PARTIALLY ORDERED AUTOMATA AND PIECEWISE TESTABILITY

TOMÁŠ MASOPUST AND MARKUS KRÖTZSCH

Department of Computer Science, Palacky University in Olomouc, and Institute of Mathematics of the Czech Academy of Sciences, Prague, Czechia
e-mail address: tomas.masopust@upol.cz

Knowledge-Based Systems Group, TU Dresden, Germany
e-mail address: markus.kroetzsch@tu-dresden.de

Abstract. *Partially ordered automata* are automata where the transition relation induces a partial order on states. The expressive power of partially ordered automata is closely related to the expressivity of fragments of first-order logic on finite words or, equivalently, to the language classes of the levels of the Straubing-Thérien hierarchy. Several fragments (levels) have been intensively investigated under various names. For instance, the fragment of first-order formulae with a single existential block of quantifiers in prenex normal form is known as *piecewise testable languages* or *J-trivial languages*. These languages are characterized by confluent partially ordered DFAs or by complete, confluent, and self-loop-deterministic partially ordered NFAs (ptNFAs for short). In this paper, we study the complexity of basic questions for several types of partially ordered automata on finite words; namely, the questions of inclusion, equivalence, and (k-)piecewise testability. The lower-bound complexity boils down to the complexity of universality. The universality problem asks whether a system recognizes all words over its alphabet. For ptNFAs, the complexity of universality decreases if the alphabet is fixed, but it is open if the alphabet may grow with the number of states. We show that deciding universality for general ptNFAs is as hard as for general NFAs. Our proof is a novel and nontrivial extension of our recent construction for self-loop-deterministic partially ordered NFAs, a model strictly more expressive than ptNFAs. We provide a comprehensive picture of the complexities of the problems of inclusion, equivalence, and (k-)piecewise testability for the considered types of automata.

2012 ACM CCS: [**Theory of computation**]: Formal languages and automata theory.
2010 Mathematics Subject Classification: 68Q45, 68Q17, 68Q25, 03D05.
Key words and phrases: Automata, Nondeterminism, Complexity.

* This paper is a revised and full version of our recent work partially presented at MFCS 2016 [49] and SOFSEM 2018 [52]. Except for full proofs, it contains new results and provides an overview of the complexity results for several types of partially ordered NFAs and operations of universality, inclusion, equivalence, and (k-)piecewise testability.

Supported in part by the German Research Foundation (DFG) in project number 389792660 (TRR 248, Center for Perspicuous Systems) and Emmy Noether grant KR 4381/1-1 (DIAMOND), by the Ministry of Education, Youth and Sports under the INTER-EXCELLENCE project LTAUSA19098, by the Czech Science Foundation grant GC19-06175J, and by RVO 67985840.

1. Introduction

Partially ordered automata – also known as *1-weak*, *very weak*, *linear*, *acyclic* or *extensive automata* [18, 40, 58] – are finite automata where the transition relation induces a partial order on states. This restriction on the behaviour of the automata implies that as soon as a state is left during the computation, it is never visited again. In other words, the only cycles of partially ordered automata are self-loops.

Partially ordered automata attracted attention of many researchers because of their relation to logical, algebraic, and combinatorial charazterizations of languages. From the logical perspective, they characterize (Boolean combinations of) fragments of first-order logic on finite words. For an integer $i \geq 1$, the fragment Σ_i of the first-order logic FO[$<$] on finite words consists of all FO[$<$] formulae in prenex normal form with i blocks of quantifiers, starting with a block of existential quantifiers [19]. From the algebraic perspective, they characterize regular languages whose syntactic monoids possess some algebraic properties. For instance, the syntactic monoid of the language is J-trivial or R-trivial, where J and R are Green's relations [25], see also Pin [58]. From the combinatorial perspective, they characterize some levels of the Straubing-Thérien hierarchy [78, 80]. For an alphabet Σ, level 0 of the Straubing-Thérien hierarchy is defined as $\mathscr{L}(0) = \{\emptyset, \Sigma^*\}$, and for integers $n \geq 0$, the half levels $\mathscr{L}(n + \frac{1}{2})$ consist of all finite unions of languages $L_0 a_1 L_1 a_2 \cdots a_k L_k$, with $k \geq 0$, $L_0, \ldots, L_k \in \mathscr{L}(n)$, and $a_1, \ldots, a_k \in \Sigma$, and the full levels $\mathscr{L}(n+1)$ consist of all finite Boolean combinations of languages from level $\mathscr{L}(n + \frac{1}{2})$. The hierarchy does not collapse on any level [10]; see also Pin [58] for more details. The Straubing-Thérien hierarchy has a close relation to the dot-depth hierarchy [10, 15, 79] and to complexity theory [83]. We now provide more details and mention related research.

As early as 1972, Simon [74] studied a subclass of regular languages called *piecewise testable languages* and provided its characterization in terms of partially ordered automata. A regular language over Σ is piecewise testable if it is a finite Boolean combination of languages of the form $\Sigma^* a_1 \Sigma^* a_2 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$, where $a_i \in \Sigma$ and $n \geq 0$. If $n \leq k$, the language is *k-piecewise testable*. Simon showed that piecewise testable languages are characterized by *confluent* partially ordered deterministic finite automata (DFAs), where confluence means that for all states q and letters a and b, if there are transitions $q \xrightarrow{a} q_a$ and $q \xrightarrow{b} q_b$, then there is a word $w \in \{a, b\}^*$ that leads the automaton from both q_a and q_b to the same state. Piecewise testable languages are known under many names in the literature. Considering the perpectives described above, piecewise testable languages are known as finite boolean combinations of the fragment Σ_1 of the first-order logic FO[$<$] on finite words [19]; as *J-trivial languages*, because their syntactic monoids are J-trivial [74]; or as level 1 of the Straubing-Thérien hierarchy. This list is, indeed, not exhaustive. Piecewise testable languages appear in many contexts; see, *e.g.*, Karandikar and Schnoebelen [33] who introduced the *height* of a language and used it to study the expressivity of a two-variable fragment of first-order logic of sequences with the subword ordering. This fragment can only express piecewise testable properties. Although we study piecewise testable languages on classical $*$-languages in this paper, it is worth mentioning that piecewise testable languages have been extended from words to trees by Bojanczyk, Segoufin and Straubing [6].

In 1980, Brzozowski and Fich [9] investigated the expressivity of *partially ordered DFAs* and showed that they characterize the class of languages whose syntactic monoid is R-trivial. Consequently, the languages of partially ordered DFAs are called *R-trivial languages* and are strictly more powerful than piecewise testable languages.

Schwentick et al. [70] studied *partially ordered nondeterministic finite automata* (poNFAs) and showed that they characterize the first-order fragment Σ_2 on finite words or, equivalently, level $\frac{3}{2}$ of the Straubing-Thérien hierarchy. Partially ordered NFAs are thus strictly more powerful than their deterministic counterpart, partially ordered DFAs. Bouajjani et al. [7] characterized the class of languages accepted by partially ordered NFAs as languages effectively closed under permutation rewriting, that is, closed under the iterative application of rules of the form $ab \to ba$, and call this class of languages *Alphabetical Pattern Constraints*.

Schwentick et al. [70] further showed that *partially ordered two-way DFAs* coincide with the first-order fragment Δ_2, which consists of all languages where both the language and its complement are Σ_2-definable. Hence, it is the largest subclass of Σ_2 closed under complementation. This class of languages is also known as *unambiguous languages*, introduced by Schützenberger [69], who showed that unambiguous languages are exactly those languages whose syntactic monoid is in the variety DA; see also Grosshans et al. [26] for their relation to programs over monoids. Lodaya et al. [43] further characterized the languages of partially ordered two-way DFAs as a fragment of interval temporal logic.

Considering our contribution to the automata characterization of the discussed language classes, we defined *self-loop-deterministic partially ordered NFAs* and showed that they are expressively equivalent to partially ordered DFAs [39]. A partially ordered NFA is self-loop-deterministic if, in every state, the automaton has never a choice, under the same letter, between staying in the state or leaving the state. Furthermore, we defined and studied a nondeterministic counterpart of confluent partially ordered DFAs recognizing piecewise testable languages called *complete, confluent, and self-loop-deterministic partially ordered NFAs* (or ptNFAs for short) [49, 53].

There is a constant interest in automata characterizations of the levels of the Straubing-Thérien hierarchy (as well as of the fragments of first-order logic), in particular in decidability and complexity of checking membership of a language in a specific level of the hierarchy. Despite a recent progress [2, 60, 63], decidability of whether a language belongs to level k of the Straubing-Thérien hierarchy is open for $k > \frac{7}{2}$. The recent results were obtained by considering more general problems than membership, and the investigation has brought *separability* and *covering* problems into focus. We do not provide more details about these results and rather refer the reader to the literature [64].

Although we consider only $*$-languages in this paper, we now briefly mention related research for tree- and ω-languages. Héam [28] studied tree regular languages and showed that the class of tree regular languages accepted by Σ_2 formulae is strictly included in the class of languages accepted by partially ordered tree automata. Kufleitner and Lauser [40] extended the results of Schwentick et al. [70] from finite words to infinite words. They defined *partially ordered two-way Büchi automata* and characterized their expressivity in terms of first-order logic. Partially ordered Büchi automata can be used, *e.g.*, to characterize the common fragment of the two temporal logics ACTL and LTL [5, 45]. They showed that nondeterministic partially ordered two-way Büchi automata are equivalent to their one-way counterpart, and that their expressivity coincides with the first-order fragment Σ_2. Considering deterministic partially ordered two-way Büchi automata, they showed that they characterize the first-order fragment Δ_2, and that deterministic partially ordered two-way Büchi automata are more expressive than deterministic partially ordered one-way Büchi automata. Over finite words, the fragment Δ_2 coincides with the fragment FO^2 of first-order logic with only two variables [81], and the corresponding languages are exactly unambiguous languages [59]. The situation is, however, different over infinite words, where

the fragment Δ_2 is a proper subclass of FO^2, and only *restricted unambiguous languages* are Δ_2-definable. They further showed that deterministic partially ordered two-way Büchi automata are effectively closed under Boolean operations, and discussed the complexity of several problems for partially ordered two-way Büchi automata, including emptiness, inclusion, universality, and equivalence. The same problems were studied by Lodaya et al. [44] for partially ordered two-way automata over finite words and by Sistla et al. [75] for one-way Büchi automata.

We finally point out that there is also a kind of partially ordered alternating automata, motivated by applications in specification and verification of nonterminating programs. Namely, Kupferman and Vardi [41] discussed weak alternating automata of Muller et al. [56], where the state space is partitioned into partially ordered sets, and the automaton can move from a set only to a smaller set.

Our interest in partially ordered automata has several reasons. First, it comes from the supervisory control synthesis of discrete event systems [36, 37] and from the verification of properties of discrete-event systems [50, 54]. Given an automaton modelling a system (manufacturing, technological, etc.) and a regular language describing a specification, the aim of supervisory control is to automatically design a controller such that, running the controller and the system in a closed-loop feedback manner, the controlled system satisfies the prescribed specification. Although the goal of supervisory control synthesis is similar to that of reactive synthesis [27], there are significant differences between the approaches; interested readers are referred to the literature for details [20, 68]. For our interest, partially ordered automata are in some sense and on some level of abstraction the simplest models of deadlock-free discrete event systems, and hence convenient to study the lower-bound complexity of the problems under consideration, such as detectability, diagnosability, opacity, etc. [8, 11, 31, 42, 57, 65, 67, 72, 73].

Our second motivation comes from database theory and schema languages for XML data, namely from efficient approximate query answering and increasing the user-friendliness of XML Schema. Both problems are motivated by scenarios in which we want to describe something complex by means of a simple language. The technical core of these scenarios consists of *separation* problems, which are usually of the form "Given two languages K and L, does there exist a language S, coming from a family $\mathcal{F}$ of 'simple' languages, such that S contains everything from K and nothing from L?" The family $\mathcal{F}$ of simple languages could be, for example, languages definable by a first-order fragment, piecewise testable languages, languages definable by a special class of automata, etc. [17, 29, 46, 61, 62]. The separability technique is further closely related to *interpolation*, a method providing means to compute separation between good and bad states in program verification [16, 66].

Our third motivation comes from the evaluation of regular path queries in graph databases. Regular path queries are an important feature of modern query languages, such as SPARQL 1.1, allowing queries about arbitrarily long paths in the graph database. Regular path queries are regular expressions that are matched against labeled directed paths of a graph. The expressions are often of a specific and simple form [47, 48]. Our particular interest is in translating regular path queries to simple (*e.g.*, partially ordered) automata.

All of our motivations boil down to the complexity questions of basic operations, such as inclusion and equivalence. Since the universality question provides the lower-bound complexity for both inclusion and equivalence, we in particular focus on the complexity of deciding universality.

	$\lvert\Sigma\rvert = 1$	$\lvert\Sigma\rvert \geq 2$	Σ is growing
DFA	L-c [32]	NL-c [32]	NL-c [32]
ptNFA	NL-c (Thm. 3.1)	coNP-c (Thm. 3.2)	PSpace-c (Thm. 3.8)
rpoNFA	NL-c [39]	coNP-c [39]	PSpace-c [39]
poNFA	NL-c [39]	PSpace-c [39]	PSpace-c [1]
NFA	coNP-c [77]	PSpace-c [1]	PSpace-c [1]

Table 1: Complexity of deciding universality; Σ denotes the input alphabet.

Universality is a fundamental question asking whether a given system recognizes all words over its alphabet. The study of universality has a long tradition in formal languages with many applications across computer science, *e.g.*, in knowledge representation and database theory [4, 12, 76] or in verification [3]. Deciding universality for NFAs is PSpace-complete [55], and there are two typical proof techniques for showing hardness. One is based on the reduction from the *DFA-union-universality* problem [38] and the other on the reduction from the *word problem* for polynomially-space-bounded Turing machines [1]. Kozen's [38] proof showing PSpace-hardness of DFA-union universality (actually of its complemented equivalent – DFA-intersection emptiness) results in DFAs consisting of nontrivial cycles, and these cycles are essential for the proof; indeed, if all cycles of the DFAs were only self-loops, then the problem would be easier (namely, coNP-complete, see Theorem 3.3).

Deciding universality for partially ordered NFAs has the same worst-case complexity as for general NFAs, even if restricted to binary alphabets [39]. This could be caused by an unbounded number of nondeterministic steps admitted in partially ordered NFAs – a partially ordered NFA either stays in the same state or moves to another state. Forbidding this kind of nondeterminism, that is, considering self-loop-deterministic partially ordered NFAs, indeed affects the complexity of universality – it is coNP-complete if the alphabet is fixed, but remains PSpace-complete if the alphabet may grow polynomially with the number of states [39]. The growth of the alphabet thus, in some sense, compensates for the restricted number of nondeterministic steps.

In this paper, we study the complexity of deciding universality for ptNFAs – complete, confluent, and self-loop-deterministic partially ordered NFAs on finite words. Since we use a different definition of these automata in this paper than in our previous work, we first show that these two definitions coincide (Lemma 2.1). Then we show that deciding universality for ptNFAs is NL-complete if the input alphabet is unary (Theorem 3.1), coNP-complete if the input alphabet is fixed (Therem 3.2), and PSpace-complete in general (Therem 3.8). The proof of Therem 3.8 requires a novel and nontrivial extension of our recent construction for self-loop-deterministic partially ordered NFAs [39]. The proof is based on a construction of its own interest; namely, on a construction of a ptNFA accepting all but a single exponentially long word (Lemma 3.5). These results are summarized in Table 1. Then we show how to reduce universality to the question of whether the language of a given automaton is k-piecewise testable (Lemma 4.1), and we use this result to obtain the complexity results for deciding k-piecewise testability (Theorems 4.2 through 4.8); see Table 2 for an overview of the results. After that, we study the question whether the language of a given automaton is piecewise testable, that is, without the restriction to a specific k (Theorems 5.1 through 5.6); see Table 3 for an overview of the results. Finally, we discuss the complexity of the problems of inclusion and equivalence in Section 6. These results are summarized in Tables 4 and 5.

	Unary alphabet $\lvert\Sigma\rvert = 1$	Fixed alphabet $\lvert\Sigma\rvert \geq 2$	Arbitrary alphabet $k \leq 3$	$k \geq 4$
DFA	L-c (Thm. 4.7)	NL-c (Thm. 4.4)	NL-c [53]	coNP-c [34]
ptNFA	NL-c (Thm. 4.6)	coNP-c (Thm. 4.3)	PSpace-c (Thm. 4.2)	
rpoNFA	NL-c	coNP-c [39]	PSpace-c	
poNFA	NL-c (Thm. 4.6)	PSpace-c (Thm. 4.5)	PSpace-c	
NFA	coNP-c (Thm. 4.8)	PSpace-c [53]	PSpace-c [53]	

Table 2: Complexity of deciding k-piecewise testability.

	$\lvert\Sigma\rvert = 1$	$\lvert\Sigma\rvert \geq 2$	Σ is growing
DFA	L-c (Thm. 5.2)	NL-c [13]	NL-c [13]
rpoNFA	✓ (Thm. 5.1)	coNP-c (Thm. 5.6)	PSpace-c (Thm. 5.5)
poNFA	✓ (Thm. 5.1)	PSpace-c (Thm. 5.4)	PSpace-c
NFA	coNP-c (Thm. 5.3)	PSpace-c [51]	PSpace-c [51]

Table 3: Complexity of deciding piecewise testability.

	B			
A	DFA	ptNFA & rpoNFA	poNFA	NFA
DFA	L/NL	NL/coNP/PSpace	NL/PSpace	coNP/PSpace
ptNFA	NL	NL/coNP/PSpace	NL/PSpace	coNP/PSpace
rpoNFA	NL	NL/coNP/PSpace	NL/PSpace	coNP/PSpace
poNFA	NL	NL/coNP/PSpace	NL/PSpace	coNP/PSpace
NFA	NL	NL/coNP/PSpace	NL/PSpace	coNP/PSpace

Table 4: Complexity of deciding inclusion $L(A) \subseteq L(B)$ (unary/fixed[/growing] alphabet), all results are complete for the given class.

