On the Complexity of Universality for Partially Ordered NFAs*

Markus Krötzsch[1], Tomáš Masopust[2], and Michaël Thomazo[3]

1 **Institute of Theoretical Computer Science and Center of Advancing Electronics Dresden (cfaed), TU Dresden, Germany**
markus.kroetzsch@tu-dresden.de
2 **Institute of Theoretical Computer Science and Center of Advancing Electronics Dresden (cfaed), TU Dresden, Germany**
tomas.masopust@tu-dresden.de
3 **Inria, Gif-sur-Yvette, France**
michael.thomazo@inria.fr

Abstract

Partially ordered nondeterminsitic finite automata (poNFAs) are NFAs whose transition relation induces a partial order on states, i.e., for which cycles occur only in the form of self-loops on a single state. A poNFA is universal if it accepts all words over its input alphabet. Deciding universality is PSPACE-complete for poNFAs, and we show that this remains true even when restricting to a fixed alphabet. This is nontrivial since standard encodings of alphabet symbols in, e.g., binary can turn self-loops into longer cycles. A lower coNP-complete complexity bound can be obtained if we require that all self-loops in the poNFA are deterministic, in the sense that the symbol read in the loop cannot occur in any other transition from that state. We find that such restricted poNFAs (rpoNFAs) characterise the class of $\mathcal{R}$-trivial languages, and we establish the complexity of deciding if the language of an NFA is $\mathcal{R}$-trivial. Nevertheless, the limitation to fixed alphabets turns out to be essential even in the restricted case: deciding universality of rpoNFAs with unbounded alphabets is PSPACE-complete. Our results also prove the complexity of the inclusion and equivalence problems, since universality provides the lower bound, while the upper bound is mostly known or proved in the paper.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.3 Formal Languages

Keywords and phrases Automata, Nondeterminism, Partial order, Universality

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.61

1 Introduction

The universality problem asks if a given automaton (or grammar) accepts (or generates) all possible words over its alphabet. In typical cases, deciding universality is more difficult than deciding the word problem. For example, universality is undecidable for context-free grammars [3] and PSPACE-complete for nondeterministic finite automata (NFAs) [25]. The study of universality (and its complement, emptiness) has a long tradition in formal languages, with many applications across computer science, e.g., in the context of formal knowledge representation and database theory [10, 33, 4]. Recent studies investigate the problem for specific types of automata or grammars, e.g., for prefixes or factors of regular languages [28].

* This work was supported by the German Research Foundation (DFG) within the Collaborative Research Center SFB 912 (HAEC) and in Emmy Noether grant KR 4381/1-1 (DIAMOND).

© Markus Krötzsch, Tomáš Masopust, and Michaël Thomazo;
licensed under Creative Commons License CC-BY

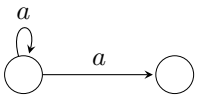

Figure 1 Forbidden pattern of rpoNFAs.

Table 1 Complexity of deciding universality.

	Unary alphabet		Fixed alphabet		Arbitrary alphabet	
DFA	in P		in P		in P	
rpoNFA	in P	(Thm. 4)	coNP-comp.	(Cor. 16)	PSpace-comp.	(Thm. 19)
poNFA	in P	(Thm. 4)	PSpace-comp.	(Thm. 3)	PSpace-comp.	[1]
NFA	coNP-comp.	[34]	PSpace-comp.	[1]	PSpace-comp.	[1]

In this paper, we are interested in the universality problem for *partially ordered NFAs* (poNFAs) and special cases thereof. An NFA is partially ordered if its transition relation induces a partial order on states: the only cycles that are allowed are self-loops on a single state. Partially ordered NFAs define a natural class of languages that has been shown to coincide with level $\frac{3}{2}$ of the Straubing-Thérien hierarchy [31] and with Alphabetical Pattern Constraint (APC) languages, a subclass of regular languages effectively closed under permutation rewriting [6]. Deciding if an automaton recognises an APC language (and hence whether it can be recognised by a poNFA) is PSpace-complete for NFAs and NL-complete for DFAs [6].

Restricting to partially ordered deterministic finite automata (poDFAs), we can capture further classes of interest: two-way poDFAs characterise languages whose syntactic monoid belongs to the variety **DA** [31], introduced by Schützenberger [30]; poDFAs characterise $\mathcal{R}$-trivial languages [9]; and confluent poDFAs characterise level 1 of the Straubing-Thérien hierarchy, also known as $\mathcal{J}$-trivial languages or piecewise testable languages [32]. Other relevant classes of partially ordered automata include partially ordered Büchi automata [20] and two-way poDFAs with look-around [21].

A first result on the complexity of universality for poNFAs is readily obtained. It is well known that universality of regular expressions is PSpace-complete [1, Lemma 10.2], and it is easy to verify that the regular expressions used in the proof can be expressed in poNFAs:

▶ **Corollary 1** (Lemma 10.2 [1]). *The universality problem for poNFAs is* PSpace*-complete.*

A closer look at the proof reveals that the underlying encoding requires an alphabet of size linear in the input: PSpace-hardness is not established for alphabets of bounded size. Usually, one could simply encode alphabet symbols σ by sequences $\sigma_1 \cdots \sigma_n$ of symbols from a smaller alphabet, say $\{0, 1\}$. However, doing this requires self-loops $q \xrightarrow{\sigma} q$ to be replaced by nontrivial cycles $q \xrightarrow{\sigma_1} \ldots \xrightarrow{\sigma_n} q$, which are not permitted in poNFAs.

We settle this open problem by showing that PSpace-hardness is retained even for binary alphabets.

This negative result leads us to ask if there is a natural subclass of poNFAs for which universality does become simpler. We consider *restricted* poNFAs (rpoNFAs), which require self-loops to be deterministic in the sense that the automaton contains no transition as in Figure 1. Large parts of the former hardness proof hinge on transitions of this form, which, speaking intuitively, allow the automaton to navigate to an arbitrary position in the input (using the loop) and, thereafter, continue checking an arbitrary pattern. Indeed, we find that the universality becomes coNP-complete for rpoNFAs with a fixed alphabet.

However, this reduction of complexity is not preserved for unrestricted alphabets. We use a novel construction of rpoNFAs that characterise certain exponentially long words to show that universality is PSPACE-complete even for rpoNFAs if the alphabet may grow polynomially. Our complexity results are summarised in Table 1.

As a by-product, we show that rpoNFAs provide another characterisation of $\mathcal{R}$-trivial languages introduced and studied by Brzozowski and Fich [9], and we establish the complexity of detecting $\mathcal{R}$-triviality and k-$\mathcal{R}$-triviality for rpoNFAs.

The complexity of the inclusion and equivalence problems of regular expressions of several special forms has been investigated by Martens et al. [22]. Some of them are expressible by poNFAs. The results have been established for alphabets of unbounded size. We point out here that our results also apply to the inclusion and equivalence problems. The complexity of universality provides the lower bound. The upper bound for the case of PSPACE-complete problems then follows from the complexity for general NFAs, whereas for the coNP-complete problems it is shown in Theorem 15. Hence the results of Table 1 also hold for inclusion and equivalence.

