Partially Ordered Automata

Expressivity, Complexity, and Applications

Habilitation Thesis

2019 **RNDr. Tomáš Masopust, Ph.D., DSc.**

Abstract

A nondeterministic finite automaton (NFA) is partially ordered if the only cycles in its transition diagram are self-loops. Expressivity of partially ordered NFAs (poNFAs) can be characterized by the Straubing-Thérien hierarchy. The most studied level of the hierarchy, level 1, is formed by *piecewise testable languages* – regular languages recognized by *confluent partially ordered DFAs*. Omitting confluence results in *partially ordered DFAs* characterizing $\mathscr{R}$*-trivial languages*, a class of languages strictly between levels 1 and $\frac{3}{2}$ of the hierarchy. Lifting the notion from DFAs to NFAs, Schwentick, Thérien and Vollmer showed that poNFAs characterize level $\frac{3}{2}$ of the hierarchy. We extend this study and provide an NFA characterization of $\mathscr{R}$-trivial languages and piecewise testable languages.

Our motivation to study partially ordered automata comes from database theory and schema languages for XML data, namely from scenarios in which we want to describe something complex by means of a simple language. The technical core consists of *separation* problems that are usually of the form "Given two languages K and L, does there exist a language S, coming from a family $\mathscr{F}$ of *simple* languages, such that S contains everything from K and nothing from L?" We focus on the case where "simple languages" are represented by piecewise testable languages. We show that the separation problem of regular languages by piecewise testable languages is PTIME-complete. Our construction is based on the non-existence of a particular infinite sequence of words alternating between the languages, called a tower. The number of words in the tower is its height. We show that two languages are separable if and only if there is no infinite tower between them. The height is further closely related to the complexity of computing a separator. We show that the upper bound on the height of finite towers is polynomial in the number of states and exponential in the size of the alphabet, and that it is asymptotically tight if the size of the alphabet is fixed. If the alphabet may grow linearly with the number of states, then the lower bound on the height of towers is exponential with respect to that number.

Another motivation comes from discrete event systems. On some level of abstraction, discrete event systems can be seen as finite automata with possible additional properties. One of the fundamental properties is that the system is deadlock free. In some sense, poNFAs are the simplest deadlock-free models, and hence the study of their lower-bound complexity questions covers most of the practical cases. The problems of our interest are the questions whether the behavior of two systems is equivalent or in inclusion. Since the lower-bound complexity of the problems is covered by the complexity of universality, we primarily investigate the question whether the behavior of a system is universal. We further discuss consequences of our results to the verification of detectability and opacity of discrete event systems.

Acknowledgements

I would like to thank all my co-authors who worked with me on the topics of this thesis. Special thanks go to the Department of Computer Science of the Palacky University, to the Institute of Mathematics of the Czech Academy of Sciences, to the Institute of applied informatics of the Bayreuth University, and to the Faculty of Informatics of the Technical University Dresden who maintained a pleasant and flexible environment for my research.

During the time, my research was supported by the following projects:

- by the Czech Science Foundation project P202/11/P028 (*Decentralized and coordination supervisory control*),
- by the Czech Science Foundation project GA15-02532S (*Modular and Decentralized Control of Discrete-Event and Hybrid Systems with Communication*),
- by the Ministry of Education, Youth, and Sport of the Czech Republic under the Kontakt II research project LH13012 (*Multi-level supervisory control (MUSIC)*),
- by the German Research Foundation (DFG) in Emmy Noether grant MA 4938/2-1 (*Querschnitte: XML und formale Sprachen – Theorie und Praxis*),
- by the German Research Foundation (DFG) in Emmy Noether grant KR 4381/1-1 (*Datenintegration und -abfrage durch die Zusammenführung von Ontologien und Datenbanken*),
- and by the Czech Science Foundation project GC19-06175J (*Compositional Methods for the Control of Concurrent Timed Discrete-Event Systems*).

My greatest thanks go to my wife Iveta and my daughter Sofie for their infinite patience and support.

Contents

1 Introduction **3**
1.1 Papers in the Collection 6

2 Basic Notation **8**

3 Descriptional Complexity **9**
3.1 Reversal 9
3.1.1 Reversal of $\mathscr{R}$-trivial Languages 9
3.1.2 Reversal of $\mathscr{J}$-trivial Languages 10

4 Separability **12**
4.1 Separability of Languages 13
4.2 Asymmetric Separation and Suffix Order 15
4.3 Computation of Piecewise Testable Separators 16
4.3.1 Bounds on the Height of Towers 17

5 Complexity of Universality **20**
5.1 Partially Ordered NFAs 22
5.2 Restricted Partially Ordered NFAs 22
5.2.1 Deciding Universality of rpoNFAs 24
5.3 PtNFAs 25
5.3.1 Deciding Universality for ptNFAs 25
5.4 Inclusion and Equivalence of Partially Ordered NFAs 28

6 Piecewise Testable Languages **30**
6.1 Finding a Boolean Combination 30
6.1.1 Step 1: Checking Piecewise Testability 31
6.1.2 Step 2: Computing the Minimal k 31
6.1.3 Step 3: Computation of Representatives 33
6.2 Piecewise Testability and Nondeterministic Automata 33
6.3 Complexity 34
6.3.1 Complexity of Deciding k-Piecewise Testability 34
6.3.2 Complexity of Deciding Piecewise Testability 35

7 Applications **37**
7.1 Deterministic Regular Expressions 37
7.2 Detectability 38
7.3 Opacity 40

8 Conclusion **44**

1. Introduction

The relationship between logic and formal languages is studied for decades. In 60th, Büchi [22] and Elgot [40] showed that a language is *regular* if and only if it is definable in *monadic second order logic*. Later, McNaughton and Papert [87] proved that *first-order logic* describes *star-free languages*, a class of regular languages for which Schützenberger [109] gave an algebraic characterization – star-free languages are those regular languages whose syntactic monoid is *aperiodic*.

Restricting the number of quantifier alternations in first-order logic formulae in the prenex normal form results in a so-called *quantifier alternation hierarchy*. The choice of predicates in the signature of the first-order logic then corresponds to several such hierarchies. The well-known and closely related hierarchies are the *Straubing-Thérien hierarchy* [122, 124] and the *dot-depth hierarchy* [20, 29, 123].

The Straubing-Thérien hierarchy is alternatively defined as follows. For an alphabet Σ, $\mathscr{L}(0) = \{\emptyset, \Sigma^*\}$ and, for integers $n \geq 0$, the levels $\mathscr{L}(n+\frac{1}{2})$ and $\mathscr{L}(n+1)$ are defined so that

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

The hierarchy does not collapse on any level [20].

There is a constant interest in automata characterizations of the levels of the Straubing-Thérien hierarchy, in particular in decision and complexity questions, such as the membership of a language in a specific level of the hierarchy. Despite a recent progress [2, 97, 99], deciding whether a language belongs to level k of the Straubing-Thérien hierarchy is still open for $k > \frac{7}{2}$. The Straubing-Thérien hierarchy further has close relations to the dot-depth hierarchy and to complexity theory [128].

The most studied level of the Straubing-Thérien hierarchy is level 1, also known as *piecewise testable languages* introduced by Simon [115]. Simon showed that piecewise testable languages are those regular languages whose syntactic monoid is $\mathscr{J}$-trivial and that they are recognized by *confluent partially ordered DFAs*. An automaton is *partially ordered* if its transition relation induces a partial order on states – the only cycles in its transition diagram are self-loops – and it is *confluent* if for any state q and any two of its successor states s and t directly accessible from q by transitions labeled by a and b, respectively, there is a word w over the alphabet $\{a,b\}$ such that a common state is reachable from both states s and t under w, see Figure 1.1 for an illustration.

Omitting confluence from the definition results in *partially ordered DFAs* (poDFAs) studied by Brzozowski and Fich [19]. They showed that poDFAs characterize *$\mathscr{R}$-trivial languages*, a class of languages strictly between level 1 and level $\frac{3}{2}$ of the Straubing-Thérien hierarchy (the name comes from the Green's $\mathscr{R}$ relation because the syntactic monoid of $\mathscr{R}$-trivial languages

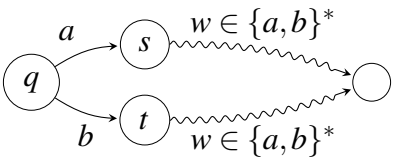

Figure 1.1: Confluence

is $\mathscr{R}$-trivial). Lifting the notion from DFAs to NFAs, Schwentick, Thérien and Vollmer [111] showed that partially ordered NFAs (poNFAs) characterize level $\frac{3}{2}$ of the Straubing-Thérien hierarchy. Hence poNFAs are more expressive than poDFAs. Languages of level $\frac{3}{2}$ are also known in the literature as *Alphabetical Pattern Constraints*, which are regular languages effectively closed under permutation rewriting [14].

Motivated by Brzozowski's minimization algorithm based on the double computation of the reverse operation, we study the state complexity of the reverse of minimal poDFAs. State complexity of an automaton or of a language is the number of states of the minimal DFA recognizing the language. We establish a tight bound on the state complexity of the reverse of poDFAs and confluent poDFAs showing that the state complexity of the reverse of a (confluent) poDFA of the state complexity n is 2^{n-1}. The witness is ternary for poDFAs and $(n-1)$-ary for confluent poDFAs, and the bound can be met neither by a binary poDFA nor by a confluent poDFA over an $(n-2)$-element alphabet. We further provide a characterization of the tight bounds for poDFAs depending on the state complexity and the size of its alphabet. Chapter 3 is devoted to the overview of this study.

Our main interest in partially ordered automata is in particular motivated by two problems.

The first problem 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 are motivated by scenarios in which we want to describe something complex by means of a simple language. The technical core of our 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 $\mathscr{F}$ of *simple* languages, such that S contains everything from K and nothing from L?" The family $\mathscr{F}$ of simple languages could be, for example, languages definable in (a fragment of) first-order logic, piecewise testable languages, or languages definable with a special class of automata.

We focus on simple languages represented by piecewise testable languages and their special variants, so-called k-piecewise testable languages. We show that the separation problem of regular languages by piecewise testable languages is PTIME-complete. Our construction showing the membership in PTIME is based on the non-existence of a particular infinite sequence of words, called a tower. A *tower* is a sequence of words alternating between two languages in such a way that every word is a subsequence of the following word. The height of a tower is the number of words in the sequence. If there is no tower of infinite height, then the height of all towers between the languages is bounded. We show that two languages are separable if and only if there is no infinite tower between them. The height of highest towers is closely related to the complexity of computing a *separator*. Therefore, we further investigate upper and lower bounds on the height of maximal finite towers. We show that the upper bound is polynomial in the number of states and exponential in the size of the alphabet, and that it is asymptotically tight if the size of the alphabet is fixed. If the alphabet may grow linearly with the number of states, then the lower bound on the height of towers is exponential with respect to that number. In this case, there is a gap between the lower bound and the upper bound, and the asymptotically optimal bound remains open. Chapter 4 is devoted to these problems.

A regular language is k-piecewise testable if it is a finite boolean combination of languages of the form $\Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$, where $a_i \in \Sigma$ and $0 \leq n \leq k$. Given a DFA $\mathscr{A}$ and $k \geq 0$, it is an NL-

complete problem to decide whether the language $L(\mathscr{A})$ is piecewise testable and, for $k \geq 4$, it is coNP-complete to decide whether the language $L(\mathscr{A})$ is k-piecewise testable [67]. We discuss the complexity for $k < 4$. Furthermore, it is known that the depth of the minimal DFA equivalent to $\mathscr{A}$ serves as an upper bound on k; if $L(\mathscr{A})$ is piecewise testable, then it is k-piecewise testable for k equal to the depth of $\mathscr{A}$ [68]. We show that some form of nondeterminism does not violate this upper bound result. Specifically, we define a class of self-loop deterministic poNFAs, called ptNFAs, that characterize piecewise testable languages and show that the depth of a ptNFA provides an (up to exponentially better) upper bound on k than the minimal DFA. Furthermore, if the language $L(\mathscr{A})$ is piecewise testable, we want to express it as a Boolean combination of languages of the above form. Our idea is as follows. If the language is piecewise testable, then it is k-piecewise testable for some k, and hence there is a congruence $\sim_k$ of finite index such that $L(\mathscr{A})$ is a finite union of $\sim_k$-classes. Each class is characterized by an intersection of languages of the from $\Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$, where $n \leq k$, and their complements. To represent the $\sim_k$-classes, we make use of the $\sim_k$-canonical DFA. We identify the states of the $\sim_k$-canonical DFA whose union forms the language $L(\mathscr{A})$ and use them to construct the required Boolean combination. We overview these problems and results in Chapter 6.

Our second motivation comes from *discrete event systems*. On some level of abstraction, discrete event systems can be seen as finite automata with some additional properties. A fundamental property is, e. g., a requirement that the system is *deadlock free*. In some sense, poNFAs are the simplest deadlock-free models, and hence the study of the lower-bound complexity questions for these models covers most of the practical cases. The problems of our interest are the questions whether the behaviors of two systems are equivalent or in inclusion. Since the lower-bound complexity of these problems is covered by the complexity of universality, we focus on the question whether the behavior of the system is universal. An automaton is universal if it accepts all words over its alphabet. Deciding universality is well known to be PSPACE-complete for NFAs and regular expressions [1], and the same proof actually shows PSPACE-completeness for poNFAs. We improve the result by showing that it remains true even when restricting to a fixed (binary) alphabet. This is already nontrivial since the standard encodings of alphabet symbols in, e. g., binary can turn self-loops into longer cycles. A lower, coNP-complete complexity bound can be obtained if we require that all self-loops are deterministic in the sense that the symbol read in the loop cannot occur in any other transition from that state. We find that such restricted poNFAs (rpoNFAs) characterize the class of $\mathscr{R}$-trivial languages, and we establish the complexity of deciding whether the language of an NFA is $\mathscr{R}$-trivial. The limitation to fixed alphabets turns out to be essential even in the restricted case: deciding universality of rpoNFAs with unbounded alphabets is PSPACE-complete. From the practical point of view, the languages of rpoNFAs are definable by deterministic (one-unambiguous) regular expressions, which makes them interesting in schema languages for XML data. Using a nontrivial extension of the proofs for rpoNFAs, we show the same complexity of the universality problem for ptNFAs. This strengthens the previous result and provides a new lower-bound complexity for some other problems, including inclusion, equivalence, and k-piecewise testability. Chapter 5 is devoted to this topic.

Finally, in Chapter 7, we show several consequences of our previous results in verification of system-theoretic properties of discrete event systems – detectability and opacity. *Detectability* arises in the state estimation of systems and asks whether the current state of the system can be determined unambiguously after a finite number of observations via some or all trajectories. *Opacity* is related to the privacy and security analysis. The system has a secret modeled as a set of secret states and an intruder is modeled as a passive observer with limited observation. The system is opaque if the intruder never knows for sure that the system is in a secret state, i. e., the secret is not revealed.

1.1 Papers in the Collection

This thesis is based on the papers listed below, with full versions attached in the appendices.

[34] W. Czerwinski, W. Martens, and T. Masopust. Efficient separability of regular languages by subsequences and suffixes. *Proc. of the 40th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 7966 of *LNCS*, pages 150–161. Springer, 2013.

Contribution: 33 %. General discussions of the problems and solutions, ideas of some proofs, including the use of the union-free decomposition, wrote some parts.

[54] Š. Holub, G. Jirásková, and T. Masopust. On upper and lower bounds on the length of alternating towers. *Proc. of the 39th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 8634 of *LNCS*, pages 315–326. Springer, 2014.

Contribution: 40 %. Author of the computation of a separator, of the upper-bound result, and of the exponential lower-bound result; main ideas of the proofs; wrote the paper.

[55] Š. Holub, T. Masopust, and M. Thomazo. On the height of towers of subsequences and prefixes. *Information and Computation* 265:77–93, 2019.

This paper is an extended and full version of paper [54].

[60] G. Jirásková and T. Masopust. On the state and computational complexity of the reverse of acyclic minimal DFAs. *Proc. of the 17th International Conference on Implementation and Application of Automata (CIAA)*, volume 7381 of *LNCS*, pages 229–239. Springer, 2012.

Contribution: 50 %. Main ideas of some proofs; writing.

[61] G. Jirásková and T. Masopust. On the state complexity of the reverse of $\mathscr{R}$- and $\mathscr{J}$-trivial regular languages. *Proc. of the 15th International Workshop on Descriptional Complexity of Formal Systems (DCFS)*, volume 8031 of *LNCS*, pages 136–147. Springer, 2013.

Contribution: 50 %. Author of the upper-bound and some lower-bound results; wrote the paper.

[72] M. Krötzsch, T. Masopust, and M. Thomazo. Complexity of universality and related problems for partially ordered NFAs. *Information and Computation* 255:177–192, 2017.

Contribution: 60 %. Main author; main ideas of the proofs; wrote the paper; the main contribution of the coauthors is in the PSPACE-hardness proof of Theorem 28 and in improving the text.

[81] T. Masopust. Piecewise testable languages and nondeterministic automata. *Proc. of the 41st International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 58 of *LIPIcs*, pages 67:1–67:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[82] T. Masopust. Complexity of deciding detectability in discrete event systems. *Automatica* 93:257–261, 2018.

[83] T. Masopust. Separability by piecewise testable languages is PTIME-complete. *Theoretical Computer Science* 711:109–114, 2018.

[84] T. Masopust and M. Krötzsch. Universality of ptNFAs is PSPACE-complete. *Proc. of 44th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, volume 10706 of *LNCS*, pages 413–427, Springer, 2018.

Contribution: 60 %. Main author; main ideas of the proofs; wrote the paper; coauthor's contribution in the PSPACE-hardness proof of Theorem 2.

[85] T. Masopust and M. Krötzsch. Partially ordered automata and piecewise testability. Manuscript, 2019.

This manuscript is a completely revised and extended version of my conference paper [81], and of the previous conference paper [84].

Contribution: 75 %. Main author; main ideas of the proofs; wrote the paper; coauthor's contribution in the PSPACE-hardness proof of the universality for ptNFAs.

[86] T. Masopust and M. Thomazo. On boolean combinations forming piecewise testable languages. *Theoretical Computer Science* 682:165–179, 2017.

Contribution: 50 %. Main author; Lemmas 5, 9, 10, Theorem 31, and Proposition 32 obtained in fruitful discussions; main ideas of some proofs; wrote the paper.