	DFA	ptNFA & rpoNFA	poNFA	NFA
DFA	L/NL	NL/coNP/PSpace	NL/PSpace	coNP/PSpace
ptNFA		NL/coNP/PSpace	NL/PSpace	coNP/PSpace
rpoNFA		NL/coNP/PSpace	NL/PSpace	coNP/PSpace
poNFA			NL/PSpace	coNP/PSpace
NFA				coNP/PSpace

Table 5: Complexity of deciding equivalence (unary/fixed[/growing] alphabet), the problems are complete for the given classes.

2. Preliminaries

2.1. **Basic Definitions.** We assume that the reader is familiar with automata and formal language theory [1]. The cardinality of a set A is denoted by $|A|$ and the power set of A by 2^A. The empty word is denoted by ε. For a word $w = xyz$, x is a *prefix*, y a *factor*, and z a *suffix* of w. Let Σ be an alphabet, and let $L_{a_1a_2\cdots a_n} = \Sigma^* a_1 \Sigma^* a_2 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$, where $a_i \in \Sigma$ for $i = 1, \ldots, n$. A word v is a *subword* of a word w, denoted by $v \preccurlyeq w$, if $w \in L_v$. A prefix (factor, suffix, subword) of w is *proper* if it is different from w.

A *nondeterministic finite automaton* (NFA) is a quintuple $\mathcal{A} = (Q, \Sigma, \delta, I, F)$, 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 $\delta\colon Q \times \Sigma \to 2^Q$ is the transition function that can be extended to the domain $2^Q \times \Sigma^*$ in the usual way. The language *accepted* by $\mathcal{A}$ is the set $L(\mathcal{A}) = \{w \in \Sigma^* \mid \delta(I, w) \cap F \neq \emptyset\}$. The automaton $\mathcal{A}$ is *complete* if for every state $q \in Q$ and every letter $a \in \Sigma$, the set $\delta(q, a)$ is nonempty, and it is *deterministic* (DFA) if $|I| = 1$ and $|\delta(q, a)| = 1$ for every state $q \in Q$ and every letter $a \in \Sigma$.

A *path* π from a state q_0 to a state q_n under a word $a_1 a_2 \cdots a_n$, for some $n \geq 0$, is a sequence of states and input symbols $q_0 a_1 q_1 a_2 \ldots q_{n-1} a_n q_n$ where $q_{i+1} \in \delta(q_i, a_{i+1})$ for all $i = 0, 1, \ldots, n-1$. Path π is *accepting* if $q_0 \in I$ is an initial state and $q_n \in F$ is an accepting state. We write $q_0 \xrightarrow{a_1 a_2 \cdots a_n} q_n$ to denote that there exists a path from q_0 to q_n under the word $a_1 a_2 \cdots a_n$. A path is *simple* if all its states are pairwise distinct. The number of states on the longest simple path of $\mathcal{A}$ that starts in an initial state, decreased by one (*i.e.*, the number of transitions on that path), is the *depth* of $\mathcal{A}$, denoted by $\mathrm{depth}(\mathcal{A})$.

2.2. **Partially Ordered Automata.** Let $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ be an NFA. The reachability relation $\leq$ on states is defined by setting $p \leq q$ if there is a word $w \in \Sigma^*$ such that $q \in \delta(p, w)$. The NFA $\mathcal{A}$ is *partially ordered (poNFA)* if the reachability relation $\leq$ is a partial order.

The NFA $\mathcal{A}$ is *confluent* provided that for every state $q \in Q$ and every pair of (not necessarily distinct) letters $a, b \in \Sigma$, if $s \in \delta(q, a)$ and $t \in \delta(q, b)$ then there exists a word $w \in \{a, b\}^*$ such that $\delta(s, w) \cap \delta(t, w) \neq \emptyset$; see Figure 1 (left) for an illustration.

The poNFA $\mathcal{A}$ is *restricted* or *self-loop-deterministic* (*rpoNFA*) if, for every state $q \in Q$ and every letter $a \in \Sigma$, $q \in \delta(q, a)$ implies that $\delta(q, a) = \{q\}$. Restricted poNFAs can thus be defined in terms of forbidden patterns similar to that of Glaßer and Schmitz [24]; Figure 1 (right) shows the forbidden pattern of rpoNFAs.

An rpoNFA is a *ptNFA* if it is complete and confluent. The name ptNFA comes from *piecewise testable*, since ptNFAs characterize piecewise testable languages [49, 53]. Notice that the disjoint union of two or more ptNFAs is a ptNFA; we use this fact in the proof of Theorem 3.8. The violation of the definitional properties of ptNFAs can be tested by several reachability checks, and these properties are NL-hard even for minimal DFAs [13]. Since NL = coNL, checking whether an NFA is a ptNFA is NL-complete.

Figure 1: Confluence (left) and the forbidden pattern of rpoNFAs (right).

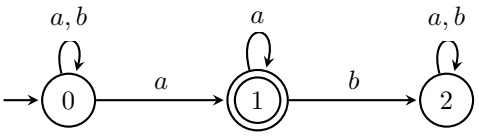

Figure 2: A confluent automaton $\mathcal{B}$ accepting a non-piecewise testable language.

2.3. **The Unique Maximal State Property.** For two states p and q, we write $p < q$ if $p \leq q$ and $p \neq q$. A state p is *maximal* if there is no state q such that $p < q$.

A poNFA $\mathcal{A}$ over Σ with the state set Q can be turned into a directed graph $G(\mathcal{A})$ with the set of vertices Q where a pair $(p, q) \in Q \times Q$ is an edge in $G(\mathcal{A})$ if there is a transition from p to q in $\mathcal{A}$. For an alphabet $\Gamma \subseteq \Sigma$, we define the directed graph $G(\mathcal{A}, \Gamma)$ with the set of vertices Q by considering only those transitions corresponding to letters in Γ. Let $\Sigma(p) = \{a \in \Sigma \mid p \xrightarrow{a} p\}$ denote all letters labeling self-loops in state p. We say that $\mathcal{A}$ satisfies the *unique maximal state* (UMS) property if, for every state q of $\mathcal{A}$, q is the unique maximal state of the connected component of $G(\mathcal{A}, \Sigma(q))$ containing q.

To decide whether, for a given DFA, there exists an equivalent confluent poDFA, or, said differently, whether the given DFA recognizes a piecewise testable language, Klíma and Polák [35] check the confluence property of the minimal DFA while Trahtman [82] checks the UMS property. Both properties have their advantages and an effect on algorithmic complexity. While Trahtman's algorithm runs in quadratic time with respect to the number of states and in linear time with respect to the size of the alphabet, Klíma and Polák's algorithm swaps the complexities; it runs in linear time with respect to the number of states and in quadratic time with respect to the size of the alphabet. From the computational complexity view, the problem is NL-complete [13].

Although the UMS property and the confluence property coincide on DFAs, they differ on NFAs. Consider the automaton $\mathcal{B}$ in Figure 2. One can verify that $\mathcal{B}$ is confluent. However, $\mathcal{B}$ does not satisfy the UMS property. Indeed, for state 0, the connected component is $G(\mathcal{B}, \Sigma(0)) = G(\mathcal{B}, \{a, b\}) = G(\mathcal{B})$, *i.e.*, the component is the whole automaton. Therefore, state 0 is not a maximal state of this component; in particular, it is not the unique maximal state. Furthermore, notice that $\mathcal{B}$ accepts a language that is not piecewise testable. Indeed, there is an infinite sequence of words $a, ab, aba, abab, \ldots$ where accepted words alternate with non-accepted words. The corresponding minimal DFA therefore must have a nontrivial cycle that contains at least one accepting and one non-accepting state. This, however, means that the minimal DFA is not partially ordered, and hence it cannot recognize a piecewise testable language. In summary, we find that the confluence property on its own is not suitable for characterizing piecewise testable languages for poNFAs. Intuitively, the problem comes from the presence of the forbidden pattern of rpoNFAs (see Figure 1), which may fool confluence but not the UMS property.

Recall that ptNFAs are complete and confluent rpoNFAs. The next lemma gives an alternative characterization of ptNFAs as poNFAs that are complete and satisfy the UMS property, which was used as the primary definition of ptNFAs in our previous work [49].

Lemma 2.1. *Partially ordered NFAs that are complete and satisfy the UMS property are exactly ptNFAs.*

Proof. First, we show that if $\mathcal{A}$ is partially ordered, complete, and satisfies the UMS property, then $\mathcal{A}$ is a ptNFA, *i.e.*, that $\mathcal{A}$ is a complete and confluent rpoNFA. Indeed, the presence of the forbidden pattern of rpoNFAs violates the UMS property, *cf.* Figure 2. Therefore, $\mathcal{A}$ is a

complete rpoNFA. It remains to show that $\mathcal{A}$ is confluent. To this aim, let r be a state of $\mathcal{A}$, and let a and b be letters of the alphabet of $\mathcal{A}$ ($a = b$ not excluded) such that $r \xrightarrow{a} s$, $r \xrightarrow{b} t$, and $s \neq t$. Let s' and t' be any maximal states reachable from s and t in the component $G(\mathcal{A}, \{a, b\})$, respectively. Since $\{a, b\} \subseteq \Sigma(s')$ by the definition of s' and the assumption that $\mathcal{A}$ is complete, t' is also in the component $G(\mathcal{A}, \Sigma(s'))$ containing s'. However, by the UMS property of $\mathcal{A}$, s' is the unique maximal state of that component, and therefore there must be a path from t' to s' under $\Sigma(s')$. Similarly, interchanging s' and t', there must be a path from s' to t' under $\Sigma(t')$. Since $\mathcal{A}$ is partially ordered, the paths give that $t' \leq s'$ and $s' \leq t'$, *i.e.*, $s' = t'$. To show that $\mathcal{A}$ is confluent, let $u \in \{a, b\}^*$ be a word labeling the path from s to s' in $G(\mathcal{A}, \{a, b\})$. Since $\mathcal{A}$ is complete, there is a path from state t under u to a state p. Let $v \in \{a, b\}^*$ be a word labeling a path from p to a maximal state, p', in $G(\mathcal{A}, \{a, b\})$; such a path exists, since $\mathcal{A}$ is complete and partially ordered. Let $w = uv$, and notice that v can be read in s'. Then we have that $s \xrightarrow{w} s'$, $t \xrightarrow{w} p'$, and, from the above, since s' and t' were arbitrary, we have that $s' = p'$. Thus, $\mathcal{A}$ is confluent.

To prove the other direction, assume that $\mathcal{A}$ is a ptNFA, *i.e.*, that $\mathcal{A}$ is a complete and confluent rpoNFA. Obviously, $\mathcal{A}$ is a complete poNFA, and it remains to show that $\mathcal{A}$ satisfies the UMS property. For the sake of contradiction, suppose that $\mathcal{A}$ does not satisfy the UMS property, *i.e.*, that there is a state q in $\mathcal{A}$ such that the component $G(\mathcal{A}, \Sigma(q))$ containing q has at least two different maximal states. Let r be a biggest state with respect to the partial order on states of that component such that at least two different maximal states, say $s \neq t$, are reachable from r under the alphabet $\Sigma(q)$; such a state exists by assumption. Obviously, $r \notin \{s, t\}$. Let $s' \in \delta(r, a)$ and $t' \in \delta(r, b)$ be two different states on the paths from r to s and from r to t, respectively, for some letters $a, b \in \Sigma(q) \setminus \Sigma(r)$; notice that such letters a and b exist because $\mathcal{A}$ is an rpoNFA, and that $a = b$ is not excluded. Then $r < s'$ and $r < t'$. Since $\mathcal{A}$ is confluent, there exists r' such that $r' \in \delta(s', w) \cap \delta(t', w)$, for some word $w \in \{a, b\}^*$. Let r'' denote a maximal state that is reachable from r' in $G(\mathcal{A}, \Sigma(q))$. Then there are three cases: (i) if $r'' = s$, then $r < t'$ and both s and t are reachable from t' under $\Sigma(q)$, which is a contradiction with the choice of r as the biggest state of the component with this property; (ii) $r'' = t$ yields a contradiction with the choice of r as in (i) by interchanging t' and s'; (iii) similarly, $r'' \notin \{s, t\}$ yields a contradiction with the choice of r, since $r < s'$ and $r < t'$ and, *e.g.*, s and r'' are two different maximal states of the component $G(\mathcal{A}, \Sigma(q))$ containing q that are both reachable from $s' > r$. Thus, $\mathcal{A}$ satisfies the UMS property, which completes the proof. □

3. Complexity of Deciding Universality for ptNFAs

We now study the complexity of deciding universality for ptNFAs. As mentioned before, these automata are known to characterize the piecewise testable languages. Our results of this section in the context of existing results are summarized in Table 6.

For unary alphabets, deciding universality for ptNFAs is solvable in polynomial time [39]. We now improve the result and show that the problem is NL-complete.

Theorem 3.1. *Deciding universality for ptNFAs over a unary alphabet is* NL*-complete.*

Proof. The problem is in NL even for unary poNFAs [39]. To prove hardness, we reduce the DAG-reachability problem [32]. Let G be a directed acyclic graph with n nodes, and let s and t be two nodes of G such that there is no edge from t. We define a ptNFA $\mathcal{A}$ as follows.

	ST	$\lvert\Sigma\rvert = 1$	$\lvert\Sigma\rvert \geq 2$	Σ is growing
DFA		L-c [32]	NL-c [32]	NL-c [32]
ptNFA	1	NL-c (Thm. 3.1)	coNP-c (Thm. 3.2)	PSpace-c (Thm. 3.8)
rpoNFA		NL-c [39]	coNP-c [39]	PSpace-c [39]
poNFA	$\frac{3}{2}$	NL-c [39]	PSpace-c [39]	PSpace-c [1]
NFA		coNP-c [77]	PSpace-c [1]	PSpace-c [1]

Table 6: Complexity of deciding universality; ST stands for the corresponding level of the Straubing-Thérien hierarchy; Σ denotes the input alphabet.

With each node of G, we associate a state in $\mathcal{A}$. Whenever there is an edge from i to j in G, we add a transition $i \xrightarrow{a} j$ to $\mathcal{A}$. The initial state of $\mathcal{A}$ is s and all states are accepting. The automaton is obviously an rpoNFA, because there are no (self-)loops. To make it complete and confluent, we add $n-1$ new non-accepting states $f_1, \ldots, f_{n-1}$ together with transitions $f_i \xrightarrow{a} f_{i+1}$, for $i = 1, \ldots, n-2$, $f_{n-1} \xrightarrow{a} t$, $t \xrightarrow{a} t$, and, for every state $q \notin \{t, f_1, \ldots, f_{n-1}\}$, we add the transition $q \xrightarrow{a} f_1$. The resulting automaton is a ptNFA.

We now show that $\mathcal{A}$ is universal if and only if t is reachable from s in G. If t is reachable from s in G, then $L(\mathcal{A}) = \{a\}^*$, since t is reachable from s via states corresponding to nodes of G, which are all accepting in $\mathcal{A}$. If t is not reachable from s in G, then t is reachable from s in $\mathcal{A}$ via a path $s \xrightarrow{a^k} q \xrightarrow{a} f_1 \xrightarrow{a} f_2 \xrightarrow{a} \ldots \xrightarrow{a} f_{n-1} \xrightarrow{a} t$ for any state q corresponding to a node of G different from t reachable from s in G. We show that a^{n-1} does not belong to $L(\mathcal{A})$. The shortest path from state s to state t in $\mathcal{A}$ is of length n for $q = s$. Thus, any word accepted in t is of length at least n. On the other hand, every word accepted in a state corresponding to a node of G different from t is of length at most $n-2$, since there are $n-1$ such states and $\mathcal{A}$ is acyclic on those states. This gives that a^{n-1} is not accepted by $\mathcal{A}$, and hence $L(\mathcal{A})$ is not universal. □

We next show that if the alphabet is fixed, deciding universality for ptNFAs is coNP-complete, and that hardness holds even if restricted to binary alphabets. Our proof is based on a construction that shows non-equivalence of regular expressions with union and concatenation to be NP-complete, even if one of the expressions has the form Σ^n for some fixed n [30, 77].

Theorem 3.2. *Deciding universality for ptNFAs over a fixed alphabet is* coNP*-complete even if the alphabet is binary.*

Proof. Membership follows from the membership for rpoNFAs [39, Corollary 24]. To show hardness, we reduce DNF validity[1]. Let $U = \{x_1, \ldots, x_n\}$ be a set of variables and $\varphi = \varphi_1 \vee \cdots \vee \varphi_m$ be a formula in DNF, where every φ_i is a conjunction of literals. Without loss of generality, we may assume that no φ_i contains both x and $\neg x$. To illustrate our

[1] A *(boolean) formula* is built from propositional variables; operators conjunction, disjunction, and negation; and parentheses. A formula is *valid* if it is true for every assignment of 1 (*true*) and 0 (*false*) to its variables. A *literal* is a variable or its negation. A formula is in *disjunctive normal form* (DNF) if it is a disjunction of one or more conjunctions of literals; *e.g.*, $\varphi = (x \wedge y \wedge z) \vee (\neg x \wedge y \wedge z)$ is a formula in DNF consisting of two conjunctions $x \wedge y \wedge z$ and $\neg x \wedge y \wedge z$. Given a formula in DNF, the DNF validity problem asks whether the formula is valid. The formula φ is not valid; it is not true for, *e.g.*, $(x, y, z) = (0, 1, 0)$.

