Generalized Synchronization and Intersection Non-Emptiness of Finite-State Automata

Dissertation

dem Fachbereich IV der Universität Trier
zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)

vorgelegt von
M.Sc. Petra Wolf

November 2021

Dissertation

Generalized Synchronization and Intersection Non-Emptiness of Finite-State Automata

vom Fachbereich IV der Universität Trier
zur Verleihung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)
genehmigte Dissertation

vorgelegt von
M.Sc. Petra Wolf

Erster Gutachter: Prof. Dr. Henning Fernau,
Universität Trier

Zweiter Gutachter: Prof. Dr. Markus Holzer,
Justus-Liebig-Universität Gießen

Eröffnung des Promotionsverfahrens am 10.11.2021.
Wissenschaftliche Aussprache am 20.01.2022.
Erschienen in Trier, 2022.

Scientific Environment

The author has carried out the research reported in this thesis at Universität Trier, Fachbereich IV, Informatikwissenschaften, Professur Theoretische Informatik.

During the work on this thesis, starting from December 1'st 2018, the author was fully funded by Deutsche Forschungsgemeinschaft (DFG), project 'Modern Aspects of Complexity of Formal Languages', Project number 407073110, FE560/9-1.

The author is deeply grateful to the DFG for this support.

During 2020 and 2021 the author and her supervisor Henning Fernau from Universität Trier were part of a collaboration with Emmanuel Arrighi and Mateus de Oliveira Oliveira from the University of Bergen, Norway. This collaboration was supported by Deutscher Akademischer Austauschdienst (DAAD), Programm des Projektbezogenen Personenaustauschs (PPP), with the project number DAAD PPP 57525246 on the German side, and on the Norwegian side by IS-DAAD, project number 309319.

All collaborators of the project are deeply grateful to the DAAD for this support.

Academic CV of the Author

While the author of this thesis published her research papers under the name Petra Wolf, the full name of the author is Petra Henrike Karola Wolf.

The academic career of the author is shaped by the following milestones.[1]

Academic Career

The German grading-scale reaches from 4.0 (barely sufficient) to 1.0 (best possible grade).

Abitur High School Diploma with Award obtained on May 26'th 2011 at Technical High School Balingen, Germany, with overall grade 1.0.

B. Sc. Bachelor of Science in Computer Science obtained on December 1'st 2016 at Eberhard Karls Universität Tübingen, Germany, with overall grade 1.3.

M. Sc. Master of Science in Computer Science with Distinction obtained on October 12'th 2018 at Eberhard Karls Universität Tübingen, Germany, with overall grade 1.0.

PhD Since December 1'st 2018 PhD Student at Universität Trier, Germany, under the supervision of Prof. Dr. Henning Fernau.

[1] This is a formal requirement of Universität Trier.

Acknowledgments

First of all, I owe my deepest thanks to my supervisor, Henning Fernau. You helped me to develop my potential, gave me the freedom I needed and always trusted that I will master the assigned tasks. I am proud to say that I have not disappointed you. Especially, I am deeply grateful for you always taking time for me whenever I called you and that you always found a solution to any problems that arose. I learned a lot from you not only with respect to research but also about the scientific world. We were an awesome team in both research and student supervision! Thank you for everything and I hope we keep on collaborating :)

I am also deeply grateful to Markus Holzer for the work we did together and for agreeing to review this thesis. I am a huge fan of your work which inspired me at the beginning of my PhD and which was always a great example for me. I learned a lot from you and hope that we keep on working together.

I am also very grateful to Markus L. Schmid for taking care of me when I arrived in Trier as a new PhD student and for helping me to adapt to the new environment.

A special thanks go to the mentors who had a major impact on my journey through life that has led me to this point. The first to mention here is Christine Heil who was my class teacher in high school and the first to believe in my potential. You taught me to believe in myself and that effort is rewarded. I would not have accomplished any of my achievements without the things you taught me! The next mentor who had a big impact on my life was Britta Dorn who was the supervisor of my Bachelor thesis. With your courageous and innovative teaching concepts you got me excited about science and research. Thank you a lot! I hope you keep on igniting the interest of young people for science and math! I also want to sincerely thank Sebastian Schöner who was my tutor in plenty of theory classes during my bachelor's and master's study. You have always impressed me with your extensive knowledge and have strengthened my interest in theoretical computer science. Last but not least I owe them my deepest respect and gratitude to my mentor Klaus-Jörn Lange. Words cannot express how grateful I am to you for everything you have done for me and what a great influence you have had on

my life. You paved my way into science. Thanks to the trust that you have shown me, I was able to learn to be a teacher and a researcher.

An enrichment of my life are the co-authors who became friends along the way. Especially to mention here are Emmanuel Arrighi, Nils Morawietz, Ismaël Jecker, and Nicolas Mazzocchi. I enjoyed working with you guys so much and I hope that our friendship lasts even in case our scientific paths might diverge.

I am also very grateful to Mateus de Oliveira Oliveira for the plenty projects and great work we did together and are still doing. You are a great enrichment to any project and there is still so much I can learn from you. I am also very thankful to the German Academic Exchange Service who generously supported our collaboration with the grand DAAD PPP 57525246 / IS-DAAD 309319.

I am deeply grateful to Deutsche Forschungsgemeinschaft (DFG) for funding my position and all of my research done in the last three years with the DFG project FE560/9-1.

A source of great joy during my time in Trier were my students. Here, I want to explicitly mention Kevin Goergen, Sven Fiergolla and Patrick Neises. Our Quantum 'Selbsthilfegruppe' was a lot of fun and helped all of us to dig deeper in the field until we finally reached the frontier of new research and broke through it! Further, my special thanks go to Kevin and Sven who supported the group and me in teaching and research with their work as a student assistant.

A steady source of joy and energy were my fellow campaigners at the joined Scientists For Future group of the Universität Trier and Hochschule Trier of which I have been an active member since it was founded. Please keep on with your great work as science must be heard in society! I am particular grateful for the success of the Lectures For Future series which I organized together with Rebekka Kanesu and Tobias Kranz. Thanks a lot Rebekka and Tobias for the great work we did together and for all of your support!

I would not be the person I am today without the friends I met during my studies and who have accompanied me on my way through life to this day. I am in particular grateful to Thomas Stüber, Jonas Lingg, Marcel Früh, Yves Röhm, and Ruth Dobrota. You are truly an enrichment for my life!

Finally, I would like to thank everyone else I have worked with over the past three years.

Ultimately, none of this would have been possible without my parents to whom I owe my life and who have always supported me. Thank you for everything!

Abstract

The main focus of this work is to study the computational complexity of generalizations of the synchronization problem for deterministic finite automata (DFA). This problem asks for a given DFA, whether there exists a word w that maps each state of the automaton to one state. We call such a word w a *synchronizing word*. A synchronizing word brings a system from an unknown configuration into a well defined configuration and thereby *resets* the system.

We generalize this problem in four different ways. First, we restrict the set of potential synchronizing words to a fixed regular language associated with the synchronization *under regular constraint* problem. The motivation here is to control the structure of a synchronizing word so that, for instance, it first brings the system from an operate mode to a reset mode and then finally again into the operate mode.

The next generalization concerns the order of states in which a synchronizing word transitions the automaton. Here, a DFA A and a partial order R is given as input and the question is whether there exists a word that synchronizes A and for which the induced state order is consistent with R. Thereby, we study different ways for a word to induce an order on the state set.

Then, we change our focus from DFAs to *push-down automata* and generalize the synchronization problem to push-down automata and in the following work, to *visibly* push-down automata. Here, a synchronizing word still needs to map each state of the automaton to one state but it further needs to fulfill some constraints on the stack. We study three different types of stack constraints where after reading the synchronizing word, the stacks associated to each run in the automaton must be (1) empty, (2) identical, or (3) can be arbitrary. We observe that the synchronization problem for general push-down automata is undecidable and study restricted sub-classes of push-down automata where the problem becomes decidable. For visibly push-down automata we even obtain efficient algorithms for some settings.

The second part of this work studies the *intersection non-emptiness* problem for DFAs.

This problem is related to the problem of whether a given DFA A can be synchronized into a state q as we can see the set of words synchronizing A into q as the intersection of languages accepted by automata obtained by copying A with different initial states and q as their final state.

For the intersection non-emptiness problem, we first study the complexity of the, in general PSPACE-complete, problem restricted to subclasses of DFAs associated with the two well known Straubing-Thérien and Cohen-Brzozowski dot-depth hierarchies. Finally, we study the problem whether a given minimal DFA A can be represented as the intersection of a finite set of *smaller* DFAs such that the language $\mathcal{L}(A)$ accepted by A is equal to the intersection of the languages accepted by the smaller DFAs. There, we focus on the subclass of permutation and commutative permutation DFAs and improve known complexity bounds.

Zusammenfassung

Das Hauptaugenmerk dieser Arbeit liegt auf der Untersuchung der Komplexität von Verallgemeinerungen des Synchronisationsproblems für deterministische endliche Automaten (DEAs). Dieses Problem fragt für einen gegebenen DEA, ob es ein Wort w gibt, das jeden beliebigen Zustand des Automaten in einen einzelnen Zustand überführt. Wir nennen ein solches Wort w ein *synchronisierendes Wort*. Ein synchronisierendes Wort bringt ein System aus einer unbekannten Konfiguration in eine wohldefinierte Konfiguration und setzt es damit zurück.

Wir verallgemeinern dieses Problem auf vier verschiedene Arten. Zuerst beschränken wir die Menge potentiell synchronisierender Wörter auf eine feste reguläre Sprache, die dem so genannten *Synchronisierungsproblem unter regulären Randbedingungen* zugeordnet ist. Die Motivation dabei ist, die Struktur eines synchronisierenden Wortes zu kontrollieren, sodass dieses zum Beispiel das System zunächst aus einem Arbeitsmodus in einen Resetmodus bringt und nach vollendeter Synchronisierung wieder in den Arbeitsmodus überführt.

Die nächste Verallgemeinerung betrifft die Reihenfolge der Zustände, in denen ein synchronisierendes Wort den Automaten durchläuft. Hierbei besteht die Eingabe aus einem DEA A und einer partiellen Ordnung R und die Frage ist, ob ein Wort existiert, das A synchronisiert und dessen induzierte Zustandsordnung mit der partiellen Ordnung R übereinstimmt. Dabei untersuchen wir verschiedene Möglichkeiten, wie ein Wort eine Ordnung auf der Zustandsmenge induzieren kann.

Dann verlagern wir unseren Fokus von DEAs auf *Kellerautomaten* und verallgemeinern das Synchronisationsproblem auf Kellerautomaten, sowie in der vierten Arbeit auf *sichtbare* Kellerautomaten. Hierbei muss ein synchronisierendes Wort immer noch alle Zustände des Automatens auf einen einzelnen Zustand abbilden, aber zusätzlich müssen noch einige Bedingungen für dem Keller erfüllt sein. Wir untersuchen drei verschiedene Arten von Keller-Bedingungen, bei denen nach dem Lesen eines synchronisierenden Wortes, die jedem Lauf im Automaten zugeordneten Keller entweder (1) leer oder (2) identisch sein müssen, oder (3) beliebig sein dürfen. Wir beobachten, dass das Synchro-

nisationsproblem für allgemeine Kellerautomaten unentscheidbar ist und untersuchen eingeschränkte Unterklassen von Kellerautomaten, bei denen das Problem entscheidbar wird. Für sichtbare Kellerautomaten erhalten wir für einige Varianten sogar effiziente Algorithmen.

Der zweite Teil dieser Arbeit untersucht das *Schnitt-Leerheitsproblem* für DEAs. Dieses Problem ist verwandt mit dem Problem, ob ein gegebener DEA A in einen bestimmten Zustand q synchronisiert werden kann, da wir die Menge der Wörter, die A in q synchronisieren, ansehen können als den Schnitt der Sprachen, die von Automaten erkannt werden, die durch ein Kopieren von A mit unterschiedlichen Anfangszuständen und q als einzigem Endzustand erzeugt werden.

Bei dem Schnitt-Leerheitsproblem untersuchen wir zunächst die Komplexität des Problems, welches im Allgemeinen PSPACE-vollständig ist, auf Teilklassen von DEAs, die mit den beiden bekannten Straubing-Thérien und Cohen-Brzozowski dot-depth Hierarchien assoziiert sind. Zum Schluss untersuchen wir noch das Problem, ob ein gegebener minimaler DEA A dargestellt werden kann als Schnitt *kleinerer* DEAs, sodass die von A erkannte Sprache $\mathcal{L}(A)$ identisch ist zu dem Schnitt der von den kleineren Automaten erkannten Sprachen. Hierbei fokussieren wir uns auf die Teilklassen der Permutations DEAs und kommutativen Permutations DEAs und verbessern die bisher bekannten Komplexitätsschranken für dieses Problem.

List of Publications

In this section, a list of research papers co-authored by myself are presented. The list is segmented into papers belonging to a common line of research. Items drawn in gray were published while the author was a master's student.

Part II is based on the publications 1-6.

Generalizing Synchronization of Finite Automata

1. Henning Fernau, Vladimir V. Gusev, Stefan Hoffmann, Markus Holzer, Mikhail V. Volkov, Petra Wolf. Computational Complexity of Synchronization under Regular Constraints. In *MFCS 2019: Leibniz International Proceedings in Informatics (LIPIcs)* 138 (2019) pp. 63:1 – 63:14. DOI: 10.4230/LIPIcs.MFCS.2019.63.

2. Petra Wolf. Synchronization Under Dynamic Constraints. In *FSTTCS 2020: Leibniz International Proceedings in Informatics (LIPIcs)* 182 (2020) pp. 58:1 – 58:14. DOI: 10.4230/LIPIcs.FSTTCS.2020.58.
   Awarded with the annual (2021) best publication award of the Graduate Center of the University of Trier in the joint category Maths, Computer Science, Economics, and Social Sciences of Fachbereich IV.

3. Henning Fernau, Petra Wolf, Tomoyuki Yamakami. Synchronizing Deterministic Push-Down Automata Can Be Really Hard. In *MFCS 2020: Leibniz International Proceedings in Informatics (LIPIcs)* 170 (2020) pp. 33:1 – 33:15.
   DOI: 10.4230/LIPIcs.MFCS.2020.33.

4. Henning Fernau, Petra Wolf. Synchronization of Deterministic Visibly Push-Down Automata. In *FSTTCS 2020: Leibniz International Proceedings in Informatics (LIPIcs)* 182 (2020) pp. 45:1 – 45:15. DOI: 10.4230/LIPIcs.FSTTCS.2020.45.

Intersection of Finite Automata

5. Emmanuel Arrighi, Henning Fernau, Stefan Hoffmann, Markus Holzer, Ismaël Jecker, Mateus de Oliveira Oliveira, Petra Wolf. On the Complexity of Intersection Non-emptiness for Star-Free Language Classes. In *FSTTCS 2021: Leibniz International Proceedings in Informatics (LIPIcs)* 213 (2021) pp. 34:1 – 34:15. DOI: 10.4230/LIPIcs.FSTTCS.2021.34

6. Ismaël Jecker, Nicolas Mazzocchi, Petra Wolf. Decomposing Permutation Automata. In *CONCUR 2021: Leibniz International Proceedings in Informatics (LIPIcs)* 203 (2021) pp. 18:1 – 18:19. DOI: 10.4230/LIPIcs.CONCUR.2021.18.

Finding Positive Instances of Computational Problems in Regular Languages

7. Demen Güler, Andreas Krebs, Klaus-Jörn Lange, Petra Wolf. Deciding Regular Intersection Emptiness of Complete Problems for PSPACE and the Polynomial Hierarchy. In *LATA 2018: Lecture Notes in Computer Science (LNCS)* 10792 (2018) pp. 156 – 168. DOI: 10.1007/978-3-319-77313-1_12.

8. Petra Wolf. On the Decidability of Finding a Positive ILP-Instance in a Regular Set of ILP-Instances. In *DCFS 2019: Lecture Notes in Computer Science (LNCS)* 11612 (2019) pp. 272 – 284. DOI: 10.1007/978-3-030-23247-4_21.
Submitted for journal publication.

9. Petra Wolf. From Decidability to Undecidability by Considering Regular Sets of Instances. In *Theoretical Computer Science* 899 (2022) pp. 25 – 38.
DOI: 10.1016/j.tcs.2021.11.006.

10. Volker Diekert, Henning Fernau, Petra Wolf. Properties of Graphs Specified by a Regular Language. In *DLT 2021: Lecture Notes in Computer Science (LNCS)* 12811 (2021) pp. 117 – 129. DOI: 10.1007/978-3-030-81508-0_10.
Submitted for journal publication.

Parameterized Complexity of Ordering Related Problems and Diversity of Solutions

11. Emmanuel Arrighi, Henning Fernau, Mateus de Oliveira Oliveira, Petra Wolf. Width Notions for Ordering-Related Problems. In *FSTTCS 2020: Leibniz In-*

ternational Proceedings in Informatics (LIPIcs) 182 (2020) pp. 9:1 – 9:18.
DOI: 10.4230/LIPIcs.FSTTCS.2020.9.

12. Emmanuel Arrighi, Henning Fernau, Mateus de Oliveira Oliveira, Petra Wolf. Order Reconfiguration Under Width Constraints. In *MFCS 2021: Leibniz International Proceedings in Informatics (LIPIcs)* 202 (2021) pp. 8:1 – 8:15.
DOI: 10.4230/LIPIcs.MFCS.2021.8.

13. Emmanuel Arrighi, Henning Fernau, Daniel Lokshtanov, Mateus de Oliveira Oliveira, Petra Wolf. Diversity in Kemeny Rank Aggregation: A Parameterized Approach. In *IJCAI 2021: Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence* (2021) Main Track pp. 10 – 16.
DOI: 10.24963/ijcai.2021/2.

14. Synchronization and Diversity of Solutions. Emmanuel Arrighi, Henning Fernau, Mateus de Oliveira Oliveira, Petra Wolf. *Submitted manuscript* (2021).

Games Played on Graphs

15. Ronald de Haan, Petra Wolf. Restricted Power - Computational Complexity Results for Strategic Defense Games. In *FUN 2018: Leibniz International Proceedings in Informatics (LIPIcs)* 100 (2018) pp. 17:1 – 17:14.
DOI: 10.4230/LIPIcs.FUN.2018.17.

16. Nils Morawietz, Petra Wolf. A Timecop's Chase Around the Table. In *MFCS 2021: Leibniz International Proceedings in Informatics (LIPIcs)* 202 (2021) pp. 77:1 – 77:18. DOI: 10.4230/LIPIcs.MFCS.2021.77

Data Compression

17. Sven Fiergolla, Petra Wolf. Improving Run Length Encoding by Preprocessing. In *DCC 2021: Institute of Electrical and Electronics Engineers (IEEE)* (2021) pp. 341. DOI: 10.1109/DCC50243.2021.00051.

Learning Finite Automata from Samples

18. Learning from Positive and Negative Examples: Dichotomies and Parameterized Algorithms. Jonas Lingg, Mateus de Oliveira Oliveira, Petra Wolf.
Submitted manuscript (2021).

Contents

Part I

Background

Chapter 1

Introduction

Synchronization is an important concept for many applied areas: parallel and distributed programming, system and protocol testing, information coding, robotics, etc. At least some aspects of synchronization are captured by the notion of a synchronizing automaton; for instance, synchronizing automata adequately model situations in which one has to direct a certain system to a particular state without a priori knowledge of its current state. We only refer to some survey papers [Sandberg, 2005, Volkov, 2008], as well as to Chapter 13 in [Kohavi and Jha, 2009], that also report on some of these applications. We will name in this introduction several automata models and complexity classes that are only formally defined in the next two chapters. We recommend the reader to jump to the Chapter 2 for definitions of automaton models and to Chapter 3 for definitions of complexity classes, if necessary.

An automaton is called synchronizing if there exists a word that brings it to a known state independently of the starting state. This concept is quite natural and has been investigated intensively in the last six decades. It is related to the arguably most famous open combinatorial question in automata theory, formulated by Černý and Starke in [Černý, 1964, Starke, 1966]. The Černý conjecture states that every n-state synchronizing automaton can be synchronized by a word of length smaller or equal $(n-1)^2$. Although this bound was proven for several classes of finite-state automata, the general case is still widely open. The currently best upper bound on this length is cubic, and only very little progress has been made, basically improving on the multiplicative constant factor in front of the cubic term, see [Shitov, 2019, Szykuła, 2018].

Due to the importance of this notion of synchronizing words, quite a large number of generalizations and modifications have been considered in the literature. We only mention four of these in the following. Instead of synchronizing the whole set of states, one

could be interested in synchronizing only a subset of states. This and related questions were first considered by Rystsov in [Rystsov, 1983]. Instead of considering deterministic finite automata (DFAs), one could alternatively study the notion of synchronizability for nondeterministic finite automata [Gazdag et al., 2009, Martyugin, 2014]. The notion of synchronizability naturally transfers to partially defined transition functions where a synchronizing automata avoiding undefined transitions is called *carefully synchronizing*, see [de Bondt et al., 2019, Martyugin, 2009, Martyugin, 2014]. To capture more adaptive variants of synchronizing words, synchronizing strategies have been introduced in [Larsen et al., 2014]. Recall that the question of synchronizability (without length bounds) is solvable in polynomial time for complete DFAs [Starke, 1966, Volkov, 2008]. However, in all of the mentioned generalizations, this synchronizability question becomes even PSPACE-complete. This general tendency can also be observed in the generalization that we introduce in this thesis.

The classical *synchronization problem* asks, for a given deterministic finite automaton (DFA), if there exists a *synchronizing word*, i.e., an input that brings all states of the automaton to a single state. The idea of bringing an automaton to a well-defined state by reading a word, starting from any state, can be seen as implementing a software reset. This is why a synchronizing word is also sometimes called a *reset word*. Restricting the length of a potential synchronizing word of some DFA by an integer parameter in the input also yields a harder problem, namely the NP-complete short synchronizing word problem [Rystsov, 1980, Eppstein, 1990]. In the following, we will introduce several generalizations of the synchronization problems and its mentioned variants. Our main objective thereby is the study of the computational complexity of the introduced problems. Thereby, we are not only obtaining classical complexity results but also parameterized complexity results concerning parameterized versions of the problem. A first study of the parameterized complexity of the short synchronizing word problem was performed by Fernau et al. in [Fernau et al., 2015] which entailed a series of research papers on the parameterized complexity of synchronization problems [Vorel and Roman, 2015, Fernau, 2019, Bruchertseifer and Fernau, 2020, Bruchertseifer and Fernau, 2021].

Regular Constraints

In the theory of synchronizing automata, one normally allows the reset word to be an arbitrary word over the input language of the corresponding automaton. In reality, however, available instructions might be subject to certain restrictions; for instance, it is quite natural to assume that a directing instruction should always start and end with a specific command that first switches the automaton to a 'directive' mode and then returns the automaton to its usual mode. In its simplest form, the switching

between 'normal mode' and 'directive' (synchronization) mode can be modeled as ab^*a. These constraints are defined by some (fixed) finite automaton (called the constrained automaton) describing a regular language R, and the question is, given some DFA A, if A has some synchronizing word from R. This notion explicitly appeared in [Gusev, 2012] as an auxiliary tool.

Constraints of this form, i.e., which can be modeled by a fixed regular language on the side, restricting the set of potential synchronizing words for an input automaton are called *regular constraints* and the topic of the first research paper in Chapter 7 of Part II. There, a complete analysis of the complexity of the synchronization problem under regular constraints for constraint partial automata with two states and at most three letters is performed. Building up on this work, a line of research restricting the class of regular languages from which the constraint language is chosen was pursued by Stefan Hoffmann, e.g., in [Hoffmann, 2020a, Hoffmann, 2021a, Hoffmann, 2021b, Hoffmann, 2020b, Hoffmann, 2020c].

Dynamic Constraints

Apart from software reset words, one of the oldest applications of the intensively studied topic of synchronizing automata is the problem of designing parts orienters, which are robots or machines that get an object in an (due to a lack of expensive sensors) unknown orientation and transform it into a defined orientation [Ananichev and Volkov, 2004]. In his pioneering work, Natarajan [Natarajan, 1986] modeled the parts orienters as deterministic complete automata (see Figure 1.1) where a state corresponds to a possible orientation of a part and a transition of some letter a from state q corresponds to applying the modifier corresponding to a to a part in orientation q. He proved that the synchronization problem is solvable in polynomial time for – what is later called – the class of *orientable automata* [Ryzhikov, 2019] if the cyclic order respected by the automaton is part of the input.

Many different classes of automata have since been studied regarding their synchronization behavior. We refer to [Volkov, 2008, Béal and Perrin, 2016, Truthe and Volkov, 2019] for an overview. The original motivation of designing a parts orienter was revisited in [Türker and Yenigün, 2015] where Türker and Yenigün modeled the design of an assembly line, which again brings a part from an unknown orientation into a known orientation, where different modifiers have different costs. For example, a robot arm is much more expensive than a simple obstacle wall. Therefor, they enhance the alphabet with a cost function and ask whether the alphabet of the automaton can be narrowed such that the cost of the alphabet is below a given threshold and the automaton is still synchro-

Consider a two dimensional belt (infinite in the third dimension) as shown in Fig.1. The belt

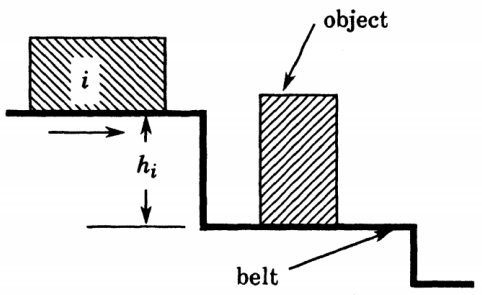


Fig. 1. The belt paradigm

moves from left to right and has vertical jumps at various points. The two dimensional convex polygonal object (infinite in the third dimension) is placed on the left end of the belt in some orientation. As the object traverses a jump, it could be reoriented depending on the height of the jump and the orientation in which the object approaches the jump.