2. Basic Notation

In this section, we briefly unify the basic notation used in this thesis. Specific notions are introduced in the respective sections. The notation used in the papers on which the thesis is based may differ.

We assume that the reader is familiar with the basic notions of automata theory [1, 117] and complexity theory [93]. The cardinality of a set A is denoted by $|A|$ and the power set of A by 2^A. An alphabet is a finite nonempty set. The free monoid generated by an alphabet Σ is denoted by Σ^*. A word over Σ is any element of Σ^*; the empty word is denoted by ε. For a word $w \in \Sigma^*$, $|w|_a$ denotes the number of occurrences of letter a in w. If $w = xyz$, then x is a *prefix*, y a *factor*, and z a *suffix* of w. A prefix (factor, suffix) of w is *proper* if it is different from w. A language over Σ is a subset of Σ^*. For a language L over Σ, let $L^c = \Sigma^* \setminus L$ denote the complement of L.

A *nondeterministic finite automaton* (NFA) is a quintuple $\mathscr{A} = (Q, \Sigma, \delta, I, F)$, 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 $2^Q \times \Sigma^*$ by induction. The language *accepted* by $\mathscr{A}$ is the set $L(\mathscr{A}) = \{w \in \Sigma^* \mid \delta(I, w) \cap F \neq \emptyset\}$.

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 $q_0 a_1 q_1 a_2 \ldots q_{n-1} a_n q_n$ such that $q_{i+1} \in \delta(q_i, a_{i+1})$, for all $i = 0, 1, \ldots, n-1$. The path π is *accepting* if $q_0 \in I$ and $q_n \in F$. We use the notation $q_0 \xrightarrow{a_1 a_2 \cdots a_n} q_n$ 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 states of the path are pairwise distinct. The number of states on the longest simple path of $\mathscr{A}$, starting in an initial state, decreased by one (i. e., the number of transitions on that path) is called the *depth* of $\mathscr{A}$, denoted by $\mathit{depth}(\mathscr{A})$.

An automaton $\mathscr{A}$ is *complete* if every transition is defined in every state, that is, for every state q of $\mathscr{A}$ and every letter $a \in \Sigma$, the set $\delta(q, a)$ is nonempty.

An NFA $\mathscr{A}$ is *deterministic* (DFA) if $|I| = 1$ and $|\delta(q, a)| \leq 1$ for every q in Q and a in Σ. Note that we allow transitions to be undefined. To obtain a complete automaton, it is then necessary to add a sink state.

The reachability relation $\leq$ on the set of states is defined by $p \leq q$ if there exists a word $w \in \Sigma^*$ such that $q \in \delta(p, w)$. An NFA $\mathscr{A}$ is *partially ordered (poNFA)* if the reachability relation $\leq$ is a partial order. For two states p and q of $\mathscr{A}$, 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 $p < q$. Partially ordered automata are sometimes called acyclic in the literature, but the reader should keep in mind that in this case self-loops are allowed.

3. Descriptional Complexity

Descriptional complexity, also known as state complexity, studies the questions of the form:

> Given automata $\mathscr{A}_1, \mathscr{A}_2, \ldots, \mathscr{A}_n$ and an n-ary operation op on languages. What is the minimum number of states of the automaton (deterministic or nondeterministic) recognizing the language $op(L(\mathscr{A}_1), L(\mathscr{A}_2), \ldots, L(\mathscr{A}_n))$?

Indeed, the question makes sense if the considered class of languages is closed under the operation op, that is, if the automata $\mathscr{A}_i$ are of some type, then there is an automaton of the same type recognizing the language $op(\mathscr{A}_1, \ldots, \mathscr{A}_n)$. A typical and most studied example are deterministic and nondeterministic finite automata with the binary Boolean operations or with the unary operation reversal, which is the operation of our interest in this chapter.

3.1 Reversal

The reverse w^R of a word w is inductively defined by $\varepsilon^R = \varepsilon$ and, for a word v in Σ^* and a symbol a in Σ, $(va)^R = av^R$. The reverse of a language L is the language $L^R = \{w^R \mid w \in L\}$. The reverse of a DFA $\mathscr{A} = (Q, \Sigma, \delta, i, F)$ is the NFA $\mathscr{A}^R$ obtained from $\mathscr{A}$ by reversing all the transitions and by swapping the role of the initial and final states, that is, $\mathscr{A}^R = (Q, \Sigma, \delta^R, F, \{i\})$, where $\delta^R(q,a) = \{p \in Q \mid \delta(p,a) = q\}$.

Let $rev(\mathscr{A})$ denote the minimal DFA equivalent to $\mathscr{A}^R$. It is known that if $\mathscr{A}$ is a deterministic finite automaton with n states, then the size of $rev(\mathscr{A})$ can be in the worst case of size 2^n [89]. With this in mind, it could seem that the formula

$$rev(rev(\mathscr{A}))$$

results in a DFA with a double exponential 2^{2^n} state space, compared with the size of $\mathscr{A}$. It is not the case and, perhaps surprisingly, the result of these two exponential operations is the minimal DFA recognizing the language of $\mathscr{A}$. This is known as Brzozowski's minimization algorithm [18]. The principle of the double reverse further plays a role in the relationship between algebra and co-algebra [30].

3.1.1 Reversal of $\mathscr{R}$-trivial Languages

Motivated by Brzozowski's minimization algorithm, we asked the question about the state complexity of the reverse for partially ordered DFAs, recognizing $\mathscr{R}$-trivial languages, and showed that the size of the DFA for the reverse is at most 2^{n-1} and that the bound can be reached by DFAs over a ternary alphabet.

Theorem 1. *Let L be a language accepted by a minimal poDFA with n states. Then the minimal DFA accepting the reverse of the language L has at most 2^{n-1} states. The bound is met by a ternary poDFA with a sink state, or by a poDFA over a growing alphabet without the sink state.*

$n = sc(L)$	Worst-case $sc(L^R)$ where DFA for L is without sink state	with sink state	Upper bound $2^{n-2}+n-1$	Lower bound 2^{n-2}
1	1	1	-	-
2	2	2	2	1
3	4	4	4	2
4	7	7	7	4
5	12	12	12	8
6	21	21	21	16
7	**34**	34	**38**	32
8	55	**64**	71	**64**

Table 3.1: Tight bounds for the reverse of binary $\mathscr{R}$-trivial regular languages

If the alphabet of the DFA is binary, then the bound is at most $2^{n-2}+n-1$, and hence the ternary alphabet is optimal to obtain the worst-case state complexity. Moreover, for $n \geq 8$, the tight bound for the binary case is 2^{n-2}. We further provided a complete characterization of the tight bounds for $\mathscr{R}$-trivial regular languages depending on the state complexity of the language and the size of its alphabet. To formulate this result, we denote by $f_k(n)$ the state complexity function of the reverse on $\mathscr{R}$-trivial regular languages over a k-element alphabet, that is, we define

$$f_k(n) = \max\{sc(L^R) \mid L \subseteq \Sigma^*, |\Sigma| = k, L \text{ is } \mathscr{R}\text{-trivial regular, and } sc(L) = n\}$$

where $sc(L)$ denotes the state complexity of the language L, that is, the number of states of its minimal DFA.

Using this notation, we can summarize our results in the following theorem and Table 3.1.

Theorem 2. *Let $n \geq 1$ and let $f_k(n)$ be the state complexity of the reverse on $\mathscr{R}$-trivial regular languages over a k-element alphabet. Then*

$$f_1(n) = n,$$

$$f_2(n) = \begin{cases} 1, & \text{if } n = 1, \\ 2^{n-2}+n-1, & \text{if } 2 \leq n \leq 6, \\ 34, & \text{if } n = 7, \\ 2^{n-2}, & \text{otherwise,} \end{cases}$$

$$f_3(n) = f_k(n) = 2^{n-1}, \text{ for every } k \geq 3.$$

□

3.1.2 Reversal of $\mathscr{J}$-trivial Languages

Considering the case of $\mathscr{J}$-trivial languages recognized by confluent poDFAs, we have shown that to reach the bound 2^{n-1} requires at least $n-1$ letters.

Theorem 3. *At least $n-1$ letters are necessary for a confluent poDFA with n states to reach the state complexity 2^{n-1} for the reverse operation.*

In other words, we have shown that the upper bound on the state complexity of the reverse cannot be met by a $\mathscr{J}$-trivial language over an $(n-2)$-element alphabet. We further showed a tight bound for $\mathscr{J}$-trivial languages over $(n-2)$-element alphabets and several tight bounds for binary $\mathscr{J}$-trivial languages.

Theorem 4. *Let $n \geq 3$ and let L be a $\mathscr{J}$-trivial language over an $(n-2)$-element alphabet with $sc(L) = n$. Then $sc(L^R) \leq 2^{n-1} - 1$ and the bound is tight.*

Corollary 5. *Let L be a binary $\mathscr{J}$-trivial language with $sc(L) = n$, where $n \geq 4$, then $sc(L^R) \leq 2^{n-3} + \min(\max(2n-3, (n-2)^2), 2^{n-3}) + (n-1)$. A few tight bounds for $2 \leq n \leq 7$ are given in Table 3.1, since the bounds there are obtained for $\mathscr{R}$-trivial languages that are also $\mathscr{J}$-trivial.*

The case of $\mathscr{J}$-trivial languages over $(n-k)$-element alphabets, for $2 \leq k \leq n-3$, is left open. However, Campeanu et al. [25] showed that there are finite binary languages whose reverse is of state complexity $3 \cdot 2^{\frac{n}{2}-1} - 1$, if n even, and $2^{\frac{n+1}{2}} - 1$, if n odd. Since every finite language is $\mathscr{J}$-trivial, we obtain at least these lower bounds for binary $\mathscr{J}$-trivial languages.

The results of this section are mainly based on papers [60, 61] attached in Appendix **??**

4. Separability

The motivation to study separability comes from scenarios in which we want to describe something complex by means of a simple language. The *separation* problem of our interest is of the following form:

> Given two languages K and L and a family of languages $\mathscr{F}$. Is there a language S in $\mathscr{F}$ such that S contains everything from K and nothing from L?

The family $\mathscr{F}$ of simple languages could be, for example, languages definable in first-order logic, piecewise testable languages, or languages definable with a particular type of automata.

For our theoretical research, we were motivated by several problems coming from practice: (a) increasing the user-friendliness of XML Schema and (b) efficient approximate query answering.

XML Schema is currently the only industrially accepted and widely supported schema language for XML, designed to alleviate the limited expressiveness of Document Type Definition (DTD) [15], thereby making DTDs obsolete. However, XML Schema's extra expressiveness comes at the cost of simplicity. Its code is designed to be machine-readable rather than human-readable and its logical core, based on *complex types*, does not seem well-understood by users [77, 90]. One reason may be that the specification of XML Schema's core consists of over 100 pages of intricate text [44]. The BonXai schema language designed by Martens, Neven, Niewerth and Schwentick [77, 78] is an attempt to overcome these issues and to combine the simplicity of DTDs with the expressiveness of XML Schema. It has the same expressive power as XML Schema, is designed to be human-readable, and avoids the use of complex types, aiming at simplifying the development or analysis of XSDs. The BonXai schema is a set of rules $L_1 \to R_1, \ldots, L_n \to R_n$ in which all L_i and R_i are regular expressions. An unranked tree t (an XML document) is in the language of the schema if, for every node u, the word formed by the labels of u's children is in the language R_k, where k is the largest number such that the word of ancestors of u is in L_k. This semantical definition is designed to ensure full compatibility with XML Schema [77, 78]. When translating an XML Schema Definition (XSD) into an equivalent BonXai schema, the regular expressions L_i are obtained from a finite automaton that is embedded in the XSD. Since the current state-of-the-art in translating automata to expressions does not yet generate human-readable results, we investigated simpler classes of expressions that we expect to suffice in practice. Practical and theoretical studies show evidence that regular expressions of the form $\Sigma^* w$ (with $w \in \Sigma^+$) and $\Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n$ (with $a_1, \ldots, a_n \in \Sigma$) and variations thereof seem to be well-suited [47, 65, 80].

Our second motivation comes from efficient approximate query answering. An efficient evaluation of regular expressions is relevant in a wide array of fields, for instance in graph databases and in the context of the SPARQL language [5, 51, 76, 94] for querying RDF data. Typically, regular expressions are used to match paths between nodes in a huge graph. In fact, the data can be so huge that the exact evaluation of a regular expression r over the graph (which can lead to a product construction between an automaton for the expression and the graph [76, 94]) may not be feasible within reasonable time. Therefore, as a trade-off to exact evaluation, one

could imagine that we try to rewrite the regular expression r as an expression that we can evaluate much more efficiently and is close enough to r. Specifically, we could specify two expressions r_{pos} (resp., r_{neg}) that define the language we want to (resp., do not want to) match in our answer and ask whether there exists a simple query (e. g., defining a piecewise testable language) that satisfies these constraints. Notice that the scenario of approximating an expression r in this way is very general and not even limited to databases. (Also, we can take r_{neg} to be the complement of r_{pos}.)

At first sight, these two motivating scenarios may seem to be fundamentally different. In the first, we want to compute an *exact* simple description of a complex object and in the second we want to compute an *approximate* simple query that can be evaluated more efficiently. However, both scenarios boil down to the same underlying question of language separation.

The effort to approximate languages by simpler languages is further related to system verification. The goal here is to express a given bad property in a simple logic for which verification is feasible or even efficient. If the property is not expressible in a simple logic, it could perhaps be over-approximated in a simple logic. If the system is then safe with respect to the over-approximated property, i. e. it does not satisfy the over-approximated bad property, then it is also safe with respect to the original property.

Another situation is to require that every fair run of the system satisfies some additional conditions. Expressing fairness by a formula φ_1 and the additional conditions by a formula φ_2, we need to verify that the system satisfies the formula $\varphi_1 \Rightarrow \varphi_2$. Now we would like to interpolate φ_1 by a formula of a simpler form, say ψ, such that $\varphi_1 \Rightarrow \psi \Rightarrow \varphi_2$. Moreover, if we consider $\neg\varphi_2$ rather than φ_2, then the set of models satisfying ψ separates the sets of models of φ_1 and $\neg\varphi_2$, respectively. Hence, this problem again boils down to the separation problem. The reader familiar with program verification might have noticed a close relationship to interpolation, a method providing means to compute separation between good and bad states [31, 105].

Separability is further of interest in logic on words. Place and Zeitoun [100] used separability to obtain new decidability results for the membership problem for some levels of the Straubing-Thérien hierarchy [122, 124]. However, the problem remains open for almost all levels of the hierarchy [2, 97, 99].

Separability by piecewise testable languages has also been investigated outside the family of regular languages. Although separability of context-free languages by regular languages is undecidable [58], separability by piecewise testable languages is decidable [35]. The problem was further generalized to regular tree languages by Goubault-Larrecq and Schmitz [50], and studied for other types of languages as well, see Czerwiński and Lasota [33] for an overview of the latest development.

4.1 Separability of Languages

A language S *separates language* K *from* L if S contains K and does not intersect L. We say that S *separates* K *and* L if it either separates K from L or L from K. Let $\mathscr{F}$ be a family of languages. Languages K and L are *separable by* $\mathscr{F}$ if there exists a language S in $\mathscr{F}$ that separates K and L.

Languages K and L are *layer-separable by* $\mathscr{F}$ if there exists a finite sequence of languages $S_1, \ldots, S_m$ in $\mathscr{F}$ such that

1. for all $1 \leq i \leq m$, language $S_i \setminus \bigcup_{j=1}^{i-1} S_j$ intersects at most one of K and L, and
2. K or L (possibly both) is included in $\bigcup_{j=1}^{m} S_j$.

Separability implies layer-separability, but the opposite implication does not hold.

Our motivation for layered separability comes from the BonXai schema language. We need to solve layer-separability if we want to decide whether an XML Schema has an equivalent BonXai

schema with simple regular expressions (defining languages in $\mathscr{F}$). Layered separability implies that languages are, in a sense, separable by languages from $\mathscr{F}$ in a priority-based system—if we consider the ordered sequence of languages $S_1, S_2, \ldots, S_m$, then, in order to classify a word w from $K \cup L$ as either in K or in L, we have to match it against the S_i in the increasing order of the index i. Then, as soon as we find the lowest index j for which $w \in S_j$, we know whether $w \in K$ or $w \in L$.

A quasi-order is a reflexive and transitive relation. A quasi-order $\preccurlyeq$ on a set X is a *well-quasi-ordering* (WQO) if for every infinite sequence $(x_i)_{i=1}^{\infty}$ of elements of X, there exist indices $i < j$ such that $x_i \preccurlyeq x_j$. It is known that every WQO is *well-founded*, that is, there exist no infinite descending sequences $x_1 \succcurlyeq x_2 \succcurlyeq \cdots$ such that $x_i \not\preccurlyeq x_{i+1}$ for all i. For a quasi-order $\preccurlyeq$, the *(upward) $\preccurlyeq$-closure* of a language L is the set $\mathrm{up}^{\preccurlyeq}(L) = \{w \mid v \preccurlyeq w \text{ for some } v \in L\}$. Language L is *(upward) $\preccurlyeq$-closed* if $L = \mathrm{up}^{\preccurlyeq}(L)$.

We now extend the notion of *alternating towers* of Stern [119] and use it to determine when two languages are *not* separable. For languages K and L and a quasi-order $\preccurlyeq$, we say that a sequence $(w_i)_{i=1}^{k}$ of words is a *$\preccurlyeq$-tower between K and L* if $w_1 \in K \cup L$ and, for all $i = 1, \ldots, k-1$:

$$(1)\ w_i \preccurlyeq w_{i+1};\ (2)\ w_i \in K \text{ implies } w_{i+1} \in L;\ \text{and } (3)\ w_i \in L \text{ implies } w_{i+1} \in K.$$

We say that k is the *height* of the $\preccurlyeq$-tower. We similarly define an infinite sequence of words to be an *infinite $\preccurlyeq$-tower between K and L*.

For $v = a_1 \cdots a_n$ and $w \in \Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$, we say that v is a *subsequence* of w, and use the notation $v \preceq w$ to denote the subsequence order. If we consider the subsequence order $\preceq$, then we simply talk about a *tower* rather than about a $\preceq$-tower.

We now characterize separability of two languages by a family of $\preccurlyeq$-closed languages for a WQO relation $\preccurlyeq$.