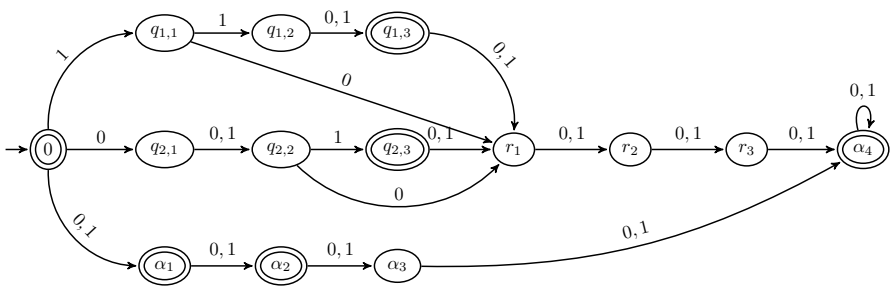

Figure 3: The ptNFA for the formula $(x \wedge y) \vee (\neg x \wedge z)$ from the proof of Theorem 3.2.

construction, we use the formula $(x \wedge y) \vee (\neg x \wedge z)$ over three variables x, y, z. For every $i = 1, \ldots, m$, we define a regular expression $\beta_i = \beta_{i,1} \ldots \beta_{i,n}$, where

$$\beta_{i,j} = \begin{cases} 0+1 & \text{if neither } x_j \text{ nor } \neg x_j \text{ appear in } \varphi_i \\ 0 & \text{if } \neg x_j \text{ appears in } \varphi_i \\ 1 & \text{if } x_j \text{ appears in } \varphi_i \end{cases}$$

for $j = 1, \ldots, n$. For our formula $(x \wedge y) \vee (\neg x \wedge z)$, we obtain $\beta_1 = 11(0+1)$ and $\beta_2 = 0(0+1)1$. We now define a regular expression $\beta = \sum_{i=1}^{m} \beta_i$ as the alternative of all expressions β_i. Then $w \in L(\beta)$ if and only if w corresponds to a truth assignment that satisfies some φ_i. That is, $L(\beta) = \{0,1\}^n$ if and only if φ is valid. By construction, the length of each β_i is proportional to n.

We now construct a ptNFA $\mathcal{M}$ as follows (the transitions are the minimal sets satisfying the definitions). The initial state of $\mathcal{M}$ is state 0. For every β_i, we construct a deterministic path consisting of $n+1$ states $\{q_{i,0}, q_{i,1}, \ldots, q_{i,n}\}$ and transitions $q_{i,\ell} \xrightarrow{\beta_{i,\ell+1}} q_{i,\ell+1}$, for $0 \leq \ell < n$, $q_{i,0} = 0$, and $q_{i,n}$ is accepting, where, for $\beta_{i,\ell+1} = 0+1$, $q_{i,\ell} \xrightarrow{\beta_{i,\ell+1}} q_{i,\ell+1}$ denotes two transitions: $q_{i,\ell} \xrightarrow{0} q_{i,\ell+1}$ and $q_{i,\ell} \xrightarrow{1} q_{i,\ell+1}$. This ensures that $\mathcal{M}$ accepts $L(\beta)$. The overall construction for our formula $(x \wedge y) \vee (\neg x \wedge z)$ is illustrated in Figure 3.

To accept all words of length different from n, we add $n+1$ states $\{\alpha_1, \alpha_2, \ldots, \alpha_{n+1}\}$ and transitions $\alpha_\ell \xrightarrow{a} \alpha_{\ell+1}$, for $0 \leq \ell < n+1$, and $\alpha_{n+1} \xrightarrow{a} \alpha_{n+1}$, where $a \in \{0,1\}$, $\alpha_0 = 0$, and $\{\alpha_0, \alpha_1, \ldots \alpha_{n+1}\} \setminus \{\alpha_n\}$ are accepting.

Finally, to make the automaton complete and confluent, we add n non-accepting states $\{r_1, \ldots, r_n\}$ and transitions $r_i \xrightarrow{a} r_{i+1}$, for $1 \leq i < n$, and $r_n \xrightarrow{a} \alpha_{n+1}$, where $a \in \{0,1\}$. These states are used to complete $\mathcal{M}$ by adding a transition from every state q to r_1 under a if a is not defined in q. Then completeness and the fact that state α_{n+1} is reachable from every state implies confluence of the automaton. Notice that the states $r_1, \ldots, r_n$ and the corresponding transitions do not accept any word of length n that does not belong to $L(\beta)$.

Altogether, the accepting states of $\mathcal{M}$ are states $\{0\} \cup \{q_{1,n}, \ldots, q_{m,n}\} \cup \{\alpha_1, \ldots \alpha_{n+1}\} \setminus \{\alpha_n\}$. Thus, $\mathcal{M}$ is a ptNFA accepting the language $L(\mathcal{M}) = L(\beta) \cup \{w \in \{0,1\}^* \mid |w| \neq n\}$, and hence $L(\mathcal{M}) = \{0,1\}^*$ if and only if $L(\beta) = \{0,1\}^n$, which is if and only if φ is valid. □

If the alphabet may grow polynomially with the number of states, there are basically two approaches how to tackle the universality problem for ptNFAs to show PSPACE-hardness: (1) to use a reduction from Kozen's DFA-union-universality problem [38], or (2) to use a reduction from the word problem of polynomially-space-bounded Turing machines à la Aho, Hopcroft and Ullman [1]. We discuss these two options in the following two subsections. In

the first subsection, we show that the partially ordered variant of the DFA-union-universality problem is easier than its general counterpart, and hence not suitable to prove PSPACE-hardness of deciding universality for ptNFAs. Therefore, in the second subsection, we use the reduction from the word problem of a polynomially-space-bounded Turing machine to prove the result.

3.1. **Partially Ordered DFA Union Universality.** To use the union-universality problem for our purposes, we would need to use partially ordered DFAs (poDFAs) rather than general DFAs to ensure that the union of the DFAs is partially ordered. We now show that the difficulty of the DFA-union-universality problem comes from nontrivial cycles, and hence its partially-ordered variant is easier unless PSPACE = NP.

We consider the complemented equivalent of the problem – the *DFA-intersection emptiness*. It asks, given n DFAs, whether the intersection of their languages is empty. Clearly, the union of n DFA languages is universal if and only if the intersection of their complements is empty.

Theorem 3.3. *The intersection-emptiness problem for poDFAs/poNFAs is* coNP*-complete. It is* coNP*-hard even if the alphabet is binary.*[2]

Proof. We show membership in coNP for poNFAs and coNP-hardness for poDFAs.

Let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be poNFAs and assume that $w \in \bigcap_{i=1}^{n} L(\mathcal{A}_i)$. Let k_i be the depth of $\mathcal{A}_i$ and consider a fixed path of $\mathcal{A}_i$ accepting w. Along this path, we mark (at most) k_i letters of w that cause the change of state of $\mathcal{A}_i$. Doing this for all of the n automata, we mark at most $k_1 + k_2 + \cdots + k_n$ letters in w. Since all non-marked letters of w correspond to self-loops in the automata, we can remove them to obtain a subword w' of w accepted by every $\mathcal{A}_i$ that is of length at most $\sum_{i=1}^{n} k_i$, which is polynomial in the size of the input. Thus, if the intersection is nonempty, there is a polynomial certificate. Therefore, the intersection-emptiness problem is in coNP.

To show hardness, we reduce DNF validity. Let φ be a formula in DNF with n variables and m conjunctions of literals. For $i = 1, \ldots, m$, we define an expression β_i as in the proof of Theorem 3.2. Since every β_i represents a finite language, we construct a poDFA recognizing $L(\beta_i)$ and take its complement, denoted by $\mathcal{A}_i$, which is also a poDFA. Then $\{0,1\}^n \cap \bigcap_{i=1}^{m} L(\mathcal{A}_i) = \emptyset$ if and only if $\bigcup_{i=1}^{m} L(\beta_i) = \{0,1\}^n$, which is if and only if φ is valid. □

Thus, we have the following corollary.

Corollary 3.4. *The poDFA-union-universality problem is* coNP*-complete.* □

We point out that the previous result cannot be further strengthened to the case of unary alphabets, since the intersection-emptiness problem for unary poNFAs can be solved in polynomial time. Indeed, if there is a word in the intersection $\bigcap_{i=1}^{n} L(\mathcal{A}_i)$, then it is a prefix of the word $a^{k_1+\cdots+k_n}$, where the k_i's are as in the proof of Theorem 3.3, and this word is of polynomial length.

[2]We thank G. Zetzsche and O. Klíma, who suggested the membership proof.

3.2. **Complexity of Deciding Universality for ptNFAs.** In this section, we show that the universality problem for ptNFAs is PSPACE-complete if the alphabet may grow polynomially with the number of states of the automaton. The proof is a novel and nontrivial extension of our recent proof showing a similar result for self-loop-deterministic poNFAs [39].

To prove our result, we use the idea of Aho, Hopcroft and Ullman [1] to take, for a polynomial p, a p-space-bounded deterministic Turing machine $\mathcal{M}$ together with an input x, and to encode the computations of $\mathcal{M}$ on x as words over some alphabet Σ, where Σ depends on $\mathcal{M}$. One then constructs a regular expression (or an NFA) R_x representing all computations that do not encode an accepting run of $\mathcal{M}$ on x. That is, $L(R_x) = \Sigma^*$ if and only if $\mathcal{M}$ does not accept x [1].

The form of R_x is relatively simple, consisting of a union of expressions of the form

$$\Sigma^* \, K \, \Sigma^* \tag{3.1}$$

where K is a finite language of words of length $O(p(|x|))$. Intuitively, K encodes possible "violations" of a correct computation of $\mathcal{M}$ on x, such as the initial configuration does not contain the input x, or the step from a configuration to the next one does not correspond to a rule of $\mathcal{M}$. These checks are local, involving at most two consecutive configurations of $\mathcal{M}$, each of polynomial size. Hence they can be encoded as the finite language K. The initial segment Σ^* of (3.1) nondeterministically guesses a position of the computation where a violation encoded by K occurs, and the last Σ^* reads the rest of the word if the violation check was successful. However, this idea cannot be directly used to prove our result for two reasons:

(1) Although expression (3.1) can easily be translated to a poNFA, it is not true for ptNFAs because the translation of the leading part Σ^*K may not be self-loop-deterministic;
(2) The constructed poNFA may be incomplete and its "standard" completion by adding the missing transitions to a new sink state may violate confluence.

We addressed problem (1) in our previous work [39] where we proved PSPACE-hardness of deciding universality for rpoNFAs. However, the constructed rpoNFA is not a ptNFA, and because of different expressive powers, it is not always possible to complete an rpoNFA to obtain a ptNFA. To solve problem (2), we use an observation that the length of the encoding of a computation of $\mathcal{M}$ on x is at most exponential with respect to the size of $\mathcal{M}$ and x, and hence we could use an exponentially long word to make the automaton confluent in a similar way as we did in our previous constructions above. Since such a word cannot be constructed by a polynomial-time reduction, we need to encode it by a ptNFA, which exists and is of polynomial size as we show in Lemma 3.5 – there we construct, in polynomial time, a ptNFA $\mathcal{A}_{n,n}$ that accepts all words but a single one, $W_{n,n}$, of exponential length. The automaton consists of two parts (the upper and lower parts in Figure 4). The upper part (together with state *max*) is the rpoNFA we used to tackle problem (1) in our previous work [39]. The lower part not only makes the rpoNFA complete and confluent, but it also encodes an exponentially long word that we use to tackle problem (2).

In our proof we do not get the same language as defined by the regular expression R_x, but the language of the constructed ptNFA is universal if and only if the language of R_x is, which suffices for our reduction.

It is not hard to see that an automaton that is complete and has a single maximal state reachable from every state must also be confluent. We use this fact to simplify the proof.

Thus, the first step of the construction is to construct the ptNFA $\mathcal{A}_{n,n}$, which accepts all words but the single word $W_{n,n}$ of exponential length. This automaton is the core of

the proof. The considered language is the same as in our previous work [39, Lemma 17], where the constructed automaton is an rpoNFA that is not a ptNFA. As already pointed out above, that rpoNFA is formed by the upper part of Figure 4 together with state *max*, *cf.* Corollary 3.7. In the following lemma, we present the basic idea how to transform this rpoNFA into a ptNFA. This idea is used and further developed in the proof of Theorem 3.8. On an intuitive level, the states of the lower part of Figure 4, except for state *max*, are used to make the rpoNFA (as well as the rpoNFAs constructed in the proof of Theorem 3.8) complete and confluent, and hence a ptNFA. Recall that because of a weaker expressivity of ptNFAs, this transformation is not possible in general.

Lemma 3.5. *For all integers $k, n \geq 1$, there exists a ptNFA $\mathcal{A}_{k,n}$ over an n-letter alphabet with $n(2k+1)+1$ states, such that the unique non-accepted word of $\mathcal{A}_{k,n}$ is of length $\binom{k+n}{k}-1$.*

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

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

The length of $W_{k,n}$ is $\binom{k+n}{n} - 1$ [53]. Notice that the letter a_n appears exactly k times in $W_{k,n}$. We further set $W_{k,n} = \varepsilon$ whenever $kn = 0$, since this is useful for defining $\mathcal{A}_{k,n}$ below.

We construct a ptNFA $\mathcal{A}_{k,n}$ 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 a DFA for $\{a_1\}^* \setminus \{a_1^k\}$ with k additional unreachable states used to address confluence of problem (2) and included here for uniformity (see Corollary 3.6). Thus, $\mathcal{A}_{k,1}$ consists of $2k+1$ states of the form $(i;1)$ and a state *max* together with the given a_1-transitions, see Figure 4 for an illustration. All states but $(i;1)$, for $i = k, \ldots, 2k$, are accepting, and $(0;1)$ is initial. All undefined transitions in Figure 4 go to state *max*.

Given a ptNFA $\mathcal{A}_{k,n-1}$, we recursively construct $\mathcal{A}_{k,n}$ as defined next. The construction for $n = 3$ is illustrated in Figure 4. We obtain $\mathcal{A}_{k,n}$ from $\mathcal{A}_{k,n-1}$ by adding $2k+1$ states $(0;n), (1;n), \ldots, (2k;n)$, where $(0;n)$ is added to the initial states, and all states $(i;n)$ with $i < k$ are added to the accepting states. The automaton $\mathcal{A}_{k,n}$ therefore has $n(2k+1)+1$ states. The additional transitions of $\mathcal{A}_{k,n}$ consist of the following groups:

(1) Self-loops $(i;n) \xrightarrow{a_j} (i;n)$ for $i \in \{0, 1, \ldots, 2k\}$ and $a_j = a_1, a_2, \ldots, a_{n-1}$;
(2) Transitions $(i;n) \xrightarrow{a_n} (i+1;n)$ for $i \in \{0, 1, \ldots, 2k-1\} \setminus \{k\}$;
(3) Transitions $(k,n) \xrightarrow{a_n} max$, $(2k,n) \xrightarrow{a_n} max$, and the self-loop $max \xrightarrow{a_n} max$;
(4) Transitions $(i;n) \xrightarrow{a_n} (i+1;m)$ for $i = 0, 1, \ldots, k-1$ and $m = 1, \ldots, n-1$;
(5) Transitions $(i;m) \xrightarrow{a_n} max$ for every accepting state $(i;m)$ of $\mathcal{A}_{k,n-1}$;
(6) Transitions $(i;m) \xrightarrow{a_n} (k+1,n)$ for every non-accepting state $(i;m)$ of $\mathcal{A}_{k,n-1}$.

By construction, $\mathcal{A}_{k,n}$ is complete and partially ordered. It satisfies the UMS property because if there is a self-loop in a state $q \neq max$ under a letter a, then there is no other incoming or outgoing transition of q under a. This means that the component of the graph $G(\mathcal{A}_{k,n}, \Sigma(q))$ containing q is only state q, which is indeed the unique maximal state. Hence, it is a ptNFA.

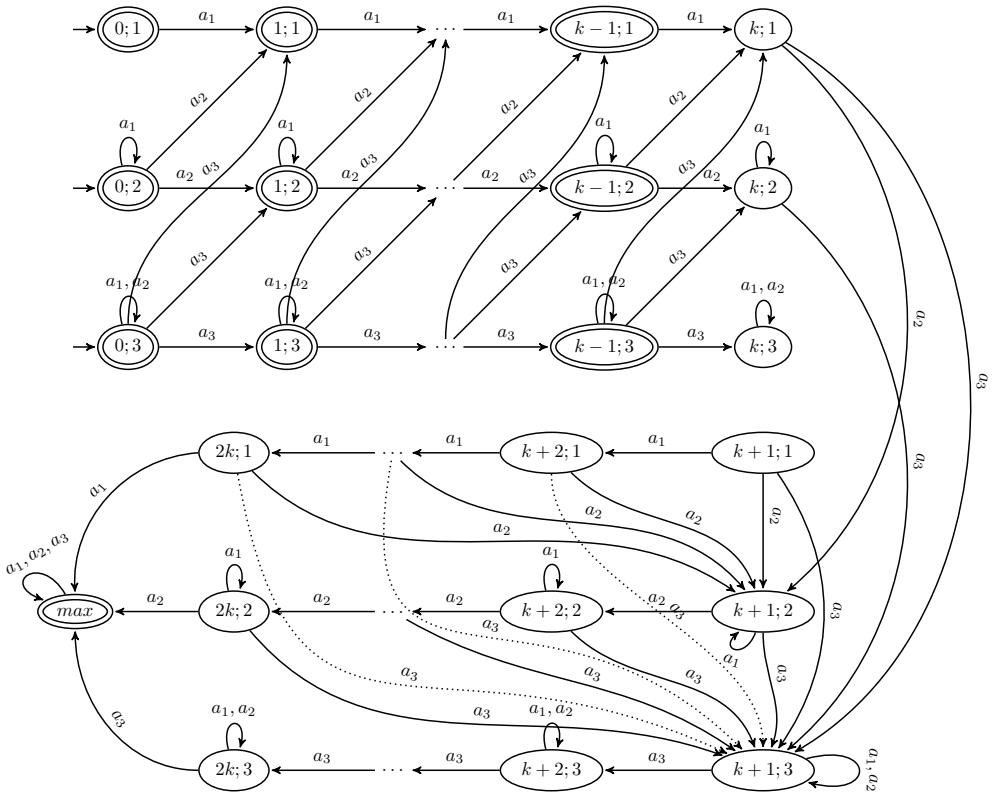

Figure 4: The ptNFA $\mathcal{A}_{k,3}$ with $3(2k+1)+1$ states; all undefined transitions go to state *max*; for readability, transitions $(k+i;1) \xrightarrow{a_3} (k+1;3)$ for $i = 2, 3, \ldots, k$, are dotted.