Figure 1.1: Illustration of parts orienters as jumps on an assembly line in the introductory work of Natarajan [Natarajan, 1986]. Here, a state corresponds to a possible orientation of a part and a transition of some letter a from state q corresponds to applying the modifier corresponding to a to a part in orientation q.

nizable. They showed that this problem is NP-complete for DFAs and PSPACE-complete for partial DFAs with respect to careful synchronization.

What has not been considered so far is that different modifiers can have different impact on the parts and as we do not know the current orientation we might want to restrict the chronology of applied modifiers. For example, if the part is a box with a fold-out lid, turning it upside-down will cause the lid to open. In order to close the lid one might need another modifier such as a low bar which brushes the lid and closes it again, as illustrated in Figure 1.2. To specify that a parts orienter should deliver the box facing upward with a closed lid one needs to encode something like: "When the box is in the state *facing down*, it later needs to be in the state *lid closed*". But this does not stop us from opening the lid again, so we need to be more precise and encode: "After **the last time** the box was in the state *facing down*, it needs to visit the state *lid closed* at least once".

Conditions of this type form *positive* constraints for the future as they demand for the future visit of some state. But also the opposite situation of forbidding the future visit of a state is quite natural as our next example shows. Let us again picture the box with

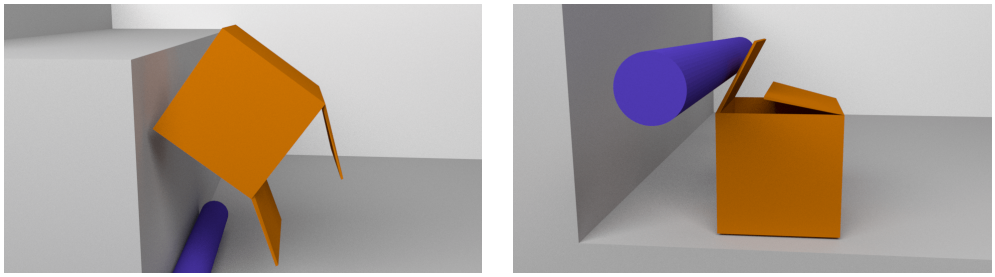


Figure 1.2: Illustration of a box with lids that open when the box is rotated together with a modifier closing the lids.

the lid in mind, but this time the box initially contains some water. We would like to have the box in a specific orientation with the lid open but the water should not be shed during orientating. We have a modifier that opens the lid and a modifier which rotates the box. Clearly we do not want the box to face downwards after the lid has been opened. So, we encode: "As soon as the state *lid open* has been reached, the state *facing downwards* should never be entered again".

Modeling conditions of the above discussed form that restrict the dynamic behavior of a synchronizing word are the focus of the second research paper presented in Chapter 8 of Part II[1]. There, we implement those conditions in our model of a parts orienter by enhancing a given DFA with a relation R. We will then consider different ways of how a synchronizing word implies an order on the states and ask whether there exists a synchronizing word whose implied state order agrees with the input relation R.

Push-Down Automata

The idea of bringing an automaton to a well-defined state by reading a word, starting from any state is obviously not restricted to finite automata. For the next two research papers in Chapter 9 and 10 of Part II we move away from deterministic finite automata to more general deterministic push-down automata.

What should a synchronizing word mean in this context? Mikami and Yamakami first studied in [Mikami and Yamakami, 2020] three different models, depending on requirements of the stack contents when a word w drives the automaton into a synchronizing state, irrespectively of the state where processing w started: we could require at the end (a) that the stack is always empty; or (b) that the stack contents is always the same (but not necessarily empty); or (c) that the stack contents is completely irrelevant upon

[1]This research paper was awarded with the 2021 publication prize of the Graduate Center of the University of Trier, Faculty IV. A video (in German) of the presentation of the results to the public during the award can be found under https://youtu.be/kmDPUmUt-R4.

entering the synchronizing state. For those three models of synchronizing push-down automata, they demonstrated in [Mikami and Yamakami, 2020] some upper and lower bounds on the maximum length of the shortest synchronizing word dependent on the stack height. Here, we study these three models from a complexity-theoretic perspective. However, as it is shown in Chapter 9, synchronizability becomes undecidable when asking about synchronizability in any of the stack models. Clearly, by restricting the length of a potential synchronizing word of some push-down automaton by an integer parameter (given in unary), we can observe that the corresponding synchronization problems all become NP-complete, as the hardness is trivially inherited from what is known about DFA synchronizability. Therefore, those length-bounded problem variants are not further considered here. Yet, it remains interesting to observe that with DFAs, introducing a length bound on the synchronizing word means an increase of complexity, while for deterministic push-down automata, this introduction means dropping from undecidability close to feasibility.

Beside general deterministic push-down automata, these stack model variants of synchronization are studied for subclasses of deterministic push-down automata such as deterministic counter automata, deterministic (partially) blind counter automata and finite-turn variants of thereof.

The DFA synchronization problem has been generalized in the literature to other automata models including infinite-state systems with infinite branching such as weighted and timed automata [Doyen et al., 2014, Shirmohammadi, 2014] or register automata [Babari et al., 2016]. For instance, register automata are infinite state systems where a state consists of a control state and register contents. A synchronizing word for a register automaton brings all (infinitely many) states to the same state (and same register content). The synchronization problem for deterministic register automata (DRA) is PSPACE-complete and NLOG-complete for DRAs with only one register.

Finally, we want to mention that the term *synchronization of push-down automata* has already some occurrences in the literature, i.e., in [Caucal, 2006, Arenas et al., 2011], but here the term *synchronization* refers to some relation of the input symbols to the stack behavior [Caucal, 2006] or to reading different words in parallel [Arenas et al., 2011] and is not to be confused with our notion of synchronizing states.

Building up on our introductory work, the concept of synchronizing push-down automata has recently been generalized to *adaptive* synchronization of push-down automata [Balasubramanian and Thejaswini, 2021]. There, the current state of the system is not known but there is an *observer* who can give inputs to the machine and observe the implied modification of the stack. Then, the question is whether the system can be brought from

any state to a predetermined state by giving inputs in an *adaptive* manner, i.e., the next input letter may depend on the reaction of the system to the previous input. Balasubramanian and Thejaswini also considered a subset variant of the adaptive synchronization problem where the question is whether a subset of states can be adaptively synchronized. They further distinguished between deterministic and non-deterministic push-down automata and showed that the adaptive synchronization and subset synchronization problem are 2-EXPTIME-complete in the non-deterministic setting and EXPTIME-complete in the deterministic setting.

Visibly Push-Down Automata

Another automaton model, where the state set is enhanced with a possibly infinite memory structure, is the class of *nested word automata* (NWAs were introduced in [Alur and Madhusudan, 2009]), where an input word is enhanced with a matching relation determining at which pair of positions in a word a symbol is pushed to and popped from the stack. The class of languages accepted by NWAs is identical to the class of *visibly push-down languages* (VPL) accepted by *visibly push-down automata* (VPDA) and form a proper subclass of the deterministic context-free languages. VPDAs have first been studied by Mehlhorn [Mehlhorn, 1980] under the name *input-driven pushdown automata* and became quite popular more recently due to the work by Alur and Madhusudan [Alur and Madhusudan, 2004], showing that VPLs share several nice properties with regular languages. For more on VPLs we refer to the survey [Okhotin and Salomaa, 2014]. In [Chistikov et al., 2019], the synchronization problem for NWAs was studied. There, the concept of synchronization was generalized to bringing all states to one single state such that for all runs the stack is empty (or in its start configuration) after reading the synchronizing word. In this setting, the synchronization problem is solvable in polynomial time (again indicating similarities of VPLs with regular languages), while the short synchronizing word problem (with length bound given in binary) is PSPACE-complete; the question of synchronizing from or into a subset is EXPTIME-complete. Also, matching exponential upper bounds on the length of a synchronizing word are given.

In the research paper presented in Chapter 10 we study the synchronization problem for real-time (no ϵ-transitions) deterministic visibly push-down automata (DVPDA) and several subclasses thereof, like real-time deterministic very visibly push-down automata (DVVPDA for short; this model was introduced in [Ludwig, 2019]), real-time deterministic visibly counter automata (DVCA for short; this model appeared among others in [Bárány et al., 2006, Srba, 2009, Bollig, 2016, Hahn et al., 2015, Krebs et al., 2015a, Krebs et al., 2015b]) and finite turn variants thereof. We want to point out that, despite the equivalence of the accepted language class, the automata models of nested

word automata and visibly push-down automata still differ and the results from [Chistikov et al., 2019] do not immediately transfer to VPDAs, as for NWAs an input word is equipped with a matching relation, which VPDAs lack of. In general, the complexity of the synchronization problem can differ for different automata models accepting the same language class. For instance, in contrast to the polynomial-time solvable synchronization problem for DFAs, the generalized synchronization problem for finite automata with one ambiguous transition is PSPACE-complete, as well as the problem of carefully synchronizing a DFA with one undefined transition [Martyugin, 2012].

We will not only consider the synchronization model introduced in [Chistikov et al., 2019], where reading a synchronizing word results in an empty stack on all runs; but we will also consider a synchronization model where not only the final state on every run must be the same but also the stack contents need to be identical, as well as a model where only the states needs to be synchronized and the stack contents might be arbitrary. These three models of synchronization have been introduced in [Mikami and Yamakami, 2020], where length bounds on a synchronizing word for general DPDAs have been studied dependent on the stack height. The complexity of these three concepts of synchronization for general DPDAs are considered in [Fernau et al., 2020] (Chapter 9), where it is shown that synchronizability is undecidable for general DPDAs and deterministic counter automata (DCA). It becomes decidable for deterministic partially blind counter automata and is PSPACE-complete for some types of finite turn DPDAs, while it is still undecidable for other types of finite turn DPDAs.

In contrast, for DVPDAs and considered subclasses hereof, the synchronization problem for all three stack models, with restricted or unrestricted number of turns, is in EXPTIME and hence decidable. For DVVPDAs and DVCAs, the synchronization problems for all three stack models (with unbounded number of turns) are even in P. Like the synchronization problem for NWAs in the empty stack model considered in [Chistikov et al., 2019], we observe that the synchronization problem for DVPDAs in the empty stack model is solvable in polynomial time, whereas synchronization of DVPDAs in the same and arbitrary stack models is at least PSPACE-hard. If the number of turns caused by a synchronizing word on each run is restricted, the synchronization problem becomes PSPACE-hard for all considered automata models for $n > 0$ and is only in P for $n = 0$ in the empty stack model. We will further introduce variants of synchronization problems distinguishing the same and arbitrary stack models by showing complementary complexities in these models. For problems considered in [Fernau et al., 2020], these two stack models have always shared their complexity status.

Connection between Synchronizing Automata and Intersection of Automata

The Intersection Non-emptiness problem for finite automata is one of the most fundamental and well studied problems in the interplay between algorithms, complexity theory, and automata theory, see [Kozen, 1977, Kasai and Iwata, 1985, Lange and Rossmanith, 1992, Wareham, 2000, Karakostas et al., 2003, Wehar, 2014, Fernau and Krebs, 2017, Wehar, 2017]. Given a list $A_1, A_2, \ldots, A_n$ of finite automata over a common alphabet Σ, the goal is to determine whether there is a string $w \in \Sigma^*$ that is accepted by each of the automata in the list. Or in other words, the question is whether $\mathcal{L}(A_1) \cap \mathcal{L}(A_2) \cap \cdots \cap \mathcal{L}(A_n) \neq \emptyset$.

For unbounded alphabets, the Intersection Non-emptiness problem is PSPACE-complete by a reduction from the general word problem of deterministic polynomial-space bounded Turing machines by Kozen [Kozen, 1977]. The problem remains PSPACE-complete for binary alphabets as we observe in Chapter 11 by re-analyzing a result in [Krötzsch et al., 2017, Theorem 3] showing that the closely related Non-universality problem for NFAs is PSPACE-complete for binary alphabets. For unary input alphabets, the Intersection Non-emptiness problem becomes NP-complete (a result implicitly contained already in [Stockmeyer and Meyer, 1973]). The problem is also NP-complete when the input automata all accept finite languages [Rampersad and Shallit, 2010].

The Intersection Non-emptiness problem is closely related to synchronizing automata. On one hand, the question whether an automaton A on n states can be synchronized into some state q can be modeled as the intersection non-emptiness problem of n automata, where each automaton is a copy of A with q as its single final state and some different start state each. Then, the intersection of those n automata accept exactly those words that synchronize A into the state set $\{q\}$. On the other hand, the Intersection Non-emptiness problem for DFAs can be reduced to the Careful Synchronization problem which asks, given a *partial* DFA A, does there exists a word w that synchronizes A without taking any undefined transition while reading w on any path starting from any state of A. Martyugin showed in [Martyugin, 2014] that the Careful Synchronization problem is PSPACE-complete even if the input automaton has only one undefined transition by a reduction from the Intersection Non-emptiness problem. Due to this natural relation, the remaining two research papers in Chapter 11 and 12 are concerned with the Intersection Non-emptiness problem.

For a fixed number of input DFAs, the Intersection Non-emptiness problem is solvable in polynomial time, i.e., in time $\mathcal{O}(n^k)$ where n is the size of the biggest automaton and k is the fixed number of automata [Rabin and Scott, 1959].

Recently, it was shown in [de Oliveira Oliveira and Wehar, 2020] that solving Intersection Non-emptiness for a fixed number of k DFAs in time slightly faster than $\mathcal{O}(n^k)$ would imply the existence of deterministic sub-exponential time algorithms for the simulation of nondeterministic linear space bounded computations strengthening the existing conditional lower bounds for Intersection Non-emptiness [Kasai and Iwata, 1985, Fernau and Krebs, 2017, Karakostas et al., 2003, Wehar, 2014, Wehar, 2017]. Fine grained complexity results drawing a connection between the Intersection Non-emptiness problem for two and three DFA's over a binary alphabet and the problems Triangle Finding and 3Sum were obtained in [de Oliveira Oliveira and Wehar, 2020]. Further results on the parameterized complexity of the Intersection Non-emptiness problem can be found in [Kasai and Iwata, 1985, Lange and Rossmanith, 1992, Wareham, 2000]. Combining a finite list of k automata with one push-down automaton gives even finer results on the running time $\mathcal{O}(n^k)$. Namely it was shown in [Swernofsky and Wehar, 2015] that the complexity class P can be characterized in this way. More precisely, Swernofsky and Wehar showed that there exist constants c_1 and c_2 such that for every k, intersection non-emptiness for one PDA and k DFA's is solvable in time $\mathcal{O}(n^{c_1 k})$, but is not solvable in time $\mathcal{O}(n^{c_2 k})$. If we drop the push-down automaton in the intersection, then the complexity class NLOG can be characterized via the Intersection Non-emptiness problem for deterministic finite automata over a binary alphabet [Wehar, 2014].

Intersection Non-Emptiness of Star-Free Languages

In Chapter 11, we analyze the complexity of the Intersection Non-emptiness problem under the assumption that the languages accepted by the input automata belong to a given level of the Straubing-Thérien hierarchy [Place and Zeitoun, 2019, Straubing, 1981, Straubing, 1985, Thérien, 1981] or to some level of the Cohen-Brzozowski dot-depth hierarchy [Brzozowski, 1976, Cohen and Brzozowski, 1971, Place and Zeitoun, 2019]. Somehow, these languages are severely restricted, in the sense that both hierarchies, which are infinite, are entirely contained in the class of star-free languages, a class of languages that can be represented by expressions that use union, concatenation, and complementation, but *no* Kleene star operation [Brzozowski, 1976, Brzozowski and Knast, 1978, Place and Zeitoun, 2019]. Yet, languages belonging to fixed levels of either hierarchy may already be very difficult to characterize, in the sense that the very problem of deciding whether the language accepted by a given finite automaton belongs to a given full level or half-level k of either hierarchy is open, except for a few values of k [Almeida and Klíma, 2010, Glaßer and Schmitz, 2001, Glaßer and Schmitz, 2000, Place and Zeitoun, 2019]. It is worth noting that while the problem of determining

whether a given automaton accepts a language in a certain level of either the dot-depth or of the Straubing-Thérien hierarchy is computationally hard (at least PSPACE-hard for non-deterministic finite automata [III and Rosenkrantz, 1978]), automata accepting languages in lower levels of these hierarchies arise naturally in a variety of applications such as model checking where the Intersection Non-emptiness problem is of fundamental relevance [Abdulla, 2012, Bouajjani et al., 2000, Bouajjani et al., 2007].

An interesting question to consider is how the complexity of the Intersection Non-emptiness problem changes as we move up in the levels of the Straubing-Thérien hierarchy or in the levels of the dot-depth hierarchy. In particular, does the complexity of this problem changes gradually, as we increase the complexity of the input languages? In the work presented in Chapter 11, we show that this is actually not the case, and that the complexity landscape for the Intersection Non-emptiness problem is already determined by the very first levels of either hierarchy. The first main result states that the Intersection Non-emptiness problem for NFAs and DFAs accepting languages from the level $1/2$ of the Straubing-Thérien hierarchy are NLOG-complete and LOG-complete, respectively, under AC^0 reductions. Additionally, this completeness result holds even in the case of unary languages. To prove hardness for NLOG and LOG, respectively, we will use a simple reduction from the reachability problem for DAGs and for directed trees, respectively. Nevertheless, the proof of containment in NLOG and in LOG, respectively, will require a new insight that may be of independent interest. More precisely, we will use a characterization of languages in the level $1/2$ of the Straubing-Thérien hierarchy as shuffle ideals to show that the Intersection Non-emptiness problem can be reduced to Concatenation Non-Emptiness. This allows us to decide Intersection Non-emptiness by analyzing each finite automaton given at the input individually. It is worth mentioning that this result is optimal in the sense that the problem becomes NP-hard even if we allow a single DFA to accept a language from $\mathcal{L}_1$, and require all the others to accept languages from $\mathcal{L}_{1/2}$.

Subsequently, we analyze the complexity of Intersection Non-emptiness when all input automata are assumed to accept languages from one of the levels of $\mathcal{B}_0$ or $\mathcal{B}_{1/2}$ of the dot-depth hierarchy, or from the levels $\mathcal{L}_1$ or $\mathcal{L}_{3/2}$ of the Straubing-Thérien hierarchy. It is worth noting that NP-hardness follows straightforwardly from the fact that Intersection Non-emptiness for DFAs accepting finite languages is already NP-hard [Rampersad and Shallit, 2010]. Containment in NP, on the other hand, is a more delicate issue, and here the representation of the input automaton plays an important role. A characterization of languages in $\mathcal{L}_{3/2}$ in terms of languages accepted by partially ordered NFAs [Schwentick et al., 2001] is crucial for us, combined with the fact that Intersection Non-emptiness when the input is given by such automata is NP-complete [Masopust and Thomazo, 2015]. Intuitively, the proof in [Masopust and Thomazo, 2015]

follows by showing that the minimum length of a word in the intersection of languages in the level 3/2 of the Straubing-Thérien hierarchy is bounded by a polynomial on the sizes of the minimum partially ordered NFAs accepting these languages. To prove that Intersection Non-emptiness is in NP when the input automata are given as DFAs, we prove a new result establishing that the number of Myhill-Nerode equivalence classes in a language in the level $\mathcal{L}_{3/2}$ is at least as large as the number of states in a minimum partially ordered automaton representing the same language.

Interestingly, we show that the proof technique used to prove this last result does not generalize to the context of NFAs. To prove this, we carefully design a sequence $(L_n)_{n\in\mathbb{N}_{\geq 1}}$ of languages over a binary alphabet such that for every $n \in \mathbb{N}_{\geq 1}$, the language L_n can be accepted by an NFA of size n, but any partially ordered NFA accepting L_n has size $2^{\Omega(\sqrt{n})}$. This lower bound is ensured by the fact that the syntactic monoid of L_n has many $\mathcal{J}$-factors. Our construction is inspired by a technique introduced by Klein and Zimmermann, in a completely different context, to prove lower bounds on the amount of look-ahead necessary to win infinite games with delay [Klein and Zimmermann, 2016]. To the best of our knowledge, this is the first exponential separation between the state complexity of general NFAs and that of partially ordered NFAs. While this result does not exclude the possibility that Intersection Non-emptiness for languages in $\mathcal{L}_{3/2}$ represented by general NFAs is in NP, it gives some indication that proving such a containment requires substantially new techniques.

Finally, we show that Intersection Non-emptiness for both DFAs and for NFAs is already PSPACE-complete if all accepting languages are from the level $\mathcal{B}_1$ of the dot-depth hierarchy or from the level $\mathcal{L}_2$ of the Straubing-Thérien hierarchy. We can adapt Kozen's classical PSPACE-completeness proof by using the complement of languages introduced in [Masopust and Krötzsch, 2021] in the study of partially ordered automata. Since the languages in [Masopust and Krötzsch, 2021] belong to $\mathcal{L}_{3/2}$, their complement belong to $\mathcal{L}_2$ (and to $\mathcal{B}_1$), and therefore, the proof follows.

Decomposing Automata

Not only can we represent the synchronization problem as a intersection non-emptiness problem of DFAs, but we can also represent some minimal DFAs themselves as the intersection of finitely many *smaller* DFAs. In this sense, we *decompose* the bigger automaton into a set of smaller automata, such that their intersection accepts the same language as the original automaton. If we consider only *minimal* DFAs, whether or not such a decomposition is possible is a property determined by the accepted language.

Compositionality is a fundamental notion in numerous fields of computer science [de Roever et al., 1998]. This principle can be summarized as follows: Every system should be designed by composing simple parts such that the meaning of the system can be deduced from the meaning of its parts, and how they are combined. For instance, this is a crucial aspect of modern software engineering: a program split into simple modules will be quicker to compile and easier to maintain. The use of compositionality is also essential in theoretical computer science: it is used to avoid the *state explosion* issues that usually happen when combining parallel processes together, and also to overcome the *scalability* issues of problems with a high theoretical complexity. In the work presented in Chapter 12, we study compositionality in the setting of formal languages: we show how to make languages simpler by decomposing them into *intersections* of bigger languages accepted by smaller automata. This is motivated by *model-checking* problems. For instance, the LTL model-checking problem asks, given a linear temporal logic formula φ and a finite state machine M, whether every execution of M satisfies φ. This problem is decidable, but has a high theoretical complexity (PSPACE) with respect to the size of φ [Baier and Katoen, 2008]. If φ is too long, it cannot be checked efficiently. This is where compositionality comes into play: if we can decompose the specification language into an intersection of simple languages, that is, decompose φ into a conjunction $\varphi = \varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_k$ of small specifications, it is sufficient to check whether all the φ_i are satisfied separately.

In the work in Chapter 12, we focus our study on *permutation* DFAs, which are DFAs whose transition monoids are groups: each letter induces a one-to-one map from the state set into itself. These DFAs are also called *reversible* DFAs [Kunc and Okhotin, 2013, Pin, 1992]. Reversibility is stronger than determinism: this powerful property allows to deterministically navigate *back and forth* between the steps of a computation. This is particularly relevant in the study of the physics of computation, since irreversibility causes energy dissipation [Landauer, 1961]. Remark that in the setting of DFAs, this power results in a loss of expressiveness: contrary to more powerful models (for instance Turing machines), reversible DFAs are less expressive than general DFAs.

The problem of interest for this work is the so called Decomp problem, that asks for a given DFA A whether it is composite, i.e., whether there exists a finite set of DFAs $\{A_1, A_2, \ldots, A_n\}$ such that $\mathcal{L}(A) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2) \cap \cdots \cap \mathcal{L}(A_n)$ and each of the automata in the intersection has strictly less states than A. The DFA Decomp problem was first introduced by Kupferman and Moscheiff [Kupferman and Mosheiff, 2015]. They proved that it is decidable in EXPSPACE, but left open the exact complexity: the best known lower bound is hardness for NLOG. They gave more efficient algorithms for restricted domains: a PSPACE algorithm for *permutation* DFAs, and a polynomial-time algorithm for *normal* permutation DFAs, a class of DFAs that contains all *commutative*

permutation DFAs. Recently, the DECOMP problem was proved to be decidable in logarithmic space for DFAs with a singleton alphabet [Jecker et al., 2020]. The trade-off between number and size of factors was studied in [Netser, 2018], where automata showing extreme behavior are presented, i.e., DFAs that can either be decomposed into a large number of small factors, or a small number of large factors.

In the research paper presented in Chapter 12, we expand the domain of instances over which the DECOMP problem is tractable. There, we focus on permutation DFAs and propose new techniques that improve the known complexities.

Structure of this Work

This work is structured as followed. In Chapter 2 we formally introduce language and automata classes discussed in this thesis. We begin with the class of *regular languages* and introduce several equivalent models recognizing or describing regular languages. Then, we introduce subclasses of regular languages such as star-free languages and two infinite hierarchies within the class of star-free languages, namely the Straubing-Thérien hierarchy and Cohen-Brzozowski dot-depth hierarchy. Next, we focus on the bigger class of *context-free languages* containing the regular languages as a strict subclass. Here, again we consider several language classes between regular and context-free such as *deterministic context-free languages* or *visibly push-down languages*.