Theorem 6. *For languages K and L and a WQO $\preccurlyeq$ on words, the following are equivalent.*

1. *K and L are separable by a boolean combination of $\preccurlyeq$-closed languages.*
2. *K and L are layer-separable by $\preccurlyeq$-closed languages.*
3. *There does not exist an infinite $\preccurlyeq$-tower between K and L.*

Some of the equivalences hold even if the assumptions are weakened. For example, the equivalence between (1) and (2) does not require $\preccurlyeq$ to be a WQO.

Since the subsequence order $\preceq$ is a WQO on words, the languages K and L are separable by piecewise testable languages if and only if they are layer-separable by $\preceq$-closed languages. Actually, since $\preceq$ as a WQO has only finitely many minimal elements within a language, the latter is equivalent to being layer-separable by shuffle ideals, that is, by languages of the form $\Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$.

We have shown how to decide the existence of an infinite tower between two regular languages, given as regular expressions or NFAs, in polynomial time, which is by Theorem 6 equivalent to deciding whether the two languages can be separated by a piecewise testable language.

Theorem 7. *Given two NFAs $\mathscr{A}$ and $\mathscr{B}$, it is possible to test in polynomial time whether $L(\mathscr{A})$ and $L(\mathscr{B})$ can be separated by a piecewise testable language.*

We have shown that if there is an infinite tower between two regular languages, then there is an infinite tower of a special form in which every word can be decomposed in some synchronized manner. We can find these special forms of towers in polynomial time in the NFAs. The main

$\mathscr{F}(O,C)$	single	unions	bc (boolean combinations)
$\preceq$ (subsequence)	NP-complete	PTIME	PTIME-complete
$\preceq_s$ (suffix)	PTIME	PTIME	PTIME

Table 4.1: The complexity of deciding separability for regular languages K and L.

features are that our algorithm runs exponentially faster in the alphabet size than the previous state-of-the-art [4] and that our algorithm and its proof of correctness do not require knowledge of the algebraic perspective on regular languages.

To conclude this part, we have further shown that the considered problem is PTIME-hard. Consequently, the problem cannot be efficiently parallelized [6].

Theorem 8. *Deciding separability of regular languages, represented as NFAs, by piecewise testable languages is* PTIME*-complete. It remains* PTIME*-hard even for minimal DFAs.*

4.2 Asymmetric Separation and Suffix Order

In this section, we present a bigger picture on efficient separations relevant to our motivation scenarios. Namely, we consider what happens when we restrict the allowed boolean combinations of languages. This means that separation is no longer symmetric. We also consider the suffix order $\preceq_s$ between words in which $v \preceq_s w$ if and only if v is a (not necessarily strict) suffix of w. An important technical difference is that the suffix order is not a WQO. Indeed, the suffix order $\preceq_s$ has an infinite antichain, e. g., $a, ab, abb, abbb, \ldots$. The results we present here for suffix order hold true for prefix order as well.

Let $\mathscr{F}$ be a family of languages. Language K is *separable from a language L by $\mathscr{F}$* if there exists a language S in $\mathscr{F}$ that separates K from L, i. e., S contains K and does not intersect L. Thus, if $\mathscr{F}$ is closed under complement, then K is separable from L implies L is separable from K. The *separation problem by $\mathscr{F}$* asks, given an NFA for K and an NFA for L, whether K is separable from L by $\mathscr{F}$.

We consider separation by families of languages $\mathscr{F}(O,C)$, where O ("order") specifies the ordering relation and C ("combinations") specifies how we are allowed to combine (upward) O-closed languages. Specifically, O is either the subsequence order $\preceq$ or the suffix order $\preceq_s$. We allow C to be one of *single*, *unions*, or *bc* (boolean combinations), meaning that each language in $\mathscr{F}(O,C)$ is either the O-closure of a single word, a finite union of the O-closures of single words, or a finite boolean combination of the O-closures of single words. Thus, $\mathscr{F}(\preceq, bc)$ is the family of piecewise testable languages and $\mathscr{F}(\preceq_s, bc)$ is the family of suffix-testable languages. With this convention in mind, the main result of this section is to provide a complete complexity overview of the six possible cases of separation by $\mathscr{F}(O,C)$.

Theorem 9. *For $O \in \{\preceq, \preceq_s\}$ and C being one of single, unions, or boolean combinations, we have that the complexity of the separation problem by $\mathscr{F}(O,C)$ is as indicated in Table 4.1.*

Since the separation problem for prefix order is basically the same as the separation for suffix order and has the same complexity, we did not list it separately in the table. Some other types of languages, such as union-free languages, are further discussed in the original paper [34]. Separability by k-piecewise testable languages and the variants thereof has been studied by Hofman et al. [53].

4.3 Computation of Piecewise Testable Separators

We now focuse on the construction of a piecewise testable separator of two regular languages, if such a separator exists. It was independently shown by Czerwiński et al. [34] and Place et al. [98] that the *non*-separability by piecewise testable languages is equivalent to the existence of a common pattern in the two automata (called an $(\mathbf{u},\mathbf{B})$-*path* in Place et al. [98] and *synchronized languages* in Czerwiński et al. [34]). This pattern is further equivalent to the existence of an infinite tower between the languages [34], and its existence can be detected in polynomial time. A similar pattern has recently been identified for general word languages [36].

To construct an actual separator is a more difficult task. Place et al. [98] construct the separator as a union of $\sim_\kappa$-equivalence classes, where $u \sim_\kappa v$ if and only if the words u and v have the same set of subsequences of length up to κ. The difficult part is to find a suitable κ. Place et al. [98] find such a κ that is exponential in the size of the automaton and doubly exponential in the size of the alphabet. The separator is then κ-piecewise testable and it can be constructed as the union of $\sim_\kappa$ classes that cover one of the languages.

Our approach is different. For a word $w = a_1 a_2 \cdots a_\ell$, where $a_i \in \Sigma$, let $\mathrm{up}(w)$ denote the language $\Sigma^* a_1 \Sigma^* a_2 \Sigma^* \cdots \Sigma^* a_\ell \Sigma^*$ of all supersequences of w (the *upward closure*). For a language L, let $\mathrm{up}(L) = \bigcup_{w \in L} \mathrm{up}(w)$. Then $\mathrm{up}(L) = \bigcup_{w \in M_L} \mathrm{up}(w)$, where M_L is the set of all minimal elements of L with respect to the subsequence relation $\preceq$, which is finite by Higman's Lemma [52]. By construction, $\mathrm{up}(L)$ is piecewise testable for any language L.

Let K and L be two disjoint languages over Σ. To construct a piecewise testable language $S \supseteq K$ disjoint from L – a piecewise testable separator – we choose $\mathrm{up}(K)$ as the first approximation of S. Typically, $\mathrm{up}(K)$ is not disjoint from L, and hence we try to fix it by putting $L_1 = \mathrm{up}(K) \cap L$ and by taking $S_0 = \mathrm{up}(K) \setminus \mathrm{up}(L_1)$. Although S_0 is obviously disjoint from L, it may not be a superset of K, namely if $K_1 = K \cap \mathrm{up}(L_1)$ is nonempty. We therefore repeat the construction for K_1, and construct another "layer" of S defining $L_2 = \mathrm{up}(K_1) \cap L$ and $S_1 = \mathrm{up}(K_1) \setminus \mathrm{up}(L_2)$. In this way, we obtain a sequence $S_0, S_1, S_2, \ldots$ of piecewise testable sets defined by $K_0 = \mathrm{up}(K)$ and by

$$\begin{aligned} L_{i+1} &= \mathrm{up}(K_i) \cap L, \\ K_{i+1} &= \mathrm{up}(L_{i+1}) \cap K, \\ S_i &= \mathrm{up}(K_i) \setminus \mathrm{up}(L_{i+1}). \end{aligned}$$

Finally, we define $S = \bigcup_{i \geq 0} S_i$. Definitions imply that $w \in L_{i+1}$ if and only if there is a tower $w_1 \preceq w'_1 \preceq w_2 \preceq w'_2 \preceq \cdots \preceq w_i \preceq w'_i = w$ between K and L. Therefore, if the maximum height of a tower between K and L is $r \leq 2j-1$, then L_{j+1} is empty. Then $S_j = \mathrm{up}(K_j)$ and $S = \bigcup_{i=0}^{j} S_i$ is the piecewise testable separator we are looking for. Notice that the complexity of the above construction depends on the maximal height of the tower between K and L, which motivates our study on the upper and lower bounds on the height of finite towers discussed below.

The relationship between the maximal height of towers and the number κ of Place et al. is another interesting question. The number of classes of the equivalence relation $\sim_\kappa$ indeed depends on κ and was investigated by Karandikar et al. [63]. We showed that, in some sense, κ provides an upper bound on the maximal height of towers, and that κ can be arbitrarily larger than the maximal height of towers [55].

The complexity of a separator S can also be measured by the number of elementary languages of the form $\mathrm{up}(w)$ needed in the boolean expression defining S. Let F be the set of words such that S is a boolean combination of languages $\mathrm{up}(w)$, where $w \in F$. For each word $u \in \Sigma^*$, the truth value of $u \in K$ is determined by the set $\sigma(u) = \{w \in F \mid u \in \mathrm{up}(w)\}$. In particular, $\sigma(u) = \sigma(v)$ implies that $u \in S$ if and only if $v \in S$. Observe that $u \preceq u'$ implies that $\sigma(u) \subseteq \sigma(u')$. We now deduce that $\sigma(w_1) \subsetneq \sigma(w_2) \subsetneq \cdots \subsetneq \sigma(w_r) \subseteq F$ for any tower $(w_i)_{i=1}^r$ between two languages K

and L, and hence $|F| \geq r-1$. This means that any such a boolean expression requires at least as many elements as is the height of the maximal tower.

4.3.1 Bounds on the Height of Towers

Not much is known about the upper bound on the height of towers between two regular languages if no infinite tower exists. The only result we are aware of is a result by Stern [119] giving an exponential upper bound $2^{|\Sigma|^2 n}$ on the height of towers between a piecewise testable language over an alphabet Σ represented by an n-state minimal DFA and its complement. We present a better bound that holds in a general setting of two arbitrary regular languages (having no infinite tower) represented by NFAs.

Theorem 10. *Let $\mathscr{A}_0$ and $\mathscr{A}_1$ be NFAs with n and m states, respectively, over an alphabet Σ. Assume that there is no infinite tower between the languages $L(\mathscr{A}_0)$ and $L(\mathscr{A}_1)$, and let $(w_i)_{i=1}^r$ be a tower between the languages such that $w_i \in L(\mathscr{A}_{i \bmod 2})$. Let $1 < \mu \leq \max(n,m)$ denote the maximum of the depths of $\mathscr{A}_0$ and $\mathscr{A}_1$. Then $r \leq \frac{\mu^{|\Sigma|+1}-1}{\mu-1}$.*

Thus, the upper bound on the height of towers between two regular languages represented by NFAs is polynomial with respect to the depth of the NFAs and exponential with respect to the size of the alphabet.

The question now is how good this bound is. We study this question next and show that it is tight if the alphabet is fixed. If the alphabet grows with the number of states of the automata, then we can construct a tower of exponential height with respect to the number of states of the automata (as well as with respect to the size of the alphabet). However, we do not know whether this bound is tight. We formulate this question as the following open problem asking how much the size of the alphabet can increase the height of the tower, given the number of states (or the depth).

Open Problem 11. Let $\mathscr{A}_0$ and $\mathscr{A}_1$ be NFAs with n and m states, respectively, over an alphabet Σ with $|\Sigma| \geq n+m$. Let μ be the maximum depth of $\mathscr{A}_0$ and $\mathscr{A}_1$. Assume that there is no infinite tower between the languages $L(\mathscr{A}_0)$ and $L(\mathscr{A}_1)$, and let $(w_i)_{i=1}^r$ be a tower between them. Is it true that $r \leq \frac{\mu^{n+m+1}-1}{\mu-1}$ or even that $r \leq 2^{n+m}$?

The upper bound result indicates that the size of the alphabet is significant for the height of towers. This is confirmed by lower bounds considered now. We consider two cases, namely (i) the size of the alphabet is fixed and (ii) the size of the alphabet may grow with the size of the automata. We show that the upper bound is asymptotically tight if the size of the alphabet is fixed, and that the lower bound may be exponential with respect to the size of the automata if the alphabet may grow. In this case, the size of the alphabet is approximately the number of states of the automata.

Theorem 12. *For all integers $n,m \geq 2$ there exist two NFAs with n and m states over an alphabet of cardinality $n+m-2$ having a tower of height $2^{n+m-2} - 2^{m-1} + 2$ and no infinite tower.*

We can adapt the theorem to deterministic automata as follows.

Theorem 13. *For all integers $k \geq 1$, $d \geq 2$ and every odd positive integer e, there exist two DFAs with $(k+1)d+k-1$ and $e+1$ states over an alphabet of cardinality $k+1$ having a tower of height $(e+1)d^k + 2d^{k-1}$ and no infinite tower.*

Consequently, we have that the upper bound is tight for a fixed alphabet even for DFAs.

Corollary 14. *Let $k \geq 2$ be a constant. Then the maximum height of a tower between two DFAs with at most n states over an alphabet of cardinality k having no infinite tower is in $\Omega\left(n^k\right)$.*

If the alphabet is allowed to grow with the number of states, we have shown that the height of a tower can be exponential in the number of states of NFAs. For DFAs with at most n states, we obtain that the height of a tower is $\Omega\left((n+1)2^{\frac{n}{3}}\right)$. To obtain a better lower bound for DFAs, we introduced a "determinization" strategy [55].

Theorem 15. *For every $n \geq 0$, there exist two DFAs with at most $n+1$ states over an alphabet of cardinality $\frac{n(n+1)}{2}+1$ having a tower of height 2^n and no infinite tower.*

The "determinization" strategy further allows us to prove the following results.

Theorem 16. *For every two NFAs $\mathscr{A}$ and $\mathscr{B}$ with at most n states and k input letters, there exist two DFAs $\mathscr{A}'$ and $\mathscr{B}'$ with $O\left(n^2\right)$ states and $O\left(k+n\right)$ input letters such that there is a tower of height r between $\mathscr{A}$ and $\mathscr{B}$ if and only if there is a tower of height r between $\mathscr{A}'$ and $\mathscr{B}'$. In particular, there is an infinite tower between $\mathscr{A}$ and $\mathscr{B}$ if and only if there is an infinite tower between $\mathscr{A}'$ and $\mathscr{B}'$.*

Theorem 17. *For every two NFAs $\mathscr{A}$ and $\mathscr{B}$ with at most n states and k input letters, there exist two DFAs $\mathscr{A}'$ and $\mathscr{B}'$ with $O(kn)$ states and $O(kn)$ input letters such that there is a tower of height r between $\mathscr{A}$ and $\mathscr{B}$ if and only if there is a tower of height r between $\mathscr{A}'$ and $\mathscr{B}'$. In particular, there is an infinite tower between $\mathscr{A}$ and $\mathscr{B}$ if and only if there is an infinite tower between $\mathscr{A}'$ and $\mathscr{B}'$.*

The lower bounds are not asymptotically equal to the upper bound and it is not known what the (asymptotically) tight upper bound actually is. Specifically, we do not know whether an alphabet of size greater than the number of states may help to build higher towers.

Interestingly, the towers used in the constructions to demonstrate lower bounds are mostly sequences of prefixes. Therefore, we also investigated *towers of prefixes*. We provided a pattern that characterizes the existence of an infinite tower of prefixes and proved tight bounds on the height of towers of prefixes for DFAs and NFAs. This study can be found in our original paper [55].

Our main results are summarized in Table 4.2.

The main papers [34, 83, 54, 55] on which the results of this chapter are based are attached in Appendix **??**.

	Upper bound	Lower bound	
		$\lvert\Sigma\rvert = k$	$\lvert\Sigma\rvert \geq n+m$
NFAs DFAs	$\frac{\mu^{\lvert\Sigma\rvert+1}-1}{\mu-1}$	$\Theta\left(\mu^k\right)$	$\Omega\left(2^{n+m}\right)$

(a) Towers of subsequences over Σ; $\mu = \max(n,m)$

	Upper bound	Lower bound	
		$\lvert\Sigma\rvert = 2$	$\lvert\Sigma\rvert \geq n+m$
NFAs	$\frac{(2^n-1)(2^m-1)+1}{2}$	$\Omega\left(2^{\sqrt{\frac{2\nu}{\log 2\nu}}}\right)$	$2^{n+m-2-o(1)}$
DFAs	$\frac{nm}{2}+1$	$\frac{nm}{2}+1$	$\frac{nm}{2}+1$

(b) Towers of prefixes; $\nu = \min(n,m)$

Table 4.2: Upper and lower bounds on the height of towers of subsequences and prefixes for automata with n and m states

5. Complexity of Universality

Universality is a fundamental question asking whether a given system recognizes all words over its alphabet. Deciding universality is typically more difficult than deciding the word problem. The study of universality (and its dual – emptiness) has a long tradition in formal languages with many applications across computer science, e. g., in knowledge representation and database theory [9, 24, 118] or in verification [8]. Recent studies investigate the problem for specific types of automata or grammars, e. g., for prefixes or factors of regular languages [102].

Deciding universality for systems modeled by NFAs is PSPACE-complete [88], and there are two typical proof techniques to show hardness. One is based on the reduction from the *DFA-union-universality* problem [70], and the other on the reduction from the *word problem* for polynomially-space-bounded Turing machines [1]. Kozen's [70] 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; if all cycles of the DFAs were only self-loops, then the problem would be easier:

Theorem 18. *The intersection-emptiness problem for poDFAs/poNFAs is* coNP*-complete. It is* coNP*-hard even if the alphabet is binary.*

We show that deciding universality for poNFAs has the same worst-case complexity as for general NFAs, even if restricted to binary alphabets [72]. This is caused by an unbounded number of nondeterministic steps admitted in poNFAs – they either stay in the same state or move to another. Forbidding this kind of nondeterminism affects the complexity of deciding universality – it is coNP-complete if the alphabet is fixed but remains PSPACE-complete if the alphabet may grow polynomially [72]. The growth of the alphabet thus compensates for the restricted number of nondeterministic steps. Adding further a structural assumption of confluence on top of these models preserves the complexity. Our results are summarized in Table 5.1 and further discussed below with more details.