We show that $\mathcal{A}_{k,n}$ accepts $\Sigma_n^* \setminus \{W_{k,n}\}$. The transitions (1), (2), and (3) ensure acceptance of every word that does not contain exactly k occurrences of a_n. The transitions (4) and (5) ensure acceptance of all words in $(\Sigma_{n-1}^* a_n)^i L(\mathcal{A}_{k-i,n-1}) a_n \Sigma_n^*$, for which the longest factor before the $(i+1)$th occurrence of a_n is not of the form $W_{k-i,n-1}$, and hence is not a correct factor of $W_{k,n} = W_{k,n-1}a_n \cdots a_n W_{k-i,n-1}a_n \cdots a_n W_{1,n-1}a_n$. Together, these transitions ensure that $\mathcal{A}_{k,n}$ accepts every input other than $W_{k,n}$. Notice that transitions (6) are not needed to obtain the language, and we show below that they do not change the language. This corresponds to the informal explanation that the lower part of the automaton is used to take care of confluence and has no effect on the language. We formulate this precisely as Corollary 3.7.

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 $W_{k,1} = a_1^k$ is not accepted by $\mathcal{A}_{k,1}$ (*cf.* Figure 4).

For the inductive case $(k,n) \geq (1,2)$, assume that $\mathcal{A}_{k',n'}$ 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. Therefore, after reading $W_{k,n-1}a_n$, automaton $\mathcal{A}_{k,n}$ must be in one of the states $(1;m)$, $1 \leq m \leq n$, or $(k+1;n)$. However, states $(1;m)$, $1 \leq m \leq n$, are the initial states of $\mathcal{A}_{k-1,n}$, which does not accept $W_{k-1,n}$ by induction, and hence neither $\mathcal{A}_{k,n}$ accepts $W_{k-1,n}$ from the states $(1;m)$, $1 \leq m \leq n$; indeed, after reading $W_{k-1,n}$, automaton $\mathcal{A}_{k-1,n}$

ends up in non-accepting states, and since $\mathcal{A}_{k,n}$ starting from the states $(1;m)$, $1 \leq m \leq n$, differs from $\mathcal{A}_{k-1,n}$ only in states $(2k;i)$, $i = 1, \ldots, n$, it ends up in the same non-accepting states after reading $W_{k-1,n}$ as $\mathcal{A}_{k-1,n}$. Thus, assume that $\mathcal{A}_{k,n}$ is in state $(k+1;n)$ after reading $W_{k,n-1}a_n$. Since $W_{k-1,n}$ has exactly $k-1$ occurrences of letter a_n, $\mathcal{A}_{k,n}$ is in state $(2k;n)$ after reading $W_{k-1,n}$. Hence $W_{k,n}$ is not accepted by $\mathcal{A}_{k,n}$. □

The last part of the previous proof shows that the suffix $W_{k-1,n}$ of the word $W_{k,n} = W_{k,n-1}a_nW_{k-1,n}$ is not accepted from state $(k+1;n)$. This can be generalized as follows.

Corollary 3.6. *For any suffix a_iw of $W_{k,n}$, w is not accepted from state $(k+1;i)$ of $\mathcal{A}_{k,n}$.*

Proof. Consider the word $W_{k,n}$ over $\Sigma_n = \{a_1, a_2, \ldots, a_n\}$ constructed in the proof of Lemma 3.5, and let $i \in \{1, \ldots, n\}$ be the maximal number for which there is a suffix a_iw of $W_{k,n}$ such that w is accepted by $\mathcal{A}_{k,n}$ from state $(k+1;i)$. Then $W_{k,n} = w_1a_iw_2w_3$, where $w_2 \in \{a_1, \ldots, a_i\}^*$ is the shortest word labeling the path from state $(k+1;i)$ to state max. By the construction of $\mathcal{A}_{k,n}$, word a_iw_2 must contain $k+1$ letters a_i. We shown that $W_{k,n}$ does not contain more than k letters a_i interleaved only with letters a_j for $j < i$, which yields a contradiction that proves the claim.

By definition, every longest factor of $W_{k,n}$ over $\{a_1, \ldots, a_i\}$ is of the form $W_{k-\ell,i}$, for $\ell \in \{0, \ldots, k-1\}$. Since $W_{k-\ell,i} = W_{k-\ell,i-1}\, a_i\, W_{k-\ell-1,i-1}\, a_i \cdots a_i\, W_{1,i-1}\, a_i$, the number of occurrences of a_i interleaved only with letters a_j for $j < i$ is at most $k - \ell$, which results in the maximum of k for $\ell = 0$ as claimed above. □

As already pointed out in the proof of Lemma 3.5, transitions (6) are redundant and present only to take care of confluence.

Corollary 3.7. *Removing from $\mathcal{A}_{k,n}$ the non-accepting states $(k+1,i), \ldots, (2k,i)$, for $1 \leq i \leq n$, and the corresponding transitions results in an rpoNFA that accepts the same language.*

Proof. By the proof of Lemma 3.5, removing the states with corresponding transitions has no effect on the accepted language. The resulting automaton is indeed an rpoNFA. This rpoNFA is exactly the rpoNFA used in our previous work [39]. □

We now have the necessary results to show, using a reduction from the word problem of polynomially-space-bounded Turing machines, that the universality problem for ptNFAs, where the alphabet may grow polynomially with the number of states, is PSPACE-complete.

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 Aho et al. [1] for details.

Theorem 3.8. *Deciding universality for ptNFAs is* PSPACE*-complete.*

Proof. Membership follows since universality is in PSPACE for NFAs [23].

To prove PSPACE-hardness, we consider a polynomial p and a p-space-bounded DTM $\mathcal{M} = (Q, T, I, \delta, \sqcup, q_o, q_f)$ with a nonempty input x. The basic idea of the proof is to use the alphabet $\Pi = \Sigma_n \times \Delta$, where $\Sigma_n = \{a_1, \ldots, a_n\}$ and Δ is used to encode runs of the Turing machine $\mathcal{M}$. The aim is to construct a ptNFA that accepts all words over Π, where the projection to the first component does not equal $W_{n,n}$ (the word constructed in Lemma 3.5) or the projection to the second component does not encode an accepting run of $\mathcal{M}$ on x.

Before elaborating on the details of the construction, we make several assumptions on the DTMs that simplify the proof and under which the word problem for these DTMs clearly remains PSPACE-hard. We assume that

(1) the initial and accepting states of $\mathcal{M}$ are different, *i.e.*, $q_o \neq q_f$;
(2) $\mathcal{M}$ accepts by looping in the accepting state q_f indefinitely, *i.e.*, no transition from state q_f modifies the tape, state, or head position;
(3) and $\mathcal{M}$ always accepts with the head at the very beginning of the tape.

A configuration of $\mathcal{M}$ on x consists of a current state $q \in Q$, the position $1 \leq \ell \leq p(|x|)$ of the read/write head, and the tape contents $\theta_1, \ldots, \theta_{p(|x|)}$ with $\theta_i \in T$. We represent it by a sequence

$$\langle \theta_1, \bullet \rangle \cdots \langle \theta_{\ell-1}, \bullet \rangle \langle \theta_\ell, q \rangle \langle \theta_{\ell+1}, \bullet \rangle \cdots \langle \theta_{p(|x|)}, \bullet \rangle$$

of symbols from $\Delta = T \times (Q \cup \{\bullet\})$. A run of $\mathcal{M}$ on x is represented as a 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 recognizing all words over $\Delta \cup \{\#\}$ that do not correctly encode a run of $\mathcal{M}$ (in particular are not of the form $\#w_1\#w_2\# \cdots \#w_m\#$) or that encode a run that is not accepting [1]. Such a regular expression can be constructed in the following three steps: we detect all words that

(A): do not start with the initial configuration;
(B): do not encode a valid run since they violate a transition rule (including words with an invalid encoding);
(C): encode non-accepting runs or runs that end prematurely.

If $\mathcal{M}$ has an accepting run on x, it has one without repeated configurations. There are $C(x) = |\Delta|^{p(|x|)}$ distinct configuration words in our encoding. Considering a separator symbol $\#$, the length of the encoding of a run without repeated configurations is at most $1 + C(x)(p(|x|) + 1)$, because every configuration word ends with $\#$ and is thus of length $p(|x|) + 1$. Let n be the least number such that $|W_{n,n}| \geq 1 + C(x)(p(|x|) + 1)$, where $W_{n,n}$ is the word constructed in Lemma 3.5. 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 $\mathcal{M}$ and x.

Consider the ptNFA $\mathcal{A}_{n,n}$ over the alphabet $\Sigma_n = \{a_1, \ldots, a_n\}$ of Lemma 3.5, and define the alphabet $\Delta_{\#\$} = \Delta \cup \{\#, \$\}$. We consider the alphabet $\Pi = \Sigma_n \times \Delta_{\#\$}$ where the first component is an input for $\mathcal{A}_{n,n}$ and the second component is used for encoding a run as described above; that is, letters of Π are pairs that might have pairs (of tape symbols and states) as their second element. Recall that $\mathcal{A}_{n,n}$ accepts all words different from $W_{n,n}$. Therefore, only those words over Π are of our interest, where the first components form the word $W_{n,n}$. Since the length of $W_{n,n}$ 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 $w[1] = a_{i_1} \cdots a_{i_\ell} \in \Sigma_n^\ell$ as the projection of w to the first components, and $w[2] = \delta_1 \ldots \delta_\ell \in \Delta_{\#\$}^\ell$ as the projection to the second components. Conversely, for a word $a_{i_1} \cdots a_{i_\ell} \in \Sigma_n^\ell$, we write $a_{i_1} \cdots a_{i_\ell} \otimes \Delta_{\#\$}$ to denote the set $(\{a_{i_1}\} \times \Delta_{\#\$}) \cdots (\{a_{i_\ell}\} \times \Delta_{\#\$})$ of all words $w \in \Pi^\ell$ with $w[1] = a_{i_1} \cdots a_{i_\ell}$. Similarly, for $\delta_1 \ldots \delta_\ell \in \Delta_{\#\$}^\ell$, we write $\Sigma_n \otimes \delta_1 \ldots \delta_\ell$ to denote the set $(\Sigma_n \times \{\delta_1\}) \cdots (\Sigma_n \times \{\delta_\ell\})$ of all words $w \in \Pi^\ell$ with $w[2] = \delta_1 \ldots \delta_\ell$. We extend this notation to sets of words.

Let $\mathrm{enc}(\mathcal{A}_{n,n})$ denote the automaton $\mathcal{A}_{n,n}$ with each transition $q \xrightarrow{a_i} q'$ replaced by all transitions $q \xrightarrow{\pi} q'$ with $\pi \in \{a_i\} \times \Delta_{\#\$}$. Then $\mathrm{enc}(\mathcal{A}_{n,n})$ accepts the language $\Pi^* \setminus (W_{n,n} \otimes \Delta_{\#\$})$. We say that a word w encodes an accepting run of $\mathcal{M}$ 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 $i \in \{1, 2, \ldots, m\}$ for which $\#w_1\#\cdots\#w_i\#$ encodes an accepting run of $\mathcal{M}$ on x, $w_k = w_i$ for all $k \in \{i+1, \ldots, m\}$, and $j \leq p(|x|)$. That is, 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 symbol \$. This extension is possible due to the assuption that $\mathcal{M}$ loops in the accepting configuration.

For **(A)**, we want to detect all words that do not start with the word

$$w[2] = \#\langle x_1, q_0\rangle\langle x_2, \bullet\rangle\cdots\langle x_{|x|}, \bullet\rangle\langle \sqcup, \bullet\rangle\cdots\langle \sqcup, \bullet\rangle\# \tag{3.2}$$

of length $p(|x|)+2$. This happens if **(A.1)** the word is shorter than $p(|x|)+2$, or **(A.2)** at position j, for $0 \leq j \leq p(|x|)+1$, there is a letter from the alphabet $\Delta_{\#\$} \setminus \{\delta_j\}$, where δ_j is the jth letter of the expected initial word (3.2). Let $\bar{E}_j = \Sigma_n \times (\Delta_{\#\$} \setminus \{\delta_j\})$. We can capture **(A.1)** and **(A.2)** in the regular expression

$$\left(\varepsilon + \Pi + \Pi^2 + \cdots + \Pi^{p(|x|)+1}\right) + \sum_{0\leq j\leq p(|x|)+1} (\Pi^j \cdot \bar{E}_j \cdot \Pi^*) \tag{3.3}$$

where Π^k is an abbreviation of the concatenation of Π k-times.

Expression (3.3) is polynomial in size. It can be captured by a ptNFA as follows. Each of the first $p(|x|)+2$ expressions defines a finite language and can easily be captured by a ptNFA (by a confluent DFA) of size of the expression. (By size of a regular expression, we mean the ordinary length, *i.e.*, the total number of symbols, including parentheses; *cf.* Ellul et al. [21] for more options and a detailed discussion.) The disjoint union of these ptNFAs then clearly forms a single ptNFA recognizing the language $\varepsilon + \Pi + \Pi^2 + \cdots + \Pi^{p(|x|)+1}$.

To express the language $\Pi^j \cdot \bar{E}_j \cdot \Pi^*$ as a ptNFA, we first construct the minimal incomplete DFA recognizing this language (states $0, 1, \ldots, j, max$ in Figure 5). However, we cannot complete this DFA by simply adding the missing transitions under $\Sigma_n \times \{\delta_j\}$ from state j to a new sink state because it results in a DFA with two maximal states – max and the sink state – violating the UMS property. Instead, we use a copy of the ptNFA $\mathrm{enc}(\mathcal{A}_{n,n})$ and add the missing transitions from state j under $\langle a_i, \delta_j\rangle \in \Sigma_n \times \{\delta_j\}$ to state $(n+1; i)$; see Figure 5 for an illustration. Notice that states $(n+1; i)$ are the states $(k+1; i)$ in Figure 4. The resulting automaton is a ptNFA, since it is complete, partially ordered, and satisfies the UMS property – for every state q different from max, the component co-reachable and reachable under the letters of self-loops in state q is only state q itself. This ptNFA accepts all words of $\Pi^j \cdot \bar{E}_j \cdot \Pi^*$.

We now show that any word w that is accepted by this ptNFA and that does not belong to $\Pi^j \cdot \bar{E}_j \cdot \Pi^*$ is such that $w[1] \neq W_{n,n}$, *i.e.*, w belongs to $\Pi^* \setminus (W_{n,n} \otimes \Delta_{\#\$})$. To this aim, assume that $w[1] = W_{n,n}$ and that w is of the form $w = u\langle a_i, \delta_j\rangle v$ such that $|u| = j$. Then, $\langle a_i, \delta_j\rangle$ is the letter under which the state $(n+1; i)$ of $\mathcal{A}_{n,n}$ is reached and v is read in the $\mathcal{A}_{n,n}$-part of the ptNFA. By Corollary 3.6, v is not accepted from state $(n+1; i)$, and hence the ptNFA does not accept w. Therefore, the ptNFA accepts the language $\Pi^j \cdot \bar{E}_j \cdot \Pi^* + (\Pi^* \setminus (W_{n,n} \otimes \Delta_{\#\$}))$. Constructing such a ptNFA for polynomially many expressions $\Pi^j \cdot \bar{E}_j \cdot \Pi^*$ and taking their disjoint union results in a polynomially large ptNFA accepting the language $\sum_{j=0}^{p(|x|)+1}(\Pi^j \cdot \bar{E}_j \cdot \Pi^*) + (\Pi^* \setminus (W_{n,n} \otimes \Delta_{\#\$}))$.

Notice that we ensure that the surrounding # in the initial configuration are present.

For **(B)**, we check for incorrect transitions and invalid encodings. Consider again the encoding $\#w_1\#\cdots\#w_m\#$ of a sequence of configurations with a word over $\Delta \cup \{\#\}$. We can assume that w_1 encodes the initial configuration according to **(A)**. In an encoding of a valid

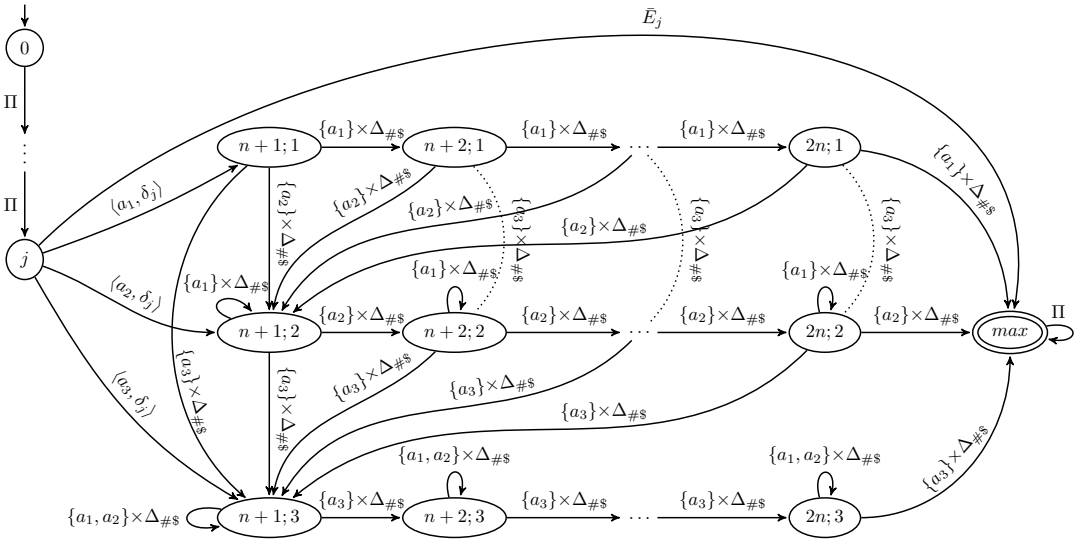

Figure 5: A ptNFA that accepts $\Pi^j \cdot \bar{E}_j \cdot \Pi^* + (\Pi^* \setminus (W_{n,n} \otimes \Delta_{\#\$}))$ with $\bar{E}_j = \Sigma_n \times (\Delta_{\#\$} \setminus \{\delta_j\})$ illustrated for $\Sigma_n = \{a_1, a_2, a_3\}$; only the relevant part of $\mathcal{A}_{n,n}$ is depicted.

run, the symbol at any position $j \geq p(|x|) + 2$ is uniquely determined by the three symbols at positions $j - p(|x|) - 2$, $j - p(|x|) - 1$, and $j - p(|x|)$, corresponding to the cell and its left and right neighbor 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 cases where $\delta = \#$, we define $f(\delta_\ell, \delta, \delta_r) = \#$, ensuring that the separator $\#$ is always present in successor configurations as well. We extend f to $f\colon \Delta^3_{\#\$} \to \Delta_{\#\$}$. We can then check for invalid transitions using the regular expression

$$\Pi^* \sum_{\delta_\ell, \delta, \delta_r \in \Delta_{\#\$}} (\Sigma_n \otimes \delta_\ell \delta \delta_r) \cdot \Pi^{p(|x|)-1} \cdot \hat{f}(\delta_\ell, \delta, \delta_r) \cdot \Pi^* \tag{3.4}$$

where $\hat{f}(\delta_\ell, \delta, \delta_r)$ is $\Pi \setminus (\Sigma_n \times \{f(\delta_\ell, \delta, \delta_r), \$\})$. Here, we also consider \$ as a correct continuation instead of the expected next configuration symbol. Note that (3.4) only detects wrong transitions and invalid encodings if a long enough next configuration exists. The case that the run stops prematurely is covered in **(C)**.

