that $\tilde{s} \in \delta_{\mathcal{A}}(s, a)$. The initial state is (s, ε). Accepting states are pairs $(s', \bot)$ and (s', p), where $p \notin W$. This definition implies that $\mathcal{A}'$ can be constructed in polynomial time. To prove the correctness of the construction, note that processing a word $w \in W \cap \{0,1\}^{\leq |S|}$ from (s, ε) finishes at the state (s', w) which is not accepting. For a word $w \in L(\mathcal{A}_{ss'}) \setminus W$ there exists an accepting run of $\mathcal{A}_{ss'}$. The corresponding run of $\mathcal{A}'$ finishes at a state of the form (s', w) or $(s', \bot)$. Thus, w is accepted by A'.

In the fourth step, the algorithm checks whether $L(\mathcal{A}_{ss'}) \setminus W$ contains a word of odd length. It can be done in polynomial time since words of odd length form a regular language and the intersection of this language with $L(\mathcal{A}_{ss'}) \setminus W$ is recognized by an NFA with $2|S'|$ states, where S' is the state set of $\mathcal{A}'$. If the answer is 'yes', then the algorithm adds q' to S^+. The correctness of this step is guaranteed by Requirement 5.

In the fifth step, the algorithm constructs an NFA $\mathcal{A}''$ that accepts exactly the words of even length from $L(\mathcal{A}_{ss'}) \setminus W$, apply to it the algorithm of Proposition 36, updates S^+, S^- if necessary, and continues with the next state.

It is clear from the above remarks that at the end $S^+ \subseteq \delta_{\mathcal{A}}^L(s)$, $S^- \subseteq \delta_{\mathcal{A}}^{\bar{L}}(s)$.

Suppose that $s' \in \delta_{\mathcal{A}}^L(s)$. If $L(\mathcal{A}_{ss'})$ is infinite then s' is added to S^+ at the first step. If $L(\mathcal{A}_{ss'}) \cap W \cap L \cap \{0,1\}^{\leq |Q|} \neq \varnothing$, then s' is added at the second step. Otherwise, $(L(\mathcal{A}_{ss'}) \cap L) \setminus W$ should be non-empty. If there are words of odd length in $L(\mathcal{A}_{ss'}) \setminus W$, then s' is added at the fourth step. And, finally, if $L(\mathcal{A}_{ss'}) \setminus W$ consists of words of even length only, s' is added at the fifth step due to Proposition 36. Therefore, $S^+ = \delta_{\mathcal{A}}^L(s)$ at the end of the algorithm.

Suppose that $s' \in \delta_{\mathcal{A}}^{\bar{L}}(s)$. If $L(\mathcal{A}_{ss'})$ is infinite then s' is added to S^- at the first step. If $L(\mathcal{A}_{ss'}) \cap W \cap \bar{L} \cap \{0,1\}^{\leq |S|} \neq \varnothing$, then s' is added at the second step. Otherwise, $(L(\mathcal{A}_{ss'}) \cap \bar{L}) \setminus W \neq \varnothing$ and s' is added at the fifth step due to Proposition 36. Therefore, $S^- = \delta_{\mathcal{A}}^{\bar{L}}(s)$ at the end of the algorithm. $\qquad\square$

Now we prove that Requirements 1–5 on L are compatible.

Lemma 37. *There exists L satisfying Requirements 1–5.*

Proof. We define W at first. For each triple a, b, c of non-empty binary words there are two words in W in the form $ab^{2r(a,b,c)}c$ and $ab^{2q(a,b,c)+1}c$ and for each $w \in W$ there exists a unique triple a, b, c such that either $w = ab^{2r(a,b,c)}c$ or $w = ab^{2q(a,b,c)+1}c$. The definition of W is inductive. Order all triples x, y, z of non-empty binary words with respect to the length of xyz and order the triples with the same length of xyz with respect to the lexicographical order on the triples of binary words (binary words are also ordered lexicographically).