Finally, we mention the relationship to deterministic regular expressions (DRE) [7], which are of interest in schema languages for XML data – Document Type Definition (DTD) and XML Schema Definition (XSD) – since the World Wide Web Consortium standards require that the regular expressions in their specification are deterministic. The important question is then whether a regular expression or an NFA is expressible as a DRE. This problem has been shown to be PSPACE-complete [12]. Since the non-DRE-definable language $(a+b)^*b(a+b)$ [7] can be expressed by a poNFA, the problem is nontrivial for poNFAs. Its complexity (PSPACE-complete), however, follows from the existing results, namely from the proof given in [5] showing PSPACE-hardness of DRE-definability for regular expressions, since the regular expression constructed there can be expressed as a poNFA. On the other hand, all rpoNFA languages are DRE-definable by the automata characterization presented in [7].

Proofs omitted in the text can be found in the corresponding technical report [19].

2 Preliminaries and Definitions

We assume that the reader is familiar with automata theory [1]. The cardinality of a set A is denoted by $|A|$ and the power set of A by 2^A. An *alphabet* Σ is a finite nonempty set. A *word* over Σ is any element of the free monoid Σ^*, the *empty word* is denoted by ε. A *language* over Σ is a subset of Σ^*. For a language L over Σ, let $\overline{L} = \Sigma^* \setminus L$ denote its complement.

A *subword* of w is a word u such that $w = w_1uw_2$, for some words w_1, w_2; u is a *prefix* of w if $w_1 = \varepsilon$ and it is a *suffix* of w if $w_2 = \varepsilon$.

A *nondeterministic finite automaton* (NFA) is a quintuple $\mathcal{A} = (Q, \Sigma, \cdot, I, F)$, where Q is a finite nonempty set of states, Σ is an input alphabet, $I \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of accepting states, and $\cdot : Q \times \Sigma \to 2^Q$ is the transition function that can be extended to the domain $2^Q \times \Sigma^*$ by induction. The language *accepted* by $\mathcal{A}$ is the set $L(\mathcal{A}) = \{w \in \Sigma^* \mid I \cdot w \cap F \neq \emptyset\}$. We often omit $\cdot$ and write simply Iw instead. The NFA $\mathcal{A}$ is *complete* if for every state q and every letter a in Σ, the set $q \cdot a$ is nonempty. It is *deterministic* (DFA) if $|I| = 1$ and $|q \cdot a| = 1$ for every state q in Q and every letter a in Σ.

A *path* π from a state q_0 to a state q_n under a word $a_1a_2\cdots a_n$, for some $n \geq 0$, is a sequence of states and input symbols $q_0a_1q_1a_2 \ldots q_{n-1}a_nq_n$ such that $q_{i+1} \in q_i \cdot a_{i+1}$, for $i = 0, 1, \ldots, n-1$. Path π is *accepting* if $q_0 \in I$ and $q_n \in F$. A path is *simple* if all the states are pairwise distinct.

A *deterministic Turing machine* (DTM) is a tuple $M = (Q, T, I, \delta, \sqcup, q_o, q_f)$, where Q is the finite state set, T is the tape alphabet, $I \subseteq T$ is the input alphabet, $\sqcup \in T \setminus I$ is the

blank symbol, q_o is the initial state, q_f is the accepting state, and δ is the transition function mapping $Q \times T$ to $Q \times T \times \{L, R, S\}$, see the details in [1].

The *universality problem* asks, given an automaton $\mathcal{A}$ over Σ, whether $L(\mathcal{A}) = \Sigma^*$.

3 Partially Ordered NFAs

In this section, we introduce poNFAs, recall their characterisation in terms of the Straubing-Thérien hierarchy, and show that universality remains PSPACE-complete even when restricting to binary alphabets. Merely the case of unary alphabets turns out to be simpler.

▶ **Definition 2.** Let $\mathcal{A}$ be an NFA. A state q is *reachable* from a state p, written $p \leq q$, if there is a word $w \in \Sigma^*$ such that $q \in p \cdot w$. We write $p < q$ if $p \leq q$ and $p \neq q$. $\mathcal{A}$ is a *partially ordered NFA* (*poNFA*) if $\leq$ is a partial order.

The expressive power of poNFAs can be characterised by the *Straubing-Thérien (ST) hierarchy* [35, 37]. For an alphabet Σ, level 0 of this hierarchy is defined as $\mathcal{L}(0) = \{\emptyset, \Sigma^*\}$. For integers $n \geq 0$, the levels $\mathcal{L}(n)$ and $\mathcal{L}(n + \frac{1}{2})$ are as follows:

- $\mathcal{L}(n + \frac{1}{2})$ consists of all finite unions of languages $L_0 a_1 L_1 a_2 \dots a_k L_k$, with $k \geq 0$, $L_0, \dots, L_k \in \mathcal{L}(n)$, and $a_1, \dots, a_k \in \Sigma$;
- $\mathcal{L}(n+1)$ consists of all finite Boolean combinations of languages from level $\mathcal{L}(n + \frac{1}{2})$.

Note that the levels of the hierarchy contain only *star-free* languages by definition. It is known that the hierarchy does not collapse on any level [8], but the problem of deciding if a language belongs to some level k is largely open for $k > \frac{5}{2}$ [2, 27]. The ST hierarchy further has close relations to the *dot-depth hierarchy* [11, 8, 36] and to complexity theory [38].

Interestingly, the languages recognised by poNFAs are exactly the languages on level $\frac{3}{2}$ of the Straubing-Thérien hierarchy [31]. Since the hierarchy is proper, this means that poNFAs can only recognise a strict subset of star-free regular languages. In spite of this rather low expressive power, the universality problem of poNFAs has the same worst-case complexity as for general NFAs, even when restricting to a fixed alphabet with only a few letters.

▶ **Theorem 3.** *For every alphabet Σ with $|\Sigma| \geq 2$, the universality problem for poNFAs over Σ is* PSPACE*-complete.*

Proof. Membership follows from the fact that universality is in PSPACE for NFAs [14].

To show hardness, we modify the construction of Aho et al. [1] to work on a two-letter alphabet. Consider a polynomial p and a p-space-bounded DTM $M = \langle Q, T, I, \delta, \sqcup, q_o, q_f \rangle$. Without loss of generality, we assume $q_0 \neq q_f$. We define an encoding of runs of M as a word over a given alphabet. For any input $x \in I^*$, we construct, in polynomial time, a regular expression R_x that represents all words that do *not* encode an accepting run of M on x. Therefore, R_x matches all words iff M does not accept x. The claim then follows by showing that R_x can be encoded by a poNFA.

A configuration of M on an input x consists of a current state $q \in Q$, the position $0 \leq \ell \leq p(|x|)$ of the read/write head, and the current tape contents $\theta_0, \dots, \theta_{p(|x|)}$ with $\theta_i \in T$. We represent it by a sequence $\langle \theta_0, \varepsilon \rangle \cdots \langle \theta_{\ell-1}, \varepsilon \rangle \langle \theta_\ell, q \rangle \langle \theta_{\ell+1}, \varepsilon \rangle \cdots \langle \theta_{p(|x|)}, \varepsilon \rangle$ of symbols from $T \times (Q \cup \{\varepsilon\})$. We denote $T \times (Q \cup \{\varepsilon\})$ by Δ. A potential run of M on x is represented by word $\#w_1\#w_2\#\cdots\#w_m\#$, where $w_i \in \Delta^{p(|x|)}$ and $\# \notin \Delta$ is a fresh separator symbol. One can construct a regular expression recognising all words over $\Delta \cup \{\#\}$ that do not correctly encode a run of M at all, or that encode a run that is not accepting [1].