Expression (3.4) is not readily encoded in a ptNFA because of the leading Π^*. To address this issue, we replace Π^* by the expression $\Pi^{\leq |W_{n,n}|-1}$, which matches every word $w \in \Pi^*$ with $|w| \leq |W_{n,n}| - 1$. This suffices for our purpose because the computations of interest are of length $|W_{n,n}|$ and a violation of a correct computation must occur. As $|W_{n,n}| - 1$ is exponential, we cannot encode it directly and we use $\mathrm{enc}(\mathcal{A}_{n,n})$ instead.

In detail, let E be the expression obtained from (3.4) by omitting the initial Π^*, and let $\mathcal{B}_1$ be an incomplete DFA that accepts the language of E constructed as follows. From the initial state, we construct a tree-shaped DFA corresponding to all words of length three of the finite language $\sum_{\delta_\ell, \delta, \delta_r \in \Delta_{\#\$}} (\Sigma_n \otimes \delta_\ell \delta \delta_r)$. To every leaf state, we add a path under Π of length $p(|x|) - 1$. The result corresponds to the language $\sum_{\delta_\ell, \delta, \delta_r \in \Delta_{\#\$}} (\Sigma_n \otimes \delta_\ell \delta \delta_r) \cdot \Pi^{p(|x|)-1}$. Let $q_{\delta_\ell \delta \delta_r}$ denote the states uniquely determined by the words in $(\Sigma_n \otimes \delta_\ell \delta \delta_r) \cdot \Pi^{p(|x|)-1}$. We

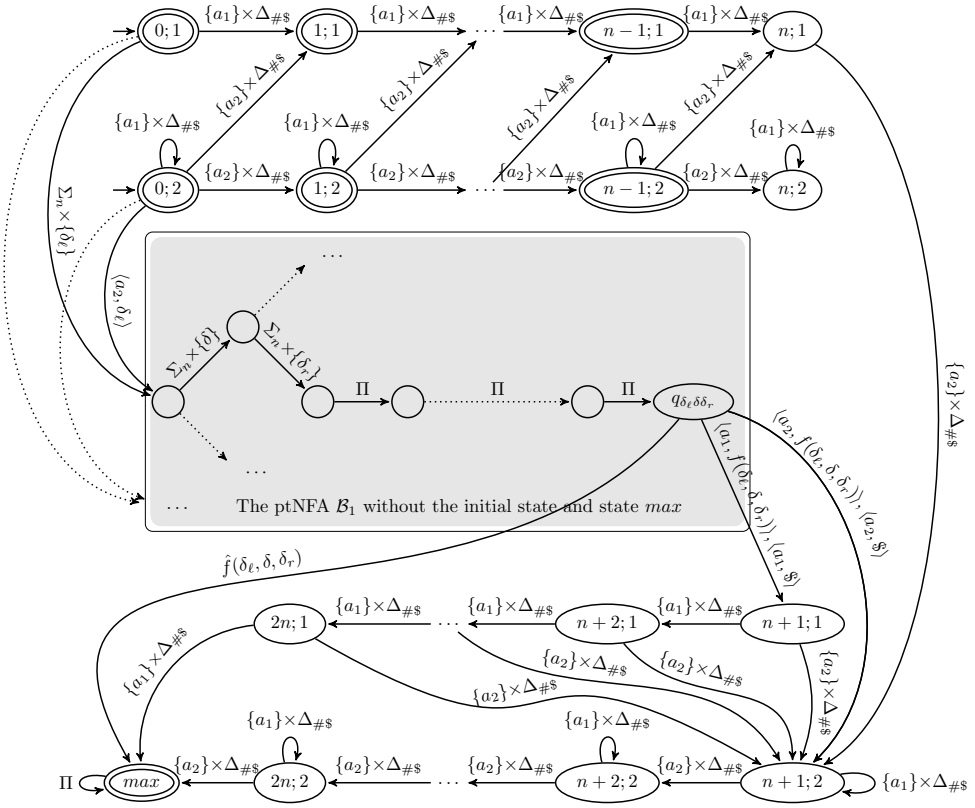

Figure 6: The ptNFA $\mathcal{B}$ that combines ptNFA $\mathrm{enc}(\mathcal{A}_{n,n})$ with ptNFA $\mathcal{B}_1$, for $n = 2$; the edges from accepting states of $\mathrm{enc}(\mathcal{A}_{n,n})$ to the second states of $\mathcal{B}_1$ are illustrated only for states $(0;1)$ and $(0;2)$.

add the transitions $q_{\delta_\ell\delta\delta_r} \xrightarrow{\hat{f}(\delta_\ell,\delta,\delta_r)} max$, where max is the state of $\mathrm{enc}(\mathcal{A}_{n,n})$. Automaton $\mathcal{B}_1$ is illustrated in the middle part of Figure 6, except for the initial state that is identified with accepting states of the ptNFA $\mathrm{enc}(\mathcal{A}_{n,n})$ as described below, and state max that is a state of $\mathrm{enc}(\mathcal{A}_{n,n})$. It is an incomplete DFA for the language of E of polynomial size. It is incomplete only in states $q_{\delta_r\delta\delta_\ell}$ due to the missing transitions under $\Sigma_n \times \{f(\delta_\ell, \delta, \delta_r)\}$ and $\Sigma_n \times \{\$\}$. We complete it by adding the missing transitions to the states of the ptNFA $\mathrm{enc}(\mathcal{A}_{n,n})$. Namely, for $z \in \{\langle a_i, f(\delta_\ell, \delta, \delta_r)\rangle, \langle a_i, \$\rangle\}$, we add $q_{\delta_\ell\delta\delta_r} \xrightarrow{z} (n+1; i)$; see Figure 6 for an illustration.

We construct a ptNFA $\mathcal{B}$ accepting the language $(\Pi^* \setminus (W_{n,n} \otimes \Delta_{\#\$})) + (\Pi^{\leq |W_{n,n}|-1} \cdot E)$ by merging $\mathrm{enc}(\mathcal{A}_{n,n})$ with the DFA $\mathcal{B}_1$, where we add edges labeled with $(\Sigma_n \setminus \Sigma(q)) \times \{\delta_\ell\}$ from any accepting state q of $\mathrm{enc}(\mathcal{A}_{n,n})$ to the states of the second level of the tree-shaped DFA $\mathcal{B}_1$ (the leftmost states in the middle part of Figure 6). This step is justified by Corollary 3.7, since we do not need to consider connecting $\mathcal{B}_1$ to non-accepting states of $\mathrm{enc}(\mathcal{A}_{n,n})$ and it is not possible to connect it to state max. The fact that $\mathrm{enc}(\mathcal{A}_{n,n})$ alone accepts $\Pi^* \setminus (W_{n,n} \otimes \Delta_{\#\$})$ was shown in Lemma 3.5. 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 $\mathcal{B}_1$ after reading a word of length $\geq |W_{n,n}|$ is possible but all such words are longer than $W_{n,n}$, and hence in $\Pi^* \setminus (W_{n,n} \otimes \Delta_{\#\$})$.

To show that the completion does not affect the language, let w be a word that is read but not accepted by $\mathcal{B}_1$, and let u lead $\mathrm{enc}(\mathcal{A}_{n,n})$ to a state from which w is read in $\mathcal{B}_1$. Since w is not accepted, there is a letter z and a word v such that uwz goes to state $(n+1;i)$ of $\mathrm{enc}(\mathcal{A}_{n,n})$ (for $z[1] = a_i$) and v leads $\mathrm{enc}(\mathcal{A}_{n,n})$ from state $(n+1;i)$ to state max. If $u[1]w[1]a_iv[1] = W_{n,n,}$, then v is not accepted from $(n+1;i)$ by Corollary 3.6, and hence $uwzv[1] \neq W_{n,n}$; thus, $uwzv \notin (W_{n,n} \otimes \Delta_{\#\$})$.

It remains to show that for every proper prefix $w_{n,n}$ of $W_{n,n}$, there is a state in $\mathcal{A}_{n,n}$ reached by $w_{n,n}$ that has transitions to the second states of $\mathcal{B}_1$, and hence the check represented by E in $\Pi^{\leq |W_{n,n}|-1} \cdot E$ can be performed. In other words, if $a_{n,n}$ denotes the letter following $w_{n,n}$ in $W_{n,n}$, then there must be a state reachable by $w_{n,n}$ in $\mathcal{A}_{n,n}$ that does not have a self-loop under $a_{n,n}$. However, this follows from the fact that $\mathcal{A}_{n,n}$ accepts everything but $W_{n,n}$, since then the DFA obtained from $\mathcal{A}_{n,n}$ by the standard subset construction has a path of length $\binom{2n}{n} - 1$ labeled with $W_{n,n}$ without any loop. Moreover, any state of this path in the DFA is a subset of states of $\mathcal{A}_{n,n}$, and therefore at least one of the states reachable under $w_{n,n}$ in $\mathcal{A}_{n,n}$ does not have a self-loop under $a_{n,n}$.

The ptNFA $\mathcal{B}$ thus accepts the language $(\Pi^{\leq |W_{n,n}|-1} \cdot E) + (\Pi^* \setminus (W_{n,n} \otimes \Delta_{\#\$}))$.

Finally, for **(C)**, we detect all words that **(C.1)** end in a configuration that is incomplete (too short), possibly followed by at most $p(|x|)$ trailing \$, **(C.2)** end in a configuration that is not in the accepting state q_f, which must be the very first tape symbol by our assumption, **(C.3)** end with more than $p(|x|)$ trailing \$, or **(C.4)** contain \$ not only at the last positions, that is, we detect all words where \$ is followed by a different symbol. For a word v, we use $v^{\leq i}$ to abbreviate $\varepsilon + v + \cdots + v^i$, and we define $\bar{E}_f = (T \times (Q \setminus \{q_f\}))$. Then these properties are expressed by the following expressions:

$$\begin{aligned}
&\textbf{(C.1)}\ \ \Pi^* \left(\Sigma_n \times \{\#\}\right) \left(\Pi + \cdots + \Pi^{p(|x|)}\right) \left(\Sigma_n \times \{\$\}\right)^{\leq p(|x|)} + \\
&\textbf{(C.2)}\ \ \Pi^* \left(\Sigma_n \times \bar{E}_f\right) \Pi^{p(|x|)-1} \left(\Sigma_n \times \{\#\}\right) \left(\Sigma_n \times \{\$\}\right)^{\leq p(|x|)} + \\
&\textbf{(C.3)}\ \ \Pi^* \left(\Sigma_n \times \{\$\}\right)^{p(|x|)+1} + \\
&\textbf{(C.4)}\ \ \left(\Pi \setminus \left(\Sigma_n \times \{\$\}\right)\right)^* \left(\Sigma_n \times \{\$\}\right) \left(\Sigma_n \times \{\$\}\right)^* \left(\Pi \setminus \left(\Sigma_n \times \{\$\}\right)\right) \Pi^*
\end{aligned} \tag{3.5}$$

As before, we cannot encode the expression directly as a ptNFA, but we can perform a similar construction as the one used for encoding (3.4). Namely, ptNFAs for **(C.1)** and **(C.2)** are illustrated in Figure 7 and for **(C.3)** in Figure 8, where, analogously to Figure 6, the edges from the accepting states of $\mathrm{enc}(\mathcal{A}_{n,n})$ to the second state of the automaton $\mathcal{C}_i$ that recognizes the language of expression **(C.i)** without the initial Π^*, for $i = 1, 2, 3$, are illustrated only for states $(0;1)$ and $(0;2)$. However, the reader should keep in mind that such transitions lead from every accepting state of $\mathrm{enc}(\mathcal{A}_{n,n})$ to the second state of $\mathcal{C}_i$. Finally, **(C.4)** can be represented by a three-state partially ordered and confluent DFA.

The expressions (3.3)–(3.5) together then detect all non-accepting or wrongly encoded runs of $\mathcal{M}$. In particular, if we start from the correct initial configuration ((3.3) does not match), then for (3.4) not to match, all complete future configurations must have exactly one state, be delimited by encodings of #, and correctly follow from the previous configurations. We have shown how to express each of the expressions as a ptNFA. Taking the disjoint union of all these ptNFAs results in a single ptNFA. This ptNFA is of polynomial size, and hence we have reduced the word problem of polynomially-space-bounded Turing machines to the universality problem for ptNFAs. □

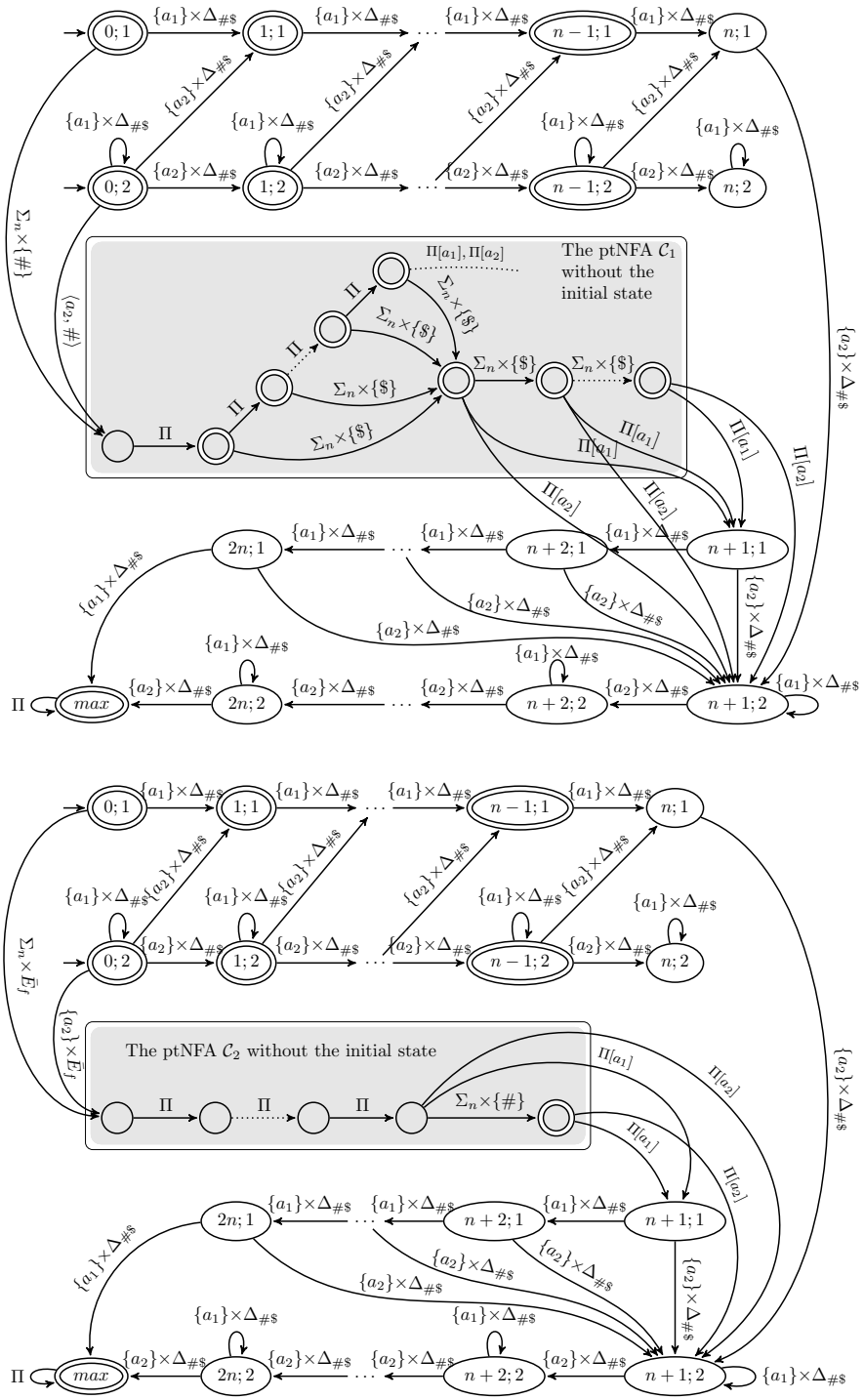

Figure 7: ptNFAs for **(C.1)** (top) and **(C.2)** (bottom) illustrated for $n = 2$; automata $\mathcal{C}_1$ and $\mathcal{C}_2$, recognizing resp. **(C.1)** and **(C.2)** without the initial Π^*, are completed by adding transitions $q \xrightarrow{\Pi[a_i]} (n+1; i)$, where $\Pi[a_i]$ denotes all letters of Π undefined in state q with the first component of the letter being a_i.