As already pointed out, we are interested in the universality problem for *partially ordered NFAs* (poNFAs) and special cases thereof. An NFA is partially ordered if its transition relation

	ST	$\lvert\Sigma\rvert = 1$	$\lvert\Sigma\rvert = k \geq 2$	Σ is growing
DFA		L-comp. [62]	NL-comp. [62]	NL-comp. [62]
spoNFA	$\frac{1}{2}$	AC^0	AC^0	AC^0
ptNFA	1	NL-comp.	coNP-comp.	PSPACE-comp.
rpoNFA		NL-comp.	coNP-comp.	PSPACE-comp.
poNFA	$\frac{3}{2}$	NL-comp.	PSPACE-comp.	PSPACE-comp. [1]
NFA		coNP-comp. [121]	PSPACE-comp. [1]	PSPACE-comp. [1]

Table 5.1: Complexity of deciding universality for poNFAs and special classes thereof; ST stands for the corresponding level of the Straubing-Thérien hierarchy; Σ denotes the input alphabet

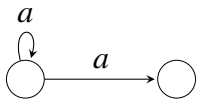

Figure 5.1: Forbidden pattern of rpoNFAs

induces a partial order on states: the only cycles allowed are self-loops on a single state. Partially ordered NFAs define a natural class of languages that has been shown to coincide with level $\frac{3}{2}$ of the Straubing-Thérien hierarchy [111] and with Alphabetical Pattern Constraint (APC) languages, a subclass of regular languages effectively closed under permutation rewriting [14]. Deciding whether an automaton recognizes an APC language (and hence whether it can be recognized by a poNFA) is PSPACE-complete for NFAs and NL-complete for DFAs [14].

Restricting to partially ordered deterministic finite automata (poDFAs), we can capture further classes of interest: two-way poDFAs characterize languages whose syntactic monoid belongs to the variety **DA** [111], introduced by Schützenberger [110]; poDFAs characterize $\mathscr{R}$-trivial languages [19]; and confluent poDFAs characterize level 1 of the Straubing-Thérien hierarchy, also known as $\mathscr{J}$-trivial languages or piecewise testable languages [115]. Other relevant classes of partially ordered automata include partially ordered Büchi automata [73] and two-way poDFAs with look-around [75].

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

Corollary 19 (Lemma 10.2 [1])**.** *The universality problem for poNFAs is* PSPACE*-complete.*

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

We settle this open problem by showing that PSPACE-hardness is retained even for binary alphabets. This negative result leads us to ask if there is a natural subclass of poNFAs for which universality does become simpler. We consider *restricted* poNFAs (rpoNFAs), which require self-loops to be deterministic in the sense that the automaton contains no transition as in Figure 5.1, which we call *nondeterministic self-loops*. Large parts of the former hardness proof hinge on transitions of this form, which, speaking intuitively, allow the automaton to navigate to an arbitrary position in the input (using the loop) and, thereafter, continue checking an arbitrary pattern. Indeed, we find that the universality becomes CONP-complete for rpoNFAs with a fixed alphabet.

However, this reduction of complexity is not preserved for unrestricted alphabets. We have used a novel construction of rpoNFAs that characterize certain exponentially long words to show that universality is PSPACE-complete even for rpoNFAs if the alphabet may grow polynomially.

As a by-product, we have shown that rpoNFAs provide another characterization of $\mathscr{R}$-trivial languages introduced and studied by Brzozowski and Fich [19], and we have established the complexity of deciding $\mathscr{R}$-triviality and k-$\mathscr{R}$-triviality for rpoNFAs.

From the practical point of view, the problems of inclusion and equivalence of two languages, which are closely related to universality, are of interest, e. g., in optimization. Indeed, universality can be expressed either as the inclusion $\Sigma^* \subseteq L$ or as the equivalence $\Sigma^* = L$. Although equivalence can be seen as two inclusions, the complexity of inclusion does not play the role of a lower bound. For instance, for two deterministic context-free languages, inclusion is undecidable [42], whereas equivalence is decidable [112]. However, the complexity of universality gives a lower

bound on the complexity of both inclusion and equivalence, and we have shown that, for the studied partially ordered NFAs, the complexities of inclusion and equivalence coincide with the complexity of universality.

5.1 Partially Ordered NFAs

The languages recognized by poNFAs are exactly the languages on level $\frac{3}{2}$ of the Straubing-Thérien hierarchy [111]. Since the hierarchy is proper, this means that poNFAs can only recognize a strict subset of star-free regular languages. In spite of this rather low expressive power, the universality problem of poNFAs has the same worst-case complexity as for general NFAs, even when restricted to a fixed alphabet with only a few letters.

Theorem 20. *For every alphabet Σ with $|\Sigma| \geq 2$, the universality problem for poNFAs over Σ is* PSPACE*-complete.*

Ellul et al. [41, Section 5] give an example of a regular expression over a 5-letter alphabet such that the shortest non-accepted word is of exponential length, and which can also be encoded as a poNFA. Our previous proof shows such an example for an alphabet of two letters, if we use a Turing machine that runs for exponentially many steps before accepting. Note, however, that this property alone would not imply Theorem 20.

Reducing the size of the alphabet to one leads to a reduction in complexity. This is expected, since the universality problem for NFAs over a unary alphabet is merely CONP-complete [121]. For poNFAs, the situation is even simpler:

Theorem 21. *The universality problem for poNFAs over a unary alphabet is* NL*-complete. It can be checked in linear time.*

5.2 Restricted Partially Ordered NFAs

Restricted poNFAs are distinguished by deterministic self-loops. We relate them to the known class of $\mathscr{R}$-trivial languages, and we establish complexity results for deciding whether a language falls into this class.

Definition 22. A *restricted partially ordered NFA (rpoNFA)* is a poNFA such that, for every state q and symbol a, if $q \in \delta(q,a)$ then $\delta(q,a) = \{q\}$.

We show that rpoNFAs characterize $\mathscr{R}$-trivial languages, a subclass of regular languages defined by Brzozowski and Fich [19]. To introduce this class of languages, we require some auxiliary definitions. A word $v = a_1 a_2 \cdots a_n$ is a *subsequence* of a word w, denoted by $v \preccurlyeq w$, if $w \in \Sigma^* a_1 \Sigma^* a_2 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$. For $k \geq 0$, we write $\mathrm{sub}_k(v) = \{u \in \Sigma^* \mid u \preccurlyeq v, |u| \leq k\}$ for the set of all subsequences of v of length up to k. Two words w_1, w_2 are *$\sim_k$-equivalent*, written $w_1 \sim_k w_2$, if $\mathrm{sub}_k(w_1) = \mathrm{sub}_k(w_2)$. Then $\sim_k$ is a congruence (for concatenation) of finite index (i. e., with finitely many equivalence classes) [115]. $\mathscr{R}$-trivial languages are defined by defining a related congruence $\sim_k^{\mathscr{R}}$ that considers subsequences of prefixes:

Definition 23. Let $x, y \in \Sigma^*$ and $k \geq 0$. Then $x \sim_k^{\mathscr{R}} y$ if and only if

- for each prefix u of x, there exists a prefix v of y such that $u \sim_k v$, and
- for each prefix v of y, there exists a prefix u of x such that $u \sim_k v$.

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

It is known that $x \sim_k^{\mathscr{R}} y$ implies $x \sim_k y$ and (if $k \geq 1$) $x \sim_{k-1}^{\mathscr{R}} y$ [19]. Therefore, every *k-$\mathscr{R}$*-trivial language is also $(k+1)$-$\mathscr{R}$-trivial. Moreover, it has been shown that a language L is $\mathscr{R}$-trivial if and only if the minimal DFA recognizing L is partially ordered [19]. We can lift this result to characterize the expressive power of rpoNFAs.

Theorem 24. *A regular language is $\mathscr{R}$-trivial if and only if it is accepted by an rpoNFA.*

This characterization in terms of automata with forbidden patterns can be compared to results of Glaßer and Schmitz, who use DFAs with a forbidden pattern to obtain a characterization of level $\frac{3}{2}$ of the dot-depth hierarchy [49, 108].

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

Theorem 25. *The language recognized by a complete rpoNFA $\mathscr{A}$ is* $\mathrm{depth}(\mathscr{A})$*-$\mathscr{R}$-trivial.*

Similar relationships have been studied for $\mathscr{J}$-trivial languages [68, 81], but we are not aware of any such investigation for $\mathscr{R}$-trivial languages.

We may ask how difficult it is to decide whether a given NFA $\mathscr{A}$ accepts a language that is $\mathscr{R}$-trivial or *k-$\mathscr{R}$*-trivial for a specific $k \geq 0$. For most levels of the Straubing-Thérien hierarchy, it is not even known if this problem is decidable, and when it is, exact complexity bounds are often missing [99]. The main exception are $\mathscr{J}$-trivial languages—level 1 of the hierarchy.

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

Theorem 26. *Given an NFA $\mathscr{A}$, it is* PSPACE*-complete to decide whether the language accepted by $\mathscr{A}$ is $\mathscr{R}$-trivial.*

Theorem 27. *Given an NFA $\mathscr{A}$ and $k \geq 0$, it is* PSPACE*-complete to decide whether the language accepted by $\mathscr{A}$ is k-$\mathscr{R}$-trivial.*

In both previous theorems, hardness is shown by reduction from the universality problem for NFAs [1, 88]. Hence it holds even for binary alphabets. For a unary alphabet, we can obtain the following result.

Theorem 28. *Given an NFA $\mathscr{A}$ over a unary alphabet, the problems of deciding whether the language accepted by $\mathscr{A}$ is $\mathscr{R}$-trivial, or k-$\mathscr{R}$-trivial for a given $k \geq 0$, are both* CONP*-complete.*

We now briefly discuss the complexity of the problem if the language is given as a poNFA rather than an NFA.

Theorem 29. *Given a poNFA $\mathscr{A}$, the problems of deciding whether the language accepted by $\mathscr{A}$ is $\mathscr{R}$-trivial, or k-$\mathscr{R}$-trivial for a given $k \geq 0$, are both* PSPACE*-complete.*

We point out that the previous result holds even if the input alphabet of $\mathscr{A}$ is binary. For unary alphabets, the classes of languages of unary poNFAs and of unary $\mathscr{R}$-trivial languages coincide.

Theorem 30. *The classes of unary poNFA languages and unary $\mathscr{R}$-trivial languages coincide.*

5.2.1 Deciding Universality of rpoNFAs

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

Lemma 31. *The universality problem of rpoNFAs is* CONP*-hard even when restricting to alphabets with two letters.*

For a matching upper bound, if the size $|\Sigma|$ of the alphabet is bounded, then non-universality is witnessed by a word of polynomial length. Together with Lemma 31, this allows us to establish the following result and its immediate corollary.

Theorem 32. *Let Σ be a fixed non-unary alphabet, and let $\mathscr{B}$ be an rpoNFA over Σ. If $\mathscr{A}$ is an NFA (poNFA, rpoNFA, DFA, poDFA) over Σ, then the problem whether $L(\mathscr{A}) \subseteq L(\mathscr{B})$ is* CONP*-complete.*

Corollary 33. *Let Σ be a fixed non-unary alphabet. Then the universality problem for rpoNFAs over Σ is* CONP*-complete.*

Since the proof of Theorem 21 also applies to rpoNFAs, we immediately have the following result.

Corollary 34. *The universality problem for rpoNFAs over a unary alphabet is* NL*-complete.* □

Without fixing the alphabet, universality remains PSPACE-hard even for rpoNFAs, but a proof along the lines of Theorem 20 is not straightforward. In essence, rpoNFAs lose the ability to navigate to an arbitrary position within a word for checking some pattern there. Expressions of the form $(\Sigma^* \cdots)$, which we frequently used there (see [72, Theorem 3] for details), are therefore excluded. This is problematic since the run of a polynomially space-bounded Turing machine may be of exponential length, and we need to match patterns across the full length of our (equally exponential) encoding of this run. How can we navigate such a long word without using Σ^*? Our answer is to first define an rpoNFA that accepts all words except for a single, exponentially long word. This word will then be used as an rpoNFA-supported "substrate" for our Turing machine encoding.

Lemma 35. *For all positive integers k and n, there exists an rpoNFA $\mathscr{A}_{k,n}$ over an n-letter alphabet with $n(k+2)$ states such that the unique word not accepted by $\mathscr{A}_{k,n}$ is of length $\binom{k+n}{k} - 1$.*

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

Corollary 36. *For every integer $n \geq 1$, there is an rpoNFA $\mathscr{A}_n$ over an n-letter alphabet with $n(n+2)$ states such that the shortest word not accepted by $\mathscr{A}_n$ is of length $\binom{2n}{n} - 1$. Therefore, any minimal DFA accepting the same language has at least $\binom{2n}{n}$ states.*

To simulate exponentially long runs of a Turing machine, we start from an encoding of runs using words $\#w_1\# \cdots \#w_m\#$ (see the details in the original paper [72] in the appendix) and we combine every letter of this encoding with one letter of the alphabet of $\mathscr{A}_n$. We then accept all words for which the projection to the alphabet of $\mathscr{A}_n$ is accepted by $\mathscr{A}_n$, i. e., all but those words of exponential length that are based on the unique word not accepted by $\mathscr{A}_n$. We ensure that, if there is an accepting run, it will have an encoding of this length. It remains to eliminate (accept)

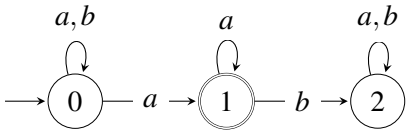

Figure 5.2: A confluent automaton accepting a non-piecewise testable language

all words that correspond to a non-accepting or wrongly encoded run. We can check this by restricting to the first components of our combined alphabet. The self-loop that was used to encode Σ^* in poNFAs is replaced by a full copy of $\mathscr{A}_n$, with an additional transition from each state that allows us to leave this "loop". This does not simulate the full loop, but it allows us to navigate the entirety of our exponential word, which is all we need (more intuition in the next section; details in the appendix).

Theorem 37. *The universality problem for rpoNFAs is* PSPACE*-complete.*

5.3 PtNFAs

An NFA $\mathscr{A}$ is a *ptNFA* if it is an rpoNFA that is complete and confluent; the name comes from piecewise testable, since ptNFAs characterize piecewise testable languages [81, 86]. An alternative and our original definition of ptNFAs uses the notion of unique maximal state property, which we now briefly discuss.

Recall that 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 $\mathscr{A}$ over Σ with the state set Q can be turned into a directed graph $G(\mathscr{A})$ with the set of vertices Q where a pair $(p,q) \in Q \times Q$ is an edge in $G(\mathscr{A})$ if there is a transition from p to q in $\mathscr{A}$. For an alphabet $\Gamma \subseteq \Sigma$, we define the directed graph $G(\mathscr{A},\Gamma)$ 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 $\mathscr{A}$ satisfies the *unique maximal state* (UMS) property if, for every state q of $\mathscr{A}$, q is the unique maximal state of the connected component of $G(\mathscr{A},\Sigma(q))$ containing q.

To decide whether a DFA recognizes a piecewise testable language, Klíma and Polák [68] checks confluence while Trahtman [126] checks the UMS property.

Both notions have their advantages and an effect on the complexity. While Trahtman's algorithm runs in time quadratic with respect to the number of states and linear with respect to the size of the alphabet, Klíma and Polák's algorithm runs in time linear with respect to the number of states and quadratic with respect to the size of the alphabet. Notice that Cho and Huynh [27] proved that deciding piecewise testability for DFAs is NL-complete.

Although the notions of UMS and confluence coincide for DFAs, they differ for NFAs. The automaton in Figure 5.2 is confluent, but it does not satisfy the UMS property. Its language is not piecewise testable, since there is an infinite sequence $a, ab, aba, abab, \ldots$ that alternates between accepted and non-accepted states, and hence there is a non-trivial cycle in the corresponding minimal DFA.

We now relate these two notions and use the following lemma as an alternative definition of ptNFAs (we used it as a definition in our previous work [81]).

Lemma 38. *PoNFAs that are complete and satisfy the UMS property are exactly ptNFAs.*

5.3.1 Deciding Universality for ptNFAs

We now study the complexity of deciding universality for ptNFAs.

For unary alphabets, deciding universality for ptNFAs is solvable in polynomial time [72]. We now improve the result and show that the problem can be efficiently parallelized.

Theorem 39. *Deciding universality for ptNFAs over a unary alphabet is* NL*-complete.*

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 the construction that non-equivalence for regular expressions with operations union and concatenation is NP-complete even if one of them is of the form Σ^n for some fixed n [57, 121].

Theorem 40. *Deciding universality for ptNFAs over a fixed alphabet is* coNP*-complete even if the alphabet is binary.*

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 [70], or (2) to use a reduction from the word problem of a polynomially-space-bounded Turing machines à la Aho, Hopcroft and Ullman [1].

To use the union-universality problem for our purposes, we would need to use partially ordered DFAs rather than general DFAs to ensure that the union of the DFAs is partially ordered. However, we have shown 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 for which we can prove a stronger result (cf. Theorem 18). The *DFA-intersection emptiness* problem asks, given n DFAs, whether the intersection of their languages is empty. Indeed, the union of n DFA languages is universal if and only if the intersection of their complements is empty. Thus, we have the following corollary.

Corollary 41. *The poDFA-union-universality problem is* coNP*-complete.* □

We point out that the result cannot be further improved by restricting the size of the alphabet since the intersection-emptiness problem for unary poNFAs can be solved in polynomial time. Indeed, if there is a word in the intersection $\bigcap_{i=1}^{n} L(\mathscr{A}_i)$, then there is one that is a prefix of the word $a^{k_1+\cdots+k_n}$, where k_i is the depth of $\mathscr{A}_i$, which is of polynomial length.

It can be shown, and it is somehow intuitive, that every rpoNFA is a union of a number of poDFAs. In other words, every rpoNFA can be decomposed into the union of a number of poDFAs. Then the question whether an rpoNFA is universal is equivalent to the question whether the union of the languages of the poDFAs is universal. Since deciding universality for rpoNFAs is PSPACE-complete (and we show the same complexity for ptNFAs), the previous corollary implies that the decomposition cannot be constructed in polynomial time.

Thus, we cannot adapt Kozen's construction to show PSPACE-hardness of deciding universality for ptNFAs. The situation is, however, different for the reduction from the word problem of a polynomially-space-bounded Turing machines, which we have modified to prove PSPACE-hardness for the problem if the alphabet may grow polynomially with the number of states of the automaton.