We encode symbols of $\Delta \cup \{\#\}$ using a fixed alphabet $\Sigma = \{0, 1\}$. For each $\delta \in \Delta \cup \{\#\}$, let $\hat{\delta}_1 \cdots \hat{\delta}_K \in \{0,1\}^K$ be a unique binary encoding of length $K = \lceil \log(|\Delta \cup \{\#\}|) \rceil$. We

define $enc(\delta)$ to be the binary sequence $001\hat{\delta}_1 1\hat{\delta}_2 1 \cdots \hat{\delta}_K 1$ of length $L = 2K + 3$. We extend enc to words and sets of symbols as usual: $enc(\delta_1 \cdots \delta_m) = enc(\delta_1) \cdots enc(\delta_m)$ and $enc(\Delta') = \{enc(\delta) \mid \delta \in \Delta'\}$. Importantly, any word of the form $enc(\delta_1 \cdots \delta_m)$ contains 00 only at positions that are multiples of L, marking the start of one encoded symbol.

We now construct the regular expression R_x that matches all words of Σ^* that do not represent an accepting computation of M on x. We proceed in four steps: (A) we detect all words that contain words from Σ^* that are not of the form $enc(\delta)$; (B) we detect all words that do not start with the initial configuration; (C) we detect all words that do not encode a valid run since they violate a transition rule; and (D) we detect all words that encode non-accepting runs, or runs that end prematurely.

For (A), note that a word $w \in \Sigma^*$ that is not of the form $enc(v)$ for any word $v \in (\Delta \cup \{\#\})^*$ must either (A.1) start with 1 or 01; (A.2) end with 0; (A.3) contain a word $00\Sigma^{L-2}$ that is not in $enc(\Delta \cup \{\#\})$; (A.4) contain a word from $enc(\Delta \cup \{\#\})\{1, 01\}$; or (A.5) end in a word $00\Sigma^M$ with $M < L - 2$. Using E to abbreviate $enc(\Delta \cup \{\#\})$ and $\bar{E}$ to abbreviate $00\Sigma^{L-2} \setminus E$ (both sets of polynomially many binary sequences), we can express (A.1)–(A.5) in the regular expression

$$(1\Sigma^* + 01\Sigma^*) + (\Sigma^* 0) + \left(\Sigma^* \bar{E} \Sigma^*\right) + (\Sigma^* E(1 + 01)\Sigma^*) + \left(\Sigma^* 00(\Sigma + \Sigma^2 + \ldots + \Sigma^{L-3})\right) \quad (1)$$

where we use finite sets $\{e_1, \ldots, e_m\}$ to denote regular expressions $(e_1 + \ldots + e_m)$, as usual. All sets in (1) are polynomial in size, so that the overall expression is polynomial. The expression (1) can be captured by a poNFA since the only cycles required arise when translating Σ^*; they can be expressed as self-loops. All other repetitions of the form Σ^i in (1) can be expanded to polynomial-length sequences without cycles.

For (B), we want to detect all words that do not start with the word $w = enc(\#\langle x_1, q_0\rangle \langle x_2, \varepsilon\rangle \cdots \langle x_{|x|}, \varepsilon\rangle \langle \sqcup, \varepsilon\rangle \cdots \langle \sqcup, \varepsilon\rangle \#)$ of length $(p(|x|) + 2)L$. This happens if (B.1) the word is shorter than $(p(|x|) + 2)L$, or (B.2), starting at position jL for $0 \leq j \leq p(|x|) + 1$, there is a word from the polynomial set $\Sigma^L \setminus \{enc(w_j)\}$, which we abbreviate by $\bar{E}_j$. We can capture (B.1) and (B.2) in the regular expression

$$\left(\varepsilon + \Sigma + \Sigma^2 + \ldots + \Sigma^{L(p(|x|)+2)-1}\right) + \sum_{0 \leq j \leq p(|x|)+1} (\Sigma^{jL} \cdot \bar{E}_j \cdot \Sigma^*) \quad (2)$$

The empty expression ε is used for readability; it can easily be expressed in the NFA encoding. As before, it is easy to see that this expression is polynomial and does not require any nontrivial cycles when encoded in an NFA. Note that we ensure that the surrounding # in the initial configuration are present.

For (C), we need to check for incorrect transitions. Consider again the encoding $\#w_1\# \ldots \#w_m\#$ of a sequence of configurations with a word over $\Delta \cup \{\#\}$, where we can assume that w_1 encodes the initial configuration according to (A) and (B). In an encoding of a valid run, the symbol at any position $j \geq p(|x|) + 2$ is uniquely determined by the symbols at positions $j - p(|x|) - 2$, $j - p(|x|) - 1$, and $j - p(|x|)$, corresponding to the cell and its left and right neighbour in the previous configuration. Given symbols $\delta_\ell, \delta, \delta_r \in \Delta \cup \{\#\}$, we can therefore define $f(\delta_\ell, \delta, \delta_r) \in \Delta \cup \{\#\}$ to be the symbol required in the next configuration. The case where $\delta_\ell = \#$ or $\delta_r = \#$ corresponds to transitions applied at the left and right edge of the tape, respectively; for the case that $\delta = \#$, we define $f(\delta_\ell, \delta, \delta_r) = \#$, ensuring that the separator # is always present in successor configurations as well. We can then check for invalid transitions using the regular expression

$$\sum_{\delta_\ell, \delta, \delta_r \in \Delta \cup \{\#\}} \Sigma^* \cdot enc(\delta_\ell \delta \delta_r) \cdot \Sigma^{L(p(|x|)-1)} \cdot enc(\overline{f}(\delta_\ell, \delta, \delta_r)) \cdot \Sigma^* \quad (3)$$

where $\overline{f}(\delta_\ell, \delta, \delta_r) = \Delta \cup \{\#\} \setminus \{f(\delta_\ell, \delta, \delta_r)\}$. Polynomiality and poNFA-expressibility are again immediate. Note that expression (3) only detects wrong transitions if a (long enough) next configuration exists. The case that the run stops prematurely is covered next.

Finally, for (D) we detect all words that either (D.1) end in a configuration that is incomplete (too short) or (D.2) end in a configuration that is not in the final state q_f. Abbreviating $T \times (Q \setminus \{q_f\})$ as $\bar{E}_f$, and using similar ideas as above, we obtain

$$\Big(\Sigma^* enc(\#)(\Sigma^L + \ldots + \Sigma^{p(|x|)L})\Big) + \Big(\Sigma^* \bar{E}_f(\varepsilon + \Sigma^L + \ldots + \Sigma^{(p(|x|)-1)L}) enc(\#)\Big) \tag{4}$$

and this can again be expressed as a polynomial poNFA.

The expressions (1)–(4) together then detect all non-accepting or wrongly encoded runs of M. In particular, if we start from the correct initial configuration ((2) does not match), then for (3) not to match, all complete future configurations must have exactly one state and be delimited by encodings of #. Expressing the regular expressions as a single poNFA of polynomial size, we have thus reduced the word problem of polynomially space-bounded Turing machines to the universality problem of poNFAs. ◀

Ellul et al. give an example of a regular expression over a 5-letter alphabet such that the shortest non-accepted word is of exponential length, and which can also be encoded as a poNFA [13, Section 5]. Our previous proof shows such an example for an alphabet of two letters, if we use a Turing machine that runs for exponentially many steps before accepting. Note, however, that this property alone would not imply Theorem 3.