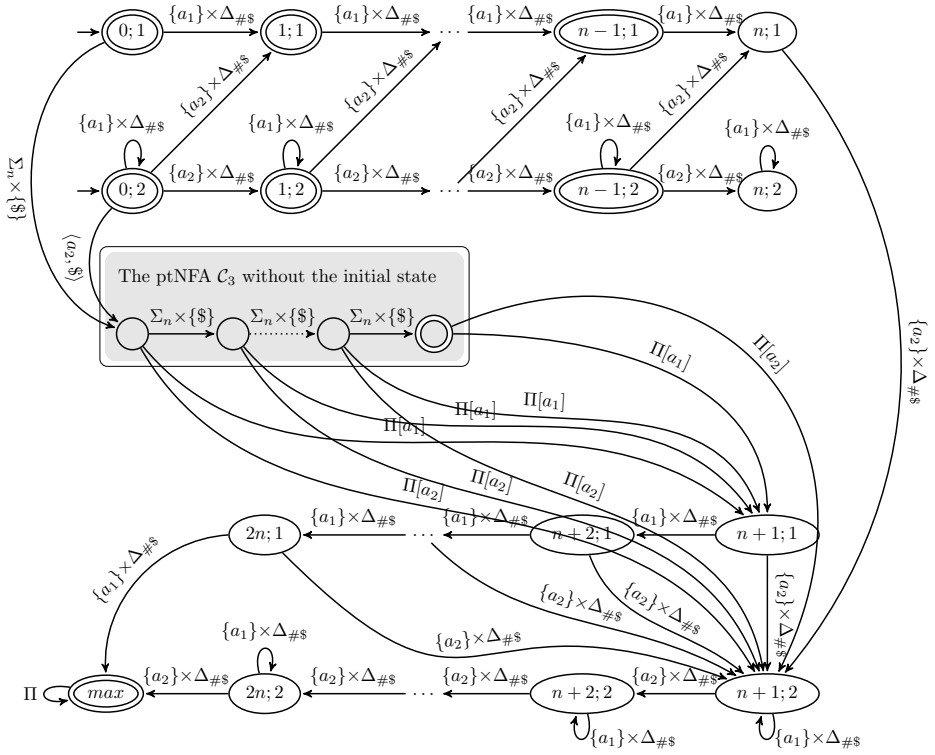

Figure 8: The ptNFA for **(C.3)** illustrated for $n = 2$; automaton $\mathcal{C}_3$, recognizing **(C.3)** without the initial Π^*, is completed by adding transitions $q \xrightarrow{\Pi[a_i]} (n+1;i)$, where $\Pi[a_i]$ denotes all letters of Π undefined in q with the first component of the letter being a_i.

4. Complexity of Deciding k-Piecewise Testability

The effort to simplify XML Schema using the BonXai language [46] led to the study of (k-)piecewise testable languages [17, 29]. A regular language over Σ is *piecewise testable* if it is a finite boolean combination of languages of the form $\Sigma^* a_1 \Sigma^* a_2 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$, where $a_i \in \Sigma$ for $i = 1, \ldots, n$, $n \geq 0$. Let $k \geq 0$ be an integer. The language is *k-piecewise testable* if $n \leq k$. The *k-piecewise testability problem* asks whether a given automaton recognizes a k-piecewise testable language.

In this section, we study the complexity of deciding k-piecewise testability for partially ordered automata. Our results are summarized in Table 7.

To simplify proofs, we make use of the following lemma that will save us a lot of work by directly obtaining the lower bounds from the complexity results for universality.

We first need some additional definitions. For $k \geq 0$, let $\mathrm{sub}_k(v) = \{u \in \Sigma^* \mid u \preccurlyeq v, |u| \leq k\}$. For two words w_1, w_2, we write $w_1 \sim_k w_2$ if $\mathrm{sub}_k(w_1) = \mathrm{sub}_k(w_2)$. The relation $\sim_k$ is a congruence with finite index, and every k-piecewise testable language is a finite union of $\sim_k$ classes [74].

Lemma 4.1. *Let $k \geq 0$ be a constant. Then the universality problem for ptNFAs (resp. rpoNFAs, poNFAs, NFAs, DFAs) is log-space reducible to the k-piecewise testability problem for ptNFAs (resp. rpoNFAs, poNFAs, NFAs, DFAs).*

	Unary alphabet	Fixed alphabet	Arbitrary alphabet	
	$\lvert\Sigma\rvert = 1$	$\lvert\Sigma\rvert \geq 2$	$k \leq 3$	$k \geq 4$
DFA	L-c (Thm. 4.7)	NL-c (Thm. 4.4)	NL-c [53]	coNP-c [34]
ptNFA	NL-c (Thm. 4.6)	coNP-c (Thm. 4.3)	PSpace-c (Thm. 4.2)	
rpoNFA	NL-c	coNP-c [39]	PSpace-c	
poNFA	NL-c (Thm. 4.6)	PSpace-c (Thm. 4.5)	PSpace-c	
NFA	coNP-c (Thm. 4.8)	PSpace-c [53]	PSpace-c [53]	

Table 7: Complexity of deciding k-piecewise testability.

Proof. Let $\mathcal{M}$ over Σ be a ptNFA (resp. rpoNFA, poNFA, NFA, DFA) recognizing a nonempty language. We construct a ptNFA (resp. rpoNFA, poNFA, NFA, DFA) $\mathcal{M}_k$ over Σ from $\mathcal{M}$ as depicted in Figure 9. Namely, we add $|\Sigma|k$ new states $i_{j,1}, \ldots, i_{j,|\Sigma|k}$ for every initial state i_j of $\mathcal{M}$. For $1 \leq \ell < |\Sigma|k$, we add transitions from $i_{j,\ell}$ to $i_{j,\ell+1}$ and a transition from $i_{j,|\Sigma|k}$ to the initial state i_j of $\mathcal{M}$ under all letters of Σ. The initial states of $\mathcal{M}_k$ are the states $i_{j,1}$, the accepting states are the accepting states of $\mathcal{M}$ and the states $i_{j,k+1}, \ldots, i_{j,|\Sigma|k}$. Note that $\mathcal{M}_k$ is a ptNFA (resp. rpoNFA, poNFA, NFA, DFA) constructible in logarithmic space.

If the language $L(\mathcal{M})$ is universal, *i.e.*, $L(\mathcal{M}) = \Sigma^*$, then the language $L(\mathcal{M}_k) = \Sigma^k\Sigma^*$ is k-piecewise testable because it consists of all words of length at least k, and hence if there are $x \in L(\mathcal{M}_k)$ and $y \notin L(\mathcal{M}_k)$, then the length of y is less than k, and hence $x \sim_k y$ does not hold because $\mathrm{sub}_k(x)$ contains a word of length k that is not in $\mathrm{sub}_k(y)$.

If the language $L(\mathcal{M})$ is not universal, then there exist $x \in L(\mathcal{M})$ and $y \notin L(\mathcal{M})$. Let Σ be $\{a_1, a_2, \ldots, a_{|\Sigma|}\}$. Then $(a_1a_2\cdots a_{|\Sigma|})^k x \sim_k (a_1a_2\cdots a_{|\Sigma|})^k y$, since $\mathrm{sub}_k((a_1a_2\cdots a_{|\Sigma|})^k) = \{u \in \Sigma^* \mid |u| \leq k\}$, and $(a_1a_2\cdots a_{|\Sigma|})^k x \in L(\mathcal{M}_k)$ and $(a_1a_2\cdots a_{|\Sigma|})^k y \notin L(\mathcal{M}_k)$, which shows that the language $L(\mathcal{M}_k)$ is not k-piecewise testable. □

We immediately have the following consequences. The first consequence is that deciding k-piecewise testability for ptNFAs, where the alphabet may grow with the number of states, is PSpace-complete.

Theorem 4.2. *Deciding k-piecewise testability for ptNFAs is* PSpace*-complete.*

Proof. Membership follows from the results for NFAs, hardness follows from Lemma 4.1 and Theorem 3.8. □

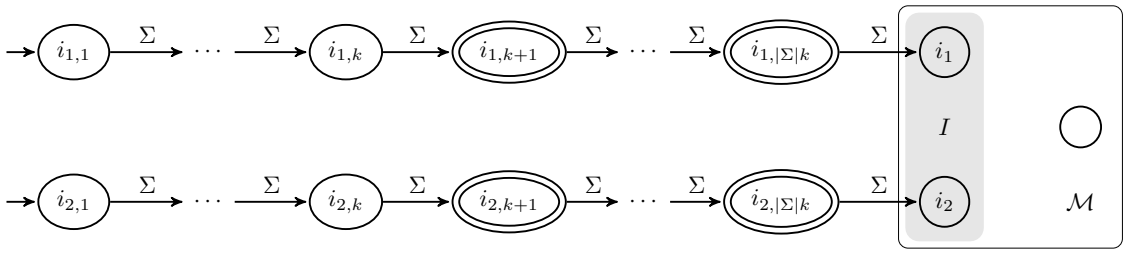

Figure 9: The ptNFA $\mathcal{M}_k$ constructed from a ptNFA $\mathcal{M}$ with two initial states.

The second consequence is that the complexity decreases if we only consider ptNFAs over a fixed alphabet. We distinguish two cases: (i) at least binary alphabets, and (ii) unary alphabets.

Theorem 4.3. *Let Σ be a fixed alphabet with at least two letters. Deciding k-piecewise testability for ptNFAs over Σ is* coNP*-complete.*

Proof. Hardness follows from Lemma 4.1 and Theorem 3.2, membership follows from the result for rpoNFAs [39, Corollary 24]. □

This result is in contrast with an analogous result for DFAs. Deciding k-piecewise testability for DFAs over a fixed alphabet is in P [34]. A more precise complexity can be shown.

Theorem 4.4. *Let Σ be a fixed alphabet with at least two letters. Deciding k-piecewise testability for DFAs over Σ is* NL*-complete.*

Proof. Hardness follows from Lemma 4.1 because deciding universality for DFAs is NL-complete [32]. Membership can be shown as follows. Since Σ and k are fixed, there is a constant number of k-piecewise testable languages over Σ, and hence we may assume that the minimal DFAs of all these languages are precomputed. Let $\mathcal{A}$ be a DFA. Then $L(\mathcal{A})$ is k-piecewise testable if and only if it is equivalent to one of the precomputed languages. This can be verified in NL by guessing a precomputed minimal DFA and checking equivalence (see the next section for more details). □

In comparison with ptNFAs or rpoNFAs, fixing the alphabet does not affect the complexity for poNFAs.

Theorem 4.5. *Let Σ be a fixed alphabet with at least two letters. Deciding k-piecewise testability for poNFAs over Σ is* PSpace*-complete.*

Proof. Membership follows from the results for NFAs, hardness follows from Lemma 4.1 and the fact that deciding universality for poNFAs over Σ is PSpace-complete [39]. □

Theorems 4.3 and 4.5 show hardness even for binary alphabets, which improves our recent result where the alphabet had at least three letters [49]. Furthermore, we point out that hardness in Theorem 4.3 does not follow from the coNP-hardness proof of Klíma et al. [34] showing coNP-completeness of deciding k-piecewise testability for DFAs for $k \geq 4$, since their proof requires a growing alphabet.

It remains to consider the case of unary alphabets. We first focus on the case of nondeterministic partially ordered automata and the variants thereof.

Theorem 4.6. *Deciding k-piecewise testability for poNFAs, rpoNFAs, and ptNFAs over a unary alphabet is* NL*-complete. It holds even if k is given as part of the input.*

Proof. Hardness follows from Lemma 4.1 and Theorem 3.1. We now show membership for poNFAs, which covers all the cases. Let $\mathcal{A}$ be a poNFA over the alphabet $\{a\}$ with n states. If the language $L(\mathcal{A})$ is infinite, then there exists $d \leq n$ such that $a^d a^* \subseteq L(\mathcal{A})$; indeed, $L(\mathcal{A})$ is infinite if and only if there is an accepting state that is reachable via a state with a self-loop, and hence d is bounded by the number of states on such a path. Therefore, the language $L(\mathcal{A})$ is *not* k-piecewise testable if and only if there exists ℓ with $k < \ell \leq d$ such that $a^k \in L(\mathcal{A})$ if and only if $a^\ell \notin L(\mathcal{A})$. Such an ℓ can be guessed in binary and the property verified in NL, since NL is closed under complement. If $L(\mathcal{A})$ is finite, its complement, which is k-piecewise testable if and only if $L(\mathcal{A})$ is, is infinite. □

Now we focus on the case of deterministic automata.

Theorem 4.7. *Deciding k-piecewise testability for DFAs over a unary alphabet is L-complete.*

Proof. Hardness follows from Lemma 4.1 and the fact that deciding universality for unary DFAs is L-complete [32]. Membership in L can be shown as follows. Let n be the number of states of the DFA. Then the language is k-piecewise testable if and only if $a^k, a^{k+1}, \ldots, a^{k+n}$ all belong to the language or none does. ($k+n$ because there may be a cycle to the initial state.) □

Finally, we focus on the case of general nondeterministic automata.

Theorem 4.8. *Deciding k-piecewise testability for NFAs over a unary alphabet is* coNP*-complete.*

Proof. Hardness follows from Lemma 4.1 and the fact that deciding universality for unary NFAs is coNP-complete [77]. To show membership, we first show that deciding piecewise testability for NFAs over a unary alphabet is in coNP. To do this, we show how to check non-piecewise testability in NP. Intuitively, we need to check that the corresponding DFA is partially ordered and confluent. However, confluence is trivially satisfied because there is no branching in a DFA over a single letter. Partial order is violated if and only if there exist three words a^{ℓ_1}, a^{ℓ_2} and a^{ℓ_3} with $\ell_1 < \ell_2 < \ell_3$ such that $\delta(I, a^{\ell_1}) = \delta(I, a^{\ell_3}) \neq \delta(I, a^{\ell_2})$ and one of these sets is accepting (as a state of the DFA) and the other is not (otherwise they are equivalent). The lengths of the words are bounded by 2^n, where n denotes the number of states of the NFA, and can thus be guessed in binary. The matrix multiplication (fast exponentiation) can then be used to compute the sets of states reachable under those words in polynomial time.

Thus, we can check in coNP whether the language of an NFA is piecewise testable. If so, then it is 2^n-piecewise testable, since the depth of the minimal DFA is bounded by 2^n, where n is the number of states of the NFA [49]. Let M be the transition matrix of the NFA. To show that it is not k-piecewise testable, we need to find two $\sim_k$-equivalent words such that exactly one of them belongs to the language of the NFA. Since every $\sim_k$ class defined by a^ℓ, for $\ell < k$, is a singleton, we need to find $k < \ell \leq 2^n$ such that $a^k \sim_k a^\ell$ and only one of them belongs to the language. This can be done in nondeterministic polynomial time by guessing ℓ in binary, using the matrix multiplication to obtain the corresponding reachable sets in M^k and M^ℓ, and verifying that one set contains an accepting state and the other does not. □

5. Complexity of Deciding Piecewise Testability

The *piecewise testability problem* asks, given an automaton, whether it recognizes a piecewise testable language. We now study the complexity of deciding piecewise testability for partially ordered automata. Our results are summarized in Table 8.

To simplify proofs, we would like to use a result similar to Lemma 4.1. Unfortunately, there is no such result preserving the alphabet. If there were, it would imply that deciding piecewise testability for ptNFAs has a nontrivial complexity, but these languages are trivially

[3]Cho and Huynh [13] showed hardness for a three-letter alphabet. However, the result holds also for binary alphabets, using, *e.g.*, a reduction from the reachability problem for directed acyclic graphs with out-degree at most two.

	$\lvert\Sigma\rvert = 1$	$\lvert\Sigma\rvert \geq 2$	Σ is growing
DFA	L-c (Thm. 5.2)	NL-c [13][3]	NL-c [13]
rpoNFA	✓ (Thm. 5.1)	coNP-c (Thm. 5.6)	PSpace-c (Thm. 5.5)
poNFA	✓ (Thm. 5.1)	PSpace-c (Thm. 5.4)	PSpace-c
NFA	coNP-c (Thm. 5.3)	PSpace-c [51]	PSpace-c [51]

Table 8: Complexity of deciding piecewise testability.

piecewise testable. Similarly, it would imply that deciding piecewise testability of unary (r)poNFAs is nontrivial, but we show below that they are trivially piecewise testable.

Recall that $\mathcal{R}$-trivial languages, poDFA-languages, and rpoNFA-languages coincide.

Theorem 5.1. *The classes of unary poNFA languages, unary $\mathcal{R}$-trivial languages, and unary piecewise testable languages coincide.*

Proof. Since every piecewise testable language is an $\mathcal{R}$-trivial language, and every $\mathcal{R}$-trivial language is a poNFA language, we only need to prove that unary poNFA languages are piecewise testable. If the language of a poNFA is finite, then it is piecewise testable. If it is infinite, then there is an integer n bounded by the number of states of the poNFA such that the poNFA accepts all words of length longer than n. The minimal DFA equivalent to the poNFA is thus partially ordered and confluent. □

We first discuss the complexity of deciding piecewise testability for unary DFAs.

Theorem 5.2. *Deciding piecewise testability for DFAs over a unary alphabet is L-complete.*

Proof. To prove hardness, we reduce from the DAG-reachability problem where no vertex has more than one outgoing directed edge [32]. Let G be a directed acyclic graph where no vertex has more than one outgoing directed edge with vertices $1, 2, \ldots, n$, $n > 1$. We define a DFA $\mathcal{A} = (\{0, 1, \ldots, n\}, \{a\}, \delta, 1, \{n\})$ using exactly the same reduction as Jones [32, Theorem 26] defining $\delta(i, a) = j$ if (i, j) is an edge of G and $i \neq n$, $\delta(n, a) = 1$, and $\delta(i, a) = 0$ for other values of i. Then n is reachable from 1 in G if and only if the language $L(\mathcal{A})$ is infinite as well as the language $\{a\}^* \setminus L(\mathcal{A})$, which implies non-piecewise testability of the language $L(\mathcal{A})$ because the minimal DFA for $L(\mathcal{A})$ needs to have a nontrivial cycle.

To show membership, we need to check that there is no nontrivial cycle in the minimal DFA equivalent to the given DFA. This can be done by checking that the words $a^n, \ldots, a^{2n}$ all have the same accepting status, where n is the number of states of the given DFA. □

Next we discuss the case of unary NFAs.