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

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

$$\Sigma^* K \Sigma^* \tag{5.1}$$

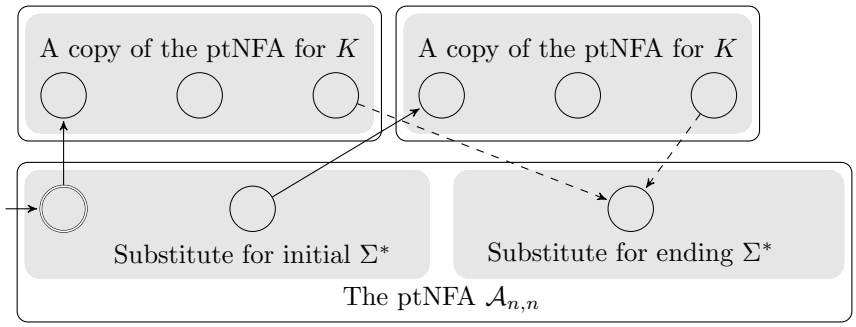

Figure 5.3: Construction of a self-loop-deterministic poNFA (solid edges) solving problem **(i)**, illustrated for two copies of the ptNFA for K, and its completion to a ptNFA (dashed edges) solving problem **(ii)**

where K is a finite language of words of length $O(p(|x|))$. Intuitively, K encodes possible violations of a correct computation of $\mathscr{M}$ 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 $\mathscr{M}$. These checks are local, involving at most two consecutive configurations of $\mathscr{M}$, each of polynomial size. Hence they can be encoded as the finite language K. The initial segment Σ^* of (5.1) 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.

Nonetheless, this idea cannot be directly used to prove our result for two reasons:

(i) Although expression (5.1) 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;

(ii) The constructed poNFA may be incomplete and its "standard" completion by adding the missing transitions to a new sink state may violate confluence.

A first observation to overcome these problems is that the length of the encoding of a computation of $\mathscr{M}$ on x is at most exponential with respect to the size of $\mathscr{M}$ and x. It would therefore be sufficient to replace the initial segment Σ^* in (5.1) by prefixes of an exponentially long word. However, such a word cannot be constructed by a polynomial-time reduction. Instead, we replace the leading Σ^* with a ptNFA encoding such an exponential word, which exists and is of polynomial size as we show in Lemma 42 – there we construct, in polynomial time, a ptNFA $\mathscr{A}_{n,n}$ that accepts all words but a single one, $W_{n,n}$, of exponential length.

Lemma 42. *For all integers $k, n \geq 1$, there exists a ptNFA $\mathscr{A}_{k,n}$ over an n-letter alphabet with $n(2k+1)+1$ states, such that the unique non-accepted word of $\mathscr{A}_{k,n}$ is of length $\binom{k+n}{k} - 1$.*

Since the language K of (5.1) is finite, and hence piecewise testable, there is a ptNFA recognizing K. For every state of $\mathscr{A}_{n,n}$, we make a copy of the ptNFA for K and identify its initial state with the state of $\mathscr{A}_{n,n}$ if it does not violate self-loop-determinism; see Figure 5.3 for an illustration. We keep track of the words read by both $\mathscr{A}_{n,n}$ and the ptNFA for K by taking the Cartesian product of their alphabets. A letter is then a pair of symbols, where the first symbol is the input for $\mathscr{A}_{n,n}$ and the second is the input for the ptNFA for K. A word over this alphabet is accepted if the first components do not form the word $W_{n,n}$ or the second components form a word that is not a correct encoding of a run of $\mathscr{M}$ on x. This results in a self-loop-deterministic poNFA that overcomes problem **(i)**.

However, this technique is not sufficient to resolve problem **(ii)**. Although the construction yields a self-loop-deterministic poNFA (rpoNFA) that is universal if and only if the regular expression R_x is [72], it is incomplete and its "standard" completion by adding the missing

$A\backslash B$	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 5.2: Complexity of deciding inclusion $L(A) \subseteq L(B)$ (unary/fixed[/growing] alphabet), all results are complete for the given class

transitions to an additional sink state violates confluence. Because of different expressive powers, it is not always possible to complete an rpoNFA to obtain a ptNFA. But we show that it is possible in our case because the length of the input that is of interest is bounded by the length of the word $W_{n,n}$. The maximal state of $\mathscr{A}_{n,n}$ is accepting, and therefore all the missing transitions can be added so that the paths required by confluence meet in the maximal state of $\mathscr{A}_{n,n}$. Since all words longer than $|W_{n,n}|$ are accepted by $\mathscr{A}_{n,n}$, we could complete the self-loop-deterministic poNFA by adding paths longer than $|W_{n,n}|$ to the maximal state of $\mathscr{A}_{n,n}$. However, this cannot be done by a polynomial-time reduction, since the length of $W_{n,n}$ is exponential. Instead, we add a ptNFA to encode such paths in the formal definition of $\mathscr{A}_{n,n}$ as given in Lemma 42. We then ensure confluence by adding the missing transitions to states of the ptNFA $\mathscr{A}_{n,n}$ from which the unread part of $W_{n,n}$ is not accepted and from which the maximal state of $\mathscr{A}_{n,n}$ is reachable under the symbol of the added transition. The second condition ensures confluence, since all the transitions meet in the maximal state of $\mathscr{A}_{n,n}$. The idea is illustrated in Figure 5.3.

Theorem 43. *Deciding universality for ptNFAs is* PSPACE*-complete.*

5.4 Inclusion and Equivalence of Partially Ordered NFAs

Universality is closely related to the inclusion and equivalence problems, which are of interest mainly from the point of view of optimization, e. g., in query answering.

Given two languages K and L over Σ, the *inclusion problem* asks whether $K \subseteq L$ and the *equivalence problem* asks whether $K = L$. Universality can then be expressed as the inclusion $\Sigma^* \subseteq L$ or the equivalence $\Sigma^* = L$.

Although equivalence means two inclusions, complexities of these two problems may differ significantly, e. g., inclusion is undecidable for deterministic context-free languages [42] while equivalence is decidable [112].

The relation of universality to inclusion and equivalence lies in the fact that the complexity of universality provides a lower bound on the complexity of both inclusion and equivalence. Therefore, it remains to show memberships of our results summarized in Tables 5.2 and 5.3.

The complexity of inclusion and equivalence for regular expressions of special forms has been investigated by Martens et al. [79]. For a few of them, PSPACE-completeness of the inclusion problem has been achieved. The results were established for alphabets of unbounded size. Since some of the expressions define languages expressible by poNFAs, we readily have that the inclusion problem for poNFAs is PSPACE-complete. However, using Theorem 20 and the well-known PSPACE upper bound on inclusion and equivalence for NFAs, we obtain that the inclusion and equivalence problems for poNFAs are PSPACE-complete even if the alphabet is binary.

The expressions in Martens et al. [79] cannot be expressed as rpoNFAs. Hence the question for rpoNFAs was open. Using Theorem 37 and the upper bound for NFAs, we can easily establish

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

that the inclusion and equivalence problems for rpoNFAs are PSPACE-complete. If the alphabet is fixed, the complexity of inclusion (and of equivalence) is covered by Theorem 32.

For the unary case, it is known that the inclusion and equivalence problems for NFAs over a unary alphabet are coNP-complete [56, 121]. For poNFAs we have shown that the inclusion and equivalence problems for poNFAs over a unary alphabet are NL-complete.

We now briefly explain the results summarized in the tables. Let $\mathscr{A}$ be an automaton of any of the considered types. We now discuss the cases depending on the type of $\mathscr{B}$. We assume that both automata are over the same alphabet specified by $\mathscr{B}$.

If $\mathscr{B}$ is a DFA, then $L(\mathscr{A}) \subseteq L(\mathscr{B})$ if and only if $L(\mathscr{A}) \cap L(\overline{\mathscr{B}}) = \emptyset$, which can be checked in NL (or in L if both automata are unary DFAs), where $\overline{\mathscr{B}}$ denotes the DFA obtained by complementing $\mathscr{B}$. This covers the first column of Table 5.2.

If $\mathscr{B}$ is an rpoNFA over a fixed alphabet, then deciding $L(\mathscr{A}) \subseteq L(\mathscr{B})$ is in coNP [72, 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.

If $\mathscr{B}$ is a unary poNFA, we distinguish several cases. First, deciding whether the language of an NFA is finite is in NL. Thus, if $L(\mathscr{A})$ is infinite and $L(\mathscr{B})$ is 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 $L(\mathscr{B})$ is infinite, then there is n bounded by the number of states of $\mathscr{B}$ such that $L(\mathscr{B})$ 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 $L(\mathscr{A})$ that is not in $L(\mathscr{B})$, which can again be checked in NL.

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

If $\mathscr{B}$ is a unary NFA, then if $L(\mathscr{B})$ is finite, we proceed as in the case of $\mathscr{B}$ being a unary poNFA. Therefore, assume that $L(\mathscr{B})$ is infinite and $\mathscr{B}$ has n states. Then the minimal DFA recognizing $L(\mathscr{B})$ has at most 2^n states (a better bound is shown by Chrobak [28]). If the inclusion $L(\mathscr{A}) \subseteq L(\mathscr{B})$ does not hold and $\mathscr{A}$ has m states, then there exists $k \leq m \cdot 2^n$, the number of states of $\mathscr{A} \times \overline{\mathscr{B}}$, such that $a^k \in L(\mathscr{A}) \setminus L(\mathscr{B})$. 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.

The main papers [72, 84, 85] on which the results of this chapter are based are attached in Appendix **??**.

6. Piecewise Testable Languages

Recall that a regular language over Σ is *piecewise testable* (PT) 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 is bounded by a constant k, then the language is *k-piecewise testable* (k-PT). Piecewise testable languages are exactly those regular languages whose syntactic monoid is $\mathscr{J}$-trivial [115]. Simon [116] provided various characterizations of piecewise testable languages, e. g., in terms of monoids or automata. These languages are of interest in many disciplines of mathematics, such as semigroup theory [3, 4, 96] for their relation to Green's relations or in logic on words [38, 98] for their relation to first-order logic FO[$<$] and the Straubing-Thérien hierarchy [95, 122, 124, 125]. They are indeed studied in formal languages and automata theory [68], recently mainly in the context of separation [98, 127]. Piecewise testable languages form a strict subclass of star-free languages or, in other words, of the languages definable by LTL. They are investigated in natural language processing [43, 103], in cognitive and sub-regular complexity [104], in learning theory [45, 69], and in databases in the context of XML schema languages [34, 53, 54]. They have been extended from words to trees [13, 46].

6.1 Finding a Boolean Combination

We studied the problem of translating an automaton accepting a piecewise testable language into a Boolean combination of languages of the form $L_{a_1 a_2 \cdots a_n} = \Sigma^* a_1 \Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$. Our motivation comes from the simplification of XML Schema, since such expressions resemble XPath-like expressions used in the BonXai schema language [78].

Since every piecewise testable language is k-PT for some $k \geq 0$, and a k-PT language is also $(k+1)$-PT, we focus on the Boolean combination of languages L_u, where the length of u is bounded by the minimal k for which the language is k-PT. From this point of view, we are interested in translating an automaton to the form of a generalized regular expression (an expression allowing the operation of complement). Notice that generalized regular expressions can be non-elementary more succinct than classical regular expressions [37, 121, 48] and that not much is known about these transformations [41].

There are many possible different Boolean combinations describing the same language, and it is not clear which of them is the best representation. The choice significantly depends on the applications. We are interested in those Boolean combinations that resemble the disjunctive normal form of logical formulas rather than in the most concise representation.

The basic idea of our translation can be outlined as follows. Let L be a language over Σ represented by its minimal DFA, and let the equivalence relation $\sim_k$ on Σ^* be defined by $u \sim_k v$ if u and v have the same sets of (scattered) subwords up to length k, denoted by $sub_k(u) = sub_k(v)$. Then L is piecewise testable if and only if there exists a nonnegative integer k such that $\sim_k \subseteq \sim_L$, where $\sim_L$ is the Myhill congruence [91]. Thus, every k-PT language is a finite union of $\sim_k$-classes. As shown, for instance, by Klíma [66], the $\sim_k$-classes can be described by languages of

the form

$$[w]_{\sim_k} = \bigcap_{u \in sub_k(w)} L_u \cap \bigcap_{u \notin sub_k(w), |u| \leq k} L_u^c,$$

where L_u^c denotes the complement of L_u. The high-level approach is thus:

1. Check whether the regular language L is piecewise testable.
2. If so, compute the minimal $k \geq 0$ for which L is k-piecewise testable.
3. Compute the finite number of representatives of the equivalence classes that form the union of the language L, express them as above and form their union.

6.1.1 Step 1: Checking Piecewise Testability

The complexity of the first step has been studied in the literature. Simon [115] proved that PT languages are exactly those regular languages whose syntactic monoid is $\mathscr{J}$-trivial, which gives decidability. Stern [120] showed that the problem is decidable in polynomial time for languages represented by DFAs and Cho and Huynh [27] proved NL-completeness for DFAs. Later, Trahtman [126] showed that the problem is solvable in time quadratic with respect to the number of states of the DFA and linear with respect to the size of the alphabet, and Klíma and Polák [68] gave an algorithm that is quadratic in the size of the input alphabet and linear in the number of states of the DFA. For languages represented by NFAs, we have shown that the problem is PSPACE-complete [83].

6.1.2 Step 2: Computing the Minimal k

The second step gives rise to the *k-piecewise testability* problem formulated as follows:

INPUT: an automaton (DFA or NFA) $\mathscr{A}$
OUTPUT: YES if and only if $L(\mathscr{A})$ is k-piecewise testable

The problem is trivially decidable for any k because there are only finitely many k-PT languages over the alphabet of $\mathscr{A}$. We investigated the computational complexity of this problem.

The CONP upper bound complexity for DFAs has been independently obtained by Hofman and Martens [53] (formulated in terms of separability and presented without proof), Klíma, Kunc and Polák [67], and Masopust and Thomazo [86]. Klíma, Kunc and Polák further proved that the problem is CONP-complete for DFAs if $k \geq 4$. What is the complexity if $k < 4$? We now answer this question.

0-Piecewise Testability Let $\mathscr{A}$ be a minimal DFA over an alphabet Σ. The language $L(\mathscr{A})$ is 0-PT if and only if it has a single state, that is, it recognizes either Σ^* or $\emptyset$. Thus, it is decidable in $O(1)$ whether $L(\mathscr{A})$ is 0-PT.

1-Piecewise Testability We showed that the 1-PT problem belongs to AC^0, which is a strict subset of LOGSPACE. There is an infinite hierarchy of classes Σ_i (Π_i) in AC^0 based on the number of alternating levels of disjunctions and conjunctions. Specifically, Σ_i (Π_i) is the class of problems solvable by uniform families of unlimited fan-in circuits of constant depth and polynomial size with i alternating levels of AND and OR gates (with NOT gates only in the input) and with the output gate being an OR gate (an AND gate) [10].

Theorem 44. *To decide whether a minimal DFA recognizes a 1-PT language is in AC^0.*

The proof gives that the problem belongs to Π_3 of the hierarchy. However, we do not know whether the 1-PT problem is Π_3-hard in the AC^0 hierarchy or not.

As a consequence of our construction, if a minimal DFA over Σ has more than $2^{|\Sigma|}$ states, then its language is not 1-piecewise testable.

2-Piecewise Testability We showed that to decide whether a minimal DFA recognizes a 2-PT language is NL-complete. Notice that this complexity coincides with the complexity of deciding whether a regular language is piecewise testable, that is, whether there exists a k for which the language is k-piecewise testable.

Theorem 45. *To decide whether a minimal DFA recognizes a 2-PT language is* NL*-complete.*

Blanchet-Sadri [12] has shown that 1-PT languages are characterized as those languages whose syntactic monoid satisfies the equations $x = x^2$ and $xy = yx$, and that 2-PT languages are those languages whose syntactic monoid satisfies the equations $xyzx = xyxzx$ and $(xy)^2 = (yx)^2$. These equations could be directly used to achieve NL algorithms. Our characterizations [86], however, improve these results and show that, for 1-PT languages, it is sufficient to verify the equations $x = x^2$ and $xy = yx$ on letters (generators) rather than on words, and that for 2-PT languages, equation $xyzx = xyxzx$ can be verified on letters (generators) up to the element y, which is a word (a general element of the monoid). Our results thus decrease the complexity of the problems. In addition, the partial order and confluence can be checked instead of the equation $(xy)^2 = (yx)^2$.

3-Piecewise Testability For this case, we made use of the equations $(xy)^3 = (yx)^3$, $xzyxvxwy = xzxyxvxwy$ and $ywxvxyzx = ywxvxyxzx$ characterizing the variety of 3-PT languages [12] to show NL-completeness of the 3-piecewise testability problem.

Theorem 46. *To decide whether a minimal DFA recognizes a 3-PT language is* NL*-complete.*

Structural Upper Bound on k

There is an interesting observation by Klíma and Polák [68] that if the depth of a minimal DFA recognizing a PT language is k, then the language is k-PT. (Bounds for finite languages and upward and downward closures have been investigated by Karandikar and Schnoebelen [64].) The observation reduces Step 2 of our approach to solving a finite number of k-piecewise testability problems, since the upper bound on k is given by the depth of the minimal DFA equivalent to $\mathscr{A}$.

The opposite implication does not hold, and hence we investigated the relationship between the depth of an NFA and k-piecewise testability of its language. We showed that, for every $k \geq 0$, there exists a k-PT language with an NFA of depth $k-1$ and with the minimal DFA of depth $2^k - 1$.

Theorem 47. *For every $n \geq 1$, there exists an n-PT language that is not $(n-1)$-PT, it is recognized by an NFA of depth $n-1$, and the minimal DFA recognizing it has depth $2^n - 1$.*

Although it is a well-known fact that DFAs can be exponentially larger than NFAs, an interesting by-product of the proof of the previous theorem [86] is that there are NFAs such that all the exponential number of states of their minimal DFAs form a simple path. Since the reverse of the NFA constructed in the proof is a DFA, our result also contributes to the state complexity of the reverse of piecewise testable languages, cf. [21, 61].