Unary Alphabet. Reducing the size of the alphabet to one leads to a reduction in complexity. This is expected, since the universality problem for NFAs over a unary alphabet is merely coNP-complete [34]. For poNFAs, however, the situation is even simpler:

▶ **Theorem 4.** *The universality problem for poNFAs over a unary alphabet is in* P.

Proof. If the language is infinite, then there must be a simple path from an initial state to an accepting state via a state with a self-loop. Let k denote the length of this path, which is bounded by the number of states. Then this path accepts all words of length at least k, that is, all words of the form $a^k a^*$. It remains to check that all words up to length k are also accepted, which can be done in polynomial time. ◀

4 Restricted Partially Ordered NFAs

We now introduce restricted poNFAs, which are distinguished by the forbidden pattern of Figure 1. We relate them to the known class of $\mathcal{R}$-trivial languages, and we establish complexity results for deciding if a language falls into this class.

▶ **Definition 5.** A *restricted partially ordered NFA (rpoNFA)* is a poNFA such that, for every state q and symbol a, if $q \in q \cdot a$ then $q \cdot a = \{q\}$.

We will show below that rpoNFAs characterise $\mathcal{R}$-trivial languages [9]. To introduce this class of languages, we first require some auxiliary definitions. A word $v = a_1 a_2 \cdots a_n$ is a *subsequence* of a word w, denoted $v \preccurlyeq w$, if $w \in \Sigma^* a_1 \Sigma^* a_2 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$. For $k \geq 0$, we write $sub_k(v) = \{u \in \Sigma^* \mid u \preccurlyeq v, |u| \leq k\}$ for the set of all subsequences of v of length up to k. Two words w_1, w_2 are $\sim_k$*-equivalent*, written $w_1 \sim_k w_2$, if $sub_k(w_1) = sub_k(w_2)$. Then $\sim_k$ is a congruence (for $\cdot$) of finite index (i.e., with finitely many equivalence classes) [32]. $\mathcal{R}$-trivial languages are defined by defining a related congruence $\sim_k^{\mathcal{R}}$ that considers subsequences of prefixes:

▶ **Definition 6.** Let $x, y \in \Sigma^*$ and $k \geq 0$. Then $x \sim_k^{\mathcal{R}} y$ if and only if
- for each prefix u of x, there exists a prefix v of y such that $u \sim_k v$, and
- for each prefix v of y, there exists a prefix u of x such that $u \sim_k v$.

A regular language is *k-$\mathcal{R}$-trivial* if it is a union of $\sim_k^{\mathcal{R}}$ classes, and it is *$\mathcal{R}$-trivial* if it is k-$\mathcal{R}$-trivial for some $k \geq 0$.

It is known that $x \sim_k^{\mathcal{R}} y$ implies $x \sim_k y$ and (if $k \geq 1$) $x \sim_{k-1}^{\mathcal{R}} y$ [9]. Therefore, every k-$\mathcal{R}$-trivial language is also $(k+1)$-$\mathcal{R}$-trivial. Moreover, it has been shown that a language L is $\mathcal{R}$-trivial if and only if the minimal DFA recognising L is partially ordered [9]. We can lift this result to characterise the expressive power of rpoNFAs. Namely, it is known that a language is $\mathcal{R}$-trivial if and only if it is a finite union of $\mathcal{R}$-expressions, i.e., expressions of the form $\Sigma_1^* a_1 \Sigma_2^* a_2 \cdots \Sigma_m^* a_m \Sigma_{m+1}^*$, for some $m \geq 0$, where $a_i \notin \Sigma_i$ for $1 \leq i \leq m$. The characterization goes back to Eilenberg and can be found, e.g., in [26]. Thus, we have the following.

▶ **Theorem 7.** *A regular language is $\mathcal{R}$-trivial if and only if it is accepted by an rpoNFA.*

This characterisation in terms of automata with forbidden patterns can be compared to results of Glaßer and Schmitz, who use DFAs with a forbidden pattern to obtain another characterisation of level $\frac{3}{2}$ of the Straubing-Thérien hierarchy [15, 29].

We can further relate the *depth* of rpoNFAs to k-$\mathcal{R}$-trivial languages. Recall that the depth of an atomaton $\mathcal{A}$, denoted $\mathit{depth}(\mathcal{A})$, is the number of input symbols on the longest simple path of $\mathcal{A}$ that starts in an initial state.

▶ **Theorem 8.** *The language recognised by a complete rpoNFA $\mathcal{A}$ is $\mathit{depth}(\mathcal{A})$-$\mathcal{R}$-trivial.*

Similar relationships have been studied for $\mathcal{J}$-trivial languages [18, 23], but we are not aware of any such investigation for $\mathcal{R}$-trivial languages.

Finally, we may ask how difficult it is to decide whether a given NFA $\mathcal{A}$ accepts a language that is $\mathcal{R}$-trivial or k-$\mathcal{R}$-trivial for a specific $k \geq 0$. For most levels of the ST hierarchy, it is not even known if this problem is decidable, and when it is, exact complexity bounds are often missing [27]. The main exception are $\mathcal{J}$-trivial languages – level 1 of the hiearchy – which have recently attracted some attention, motivated by applications in algebra and XML databases [16, 18, 24].

The following result is a special case of a more general result in [17, Theorem 3.1].

▶ **Theorem 9.** *Given an NFA $\mathcal{A}$, it is* PSPACE*-complete to decide if the language accepted by $\mathcal{A}$ is $\mathcal{R}$-trivial.*

To the best of our knowledge, the following complexity results for recognising (k-)$\mathcal{R}$-trivial languages had not been obtained previously.

▶ **Theorem 10.** *Given an NFA $\mathcal{A}$ and $k \geq 0$, it is* PSPACE*-complete to decide if the language accepted by $\mathcal{A}$ is k-$\mathcal{R}$-trivial.*

In both previous theorems, hardness is shown by reduction from the universality problem for NFAs, hence it holds even for binary alphabets [14]. For a unary alphabet, we can obtain the following result.

▶ **Theorem 11.** *Given an NFA $\mathcal{A}$ over a unary alphabet, the problems of deciding if the language accepted by $\mathcal{A}$ is $\mathcal{R}$-trivial, or k-$\mathcal{R}$-trivial for a given $k \geq 0$, respectively, are both* coNP*-complete.*

5 Deciding Universality of rpoNFAs

In this section, we return to the universality problem for the case of rpoNFAs. We first show that we can indeed obtain the hoped-for reduction in complexity when using a fixed alphabet. For the general case, however, we can recover the same PSPACE lower bound as for poNFAs, albeit with a more involved proof. Even for fixed alphabets, we can get a coNP lower bound:

▶ **Lemma 12.** *The universality problem of rpoNFAs is* coNP*-hard even when restricting to alphabets with two letters.*

The proof proceeds by a direct reduction of propositional logic satisfiability to the emptiness of rpoNFAs. For a matching upper bound, we use some results from the literature.

▶ **Lemma 13** ([9])**.** *Every congruence class of $\sim_k^{\mathcal{R}}$ contains a unique element of minimal length. If $a_1, a_2, \ldots, a_n \in \Sigma$, then $a_1 a_2 \cdots a_n$ is minimal if and only if $sub_k(\varepsilon) \subsetneq sub_k(a_1) \subsetneq sub_k(a_1 a_2) \subsetneq \ldots \subsetneq sub_k(a_1 a_2 \ldots a_n)$.*

The maximal length of such a word has also been studied [24].