Assume that for all (x, y, z) less than (a, b, c) we have defined $s(x, y, z)$ and $t(x, y, z)$ properly. Thus the set $W' \subseteq W$ has been already defined. The total

number of (x, y, z) less than (a, b, c) does not exceed $\binom{|abc|-1}{2} \cdot (2^{|abc|} - 1)$. Thus, there are at most $2\binom{|abc|-1}{2} \cdot (2^{|abc|} - 1)$ words from W' having lengths in the range $[2^{3|abc|+3}, 2^{3|abc|+4} - 1]$. So, there exist at least

$$\frac{2^{3|abc|+3}}{2\binom{|abc|-1}{2} \cdot (2^{|abc|} - 1)} - 1 > 2|abc| > 2|b|$$

consecutive integers i in the range such that no word in W' has the length i. At least $|b|$ of them are even and at least $|b|$ of them are odd. It guarantees that the sets

$$E = \{j : ab^{2j}c \notin W', \ 2^{3|abc|+3} \leq |ab^{2j}c| < 2^{3|abc|+4}\} \text{ and}$$
$$O = \{j : ab^{2j+1}c \notin W', \ 2^{3|abc|+3} \leq |ab^{2j+1}c| < 2^{3|abc|+4}\}$$

are non-empty. Set $r(a, b, c)$ be the minimal j in E and $q(a, b, c)$ be the minimal j in O.

To define L, we require that the words from W in the form $ab^{2q(a,b,c)+1}c$ are in L, while the words from W in the form $ab^{2r(a,b,c)}c$ are in $\bar{L}$. Note that it implies Requirement 2, since any infinite regular language contains all words in the form $ab^k c$, $k > 0$, for some a, b, c.

By construction, for each $w \in W$ the length of defining triple a, b, c is logarithmic in the length of w. Thus $|W \cap \{0, 1\}^{\leq n}| = \mathrm{poly}(n)$. To construct the list of words in $L \cap W \cap \{0, 1\}^{\leq n}$ and the list of words in $\bar{L} \cap W \cap \{0, 1\}^{\leq n}$ one need to perform only polynomial number of steps of the defining procedure and each step can be performed in polynomial time. Therefore, Requirement 3 is satisfied.

The rest of L is defined to satisfy Requirements 1, 4, and 5. Note that $\mathrm{sq}(\{0, 1\}^*) \cap W = \varnothing$, since, for each $x \in \{0, 1\}^*$, $\mathrm{sq}(x)$ does not contain proper periodic subwords of length greater $|\mathrm{sq}(x)|/2$ but each word in W do contain such words. It means that the construction of W does not conflict with Requirement 1. $\qquad\square$

Now Theorem 34 follows from Lemma 35 and Lemma 37.

Proof of Theorem 34. Choose L as in the proof of Lemma 37. The reduction $X \leq_{\log} \mathrm{NRR}(\mathsf{P})$ is given by a map $x \mapsto \mathrm{sq}(x)\#+$. It is clear that the map is computed in logarithmic space. The correctness of reduction follows from Requirement 1.

Now we describe the second reduction, $\mathrm{NRR}(\mathsf{P}) \leq_{\mathrm{T}}^{\mathrm{P}} X$. Let $\mathcal{A}$ be an input NFA for $\mathrm{NRR}(\mathsf{P})$. The reducing algorithm computes all sets $\delta_{\mathcal{A}}^L(s)$, $\delta_{\mathcal{A}}^{\bar{L}}(s)$ using Lemma 35.

Let $\mathcal{B}$ be an NFA with the same state set as $\mathcal{A}$. The alphabet of B is $\{y, n, \#, +, -, r\}$. Transitions $\delta_{\mathcal{B}}(s, a)$ coincide with transitions $\delta_{\mathcal{A}}(s, a)$ for $a \in \{\#, +, -, r\}$. For the rest of transitions, $\delta_{\mathcal{B}}(s, y) = \delta_{\mathcal{A}}^L(s)$ and $\delta_{\mathcal{B}}(s, n) = \delta_{\mathcal{A}}^{\bar{L}}(s)$. The initial state and the accepting states of $\mathcal{B}$ and of $\mathcal{A}$ coincide.