In Chapter 3 we then switch from formal language classes to complexity classes. After introducing the necessary machine models to define the considered complexity classes, we define several classical complexity classes. The second part of this chapter is then concerned with the quite modern concept of *parameterized complexity theory* where we define the parameterized complexity classes appearing in this thesis.

Now, we are ready to state in Chapter 4 the exact results of each research paper presented in Part II. Therefore, we give the definition of the introduced and considered problems for each paper and state in a compact way the obtained results of the paper as well as the potentially remaining open problems in this direction.

Finally, we discuss new potential directions of future research for the topics introduced in this thesis in Chapter 5. These contain among other a recently discovered new parameterized complexity classes based on a parameterized synchronization problem as a defining complete problem, as well as ideas on how to generalize the concept of synchronizing automata to *quantum finite automata*. Both directions are new, promising and fast growing fields of recent research in the theory of formal languages and complexity theory.

Part II consists of the six research papers introduced above. Each one is a slight modification of the published work in the sense that missing proofs are reintegrated. Usually, technical proofs, further examples, and algorithms given in pseudo code are removed from page limited conference papers after the review progress and before publication. Those parts are reintegrated in the versions of published papers presented in Part II.

Chapter 2

Formal Language Theory

2.1 Regular Languages

We refer to the empty word as ε. For a finite alphabet Σ we denote with Σ^* the set of all words over Σ and with $\Sigma^+ = \Sigma\Sigma^*$ the set of all non-empty words. Let $\mathbb{N} = \{0, 1, \dots\}$ be the set of natural numbers. For $i \in \mathbb{N}$ we set $[i] = \{1, 2, \dots, i\}$ with the special case $[0] = \emptyset$. For a set S, we denote with $|S|$ the *cardinality* of S. For $w \in \Sigma^*$ we denote with $|w|$ the length of w, with $w[i]$ for $i \in [|w|]$ the i'th symbol of w, and with $w[i..j]$ for $i, j \in [|w|]$ the factor $w[i]w[i+1]\dots w[j]$ of w. We call $w[1..i]$ a prefix and $w[i..|w|]$ a suffix of w. We make the convention $w[0] = \varepsilon$. The reversal of w is denoted by w^R, i.e., for $|w| = n$, $w^R = w[n]w[n-1]\dots w[1]$.

For words $u, v \in \Sigma^*$ with $u = u_1u_2\dots u_n$ and $v = v_1v_2\dots v_m$, we denote with $u \cdot v$ the *concatenation* $u \cdot v = u_1u_2\dots u_nv_1v_2\dots v_m$ of u and v. We might also write uv instead of $u \cdot v$. For two languages $L_1, L_2 \subseteq \Sigma^*$, their concatenation is defined as $L_1 \cdot L_2 = \{uv \mid u \in L_1, v \in L_2\}$, again writing L_1L_2 if convenient. We abbreviate the concatenation of a language L with itself as L^2 and inductively $L^n = L \cdot L^{n-1}$ for $n \in \mathbb{N}$. Then, the *Kleene closure* of a language L is defined as $L^* = \bigcup_{n\in\mathbb{N}} L^n$ with the convention $L^0 = \{\varepsilon\}$. Note that L^* always contains the empty word ε.

2.1.1 Models of Representation

A direct way to represent a regular language is by a *regular expression.*

Definition 1 (Regular Expression)**.** Let Σ be a finite alphabet. We say that R is a regular expression if R is

- the empty set $\emptyset$,
- the empty word ε,
- a letter $\sigma \in \Sigma$,
- the alternation $(R_1|R_2)$ of two regular expressions R_1 and R_2,
- the concatenation $(R_1 \cdot R_2)$ of two regular expressions R_1 and R_2,
- the Kleene closure (R_1^*) of a regular expression R_1.

The language $\mathcal{L}(R)$ described by a regular expression R is inductively defined as

- if $R = \emptyset$, then $\mathcal{L}(R) = \emptyset$,
- if $R = \varepsilon$, then $\mathcal{L}(R) = \{\varepsilon\}$,
- if $R = \sigma$, for some $\sigma \in \Sigma$, then $\mathcal{L}(R) = \{\sigma\}$,
- if $R = (R_1|R_2)$, then $\mathcal{L}(R) = \mathcal{L}(R_1) \cup \mathcal{L}(R_2)$,
- if $R = (R_1 \cdot R_2)$, then $\mathcal{L}(R) = \mathcal{L}(R_1) \cdot \mathcal{L}(R_2)$,
- if $R = (R_1^*)$, then $\mathcal{L}(R) = \mathcal{L}(R_1)^*$.

As the alternation and concatenation is associative, we might omit brackets. For a finite set $S \subseteq \Sigma^*$ we might abbreviate an alternation over all elements of S by writing S instead. We now give some examples of regular languages which can easily be represented by regular expressions.

Example 1. The set of all words of even length can be expressed as $((\Sigma\Sigma)^*)$. A language over the alphabet $\Sigma = \{a, b, c\}$ consisting of all words which either start and end with an a or contain no a at all can be expressed as $((a\Sigma^*a)|((b|c)^*))$. For a binary alphabet $\Sigma = \{0, 1\}$, a language of special interest is *parity* consisting of all words with an *odd* number of 1's which can be expressed as $(((0^*)1(0^*)1)^*)(0^*)1(0^*)$.

If the context is clear we might identify a regular language L with a regular expression R describing L and omit brackets if there is no fear of confusion. Other important computational models that characterize the class of regular languages are non-deterministic and deterministic finite automata.

Definition 2 (NFA)**.** We call $A = (Q, \Sigma, \delta, q_0, F)$ a *non-deterministic finite automaton* (NFA for short) if Q is a finite set of states, Σ is a finite input alphabet, δ is a transition function $Q \times \Sigma \to 2^Q$, q_0 is the initial state and $F \subseteq Q$ is the set of final states.

The transition function δ is generalized to sets of states $Q' \subseteq Q$ by $\delta(Q', \sigma) = \bigcup_{q \in Q'} \delta(q, \sigma)$. We further generalize δ to words by $\delta(q, w) = \delta(\delta(q, w[1]), w[2..|w|])$ for $w \in \Sigma^*$. A word $w \in \Sigma^*$ is accepted by A if $\delta(q_0, w) \cap F \neq \emptyset$ and the language accepted by A is defined by $\mathcal{L}(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset\}$. We extend δ to sets of letters $\Sigma' \subseteq \Sigma$, letting $\delta(q, \Sigma') = \bigcup_{\sigma \in \Sigma'} \delta(q, \sigma)$.

Definition 3 (DFA)**.** Let $A = (Q, \Sigma, \delta, q_0, F)$ be a NFA. We further call A a *deterministic finite automaton*, (DFA for short) if for each pair of states $q \in Q$ and letter $\sigma \in \Sigma$ the image of δ is a singleton set, i.e., $|\delta(q, \sigma)| = 1$.

Thereby, every DFA is also an NFA. For simplicity, we might define δ as $\delta \colon Q \times \Sigma \to Q$ for DFAs, i.e., δ maps to single states instead of sets of states. By definition, δ is a total function. Later, we might also consider the case that δ is a *partial* function. In this case, we call A a *partial* DFA. As the synchronization problem does not consider start or final states, we might omit q_0 and F in the description of A. If we want to emphasize this, we might call a DFA $A = (Q, \Sigma, \delta)$ without start and final states a deterministic *semi-automaton* (DSA) for short. If it is not of relevance to distinguish between non-deterministic or deterministic finite automata, we might simply write finite automaton.

One can also consider regular languages as algebraic objects. In this perception, Σ^* is the free monoid generated by Σ. Here, we will consider three important monoids associated with a regular language and a DFA. We begin with the equivalence relation introduced by Myhill and Nerode [Myhill, 1957, Nerode, 1958] which describes the sets of words leading from the start state to each state in the minimal DFA representing a regular language.

Definition 4 (Myhill-Nerode Equivalence Relation)**.** Let $L \subseteq \Sigma^*$ be a regular language. We define the *Myhill-Nerode* equivalence relation $\sim_L$ induced by L as

$$\forall u, v \in \Sigma^* \colon \ (u \sim_L v \Leftrightarrow \forall w \in \Sigma^* \colon \ (uw \in L \Leftrightarrow vw \in L)) \, .$$

The equivalence classes $\Sigma^*/\sim_L$ describe the state set of the minimal (with respect to number of states) DFA accepting L. Namely, for each states q in the minimal DFA $A = (Q, \Sigma, \delta, q_0 F)$ accepting L, there exists one equivalence class $[w]_{\sim_L}$ such that $\delta(q_0, w) = q$, and even stronger $[w]_{\sim_L} = \{w \in \Sigma^* \mid \delta(q_0, w) = q\}$. Thereby, the index of the equivalence relation $\sim_L$ is the number of states in the minimal DFA A accepting L and more general, a language L is regular if and only if the index of $\sim_L$ is finite.

While $\sim_L$ identifies words behaving the same way in the minimal automaton with respect to the initial state q_0, the next relation refines $\Sigma^*/\sim_L$ to identify words behaving the

same way regardless from which state they are read.

Definition 5 (Syntactic Congruence Relation)**.** Let $L \subseteq \Sigma^*$ be a regular language. We define the *syntactic congruence relation* $\equiv_L$ induced by L as

$$\forall u, v \in \Sigma^* \colon (u \equiv_L v \leftrightarrow \forall x, y \in \Sigma^* \colon (xuy \in L \leftrightarrow xvy \in L)) .$$

Here, words need to behave the same way with respect to membership in L under simultaneous concatenation of words from both sides. The syntactic congruence relation defines the syntactic monoid $M_L = (\Sigma^*/\equiv_L, \cdot, [\varepsilon])$ with the concatenation operation and the equivalence class of the empty word ε as neutral element. We can use the syntactic monoid to define a regular language in the following way. Let $h_L \colon \Sigma^* \to \Sigma^*/\equiv_L$ be the syntactic homomorphism that maps each word $w \in \Sigma^*$ to its equivalence class under $\equiv_L$. Then, L is recognized by M_L if there exists a set $S \subseteq \Sigma^*/\equiv_L$ such that $h_L^{-1}(S) = L$. Note that the syntactic monoid $\Sigma^*/\equiv_L$ can be much bigger than $\Sigma^*/\sim_L$.

The next monoid considers the transitions induced by words on the state set of a DFA.

Definition 6 (Transition Monoid)**.** Let $A = (Q, \Sigma, \delta)$ be a DFA. For $\sigma \in \Sigma$ let $f_\sigma \colon Q \to Q$ be the transition function of the letter σ defined as $f_\sigma(q) = \delta(q, \sigma)$. Let T_A be the semi-group generated by the set $\{f_\sigma \mid \sigma \in \Sigma\}$ with the function composition operator. Together with the identity function $f_\varepsilon \colon Q \to Q$ mapping each state to itself, T_A forms a monoid and is called the *transition monoid* of A.

For the minimal DFA A, the transition monoid of A is isomorphic to the syntactic monoid of $\mathcal{L}(A)$ [Straubing, 1994, p. 56].

We are now ready to illustrate the concept of a DFA, the Myhill-Nerode equivalence relation, the syntactic monoid and the transition monoid on a concrete example.

Example 2. Let us consider the language L described by the regular expression $(b^*ab^*a(a|b))^*$. A minimal DFA A accepting L is depicted in Figure 2.1. The set of equivalence classes of the Myhill-Nerode equivalence relations of L can be represented by $\{[\varepsilon], [a], [aa]\}$. A set of representatives of the syntactic monoid M_L of L together with the corresponding transitions in the transition monoid T_A of A is listed in Table 2.1. A good tool to compute and illustrate the syntactic monoid of a regular language can be found in [Paperman, 2021].

Theorem 1 (Regular Languages [Hopcroft and Ullman, 1979])**.** *The class of regular languages is equal to the set of languages recognized by NFAs, or DFAs, or described by regular expressions. Further, a language L is regular if and only if its syntactic monoid M_L is finite.*

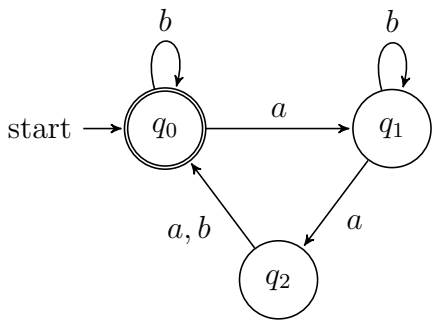


Figure 2.1: Minimal DFA A accepting the language L described by the regular expression $(b^*ab^*a(a|b))^*$.

M_L	T_A		
ε	q_0	q_1	q_2
a	q_1	q_2	q_0
aa	q_2	q_0	q_1
b	q_0	q_1	q_0
ba	q_1	q_2	q_1
baa	q_2	q_0	q_2
bab	q_1	q_0	q_1
$baba$	q_2	q_1	q_2
$babaa$	q_0	q_2	q_0
$baab$	q_0	q_0	q_0
$baaba$	q_1	q_1	q_1
$baabaa$	q_2	q_2	q_2

M_L	T_A		
ab	q_1	q_0	q_0
aba	q_2	q_1	q_1
$abaa$	q_0	q_2	q_2
$abab$	q_0	q_1	q_1
$ababa$	q_1	q_2	q_2
$ababaa$	q_2	q_0	q_0
aab	q_0	q_0	q_1
$aaba$	q_1	q_1	q_2
$aabaa$	q_2	q_2	q_0
$aabab$	q_1	q_1	q_0
$aababa$	q_2	q_2	q_1
$aababaa$	q_0	q_0	q_2

Table 2.1: For the in Figure 2.1 introduced language L and minimal automaton A, the table shows a choice of representatives of the congruence classes of the syntactic monoid M_L of L on the left with the corresponding transitions of the transition monoid T_A of A on the right. For readability, the table is split in half.

There are even more models of computations in which the class of regular languages can be characterized. For instance, the set of languages which can be described as *Monadic Second Order* sentences over words using the $+1$ predicate is exactly the class of regular languages. With respect to grammars, the class of regular languages consists of all languages which can be generated by grammars, where all production rules are of the form $A \to \varepsilon$, $A \to a$ or $a \to aB$ where A, B are non-terminals and a is a terminal. Further, regarding circuit classes, the regular languages are contained in the circuit class NC^1 which consists of polynomial size circuits of logarithmic depth and bounded fan-in, see [Straubing, 1994] for details.

2.1.2 Subclasses of Regular Languages

Despite its apparent simplicity, the class of regular languages contains quite a number of interesting strict subclasses such as star-free languages, and the subclasses of commutative and permutation regular languages.

Star-Free Languages

We begin with the famous subclass of *star-free* languages, which can be classified by plenty of models of computation, as it is the case for regular languages. The class of star-free languages itself contains two well known infinite hierarchies of language classes, which we will discuss in the next chapter. We first give a definition for star-free languages by a certain type of regular expressions avoiding the Kleene star, which also gave the class its name.

Definition 7 (Star-Free Expression)**.** Let Σ be a finite alphabet. We say that R is a *star-free* expression if R is $\emptyset$, ε, σ for $\sigma \in \Sigma$, $(R_1|R_2)$ or $(R_1 \cdot R_2)$ for two star-free expressions R_1 and R_2, or $(\overline{R_1})$, the complement of a star-free expression R_1. The language $\mathcal{L}(R)$ described by a star-free expression R is defined analogously to regular expressions except for the last case which is replaced by the following. If $R = (\overline{R_1})$, then $\mathcal{L}(R) = \overline{\mathcal{L}(R_1)} = \Sigma^* \setminus \mathcal{L}(R_1)$.

Star-free expressions forbid the use of the Kleene star but allow for using the complement operator on languages. Nonetheless, several languages of which their 'natural' description uses the Kleene star can still be represented by a star-free expression. For instance, the language Σ^* can be represented by the star-free expression $\overline{\emptyset}$, and the language $1(0^*)1$ over the alphabet $\{0, 1\}$ can be represented by the star-free expression $1(\overline{\overline{\emptyset}1\overline{\emptyset}})1$. On the contrary, it is not possible to represent languages with inherently nested stars by a star-free expression or stars reaching over several consecutive letters. For instance, the language $(1(0^*)1)^*$ is not star-free.

The class of star-free languages can also be characterized through properties of the syntactic monoid of the language. Namely, a regular language L is star-free if and only if its syntactic monoid M_L is aperiodic.

Definition 8 (Aperiodic Monoids)**.** Let $M = (S, \circ, 0)$ be a monoid. For an element $x \in S$, we inductively define $x^n \circ x = x^{n+1}$ for $n \in \mathbb{N}$. Then, M is called aperiodic if and only if for each $x \in S$ there exists an $n \in \mathbb{N}$ such that $x^n = x^{n+1}$.

This means that for every element x, the cyclic subgroup of the submonoid generated

by x is trivial [Rozenberg and Salomaa, 1997, p. 698].

Theorem 2 ([Schützenberger, 1965])**.** *A language is star-free if and only if its syntactic monoid is finite and aperiodic.*

Therefore, the class of languages which can be described by *star-free* expressions is identical to the class of regular languages with an aperiodic syntactic monoid. The class of star-free languages can further be characterized as the class of languages which can be described by *First Order* sentences over words using the $<$ predicate, i.e., the class FO[$<$], and by the circuit class AC^0 [Straubing, 1994].

Straubing-Therien and Dot-Depth Hierarchy

While the star-free languages are quite restrictive, they still contain two interesting infinite hierarchies of classes of languages, which we introduce next. Namely, the Straubing-Thérien [Straubing, 1981, Thérien, 1981] and Cohen-Brzozowski's dot-depth hierarchy [Cohen and Brzozowski, 1971].

Definition 9 (Polynomial Closure)**.** Let $L \subseteq \Sigma^*$ be a language. We say that L is a *marked product* of the languages $L_0, L_1, \ldots, L_k$ if $L = L_0\sigma_1 L_1 \cdots \sigma_k L_k$, where the σ_i's are letters from Σ and $L_i \subseteq \Sigma^*$ for $0 \leq i \leq k$. For a class of languages $\mathcal{M}$, the *polynomial closure* of $\mathcal{M}$ is the set of languages that are finite unions of marked product of languages from $\mathcal{M}$.

Definition 10 (Concatenation Hierarchy)**.** Let $\mathcal{M}$ be a class of languages. We define the *concatenation hierarchy* of basis $\mathcal{M}$ as follows.

- For the basis level 0, let $\mathcal{M}_0 = \mathcal{M}$.
- For level $n + 1/2$, $\mathcal{M}_{n+1/2}$ is the polynomial closure of level n, and
- for level $n + 1$, $\mathcal{M}_{n+1}$ is the Boolean closure of level $n + 1/2$.

Definition 11 (Straubing-Thérien Hierarchy)**.** The Straubing-Thérien Hierarchy is the concatenation hierarchy of the basis $\{\emptyset, \Sigma^*\}$. Their levels are denoted by $\mathcal{L}_n$, respectively $\mathcal{L}_{n+1/2}$.

Definition 12 (Dot-Depth Hierarchy)**.** Let $\mathcal{F}$ be the class of all finite and co-finite languages. Then, the dot-depth hierarchy is the concatenation hierarchy of basis $\mathcal{F}$. The levels of the dot-depth hierarchy are denoted by $\mathcal{B}_n$, respectively $\mathcal{B}_{n+1/2}$.

Apart from level $\mathcal{B}_0$, the dot-depth hierarchy coincides with the concatenation hierarchy starting with the language class $\{\emptyset, \{\varepsilon\}, \Sigma^+, \Sigma^*\}$. Clearly, within each hierarchy each level $n \geq 0$ (respectively half level) is contained in the next half level (respectively full level),

$$\mathcal{B}_{n+1/2} \subseteq \mathcal{B}_{n+1} \subseteq \mathcal{B}_{n+3/2} \quad \text{and} \quad \mathcal{L}_{n+1/2} \subseteq \mathcal{L}_{n+1} \subseteq \mathcal{L}_{n+3/2}.$$

Concerning the respectively other hierarchy, each level is contained between two full levels of the other hierarchy. More formally, we have the following relations [Pin, 1998].

Theorem 3. *For $n \geq 1$ it holds that*

$$\mathcal{L}_{n-1/2} \subseteq \mathcal{B}_{n-1/2} \subseteq \mathcal{L}_{n+1/2} \quad \textit{and} \quad \mathcal{L}_n \subseteq \mathcal{B}_n \subseteq \mathcal{L}_{n+1}.$$

In particular, $\mathcal{L}_0 \subseteq \mathcal{B}_0$, $\mathcal{B}_0 \subseteq \mathcal{B}_{1/2}$, and $\mathcal{L}_0 \subseteq \mathcal{L}_{1/2}$.

Both hierarchies are infinite for alphabets of at least two letters and completely exhaust the class of star-free languages, meaning that the infinite union of each hierarchy yields the class of star-free languages. For singleton letter alphabets, both hierarchies collapse to $\mathcal{B}_0$ and $\mathcal{L}_1$, respectively. Next, we give some more intuition on the first levels of both hierarchies.

Straubing-Thérien hierarchy: A language of Σ^* is of level 0 if and only if it is empty or equal to Σ^*. The languages of level $1/2$ are exactly those languages that are a finite (possibly empty) union of languages of the form $\Sigma^*\sigma_1\Sigma^*\sigma_2\cdots\sigma_k\Sigma^*$, where the σ_i's are letters from Σ. The languages of level 1 are finite Boolean combinations of languages of the form $\Sigma^*\sigma_1\Sigma^*\sigma_2\cdots\sigma_k\Sigma^*$, where the σ_i's are letters. These languages are also called *piecewise* testable languages. In particular, all finite and co-finite languages are of level 1. Finally, the languages of level $3/2$ of Σ^* are the finite unions of languages of the form $\Sigma_0^*\sigma_1\Sigma_1^*\sigma_2\cdots\sigma_k\Sigma_k^*$, where the σ_i's are letters from Σ and the Σ_i are subsets of Σ.

Dot-depth hierarchy: A language of Σ^* is of dot-depth (level) 0 if and only if it is finite or co-finite. The languages of dot-depth $1/2$ are exactly those languages that are a finite union of languages of the form $u_0\Sigma^*u_1\Sigma^*u_2\cdots u_{k-1}\Sigma^*u_k$, where $k \geq 0$ and the u_i's are words from Σ^*. The languages of dot-depth 1 are finite Boolean combinations of languages of the form $u_0\Sigma^*u_1\Sigma^*u_2\cdots u_{k-1}\Sigma^*u_k$, where $k \geq 0$ and the u_i's are words from Σ^*.

Given a language represented by an NFA, it is not an easy task to decide the level of this language in the hierarchies.

Theorem 4 ([Arrighi et al., 2021a]). *For each level $\mathcal{L}$ of the Straubing-Thérien or the dot-depth hierarchies, the problem whether the language accepted by a given NFA is contained in $\mathcal{L}$ is PSPACE-hard, even when restricted to binary alphabets.*

For some of the lower levels of the hierarchies, we also have containment in PSPACE but in general the exact complexity is unclear. Already for the third level of the Straubing-Thérien hierarchy $\mathcal{L}_3$ it is a famous open question whether the problem is decidable at all. See [Masopust, 2018, Place and Zeitoun, 2019] for an overview on the decidability status of these questions. Checking containment for $\mathcal{L}_0$ up to $\mathcal{L}_2$ and $\mathcal{B}_0$ up to $\mathcal{B}_1$ for DFAs can be done in NLOG and is also complete for this class by ideas similar to the ones used in [Cho and Huynh, 1991].

A class of automata that is closely related to the lower levels of these hierarchies is the class of *partially ordered automata*.

Definition 13 (Partially Ordered Automaton). Let $A = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton. We say that a state q is *reachable* from a state p, written $p \leq q$, if there is a word $w \in \Sigma^*$ such that $q \in \delta(p, w)$. We call A *partially ordered* if $\leq$ is a partial order.

We refer to a partially ordered NFA (DFA, respectively) as poNFA (poDFA, respectively). Partially ordered NFAs (with multiple initial states) characterize the class $\mathcal{L}_{3/2}$ [Schwentick et al., 2001], while partially ordered DFAs characterize the class of $\mathcal{R}$-trivial languages [Brzozowski and Fich, 1980], a class that is strictly in between $\mathcal{L}_1$ and $\mathcal{L}_{3/2}$.

Commutative Regular Languages

Next, we consider the subclass of *commutative* regular languages. For this class of languages, membership of a word w in the language is completely determined by the Parikh-image of w (for Parikh-image, see [Parikh, 1966]) or in other words, the order of the letters in a word does not matter. Therefore, commutative languages generalize unary languages.

Definition 14 (Commutative DFA). Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA. We call A a *commutative DFA* if for every state $q \in Q$ and every pair of words $u, v \in \Sigma^*$ it holds that $\delta(q, uv) = \delta(q, vu)$. We call a regular language L a *commutative* regular language, if it can be accepted by a commutative DFA.

Clearly, the class of commutative regular languages forms a strict subclass of the class of regular languages. Considering commutative star-free languages, we get an interesting

picture of the Straubing-Thérien and dot-depth hierarchies restricted to commutative languages [Arrighi et al., 2021a, Hoffmann, 2021c].

Theorem 5 ([Arrighi et al., 2021a])**.** *For* commutative star-free languages *the levels* $\mathcal{L}_n$ *of the Straubing-Thérien and* $\mathcal{B}_n$ *of the dot-depth hierarchy coincide for all full and half levels, except for* $\mathcal{L}_0$ *and* $\mathcal{B}_0$*. Moreover, the hierarchy collapses at level one.*

Permutation Regular Languages

Next, we introduce the class of *permutation regular languages* introduced in [Thierrin, 1968]. This class is characterized by DFAs with the constraint that no state has two incoming transitions with the same letter.