6.1.3 Step 3: Computation of Representatives

The last step of our approach is to compute those $\sim_k$-classes, whose union forms the language L, and to express them as the intersection of languages of the form L_u or its complements. To identify these equivalence classes, we make use of the $\sim_k$-canonical DFA, whose states correspond to $\sim_k$-classes. We construct the $\sim_k$-canonical DFA and compute its accepting states by intersection with the input automaton. The accepting states then represent the $\sim_k$-classes forming the language L. The $\sim_k$-canonical DFA can be effectively constructed. Moreover, although the precise size of the $\sim_k$-canonical DFA is not known, see the estimations in Karandikar, Kufleitner and Schnoebelen [63], we show that the tight upper bound on its depth is $\binom{k+n}{k} - 1$, where n is the cardinality of the alphabet. This result has also been independently obtained by Klíma, Kunc and Polák [67].

Theorem 48. *For any natural numbers k and n, the depth of the minimal DFA recognizing a k-PT language over an n-letter alphabet is at most* $\binom{k+n}{k} - 1$. *The bound is tight for any k and n.*

As already pointed out, this work can be seen as translating an automaton into a form of a generalized regular expression. Generalized regular expressions can be non-elementary more succinct than classical regular expressions, however it is not yet known whether this non-elementary succinctness can be witnessed by a piecewise testable language.

6.2 Piecewise Testability and Nondeterministic Automata

The knowledge of a minimal or reasonably small k for which the language is k-piecewise testable is of interest in many applications, see, e. g., Martens et al. [78]. The complexity to test whether a piecewise testable language is k-piecewise testable is coNP-complete for $k \geq 4$ if the language is given as a DFA [67] and PSPACE-complete if the language is given as an NFA [86].

Theorem 49. *The k-piecewise testability problem for NFAs is* PSPACE*-complete.*

In Section 5.3, we defined a class of nondeterministic finite automata, called *ptNFAs*. Let us recall the definition of ptNFAs we use in this section.

Definition 50. An NFA $\mathscr{A}$ is called a *ptNFA* if it is partially ordered, complete, and satisfies the UMS property.

The prefix "pt" in their name comes from piecewise testable, since, as we have shown [86], they characterize piecewise testable languages. And indeed include all minimal DFAs recognizing piecewise testable languages.

Theorem 51. *A regular language is piecewise testable if and only if it is recognized by a ptNFA.*

The reason why we use the UMS property in the definition of ptNFAs rather than confluence is simply because confluence does not naturally generalize to NFAs. However, it is known that partially ordered NFAs characterize the level $\frac{3}{2}$ of the Straubing-Thérien hierarchy [111] and that rpoNFAs characterize $\mathscr{R}$-trivial languages [71]. Adding confluence and completeness on top of these properties results in ptNFAs [71].

To check whether an NFA is a ptNFA requires to check whether the automaton is partially ordered, complete and satisfies the UMS property. The violation of these properties can be tested by several reachability tests, and hence its complexity belongs to coNL=NL. On the other hand, checking the properties is NL-hard even for minimal DFAs [27].

Theorem 52. *It is* NL*-complete to check whether an NFA is a ptNFA.*

	Unary alphabet	Fixed alphabet	Arbitrary alphabet	
	$\lvert\Sigma\rvert = 1$	$\lvert\Sigma\rvert \geq 2$	$k \leq 3$	$k \geq 4$
DFA	L-c	NL-c	NL-c	coNP-c [67]
ptNFA	NL-c	coNP-c	PSPACE-c	
rpoNFA	NL-c	coNP-c	PSPACE-c	
poNFA	NL-c	PSPACE-c	PSPACE-c	
NFA	coNP-c	PSPACE-c	PSPACE-c	

Table 6.1: Complexity of deciding k-piecewise testability

We show that, analogously to the minimal DFA case, the depth of ptNFAs provides an upper bound on k-piecewise testability and that this new bound is up to exponentially lower than the one given by minimal DFAs.

Theorem 53. *If the depth of a ptNFA $\mathscr{A}$ is k, then the language $L(\mathscr{A})$ is k-piecewise testable.*

In other words, the previous theorem says that if k is the minimum number for which a piecewise testable language L is k-piecewise testable, then the depth of any ptNFA recognizing L is at least k. This property does not hold for general NFAs, and the gap between k-piecewise testability and the depth of NFAs can be arbitrarily large.

The opposite implication of Theorem 53 does not hold. Although the depth of ptNFAs is more suitable to provide bounds on k-piecewise testability, the depth is significantly influenced by the size of the alphabet. For instance, for an alphabet Σ, the language $L = \bigcap_{a\in\Sigma} L_a$ of all words containing all letters of Σ is a 1-piecewise testable language such that any NFA recognizing it requires at least $2^{|\Sigma|}$ states and is of depth $|\Sigma|$. The depth follows from the fact that the shortest accepted word is of length $|\Sigma|$, and hence any path from an initial state to an accepting state must be of length at least $|\Sigma|$.

The dependence on the alphabet is even stronger as shown below.

Lemma 54. *For any alphabet of cardinality $n > 1$, there exists an n^2-piecewise testable language such that any NFA recognizing it is of depth at least n^{n-1}.*

6.3 Complexity

In this subsection, we discuss the complexity of deciding (k-)piecewise testability for languages given as a considered type of partially ordered NFA, as well as by a DFA or by an NFA.

6.3.1 Complexity of Deciding k-Piecewise Testability

Recall that 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 focus on the complexity of deciding k-piecewise testability for partially ordered automata. Our results are summarized in Table 6.1. These results revise and improve our previous results of paper [81] and are part of the manuscript [85]. The proof technique is based on the following new lemma.

Lemma 55. *Let $k \geq 0$ be a constant. Then the universality problem is log-space reducible to the k-piecewise testability problem.*

	$\lvert\Sigma\rvert = 1$	$\lvert\Sigma\rvert \geq 2$	Σ is growing
DFA	L-c	NL-c [27][1]	NL-c [27]
rpoNFA	✓	coNP-c	PSPACE-c
poNFA	✓	PSPACE-c	PSPACE-c
NFA	coNP-c	PSPACE-c	PSPACE-c

Table 6.2: Complexity of deciding piecewise testability

We then immediately have the following results.

Theorem 56. *Deciding k-piecewise testability for ptNFAs is* PSPACE*-complete.*

Theorem 57. *Let Σ be a fixed alphabet with at least two letters. Deciding k-piecewise testability for ptNFAs over Σ is* coNP*-complete.*

This result is in contrast with an analogous result for DFAs, where deciding k-piecewise testability for DFAs over a fixed alphabet is in PTIME [67]. A more precise complexity can be shown.

Theorem 58. *Let Σ be a fixed alphabet with at least two letters. Deciding k-piecewise testability for DFAs over Σ is* NL*-complete.*

Theorem 59. *Let Σ be a fixed alphabet with at least two letters. Deciding k-piecewise testability for poNFAs over Σ is* PSPACE*-complete.*

Theorems 57 and 59 show hardness even for binary alphabets, which improves our recent results where the alphabet had at least three letters [81]. Furthermore, we point out that hardness in Theorem 57 does not follow from the coNP-hardness proof of Klíma et al. [67] 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.

Theorem 60. *Deciding k-piecewise testability for ptNFAs over a unary alphabet is* NL*-complete. It holds even if k is given as part of the input.*

Theorem 61. *Deciding k-piecewise testability for poNFAs and rpoNFAs over a unary alphabet is* NL*-complete. It holds even if k is given as part of the input.*

Theorem 62. *Deciding k-piecewise testability for DFAs over a unary alphabet is L-complete.*

Theorem 63. *Deciding k-piecewise testability for NFAs over a unary alphabet is* coNP*-complete.*

6.3.2 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 6.2.

[1] Cho and Huynh [27] 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.

To simplify proofs, we would like to use a result similar to Lemma 55. 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 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 $\mathscr{R}$-trivial languages, poDFA-languages, and rpoNFA-languages coincide.

Theorem 64. *The classes of unary poNFA languages, unary $\mathscr{R}$-trivial languages, and unary piecewise testable languages coincide.*

Theorem 65. *Deciding piecewise testability for DFAs over a unary alphabet is L-complete.*

Theorem 66. *Deciding piecewise testability for NFAs over a unary alphabet is* coNP*-complete.*

Theorem 67. *Let Σ be a fixed alphabet with at least two letters. Deciding piecewise testability for poNFAs over Σ is* PSPACE*-complete.*

Theorem 68. *Deciding piecewise testability for rpoNFAs is* PSPACE*-complete.*

Theorem 69. *Deciding piecewise testability for rpoNFAs over a fixed alphabet is* coNP*-complete even if the alphabet is binary.*

The main papers [81, 86] on which the results of this chapter are based are attached in Appendix **??**, including the manuscript [85].

7. Applications

We now present a few applications of our results in regular expressions used in the schema languages for XML data according to the W3C standards, and in the system-theoretic properties of discrete-event systems.

7.1 Deterministic Regular Expressions

In this section, we discuss the relationship of partially ordered NFAs to deterministic regular expressions (DREs) [16]. DREs are of interest in schema languages for XML data – Document Type Definition (DTD) and XML Schema Definition (XSD) – since the World Wide Web Consortium standards require that the regular expressions in their specification must be deterministic.

The *regular expressions* (REs) over an alphabet Σ are defined as follows: $\emptyset$, ε and a, $a \in \Sigma$, are regular expressions. If r and s are regular expressions, then $(r \cdot s)$, $(r+s)$ and $(r)^*$ are regular expressions. The language defined by a regular expression r, denoted by $L(r)$, is inductively defined by $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$, $L(a) = \{a\}$, $L(r \cdot s) = L(r) \cdot L(s)$, $L(r+s) = L(r) \cup L(s)$, and $L(r^*) = \{\varepsilon\} \cup \bigcup_{i=1}^{\infty} L(r)^i$, where $L(r) \cdot L(s)$ denotes the concatenation of the languages $L(r)$ and $L(s)$. Let r be a regular expression, and let $\overline{r}$ be a regular expression obtained from r by replacing the i-th occurrence of symbol a in r by a_i. For instance, if $r = (a+b)^*b(a+b)$, then $\overline{r} = (a_1+b_1)^*b_2(a_2+b_3)$. A regular expression r is *deterministic* (one-unambiguous [16] or DRE) if there are no words wa_iv and wa_jv' in $L(\overline{r})$ such that $i \neq j$. For instance, the expression $(a+b)^*b(a+b)$ is not deterministic since the strings b_2a_2 and $b_1b_2a_2$ are both in $L((a_1+b_1)^*b_2(a_2+b_3))$. A regular language is *DRE definable* if there exists a DRE that defines it. Brüggemann-Klein and Wood [16] showed that not all regular languages are DRE definable.

The important question is then whether a given regular language is DRE definable. This problem has been shown to be PSPACE-complete [32]. Since the language of the expression $(a+b)^*b(a+b)$ is not DRE definable [16], but it can be easily expressed by a poNFA, DRE definability is nontrivial for poNFAs. However, the complexity of deciding whether a poNFA language is DRE definable follows from the existing results, namely from the proof in Bex et al. [11] showing PSPACE-hardness of DRE-definability for regular expressions; the regular expression constructed there can be expressed as a poNFA. Thus, we readily have the following:

Corollary 70. *To decide whether the language of a poNFA is DRE definable is* PSPACE*-complete.*

On the other hand, the problem is trivial for the languages of rpoNFAs, which makes rpoNFAs interesting for the XML schema languages.

Theorem 71. *Every rpoNFA language is DRE definable.*

To prove the theorem, we need to introduce a few notions. For a state q of an NFA $\mathscr{A}$, the *orbit of* q is the maximal strongly connected component of $\mathscr{A}$ containing q. State q is called a

gate of the orbit of q if q is accepting or has an outgoing transition that leaves the orbit. The orbit automaton of state q is the sub-automaton of $\mathscr{A}$ consisting of the orbit of q in which the initial state is q and the accepting states are the gates of the orbit of q. We denote the orbit automaton of q by $\mathscr{A}_q$. The orbit language of q is $L(\mathscr{A}_q)$. The orbit languages of $\mathscr{A}$ are the orbit languages of states of $\mathscr{A}$.

An NFA $\mathscr{A}$ has the *orbit property* if, for every pair of gates q_1, q_2 in the same orbit in $\mathscr{A}$, the following properties hold: (i) q_1 is accepting if and only if q_2 is accepting, and (ii) for all states q outside the orbit of q_1 and q_2, there is a transition $q \in q_1 \cdot a$ if a and only if there is a transition $q \in q_2 \cdot a$.

Brüggemann-Klein and Wood [16] have shown that the language of a minimal DFA $\mathscr{A}$ is DRE-definable if and only if $\mathscr{A}$ has the orbit property and all orbit languages of $\mathscr{A}$ are DRE-definable.

Lemma 72. *Every language of a minimal partially ordered DFA is DRE-definable.*

This lemma implies Theorem 71. Note that the converse of Theorem 71 does not hold. The expression $b^*a(b^*a)^*$ is deterministic [32] and it can be easily verified that its minimal DFA is not partially ordered, and hence the expression defines a language that is not $\mathscr{R}$-trivial (rpoNFA definable).

7.2 Detectability

A *discrete event system* (DES) is modeled as an NFA G with all states accepting. Hence we simply write $G = (Q, \Sigma, \delta, I)$ without specifying the set of accepting states. The alphabet Σ is partitioned into two disjoint subsets Σ_o and $\Sigma_{uo} = \Sigma \setminus \Sigma_o$, where Σ_o is the set of *observable events* and Σ_{uo} the set of *unobservable events*.

The detectability problem of discrete event systems is a question whether the current and subsequent states of a DES can be determined based on observations. The problem was introduced and studied by Shu et al. [113, 114]. Detectability generalizes other notions studied in the literature [23, 101], such as stability of Ozveren and Willsky [92]. Shu et al. further argue that many practical problems can be formulated as detectability.

Four variants of detectability have been defined: strong and weak detectability and strong and weak periodic detectability [114]. Let Σ be an alphabet, $\Sigma_o \subseteq \Sigma$ the set of observable events, and P the projection from Σ to Σ_o; the *projection* $P\colon \Sigma^* \to \Sigma_o^*$ is a morphism defined by $P(a) = \varepsilon$ for $a \in \Sigma \setminus \Sigma_o$, and $P(a) = a$ for $a \in \Sigma_o$. As usual when detectability is studied, we make the following two assumptions on the DES:

1. G is *deadlock free*, that is, for every state of the system, at least one event can occur. Formally, for every $q \in Q$, there is $\sigma \in \Sigma$ such that $\delta(q, \sigma) \neq \emptyset$.

2. No loop in G consists solely of unobservable events: for every $q \in Q$ and every $w \in \Sigma_{uo}^+$, $q \notin \delta(q, w)$.

The set of infinite sequences of events generated by the DES G is denoted by $L^\omega(G)$. Given $Q' \subseteq Q$, the set of all possible states reachable from the states of Q' after observing a word $t \in \Sigma_o^*$ is denoted by $R(Q', t) = \cup_{w \in \Sigma^*, P(w)=t} \delta(Q', w)$. For $w \in L^\omega(G)$, we denote the set of its prefixes by $Pr(w)$.

Definition 73. A DES $G = (Q, \Sigma, \delta, I)$ is *strongly detectable* with respect to Σ_{uo} if we can determine, after a finite number of observations, the current and subsequent states of the system for all trajectories of the system, i. e.,

$$(\exists n \in \mathbb{N})(\forall s \in L^\omega(G))(\forall t \in Pr(s))|P(t)| > n \Rightarrow |R(I, P(t))| = 1\,.$$

Definition 74. A DES $G = (Q, \Sigma, \delta, I)$ is *strongly periodically detectable* with respect to Σ_{uo} if we can periodically determine the current state of the system for all trajectories of the system, i. e.,

$$(\exists n \in \mathbb{N})(\forall s \in L^{\omega}(G))(\forall t \in Pr(s))(\exists t' \in \Sigma^*)tt' \in Pr(s) \wedge |P(t')| < n \wedge |R(I, P(tt'))| = 1\,.$$

Definition 75. A DES $G = (Q, \Sigma, \delta, I)$ is *weakly detectable* with respect to Σ_{uo} if we can determine, after a finite number of observations, the current and subsequent states of the system for some trajectories of the system, i. e.,

$$(\exists n \in \mathbb{N})(\exists s \in L^{\omega}(G))(\forall t \in Pr(s))|P(t)| > n \Rightarrow |R(I, P(t))| = 1\,.$$

Definition 76. A DES $G = (Q, \Sigma, \delta, I)$ is *weakly periodically detectable* with respect to Σ_{uo} if we can periodically determine the current state of the system for some trajectories of the system, i. e.,

$$(\exists n \in \mathbb{N})(\exists s \in L^{\omega}(G))(\forall t \in Pr(s))(\exists t' \in \Sigma^*)tt' \in Pr(s) \wedge |P(t')| < n \wedge |R(I, P(tt'))| = 1\,.$$

Shu et al. [114] investigated detectability for deterministic DESs. A deterministic DES is modeled as a deterministic finite automaton with a set of initial states rather than a single initial state. The motivation for more initial states comes from the observation that it is often not known which state the system is initially in. They designed exponential algorithms to decide detectability based on the computation of the observer.[1] Later, to be able to handle more problems, they extended their study to nondeterministic DESs and improved the algorithms for strong (periodic) detectability of nondeterministic DESs to polynomial time [113]. Concerning the complexity of deciding weak detectability, Zhang [130] showed that the problem is PSPACE-complete and that PSPACE-hardness holds even for deterministic DESs with all events observable. We improve these results to binary alphabets and to the DFA representation of DESs.

Theorem 77. *Deciding whether a deterministic DES over a binary alphabet is weakly (periodically) detectable is* PSPACE*-complete.*

By a minor modification of the construction in the proof of the previous theorem [82], we have the following corollary.

Corollary 78. *Deciding whether a DES modeled as a DFA is weakly (periodically) detectable is* PSPACE*-complete even if the DES has only three events, one of which is unobservable.*

The unobservable event in the corollary is unavoidable because any DES modeled as a DFA with all events observable is always in a unique state, and hence trivially detectable. We now show that two observable events are also necessary for PSPACE-hardness.