Theorem 5.3. *Deciding piecewise testability for NFAs over a unary alphabet is* coNP*-complete.*

Proof. Membership is shown in the proof of Theorem 4.8. To show hardness, we modify the proof of Stockmeyer and Meyer [77]. Let φ be a formula in 3CNF with n distinct variables, and let C_k be the set of literals in the kth conjunct, $1 \leq k \leq m$. The assignment to the variables can be represented as a binary vector of length n. Let $p_1, p_2, \ldots, p_n$ be the first n prime numbers. For a natural number z congruent with 0 or 1 modulo p_i, for every

$1 \leq i \leq n$, we say that z satisfies φ if the assignment $(z \bmod p_1, z \bmod p_2, \ldots, z \bmod p_n)$ satisfies φ. Let

$$E_0 = \bigcup_{k=1}^{n} \bigcup_{j=2}^{p_k-1} 0^j \cdot (0^{p_k})^*$$

that is, $L(E_0) = \{0^z \mid \exists k \leq n, z \not\equiv 0 \bmod p_k \text{ and } z \not\equiv 1 \bmod p_k\}$ is the set of natural numbers that do not encode an assignment to the variables. For each conjunct C_k, we construct an expression E_k such that if $0^z \in L(E_k)$ and z is an assignment, then z does not assign the value 1 to any literal in C_k. For example, if $C_k = \{x_r, \neg x_s, x_t\}$, for $1 \leq r, s, t \leq n$ and r, s, t distinct, let z_k be the unique integer such that $0 \leq z_k < p_r p_s p_t$, $z_k \equiv 0 \bmod p_r$, $z_k \equiv 1 \bmod p_s$, and $z_k \equiv 0 \bmod p_t$. Then

$$E_k = 0^{z_k} \cdot (0^{p_r p_s p_t})^* .$$

Now, φ is satisfiable if and only if there exists z such that z encodes an assignment to φ and $0^z \notin L(E_k)$ for all $1 \leq k \leq m$, which is if and only if $L(E_0 \cup \bigcup_{k=1}^{m} E_k) \neq 0^*$.

The proof up to now shows that universality is coNP-hard for NFAs over a unary alphabet. Let now $p_n\# = \Pi_{i=1}^{n} p_i$. If z encodes an assignment of φ, then, for any natural number c, $z + c \cdot p_n\#$ also encodes an assignment of φ; indeed, if $z \equiv x_i \bmod p_i$, then $z + c \cdot p_n\# \equiv x_i \bmod p_i$, for every $1 \leq i \leq n$. This shows that if, in addition, $0^z \notin L(E_k)$ for all k, then $0^z(0^{p_n\#})^* \cap L(E_0 \cup \bigcup_{k=1}^{m} E_k) = \emptyset$. Since both the languages of the intersection are infinite, the minimal DFA recognizing the language $L(E_0 \cup \bigcup_{k=1}^{m} E_k)$ must have a non-trivial cycle alternating between accepting and non-accepting states. Therefore, if the language $L(E_0 \cup \bigcup_{k=1}^{m} E_k)$ is universal, then it is piecewise testable, and if it is non-universal, then it is not piecewise testable. □

We next show that deciding piecewise testability for poNFAs is PSpace-complete even if the alphabet is binary.

Theorem 5.4. *Let Σ be a fixed alphabet with at least two letters. Deciding piecewise testability for poNFAs over Σ is* PSpace*-complete.*

Proof. Membership in PSpace follows from the results for NFAs. PSpace-hardness follows from an analogous result for rpoNFAs [39] where we construct, given a polynomial-space-bounded DTM M and an input x, a binary poNFA $\mathcal{A}_x$ in polynomial time such that if M does not accept x, then $L(\mathcal{A}_x) = \{0, 1\}^*$, which is piecewise testable, and if M accepts x, then $L(\mathcal{A}_x)$ is not $\mathcal{R}$-trivial, and hence neither piecewise testable. The language of $\mathcal{A}_x$ is thus piecewise testable if and only if M does not accept x. □

The case of rpoNFAs is more complicated. In the next theorem, we show that deciding piecewise testability for rpoNFAs is PSpace-complete if the alphabet is not fixed, and then we discuss the case of rpoNFAs over a fixed (binary) alphabet.

Theorem 5.5. *Deciding piecewise testability for rpoNFAs is* PSpace*-complete.*

Proof. Membership follows from the result for NFAs. To prove hardness, we reduce the universality problem for rpoNFAs, which is PSpace-complete in general and coNP-complete for fixed alphabets [39].

Let $\mathcal{A}$ be an rpoNFA, and let Σ be its alphabet. We construct an rpoNFA $\mathcal{B}$ from $\mathcal{A}$ by adding two fresh letters $a, b \notin \Sigma$ and by adding two new states 1 and 2. State 1 is the only accepting state of $\mathcal{B}$. From every non-accepting state of $\mathcal{A}$, we add an a-transition to state 1 and a b-transition to state 2. From every accepting state of $\mathcal{A}$, we add an a- and a

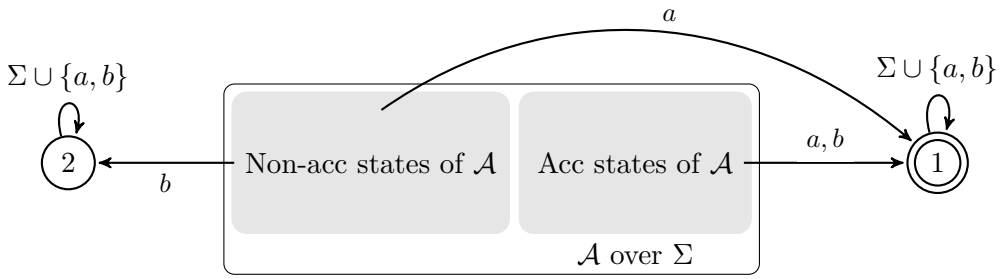

Figure 10: The illustration from the proof of Theorem 5.5.

b-transition to 1. Finally, states 1 and 2 contain self-loops under all lettrers from $\Sigma \cup \{a, b\}$. The construction is illustrated in Figure 10. We now show that $L(\mathcal{B})$ is piecewise testable if and only if $\mathcal{A}$ is universal.

If $L(\mathcal{A}) = \Sigma^*$, then $L(\mathcal{B}) = \Sigma^*(a + b)(\Sigma \cup \{a, b\})^*$, because for every $w \in L(\mathcal{A})$, the set of reachable states in $\mathcal{B}$ under w contains an accepting state, and hence both wa and wb lead to state 1. Language $\Sigma^*(a+b)(\Sigma \cup \{a, b\})^*$ is piecewise testable; it can be seen by computing the two-state minimal DFA and verifying that it is partially ordered and confluent.

If $L(\mathcal{A})$ is not universal, then there is a $w \in \Sigma^* \setminus L(\mathcal{A})$. Then the set of states reachable under w in $\mathcal{B}$ consists only of non-accepting states. By the construction, only state 1 is reachable under wa, and only state 2 is reachable under wb. Let $L_1 = wa(ba)^*$ and $L_2 = wb(ab)^*$. Every word of L_1 is accepted by $\mathcal{B}$ whereas none word of L_2 is. If $L(\mathcal{B})$ was piecewise testable, then there would be $k \geq 0$ such that for any words w_1 and w_2 with $w_1 \sim_k w_2$, either both words belong to $L(\mathcal{B})$ or neither does. However, for every $k \geq 0$, we have that $wa(ba)^k \sim_k wb(ab)^k$ and the acceptance status of the two words is different. Therefore, the language $L(\mathcal{B})$ is not piecewise testable. □

We now discuss the complexity of deciding piecewise testability for rpoNFAs over a fixed (binary) alphabet.

Theorem 5.6. *Let Σ be a fixed alphabet with at least two letters. Deciding piecewise testability for rpoNFAs over Σ is* coNP*-complete.*

Proof. To show membership in coNP, we proceed as follows. Let $\mathcal{A} = (Q, \Sigma, \delta_{\mathcal{A}}, Q_0, F)$ be an rpoNFA over a fixed alphabet $\Sigma = \{a_1, a_2, \ldots, a_c\}$, and consider the minimal DFA $\mathcal{D}$ equivalent to $\mathcal{A}$. Then $L(\mathcal{A})$ is piecewise testable if and only if $\mathcal{D}$ satisfies the UMS property (*cf.* Subsection 2.2). We proceed by first showing two auxiliary claims.

Claim 5.7. *Every state of $\mathcal{D}$ is reachable by a word of polynomial length with respect to the size of $\mathcal{A}$.*

Proof. To show this claim, we briefly recall basic definitions and results we need here. For more details, we refer the reader to Krötzsch et al. [39].

Similarly to piecewise testable languages, $\mathcal{R}$-trivial languages can be defined by a congruence $\sim_k^{\mathcal{R}}$ that considers subsequences of prefixes. For $x, y \in \Sigma^*$ and $k \geq 0$, we define $x \sim_k^{\mathcal{R}} y$ if and only if (i) for each prefix u of x, there exists a prefix v of y such that $u \sim_k v$, and (ii) 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 $k \geq 0$. Every $\sim_k^{\mathcal{R}}$ class has a unique minimal representative [9]. It is known that

every k-$\mathcal{R}$-trivial language is also $(k+1)$-$\mathcal{R}$-trivial, and that the language recognized by a complete rpoNFA $\mathcal{B}$ is depth($\mathcal{B}$)-$\mathcal{R}$-trivial.

Let d be the depth of (the completion of) $\mathcal{A}$. Then, the language $L(\mathcal{A})$ is d-$\mathcal{R}$-trivial [39, Theorem 8], and hence there is a congruence $\sim_d^{\mathcal{R}}$, where every $\sim_d^{\mathcal{R}}$ class has a unique minimal representative. Let s be a state of $\mathcal{D}$ and w a word reaching state s from the initial state of $\mathcal{D}$. Let w' denote the unique minimal representative of the $\sim_d^{\mathcal{R}}$-class containing w. If w' leads $\mathcal{D}$ to a state $t \neq s$, then there is u distinguishing s and t in $\mathcal{D}$ because $\mathcal{D}$ is minimal. Since $\sim_d^{\mathcal{R}}$ is a congruence and $w \sim_d^{\mathcal{R}} w'$, we have that $wu \sim_d^{\mathcal{R}} w'u$ and $wu \in L(\mathcal{A})$ if and only if $w'u \notin L(\mathcal{A})$, which is a contradiction to the fact that $L(\mathcal{A})$ is d-$\mathcal{R}$-trivial, *i.e.*, a union of $\sim_d^{\mathcal{R}}$ classes. Therefore, w' leads $\mathcal{D}$ to state s. The length of w' is polynomial, namely $O(d^c)$, where c is the cardinality of Σ [39, Lemma 15]. □

Claim 5.8. *If s is a state of $\mathcal{D}$ reachable by a word w, and $\{b_1, \ldots, b_m\} \subseteq \Sigma(s)$, then, for $v = b_1 b_2 \cdots b_m$, $\delta_{\mathcal{A}}(Q_0, wv^n) = \delta_{\mathcal{A}}(Q_0, wv^{n+1})$, where n is the number of states of $\mathcal{A}$. Moreover, $\{b_1, \ldots, b_m\} \subseteq \Sigma(\delta_{\mathcal{A}}(Q_0, wv^n)) \subseteq \Sigma(s)$.*

Proof. Let Q denote the set of states of $\mathcal{A}$ and extend the partial order of $\mathcal{A}$ to a linear order. Let $Q' \subseteq Q$ be the set of all states with self-loops under all letters of $\{b_1, \ldots, b_m\}$, and let $Q'' = Q \setminus Q' = \{p_1, \ldots, p_{n'}\}$. We assume that $p_1 < p_2 < \ldots < p_{n'}$ in the linear order. Let $\delta_{\mathcal{A}}(Q_0, w) = X_1 \cup Z_1$, where $X_1 \subseteq Q''$ and $Z_1 \subseteq Q'$. Then $X_1 \subseteq \{p_i, p_{i+1}, \ldots, p_{n'}\}$ for some i such that $p_i \in X_1$. Let $X_1 \xrightarrow{v} X_2$. Then the minimal state p_j of X_2 is strictly greater than p_i, since $\mathcal{A}$ is an rpoNFA, and hence $X_2 \subseteq \{p_j, \ldots, p_{n'}\}$ with $j > i$. By induction, we have that $X_1 \xrightarrow{v^n} Z_2$, where $Z_2 \subseteq Q'$. Let $Z = Z_2 \cup Z_1$. Then $\delta_{\mathcal{A}}(Q_0, w) = X_1 \cup Z_1 \xrightarrow{v^n} Z = \delta_{\mathcal{A}}(Q_0, wv^n) \xrightarrow{v} Z = \delta_{\mathcal{A}}(Q_0, wv^{n+1})$.

Since $\{b_1, \ldots, b_m\} \subseteq \Sigma(Z)$, it remains to show that $\Sigma(Z) \subseteq \Sigma(s)$. For the sake of contradiction, assume that there is $a \in \Sigma(Z) \setminus \Sigma(s)$. Then we have that $Z \xrightarrow{a} Z$, and hence, for any $u \in \Sigma^*$, $wv^n u$ belongs to $L(\mathcal{A})$ if and only if $wv^n au$ does. However, in $\mathcal{D}$, $s \xrightarrow{a} s'$ for some $s' \neq s$, and hence, since $v^n \in \Sigma(s)^*$, there is $u \in \Sigma^*$ such that $wv^n u$ belongs to $L(\mathcal{D}) = L(\mathcal{A})$ if and only if $wv^n au$ does not; a contradiction. □

Now, $L(\mathcal{A}) = L(\mathcal{D})$ is not piecewise testable if and only if there are states $s \neq t$ in $\mathcal{D}$ such that s and t are two maximal states of the connected component of $G(\mathcal{D}, \Sigma(s))$ containing s; that is, $\Sigma(s) \subseteq \Sigma(t)$. By Claim 5.7, there are two words w_s and w_t of polynomial length with respect to the size of $\mathcal{A}$ reaching the states s and t of $\mathcal{D}$, respectively. Then, in $\mathcal{A}$, $Q_0 \xrightarrow{w_s} S$ for some $S \subseteq Q$. If $\Sigma(s) = \{b_1, \ldots, b_{c'}\}$, let $v = b_1 \cdots b_{c'}$. Then, by Claim 5.8, $S \xrightarrow{v^n} X_s \xrightarrow{v} X_s$, where n is the number of states of $\mathcal{A}$, and $\Sigma(X_s) = \Sigma(s)$. Analogously, $Q_0 \xrightarrow{w_t} T \xrightarrow{v^n} X_t \xrightarrow{v} X_t$ with $\Sigma(X_s) \subseteq \Sigma(X_t) \subseteq \Sigma(t)$. Furthermore, since the length of v^n is nc', which is polynomial in the size of $\mathcal{A}$, the length of the two words $w_1 = w_s v^n$ and $w_2 = w_t v^n$ is polynomial in the size of $\mathcal{A}$.

Altogether, we have shown that the language of $\mathcal{A}$ is not piecewise testable if and only if there are two different words w_1 and w_2 of polynomial length in the size of $\mathcal{A}$ such that

- $Q_0 \xrightarrow{w_1} X_s$ and $Q_0 \xrightarrow{w_2} X_t$,
- X_s and X_t are maximal with respect to $\Sigma(X_s)$, and
- X_s and X_t are non-equivalent as states of the subset automaton – which can be checked by guessing a word that distinguishes them; by Claim 5.7 applied to $\mathcal{A}$ with the set of initial states X_s (resp. X_t) instead of Q_0, which results in a subautomaton of $\mathcal{D}$, and the

existence of unique minimal representatives of the equivalence classes, *cf.* the proof of the claim, such a word is of polynomial length.

This shows that non-piecewise testability of an rpoNFA-language over a fixed alphabet is in NP, which was to be shown.

To show hardness, we reduce the DNF validity. Let $U = \{x_1, \ldots, x_n\}$ be a set of variables and $\varphi = \varphi_1 \vee \ldots \vee \varphi_m$ be a formula in DNF, where every φ_i is a conjunction of literals. We assume that no φ_i contains both x and $\neg x$. For every $i = 1, \ldots, m$, we define $\beta_i = \beta_{i,1}\beta_{i,2}\ldots\beta_{i,n}$, where

$$\beta_{i,j} = \begin{cases} 0+1 & \text{if neither } x_j \text{ nor } \neg x_j \text{ appear in } \varphi_i \\ 0 & \text{if } \neg x_j \text{ appears in } \varphi_i \\ 1 & \text{if } x_j \text{ appears in } \varphi_i \end{cases}$$

for $j = 1, 2, \ldots, n$. Let $\beta = \sum_{i=1}^{m} \beta_i$. Then $w \in L(\beta)$ if and only if w satisfies some φ_i, that is, $L(\beta) = \{0,1\}^n$ if and only if φ is valid.

We construct an rpoNFA $\mathcal{M}$ as follows. For every β_i, we construct a deterministic path $q_{i,0} \xrightarrow{\beta_{i,1}} q_{i,1} \xrightarrow{\beta_{i,2}} q_{i,2} \cdots \xrightarrow{\beta_{i,n}} q_{i,n} \xrightarrow{0,1} q_{i,n}$ with a self-loop at the end; if $\beta_{i,k} = 0+1$, the notation means that there are two transitions under both letters 0 and 1. Then we add a path $\alpha_1 \xrightarrow{0,1} \alpha_2 \xrightarrow{0,1} \ldots \xrightarrow{0,1} \alpha_n$ to accept all words of length less than n. The automaton $\mathcal{M}$ consists of these paths, where the initial states are $\{q_{i,0} \mid i = 1, \ldots, m\} \cup \{\alpha_1\}$ and the accepting states are $\{q_{i,n} \mid i = 1, \ldots, m\} \cup \{\alpha_1, \ldots, \alpha_n\}$. Notice that $\mathcal{M}$ is an rpoNFA accepting the language $L(\mathcal{M}) = L(\beta)\{0,1\}^* \cup \{w \in \{0,1\}^* \mid |w| < n\}$.

If $L(\beta) = \{0,1\}^n$, then $L(\mathcal{M}) = \{0,1\}^*$ is piecewise testable.

If $L(\beta) \neq \{0,1\}^n$, we show that $L(\mathcal{M})$ is not piecewise testable using the UMS property on the minimal DFA equivalent to $\mathcal{M}$. Since $\mathcal{M}$ is an rpoNFA, the minimal DFA is partially ordered. By the assumption that $L(\beta) \neq \{0,1\}^n$, there is a $w \in \{0,1\}^n$ such that $w\{0,1\}^* \cap L(\mathcal{M}) = \emptyset$. Since $L(\beta) \neq \emptyset$ by the construction, there is $w' \in L(\beta)$, which implies that $w'\{0,1\}^* \subseteq L(\mathcal{M})$. Since no word of $w\{0,1\}^*$ is accepted by $\mathcal{M}$, there is a path from the initial state of the minimal DFA to a rejecting state, say q_r, that is maximal under $\{0,1\}$. Similarly, since all words of $w'\{0,1\}^*$ are accepted by $\mathcal{M}$, there is a path in the minimal DFA to an accepting state, say q_a, that is maximal with respect to $\{0,1\}$. But then q_a and q_r are two maximal states violating the UMS property of the minimal DFA.