Definition 15 (Permutation DFA)**.** Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA. We call A a *permutation DFA* if the transition monoid of A is a group, i.e., each letter induces a total bijective map from the state set to itself. We call a regular language L a *permutation* regular language if it can be accepted by a permutation DFA.

Permutation DFAs are also called *reversible* DFAs [Kunc and Okhotin, 2013, Pin, 1992]. Reversibility is stronger than determinism as it allows to deterministically navigate *back and forth* between the steps of a computation. Remark that in the setting of DFAs, this power results in a loss of expressiveness: contrary to more powerful models (for instance Turing machines), reversible DFAs are less expressive than general DFAs.

2.2 Context-Free Languages

Next, we shift our focus from regular languages forming the third level of the Chomsky hierarchy to *context-free* languages forming the second level [Chomsky, 1956].

2.2.1 (Non-Deterministic) Context-Free Languages

We begin with the general class of context-free languages characterized by (non-deterministic) push-down automata.

Definition 16 (Push-down-Automata)**.** We call $M = (Q, \Sigma, \Gamma, \delta, q_0, \bot, F)$ a *push-down automaton* (PDA for short) if Q is a finite set of states; the finite sets Σ and Γ are the input and stack alphabet, respectively; δ is a finite transition function $Q \times \Sigma \times \Gamma \to$

$2^{Q\times\Gamma^*}$[1]; q_0 is the initial state; $\perp \in \Gamma$ is the stack bottom symbol which is only allowed as the first (lowest) symbol in the stack, i.e., if $\delta(q, a, \gamma) \ni (q', \gamma')$ and γ' contains $\perp$, then $\perp$ only occurs in γ' as its first letter and moreover, $\gamma = \perp$; and F is the set of final states.

We will only consider real-time push-down automata and forbid ϵ-transitions, as can be seen in the definition. Note that the bottom symbol can be removed, but then the computation gets stuck. If we want to emphasize, that a PDA M is using non-determinism, i.e., that for some combination of state q, input letter σ and stack letter γ the image $\delta(q, \sigma, \gamma)$ is of size bigger than one, we might call M a *non-deterministic* push-down automaton, NPDA for short.

Following [Chistikov et al., 2019], a *configuration* of M is a tuple $(q, \upsilon) \in Q \times \Gamma^*$. For a letter $\sigma \in \Sigma$, states $q, q' \in Q$, and a stack content υ with $|\upsilon| = n$ we write $(q, \upsilon) \xrightarrow{\sigma} (q', \upsilon[1..(n-1)]\gamma)$ if $\delta(q, \sigma, \upsilon[n]) \ni (q', \gamma)$. This means that the top of the stack υ is the right end of υ. We also denote with $\longrightarrow$ the reflexive transitive closure of the union of $\xrightarrow{\sigma}$ over all letters in Σ. The input words on top of $\longrightarrow$ are concatenated accordingly, so that $\longrightarrow = \bigcup_{w\in\Sigma^*} \xrightarrow{w}$. The language $\mathcal{L}(M)$ accepted by a PDA M is $\mathcal{L}(M) = \{w \in \Sigma^* \mid (q_0, \perp) \xrightarrow{w} (q_f, \gamma), q_f \in F\}$. We call a sequence of configurations $(q, \perp) \xrightarrow{w} (q', \gamma)$ a *run* induced by w, starting in q, and ending in q'. We might also call q' the *final state* of the run.

The class of context-free languages can further be characterized by context-free grammars. Those are grammars where the left side of each production rule consists of only one variable.

While the class of regular languages is closed under intersection, union, complement, concatenation and Kleene star, the context-free languages are not closed under intersection and complement. Further, while for regular languages the intersection non-emptiness and the equivalence problem are decidable, those problems are no longer decidable for context-free languages [Schöning, 1997].

2.2.2 Deterministic Context-Free Languages

Next, consider the class of *deterministic* context-free languages characterized by *deterministic* push-down automata.

Definition 17 (Deterministic Push-down-Automaton)**.** Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$

[1]With a finite transition function, we mean that δ only maps to finite sets in $2^{Q\times\Gamma^*}$.

be a PDA. If for each triple $(q, \sigma, \gamma) \in Q \times \Sigma \times \Gamma$ it holds that $|\delta(q, \sigma, \gamma)| \leq 1$, we call M a *deterministic* push-down automaton, DPDA for short.

Again, we might identify the singleton sets in the image of δ with its respective elements. We call a context-free language that can be recognized by a deterministic push-down automaton a *deterministic context-free language*.

In contrast to other models of computation, such as finite automata or Turing machines, the class of languages recognizable by *deterministic* push-down automata is a strict subclass of the class of languages recognizable by *non-deterministic* push-down automata. For instance, consider the language $L = \{w\$w^R \mid w \in \{0,1\}^*\}$ over the alphabet $\Sigma = \{0, 1, \$\}$, where w^R denotes the *reversal* of w. The language L is a *deterministic* context-free language as it can be accepted by a deterministic push-down automaton which acts in two phases. In the first phase, it pushes the symbols of w onto the stack until it reads the single \$ symbol in the word. This causes the DPDA to enter the second phase in which the input sequence is compared with the stack content and the word is accepted if and only if this check of phase 2 is successful, i.e., the remainder of the input word is identical with the stack content read from top to bottom. The crucial observation here is that there is a special symbol \$ which indicates the end of the sub-word w and the beginning of the sub-word w^R which causes the push-down automaton to switch from phase 1 to phase 2. Further, it is ensured that each word in the language contains only one symbol \$. If we drop this guarantee, i.e., either allow \$ to appear in w or remove the special symbol \$ at all, we arrive at a language which is still context-free but inherently needs non-determinism in an accepting push-down automaton and is hence no longer contained in the class of deterministic context-free languages.

While the class of general context-free languages is closed under union, concatenation and Kleene star, but not under intersection and complement, the picture for *deterministic* context-free languages is quite different. In contrast, this class is closed under complement but *not* closed under union, intersection, concatenation or Kleene star. Like context-free languages, the intersection non-emptiness problem for deterministic context-free languages is undecidable [Schöning, 1997].

2.2.3 Counter Automata

Not all context-free languages need the full power of the stack to store arbitrary words. For the class of *one-counter languages* it is sufficient to use the stack as a unary *counter* where the actual number on the stack is not directly accessible, but can be increased, decreased and tested for having value zero. An example for such a language is the famous

language $\{a^n b^n \mid n \in \mathbb{N}\}$.

Definition 18 (One-Counter Automaton)**.** We call $M = (Q, \Sigma, \delta, q_0, F)$ a *one-counter-automaton* if the following holds. The set Q is a finite set of states, Σ is a finite input alphabet, $\delta\colon Q \times (\Sigma \cup \{\varepsilon\}) \times \{-1, 0, 1\} \to 2^{Q \times \{-1,0-1\}}$ is a transition function[2], q_0 is the initial state and F is a set of final states.

Note that M does not have a stack alphabet and further allows for ε-transitions. Let $\operatorname{sgn}\colon \mathbb{Z} \to \{-1, 0, +1\}$ be a function returning the sign of an integer. A *configuration* of M is a tuple $(p, x) \in Q \times \mathbb{Z}$. For two configurations $(p, x), (q, y) \in Q \times \mathbb{Z}$ and a letter $\sigma \in (\Sigma \cup \{\varepsilon\})$, we write $(p, x) \xrightarrow{\sigma} (q, y)$ if $\delta(p, \sigma, s) \ni (q, \Delta)$ with $s, \Delta \in \{-1, 0, 1\}$, $s = \operatorname{sgn}(x)$, and $y = x + \Delta$. As for push-down automata, we denote with $\longrightarrow$ the reflexive transitive closure of the union of $\xrightarrow{\sigma}$ over all letters in Σ. The input words on top of $\longrightarrow$ are concatenated accordingly, so that $\longrightarrow = \bigcup_{w \in \Sigma^*} \xrightarrow{w}$. We say that a word w is accepted by a one-counter automaton M if there exists a sequence of configurations $(q_0, 0) \xrightarrow{w} (q, 0)$ with $q \in F$. Hence, w is accepted if it leads M into an accepting state such that the counter has value zero at the end of the computation. The language $\mathcal{L}(M)$ accepted by a one-counter automaton M is $\mathcal{L}(M) = \{w \in \Sigma^* \mid (q_0, 0) \xrightarrow{w} (q_f, 0), q_f \in F\}$. If the context is clear, we may refer to a one-counter automaton simply as a counter automaton. Again, if for all tuples $(q, \sigma, s) \in Q \times (\Sigma \cup \{\varepsilon\}) \times \{-1, 0, 1\}$ it holds that $|\delta(q, \sigma, s)| \leq 1$ and further, if $|\delta(q, \varepsilon, s)| = 1$ implies $|\delta(q, \sigma, s)| = 0$ for all $\sigma \in \Sigma$, we call M a *deterministic* one-counter automaton and identify the singleton sets in the image of δ with their elements.

As for context-free languages, the intersection non-emptiness problem for deterministic one-counter automata is undecidable [Böhm and Göller, 2011].

Next, we consider further restricted subclasses of counter automata, where we are not allowed to access any information about the counter value during the computation. The following two classes of counter automata were first introduced in [Greibach, 1978].

Definition 19 (Blind Counter Automaton)**.** Let $M = (Q, \Sigma, \delta, q_0, F)$ be a one-counter automaton. We say that M is *blind*, if for each $q \in Q$, and $\sigma \in (\Sigma \cup \{\varepsilon\})$ it holds that $\delta(q, \sigma, -1) = \delta(q, \sigma, 0) = \delta(q, \sigma, 1)$.

In other words, the machine must behave in exactly the same way regardless of whether the counter is negative, zero, or positive. Hence, the transition is 'blind' with respect to the state of the counter. Note that M still needs to accept with a zero counter as it is still a one-counter automaton. Next, we consider partially blind counter automata.

[2]Here, the set $\{-1, 0, 1\}$ in the domain of δ indicates whether the counter is smaller, equal, or greater than zero, while the set $\{-1, 0, 1\}$ in the image indicates whether the counter is decremented, not altered, or incremented by this transition.

Definition 20 (Partially Blind Counter Automaton)**.** Let $M = (Q, \Sigma, \delta, q_0, F)$ be a one-counter automaton. We say that M is *partially blind*, if for each $q \in Q$, and $\sigma \in (\Sigma \cup \{\varepsilon\})$ it holds that (1) $\delta(q, \sigma, -1) = \emptyset$ and (2) $\delta(q, \sigma, 0) = \delta(q, \sigma, 1)$.

The difference between partially blind and blind counter automata is that partially blind automata are not allowed to count below zero. Again, as for blind counter automata, the transition of a partially blind counter automata for non-negative counter values must not depend on the value of the counter.

The class of languages recognizable by partially blind counter automata is incomparable with the class of languages recognizable by blind automata [Boasson, 1973, Jantzen and Kurganskyy, 2003, Latteux, 1977]. Both are strict subclasses of languages recognizable by counter automata.

Example 3. An example for a language which can be accepted by all three models of counter automata is the language $\{a^n b^n \mid n \in \mathbb{N}\}$. Let $\#_\sigma$ be a function returning the number of occurrences of the letter σ in a word w. In contrast to the last example, the language $\{w \in \{a, b\}^* \mid \#_a(w) = \#_b(w)\}$, can be accepted by a blind counter automaton but not by a partially blind automaton as depending on the distribution of a's and b's in the word the counter necessarily needs to take negative values for some words. In contrast, the language $\{w \in \{a, b\}^* \mid \#_a(w) = \#_b(w) \wedge \forall i \in [|w|] \colon \#_a(w[1..i]) \geq \#_b(w[1..i])\}$ can be accepted by a partially blind automaton as the latter condition is satisfied if the counter, counting the difference in the number of a's against b's, never runs below zero. A language recognizable by a counter automaton which cannot be recognized by neither a blind, nor a partially blind automaton is for instance $\{a^x b^y c^z d^w \mid x, y, z, w \in \mathbb{N}, x = y \wedge z = w\}$. Here, reading an input word w, a counter automaton must decide after reading the last b whether the first condition enforcing the number of consecutive a's being equal to the number of consecutive b's is satisfied before the number of c' is counted. Otherwise, there is no way to distinguish the remaining difference of a's and b's from the difference of c's and d's after reading w completely.

2.2.4 Finite-Turn Push-Down Automata

Instead of restricting the stack alphabet of a push-down automaton we can also obtain a smaller language class by restricting the *behavior* of the stack. Here, one option is to restrict the number of times the automaton switches between increasing and decreasing the height of a stack. We call pushd-down automata with the restriction of switching only a finite number of times between these two phases for each word in the recognized language, a *finite-turn* push-down automata.

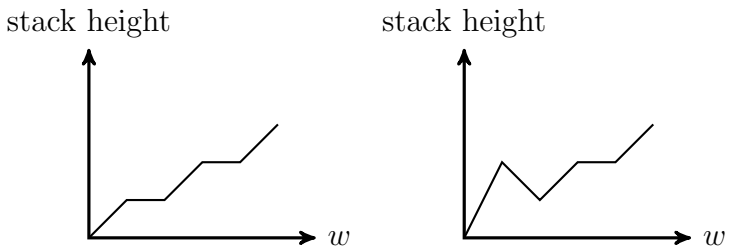


Figure 2.2: Two illustrative stack height profile of a push-down automaton reading an input word w. The one on the left consists of only one upstroke and hence has zero turns. In contrast, the height profile on the right consists of an upstroke, followed by a downstroke, and again an upstroke and hence has two turns.

Finite-turn PDAs are introduced in [Ginsburg and Spanier, 1966]. From the formal language side, it is known that one-turn PDAs characterize the well-known family of linear context-free languages, usually defined via grammars. In our setting, the automata view is more interesting. We adopt the definition in [Valiant, 1973]. For a PDA M an *upstroke* of M is a sequence of configurations induced by an input word w such that no transition decreases the stack-height. Accordingly a *downstroke* of M is a sequence of configurations in which no transition increases the stack-height. A stroke is either an upstroke or downstroke. Note that exchanging the top symbol of the stack is allowed in both an up- and a downstroke.

Definition 21 (n-Turn PDA)**.** Let M be a PDA. We call M an n-turn PDA if for all words $w \in \mathcal{L}(M)$ accepted by M, every accepting sequence of configurations induced by w can be split into at most $n + 1$ strokes.

Especially, for 1-turn PDAs each sequence of configurations induced by an accepting word consists of one upstroke followed by at most one downstroke.

Example 4 (Stack Height Profile). We illustrate the concept of a finite-turn PDA by considering the *stack height profiles* in Figure 2.2. A stack height profile plots for a sequence of configurations, the height of the stack, i.e., the size of the string $\gamma \in \Gamma^*$ on the stack, against the letters of the input word. The stack height profile depicted on the left of Figure 2.2 consists of only one upstroke and hence has zero turns. In contrast, the stack height profile on the right consists of an upstroke, followed by a downstroke, and again an upstroke and hence has two turns.

2.2.5 Visibly Push-Down Languages

Instead of restricting the stack height profile in terms of number of turns, we can associate the alteration of the stack height with the read input letter. If this alteration

is independent of the current state, we arrive at the subclass of *visibly push-down* languages which share a lot of nice properties with regular languages as we explain later. Due to these 'regular' properties, visibly push-down languages and their respective automaton model of *visibly push-down automata* are of great interest in the field of model checking [Bouajjani et al., 1997, Esparza et al., 2003, Ball and Rajamani, 2000] as they are still rich enough to model program analysis questions [Alur and Madhusudan, 2004]. A *visibly push-down automaton*, VPDA for short, is a PDA where the input alphabet Σ can be partitioned into $\Sigma = \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}$ such that the change in the stack height is determined by the partition of the alphabet.

Definition 22 (Visibly Push-Down Automaton [Alur and Madhusudan, 2004])**.** Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \bot, F)$ be a PDA. We call M a *visibly* push-down automaton if M does not have any ε-transitions and there is a partition $\Sigma = \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}$ and δ can be redefined as a partition $\delta = \delta_{\text{c}} \cup \delta_{\text{i}} \cup \delta_{\text{r}}$ such that $\delta_{\text{c}} \colon Q \times \Sigma_{\text{call}} \to Q \times (\Gamma \setminus \{\bot\})$ puts a symbol on the stack, $\delta_{\text{i}} \colon Q \times \Sigma_{\text{int}} \to Q$ leaves the stack unchanged, and $\delta_{\text{r}} \colon Q \times \Sigma_{\text{ret}} \times \Gamma \to Q$ reads and pops a symbol from the stack. If $\bot$ is the symbol on top of the stack, then $\bot$ is only read and not popped. We call letters in Σ_{call} *call* or *push* letters; letter in Σ_{int} *internal* letters; and letters in Σ_{ret} *return* or *pop* letters.

The language class accepted by VPDA is equivalent to the class of languages accepted by nested word automata (see [Chistikov et al., 2019]).

What makes visibly push-down languages especially interesting is that they share several nice properties with regular languages. For instance, the class of visibly push-down languages is closed under *intersection*, union, concatenation, Kleene-star, and *complement* with respect to the set of all well matched words [Alur and Madhusudan, 2004, Ludwig, 2019]. Recall that context-free languages are not closed under intersection and complement. Visibly push-down languages further obey nice properties with respect to decision problems. So the following questions are decidable for given visibly push-down automata M_1 and M_2: is $\mathcal{L}(M_1) = \mathcal{L}(M_2)$, is $\mathcal{L}(M_1) \subseteq \mathcal{L}(M_2)$, as well as is $\mathcal{L}(M_1) = \emptyset$, whereas the former two problems are undecidable for context-free languages [Alur and Madhusudan, 2004, Ludwig, 2019].

If not only the stack height is predefined by the input letter but also the exact stack content, we arrive at the class of *very visibly push-down automata*, VVPDA for short, introduced in [Ludwig, 2019].

Definition 23 (Very Visibly Push-Down Automaton)**.** Let $M = (Q, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, \Gamma, \delta_{\text{c}} \cup \delta_{\text{i}} \cup \delta_{\text{r}}, q_0, \bot, F)$ be a VPDA. We call M a *very visibly* push-down automaton if for each letter $\sigma \in \Sigma$ and all states $p, q \in Q$ for $\delta_{\text{c}}(p, \sigma) \ni (p', \gamma_p)$ and $\delta_{\text{c}}(q, \sigma) \ni (q', \gamma_q)$ it holds that $\gamma_p = \gamma_q$.

Example 5. Every VPDA for which $|\Gamma \setminus \{\bot\}| = 1$ holds is already a very visibly push-down automaton. Thereby, again the language $\{a^n b^n \mid n \in \mathbb{N}\}$ is an example for a very visibly push-down language.

On the contrast, the counter language $\{w \in \{a, b\}^* \mid \#_a(w) = \#_b(w)\}$ is not visible as depending on the ratio between read a's and b's each letter either has to cause a push or pop from the stack.

A language that is visibly but not very visibly [Ludwig, 2019, p. 78] is

$$\{a^m b^n a^o b^o c b^{m-n} \mid m, n, o \in \mathbb{N}, m \geq n\}.$$

Here, a VPDA must note with a special symbol on the stack when the second block of a's begins, which is not possible for VVPDAs.

An example for a context-free language that is neither a counter language nor a visibly push-down language is $\{w\$w^R \mid w \in \{0, 1\}^*\}$.

Chapter 3

Computational Complexity

3.1 Classical Time and Space Classes

Next, we introduce the concept of a Turing machine, or TM for short. We first give an intuition of the machine model. A Turing machine consists of a finite state control, an infinite work-tape divided into cells and a tape head that can access one tape cell in each step. Each computation starts with the input word w on the tape, the tape head is on the first position of w and every cell to the left and right of w is assumed to contain the blank symbol $\square$. In each step, the TM reads the symbol in the cell under the tape head, updates its current state via its transition function, rewrites the symbol in the cell under the head, and either moves the head one position to the left, to the right or stays on the current cell. Starting in the initial state q_0, the TM accepts input w if it ever reaches an accepting state $q_f \in F$ during the computation.

Definition 24 (Turing Machine)**.** Let $M = (Q, \Sigma, \Gamma, \square, \delta, q_0, F)$. We call M a *non-deterministic Turing machine* if the following holds.

- Q is a finite set of states,
- Σ is a finite input alphabet,
- $\Gamma \supseteq \Sigma$ is a finite tape alphabet,
- $\square \in \Gamma \setminus \Sigma$ is a dedicated blank symbol,
- $\delta \colon Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,N,R\}}$ is a transition function,
- $q_0 \in Q$ is the initial state, and

- $F \subseteq Q$ is a set of final states.

If for all $q \in Q, \gamma \in \Gamma$ it holds that $|\delta(q, \gamma)| \leq 1$, then we call M a *deterministic Turing machine*, DTM for short. We abbreviate a non-deterministic Turing machine as an NTM. If it is not important whether a Turing machine is deterministic or non-deterministic, we simply use the term Turing machine and abbreviate it as a TM.

We call (α, q, β) a *configuration* of M if the following holds. We have that $\alpha = \square\hat{\alpha} \in \Gamma^*$ and $\beta = \hat{\beta}\square \in \Gamma^*$ with $\hat{\alpha}\hat{\beta}$ being the content of the tape where $\hat{\alpha}$ starts with the leftmost non-blank symbol on the tape and $\hat{\beta}$ ends with the rightmost non-blank symbol on the tape. The current state of the TM is q and the tape head is on the cell separating the string $\alpha\beta$ with α written to the left of this cell and β starting in the cell under the head, written to the right. On input $w \in \Sigma^*$, the TM starts in configuration (ε, q_0, w). Let $C = (\alpha, q, \beta)$ and $C' = (\alpha', q', \beta')$ be two configurations of M. Let $|\alpha| = a$ and $|\beta| = b$. Then, we say that M transitions from configuration C to configuration C', denoted as $C \vdash_M C'$, if and only if one of the following three cases is true.

- $\delta(q, \beta[1]) \ni (q', \beta'[1], N)$, $|\alpha'| = a$, $|\beta'| = b$, $\alpha' = \alpha$, and $\beta'[2..b] = \beta[2..b]$
- $\delta(q, \beta[1]) \ni (q', \beta'[2], L)$, $|\alpha'| = a - 1$, $|\beta'| = b + 1$, $\alpha' = \alpha[1..a-1]$, $\beta'[1] = \alpha[a]$, and $\beta'[3..b+1] = \beta[2..b]$
- $\delta(q, \beta[1]) \ni (q', \alpha'[a+1], R) = 1$, $|\alpha'| = a + 1$, $|\beta'| = b - 1$, $\alpha'[1..a] = \alpha[1..a]$, and $\beta'[1..b-1] = \beta[2..b]$.

In other words, we say that M transitions from C to C' if the tape content of both configurations is identical, except for the cell under the head in configuration C, and the content of this cell as well as the current state changes according to δ. Further, the head must move at most one step between these two configurations, also according to δ.

With $\vdash_M^*$ we denote the reflexive, transitive closure of $\vdash_M$. We call a configuration (α, q, β) an *accepting configuration* if $q \in F$. Let $w \in \Sigma^*$ be an input word. We call a sequence of configurations $(\square, q_0, w\square), (\alpha_1, p_1, \beta_1), (\alpha_2, p_2, \beta_2), \ldots, (\alpha_n, p_n, \beta_n)$ a *run* of M on w, if $(\square, q_0, w\square) \vdash_M (\alpha_1, p_1, \beta_1)$ and for each $1 \leq i \leq n-1$, $(\alpha_i, p_i, \beta_i) \vdash_M (\alpha_{i+1}, p_{i+1}, \beta_{i+1})$. We call a run an *accepting run*, if (α_n, p_n, β_n) is an accepting configuration. The set of stings $\mathcal{L}(M)$ accepted by M is defined as $\{w \in \Sigma^* \mid \exists \alpha, \beta \in \Gamma^*, q \in F \colon (\square, q_0, w\square) \vdash_M^* (\alpha, q, \beta)\}$. We say that a TM M *solves* a *computational problem* $P \subseteq \Sigma^*$ if and only if $\mathcal{L}(M) = P$. We make the convention that a TM immediately halts after reaching any accepting configuration.

Definition 25 (Time and Space Bounds)**.** Let M be a TM. If there exists a function $f\colon \mathbb{N} \to \mathbb{N}$, such that $\forall n \in \mathbb{N}, \forall w \in \Sigma^n$, M halts after at most $f(n)$ steps on input w, we say that M is *time bounded* by the function f. For a function $g\colon \mathbb{N} \to \mathbb{N}$, we call M *space bounded* by the function g if $\forall n \in \mathbb{N}, \forall w \in \Sigma^n$, M halts on input w and if w is accepted, then for each accepting run of M on w, it holds that each configuration $C = (\alpha, p, \beta)$ in this run satisfies $|\alpha\beta| \leq g(n)$.

Now, we are ready to define several standard complexity classes. We say that a Turing machine M is *polynomial-time* (respectively *polynomial-space*) bounded, if there exists a polynomial function $f\colon \mathbb{N} \to \mathbb{N}$ (respectively polynomial function $g\colon \mathbb{N} \to \mathbb{N}$) such that M is time bounded by f (respectively space bounded by g). We say that a Turing machine is *exponential-time bounded*, if there exists an exponential function $f\colon \mathbb{N} \to \mathbb{N}$, i.e., a function in $2^{n^{\mathcal{O}(1)}}$, such that M is time bounded by f. We say that a Turing machine is *logarithmic-space bounded*, if there exists a logarithmic function $g\colon \mathbb{N} \to \mathbb{N}$ such that M is space bounded by g.