▶ **Lemma 14** ([24])**.** *Let Σ be an alphabet of cardinality $|\Sigma| \geq 1$, and let $k \geq 1$. The length of a longest word, w, such that $sub_k(w) = \{v \in \Sigma^* \mid |v| \leq k\}$, and, for any two distinct prefixes w_1 and w_2 of w, $sub_k(w_1) \neq sub_k(w_2)$, is $\binom{k+|\Sigma|}{k} - 1$. The bound is tight.*

Lemma 13 and 14 provide the main ingredients for showing that, if the size $|\Sigma|$ of the alphabet is bounded, then non-universality is witnessed by a word of polynomial length. Together with Lemma 12, this allows us to establish the following result, which we state in a more general form.

▶ **Theorem 15.** *Let Σ be a fixed alphabet, and let $\mathcal{A}$ and $\mathcal{B}$ be two complete rpoNFAs over Σ. Then the problem whether $L(\mathcal{A}) \subseteq L(\mathcal{B})$ is* coNP*-complete.*

Proof. Hardness follows from Lemma 12. To prove membership, we denote $|\Sigma| = m$. Let $k = \max\{depth(\mathcal{A}), depth(\mathcal{B})\}$; k is bounded by the number of states of $\mathcal{A}$ and $\mathcal{B}$. By Theorem 8, languages $L(\mathcal{A})$ and $L(\mathcal{B})$ are k-$\mathcal{R}$-trivial, which means that they are a finite union of $\sim_k^{\mathcal{R}}$ classes. According to Lemmas 13 and 14, the length of the unique minimal representatives of the $\sim_k^{\mathcal{R}}$ classes is at most $\binom{k+m}{k} - 1 < \frac{(k+m)^m}{m!}$. Since m is a constant, the bound is polynomial in k. Therefore, if the language $L(\mathcal{A})$ is not a subset of $L(\mathcal{B})$, then there exists a polynomial certificate, which can be guessed by a nondeterministic algorithm. ◀

▶ **Corollary 16.** *Let Σ be a fixed alphabet. Then the universality problem for rpoNFAs over Σ is* coNP*-complete.*

Without fixing the alphabet, universality remains PSPACE-hard even for rpoNFAs, but a proof along the lines of Theorem 3 is not straightforward. In essence, rpoNFAs lose the ability to navigate to an arbitrary position within a word for checking some pattern there. Expressions of the form $(\Sigma^* \cdots)$, which we frequently used, e.g., in (1), are therefore excluded. This is problematic since the run of a polynomially space-bounded Turing machine may be of exponential length, and we need to match patterns across the full length of our (equally exponential) encoding of this run. How can we navigate such a long word without using Σ^*? Our answer is to first define an rpoNFA that accepts all words except for a single, exponentially long word. This word will then be used as an rpoNFA-supported "substrate" for our Turing machine encoding, which again follows Theorem 3.

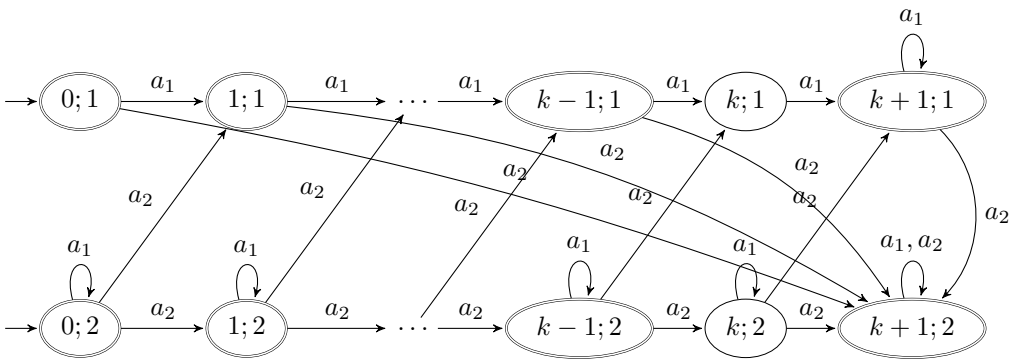

Figure 2 The rpoNFA $\mathcal{A}_{k,2}$ with $2(k+2)$ states.

Table 2 Recursive construction of words $W_{k,n}$ as used in the proof of Lemma 17.

$k\backslash n$	1	2	3
1	a_1	a_1a_2	$a_1a_2a_3$
2	a_1^2	$a_1^2a_2a_1a_2$	$a_1^2a_2a_1a_2a_3a_1a_2a_3$
3	a_1^3	$a_1^3a_2a_1^2a_2a_1a_2$	$a_1^3a_2a_1^2a_2a_1a_2a_3a_1^2a_2a_1a_2a_3a_1a_2a_3$
4	a_1^4	$a_1^4a_2a_1^3a_2a_1^2a_2a_1a_2$	$a_1^4a_2a_1^3a_2a_1^2a_2a_1a_2a_3a_1^3a_2a_1^2a_2a_1a_2a_3a_1^2a_2a_1a_2a_3a_1a_2a_3$

▶ **Lemma 17.** *For all positive integers k and n, there exists an rpoNFA $\mathcal{A}_{k,n}$ over an n-letter alphabet with $n(k+2)$ states such that the unique word not accepted by $\mathcal{A}_{k,n}$ is of length $\binom{k+n}{k} - 1$.*

Proof sketch. For integers $k, n \geq 1$, we recursively define words $W_{k,n}$ over the alphabet $\Sigma_n = \{a_1, a_2, \ldots, a_n\}$. For the base cases, we set $W_{k,1} = a_1^k$ and $W_{1,n} = a_1a_2 \ldots a_n$. The cases for $k, n > 1$ are defined recursively by setting

$$\begin{aligned} W_{k,n} &= W_{k,n-1}\, a_n\, W_{k-1,n} \\ &= W_{k,n-1}\, a_n\, W_{k-1,n-1}\, a_n\, W_{k-2,n} \\ &= W_{k,n-1}\, a_n\, W_{k-1,n-1}\, a_n \cdots a_n\, W_{1,n-1}\, a_n\,. \end{aligned} \tag{5}$$

The recursive construction is illustrated in Table 2. The length of $W_{k,n}$ is $\binom{k+n}{n} - 1$ [24]. We further set $W_{k,n} = \varepsilon$ whenever $kn = 0$, since this is useful for defining $\mathcal{A}_{k,n}$ below.

We construct an rpoNFA $\mathcal{A}_{k,n}$ over Σ_n that accepts the language $\Sigma_n^* \setminus \{W_{k,n}\}$. For $n = 1$ and $k \geq 0$, let $\mathcal{A}_{k,1}$ be the minimal DFA accepting the language $\{a_1\}^* \setminus \{a_1^k\}$. It consists of the $k+2$ states of the form $(i;1)$ in the upper part of Figure 2, together with the given transitions. All states but $(k;1)$ are final, and $(0;1)$ is initial.

Given $\mathcal{A}_{k,n-1}$, we recursively construct $\mathcal{A}_{k,n}$ as defined next. The construction for $n = 2$ is illustrated in Figure 2. We obtain $\mathcal{A}_{k,n}$ from $\mathcal{A}_{k,n-1}$ by adding $k+2$ states $(0;n), (1;n), \ldots, (k+1;n)$, where $(0;n)$ is added to the initial states, and all states other than $(k;n)$ are added to the final states. $\mathcal{A}_{k,n}$ therefore has $n(k+2)$ states.