Theorem 79. *Deciding whether a DES over a unary alphabet is weakly detectable is in P, and whether it is weakly periodically detectable is in NP.*

Zhang's result gives rise to a question whether there are structurally simpler DESs with a tractable complexity of weak (periodic) detectability. The simplest DESs are acyclic DFAs, recognizing finite languages. Acyclic DESs are not deadlock-free. To fulfill deadlock-freeness, we consider DESs modeled as DFAs with cycles only in the form of self-loops. As pointed out in the previous sections, such DFAs recognize a subclass of regular languages strictly included in *star-free languages* [19]. Star-free languages are regular languages definable by *linear temporal logic* widely used as a specification language in automated verification. The nontrivial-acyclicity condition is satisfied by poDFAs and we have shown that poDFAs and rpoNFAs recognize the same class of languages [72].

[1] An observer is a DFA obtained by replacing unobservable events by ε and by the standard subset construction.

We have further shown that deciding weak (periodic) detectability remains PSPACE-complete even if the DES is modeled as a poDFA, or as an rpoNFA with all events observable. Consequently, the problem is intractable for basically all practical cases. We point out that Zhang [130] obtained his result by reducing the DFA-intersection problem. Since his construction does not introduce any non-trivial cycles, it could seem that it also shows the result for poDFAs. This is, however, not the case because the complexity of the poDFA-intersection problem is easier unless PSPACE =NP, c. f. Theorem 18. Therefore, to prove our results, we reduced the universality problem for rpoNFAs.

Theorem 80. *Deciding weak (periodic) detectability of DESs modeled as rpoNFAs is* PSPACE*-complete even if all events are observable.*

Moreover, intractability holds even if the DESs are modeled as poDFAs over a very small alphabet.

Theorem 81. *Deciding weak (periodic) detectability of DESs modeled as poDFAs over a five-letter alphabet, two of which are unobservable, is* PSPACE*-complete.*

On the other hand, we have shown that deciding strong (periodic) detectability is NL-complete.

Theorem 82. *Deciding whether a DES is strongly (periodically) detectable is* NL*-complete.*

Since NL is the class of problems that can be efficiently parallelized [6], we obtain that strong (periodic) detectability can be efficiently verified on a parallel computer. Note that Shu and Lin [113] designed a polynomial-time algorithm to decide strong (periodic) detectability, and hence the problem was known to be in P.

7.3 Opacity

Opacity is a property related to the privacy and security analysis of discrete-event systems. The system has a secret and an intruder, modeled as a passive observer with limited observation, tries to figure it out. The system is opaque if the secret is not revealed.

If the secret is modeled as a set of secret states, the notion is called a state-based opacity. State-based opacity was introduced by Bryans et al. [17] for systems modeled by Petri nets, and later adapted for systems modelled by (stochastic) automata by Saboori and Hadjicostis [107]. Intuitively, the system is opaque if the intruder never knows for sure that the system is in a secret state. Several variants of opacity have been introduced and studied in the literature [59]. Here we consider only current-state opacity.

Definition 83. Given a DES $G = (Q, \Sigma, \delta, I)$, a projection $P\colon \Sigma^* \to \Sigma_o^*$, a set of secret states $Q_S \subseteq Q$, and a set of non-secret states $Q_{NS} \subseteq Q$. System G is *current-state opaque* if for every word w such that $\delta(I,w) \cap Q_S \neq \emptyset$, there exists a word w' such that $P(w) = P(w')$ and $\delta(I,w') \cap Q_{NS} \neq \emptyset$.

The notion of opacity independent on the structure of the system was introduced by Badouel et al. [7] and Dubreil et al. [39] under the name language-based opacity. Language-based opacity is defined over a set of secret behaviors. Lin [74] further generalized that notion to two sets of behaviors (secret and non-secret) as follows.

Definition 84. Given a DES $G = (Q, \Sigma, \delta, I)$, a projection $P\colon \Sigma^* \to \Sigma_o^*$, a secret language $L_S \subseteq L(G)$, and a non-secret language $L_{NS} \subseteq L(G)$. System G is *language-based opaque* if $L_S \subseteq P^{-1}P(L_{NS})$.

Assuming that the languages L_S and L_{NS} are regular, Wu and Lafortune [129] provided transformations among several notions of opacity, including current-state opacity and language-based opacity. Their reductions can easily be modified to deterministic log-space reductions, and hence the following results considering current-state opacity also apply to language-based opacity.

The following lemma reduces current-state opacity to the inclusion problem.

Lemma 85. *Let $G = (Q, \Sigma, \delta, I)$ be a DES, $P\colon \Sigma^* \to \Sigma_o^*$ be a projection, $Q_S \subseteq Q$ be a set of secret states, and $Q_{NS} \subseteq Q$ be a set of non-secret states. Let $G_1 = (Q, \Sigma, \delta, I, Q_S)$ and $G_2 = (Q, \Sigma, \delta, I, Q_{NS})$. Then G is current-state opaque if and only if $P(L(G_1)) \subseteq P(L(G_2))$.*

Cassez et al. [26] pointed out that the verification of current-state opacity is at least as hard as deciding universality. Indeed, $L(G) = \Sigma^*$ if and only if G is current-state opaque with respect to $Q_S = Q \setminus F$ and $Q_{NS} = F$. This observation and Lemma 85 together with the results on the complexity of universality and inclusion give us tools to show lower and upper complexity bounds for deciding opacity.

Our first result shows that deciding current-state opacity is PSPACE-complete even if the system is given as a DFA, that is, even if the DES is deterministic with a single initial state.

Theorem 86. *Deciding current-state opacity for a DES modeled by a DFA with three events, one of which is unobservable, is* PSPACE*-complete.*

Proof. Membership in PSPACE was shown by Saboori [106] and also follows immediately from Lemma 85 and the fact that deciding inclusion for two NFAs is PSPACE-complete.

To show hardness, we reduce the DFA-union universality problem [70]. Let $\mathscr{A}_1, \ldots, \mathscr{A}_n$ be DFAs over the alphabet $\{0,1\}$. Without loss of generality, we may assume that the initial state of $\mathscr{A}_i$, for all $i = 1, \ldots, n$, is not reachable from any other state. Let G denote the nondeterministic union of all $\mathscr{A}_i$'s, that is, $L(G) = \cup_{i=1}^n L(\mathscr{A}_i)$. Kozen [70] showed that deciding whether $L(G) = \Sigma^*$ is PSPACE-hard, and hence deciding current-state opacity of G is PSPACE-hard by the observation of Cassez et al. [26], see the comment below Lemma 85. Notice that the transitions of G are deterministic, but G has n initial states, say $\{q_1, \ldots, q_n\}$. We further modify G by adding a new unobservable event a and the transitions (q_i, a, q_{i+1}), for $i = 1, \ldots, n-1$, and let q_1 be the sole initial state. Denoting the result by G', we can see that G' is a DFA, and that the observers[2] of G and G' coincide (here we needed the assumption that the initial state of $\mathscr{A}_i$ is not reachable from other states of $\mathscr{A}_i$; otherwise, the languages of the observers of G and G' could be different). Altogether, G is opaque if and only if G' is opaque, which completes the proof. □

An unobservable event in the previous theorem is unavoidable because any DFA with all events observable is always in a unique state, and therefore never opaque. We now show that having only one observable event makes the problem easier.

Theorem 87. *Deciding current-state opacity of a DES modeled by an NFA with a single observable event is* coNP*-complete.*

Proof. Membership in coNP follows from Lemma 85 and the fact that inclusion for unary NFAs is coNP-complete; hardness follows from the complexity of deciding universality for unary NFAs, which is coNP-complete [121]. □

These results give rise to a question whether there are structurally simpler systems for which the opacity verification is tractable. Structurally the simplest systems are acyclic DFAs with full observation, recognizing finite languages. However, these systems are never opaque. Nontrivial

[2] An observer is a DFA obtained by replacing unobservable events by ε and by the standard subset construction, considering only the reachable part.

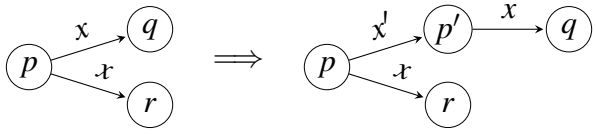

Figure 7.1: The 'determinization'; x' and p' are a new event and a new state

structures could then be acyclic NFAs that still recognize only finite languages. However, real systems are usually not that simple and often require additional properties, such as deadlock-freeness. Therefore, we consider automata with cycles in the form of self-loops (poNFAs and poDFAs) that are, in our opinion, structurally the simplest deadlock-free DES.

We then immediately obtain the following result.

Theorem 88. *Deciding current-state opacity of a DES modeled by a poNFA with only two events, both of which are observable, is* PSPACE*-complete.*

Proof. Membership in PSPACE follows from Lemma 85 and the results on the complexity of inclusion for poNFAs, and hardness from the fact that deciding universality for poNFAs with only two events is PSPACE-complete [72]. □

The situation is again easier if the model has only a single observable event.

Theorem 89. *Deciding current-state opacity of a DES modeled by a poNFA with a single observable event is NL-complete.*

Proof. Membership in NL follows from Lemma 85 and the corresponding compexity of inclusion, and hardness from the fact that deciding universality for unary poNFAs is NL-complete [72]. □

We now consider DES modeled by poDFAs. Since every DFA with all events observable is always in a unique state, and hence never opaque, some unobservable events are necessary to ensure opacity. We show that four events, two of which are unobservable, make the opacity verification PSPACE-complete even for poDFAs. Consequently, the problem is PSPACE-complete for basically all practical cases.

Theorem 90. *Deciding current-state opacity for poDFAs over an alphabet with four events, two of which are unobservable, is* PSPACE*-complete.*

Proof. Membership in PSPACE follows from Lemma 85 and the corresponding complexity of inclusion.

Let $\mathscr{A} = (Q, \{0,1\}, \delta, I, F)$ be a poNFA. By Theorem 88, deciding current-state opacity for poNFAs with two events, both observable, is PSPACE-complete. From $\mathscr{A}$, we now construct a poDFA $\mathscr{D} = (Q \cup Q', \{0,1,a,b\}, \delta', s, F)$ by 'determinizing' it with help of new events that we then encode in binary. In more detail, for every state p with two transitions (p,x,r) and (p,x,q) with $p \neq q$, we replace the transition (p,x,q) with two transitions (p,x',p') and (p',x,q), where x' is a new event and p' a new state (added to Q'); see Fig. 7.1 for an illustration. In this way, we eliminate all nondeterministic transitions. The automaton is now deterministic with the set of initial states I. Let Γ be the set of all the new events. We encode these events as binary words over $\{a,b\}$. To encode $|\Gamma|$ events, it suffices to consider words of length $m = \lceil \log(|\Gamma|) \rceil$. Let enc: $\Gamma \to \{a,b\}^m$ be an arbitrary encoding (injection). We replace every transition (p,x',p'), for $x' \in \Gamma$, by the sequence of transitions $(p, \mathrm{enc}(x'), p')$, which requires to add at most $m-2$ new states to Q'. For instance, if (p,x,p') and (p,y,p'') are two transitions with $x,y \in \Gamma$, and $\mathrm{enc}(x) = aab$ and $\mathrm{enc}(y) = aba$, then (p,x,p') is replaced by transitions $(p,a,p_1),(p_1,a,p_2),(p_2,b,p')$, where p_1, p_2 are new states added to Q', and (p,y,p'') is replaced by transitions $(p,a,p_1),(p_1,b,p_3),(p_3,a,p'')$, where p_3 is a new state added to Q'; see Fig. 7.2.

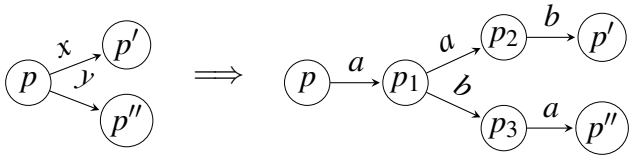

Figure 7.2: The encoding $\mathrm{enc}(x) = aab$ and $\mathrm{enc}(y) = aba$

To obtain a single initial state, assume that $I = \{q_1, \ldots, q_n\}$. Let $m = \lceil \log(n) \rceil$. We construct a binary rooted tree (with root labeled s) of depth m over $\{a, b\}$ and add it as depicted in Fig. 7.3. The number of leaves of the tree is 2^m and the number of nodes of the tree is $2^{m+1} - 1 = O(n)$. Let the leaves be denoted by $1, 2, \ldots, 2^m$. We add the transitions (i, a, q_i) for $1 \leq i \leq n$. The resulting automaton, $\mathscr{D}$, is a poDFA over the alphabet $\{0, 1, a, b\}$ with polynomially many new

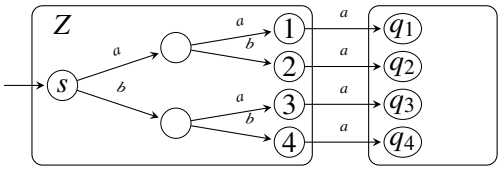

Figure 7.3: Construction of a single initial state illustrated for 4 initial states

events and states, and a single initial state s. Let P be the projection from $\{0, 1, a, b\}$ to $\{0, 1\}$, and let $P(\mathscr{D})$ denote the poNFA obtained from $\mathscr{D}$ by replacing every transition (p, a, q) by $(p, P(a), q)$. Then $\mathscr{D}$ is current-state opaque with respect to P if and only if $P(\mathscr{D})$ is current-state opaque (with respect to the identity map), which is if and only if $\mathscr{A}$ is current-state opaque (with respect to the identity map). Notice that we have not set the secret status of states of Q', and hence we have that the states of Q' are neither secret nor non-secret (alternatively, we could set their secret status according to their "parent" state from Q). □

The main papers [81, 82] on which parts the results of this chapter are based are attached in Appendices **??** and **??**, respectively. The part on opacity has not yet been published neither composed as a manuscript, which explains the presence of proofs.

8. Conclusion

We investigated the expressiveness and properties of partially ordered automata, and the complexity of related problems. Namely, we studied the state complexity of the reverse of minimal poDFAs and established a tight bound on the state complexity of the reverse of poDFAs and confluent poDFAs showing that the state complexity of the reverse of a (confluent) poDFA of the state complexity n is 2^{n-1}. The witness is ternary for poDFAs and $(n-1)$-ary for confluent poDFAs. The bound can be met neither by a binary poDFA nor by a confluent poDFA over an $(n-2)$-element alphabet. We have further provided a characterization of the tight bounds for poDFAs depending on the state complexity and the size of its alphabet.

Then we focused on the separation problem of regular languages by piecewise testable languages and showed that it is PTIME-complete. Our construction for membership is based on the non-existence of a sequence of words alternating between two languages in such a way that every word is a subsequence of the following word called a tower. We have shown that two languages are separable if and only if there is no infinite tower between them. The height of towers is closely related to the complexity of computing a separator. We investigated upper and lower bounds on the height of maximal finite towers and showed that the upper bound is polynomial in the number of states and exponential in the size of the alphabet, and that it is asymptotically tight if the size of the alphabet is fixed. If the alphabet may grow linearly with the number of states, then the lower bound on the height of towers is exponential with respect to that number. In this case, there is a gap between the lower and upper bound, and the asymptotically optimal bound remains open.

Given a DFA $\mathscr{A}$ and a $k \geq 0$, it is NL-complete to decide whether the language of $\mathscr{A}$ is piecewise testable and, for $k \geq 4$, it is CONP-complete to decide whether the language is k-piecewise testable [67]. We discussed the complexity for $k < 4$. It is known that the depth of the minimal DFA equivalent to $\mathscr{A}$ gives an upper bound on k; if $L(\mathscr{A})$ is piecewise testable, then it is k-piecewise testable for k being the depth of $\mathscr{A}$ [68]. We showed that some form of nondeterminism does not violate this upper bound result. Specifically, we defined a class of self-loop deterministic poNFAs, called ptNFAs, showed that they characterize piecewise testable languages and that their depth provides an (up to exponentially better) upper bound on k than the depth of the minimal DFA. Furthermore, if the language $L(\mathscr{A})$ is piecewise testable, we design a procedure how to express it as a Boolean combination of languages of the above form. Our idea was as follows. If the language is piecewise testable, then it is k-piecewise testable for some k, and hence there is a congruence $\sim_k$ of finite index such that $L(\mathscr{A})$ is a finite union of $\sim_k$-classes. Each class is characterized by an intersection of languages of the from $\Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$, where $n \leq k$, and their complements. To represent the $\sim_k$-classes, we made use of the $\sim_k$-canonical DFA. We identified the states of the $\sim_k$-canonical DFA whose union forms the language $L(\mathscr{A})$ and used them to construct the required Boolean combination.

On some level of abstraction, discrete event systems can be seen as finite automata with some additional properties. A fundamental property is, e. g., deadlock freeness. In some sense, partially ordered NFAs are the simplest deadlock-free models, and hence the study of their lower-

bound complexity covers most of the practical cases. The problems of interest are the questions whether the behaviors of the two systems are equivalent or in inclusion. Since the lower-bound complexity of these problems is covered by the complexity of universality, we focused on the question whether the behavior of the system is universal. Deciding universality is PSPACE-complete for NFAs and regular expressions, and the same proof shows the result for poNFAs. We improved the result by showing that it remains true even when restricting to a fixed (binary) alphabet. This is already nontrivial since the standard encodings of symbols in binary can turn self-loops into longer cycles. A lower, coNP-complete complexity bound could be obtained if we require that all self-loops are deterministic in the sense that the symbol read in the loop cannot occur in any other transition from that state. We found that such restricted poNFAs (rpoNFAs) characterize the class of $\mathscr{R}$-trivial languages, and we established the complexity of deciding whether the language of an NFA is $\mathscr{R}$-trivial. The limitation to fixed alphabets turned out to be essential even in the restricted case: deciding universality of rpoNFAs with unbounded alphabets is PSPACE-complete. Using a nontrivial extension of the proofs for rpoNFAs, we showed the same complexity results for the universality problem of ptNFAs. This strengthened the previous result and provided a new lower-bound complexity for some other problems, including inclusion, equivalence, and k-piecewise testability.

From the practical point of view, we point out that the languages of rpoNFAs are definable by deterministic (one-unambiguous) regular expressions, which makes them interesting in schema languages for XML data.

Finally, we discussed several consequences of our results in the verification of system-theoretic properties of discrete event systems, namely in the verification of detectability and opacity.

Bibliography

[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, 2015.

[3] J. Almeida, J. C. Costa, and M. Zeitoun. Pointlike sets with respect to R and J. *Journal of Pure and Applied Algebra*, 212(3):486–499, 2008.

[4] J. Almeida and M. Zeitoun. The pseudovariety J is hyperdecidable. *RAIRO – Theoretical Informatics and Applications*, 31(5):457–482, 1997.