Let $R = L(\mathcal{B}) \cap \left(y\#+ \mid n\#- \mid rr\right)^*$. Then $L(\mathcal{A}) \cap \mathsf{P} \neq \varnothing$ iff $R \neq \varnothing$. The latter condition is verified in polynomial time since R is regular. $\qquad\square$

Acknowledgments

References

[1] T. Anderson, J. Loftus, N. Rampersad, N. Santean, and J. Shallit. Special Issue: LATA 2008 Detecting palindromes, patterns and borders in regular languages. *Information and Computation*, 207(11):1096–1118, 2009.

[2] J. Berstel. *Transductions and context-free languages*. Ed. Teubner, 1979.

[3] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *International Conference on Concurrency Theory*, pages 135–150. Springer, 1997.

[4] D. Chistikov, R. Majumdar, and P. Schepper. Subcubic certificates for cfl reachability. *Proc. ACM Program. Lang.*, 6(POPL), jan 2022.

[5] M. Daley, M. Eramian, and I. Mcquillan. The bag automaton: A model of nondeterministic storage. *J. Autom. Lang. Comb.*, 13(3):185–206, June 2008.

[6] D. Dolev, S. Even, and R. Karp. On the security of ping-pong protocols. *Information and Control*, 55(1):57–68, 1982.

[7] J. E. Hopcroft and J. D. Ullman. An approach to a unified theory of automata. In *SWAT 1967*, pages 140–147, 1967.

[8] M. Kutrib, A. Malcher, and M. Wendlandt. Set automata. *International Journal of Foundations of Computer Science*, 27(02):187–214, 2016.

[9] K.-J. Lange and K. Reinhardt. Set automata. In *Combinatorics, Complexity and Logic; Proceedings of the DMTCS'96*, pages 321–329. Springer, 1996.

[10] D. Melski and T. Reps. Interconvertibility of a class of set constraints and context-free-language reachability. *Theoretical Computer Science*, 248(1-2):29–98, 2000.

[11] T. Reps, S. Horwitz, and M. Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '95, page 49–61, New York, NY, USA, 1995. Association for Computing Machinery.

[12] A. Rubtsov and M. Vyalyi. Regular realizability problems and context-free languages. In *DCFS 2015*, volume 9118 of *LNCS*, pages 256–267. Springer, 2015.

[13] A. Rubtsov and M. Vyalyi. On computational complexity of set automata. In *DLT 2017*, pages 332–344, Cham, 2017. Springer International Publishing.

[14] A. Rubtsov and M. Vyalyi. On emptiness and membership problems for set automata. In *CSR 2018*, pages 295–307, Cham, 2018. Springer International Publishing.

[15] A. Rubtsov and M. Vyalyi. On computational complexity of set automata. *Information and Computation*, to appear.

[16] M. Sipser. *Introduction to the theory of computation*. Cengage Learning, 2013.

[17] M. Vyalyi. On models of a nondeterministic computation. In *CSR 2009*, pages 334–345, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[18] M. Vyalyi. On the models of nondeterminizm for two-way automata (in Russian). *Proceedings of VIII international conference «Discrete models in the theory of control systems».*, pages 54–60, 2009.

[19] M. N. Vyalyi. On regular realizability problems. *Probl. Inf. Transm.*, 47(4):342–352, 2011.

[20] M. N. Vyalyi. On expressive power of regular realizability problems. *Probl. Inf. Transm.*, 49(3):276–291, 2013.

[21] P. Wolf. On the decidability of finding a positive ilp-instance in a regular set of ilp-instances. In *DCFS 2019*, volume 11612 of *LNCS*, pages 272–284. Springer, 2019.

[22] P. Wolf and H. Fernau. Regular intersection emptiness of graph problems: Finding a needle in a haystack of graphs with the help of automata. *CoRR*, abs/2003.05826, 2020.

[23] M. Yannakakis. Graph-theoretic methods in database theory. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '90, page 230–242, New York, NY, USA, 1990. Association for Computing Machinery.