Definition 26 (Standard Complexity Classes)**.** We denote the class of computational problems solvable by

- deterministic logarithmic-space bounded Turing machines as LOG,
- non-deterministic logarithmic-space bounded Turing machines as NLOG,
- deterministic polynomial-time bounded Turing machines as P,
- non-deterministic polynomial-time bounded Turing machines as NP,
- deterministic polynomial-space bounded Turing machines as PSPACE,
- non-deterministic polynomial-space bounded Turing machines as NPSPACE,
- deterministic exponential-time bounded Turing machines as EXPTIME,
- non-deterministic exponential-time bounded Turing machines as NEXPTIME.

One major issue in the computational complexity theory is that most relations between complexity classes are only known as inclusions, but it is open whether these inclusions are strict. One famous exception for this are the classes PSPACE and NPSPACE. Due to an impressive result of Savitch from 1970, it holds that NPSPACE = PSPACE [Savitch, 1970]. For the remaining classes introduced above, we only know, due to the time and space hierarchy theorems [Hopcroft and Ullman, 1979], that LOG and NLOG are strictly contained in PSPACE, that P is strictly contained in EXPTIME, and that NP is strictly

contained in NEXPTIME. For all other relations in the following chain it is unknown (and a very hard question), whether the inclusions are strict.

$$\mathsf{LOG} \subseteq \mathsf{NLOG} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} = \mathsf{NPSPACE} \subseteq \mathsf{EXPTIME} \subseteq \mathsf{NEXPTIME}$$

Proving membership in one of these classes is relatively easy by constructing an algorithm within the respective time and space bounds, but how can we show that a problem is *not* contained in a complexity class, especially since we cannot even show that the class inclusion itself is strict? A precise and sufficient solution to this problem is still open. A framework to tackle this issue includes identifying *hardest problems* within a complexity class. A hard problem L with respect to a complexity class $\mathcal{X}$ is a problem to which all problems in $\mathcal{X}$ can be *reduced*. We will be more formal on this in a moment. This reduction relation implies that if we could solve L more efficiently, i.e., if L is contained in a potentially smaller class $\mathcal{X}' \subseteq \mathcal{X}$, then *all* problems in $\mathcal{X}$ are contained in $\mathcal{X}'$. In practice, we are most concerned with the distinction of the complexity classes P and NP, as the former class contains problems solvable in polynomial time while for problems in the latter class in general only exponential time deterministic algorithms are known. The question whether $\mathsf{P} = \mathsf{NP}$ is one of the *Millennium Prize Problems* and endowed with one million euros [Jaffe, 2006]. The first problem shown to be a hardest problem for the complexity class NP is the satisfiability problem (SAT for short) for Boolean formulas, by a famous proof by Cook [Cook, 1971] that encodes Turing machine computations into SAT formulas. Especially, for the SAT problem a lot of effort has been made over the last, at least, five decades trying to find algorithms with a better worst case complexity than the trivial exponential time algorithm of iterating through all possible variable assignments. All of these attempts have been unsuccessful so far. This gives some indication that $\mathsf{P} \neq \mathsf{NP}$ which is the prevailing opinion in the complexity theory community. There are even quite popular tools for proving conditional lower-bounds, such as the *exponential time hypothesis* (ETH) and the *strong exponential time hypothesis* (SETH), roughly based on the assumption that SAT cannot be be solved in sub-exponential time.

We will now formally introduce the concepts of a reduction, as well as hardness and completeness for a complexity class.

Definition 27 (Reduction)**.** Let $L_1 \subseteq \Sigma^*$ and $L_2 \subseteq \Gamma^*$ be two computational problems. We call a computable function $f\colon \Sigma^* \to \Gamma^*$ a *many-one reduction* if the following condition holds,

$$\forall x \in \Sigma^*\colon x \in L_1 \Leftrightarrow f(x) \in L_2.$$

If there exists a *many-one reduction* from L_1 to L_2 we say that L_1 *reduces* to L_2, or *is*

reducible to L_2 and denote this by $L_1 \leq L_2$. If f can be computed by a polynomial-time bounded deterministic Turing machine, we call f a *polynomial-time many-one reduction*. In this case we write $L_1 \leq_P L_2$.

Definition 28 (Completeness)**.** Let $\mathcal{X} \subseteq 2^{\Sigma^*}$ be a class of computational problems. Let $\hat{L} \subseteq \Sigma^*$ be a computational problem. If $\forall L \in \mathcal{X}$ it holds that $L \leq_P \hat{L}$, then we say that $\hat{L}$ is $\mathcal{X}$-hard under polynomial-time reductions. If further $\hat{L} \in \mathcal{X}$, then we say that $\hat{L}$ is $\mathcal{X}$-complete (under polynomial-time many-one reductions).

As it is widely believed that SAT cannot be solved in polynomial time and SAT was proven to be NP-complete by Cook [Cook, 1971], showing that a problem is NP-hard is a strong indication that the problem is not contained in P.

Polynomial time reductions are only useful to identify hard problems within complexity classes including and above NP as here the reduction has less computational power than the Turing machine associated with the complexity class. For the class P (and NLOG), we need reductions computable in logarithmic space in order to identify useful hard languages (under polynomial-time reduction every problem in P is P-hard) and for LOG we even need to restrict ourselves to AC^0 reductions.

3.2 Parameterized Complexity

The theory of Parameterized Complexity, was invented by Downey and Fellows [Downey and Fellows, 1999] as an attempt to deal with the fact that there is little hope to find algorithms for NP-complete problems with a worst-case running time which is polynomial in the size n of the input. The idea behind parameterized algorithms is to identify hard problems (such as NP-hard) that do not require an exponential running time in the whole input size but instead the exponential blowup is fully dependent on a part of the input. This part is called the *parameter* of the problem. This parameter could for instance be a natural number associated with the input, such as the size of a sought vertex cover, or it could be a structural parameter, such as the treewidth of the input graph. If this parameter is small, an algorithm whose running time is only exponential in the parameter and polynomial in the whole input would still be quite efficient even though the running time is not polynomial. The class of *fixed-parameter tractable* problems (FPT for short) generalizes this idea by allowing not only exponential but *any* function in the parameter. In order to define FPT, we first generalize a computational problem $L \subseteq \Sigma^*$ to a *parameterized* computational problem (L, κ) where κ is a function returning the value of the parameter for each actual instance $x \in \Sigma^*$.

Definition 29 (FPT)**.** The class of *fixed-parameter tractable* problems consists of all parameterized problems (L, κ), with $L \subseteq \Sigma^*$, for which there exists a deterministic Turing machine M and functions $f, g\colon \mathbb{N} \to \mathbb{N}$ with the following property. On each input $x \in \Sigma^*$, M decides membership $x \in L$ in time $f(\kappa(x)) \cdot g(|x|)$, where g is a polynomial function and f is any computable function.

Considering the problems in FPT as being efficiently solvable again raises the question of whether there are parameterized problems (for instance in NP) *not* efficiently solvable, and how we could show or at least indicate this. This led to the invention of the W-hierarchy where the first two level W[1] and W[2] are of special interest. Generally speaking, we can consider the hard problems in W[1] as not likely to be fixed-parameter tractable. As before, we use the notion of a reduction to indicate that a problem not only belongs to (for instance) W[1], but is unlikely to belong to FPT. As we are now talking about parameterized problems, we also have to update the concept of a reduction to deal with the parameter.

Definition 30 (Parameterized Reduction)**.** Let (L_1, κ_1) and (L_2, κ_2) be parameterized problems with $L_1 \subseteq \Sigma^*$ and $L_2 \subseteq \Gamma^*$. We call a function $\varphi\colon \Sigma^* \to \Gamma^*$ an *fpt-reduction* if the following conditions hold.

- For all $x \in \Sigma^*$ holds $x \in L_1 \Leftrightarrow \varphi(x) \in L_2$.
- The reduction φ can be computed by a deterministic TM with running time bounded by $f(\kappa_1(x)) \cdot g(|x|)$ where g is a polynomial function and f is any computable function.
- There is a computable function $h\colon \mathbb{N} \to \mathbb{N}$ such that $\kappa_2(\varphi(x)) \leq h(\kappa_1(x))$ for all $x \in \Sigma^*$.

We will only give a definition for the first two levels of the W-hierarchy which can be defined via single and multi-tape Turing machines. A *multi-tape* Turing machine is a Turing machine that has not only one but m infinite tapes for its computation. The head of the TM is therefore split into m heads (one for each tape) which can independently move in each step. The transition function δ then depends on the current symbol under each head in combination, meaning that the symbol under the head on tape i can influence the action performed on tape j. For the definition of W[1] and W[2] we make the convention that undefined transitions, i.e., those for which δ maps to the empty set, are not included in the description of a Turing machine.

Definition 31 (Class W[1])**.** We define the class W[1] as the set of all parameterized problems that can be reduced via an fpt-reduction to the bounded Halting problem of

non-deterministic one-tape Turing machines defined as:
Input: Non-deterministic one-tape Turing machine M and integer $k \in \mathbb{N}$.
Question: Does M halt, starting on the empty tape, after at most k steps?
Here, the number k is the parameter of the problem.

As we are using fpt-reductions, the parameter of the problem we reduce to (bounded Halting problem) is bounded by some function in the parameter of the problem we reduce from. Hence, the maximal running time of the NTM in the image of the reduction is bounded by some computable function in the parameter of the problem we reduce from. Thereby, the 'allowed non-determinism' of the Turing machine in the image is bounded by the parameter of the input instance.

Definition 32 (Class $\mathsf{W}[2]$)**.** We define the class $\mathsf{W}[2]$ as the set of all parameterized problems that can be reduced via an fpt-reduction to the bounded Halting problem of non-deterministic multi-tape Turing machines defined as:
Input: Non-deterministic multi-tape Turing machine M and integer $k \in \mathbb{N}$.
Question: Does M halt, starting on the empty tape, after at most k steps?
Here, the number k is the parameter of the problem.

Note that the number of tapes is not restricted. Further, this is the only and crucial difference between the definition of $\mathsf{W}[1]$ and $\mathsf{W}[2]$. We now give examples of a problem contained (or complete) for each class.

Example 6. All three examples are parameterized version of graph problems where the parameter is the natural parameter of the problem. Hence, for each problem the input consists of an undirected graph $G = (V, E)$ and an integer k. Then, the question is

- Vertex Cover: Does there exists a *vertex cover* in G of size at most k, i.e., is there a subset $V' \subseteq V$ with $|V'| \leq k$ such that for each $e \in E$ it holds that $e \cap V' \neq \emptyset$?
- Clique: Does there exists a *clique* in G of size at least k, i.e., is there a subset $V' \subseteq V$ with $|V'| \geq k$ such that for each $u, v \in V'$ with $u \neq v$ it holds that $\{u, v\} \in E$?
- Dominating Set: Does there exists a *dominating set* of size at most k, i.e., is there a subset $V' \subseteq V$ with $|V'| \leq k$ such that for each $u \in (V \setminus V')$ there exists a vertex $v \in V'$ with $\{u, v\} \in E$?

The Vertex Cover problem with parameter k is fixed-parameter tractable, i.e., it is contained in FPT. In contrast, the Clique problem with parameter k is $\mathsf{W}[1]$-complete

and thereby believed not to be contained in FPT. The Dominating Set problem with parameter k seems to be even harder as it is W[2]-complete. We refer to [Flum and Grohe, 2006] for proofs of these claims.

Finally, we introduce a parameterized complexity class containing the complete infinite W-hierarchy. Here, the exponential blowup cannot be bounded to concern only the parameter, but we still do not need the whole input length in the exponent of an exponential function to bound the running time of an algorithm solving problems in this class.

Definition 33 (Class XP)**.** The class XP consists of all problems $L \subseteq \Sigma^*$ solvable by a deterministic Turing machine that decides membership $x \in L$ on input x in time $\mathcal{O}(|x|^{f(\kappa(x))})$ where $f\colon \mathbb{N} \to \mathbb{N}$ is some computable function.

Note that the hierarchy of parameterized complexity classes is askew to the classical hierarchy including NP, PSPACE (and the polynomial hierarchy in between). Therefore, there are PSPACE complete problems with a parameterized version in FPT and on the other hand problems in NP with a W[2]-hard parameterized version.

Chapter 4

Overview of Scientific Results in Part II

Part II consists of the six research papers introduced in Chapter 1. In this chapter, we give an overview on the main results obtained in each paper. For an introduction to each paper, we refer to Chapter 1. Here, we directly give the definitions of the studied problems and state the obtained complexity results in the form of a table. Finally, we state open problems for each paper. Each paper in Part II is a slight modification of the published work in the sense that missing proofs are reintegrated into the extended abstracts.

Computational Complexity of Synchronization under Regular Constraints

In Chapter 7, we present the work in [Fernau et al., 2019]. There, we are interested in finding synchronizing words which fulfill some regular constraints, i.e., are contained in a regular constraint language. More formally, for a fixed partial DFA $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$, we define the *constrained synchronization problem*:

Definition 34. $\mathcal{L}(\mathcal{B})$-Constr-Sync
Input: Deterministic complete finite automaton $A = (\Sigma, Q, \delta)$.
Question: Is there a synchronizing word w for A with $w \in \mathcal{L}(\mathcal{B})$?

The automaton $\mathcal{B}$ will be called the *constraint automaton*.

We give a full analysis of the $\mathcal{L}(\mathcal{B})$-Constr-Sync problem for constraint automata with two states and up to a ternary alphabet. We observe that for any two-state partial DFA $\mathcal{B}$ over a *unary* or *binary* alphabet the problem $\mathcal{L}(\mathcal{B})$-Constr-Sync is in P. For

Case	Language
1	$a(b+c)^*$
	$(a+b+c)(a+b)^*$
	$(a+b)(a+c)^*$
2	$(a+b)^*c$
	$(a+b)^*ca^*$
	$(a+b)^*c(a+b)^*$
	$(a+b)^*cc^*$
3	$a^*b(a+c)^*$
	$a^*(b+c)(a+b)^*$
4	$a^*b(b+c)^*$
	$(a+b)^*c(b+c)^*$
	$a^*(b+c)(b+c)^*$

Table 4.1: Constraint languages accepted by constraint automata with two states and three letters causing a PSPACE-hard $\mathcal{L}(\mathcal{B})$-Constr-Sync problem.

two state partial DFAs $\mathcal{B}$ over a *ternary* alphabet, a full analysis yields only solvability in P or PSPACE-completeness as complexities for $\mathcal{L}(\mathcal{B})$-Constr-Sync. The constraint automata $\mathcal{B}$ yielding PSPACE-hard versions of the problem are listed in Table 4.1 represented by their accepted language given as a regular expression. The languages are grouped by their technical similarities, i.e., reductions used to obtain the result.

We further obtain general results such as the following. If the constraint automaton $\mathcal{B}$ is *returning*, i.e., from any state there exists a word mapping this state back to the initial state q_0, then $\mathcal{L}(\mathcal{B})$-Constr-Sync is solvable in polynomial time. The problem is also solvable in polynomial time if every word $v \in \Sigma^*$ can be extended to a word in $\mathcal{L}(\mathcal{B})$, i.e., if $\{ v \in \Sigma^* \mid \exists u, w \in \Sigma^* : uvw \in \mathcal{L}(\mathcal{B}) \} = \Sigma^*$.

We further show that for general constraint automaton, the $\mathcal{L}(\mathcal{B})$-Constr-Sync is contained in NP if there is a letter σ, such that each loop in the automaton is labeled by a unary word from $\{\sigma\}^*$. In this case we also obtain an XP algorithm with parameter "number of loops labeled with σ". We further present technical results that lift the obtained upper and lower bounds to wider classes of constraint automata with arbitrary number of states and letters.

Open Problems

- Based on the work in [Fernau et al., 2019] a line of research considering subclasses of regular languages as constraint languages was pursued in [Hoffmann, 2020a, Hoffmann, 2021a, Hoffmann, 2021b, Hoffmann, 2020b, Hoffmann, 2020c]. Still,

a full classification of the complexity of L-Constr-Sync for regular constraint languages L is an open research problem.

- So far, we only observed containment in P, NP-completeness and PSPACE-completeness as complexities for $\mathcal{L}(\mathcal{B})$-Constr-Sync. Motivated by a remark of Rystsov [Rystsov, 1983] in a related setting, one could also ask if there are regular constraint languages yielding $\mathcal{L}(\mathcal{B})$-Constr-Sync problems that are complete for other levels of the polynomial-time hierarchy.

Synchronization under Dynamic Constraints

In Chapter 8, we present the work in [Wolf, 2020]. This research paper was awarded with the 2021 publication prize of the Graduate Center of the University of Trier, Faculty IV. A video (in German) of the presentation of the results to the public during the award can be found under https://youtu.be/kmDPUmUt-R4.

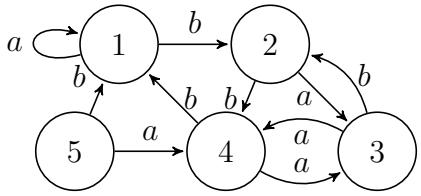
In Chapter 8, we are interested in restricting the order of states in which a synchronizing word w transitions through the automaton. Thereby, we distinguish between tracking the order of states appearing in the set of active states, i.e., in the set $Q.w[i]$ for $i \leq |w|$, and tracking the order of states on the individual paths induced by w. We study three different ways how a synchronizing word can induce an order on the state set. There, the first two orders compare the last appearances of states while the third compares last with first appearances of different states. The considered decision problem then consists of a DFA as input together with a partial order R and the question is whether there exists a synchronizing word such that the respectively induced state order coincides with the input partial order. For the third order we also consider a variant of the problem where the input partial order is actually a total order. We now define the problem and orders formally. We give an example of a DFA together with pairs contained and not contained in the induced orders in Figure 4.1. The obtained complexity results are listed in Table 4.2.

For any of the orders $\preccurlyeq_w \subseteq Q \times Q$ defined below, we define the problem of *synchronization under order* and *subset synchronization under order* as:

Definition 35 (Sync-Under-$\preccurlyeq_w$)**.** Given a DFA $A = (Q, \Sigma, \delta)$ and a relation $R \subseteq Q^2$. Does there exist a word $w \in \Sigma^*$ such that $|Q.w| = 1$ and $R \subseteq \preccurlyeq_w$?

Definition 36 (Subset-Sync-Under-$\preccurlyeq_w$)**.** Given a DFA $A = (Q, \Sigma, \delta)$, $S \subseteq Q$, and a relation $R \subseteq Q^2$. Is there a word $w \in \Sigma^*$ with $|S.w| = 1$ and $R \subseteq \preccurlyeq_w$?

It is reasonable to distinguish whether the order should include the initial configuration



	1	2	3	4	5
b	2	4	2	1	1
a	3	3	3	1	1
a	4	4	4	1	1
b	1	1	1	2	2
b	2	2	2	4	4
a	3	3	3	3	3

	✓	✗
$\propto_{w@s}^{l<l}$	$(1,2)$	$(2,4)$
$\propto_{w@s}^{l\leq l}$	$(2,4)$	$(2,1)$
$\propto_{w@p}^{l<l}$	$(1,2)$	$(4,5)$
$\propto_{w@p}^{l\leq l}$	$(5,5)$	$(2,4)$
$\propto_{w@p}^{l<f}$	$(5,2)$	$(4,3)$

Figure 4.1: DFA A (left) with all paths induced by $w = baabba$ (middle) and relations R consisting of single pairs forming a positive, resp. negative, instance for versions of Sync-Under-$\preccurlyeq_w$ (right).

of the automaton or if it should only describe the consequences of the chosen transitions. In the former case, we refer to the problem as Sync-Under-*0*-$\preccurlyeq_w$ (starting at $w[0]$), in the latter case as Sync-Under-*1*-$\preccurlyeq_w$ (starting at $w[1]$), and if the result holds for both variants, we simply refer to is as Sync-Under-$\preccurlyeq_w$. This is particularly of interest when states are left with the very first letter. Consider the order $\propto_{w@s}^{l<l}$ and a tuple (q_1, q_2). If both states are left with the very first letter and never become active while reading the synchronizing word, then, the version Sync-Under-*1*-$\preccurlyeq_w$ allows us to ignore the tuple (q_1, q_2) while for the version Sync-Under-*1*-$\preccurlyeq_w$, the tuple is violated as the state q_1 is active in the initial configuration and hence q_2 must become active at a later position.

Let $\text{first}(q, w, S)$ be the function returning the minimum of positions at which the state q appears as an active state over all paths induced by w starting at some state in S, i.e., $\text{first}(q, w, S) = \min\{i \mid i \leq |w|, q \in S.w[0..i]\}$. Accordingly, let $\text{last}(q, w, S)$ return the maximum of those positions, i.e., $\text{last}(q, w, S) = \max\{i \mid i \leq |w|, q \in S.w[0..i]\}$. Note that $\text{first}(q, w, S) = 0$ for all states $q \in S$ and > 0 for $q \in Q \backslash S$. If q does not appear on a path induced by w on S, then set $\text{first}(q, w, S) := |w| + 1$ and $\text{last}(q, w, S) := -1$. In the Sync-Under-*1*-$\preccurlyeq_w$ problem variant, the occurrence of a state at position 0 is ignored (i.e., if q occurs only at position 0 while reading w on S, then $\text{last}(q, w, S) = -1$).

Definition 37 (Order $l < l$ on sets)**.** Let $A = (Q, \Sigma, \delta)$ be a DFA. For every word $w \in \Sigma^*$ we define the relation $\propto_{w@s}^{l<l} \subseteq Q^2$ as follows. For two states $p, q \in Q$ it holds that

$$p \propto_{w@s}^{l<l} q :\Leftrightarrow \text{last}(p, w, Q) < \text{last}(q, w, Q).$$

Definition 38 (Order $l \leq l$ on sets)**.** Let $A = (Q, \Sigma, \delta)$ be a DFA. For every word $w \in \Sigma^*$ we define the relation $\propto_{w@s}^{l\leq l} \subseteq Q^2$ as follows. For two states $p, q \in Q$ it holds that

$$p \propto_{w@s}^{l\leq l} q :\Leftrightarrow \text{last}(p, w, Q) \leq \text{last}(q, w, Q).$$

Definition 39 (Order $l < l$ on paths)**.** Let $A = (Q, \Sigma, \delta)$ be a DFA. For every word

		Synchronization				*Subset Synchronization*		
Order		$l < l$	$l \leq l$	$l < f$	$l<f$-tot	$l \not< / \not\leq l$	$l < f$	$l<f$-tot
Set	*0*	PSPACE-c	PSPACE-c	–	–	PSPACE-c	–	–
	1	PSPACE-c	PSPACE-c	–	–	PSPACE-c	–	–
Path	*0*	in NP	NP-hard	PSPACE-c	P	PSPACE-c	PSPACE-c	NP-c
	1	in NP	PSPACE-c	PSPACE-c	NP-c	PSPACE-c	PSPACE-c	NP-c

Table 4.2: Overview of the computational complexity for synchronization (on the left), and subset synchronization under order (on the right) for relations $\propto^{l<l}_{w@s}$, $\propto^{l<l}_{w@p}$, $\propto^{l\leq l}_{w@s}$, $\propto^{l\leq l}_{w@p}$, and $\propto^{l<f}_{w@p}$ (tot is short for total). The upper two rows consider the set variants of the orders while the lower two rows consider the path variants. The 0 in the first column indicates that we are considering the problem variant Sync-Under-*0*-$\preccurlyeq_w$ while the 1 indicates the variant Sync-Under-*1*-$\preccurlyeq_w$, in which we ignore the initial configuration with respect to the induced order.

$w \in \Sigma^*$ we define the relation $\propto^{l<l}_{w@p} \subseteq Q^2$ as follows. For two states $p, q \in Q$ it holds that

$$p \propto^{l<l}_{w@p} q :\Leftrightarrow \forall r \in Q\colon \operatorname{last}(p, w, \{r\}) < \operatorname{last}(q, w, \{r\}).$$

Definition 40 (Order $l \leq l$ on paths)**.** Let $A = (Q, \Sigma, \delta)$ be a DFA. For every word $w \in \Sigma^*$ we define the relation $\propto^{l\leq l}_{w@p} \subseteq Q^2$ as follows. For two states $p, q \in Q$ it holds that

$$p \propto^{l\leq l}_{w@p} q :\Leftrightarrow \forall r \in Q\colon \operatorname{last}(p, w, \{r\}) \leq \operatorname{last}(q, w, \{r\}).$$

Definition 41 (Order $l < f$ on paths)**.** Let $A = (Q, \Sigma, \delta)$ be a DFA. For every word $w \in \Sigma^*$ we define the relation $\propto^{l<f}_{w@p} \subseteq Q^2$ as follows. For two states $p, q \in Q$ it holds that

$$p \propto^{l<f}_{w@p} q \Leftrightarrow \forall r \in Q\colon \operatorname{last}(p, w, \{r\}) < \operatorname{first}(q, w, \{r\}).$$

Definition 42 (Sync-Under-Total-$\propto^{l<f}_{w@p}$)**.** Given a DFA $A = (Q, \Sigma, \delta)$, a strict and total order $R \subseteq Q^2$. Is there a word $w \in \Sigma^*$ with $|Q.w| = 1$ and $R = \propto^{l<f}_{w@p}$?

The orders $\propto^{l<l}_{w@s}, \propto^{l<l}_{w@p}, \propto^{l\leq l}_{w@s}, \propto^{l\leq l}_{w@p}$ model our first introductory example of a box whose lids should be closed after rotating the box, while the order $\propto^{l<f}_{w@p}$ models the second example of a box containing water which should not be rotated after the lids were opened.