The additional transitions of $\mathcal{A}_{k,n}$ consist of four groups: (1) self-loops $(i;n) \xrightarrow{a_j} (i;n)$ for every $i = 0, \ldots, k+1$ and $a_j = a_1, \ldots, a_{n-1}$. (2) transitions $(i;n) \xrightarrow{a_n} (i+1;n)$ for every $i = 0, \ldots, k$. (3) transitions $(i;n) \xrightarrow{a_n} (i+1;m)$ for every $i = 0, \ldots, k$ and $m = 1, \ldots, n-1$. (4) transitions $(i;m) \xrightarrow{a_n} (k+1;n)$ for every accepting state $(i;m)$ of $\mathcal{A}_{k,n-1}$.

The additional states of $\mathcal{A}_{k,n}$ and transitions (1) and (2) ensure acceptance of every word that does not contain exactly k occurrences of a_n. The transitions (3) ensure acceptance of

all words in $(\Sigma_{n-1}^* a_n)^{i+1} L(\mathcal{A}_{k-(i+1),n-1}) a_n \Sigma_n^*$, for which the word between the $(i+1)$st and the $(i+2)$nd occurrence of a_n is not of the form $W_{k-(i+1),n-1}$, hence not a correct subword of $W_{k,n} = W_{k,n-1}a_n \cdots a_n W_{k-(i+1),n-1}a_n \cdots a_n W_{1,n-1}a_n$. The transitions (4) ensure that all words with a prefix $w \cdot a_n$ are accepted, where w is any word $\Sigma_{n-1}^* \setminus \{W_{k,n-1}\}$ accepted by $\mathcal{A}_{k,n-1}$. Together, these conditions ensure that $\mathcal{A}_{k,n}$ accepts every input other than $W_{k,n}$

It remains to show that $\mathcal{A}_{k,n}$ does not accept $W_{k,n}$, which we do by induction on (k,n). We start with the base cases. For $(0,n)$ and any $n \geq 1$, the word $W_{0,n} = \varepsilon$ is not accepted by $\mathcal{A}_{0,n}$, since the initial states $(0,m) = (k,m)$ of $\mathcal{A}_{0,n}$ are not accepting. Likewise, for $(k,1)$ and any $k \geq 0$, we find that $W_{k,1} = a_i^k$ is not accepted by $\mathcal{A}_{k,1}$ (the upper part of Figure 2).

For the inductive case $(k,n) \geq (1,2)$, assume $\mathcal{A}_{k',n'}$ does not accept $W_{k',n'}$ for any $(k',n') < (k,n)$. We have $W_{k,n} = W_{k,n-1}a_n W_{k-1,n}$, and $W_{k,n-1}$ is not accepted by $\mathcal{A}_{k,n-1}$ by induction. In addition, there is no transition under a_n from any non-accepting state of $\mathcal{A}_{k,n-1}$ in $\mathcal{A}_{k,n}$. Therefore, if $W_{k,n}$ is accepted by $\mathcal{A}_{k,n}$, it must be accepted in a run starting from the initial state $(0;n)$. Since $W_{k,n-1}$ does not contain a_n, we find that $\mathcal{A}_{k,n}$ can only reach the states $(0;n) \cdot W_{k,n-1}a_n = \{(1;m) \mid 1 \leq m \leq n\}$ after reading $W_{k,n-1}a_n$. These are the initial states of automaton $\mathcal{A}_{k-1,n}$, which does not accept $W_{k-1,n}$ by induction. Hence $W_{k,n}$ is not accepted by $\mathcal{A}_{k,n}$. ◀

As a corollary, we find that there are rpoNFAs $\mathcal{A} = \mathcal{A}_{n,n}$ for which the shortest non-accepted word is exponential in the size of $\mathcal{A}$. Note that $\binom{2n}{n} \geq 2^n$.

▶ **Corollary 18.** *For every integer $n \geq 1$, there is an rpoNFA $\mathcal{A}_n$ over an n-letter alphabet with $n(n+2)$ states such that the shortest word not accepted by $\mathcal{A}_n$ is of length at least $\binom{2n}{n} - 1$. Therefore, any minimal DFA accepting the same language has at least $\binom{2n}{n}$ states.*

To simulate exponentially long runs of a Turing machine, we start from an encoding of runs using words $\#w_1\# \ldots \#w_m\#$ as in Theorem 3, but we combine every letter of this encoding with one letter of the alphabet of $\mathcal{A}_n$. We then accept all words for which the projection to the alphabet of $\mathcal{A}_n$ is accepted by $\mathcal{A}_n$, i.e., all but those words of exponential length that are based on the unique word not accepted by $\mathcal{A}_n$. We ensure that, if there is an accepting run, it will have an encoding of this length. It remains to eliminate (accept) all words that correspond to a non-accepting or wrongly encoded run. We can check this as in Theorem 3, restricting to the first components of our combined alphabet. The self-loop that was used to encode Σ^* in poNFAs is replaced by a full copy of $\mathcal{A}_n$, with an additional transition from each state that allows us to leave this "loop." This does not simulate the full loop, but it allows us to navigate the entirety of our exponential word, which is all we need.

▶ **Theorem 19.** *The universality problem for rpoNFAs is* PSPACE*-complete.*

Proof. The membership follows since universality is in PSPACE for NFAs. For hardness, we proceed as explained above. Consider a p-space-bounded DTM $M = \langle Q, T, I, \delta, \sqcup, q_o, q_f \rangle$ as in the proof of Theorem 3. We encode runs of M as words over $T \times (Q \cup \{\varepsilon\}) \cup \{\#\}$ as before. We can use an unrestricted alphabet now, so no binary encoding is needed, and the regular expressions can be simplified accordingly.

If M has an accepting run, then it has one without repeated configurations. For an input word x, there are $C(x) = (|T \times (Q \cup \{\varepsilon\})|)^{p(|x|)}$ distinct configuration words in our encoding. Considering separator symbols $\#$, the maximal length of the encoding of a run without repeated configurations therefore is $1 + C(x)(p(|x|)+1)$. Let n be the least number such that $|W_{n,n}| \geq 1 + C(x)(p(|x|)+1)$. Since $|W_{n,n}| + 1 = \binom{2n}{n} \geq 2^n$, it follows that n is smaller than $\lceil \log(1 + C(x)(p(|x|)+1)) \rceil$ and hence polynomial in the size of M and x.

Consider the automaton $\mathcal{A}_{n,n}$ with alphabet $\Sigma_n = \{a_1, \dots, a_n\}$ of Lemma 17, and define $\Delta_{\#\$} = T \times (Q \cup \{\varepsilon\}) \cup \{\#, \$\}$. We consider the alphabet $\Pi = \Sigma_n \times \Delta_{\#\$}$, where the second letter is used for encoding a run as in Theorem 3. Since $|W_{n,n}|$ may not be a multiple of $p(|x|)+1$, we add \$ to fill up any remaining space after the last configuration. For a word $w = \langle a_{i_1}, \delta_1\rangle \cdots \langle a_{i_\ell}, \delta_\ell\rangle \in \Pi^\ell$, we define $w[1] = a_{i_1} \cdots a_{i_\ell} \in \Sigma_n^\ell$ and $w[2] = \delta_1 \dots \delta_\ell \in \Delta_{\#\$}^\ell$. Conversely, for a word $v \in \Delta_{\#\$}^*$, we write $enc(v)$ to denote the set of all words $w \in \Pi^{|v|}$ with $w[2] = v$. Similarly, for $v \in \Sigma_n^*$, $enc(v)$ denotes the words $w \in \Pi^{|v|}$ with $w[1] = v$. We extend this notation to sets of words.