[5] M. Arenas, S. Conca, and J. Pérez. Counting beyond a yottabyte, or how SPARQL 1.1 property paths will prevent the adoption of the standard. In *World Wide Web Conference (WWW)*, pages 629–638, 2012.

[6] S. Arora and B. Barak. *Computational Complexity – A Modern Approach*. Cambridge University Press, 2009.

[7] E. Badouel, M. Bednarczyk, A. Borzyszkowski, B. Caillaud, and P. Darondeau. Concurrent secrets. *Discrete Event Dynamic Systems*, 17(4):425–446, 2007.

[8] C. Baier and J. Katoen. *Principles of model checking*. MIT Press, 2008.

[9] P. Barceló, L. Libkin, and J. L. Reutter. Querying regular graph patterns. *Journal of the ACM*, 61(1):8:1–8:54, 2014.

[10] D. A. M. Barrington, C. Lu, P. B. Miltersen, and S. Skyum. Searching constant width mazes captures the AC^0 hierarchy. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1373 of *LNCS*, pages 73–83, 1998.

[11] G. J. Bex, W. Gelade, W. Martens, and F. Neven. Simplifying XML schema: effortless handling of nondeterministic regular expressions. In *International Conference on Management of Data (SIGMOD)*, pages 731–744, 2009.

[12] F. Blanchet-Sadri. Games, equations and the dot-depth hierarchy. *Computers & Mathematics with Applications*, 18(9):809–822, 1989.

[13] M. Bojanczyk, L. Segoufin, and H. Straubing. Piecewise testable tree languages. *Logical Methods in Computer Science*, 8(3), 2012.

[14] A. Bouajjani, A. Muscholl, and T. Touili. Permutation rewriting and algorithmic verification. *Information and Computation*, 205(2):199–224, 2007.

[15] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language XML 1.0 (fifth edition). Technical report, World Wide Web Consortium (W3C), November 2008. W3C Recommendation, http://www.w3.org/TR/2008/REC-xml-20081126/.

[16] A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Information and Computation*, 142(2):182–206, 1998.

[17] 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.

[18] J. A. Brzozowski. Canonical regular expressions and minimal state graphs for definite events. In *Symposium on Mathematical Theory of Automata*, pages 529–561, 1963.

[19] J. A. Brzozowski and F. E. Fich. Languages of *R*-trivial monoids. *Journal of Computer and System Sciences*, 20(1):32–49, 1980.

[20] J. A. Brzozowski and R. Knast. The dot-depth hierarchy of star-free languages is infinite. *Journal of Computer and System Sciences*, 16(1):37–55, 1978.

[21] J. A. Brzozowski and B. Li. Syntactic complexity of $\mathscr{R}$- and $\mathscr{J}$-trivial regular languages. *International Journal of Foundations of Computer Science*, 25(7):807–822, 2014.

[22] J. R. Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.

[23] 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.

[24] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Reasoning on regular path queries. *ACM SIGMOD Record*, 32(4):83–92, 2003.

[25] C. Câmpeanu, K. Culik II, K. Salomaa, and S. Yu. State complexity of basic operations on finite languages. In *International Workshop on Implementing Automata (WIA)*, volume 2214 of *LNCS*, pages 60–70, 2001.

[26] F. Cassez, J. Dubreil, and H. Marchand. Synthesis of opaque systems with static and dynamic masks. *Formal Methods in System Design*, 40(1):88–115, 2012.

[27] S. Cho and D. T. Huynh. Finite-automaton aperiodicity is PSPACE-complete. *Theoretical Computer Science*, 88(1):99–116, 1991.

[28] M. Chrobak. Finite automata and unary languages. *Theoretical Computer Science*, 47:149–158, 1986. Errata: Theoretical Computer Science 302 (2003) 497-498.

[29] R. S. Cohen and J. A. Brzozowski. Dot-depth of star-free events. *Journal of Computer and System Sciences*, 5(1):1–16, 1971.

[30] T. Colcombet and D. Petrişan. Automata and minimization. *ACM SIGLOG News*, 4(2):4–27, 2017.

[31] W. Craig. Linear reasoning. A new form of the Herbrand-Gentzen theorem. *Journal of Symbolic Logic*, 22(3):250–268, 1957.

[32] W. Czerwiński, C. David, K. Losemann, and W. Martens. Deciding definability by deterministic regular expressions. In *International Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*, volume 7794 of *LNCS*, pages 289–304, 2013.

[33] W. Czerwiński and S. Lasota. Regular separability of one counter automata. *Logical Methods in Computer Science*, 15(2), 2019.

[34] W. Czerwiński, W. Martens, and T. Masopust. Efficient separability of regular languages by subsequences and suffixes. In *Internation Colloquium on Automata, Languages and Programming (ICALP)*, volume 7966 of *LNCS*, pages 150–161, 2013.

[35] W. Czerwiński, W. Martens, L. van Rooijen, and M. Zeitoun. A note on decidable separability by piecewise testable languages. In *International Symposium on Fundamentals of Computation Theory FCT*, volume 9210 of *LNCS*, pages 173–185, 2015. And its extended version https://arxiv.org/abs/1410.1042 with G. Zetzsche.

[36] W. Czerwiński, W. Martens, L. van Rooijen, M. Zeitoun, and G. Zetzsche. A characterization for decidable separability by piecewise testable languages. *Discrete Mathematics & Theoretical Computer Science*, 19(4), 2017.

[37] Z. R. Dang. On the complexity of a finite automaton corresponding to a generalized regular expression. *Doklady Akademii Nauk SSSR*, 213:26–29, 1973.

[38] V. Diekert, P. Gastin, and M. Kufleitner. A survey on small fragments of first-order logic over finite words. *International Journal of Foundations of Computer Science*, 19(3):513–548, 2008.

[39] J. Dubreil, P. Darondeau, and H. Marchand. Opacity enforcing control synthesis. In *International Workshop on Discrete Event Systems (WODES)*, pages 28–35, 2008.

[40] C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98:21–52, 1961.

[41] K. Ellul, B. Krawetz, J. Shallit, and M. Wang. Regular expressions: New results and open problems. *Journal of Automata, Languages and Combinatorics*, 10(4):407–437, 2005.

[42] E. P. Friedman. The inclusion problem for simple languages. *Theoretical Computer Science*, 1(4):297–316, 1976.

[43] J. Fu, J. Heinz, and H. G. Tanner. An algebraic characterization of strictly piecewise languages. In *Theory and Applications of Models of Computation (TAMC)*, volume 6648 of *LNCS*, pages 252–263, 2011.

[44] S. Gao, C. M. Sperberg-McQueen, H. Thompson, N. Mendelsohn, D. Beech, and M. Maloney. W3C XML Schema Definition Language (XSD) 1.1 part 1: Structures. Technical report, World Wide Web Consortium, April 2009. W3C Recommendation, http://www.w3.org/TR/2009/CR-xmlschema11-1-20090430/.

[45] P. García and J. Ruiz. Learning k-testable and k-piecewise testable languages from positive data. *Grammars*, 7:125–140, 2004.

[46] P. García and E. Vidal. Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):920–925, 1990.

[47] W. Gelade and F. Neven. Succinctness of pattern-based schema languages for XML. *Journal of Computer and System Sciences*, 77(3):505–519, 2011.

[48] W. Gelade and F. Neven. Succinctness of the complement and intersection of regular expressions. *ACM Transactions on Computational Logic*, 13(1):4, 2012.

[49] C. Glaßer and H. Schmitz. Languages of dot-depth 3/2. *Theory of Computing Systems*, 42(2):256–286, 2008.

[50] J. Goubault-Larrecq and S. Schmitz. Deciding piecewise testable separability for regular tree languages. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 55 of *LIPIcs*, pages 97:1–97:15, 2016.

[51] S. Harris and A. Seaborne. SPARQL 1.1 query language. Technical report, World Wide Web Consortium (W3C), 2010. W3C Recommendation, http://www.w3.org/TR/2010/WD-sparql11-query-20100601/.

[52] G. Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, s3-2(1):326–336, 1952.

[53] 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.

[54] Š. Holub, G. Jirásková, and T. Masopust. On upper and lower bounds on the length of alternating towers. In *Mathematical Foundations of Computer Science (MFCS)*, volume 8634 of *LNCS*, pages 315–326, 2014.

[55] Š. Holub, T. Masopust, and M. Thomazo. On the height of towers of subsequences and prefixes. *Information and Computation*, 265:77–93, 2019.

[56] M. Holzer and M. Kutrib. Descriptional and computational complexity of finite automata—A survey. *Information and Computation*, 209(3):456–470, 2011.

[57] H. B. Hunt III. *On the Time and Tape Complexity of Languages*. PhD thesis, Department of Computer Science, Cornell University, Ithaca, NY, 1973.

[58] H. B. Hunt III. On the decidability of grammar problems. *Journal of the ACM*, 29(2):429–447, 1982.

[59] R. Jacob, J. Lesage, and J. Faure. Overview of discrete event systems opacity: Models, validation, and quantification. *Annual Reviews in Control*, 41:135–146, 2016.

[60] G. Jirásková and T. Masopust. On the state and computational complexity of the reverse of acyclic minimal DFAs. In *International Conference on Implementation and Application of Automata (CIAA)*, volume 7381 of *LNCS*, pages 229–239, 2012.

[61] G. Jirásková and T. Masopust. On the state complexity of the reverse of R- and J-trivial regular languages. In *International Workshop on Descriptional Complexity of Formal Systems (DCFS)*, volume 8031 of *LNCS*, pages 136–147, 2013.

[62] N. D. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 11(1):68–85, 1975.

[63] P. Karandikar, M. Kufleitner, and P. Schnoebelen. On the index of Simon's congruence for piecewise testability. *Information Processing Letters*, 115(4):515–519, 2015.

[64] P. Karandikar and P. Schnoebelen. The height of piecewise-testable languages with applications in logical complexity. In *EACSL Annual Conference on Computer Science Logic (CSL)*, volume 62 of *LIPIcs*, pages 37:1–37:22, 2016.

[65] G. Kasneci and T. Schwentick. The complexity of reasoning about pattern-based XML schemas. In *Principles of Database Systems (PODS)*, pages 155–164, 2007.

[66] O. Klíma. Piecewise testable languages via combinatorics on words. *Discrete Mathematics*, 311(20):2124–2127, 2011.

[67] O. Klíma, M. Kunc, and L. Polák. Deciding k-piecewise testability. Submitted manuscript, 2014.

[68] 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.

[69] L. Kontorovich, C. Cortes, and M. Mohri. Kernel methods for learning languages. *Theoretical Computer Science*, 405(3):223–236, 2008.

[70] D. Kozen. Lower bounds for natural proof systems. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 254–266, 1977.

[71] M. Krötzsch, T. Masopust, and M. Thomazo. On the complexity of universality for partially ordered NFAs. In *Mathematical Foundations of Computer Science (MFCS)*, volume 58 of *LIPIcs*, pages 61:1–61:14, 2016.

[72] M. Krötzsch, T. Masopust, and M. Thomazo. Complexity of universality and related problems for partially ordered NFAs. *Information and Computation*, 255:177–192, 2017.

[73] M. Kufleitner and A. Lauser. Partially ordered two-way Büchi automata. *International Journal of Foundations of Computer Science*, 22(8):1861–1876, 2011.

[74] F. Lin. Opacity of discrete event systems and its applications. *Automatica*, 47(3):496–503, 2011.

[75] 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, 2010.

[76] K. Losemann and W. Martens. The complexity of evaluating path expressions in SPARQL. In *Principles of Database Systems (PODS)*, pages 101–112, 2012.

[77] W. Martens, F. Neven, M. Niewerth, and T. Schwentick. Developing and analyzing XSDs through BonXai. *PVLDB*, 5(12):1994–1997, 2012.

[78] 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, 2015.

[79] W. Martens, F. Neven, and T. Schwentick. Complexity of decision problems for XML schemas and chain regular expressions. *SIAM Journal on Computing*, 39(4):1486–1530, 2009.

[80] W. Martens, F. Neven, T. Schwentick, and G. Bex. Expressiveness and complexity of XML Schema. *ACM Transactions on Database Systems*, 31(3):770–813, 2006.

[81] 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.

[82] T. Masopust. Complexity of deciding detectability in discrete event systems. *Automatica*, 93:257–261, 2018.

[83] T. Masopust. Separability by piecewise testable languages is PTime-complete. *Theoretical Computer Science*, 711:109–114, 2018.

[84] T. Masopust and M. Krötzsch. 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, 2018.

[85] T. Masopust and M. Krötzsch. Partially ordered automata and piecewise testability. Manuscript, 2019.

[86] T. Masopust and M. Thomazo. On boolean combinations forming piecewise testable languages. *Theoretical Computer Science*, 682:165–179, 2017.

[87] R. McNaughton and S. A. Papert. *Counter-Free Automata*. MIT Press, 1971.

[88] 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 (SWAT)*, pages 125–129, 1972.

[89] B. G. Mirkin. On dual automata. *Kibernetika*, 2:7–10, 1966. in Russian. English translation: Cybernetics 2, 6–9 (1966).

[90] A. Møller and M. I. Schwartzbach. *An introduction to XML and web technologies*. Addison-Wesley, 2006.

[91] J. Myhill. Finite automata and representation of events. Technical report, Wright Air Development Center, 1957.

[92] C. M. Ozveren and A. S. Willsky. Observability of discrete event dynamic systems. *IEEE Transactions on Automatic Control*, 35(7):797–806, 1990.

[93] C. H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[94] J. Pérez, M. Arenas, and C. Gutierrez. nSPARQL: A navigational language for RDF. *Journal of Web Semantics*, 8(4):255–270, 2010.

[95] D. Perrin and J. Pin. First-order logic and star-free sets. *Journal of Computer and System Sciences*, 32(3):393–406, 1986.

[96] D. Perrin and J.-E. Pin. *Infinite words: Automata, semigroups, logic and games*, volume 141 of *Pure and Applied Mathematics*. Academic Press, 2004.

[97] T. Place. Separating regular languages with two quantifiers alternations. In *Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 202–213, 2015.

[98] T. Place, L. van Rooijen, and M. Zeitoun. Separating regular languages by piecewise testable and unambiguous languages. In *Mathematical Foundations of Computer Science (MFCS)*, volume 8087 of *LNCS*, pages 729–740, 2013.

[99] T. Place and M. Zeitoun. Separation and the successor relation. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 30 of *LIPIcs*, pages 662–675, 2015.

[100] T. Place and M. Zeitoun. Going higher in first-order quantifier alternation hierarchies on words. *Journal of the ACM*, 66(2):12:1–12:65, 2019.

[101] P. J. Ramadge. Observability of discrete event systems. In *Conference on Decision and Control (CDC)*, pages 1108–1112, 1986.

[102] N. Rampersad, J. Shallit, and Z. Xu. The computational complexity of universality problems for prefixes, suffixes, factors, and subwords of regular languages. *Fundamenta Informatica*, 116(1–4):223–236, 2012.

[103] J. Rogers, J. Heinz, G. Bailey, M. Edlefsen, M. Visscher, D. Wellcome, and S. Wibel. On languages piecewise testable in the strict sense. In *The Mathematics of Language (MOL)*, volume 6149 of *LNAI*, pages 255–265, 2010.

[104] J. Rogers, J. Heinz, M. Fero, J. Hurst, D. Lambert, and S. Wibel. Cognitive and sub-regular complexity (fg). In *Formal Grammar*, volume 8036 of *LNCS*, pages 90–108, 2013.

[105] A. Rybalchenko and V. Sofronie-Stokkermans. Constraint solving for interpolation. *Journal of Symbolic Computation*, 45(11):1212–1233, 2010.

[106] A. Saboori. *Verification and enforcement of state-based notions of opacity in discrete event systems*. PhD thesis, University of Illinois at Urbana-Champaign, 2011.

[107] 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.

[108] H. Schmitz. *The forbidden pattern approach to concatenation hierachies*. PhD thesis, University of Würzburg, Germany, 2000.

[109] M. P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.

[110] M. P. Schützenberger. Sur le produit de concatenation non ambigu. *Semigroup Forum*, 13(1):47–75, 1976.

[111] 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.

[112] G. Sénizergues. $L(A) = L(B)$? *Electronic Notes in Theoretical Computer Science*, 9:43, 1997.

[113] S. Shu and F. Lin. Generalized detectability for discrete event systems. *Systems & Control Letters*, 60(5):310–317, 2011.

[114] S. Shu, F. Lin, and H. Ying. Detectability of discrete event systems. *IEEE Transactions on Automatic Control*, 52(12):2356–2359, 2007.

[115] I. Simon. *Hierarchies of Events with Dot-Depth One*. PhD thesis, University of Waterloo, Canada, 1972.

[116] I. Simon. Piecewise testable events. In *GI Conference on Automata Theory and Formal Languages*, pages 214–222, 1975.

[117] M. Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.

[118] 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.

[119] J. Stern. Characterizations of some classes of regular events. *Theoretical Computer Science*, 35(1985):17–42, 1985.

[120] J. Stern. Complexity of some problems from the theory of automata. *Information and Control*, 66(3):163–176, 1985.

[121] 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.

[122] H. Straubing. A generalization of the Schützenberger product of finite monoids. *Theoretical Computer Science*, 13:137–150, 1981.

[123] H. Straubing. Finite semigroup varieties of the form V*D. *Journal of Pure and Applied Algebra*, 36:53–94, 1985.

[124] D. Thérien. Classification of finite monoids: The language approach. *Theoretical Computer Science*, 14:195–208, 1981.

[125] W. Thomas. Classifying regular events in symbolic logic. *Journal of Computer and System Sciences*, 25(3):360–376, 1982.

[126] 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.

[127] L. van Rooijen. *A combinatorial approach to the separation problem for regular languages*. PhD thesis, LaBRI, University of Bordeaux, France, 2014.

[128] K. W. Wagner. Leaf language classes. In *Machines, Computations, and Universality (MCU)*, volume 3354 of *LNCS*, pages 60–81, 2004.

[129] Y.-C. Wu and S. Lafortune. Comparative analysis of related notions of opacity in centralized and coordinated architectures. *Discrete Event Dynamic Systems*, 23(3):307–339, 2013.

[130] K. Zhang. The problem of determining the weak (periodic) detectability of discrete event systems is PSPACE-complete. *Automatica*, 81:217–220, 2017.