Open Problems

- We only know that Sync-Under-$\propto^{l<l}_{w@p}$ is contained in NP but it is open whether the problem is NP-complete or if it can be solved in polynomial time.
- Conversely, for Sync-Under-*0*-$\propto^{l\leq l}_{w@p}$ the problem is NP-hard but its precise complexity is unknown. It would be quite surprising to observe membership in NP

here since it would separate the complexity of this problem from the closely related problem Sync-Under-*1*-$\propto^{l\leq l}_{w@p}$.

- Further, it remains open whether for the other orders a drop in the complexity can be observed, when R is strict and total, as it is the case for $\propto^{l<f}_{w@p}$.

Synchronizing Deterministic Push-Down Automata Can Be Really Hard

In Chapter 9 we present the work in [Fernau et al., 2020]. There, we generalize the synchronization problem from deterministic finite automata to deterministic push-down automata (DPDA for short).

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \bot, F)$ be a DPDA. We discuss three different concepts of synchronizing DPDAs. For all concepts we demand that a synchronizing word $w \in \Sigma^*$ maps all states, starting with an empty stack, to the same synchronizing state, i.e., for all $q, q' \in Q$: $(q, \bot) \xrightarrow{w} (\overline{q}, \upsilon), (q', \bot) \xrightarrow{w} (\overline{q}, \upsilon')$. In other words, for a synchronizing word all runs started on some states in Q end up in the same final state. In addition to synchronizing the states of a DPDA we will consider the following two conditions for the stack content:

$$(1)\ \upsilon = \upsilon' = \bot,$$
$$(2)\ \upsilon = \upsilon'.$$

We will call (1) the *empty stack model* and (2) the *same stack model*. In the third case, we do not put any restrictions on the stack content and call this the *arbitrary stack model*.

Definition 43 (Sync-DPDA-Empty)**.**
Given: DPDA $M = (Q, \Sigma, \Gamma, \delta, \bot)$.
Question: Is there a word $w \in \Sigma^*$ that synchronizes M in the empty stack model?

For the same stack model, we refer to the synchronization problem above as Sync-DPDA-Same and as Sync-DPDA-Arb in the arbitrary stack model. Variants of these problems are defined by replacing the DPDA in the definition above by a deterministic counter automaton (DCA), a deterministic partially blind counter automaton (DPBCA), a deterministic blind counter automaton (DBCA), or by adding turn restrictions, in particular, whether the automaton is allowed to make zero or one turns of its stack movement.

The obtained complexity results are listed in Table 4.3.

class of automata	empty stack model	same stack model	arbitrary stack model
DPDA	undecidable	undecidable	undecidable
1-Turn-Sync-DPDA	undecidable	undecidable	undecidable
0-Turn-Sync-DPDA	PSPACE-complete	undecidable	PSPACE-complete
DCA	undecidable	undecidable	undecidable
1-Turn-Sync-DCA	PSPACE-complete	PSPACE-complete	PSPACE-complete
0-Turn-Sync-DCA	PSPACE-complete	PSPACE-complete	PSPACE-complete
DPBCA	decidable	decidable	decidable
DBCA	decidable	decidable	decidable

Table 4.3: Complexity status of the synchronization problem for different classes of deterministic real-time push-down automata in different stack synchronization modes as well as finite-turn variants of the respective synchronization problem.

Open Problems

- The decidability for the synchronization problems concerning deterministic partially blind counter automata is obtained from the decidable reachability problem in Petri nets. But the latter problem is only known to be decidable with a non-elementary time complexity. Therefore, an open problem is to find an algorithm solving the Sync-DPBCA problem in the respective stack models with (at least) an elementary time complexity.

- A direction for further research is to look into variants of synchronization problems for DPDAs, such as restricting the length of a potential synchronizing word. It follows from the NP-hardness of this problem for DFAs [Rystsov, 1980, Eppstein, 1990] and the polynomial-time solvability of the membership problem for DPDAs that for unary encoded length bounds this problem is NP-complete for DPDAs as well, and contained in EXPTIME for binary encoded length bounds. The precise complexity of this problem for binary encoded length bounds is left to future research.

Synchronization of Deterministic Visibly Push-Down Automata

In Chapter 10 we present the work in [Fernau and Wolf, 2020]. There, we consider the same setting of synchronizing deterministic push-down automata as in Chapter 9 but we restrict the class of DPDAs to *visibly push-down automata*. In contrast to the undecidability of the synchronization problem in all three stack models for general DPDAs, for visibly PDAs, the synchronization problem is contained in EXPTIME and hence decidable in all three models. For the empty stack model, we even obtain solvabilty in polynomial time and for the same stack model, membership in PSPACE. The problems

class of automata	empty stack model	same stack model	arbitrary stack model
DVPDA	P	PSPACE-compl	PSPACE-hard
DVPDA-NoReturn	P	PSPACE-compl	P
DVPDA-Return	P	P	PSPACE-hard
n-Turn-Sync-DVPDA	PSPACE-hard	PSPACE-hard	PSPACE-hard
0-Turn-Sync-DVPDA	P	PSPACE-compl	PSPACE-compl
DVVPDA	P	P	P
n-Turn-Sync-DVVPDA	PSPACE-hard	PSPACE-hard	PSPACE-hard
0-Turn-Sync-DVVPDA	P	PSPACE-compl	PSPACE-compl
DVCA	P	P	P
n-Turn-Sync-DVCA	PSPACE-hard	PSPACE-hard	PSPACE-hard
1-Turn-Sync-DVCA	PSPACE-compl	PSPACE-compl	PSPACE-compl
0-Turn-Sync-DVCA	P	PSPACE-compl	PSPACE-compl

Table 4.4: Complexity status of the synchronization problem for different classes of deterministic real-time visibly push-down automata in different stack synchronization modes. For the n-turn synchronization variants, n takes all values not explicitly listed. All our problems are in EXPTIME.

Sync-DVPDA, Sync-DVVPDA, and Sync-DVCA are defined as in the previous section by restricting the input DPDA to a deterministic *visibly* push-down automaton (DVPDA), a deterministic *very visibly* push-down automaton (DVVPDA), and a deterministic *visibly counter* automaton (DVCA), respectively. The finite-turn variants are formally defined as follows.

Definition 44. n-Turn-Sync-DVPDA-Empty
Given: DVPDA $M = (Q, \Sigma, \Gamma, \delta, q_0, \bot, F)$.
Question: Is there a synchronizing word $w \in \Sigma^*$ in the empty stack model, such that for all states $q \in Q$, the sequence of configurations $(q, \bot) \xrightarrow{w} (\overline{q}, \bot)$ consists of at most $n + 1$ strokes?

For the Sync-DVPDA problem, we also consider two variants of the problem where we have the restrictions that for Sync-DVPDA-NoReturn, the set of return letters of the input DVPDA is empty, whereas for Sync-DVPDA-Return, the set of return letters of the input DVPDA *must not* be empty. By making this distinction, we can observe a difference in the complexity between the same stack and arbitrary stack setting which behaved identically in the other automaton models. The obtained results are summarized in Table 4.4.

Open Problems

- While all problems listed in Table 4.4 are contained in EXPTIME, the table lists

several problems for which their known complexity status still contains a gap between PSPACE-hardness lower bounds and EXPTIME upper bounds. Presumably, their precise complexity status is closely related to upper bounds on the length of synchronizing words which is a topic for future research.

- As for general DPDAs, a direction for future research on the synchronizability of deterministic visibly push-down automata is to consider related variants of synchronization problems such as the problem of short synchronizing words, subset synchronization, synchronization into a subset, and careful synchronization. Each of these problems is well understood for DFAs. Here is one subtlety that comes with short synchronizing words: While for finding synchronizing words of length at most k for DFAs, it does not matter if the number k is given in unary or in binary due to the known cubic upper bounds on the lengths of shortest synchronizing words, this will make a difference in other models where such polynomial length bounds are unknown. More precisely, for instance with DVPDAs, it is rather obvious that with a unary length bound k, the problem becomes NP-complete, while the status is unclear for binary length bounds. As there is no general polynomial upper bound on the length of shortest synchronizing words for VPDAs, they might be of exponential length. Hence, we do not get membership in PSPACE easily, not even for synchronization models concerning DVPDA for which general synchronizability is solvable in P, as it might be necessary to store the whole word on the stack in order to test its synchronization effects.

On the Complexity of Intersection Non-Emptiness for Star-Free Language Classes

In Chapter 11 we present the work in [Arrighi et al., 2021a] which is accepted for publication at *FSTTCS 2021*. There, we study the computational complexity of the *intersection non-emptiness* problem for finite automata restricted to different subclasses. Thereby, our main focus lies on the Straubing-Thérien hierarchy [Place and Zeitoun, 2019, Straubing, 1981, Straubing, 1985, Thérien, 1981] and the Cohen-Brzozowski dot-depth hierarchy [Brzozowski, 1976, Cohen and Brzozowski, 1971, Place and Zeitoun, 2019].

Definition 45. Intersection Non-emptiness
Given: Finite automata $A_i = (Q_i, \Sigma, \delta_i, q_{(0,i)}, F_i)$, for $1 \leq i \leq m$.
Question: Is there a word w that is accepted by all A_i, i.e., is $\bigcap_{i=1}^{m} \mathcal{L}(A_i) \neq \emptyset$?

Observe that the automata have a common input alphabet.

We show that the complexity landscape of the Intersection Non-emptiness problem

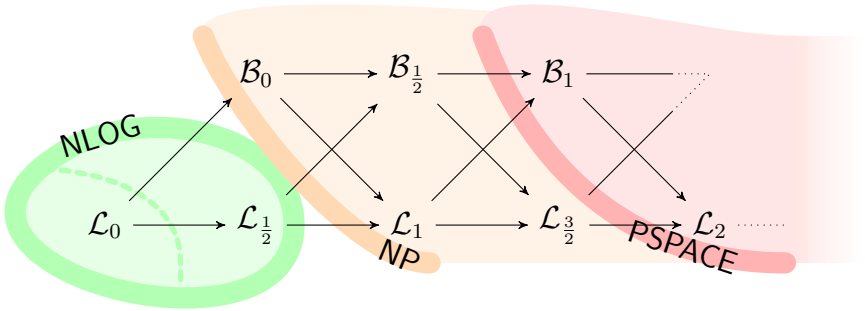


Figure 4.2: Complexity landscape of the Intersection Non-emptiness problem for the lower levels of the Straubing-Thérien and dot-depth hierarchies.

restricted to levels of those hierarchies is already determined by the very first levels of either hierarchy as depicted in Figure 4.2. The first main result states that the Intersection Non-emptiness problem for NFAs and DFAs accepting languages from the level 1/2 of the Straubing-Thérien hierarchy are NLOG-complete and LOG-complete, respectively, under AC^0 reductions. Additionally, this completeness result holds even in the case of unary languages.

This result is optimal in the sense that the problem for DFAs already becomes NP-hard if we allow a single DFA to accept a language from $\mathcal{L}_1$, and require all the others to accept languages from $\mathcal{L}_{1/2}$. Subsequently, we analyze the complexity of Intersection Non-emptiness when all input automata are assumed to accept languages from one of the levels of $\mathcal{B}_0$ or $\mathcal{B}_{1/2}$ of the dot-depth hierarchy, or from the levels $\mathcal{L}_1$ or $\mathcal{L}_{3/2}$ of the Straubing-Thérien hierarchy. It is worth noting that NP-hardness follows straightforwardly from the fact that Intersection Non-emptiness for DFAs accepting finite languages is already NP-hard [Rampersad and Shallit, 2010]. Containment in NP, on the other hand, is a more delicate issue, and here the representation of the input automaton plays an important role. A characterization of languages in $\mathcal{L}_{3/2}$ in terms of languages accepted by partially ordered NFAs [Schwentick et al., 2001] is crucial for us, combined with the fact that Intersection Non-emptiness when the input is given by such automata is NP-complete [Masopust and Thomazo, 2015]. Intuitively, the proof in [Masopust and Thomazo, 2015] follows by showing that the minimum length of a word in the intersection of languages in the level 3/2 of the Straubing-Thérien hierarchy is bounded by a polynomial in the sizes of the minimum partially ordered NFAs accepting these languages. To prove that Intersection Non-emptiness is in NP when the input automata are given as DFAs, we prove a new result establishing that the number of Myhill-Nerode equivalence classes in a language in the level $\mathcal{L}_{3/2}$ is at least as large as the number of states in a minimum partially ordered automaton representing the same language.

Interestingly, we show that the proof technique used to prove this last result does not generalize to the context of NFAs. To prove this, we carefully design a sequence $(L_n)_{n\in\mathbb{N}_{\geq 1}}$

of languages over a binary alphabet such that for every $n \in \mathbb{N}_{\geq 1}$, the language L_n can be accepted by an NFA of size n, but any partially ordered NFA accepting L_n has size $2^{\Omega(\sqrt{n})}$. To the best of our knowledge, this is the first exponential separation between the state complexity of general NFAs and that of partially ordered NFAs. While this result does not exclude the possibility that Intersection Non-emptiness for languages in $\mathcal{L}_{3/2}$ represented by general NFAs is in NP, it gives some indication that proving such a containment requires substantially new techniques.

Finally, we show that Intersection Non-emptiness for both DFAs and for NFAs is already PSPACE-complete if all accepting languages are from the level $\mathcal{B}_1$ of the dot-depth hierarchy or from the level $\mathcal{L}_2$ of the Straubing-Thérien hierarchy.

Open Problems

- First, we were not able to prove containment in NP for the Intersection Non-emptiness problem when the input automata are allowed to be NFAs accepting a language in the level 3/2 or in the level 1 of the Straubing-Thérien hierarchy. Interestingly, we have shown that such containment holds in the case of DFAs, but have shown that the technique we have used to prove this containment does not carry over to the context of NFAs. Therefore, the most immediate open question is if Intersection Non-emptiness for NFAs accepting languages in $\mathcal{B}_{1/2}$, $\mathcal{L}_1$, or $\mathcal{L}_{3/2}$ is complete for some level higher up in the polynomial-time hierarchy, or if this case is already PSPACE-complete. An intermediate step in solving this problem might be to consider so called *subatomic* non-deterministic automata introduced recently in [Myers and Urbat, 2021].

- Another open question is whether one can capture the levels of the polynomial hierarchy in terms of the Intersection Non-emptiness problem when the input automata are assumed to accept languages belonging to levels of a sub-hierarchy of $\mathcal{L}_2$. Such sub-hierarchies have been considered for instance in [Klíma and Polák, 2011].

- A systematic study of the two well-known Straubing-Thérien and dot-depth hierarchies for related problems like Non-universality for NFAs or Union Non-universality for DFAs is another promising direction of future research. Note that Union Non-universality (similar to Intersection Non-emptiness) has an implicit Boolean operation (now union instead of intersection) within the problem statement, while Non-universality lacks this implicit Boolean operation. This might lead to a small 'shift' in the discussions of the hierarchy levels that involve Boolean closure.

class of automata	DECOMP	BOUND-DECOMP
DFAs	EXPSPACE	**PSPACE**
permutation DFAs	**NP/FPT**	**PSPACE**
commutative permutation DFAs	**NLOG**	**NP-complete**
unary DFAs	LOG	**LOG**

Table 4.5: Complexity of studied problems with containing classes, with our contribution in **bold**. The membership in EXPSPACE for general DFAs is obtained in [Kupferman and Mosheiff, 2015] and the membership in LOG for unary DFAs is obtained in [Jecker et al., 2020].

Decomposing Permutation Automata

In Chapter 12 we present the work in [Jecker et al., 2021]. There, our aim is to determine whether a given DFA A can be represented as the intersection of a finite set of smaller DFAs, or in other words, if A can be *decomposed* into smaller DFAs.

We say that a DFA A is *composite* if its language can be decomposed into the intersection of the languages of smaller DFAs. More precisely, we say that A is *k-factor composite* if there exist k DFAs $(A_i)_{1 \leq i \leq k}$ with less states than A such that $\mathcal{L}(A) = \bigcap_{i=1}^{k} \mathcal{L}(A_i)$. We study the two following problems:

DFA DECOMP
Given: DFA A.
Question: Is A composite?

DFA BOUND-DECOMP
Given: DFA A and integer $k \in \mathbb{N}$.
Question: Is A k-factor composite?

Previous to our work, the best known complexity bounds for the DECOMP problem for general DFAs were membership in EXPTIME and NLOG-hardness [Kupferman and Mosheiff, 2015]. For the restricted classes of *permutation* DFAs, a PSPACE algorithm and for *normal* permutation DFAs, a P algorithm was given in [Kupferman and Mosheiff, 2015]. Recently, the DECOMP problem was shown to be decidable in LOG for DFAs with a unary alphabet [Jecker et al., 2020]. The trade-off between number and size of factors was studied in [Netser, 2018], where automata showing extreme behavior are presented, i.e., DFAs that can either be decomposed into a large number of small factors, or a small number of large factors.

In the work presented in Chapter 12, we expand the domain of instances over which the DECOMP problem is tractable. We focus on permutation DFAs, and we propose new techniques that improve the known complexities. The obtained complexity results are summarized in Table 4.5.

We give an NP algorithm for permutation DFAs, and we show that the complexity

is directly linked to the number of non-accepting states. This allows us to obtain a fixed-parameter tractable algorithm with respect to the number of non-accepting states. Moreover, we prove that permutation DFAs with a prime number of states cannot be decomposed.

We further consider *commutative* permutation DFAs, where the DECOMP problem was already known to be tractable, and we lower the complexity from P to NLOG, and even LOG if the size of the alphabet is fixed. While it is easy to decide whether a commutative permutation DFA is composite, we show that rich and complex behaviors still appear in this class: there exist families of composite DFAs that require polynomially many factors to get a decomposition. More precisely, we construct a family $(A_n^m)_{m,n\in\mathbb{N}}$ of composite DFAs such that A_n^m is a DFA of size n^m that is $(n-1)^{m-1}$-factor composite but not $(n-1)^{m-1}-1$-factor composite. Note that, prior to this result, only families of composite DFAs requiring sub-logarithmically many factors were known [Jecker et al., 2020].

Finally, we study the BOUND-DECOMP problem. For practical purposes, having many factors in a decomposition is undesirable as dealing with a huge number of small DFAs might end up being more complex than dealing with a single DFA of moderate size. The BOUND-DECOMP problem copes with this issue by limiting the number of factors allowed in the decompositions. We show that this flexibility comes at a cost: somewhat surprisingly, this problem is NP-complete for commutative permutation DFAs, a setting where the DECOMP problem is easy. We also show that this problem is in PSPACE for the general setting, and in LOG for unary DFAs.

Open Problems

- The techniques presented in this paper rely heavily on the group structure of transition monoids of permutation DFAs, thus cannot be used directly in the general setting. They still raise interesting questions: Can we also obtain an FPT algorithm with respect to the number of rejecting states in the general setting? Some known results point that bounding the number of states is not as useful in general as it is for permutation DFAs: while it is known that every permutation DFA with a single rejecting state are not composite [Kupferman and Mosheiff, 2015], there exist (non-permutation) DFAs with a single rejecting state that are composite.

- Another way to improve the complexity in the general setting would be to bound the *width* of DFAs[1]: we defined here a family of DFAs with polynomial width, do there exist families with exponential width? If this is not the case (i.e., every

[1]The width of a DFA A is the smalles k such that A is k-factor composite.

composite DFA has polynomial width), we would immediately obtain a PSPACE algorithm for the general setting.

- In this work, we focused on the Bound-Decomp problem, that limits the *number* of factors in the decompositions. Numerous other restrictions can be considered. For instance, the Fragmentation problem bounds the *size* of the factors: Given a DFA A and $k \in \mathbb{N}$, can we decompose A into DFAs of size smaller than k? Another interesting restriction is proposed by the Compression problem, that proposes a trade-off between limiting the size and the number of the factors: Given a DFA A, can we decompose A into DFAs $(A_i)_{1 \leq i \leq k}$ satisfying $\Sigma_{i=1}^{n} |A_i| < |A|$? How do these problems compare to the ones we studied? We currently conjecture that the complexity of the Fragmentation problem matches the Decomp problem, while the complexity of the Compression problem matches the Bound-Decomp problem: for commutative permutation DFAs, the complexity seems to spike precisely when we limit the number of factors.

Chapter 5

Directions for Future Research

In this chapter, we discuss more general directions of future research. Hereby, we have limited ourselves to four areas that are particularly promising and interesting in the eyes of the author.

Completing Partial Automata to Synchronizing Automata While the synchronization problem for DFAs is solvable in polynomial time, the question whether a *partial* DFA is *carefully synchronizing*, i.e., synchronizable by a word that does not take an undefined transition on any path, is PSPACE-complete. Hence, if we do not care too much about avoiding the undefined transitions, it is beneficial to simply complete the partial automaton by defining the missing transitions such that the resulting automaton is synchronizing. This is for instance the case in the original motivation of designing parts orienters for assembly lines. In practice, some combinations of an orientation of a part and the application of a modifier is simply not possible and hence not defined in the abstract model. But as this combination can never occur in practice, we do not need to restrict our model in forbidding this configuration, instead, we just do not care about it. In order to reduce the complexity of finding a synchronizing word for the obtained model we hence want to simply define these missing transitions in a way that yields a synchronizing automaton. Unfortunately, the complexity of the problem whether a given partial DFA can be completed in a way that yields a synchronizing complete DFA seems to be a hard question to answer. So far, we were only able to answer related questions like what is the complexity of the following two problems.

Definition 46 (State-Del-Car-Syn)**.**
Given: DFA $A = (Q, \Sigma, \delta)$ and a number $k \in \mathbb{N}$.
Question: Can we remove up to k states from A (with all its incident transitions) such that the resulting automaton is carefully synchronizing?

Definition 47 (Trans-Add-Car-Syn)**.**
Given: Partial DFA $A = (Q, \Sigma, \delta)$ and a number $k \in \mathbb{N}$.
Question: Can we add up to k transitions to A labeled by letters in $\Sigma \cup \{c\}$ with $c \notin \Sigma$ such that the resulting automaton A' is a carefully synchronizing deterministic partial semi-automaton?

Both problems are PSPACE-complete. The corresponding proofs can be found in the appendix of Part I in Chapter 6. For the original setting the fact that we have no control on the target of an undefined transition makes it hard to construct any reduction. This gives hope that the problem of completing a partial DFA into a synchronizing DFA can be solved in an efficient way.

Synchronization and Diversity of Solutions For the next research direction, we stay in the picture of modeling parts orienters of assembly lines via synchronizing automata. The designer of the assembly line might have additional soft constraints to consider such as the price of a modifier or physical restrictions on the space available to place the modifiers. Modeling those restrictions might lead to much harder synchronization problems, for instance assigning the alphabet with a cost-function and searching for a synchronizing sub-automaton which is induced by a subset of the input alphabet Σ yields an NP-hard synchronization problem [Türker and Yenigün, 2015]. Restricting the set of potential synchronizing words by an auxiliary regular language might even yield a PSPACE-complete synchronization problem [Fernau et al., 2019] as well as restricting the sequence of states in which a synchronizing word transitions the automaton [Wolf, 2020]. But not all of the side constraints which are of interest in designing the assembly line are hard constraints, i.e., constraints that must be fit. They further might be only vaguely expressible and are subject to subjective assessment. Hence, a good strategy might be to not formally include them in the problem specification but instead ask for a preferably diverse set of synchronizing words from which a subjective optimal synchronizing word can then be chosen by the designer of the assembly line. This setting is the topic of a recently submitted manuscript by Arrighi, Fernau, de Oliveira Oliveira, and Wolf. The work formalizes a suitable setting of the notion of a *diverse set of solutions* for synchronizing words and analyzes the complexity of the introduced setting including FPT results, as it was recently done by the same authors in collaboration with Lokshtanov for the Kemeny rank aggregation problem in [Arrighi et al., 2021b] in the context of Social Choice.

A New Complexity Class W[Sync] The multi-parameter analysis of the synchronization problem and its variants began in [Fernau et al., 2015] where the parameterized

complexity of the *short synchronizing word* problem was analyzed.

Definition 48 (Short Sync Word)**.**
Given: DFA $A = (Q, \Sigma, \delta)$ and integer $k \in \mathbb{N}$.
Question: Is there a word $w \in \Sigma^*$ that synchronizes A and for which $|w| \leq k$?

It was shown in [Fernau et al., 2015] that the problem Short Sync Word is W[2]-hard when parameterized by k, it is NP-complete for $|\Sigma| = 2$, a result already obtained by the original NP-completeness proof by Rystsov [Rystsov, 1980] and Eppstein [Eppstein, 1990], it is further in FPT for the parameter combination k and $|\Sigma|$ or the parameter $|Q|$. The problem was also studied for subclasses of DFAs in [Bruchertseifer and Fernau, 2021]. For the natural parameter k, on general DFAs, the problem was only known to be W[2]-hard but it was open whether it is also *contained* in W[2]. Attempts to find the right parameterized complexity class for the problem raised the impression that this is a hard question and further, that the known complexity classes might not be suitable for the Short Sync Word problem with parameter k. Quite the opposite, more W[2]-hard problems, which are equivalent to Short Sync Word under fpt-reductions but are not known to be included in W[2], were identified, such as Monoid Factorization with standard parameter k and Bounded DFA Intersection with parameter k.

Definition 49 (Monoid Factorization [Cai et al., 1997])**.**
Given: A finite set Q, a collection $F = \{f_0, f_1, \ldots, f_m\}$ of mappings $f_i \colon Q \to Q$, and a positive integer $k \in \mathbb{N}$
Question: Is there a selection of at most k mappings $f_{i_1}, f_{i_2}, \ldots, f_{i_{k'}}$ with $k' \leq k$, with $i_j \in \{1, \ldots, m\}$ for $j = 1, \ldots, k'$, such that $f_0 = f_{i_1} \circ f_{i_2} \circ \cdots \circ f_{i_{k'}}$?