We say that a word w encodes an accepting run of M on x if $w[1] = W_{n,n}$ and $w[2]$ is of the form $\#w_1\# \cdots \#w_m\#\$^j$ such that there is an $i \in \{1, \dots, m\}$ for which we have that

- $\#w_1\# \cdots \#w_i\#$ encodes an accepting run of M on x as in the proof of Theorem 3,
- $w_k = w_i$ for all $k \in \{i+1, \dots, m\}$, and
- $j \leq p(|x|)$.

In other words, we extend the encoding by repeating the accepting configuration until we have less than $p(|x|)+1$ symbols before the end of $|W_{n,n}|$ and fill up the remaining places with \$.

The modified encoding requires slightly modified expressions for capturing conditions (A)–(D) from the proof of Theorem 3. Condition (A) is not necessary, since we do not encode symbols in binary. Condition (B) can use the same expression as in (2), adjusted to our alphabet:

$$\left(\varepsilon + \Pi + \Pi^2 + \ldots + \Pi^{p(|x|)+1}\right) + \sum_{0 \leq j \leq p(|x|)+1} (\Pi^j \cdot \bar{E}_j \cdot \Pi^*) \tag{6}$$

where $\bar{E}_j$ is the set $\Sigma_n \times (\Delta_{\#\$} \setminus \{w_j\})$ where w_j encodes the jth symbol on the initial tape as in Theorem 3. All uses of Π^i in this expression encode words of polynomial length, which can be represented in rpoNFAs. Trailing expressions Π^* do not lead to the forbidden pattern of Figure 1.

Condition (C) uses the same ideas as in Theorem 3, especially the transition encoding function f, which we extend to $f : \Delta_{\#\$}^3 \to \Delta_{\#\$}$. For allowing the last configuration to be repeated, we define f as if the final state q_f of M had a self loop (a transition that does not modify the tape, state, or head position). Moreover, we generally permit \$ to occur instead of the expected next configuration symbol. We obtain:

$$\Pi^* \sum_{\delta_\ell, \delta, \delta_r \in \Delta_{\#\$}} enc(\delta_\ell \delta \delta_r) \cdot \Pi^{p(|x|)-1} \cdot \hat{f}(\delta_\ell, \delta, \delta_r) \cdot \Pi^* \tag{7}$$

where $\hat{f}(\delta_\ell, \delta, \delta_r)$ is $\Pi \setminus enc(\{f(\delta_\ell, \delta, \delta_r), \$\})$. Expression (7) is not readily encoded in an rpoNFA, due to the leading Π^*. To address this, we replace Π^* by the expression $\Pi^{\leq |W_{n,n}|-1}$, which matches every word $w \in \Pi^*$ with $|w| \leq |W_{n,n}| - 1$. Clearly, this suffices for our case. As $|W_{n,n}| - 1$ is exponential, we cannot encode this directly as for other expressions Π^i before and we use $\mathcal{A}(n,n)$ instead.

In detail, let E be the expression obtained from (7) when omitting the initial Π^*, and let $\mathcal{A}$ be an rpoNFA that accepts the language of E. We can construct $\mathcal{A}$ so that it has a single initial state. Moreover, let $enc(\mathcal{A}_{n,n})$ be the automaton $\mathcal{A}_{n,n}$ of Lemma 17 with each transition $q \xrightarrow{a_i} q'$ replaced by all transitions $q \xrightarrow{\pi} q'$ with $\pi \in enc(a_i)$. We construct an rpoNFA $\mathcal{A}'$ that accepts the language of $(\Pi^* \setminus \{W_{n,n}\}) + (\Pi^{\leq |W_{n,n}|-1} \cdot E)$ by merging $enc(\mathcal{A}_{n,n})$ with $n(n+1)$ copies of $\mathcal{A}$, where we identify the initial state of each such copy with a unique final state of $enc(\mathcal{A}_{n,n})$. The fact that $enc(\mathcal{A}_{n,n})$ alone already accepts $(\Pi^* \setminus \{enc(W_{n,n})\})$

was shown in the proof of Lemma 17. This also implies that it accepts all words of length $\leq |W_{n,n}| - 1$ as needed to show that $(\Pi^{\leq |W_{n,n}|-1} \cdot E)$ is accepted. Entering states of (a copy of) $\mathcal{A}$ after accepting a word of length $\geq |W_{n,n}|$ is possible, but all words accepted in such a way are longer than $W_{n,n}$ and hence in $(\Pi^* \setminus \{enc(W_{n,n})\})$.

Note that the acceptance of $(\Pi^* \setminus \{enc(W_{n,n})\})$, which is a side effect of this encoding, does not relate to expressing (7) but is still useful for our intended overall encoding.

The final condition (D) is minimally modified to allow for up to $p(|x|)$ trailing \$. For a word v, we use $v^{\leq i}$ to abbreviate $(\varepsilon + v + \ldots + v^i)$, and we define $\bar{E}_f = (T \times (Q \setminus \{q_f\}))$ as before. Since (C) does not accept words with too many trailing \$, we add this here instead. Moreover, we need to check that all the symbols \$ appear only at the end, that is, the last expression accepts all inputs where \$ is followed by a different symbol.

$$\begin{aligned}
&\Pi^* enc(\#)(\Pi + \ldots + \Pi^{p(|x|)}) enc(\$)^{\leq p(|x|)} + \\
&\Pi^* enc(\bar{E}_f)(\varepsilon + \Pi + \ldots + \Pi^{p(|x|)-1}) enc(\#) enc(\$)^{\leq p(|x|)} + \\
&\Pi^* enc(\$)^{p(|x|)+1} + \\
&(\Pi \setminus enc(\$))^* enc(\$) enc(\$)^* (\Pi \setminus enc(\$)) \Pi^*
\end{aligned} \tag{8}$$

As before, we cannot encode the leading Π^* directly as an rpoNFA, but we can perform a similar construction as in (7) to overcome this problem.

The union of the rpoNFAs for (6)–(8) constitutes an rpoNFA that is polynomial in the size of M and x, and that is universal if and only if M does not accept x. ◀

6 Conclusion

Our results regarding the complexity of deciding universality for partially ordered NFAs are summarised in Table 1. We found that poNFAs over a fixed, two-letter alphabet are still powerful enough to recognise the language of all non-accepting computations of a PSpace Turing machine. Restricting poNFAs further by forbidding the pattern of Figure 1, we could establish lower coNP complexity bounds for universality for alphabets of bounded size. We can view this as the complexity of universality of rpoNFAs in terms of the size of the automaton when keeping the alphabet fixed. Unfortunately, the complexity is PSpace-complete even for rpoNFAs over arbitrary (unbounded) alphabets. The proof uses an interesting construction where the encoding of a Turing machine computation is "piggybacked" on an exponentially long word, for which a dedicated rpoNFA is constructed.