Thus, $L(\mathcal{M})$ is piecewise testable if and only if φ is valid. □

6. Inclusion and Equivalence

A consequence of the complexity of universality is the worst-case lower-bound complexity for the inclusion and equivalence problems. These problems are of interest, *e.g.*, in optimization. The problems ask, given languages K and L, whether $K \subseteq L$, resp. $K = L$. Although equivalence means two inclusions, complexities of these two problems may differ significantly, *e.g.*, inclusion is undecidable for deterministic context-free languages [22] while equivalence is decidable [71].

Since universality can be expressed as the inclusion $\Sigma^* \subseteq L$ or the equivalence $\Sigma^* = L$, we immediately obtain the hardness results for inclusion and equivalence from the results for universality. Therefore, it remains to show memberships of our results summarized in Tables 9 and 10.

	B			
A	DFA	ptNFA & rpoNFA	poNFA	NFA
DFA	L/NL	NL/coNP/PSpace	NL/PSpace	coNP/PSpace
ptNFA	NL	NL/coNP/PSpace	NL/PSpace	coNP/PSpace
rpoNFA	NL	NL/coNP/PSpace	NL/PSpace	coNP/PSpace
poNFA	NL	NL/coNP/PSpace	NL/PSpace	coNP/PSpace
NFA	NL	NL/coNP/PSpace	NL/PSpace	coNP/PSpace

Table 9: Complexity of deciding inclusion $L(A) \subseteq L(B)$ (unary/fixed[/growing] alphabet), all results are complete for the given class.

	DFA	ptNFA & rpoNFA	poNFA	NFA
DFA	L/NL	NL/coNP/PSpace	NL/PSpace	coNP/PSpace
ptNFA		NL/coNP/PSpace	NL/PSpace	coNP/PSpace
rpoNFA		NL/coNP/PSpace	NL/PSpace	coNP/PSpace
poNFA			NL/PSpace	coNP/PSpace
NFA				coNP/PSpace

Table 10: Complexity of deciding equivalence (unary/fixed[/growing] alphabet), the problems are complete for the given classes.

6.1. **Proofs.** Let A be an automaton of any of the considered types. We now discuss the cases depending on the type of B. We assume that both automata are over the same alphabet specified by B.

If B is a DFA, then $L(A) \subseteq L(B)$ if and only if $L(A) \cap L(\overline{B}) = \emptyset$, which can be checked in NL (or in L if both automata are unary DFAs), where $\overline{B}$ denotes the DFA obtained by complementing B. This covers the first column of Table 9.

If B is an rpoNFA over a fixed alphabet, then deciding $L(A) \subseteq L(B)$ is in coNP [39, Theorem 23]. Furthermore, the case of a unary alphabet follows from the case of unary poNFAs, and the case of a growing alphabet from the case of general NFAs discussed below.

If B is a unary poNFA, we distinguish several cases. First, deciding whether the language of an NFA is finite is in NL. Thus, if $L(A)$ is infinite and $L(B)$ finite, the inclusion does not hold. If both the languages are finite, then the number of words is bounded by the number of states, and hence the inclusion can be decided in NL. If $L(B)$ is infinite, then there is n bounded by the number of states of B such that $L(B)$ contains all words of length at least n. Thus, the inclusion does not hold if and only if there is a word of length at most n in $L(A)$ that is not in $L(B)$, which can again be checked in NL.

If B is an NFA, then deciding $L(A) \subseteq L(B)$ is in PSpace using the standard on-the-fly computation of $\overline{B}$ and deciding $L(A) \cap L(\overline{B}) = \emptyset$.

If B is a unary NFA, then if $L(B)$ is finite, we proceed as in the case of B being a unary poNFA. Therefore, assume that $L(B)$ is infinite and B has n states. Then the minimal DFA recognizing $L(B)$ has at most 2^n states (a better bound is given by Chrobak [14]). If the

inclusion $L(A) \subseteq L(B)$ does not hold and A has m states, then there exists $k \leq m \cdot 2^n$, the number of states of $A \times \overline{B}$, such that $a^k \in L(A) \setminus L(B)$. We can guess k in binary and verify that the inclusion does not hold in polynomial time by computing the reachable states under a^k using the matrix multiplication. Hence, checking that the inclusion holds is in coNP.

Notice that the upper-bound complexity for equivalence follows immediately from the upper-bound complexity for inclusion, which completes this section.

7. Conclusion

We studied the complexity of deciding universality for ptNFAs, a type of nondeterministic finite automata the expressivity of which coincides with level 1 of the Straubing-Thérien hierarchy. Our proof showing PSpace-completeness required a novel and nontrivial extension of our recent construction for self-loop-deterministic poNFAs. Consequently, we obtained PSpace-completeness for several restricted types of poNFAs for problems including inclusion, equivalence, and (k-)piecewise testability.

Acknowledgements. We thank O. Klíma, M. Kunc and L. Polák for providing us with their manuscript [34], and we gratefully acknowledge very useful suggestions and comments of the anonymous referees.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
[2] J. Almeida, J. Bartoňová, O. Klíma, and M. Kunc. On decidability of intermediate levels of concatenation hierarchies. In *Developments in Language Theory (DLT)*, volume 9168 of *LNCS*, pages 58–70. Springer, 2015.
[3] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, 2008.
[4] P. Barceló, L. Libkin, and J. L. Reutter. Querying regular graph patterns. *Journal of the ACM*, 61:8:1–8:54, 2014.
[5] M. Bojańczyk. The common fragment of ACTL and LTL. In *Foundations of Software Science and Computational Structures (FOSSACS)*, volume 4962 of *LNCS*, pages 172–185. Springer, 2008.
[6] M. Bojańczyk, L. Segoufin, and H. Straubing. Piecewise testable tree languages. *LMCS*, 8, 2012.
[7] A. Bouajjani, A. Muscholl, and T. Touilim. Permutation rewriting and algorithmic verification. *Information and Computation*, 205:199–224, 2007.
[8] J. W. Bryans, M. Koutny, and P. Y. A. Ryan. Modelling opacity using petri nets. *Electronic Notes in Theoretical Computer Science*, 121:101–115, 2005.
[9] J. A. Brzozowski and F. E. Fich. Languages of R-trivial monoids. *Journal of Computer and System Sciences*, 20:32–49, 1980.
[10] J. A. Brzozowski and R. Knast. The dot-depth hierarchy of star-free languages is infinite. *Journal of Computer and System Sciences*, 16:37–55, 1978.
[11] P. E. Caines, R. Greiner, and S. Wang. Dynamical logic observers for finite automata. In *Conference on Decision and Control (CDC)*, pages 226–233, 1988.
[12] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Reasoning on regular path queries. *ACM SIGMOD Record*, 32:83–92, 2003.
[13] S. Cho and D. T. Huynh. Finite-automaton aperiodicity is PSPACE-complete. *Theoretical Computer Science*, 88:99–116, 1991.
[14] M. Chrobak. Finite automata and unary languages. *Theoretical Computer Science*, 47:149–158, 1986. Errata: Theoretical Computer Science 302 (2003) 497-498.
[15] R. S. Cohen and J. A. Brzozowski. Dot-depth of star-free events. *Journal of Computer and System Sciences*, 5:1–16, 1971.

[16] W. Craig. Linear reasoning. A new form of the herbrand-gentzen theorem. *Journal of Symbolic Logic*, 22(3):250–268, 1957.
[17] W. Czerwiński, W. Martens, and T. Masopust. Efficient separability of regular languages by subsequences and suffixes. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 7966 of *LNCS*, pages 150–161, 2013.
[18] C. Dax and F. Klaedtke. Alternation elimination by complementation (extended abstract). In *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 5330 of *LNCS*, pages 214–229. Springer, 2008.
[19] V. Diekert, P. Gastin, and M. Kufleitner. A survey on small fragments of first-order logic over finite words. *Int. J. Found. Comput. Sci.*, 19:513–548, 2008.
[20] R. Ehlers, S. Lafortune, S. Tripakis, and M. Y. Vardi. Supervisory control and reactive synthesis: a comparative introduction. *Discrete Event Dynamic Systems*, 27(2):209–260, 2017.
[21] Keith Ellul, Bryan Krawetz, Jeffrey Shallit, and Ming-Wei Wang. Regular expressions: New results and open problems. *J. Autom. Lang. Comb.*, 10:407–437, 2005.
[22] E. P. Friedman. The inclusion problem for simple languages. *Theoretical Computer Science*, 1:297–316, 1976.
[23] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
[24] C. Glaßer and H. Schmitz. Languages of dot-depth 3/2. *Theory of Computing Systems*, 42:256–286, 2008.
[25] J. A. Green. On the structure of semigroups. *Ann. of Math. (2)*, 54:163–172, 1951.
[26] N. Grosshans, P. McKenzie, and L. Segoufin. The power of programs over monoids in DA. In *Mathematical Foundations of Computer Science (MFCS)*, volume 83 of *LIPIcs*, pages 2:1–2:20. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.
[27] D. Harel and A. Pnueli. On the development of reactive systems. In *Logics and Models of Concurrent Systems*, pages 477–498, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.
[28] P.-C. Héam. A note on partially ordered tree automata. *Inform. Process. Lett.*, 108:242–246, 2008.
[29] P. Hofman and W. Martens. Separability by short subsequences and subwords. In *International Conference on Database Theory (ICDT)*, volume 31 of *LIPIcs*, pages 230–246, 2015.
[30] H. B. Hunt III. *On the Time and Tape Complexity of Languages*. PhD thesis, Department of Computer Science, Cornell University, Ithaca, NY, 1973.
[31] R. Jacob, J.-J. Lesage, and J.-M. Faure. Overview of discrete event systems opacity: Models, validation, and quantification. *Annual Reviews in Control*, 41:135–146, 2016.
[32] N. D. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 11:68–85, 1975.
[33] P. Karandikar and Ph. Schnoebelen. The height of piecewise-testable languages and the complexity of the logic of subwords. *Logical Methods in Computer Science*, 15(2), 2019.
[34] O. Klíma, M. Kunc, and L. Polák. Deciding k-piecewise testability. Manuscript, 2014.
[35] O. Klíma and L. Polák. Alternative automata characterization of piecewise testable languages. In *Developments in Language Theory (DLT)*, volume 7907 of *LNCS*, pages 289–300, 2013.
[36] J. Komenda and T. Masopust. Computation of controllable and coobservable sublanguages in decentralized supervisory control via communication. *Discrete Event Dynamic Systems*, 27(4):585–608, 2017.
[37] J. Komenda, T. Masopust, and J. H. van Schuppen. Coordination control of discrete-event systems revisited. *Discrete Event Dynamic Systems*, 25(1-2):65–94, 2015.
[38] D. Kozen. Lower bounds for natural proof systems. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 254–266. IEEE Computer Society, 1977.
[39] M. Krötzsch, T. Masopust, and M. Thomazo. Complexity of universality and related problems for partially ordered NFAs. *Information and Computation*, 255:177–192, 2017.
[40] M. Kufleitner and A. Lauser. Partially ordered two-way Büchi automata. *Int. J. Found. Comput. Sci.*, 22:1861–1876, 2011.
[41] O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Log.*, 2(3):408–429, 2001.
[42] F. Lin. Opacity of discrete event systems and its applications. *Automatica*, 47(3):496–503, 2011.

[43] K. Lodaya, P. K. Pandya, and S. S. Shah. Marking the chops: an unambiguous temporal logic. In *IFIP International Conference On Theoretical Computer Science (TCS)*, volume 273 of *IFIP*, pages 461–476. Springer, 2008.
[44] K. Lodaya, P. K. Pandya, and S. S. Shah. Around dot depth two. In *Developments in Language Theory (DLT)*, volume 6224 of *LNCS*, pages 303–315. Springer, 2010.
[45] M. Maidl. The common fragment of CTL and LTL. In *Foundations of Computer Science (FOCS)*, pages 643–652. IEEE Computer Society, 2000.
[46] W. Martens, F. Neven, M. Niewerth, and T. Schwentick. Bonxai: Combining the simplicity of DTD with the expressiveness of XML schema. In *Principles of Database Systems (PODS)*, pages 145–156. ACM, 2015.
[47] W. Martens, F. Neven, and T. Schwentick. Complexity of decision problems for XML schemas and chain regular expressions. *SIAM J. Comput.*, 39:1486–1530, 2009.
[48] W. Martens and T. Trautner. Evaluation and enumeration problems for regular path queries. In *International Conference on Database Theory (ICDT)*, volume 98 of *LIPIcs*, pages 19:1–19:21. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
[49] T. Masopust. Piecewise testable languages and nondeterministic automata. In *Mathematical Foundations of Computer Science (MFCS)*, volume 58 of *LIPIcs*, pages 67:1–67:14, 2016.
[50] T. Masopust. Complexity of deciding detectability in discrete event systems. *Automatica*, 93:257–261, 2018.
[51] T. Masopust. Separability by piecewise testable languages is PTime-complete. *Theoretical Computer Science*, 711:109–114, 2018.
[52] T. Masopust and M. Krötzsch. Deciding universality of ptNFAs is PSpace-complete. In *International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, volume 10706 of *LNCS*, pages 413–427. Springer, 2018.
[53] T. Masopust and M. Thomazo. On boolean combinations forming piecewise testable languages. *Theoretical Computer Science*, 682:165–179, 2017.
[54] T. Masopust and X. Yin. Complexity of detectability, opacity and A-diagnosability for modular discrete event systems. *Automatica*, 101:290–295, 2019.
[55] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Symposium on Switching and Automata Theory (STOC)*, pages 125–129. IEEE Computer Society, 1972.
[56] D. E. Muller, A. Saoudi, and P. E. Schupp. Alternating automata, the weak monadic theory of trees and its complexity. *Theor. Comput. Sci.*, 97(2):233–244, 1992.
[57] C. M. Ozveren and A. S. Willsky. Observability of discrete event dynamic systems. *IEEE Transactions on Automatic Control*, 35(7):797–806, 1990.
[58] J.-E. Pin. Mathematical foundations of automata theory. http://www.irif.fr/ jep/PDF/MPRI/MPRI.pdf, 2019.
[59] J.-E. Pin and P. Weil. Polynomial closure and unambiguous product. *Theory Comput. Syst.*, 30(4):383–422, 1997.
[60] T. Place. Separating regular languages with two quantifiers alternations. In *LICS*, pages 202–213. IEEE Computer Society, 2015.
[61] T. Place. Separating regular languages with two quantifier alternations. *Logical Methods in Computer Science*, 14(4), 2018.
[62] T. Place, L. van Rooijen, and M. Zeitoun. On separation by locally testable and locally threshold testable languages. *Logical Methods in Computer Science*, 10(3), 2014.
[63] T. Place and M. Zeitoun. Separation and the successor relation. In *STACS*, volume 30 of *LIPIcs*, pages 662–675, 2015.
[64] T. Place and M. Zeitoun. The covering problem. *Logical Methods in Computer Science*, 14(3), 2018.
[65] P. J. Ramadge. Observability of discrete event systems. In *Conference on Decision and Control (CDC)*, pages 1108–1112, 1986.
[66] A. Rybalchenko and V. Sofronie-Stokkermans. Constraint solving for interpolation. *Journal of Symbolic Computation*, 45(11):1212–1233, 2010.
[67] A. Saboori and C. N. Hadjicostis. Notions of security and opacity in discrete event systems. In *Conference on Decision and Control (CDC)*, pages 5056–5061, 2007.

[68] A.-K. Schmuck, Th. Moor, and R. Majumdar. On the relation between reactive synthesis and supervisory control of input/output behaviours. *IFAC-PapersOnLine*, 51(7):31–38, 2018.
[69] M. P. Schützenberger. Sur le produit de concatenation non ambigu. *Semigroup Forum*, 13:47–75, 1976.
[70] T. Schwentick, D. Thérien, and H. Vollmer. Partially-ordered two-way automata: A new characterization of DA. In *Developments in Language Theory (DLT)*, volume 2295 of *LNCS*, pages 239–250, 2001.
[71] G. Sénizergues. The equivalence problem for deterministic pushdown automata is decidable. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 1256 of *LNCS*, pages 671–681, 1997.
[72] S. Shu and F. Lin. Generalized detectability for discrete event systems. *Systems & Control Letters*, 60(5):310–317, 2011.
[73] S. Shu, F. Lin, and H. Ying. Detectability of discrete event systems. *IEEE Transactions on Automatic Control*, 52(12):2356–2359, 2007.
[74] I. Simon. *Hierarchies of Events with Dot-Depth One*. PhD thesis, University of Waterloo, Canada, 1972.
[75] A. P. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for büchi automata with appplications to temporal logic. *Theor. Comput. Sci.*, 49:217–237, 1987.
[76] G. Stefanoni, B. Motik, M. Krötzsch, and S. Rudolph. The complexity of answering conjunctive and navigational queries over OWL 2 EL knowledge bases. *Journal of Artificial Intelligence Research*, 51:645–705, 2014.
[77] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary report. In *ACM Symposium on the Theory of Computing (STOC)*, pages 1–9, 1973.
[78] H. Straubing. A generalization of the Schützenberger product of finite monoids. *Theoretical Computer Science*, 13:137–150, 1981.
[79] H. Straubing. Finite semigroup varieties of the form V*D. *J. Pure Appl. Algebra*, 36:53–94, 1985.
[80] D. Thérien. Classification of finite monoids: The language approach. *Theoretical Computer Science*, 14:195–208, 1981.
[81] D. Thérien and Th. Wilke. Over words, two variables are as powerful as one quantifier alternation. In *ACM Symposium on the Theory of Computing (STOC)*, pages 234–240. ACM, 1998.
[82] A. N. Trahtman. Piecewise and local threshold testability of DFA. In *International Symposium on Fundamentals of Computation Theory (FCT)*, volume 2138 of *LNCS*, pages 347–358, 2001.
[83] K. W. Wagner. Leaf language classes. In *MCU*, volume 3354 of *LNCS*, pages 60–81, 2004.