Definition 50 (Bounded DFA Intersection)**.**
Given: A finite set $\mathcal{A}$ of DFAs over a common alphabet Σ and a positive integer $k \in \mathbb{N}$.
Question: Is there a string $w \in \Sigma^k$ that is accepted by each DFA in $\mathcal{A}$?

This led to the definition of the complexity class W[Sync] with the Short Sync Word problem as the defining complete problem for the class under fpt-reductions [Bruchertseifer and Fernau, 2020]. The class W[Sync] is embedded in the hierarchies of known parameterized complexity classes by the inclusion relations[1]

$$\mathsf{W}[2] \subseteq \mathsf{W}[\text{Sync}] \subseteq \mathsf{WNL} \cap \mathsf{W}[P] \cap \mathsf{A}[2].$$

Other W[2]-hard problems contained in W[Sync] are Longest Common Subsequence with standard parameter k, here W[Sync]-hardness is still open, or the CSP CNF Sat-

[1] In the following we name parameterized complexity classes not defined in this work. We refer to [Flum and Grohe, 2006] for definitions and further details on these classes.

isfiablity problem with parameter k which can be reduced to Longest Common Subsequence [Fernau, 2021].

Definition 51 (Longest Common Subsequence)**.**
Given: A set of strings $\{x_1, x_2, \ldots, x_l\}$ over a common alphabet Σ and a positive integer $k \in \mathbb{N}$.
Question: Is there a string $w \in \Sigma^k$ that occurs in each string x_i for $1 \leq i \leq l$ as a subsequence?

Definition 52 (CSP CNF Satisfiablity)**.**
Given: Constraint formula φ on k variables $x_1, x_2, \ldots, x_k$ over a finite universe U, consisting of a conjunction of atomic sentences $x_i = u$ for $1 \leq i \leq k, u \in U$.
Question: Is φ satisfiable?

On the other hand, we have problems being hard for W[Sync] but which are not known to be included in W[Sync] such as the Bounded NFA Non-Universality problem with parameter k where membership in $\mathsf{A}[2] \cap \mathsf{W}[P]$ is known but even membership in WNL is open. For the related problem Bounded NFA Intersection with parameter k, W[Sync]-hardness and membership in WNL is known but membership in A[2], W[P], and hence also W[Sync] is open.

Definition 53 (Bounded NFA Non-Universality)**.**
Given: An NFA A with input alphabet Σ and a positive integer $k \in \mathbb{N}$.
Question: Is there a word $w \in \Sigma^k$ that is *not* accepted by A?

Definition 54. Bounded NFA Intersection
Given: A finite set of NFAs $\mathcal{A}$ and a positive integer $k \in \mathbb{N}$.
Question: Is there a word in Σ^k that is accepted by all NFAs in $\mathcal{A}$?

Due to the variety of problems related to Short Sync Word the field of research on the new complexity class W[Sync] is an interesting and fast growing field that aims at better understanding the structure of W-classes.

Synchronizing Quantum Finite Automata A promising and interesting field of research is the field of *Quantum Computing* that gained huge attention recently by the physical realizations of multi qubit systems by IBM [IBM, 2021] and Google [AI, 2021], among others. The field of Quantum Computing is a fast growing field and quantum analogues to well studied fields of classical computing are developed frequently. Also quantum variants of finite-state automata are introduced in the literature [Bhatia and Kumar, 2019, Ambainis and Yakaryilmaz, 2015] but in contrast to classical DFAs the

standard setting of a quantum finite automaton is not set yet. As the field of quantum finite automata is quite new and small, to the best of our knowledge, the synchronization problem has not yet been generalized to the setting of quantum finite-state automata. We will give in the following some ideas on how such a generalization could be made and leave the precise analysis of the suggested problems to future research.[2] For details on Quantum Computing and the notation used in the following (which is standard in the field of Quantum Computing), we refer to [Nielsen and Chuang, 2002].

We consider a one-way quantum finite automaton (QFA). A QFA is a 5-tuple $M = (Q, \Sigma, \{U_\sigma \mid \sigma \in \Sigma\}, q_1, R)$ where Q is a finite set of (classical) states, Σ a finite input alphabet, for each $\sigma \in \Sigma$, U_σ is a unitary transformation acting on a state in $\mathbb{C}^{|Q|}$, q_1 describes the start state, and R is an accepting condition. As we are only interested in state synchronization, we omit q_1 and R in the following. The *quantum* state of M can be any superposition of the basis states $\{|q\rangle \mid q \in Q\}$. Hence, an active state of M can be described as $|\psi\rangle = \sum_{q \in Q} \alpha_q |q\rangle$ with $\alpha_q \in \mathbb{C}$. For a word $w \in \Sigma^*$ with $|w| = n$ we denote the unitary operator associated with w as $U_w = U_{w[n]} U_{w[n-1]} \dots U_{w[1]}$.[3]

Next, we lift the concept of synchronization to quantum automata. We introduce two different concepts of synchronization.

Definition 55 (Forward Setting)**.** Let $M = (Q, \Sigma, \{U_\sigma \mid \sigma \in \Sigma\})$ be a QFA. M is *synchronizable* [*ϵ-synchronizable*] if there exists a basis state $|q\rangle$ with $q \in Q$ and a word $w \in \Sigma^*$ such that for each $p \in Q$ the probability that measuring the state $U_w |p\rangle$ against $|q\rangle$ yields $|q\rangle$ is non-zero [greater than ϵ]. In other words $\langle U_w p|q\rangle > 0$ or $\langle U_w p|q\rangle > \epsilon$, respectively.

In this setting, we can easily define an analog to Sync-Into-Subset ($\delta(Q, w) \subseteq S?$) as follows.

Definition 56 (Quantum-Sync-Into-Subset)**.**
Given: QFA $M = (Q, \Sigma, \{U_\sigma \mid \sigma \in \Sigma\})$, subset $S \subseteq Q$.
Question: Is there a word $w \in \Sigma^*$ such that for each $p \in Q$, $\sum_{s \in S} \langle U_w p|s\rangle > 0$ [$> \epsilon$]?

An analog to the Exact-Sync-Into-Subset problem ($\delta(Q, w) = S?$) can be defined as follows.

Definition 57 (Quantum-Exact-Sync-Into-Subset)**.**
Given: QFA $M = (Q, \Sigma, \{U_\sigma \mid \sigma \in \Sigma\})$, subset $S \subseteq Q$.
Question: Is there a word $w \in \Sigma^*$ such that $\forall p \in Q \colon \forall s \in S \colon \langle U_w p|s\rangle > 0$ [$> \epsilon$]?

[2] The suggested definitions were obtained in a discussion together with Mateus de Oliveira Oliveira.

[3] Note that in Quantum Computing, states are multiplied from the right. As we begin with applying the first letter of w, the unitary operators are ordered from right to left.

To formulate the standard Sync-From-Subset problem that asks whether a subset of states of an automaton can be synchronized, a backward setting seems to be more natural for defining a quantum variant of the problem.

Definition 58 (Backward Setting)**.** Let $M = (Q, \Sigma, \{U_\sigma \mid \sigma \in \Sigma\})$ be QFA. M is *synchronizable* [ϵ-*synchronizable*] if there exists a basis state $|q\rangle$ with $q \in Q$ and a word $w \in \Sigma^*$ such that for each $p \in Q$ the probability that $U_w^{-1}|q\rangle$ collapses to $|p\rangle$ (on measuring against $|p\rangle$) is non-zero [greater than ϵ].

The quantum analogue to the Sync-From-Subset ($|\delta(S, w)| = 1$) problem can then be defined as follows.

Definition 59 (Quantum-Sync-From-Subset)**.**
Given: QFA $M = (Q, \Sigma, \{U_\sigma \mid \sigma \in \Sigma\})$, subset $S \subseteq Q$.
Question: Is there a word $w \in \Sigma^*$ and a state $q \in Q$, such that for each $s \in S$, $\langle U_w^{-1} q \mid s\rangle > 0$ [$> \epsilon$]?

For the suggested definitions of synchronization problems for quantum finite sate automata, an immediate open problem is whether the forward and backward setting of the problems are equivalent. Further, it unclear how large ϵ can be in a synchronizing quantum automaton. From the perspective of this thesis, the computational complexity of these problems is one of the main questions for future research in this direction. For instance, are the suggested problems at least decidable? Is there a difference in complexity between the non-zero and greater than ϵ setting? For probabilistic automata a synchronizing word is defined in [Doyen et al., 2011] as an infinite word such that the probability mass converges in concentrating in a single state. The question whether such an infinite synchronizing word exists for a probabilistic automaton is shown to be undecidable in [Doyen et al., 2012]. This could give a hint for undecidability of the synchronization problem in our quantum setting even though we are considering only finite synchronizing words. Any progress in this direction would open a promising new direction for the study of synchronization problems.

Bibliography

[Abdulla, 2012] Abdulla, P. A. (2012). Regular model checking. *International Journal on Software Tools for Technology Transfer*, 14(2):109–118.

[AI, 2021] AI, G. Q. (2021). Explore the possibilities of quantum. `https://quantumai.google/`. accessed: 18.10.2021.

[Almeida and Klíma, 2010] Almeida, J. and Klíma, O. (2010). New decidable upper bound of the second level in the Straubing-Thérien concatenation hierarchy of star-free languages. *Discrete Mathematics and Theoretical Computer Science*, 12(4):41–58.

[Alur and Madhusudan, 2004] Alur, R. and Madhusudan, P. (2004). Visibly pushdown languages. In Babai, L., editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 202–211. ACM.

[Alur and Madhusudan, 2009] Alur, R. and Madhusudan, P. (2009). Adding nesting structure to words. *Journal of the ACM*, 56(3):16:1–16:43.

[Ambainis and Yakaryilmaz, 2015] Ambainis, A. and Yakaryilmaz, A. (2015). Automata and quantum computing. *CoRR*, abs/1507.01988.

[Ananichev and Volkov, 2004] Ananichev, D. S. and Volkov, M. V. (2004). Synchronizing monotonic automata. *Theoretical Computer Science*, 327(3):225–239.

[Arenas et al., 2011] Arenas, M., Barceló, P., and Libkin, L. (2011). Regular languages of nested words: Fixed points, automata, and synchronization. *Theory of Computing Systems*, 49(3):639–670.

[Arrighi et al., 2021a] Arrighi, E., Fernau, H., Hoffmann, S., Holzer, M., Jecker, I., de Oliveira Oliveira, M., and Wolf, P. (2021a). On the complexity of intersection non-emptiness for star-free language classes. In Bojanczyk, M. and Chekuri, C., editors, *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021, December 15-17, 2021, Virtual Conference*, volume 213 of *LIPIcs*, pages 34:1–34:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Arrighi et al., 2021b] Arrighi, E., Fernau, H., Lokshtanov, D., de Oliveira Oliveira, M., and Wolf, P. (2021b). Diversity in Kemeny rank aggregation: A parameterized approach. In Zhou, Z., editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 10–16. ijcai.org.

[Babari et al., 2016] Babari, P., Quaas, K., and Shirmohammadi, M. (2016). Synchronizing data words for register automata. In Faliszewski, P., Muscholl, A., and Niedermeier, R., editors, *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, volume 58 of *LIPIcs*, pages 15:1–15:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Baier and Katoen, 2008] Baier, C. and Katoen, J. (2008). *Principles of Model Checking*. MIT Press.

[Balasubramanian and Thejaswini, 2021] Balasubramanian, A. R. and Thejaswini, K. S. (2021). Adaptive synchronisation of pushdown automata. In Haddad, S. and Varacca, D., editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPIcs*, pages 17:1–17:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Ball and Rajamani, 2000] Ball, T. and Rajamani, S. K. (2000). Bebop: A symbolic model checker for boolean programs. In Havelund, K., Penix, J., and Visser, W., editors, *SPIN Model Checking and Software Verification, 7th International SPIN Workshop, Stanford, CA, USA, August 30 - September 1, 2000, Proceedings*, volume 1885 of *Lecture Notes in Computer Science*, pages 113–130. Springer.

[Bárány et al., 2006] Bárány, V., Löding, C., and Serre, O. (2006). Regularity problems for visibly pushdown languages. In Durand, B. and Thomas, W., editors, *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science*, volume 3884 of *Lecture Notes in Computer Science*, pages 420–431. Springer.

[Béal and Perrin, 2016] Béal, M.-P. and Perrin, D. (2016). *Synchronised Automata*, page 213–240. Encyclopedia of Mathematics and its Applications. Cambridge University Press.

[Bhatia and Kumar, 2019] Bhatia, A. S. and Kumar, A. (2019). Quantum finite automata: survey, status and research directions. *CoRR*, abs/1901.07992.

[Boasson, 1973] Boasson, L. (1973). Two iteration theorems for some families of languages. *Journal of Computer and System Sciences*, 7(6):583–596.

[Böhm and Göller, 2011] Böhm, S. and Göller, S. (2011). Language equivalence of deterministic real-time one-counter automata is NL-complete. In Murlak, F. and Sankowski, P., editors, *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*, volume 6907 of *Lecture Notes in Computer Science*, pages 194–205. Springer.

[Bollig, 2016] Bollig, B. (2016). One-counter automata with counter observability. In Lal, A., Akshay, S., Saurabh, S., and Sen, S., editors, *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016, Proceedings*, volume 65 of *LIPIcs*, pages 20:1–20:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Bouajjani et al., 1997] Bouajjani, A., Esparza, J., and Maler, O. (1997). Reachability analysis of pushdown automata: Application to model-checking. In Mazurkiewicz, A. W. and Winkowski, J., editors, *CONCUR '97: Concurrency Theory, 8th International Conference, Warsaw, Poland, July 1-4, 1997, Proceedings*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer.

[Bouajjani et al., 2000] Bouajjani, A., Jonsson, B., Nilsson, M., and Touili, T. (2000). Regular model checking. In Emerson, E. A. and Sistla, A. P., editors, *Computer Aided Verification, 12th International Conference, CAV*, volume 1855 of *LNCS*, pages 403–418. Springer.

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

[Bruchertseifer and Fernau, 2020] Bruchertseifer, J. and Fernau, H. (2020). Synchronizing words and monoid factorization: A parameterized perspective. In Chen, J., Feng, Q., and Xu, J., editors, *Theory and Applications of Models of Computation, 16th International Conference, TAMC 2020, Changsha, China, October 18-20, 2020, Proceedings*, volume 12337 of *Lecture Notes in Computer Science*, pages 352–364. Springer.

[Bruchertseifer and Fernau, 2021] Bruchertseifer, J. and Fernau, H. (2021). Synchronizing series-parallel deterministic finite automata with loops and related problems. *RAIRO-Theoretical Informatics and Applications*, 55:7.

[Brzozowski, 1976] Brzozowski, J. A. (1976). Hierarchies of aperiodic languages. *RAIRO-Theoretical Informatics and Applications*, 10(2):33–49.

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

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

[Cai et al., 1997] Cai, L., Chen, J., Downey, R. G., and Fellows, M. R. (1997). On the parameterized complexity of short computation and factorization. *Archive for Mathematical Logic*, 36(4-5):321–337.

[Caucal, 2006] Caucal, D. (2006). Synchronization of pushdown automata. In Ibarra, O. H. and Dang, Z., editors, *Developments in Language Theory, 10th International Conference, DLT 2006, Santa Barbara, CA, USA, June 26-29, 2006, Proceedings*, volume 4036 of *Lecture Notes in Computer Science*, pages 120–132. Springer.

[Černý, 1964] Černý, J. (1964). Poznámka k homogénnym experimentom s konečnými automatmi. *Matematicko-fyzikálny časopis*, 14(3):208–216.

[Chistikov et al., 2019] Chistikov, D., Martyugin, P., and Shirmohammadi, M. (2019). Synchronizing automata over nested words. *Journal of Automata, Languages and Combinatorics*, 24(2-4):219–251.

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

[Chomsky, 1956] Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124.

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

[Cook, 1971] Cook, S. A. (1971). The complexity of theorem-proving procedures. In Harrison, M. A., Banerji, R. B., and Ullman, J. D., editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM.

[de Bondt et al., 2019] de Bondt, M., Don, H., and Zantema, H. (2019). Lower bounds for synchronizing word lengths in partial automata. *International Journal of Foundations of Computer Science*, 30(1):29–60.

[de Oliveira Oliveira and Wehar, 2020] de Oliveira Oliveira, M. and Wehar, M. (2020). On the fine grained complexity of finite automata non-emptiness of intersection. In Jonoska, N. and Savchuk, D., editors, *Developments in Language Theory - 24th International Conference, DLT 2020, Tampa, FL, USA, May 11-15, 2020, Proceedings*, volume 12086 of *Lecture Notes in Computer Science*, pages 69–82. Springer.

[de Roever et al., 1998] de Roever, W. P., Langmaack, H., and Pnueli, A., editors (1998). *Compositionality: The Significant Difference, International Symposium, COMPOS'97, Bad Malente, Germany, September 8-12, 1997. Revised Lectures*, volume 1536 of *Lecture Notes in Computer Science*. Springer.

[Downey and Fellows, 1999] Downey, R. G. and Fellows, M. R. (1999). *Parameterized Complexity*. Monographs in Computer Science. Springer.

[Doyen et al., 2014] Doyen, L., Juhl, L., Larsen, K. G., Markey, N., and Shirmohammadi, M. (2014). Synchronizing words for weighted and timed automata. In Raman, V. and Suresh, S. P., editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, volume 29 of *LIPIcs*, pages 121–132. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Doyen et al., 2011] Doyen, L., Massart, T., and Shirmohammadi, M. (2011). Infinite synchronizing words for probabilistic automata. In Murlak, F. and Sankowski, P., editors, *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*, volume 6907 of *Lecture Notes in Computer Science*, pages 278–289. Springer.

[Doyen et al., 2012] Doyen, L., Massart, T., and Shirmohammadi, M. (2012). Infinite synchronizing words for probabilistic automata (erratum). *CoRR*, abs/1206.0995.

[Eppstein, 1990] Eppstein, D. (1990). Reset sequences for monotonic automata. *SIAM Journal on Computing*, 19(3):500–510.

[Esparza et al., 2003] Esparza, J., Kucera, A., and Schwoon, S. (2003). Model checking LTL with regular valuations for pushdown systems. *Information and Computation*, 186(2):355–376.

[Fernau, 2019] Fernau, H. (2019). Modern aspects of complexity within formal languages. In Martín-Vide, C., Okhotin, A., and Shapira, D., editors, *Language and Automata Theory and Applications - 13th International Conference, LATA 2019, St. Petersburg, Russia, March 26-29, 2019, Proceedings*, volume 11417 of *Lecture Notes in Computer Science*, pages 3–30. Springer.

[Fernau, 2021] Fernau, H. (2021). Some parameterized complexity results of natural combinatorial problems in automata theory and algebra. Invited talk at University of Bergen Oktober 1st, 2021.

[Fernau et al., 2019] Fernau, H., Gusev, V. V., Hoffmann, S., Holzer, M., Volkov, M. V., and Wolf, P. (2019). Computational complexity of synchronization under regular constraints. In Rossmanith, P., Heggernes, P., and Katoen, J., editors, *44th International*

Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany, volume 138 of *LIPIcs*, pages 63:1–63:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Fernau et al., 2015] Fernau, H., Heggernes, P., and Villanger, Y. (2015). A multi-parameter analysis of hard problems on deterministic finite automata. *Journal of Computer and System Sciences*, 81(4):747–765.

[Fernau and Krebs, 2017] Fernau, H. and Krebs, A. (2017). Problems on finite automata and the exponential time hypothesis. *Algorithms*, 10(1):24.

[Fernau and Wolf, 2020] Fernau, H. and Wolf, P. (2020). Synchronization of deterministic visibly push-down automata. In Saxena, N. and Simon, S., editors, *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020, December 14-18, 2020, BITS Pilani, K K Birla Goa Campus, Goa, India (Virtual Conference)*, volume 182 of *LIPIcs*, pages 45:1–45:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Fernau et al., 2020] Fernau, H., Wolf, P., and Yamakami, T. (2020). Synchronizing deterministic push-down automata can be really hard. In Esparza, J. and Král', D., editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPIcs*, pages 33:1–33:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Flum and Grohe, 2006] Flum, J. and Grohe, M. (2006). *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer.

[Gazdag et al., 2009] Gazdag, Z., Iván, S., and Nagy-György, J. (2009). Improved upper bounds on synchronizing nondeterministic automata. *Information Processing Letters*, 109(17):986–990.

[Ginsburg and Spanier, 1966] Ginsburg, S. and Spanier, E. H. (1966). Finite-turn push-down automata. *SIAM Journal on Control*, 4(3):429–453.

[Glaßer and Schmitz, 2000] Glaßer, C. and Schmitz, H. (2000). Decidable hierarchies of starfree languages. In Kapoor, S. and Prasad, S., editors, *Foundations of Software Technology and Theoretical Computer Science, 20th Conference, FST TCS*, volume 1974 of *LNCS*, pages 503–515. Springer.

[Glaßer and Schmitz, 2001] Glaßer, C. and Schmitz, H. (2001). Level 5/2 of the Straubing-Thérien hierarchy for two-letter alphabets. In Kuich, W., Rozenberg, G., and Salomaa, A., editors, *Developments in Language Theory, 5th International Conference, DLT*, volume 2295 of *LNCS*, pages 251–261. Springer.

[Greibach, 1978] Greibach, S. A. (1978). Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7:311–324.

[Gusev, 2012] Gusev, V. V. (2012). Synchronizing automata of bounded rank. In Moreira, N. and Reis, R., editors, *Implementation and Application of Automata - 17th International Conference, CIAA*, volume 7381 of *LNCS*, pages 171–179. Springer.

[Hahn et al., 2015] Hahn, M., Krebs, A., Lange, K., and Ludwig, M. (2015). Visibly counter languages and the structure of NC^1. In Italiano, G. F., Pighizzini, G., and Sannella, D., editors, *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015*, volume 9235 of *Lecture Notes in Computer Science*, pages 384–394. Springer.

[Hoffmann, 2020a] Hoffmann, S. (2020a). Computational complexity of synchronization under regular commutative constraints. In Kim, D., Uma, R. N., Cai, Z., and Lee, D. H., editors, *Computing and Combinatorics - 26th International Conference, COCOON 2020, Atlanta, GA, USA, August 29-31, 2020, Proceedings*, volume 12273 of *Lecture Notes in Computer Science*, pages 460–471. Springer.

[Hoffmann, 2020b] Hoffmann, S. (2020b). Constraint synchronization with two or three state partial constraint automata. *CoRR*, abs/2005.05907.

[Hoffmann, 2020c] Hoffmann, S. (2020c). On A class of constrained synchronization problems in NP. In Cordasco, G., Gargano, L., and Rescigno, A. A., editors, *Proceedings of the 21st Italian Conference on Theoretical Computer Science, Ischia, Italy, September 14-16, 2020*, volume 2756 of *CEUR Workshop Proceedings*, pages 145–157. CEUR-WS.org.

[Hoffmann, 2021a] Hoffmann, S. (2021a). Computational complexity of synchronization under sparse regular constraints. In Bampis, E. and Pagourtzis, A., editors, *Fundamentals of Computation Theory - 23rd International Symposium, FCT 2021, Athens, Greece, September 12-15, 2021, Proceedings*, volume 12867 of *Lecture Notes in Computer Science*, pages 272–286. Springer.

[Hoffmann, 2021b] Hoffmann, S. (2021b). Constrained synchronization and subset synchronization problems for weakly acyclic automata. In Moreira, N. and Reis, R., editors, *Developments in Language Theory - 25th International Conference, DLT 2021, Porto, Portugal, August 16-20, 2021, Proceedings*, volume 12811 of *Lecture Notes in Computer Science*, pages 204–216. Springer.

[Hoffmann, 2021c] Hoffmann, S. (2021c). Regularity conditions for iterated shuffle on commutative regular languages. In Maneth, S., editor, *Implementation and Application of Automata - 25th International Conference, CIAA 2021, Virtual Event, July*

19-22, 2021, Proceedings, volume 12803 of *Lecture Notes in Computer Science*, pages 27–38. Springer.

[Hopcroft and Ullman, 1979] Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley.

[IBM, 2021] IBM (2021). Quantum computing: tomorrow's computing, today. `https://www.ibm.com/quantum-computing/`. accessed: 18.10.2021.

[III and Rosenkrantz, 1978] III, H. B. H. and Rosenkrantz, D. J. (1978). Computational parallels between the regular and context-free languages. *SIAM Journal on Computing*, 7(1):99–114.

[Jaffe, 2006] Jaffe, A. M. (2006). The millennium grand challenge in mathematics. *Notices of the AMS*, 53(6):652–660.

[Jantzen and Kurganskyy, 2003] Jantzen, M. and Kurganskyy, A. (2003). Refining the hierarchy of blind multicounter languages and twist-closed trios. *Information and Computation*, 185(2):159–181.

[Jecker et al., 2020] Jecker, I., Kupferman, O., and Mazzocchi, N. (2020). Unary prime languages. In Esparza, J. and Král, D., editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPIcs*, pages 51:1–51:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Jecker et al., 2021] Jecker, I., Mazzocchi, N., and Wolf, P. (2021). Decomposing permutation automata. In Haddad, S. and Varacca, D., editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPIcs*, pages 18:1–18:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Karakostas et al., 2003] Karakostas, G., Lipton, R. J., and Viglas, A. (2003). On the complexity of intersecting finite state automata and NL versus NP. *Theoretical Computer Science*, 302(1-3):257–274.