We have characterised the expressive power of rpoNFAs by relating them to the class of $\mathcal{R}$-trivial languages. It is worth noting that the complexity bounds we establish for recognising $\mathcal{R}$-triviality for a given NFA agrees with the complexity of the rpoNFA universality problem for both fixed and arbitrary alphabets. Our results on universality therefore extend beyond rpoNFAs to arbitrary NFAs that recognise $\mathcal{R}$-trivial languages.

Moreover, the results on universality further extend to the complexity of inclusion and equivalence, as explained in the introduction.

Our work can be considered as a contribution to the wider field of studying subclasses of star-free regular languages. The Straubing-Thérien hierarchy provides a large field for interesting future work in this area.

Acknowledgements. We would like to thank Wim Martens for pointing out his paper [22] to our attention and to an anonymous reviewer for pointing out paper [17] and its consequence to Theorem 9.

References

1. Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
2. Jorge Almeida, Jana Bartoňová, Ondřej Klíma, and Michal Kunc. On decidability of intermediate levels of concatenation hierarchies. In *Developments in Language Theory*, volume 9168 of *LNCS*, pages 58–70. Springer, 2015.
3. Yehoshua Bar-Hillel, Micha A. Perles, and Eli Shamir. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, 14:143–172, 1961.
4. Pablo Barceló, Leonid Libkin, and Juan L. Reutter. Querying regular graph patterns. *Journal of the ACM*, 61(1):8:1–8:54, 2014.
5. Geert Jan Bex, Wouter Gelade, Wim Martens, and Frank Neven. Simplifying XML schema: Effortless handling of nondeterministic regular expressions. In *ACM SIGMOD International Conference on Management of Data*, pages 731–744. ACM, 2009.
6. Ahmed Bouajjani, Anca Muscholl, and Tayssir Touilim. Permutation rewriting and algorithmic verification. *Information and Computation*, 205(2):199–224, 2007.
7. Anne Brüggemann-Klein and Derick Wood. One-unambiguous regular languages. *Information and Computation*, 142(2):182–206, 1998.
8. Janus A. Brzozowski and Robert Knast. The dot-depth hierarchy of star-free languages is infinite. *Journal of Computer and System Sciences*, 16(1):37–55, 1978.
9. Janusz A. Brzozowski and Faith E. Fich. Languages of R-trivial monoids. *Journal of Computer and System Sciences*, 20(1):32–49, 1980.
10. Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Reasoning on regular path queries. *SIGMOD Record*, 32(4):83–92, 2003.
11. Rina S. Cohen and Janusz A. Brzozowski. Dot-depth of star-free events. *Journal of Computer and System Sciences*, 5(1):1–16, 1971.
12. Wojciech Czerwinski, Claire David, Katja Losemann, and Wim Martens. Deciding definability by deterministic regular expressions. In *International Conference on Foundations of Software Science and Computation Structures*, volume 7794 of *LNCS*, pages 289–304. Springer, 2013.
13. Keith Ellul, Bryan Krawetz, Jeffrey Shallit, and Ming-Wei Wang. Regular expressions: New results and open problems. *Journal of Automata, Languages and Combinatorics*, 10(4):407–437, 2005.
14. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
15. Christian Glaßer and Heinz Schmitz. Languages of dot-depth 3/2. *Theory of Computing Systems*, 42(2):256–286, 2008.
16. Piotr Hofman and Wim Martens. Separability by short subsequences and subwords. In *International Conference on Database Theory*, volume 31 of *LIPIcs*, pages 230–246, 2015.
17. Harry B. Hunt III and Daniel J. Rosenkrantz. Computational parallels between the regular and context-free languages. *SIAM Journal on Computing*, 7(1):99–114, 1978.
18. Ondřej Klíma and Libor Polák. Alternative automata characterization of piecewise testable languages. In *Developments in Language Theory*, volume 7907 of *LNCS*, pages 289–300. Springer, 2013.
19. Markus Krötzsch, Tomáš Masopust, and Michaël Thomazo. On the complexity of universality for partially ordered NFAs. Technical report. URL: `https://ddll.inf.tu-dresden.de/web/Inproceedings3086`.
20. Manfred Kufleitner and Alexander Lauser. Partially ordered two-way Büchi automata. *International Journal of Foundations of Computer Science*, 22(8):1861–1876, 2011.

21 Kamal Lodaya, Paritosh K. Pandya, and Simoni S. Shah. Around dot depth two. In *Developments in Language Theory*, volume 6224 of *LNCS*, pages 303–315. Springer, 2010.

22 Wim Martens, Frank Neven, and Thomas Schwentick. Complexity of decision problems for XML schemas and chain regular expressions. *SIAM Journal on Computing*, 39(4):1486–1530, 2009.

23 Tomáš Masopust. Piecewise testable languages and nondeterministic automata. In *Mathematical Foundations of Computer Science*, volume 58 of *LIPIcs*, pages 68:1–68:14, 2016.

24 Tomáš Masopust and Michaël Thomazo. On the complexity of k-piecewise testability and the depth of automata. In *Developments in Language Theory*, volume 9168 of *LNCS*, pages 364–376. Springer, 2015.

25 Alfred R. Meyer and Larry J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Symposium on Switching and Automata Theory (SWAT/FOCS)*, pages 125–129. IEEE Computer Society, 1972.

26 Jean-Éric Pin. *Varieties Of Formal Languages*. Plenum Press, New York, 1986.

27 Thomas Place and Marc Zeitoun. Separation and the successor relation. In *Symposium on Theoretical Aspects of Computer Science*, volume 30 of *LIPIcs*, pages 662–675, 2015.

28 Narad Rampersad, Jeffrey Shallit, and Zhi Xu. The computational complexity of universality problems for prefixes, suffixes, factors, and subwords of regular languages. *Fundamenta Informatica*, 116(1-4):223–236, 2012.

29 Heinz Schmitz. *The forbidden pattern approach to concatenation hierachies*. PhD thesis, University of Würzburg, 2000.

30 Marcel P. Schützenberger. Sur le produit de concatenation non ambigu. *Semigroup Forum*, 13(1):47–75, 1976.

31 Thomas Schwentick, Denis Thérien, and Heribert Vollmer. Partially-ordered two-way automata: A new characterization of DA. In *Developments in Language Theory*, volume 2295 of *LNCS*, pages 239–250. Springer, 2001.

32 Imre Simon. *Hierarchies of Events with Dot-Depth One*. PhD thesis, Department of Applied Analysis and Computer Science, University of Waterloo, Canada, 1972.

33 Giorgio Stefanoni, Boris Motik, Markus Krötzsch, and Sebastian Rudolph. The complexity of answering conjunctive and navigational queries over OWL 2 EL knowledge bases. *Journal of Artificial Intelligence Research*, 51:645–705, 2014.

34 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *ACM Symposium on the Theory of Computing*, pages 1–9. ACM, 1973.

35 Howard Straubing. A generalization of the Schützenberger product of finite monoids. *Theoretical Computer Science*, 13:137–150, 1981.

36 Howard Straubing. Finite semigroup varieties of the form $\mathbf{V}^*\mathbf{D}$. *Journal of Pure and Applied Algebra*, 36:53–94, 1985.

37 Denis Thérien. Classification of finite monoids: The language approach. *Theoretical Computer Science*, 14:195–208, 1981.

38 Klaus W. Wagner. Leaf language classes. In *Machines, Computations, and Universality*, volume 3354 of *LNCS*, pages 60–81. Springer, 2004.