[Kasai and Iwata, 1985] Kasai, T. and Iwata, S. (1985). Gradually intractable problems and nondeterministic log-space lower bounds. *Mathematical Systems Theory*, 18(2):153–170.

[Klein and Zimmermann, 2016] Klein, F. and Zimmermann, M. (2016). How much lookahead is needed to win infinite games? *Logical Methods in Computer Science*, 12(3).

[Klíma and Polák, 2011] Klíma, O. and Polák, L. (2011). Subhierarchies of the second level in the Straubing-Thérien hierarchy. *International Journal of Algebra and Computation*, 21(7):1195–1215.

[Kohavi and Jha, 2009] Kohavi, Z. and Jha, N. K. (2009). *Switching and Finite Automata Theory*. Cambridge University Press, 3rd edition.

[Kozen, 1977] Kozen, D. (1977). Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science, FOCS*, pages 254–266. IEEE Computer Society.

[Krebs et al., 2015a] Krebs, A., Lange, K., and Ludwig, M. (2015a). On distinguishing NC^1 and NL. In Potapov, I., editor, *Developments in Language Theory - 19th International Conference, DLT 2015*, volume 9168 of *Lecture Notes in Computer Science*, pages 340–351. Springer.

[Krebs et al., 2015b] Krebs, A., Lange, K., and Ludwig, M. (2015b). Visibly counter languages and constant depth circuits. In Mayr, E. W. and Ollinger, N., editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015*, volume 30 of *LIPIcs*, pages 594–607. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

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

[Kunc and Okhotin, 2013] Kunc, M. and Okhotin, A. (2013). Reversibility of computations in graph-walking automata. In Chatterjee, K. and Sgall, J., editors, *Mathematical Foundations of Computer Science 2013 - 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings*, volume 8087 of *Lecture Notes in Computer Science*, pages 595–606. Springer.

[Kupferman and Mosheiff, 2015] Kupferman, O. and Mosheiff, J. (2015). Prime languages. *Information and Computation*, 240:90–107.

[Landauer, 1961] Landauer, R. (1961). Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191.

[Lange and Rossmanith, 1992] Lange, K. and Rossmanith, P. (1992). The emptiness problem for intersections of regular languages. In Havel, I. M. and Koubek, V., editors, *Mathematical Foundations of Computer Science 1992, 17th International Symposium, MFCS'92, Prague, Czechoslovakia, August 24-28, 1992, Proceedings*, volume 629 of *Lecture Notes in Computer Science*, pages 346–354. Springer.

[Larsen et al., 2014] Larsen, K. G., Laursen, S., and Srba, J. (2014). Synchronizing strategies under partial observability. In Baldan, P. and Gorla, D., editors, *Concurrency Theory - 25th International Conference, CONCUR*, volume 8704 of *LNCS*, pages 188–202. Springer.

[Latteux, 1977] Latteux, M. (1977). Produit dans le Cône Rationnel Engendré par D. *Theoretical Computer Science*, 5(2):129–134.

[Ludwig, 2019] Ludwig, M. (2019). *Tree-Structured Problems and Parallel Computation*. PhD thesis, University of Tübingen, Germany.

[Martyugin, 2009] Martyugin, P. (2009). Complexity of problems concerning reset words for some partial cases of automata. *Acta Cybernetica*, 19(2):517–536.

[Martyugin, 2012] Martyugin, P. V. (2012). Synchronization of automata with one undefined or ambiguous transition. In Moreira, N. and Reis, R., editors, *Implementation and Application of Automata - 17th International Conference, CIAA 2012, Porto, Portugal, July 17-20, 2012. Proceedings*, volume 7381 of *Lecture Notes in Computer Science*, pages 278–288. Springer.

[Martyugin, 2014] Martyugin, P. V. (2014). Computational complexity of certain problems related to carefully synchronizing words for partial automata and directing words for nondeterministic automata. *ACM Transactions on Computer Systems*, 54(2):293–304.

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

[Masopust and Krötzsch, 2021] Masopust, T. and Krötzsch, M. (2021). Partially ordered automata and piecewise testability. *Logical Methods in Computer Science*, 17(2).

[Masopust and Thomazo, 2015] Masopust, T. and Thomazo, M. (2015). On the complexity of k-piecewise testability and the depth of automata. In Potapov, I., editor, *Developments in Language Theory - 19th International Conference, DLT 2015, Liverpool, UK, July 27-30, 2015, Proceedings*, volume 9168 of *Lecture Notes in Computer Science*, pages 364–376. Springer.

[Mehlhorn, 1980] Mehlhorn, K. (1980). Pebbling moutain ranges and its application of DCFL-recognition. In de Bakker, J. W. and van Leeuwen, J., editors, *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 422–435. Springer.

[Mikami and Yamakami, 2020] Mikami, E. and Yamakami, T. (2020). Synchronizing pushdown automata and reset words. An article appeared in Japanese as Technical Report of The Institute of Electonics, Information and Communication Engineers, COMP2019-54(2020-03), pp. 57–63.

[Myers and Urbat, 2021] Myers, R. S. R. and Urbat, H. (2021). Syntactic minimization of nondeterministic finite automata. In Bonchi, F. and Puglisi, S. J., editors, *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27, 2021, Tallinn, Estonia*, volume 202 of *LIPIcs*, pages 78:1–78:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Myhill, 1957] Myhill, J. (1957). Finite automata and the representation of events. *WADD Technical Report*, 57:112–137.

[Natarajan, 1986] Natarajan, B. K. (1986). An algorithmic approach to the automated design of parts orienters. In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 132–142. IEEE Computer Society.

[Nerode, 1958] Nerode, A. (1958). Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544.

[Netser, 2018] Netser, A. (2018). Decomposition of safe languages. Amirim Research Project report from the Hebrew University.

[Nielsen and Chuang, 2002] Nielsen, M. A. and Chuang, I. (2002). Quantum computation and quantum information.

[Okhotin and Salomaa, 2014] Okhotin, A. and Salomaa, K. (2014). Complexity of input-driven pushdown automata. *SIGACT News*, 45(2):47–67.

[Paperman, 2021] Paperman, C. (2021). Semigroup online. `https://www.paperman.name/semigroup/`. accessed: 15.10.2021.

[Parikh, 1966] Parikh, R. (1966). On context-free languages. *Journal of the ACM*, 13(4):570–581.

[Pin, 1992] Pin, J. (1992). On reversible automata. In Simon, I., editor, *LATIN '92, 1st Latin American Symposium on Theoretical Informatics, São Paulo, Brazil, April 6-10, 1992, Proceedings*, volume 583 of *Lecture Notes in Computer Science*, pages 401–416. Springer.

[Pin, 1998] Pin, J. (1998). Bridges for concatenation hierarchies. In Larsen, K. G., Skyum, S., and Winskel, G., editors, *Automata, Languages and Programming, 25th*

International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings, volume 1443 of *Lecture Notes in Computer Science*, pages 431–442. Springer.

[Place and Zeitoun, 2019] Place, T. and Zeitoun, M. (2019). Generic results for concatenation hierarchies. *ACM Transactions on Computer Systems*, 63(4):849–901.

[Rabin and Scott, 1959] Rabin, M. O. and Scott, D. S. (1959). Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125.

[Rampersad and Shallit, 2010] Rampersad, N. and Shallit, J. (2010). Detecting patterns in finite regular and context-free languages. *Information Processing Letters*, 110(3):108–112.

[Rozenberg and Salomaa, 1997] Rozenberg, G. and Salomaa, A., editors (1997). *Handbook of Formal Languages, Volume 1: Word, Language, Grammar*. Springer.

[Rystsov, 1980] Rystsov, I. K. (1980). On minimizing the length of synchronizing words for finite automata. In *Theory of Designing of Computing Systems*, pages 75–82. Institute of Cybernetics of Ukrainian Acad. Sci. (in Russian).

[Rystsov, 1983] Rystsov, I. K. (1983). Polynomial complete problems in automata theory. *Information Processing Letters*, 16(3):147–151.

[Ryzhikov, 2019] Ryzhikov, A. (2019). Synchronization problems in automata without non-trivial cycles. *Theoretical Computer Science*, 787:77–88.

[Sandberg, 2005] Sandberg, S. (2005). Homing and synchronizing sequences. In Broy, M., Jonsson, B., Katoen, J., Leucker, M., and Pretschner, A., editors, *Model-Based Testing of Reactive Systems, Advanced Lectures*, volume 3472 of *LNCS*, pages 5–33. Springer.

[Savitch, 1970] Savitch, W. J. (1970). Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192.

[Schöning, 1997] Schöning, U. (1997). *Theoretische Informatik - kurzgefaßt, 3. Auflage*. Hochschultaschenbuch. Spektrum Akademischer Verlag.

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

[Schwentick et al., 2001] Schwentick, T., Thérien, D., and Vollmer, H. (2001). Partially-ordered two-way automata: A new characterization of DA. In Kuich, W., Rozenberg, G., and Salomaa, A., editors, *Developments in Language Theory, 5th International Conference, DLT*, volume 2295 of *LNCS*, pages 239–250. Springer.

[Shirmohammadi, 2014] Shirmohammadi, M. (2014). *Qualitative Analysis of Synchronizing Probabilistic Systems. (Analyse qualitative des systèmes probabilistes synchronisants)*. PhD thesis, École normale supérieure de Cachan, France.

[Shitov, 2019] Shitov, Y. (2019). An improvement to a recent upper bound for synchronizing words of finite automata. *Journal of Automata, Languages and Combinatorics*, 24(2-4):367–373.

[Srba, 2009] Srba, J. (2009). Beyond language equivalence on visibly pushdown automata. *Logical Methods in Computer Science*, 5(1).

[Starke, 1966] Starke, P. H. (1966). Eine Bemerkung über homogene Experimente. *Elektronische Informationsverarbeitung und Kybernetik (Journal of Information Processing and Cybernetics)*, 2(4):257–259.

[Stockmeyer and Meyer, 1973] Stockmeyer, L. J. and Meyer, A. R. (1973). Word problems requiring exponential time: Preliminary report. In Aho, A. V., Borodin, A., Constable, R. L., Floyd, R. W., Harrison, M. A., Karp, R. M., and Strong, H. R., editors, *5th Annual Symposium on Theory of Computing, STOC*, pages 1–9. ACM.

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

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

[Straubing, 1994] Straubing, H. (1994). *Finite automata, formal logic, and circuit complexity*. Birkhauser Verlag.

[Swernofsky and Wehar, 2015] Swernofsky, J. and Wehar, M. (2015). On the complexity of intersecting regular, context-free, and tree languages. In Halldórsson, M. M., Iwama, K., Kobayashi, N., and Speckmann, B., editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 414–426. Springer.

[Szykuła, 2018] Szykuła, M. (2018). Improving the upper bound on the length of the shortest reset word. In Niedermeier, R. and Vallée, B., editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS*, volume 96 of *LIPIcs*, pages 56:1–56:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Thérien, 1981] Thérien, D. (1981). Classification of finite monoids: the language approach. *Theoretical Computer Science*, 14(2):195–208.

[Thierrin, 1968] Thierrin, G. (1968). Permutation automata. *Mathematical Systems Theory*, 2(1):83–90.

[Truthe and Volkov, 2019] Truthe, B. and Volkov, M. V. (2019). Journal of Automata, Languages and Combinatorics – Essays on the Černý Conjecture. `https://www.jalc.de/issues/2019/issue_24_2-4/content.html`. Accessed: 10/1/2020.

[Türker and Yenigün, 2015] Türker, U. C. and Yenigün, H. (2015). Complexities of some problems related to synchronizing, non-synchronizing and monotonic automata. *International Journal of Foundations of Computer Science*, 26(1):99–122.

[Valiant, 1973] Valiant, L. G. (1973). *Decision Procedures for Families of Deterministic Pushdown Automata*. PhD thesis, University of Warwick, Coventry, UK.

[Volkov, 2008] Volkov, M. V. (2008). Synchronizing automata and the Černý conjecture. In Martín-Vide, C., Otto, F., and Fernau, H., editors, *Language and Automata Theory and Applications, Second International Conference, LATA*, volume 5196 of *LNCS*, pages 11–27. Springer.

[Vorel and Roman, 2015] Vorel, V. and Roman, A. (2015). Parameterized complexity of synchronization and road coloring. *Discrete Mathematics and Theoretical Computer Science*, 17(1):283–306.

[Wareham, 2000] Wareham, T. (2000). The parameterized complexity of intersection and composition operations on sets of finite-state automata. In Yu, S. and Paun, A., editors, *Implementation and Application of Automata, 5th International Conference, CIAA 2000, London, Ontario, Canada, July 24-25, 2000, Revised Papers*, volume 2088 of *Lecture Notes in Computer Science*, pages 302–310. Springer.

[Wehar, 2014] Wehar, M. (2014). Hardness results for intersection non-emptiness. In Esparza, J., Fraigniaud, P., Husfeldt, T., and Koutsoupias, E., editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 354–362. Springer.

[Wehar, 2017] Wehar, M. (2017). *On the complexity of intersection non-emptiness problems*. PhD thesis, State University of New York at Buffalo.

[Wolf, 2020] Wolf, P. (2020). Synchronization under dynamic constraints. In Saxena, N. and Simon, S., editors, *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020, December 14-18, 2020, BITS Pilani, K K Birla Goa Campus, Goa, India (Virtual Conference)*, volume 182 of *LIPIcs*, pages 58:1–58:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Chapter 6

Appendix of Part I

Results Concerning the Research Direction: Completing Partial Automata to Synchronizing Automata

A complete DFA A is called *synchronizing* if there exists a word $w \in \Sigma^*$ such that $|\delta(Q, w)| = 1$. In this case we call w a *synchronizing word* for A. A partial DFA A is called *carefully synchronizing* if there exists a word $w \in \Sigma^*$ such that $\delta(q, w)$ is defined for all $q \in Q$ and $|\delta(Q, w)| = 1$. Here, we call w a *carefully synchronizing word* for A. For some word $w \in \Sigma^*$ we call $\delta(Q, w)$ the set of active states (concerning w). If for some state q every letter maps q to itself, we call q a *trap-state*.

Definition 60 (Sync-Into-Subset)**.**
Given: DFA $A = (Q, \Sigma, \delta)$ and a subset $S \subseteq Q$.
Question: Is there a word $w \in \Sigma^*$ such that $\delta(Q, w) \subseteq S$?

Definition 61 (State-Del-Car-Syn)**.**
Given: DFA $A = (Q, \Sigma, \delta)$ and a number $k \in \mathbb{N}$.
Question: Can we remove up to k states from A (with all its incident transitions) such that the resulting automaton is carefully synchronizing?

Theorem 6. *The problem* State-Del-Car-Syn *is PSPACE-complete.*

Proof. We give a reduction from the PSPACE-complete problem Sync-Into-Subset. The construction is depicted in Figure 6.1. Let $A = (Q, \Sigma, \delta)$ be a complete DFA and $S \subseteq Q$. We construct from A a complete DFA $A' = (Q', \Sigma', \delta')$ with $\Sigma' := \Sigma \cup \{c, d\}$

where $\{c, d\} \cap \Sigma = \emptyset$. We start with $Q' := Q$ and $\delta' := \delta$. For every state $q \in Q \backslash S$ we create two new states q_c and q_d. We define $\delta'(q_c, \sigma) := q_c$ and $\delta'(q_d, \sigma) := q_d$ for all $\sigma \in \Sigma'$. Further, we set $\delta'(q, c) := q_c$ and $\delta'(q, d) := q_d$. We create two additional states s_c and s_d and for all states $q \in S$ we set $\delta'(q, c) := s_c$ and $\delta'(q, d) := s_d$ for all $\sigma \in \Sigma'$. Finally, we set $\delta'(s_d, \sigma) = \delta'(s_c, \sigma) := s_c$ and $k := 2|(Q \backslash S)|$

First, assume there exists a word $w \in \Sigma^*$ such that $\delta(Q, w) \subseteq S$ in the automaton A. Then, we pick the $2|(Q \backslash S)|$ states q_c and q_d with $q \in Q \backslash S$ for deletion. Let A'' be the resulting automaton. For all states in $Q \backslash S$ the transitions labeled with c and d are now undefined in A''. Note that s_c is the only remaining trap state and that w is still defined on all remaining states. Since δ' agrees with δ on $Q \times \Sigma$, the word w also brings all states from Q into S in the automaton A''. As for all states in S the transition c is still defined, it is easy to see that the word wc is carefully synchronizing A'' into the state s_c.

For the other direction assume there exists a set of states $D \subseteq Q'$ with $|D| \leq k$ such that removing the states in D and all incident transitions from A' yields a partial DFA A'' which is carefully synchronized by some word $w \in \Sigma'^*$. The automaton A'' contains $2|(Q \backslash S)|+1$ trap states and $2|(Q \backslash S)|+2$ states from which no state in Q can be reached. In order to obtain a synchronizing automaton we need to remove at least all but one trap states since otherwise two states which can not be synchronized would remain. The threshold k forces us to keep one trap-state. If we choose one of the states q_c or q_d for $q \in Q \backslash S$ as the remaining trap state, then either s_c or s_d will remain. If we delete s_c and keep s_d the state s_d will have no defined transition left and especially the remaining other trap-state can not be reached from s_d. If we delete s_d and keep s_c we would also end up with two different trap states. Hence, we need to delete all $2|(Q \backslash S)|$ states q_c and q_d with $q \in Q \backslash S$ and keep the states s_c and s_d. Since s_c is the only remaining trap state it needs to be the synchronizing state and all states in Q must reach s_c. The only way to reach s_c is to read either a letter c or d at some point. But the transitions c and d are no longer defined on all states in $Q \backslash S$. In order to read one of these letters in a carefully synchronizing word $w = w_1 c w_2$ or $w' = w_1 d w_2$ with $w_1 \in \Sigma^*$, the subword w_1 must already map all states into S. Hence, $\delta(Q, w_1) \subseteq S$ and since δ' agrees with δ, the word w_1 also maps all states in A into the set S.

The membership in PSPACE follows from the fact that we can guess which states we delete (NPSPACE = PSPACE [Savitch, 1970]) and then test the resulting automaton for careful synchronization in polynomial space [Martyugin, 2014]. □

Definition 62 (Trans-Add-Car-Syn)**.**
Given: Partial DFA $A = (Q, \Sigma, \delta)$ and a number $k \in \mathbb{N}$.
Question: Can we add up to k transitions to A labeled by letters in $\Sigma \cup \{c\}$ with $c \notin \Sigma$

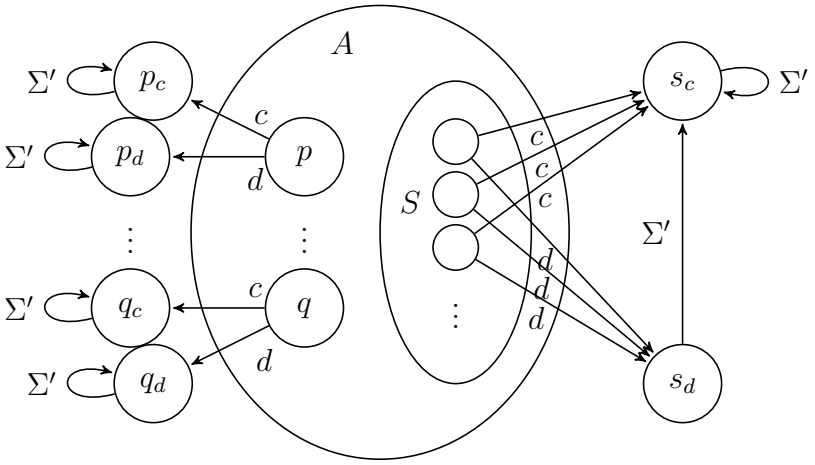


Figure 6.1: Schematic illustration of the reduction from Sync-Into-Subset to State-Del-Car-Syn (see Theorem 6).

such that the resulting automaton A' is a carefully synchronizing deterministic partial semi-automaton?

Theorem 7. *For every fixed $k \in \mathbb{N}$, the problem k-Trans-Add-Car-Syn is PSPACE-complete.*

Proof. Let $k \in \mathbb{N}$ be fixed. Then, we give a reduction from the PSPACE-complete problem Careful Synchronization [Martyugin, 2014]. Let $A = (Q, \Sigma, \delta)$ be a partial DFA. We construct from A the partial DFA $A' = (Q \cup \{q_1, q_2, \ldots, q_k\}, \Sigma \cup \{c\}, \delta)$ with $Q \cap \{q_1, q_2, \ldots, q_k\} = \emptyset$ and $c \notin \Sigma$. Note, that the transition function δ is completely taken over from A and the newly added states $q_1, q_2, \ldots, q_k$ have no defined transition. Then, the instance A', k is a yes-instance of Trans-Add-Car-Syn if and only if A is carefully synchronizing.

First, assume A is carefully synchronizing by a carefully synchronizing word $w \in \Sigma^*$. Then, there must be a letter $a \in \Sigma$ for which $\delta(Q, a)$ is defined. Pick some state $q \in Q$ for which some state maps to q with the letter a. Adding the transitions $\delta(q_1, a) = \delta(q_2, a) = \cdots = \delta(q_k, a) = q$ to A' yields a deterministic partial semi-automaton which is carefully synchronized by w.

For the other direction assume we can add up to k transitions to A' such that the resulting automaton is a carefully synchronizing deterministic partial semi-automaton. Since the states $q_1, q_2, \ldots, q_k$ cannot be reached from the states in Q, and are further no trap-states, we must bring all of these states into Q by adding transitions. As there are k pairwise not connected states $q_1, q_2, \ldots, q_k$ we need all of the k transitions to bring them into Q. As there are no transitions left to add between two states in Q, any carefully

synchronizing word for the resulting automaton is also carefully synchronizing for A.

Membership in PSPACE follows from the fact, that we can guess the added transitions and then test the resulting automaton for carefully synchronization in polynomial space. □

Part II

Publications

Chapter 7

Computational Complexity of Synchronization under Regular Constraints

Henning Fernau, Vladimir V. Gusev, Stefan Hoffmann, Markus Holzer, Mikhail V. Volkov, and Petra Wolf.

An extended abstract appeared in the proceedings of MFCS 2019:
Leibniz International Proceedings in Informatics (LIPIcs) 138 (2019) pp. 63:1 – 63:14.
DOI: 10.4230/LIPIcs.MFCS.2019.63.

Computational Complexity of Synchronization under Regular Constraints

Henning Fernau[1], Vladimir V. Gusev*[2], Stefan Hoffmann[1], Markus Holzer[3], Mikhail V. Volkov†[4], and Petra Wolf‡[1]

[1]Universität Trier, Germany
[2]University of Liverpool, UK
[3]Universität Gießen, Germany
[4]Ural Federal University, Yekaterinburg, Russia

Abstract

Many variations of synchronization of finite automata have been studied in the previous decades. Here, we suggest studying the question if synchronizing words exist that belong to some fixed constraint language, given by some partial finite automaton called constraint automaton. We show that this synchronization problem becomes PSPACE-complete even for some constraint automata with two states and a ternary alphabet. In addition, we characterize constraint automata with arbitrarily many states for which the constrained synchronization problem is polynomial-time solvable. We classify the complexity of the constrained synchronization problem for constraint automata with two states and two or three letters completely and lift those results to larger classes of finite automata.

1 Introduction

Synchronization is an important concept for many applied areas: parallel and distributed programming, system and protocol testing, information coding, robotics, etc. At least

*The author was supported by Leverhulme Trust
†The author was supported by DFG-funded project FE560/9-1
‡The author was supported by DFG-funded project FE560/9-1

some aspects of synchronization are captured by the notion of a synchronizing automaton; for instance, synchronizing automata adequately model situations in which one has to direct a certain system to a particular state without a priori knowledge of its current state. We only refer to some survey papers [Sandberg, 2005, Volkov, 2008], as well as to Chapter 13 in [Kohavi and Jha, 2009], that also report on some of these applications. An automaton is called synchronizing if there exists a word that brings it to a known state independently of the starting state. This concept is quite natural and has been investigated intensively in the last six decades. It is related to the arguably most famous open combinatorial question in automata theory, formulated by Černý in [Černý, 1964]. The Černý conjecture states that every n-state synchronizing automaton can be synchronized by a word of length smaller or equal $(n-1)^2$. Although this bound was proven for several classes of finite-state automata, such as aperiodic automata [Trakhtman, 2007] or automata with a transition monoid in a class called DS [Almeida et al., 2009], the general case is still widely open. The currently best upper bound on this length is cubic, and only very little progress has been made, basically improving on the multiplicative constant factor in front of the cubic term, see [Shitov, 2019, Szykuła, 2018].

Due to the importance of this notion of synchronizing words, quite a large number of generalizations and modifications have been considered in the literature. We only mention four of these in the following. Instead of synchronizing the whole set of states, one could be interested in synchronizing only a subset of states. This and related questions were first considered by Rystsov in [Rystsov, 1983]. Instead of considering deterministic finite automata (DFAs), one could alternatively study the notion of synchronizability for nondeterministic finite automata [Gazdag et al., 2009, Martyugin, 2014]. The notion of synchronizability naturally transfers to partially defined transition functions where a synchronizing automata avoiding undefined transitions is called *carefully synchronizing*, see [de Bondt et al., 2019, Martyugin, 2009, Martyugin, 2014]. To capture more adaptive variants of synchronizing words, synchronizing strategies have been introduced in [Larsen et al., 2014]. Recall that the question of synchronizability (without length bounds) is solvable in polynomial time for complete DFAs [Volkov, 2008]. However, in all of the mentioned generalizations, this synchronizability question becomes even PSPACE-complete. This general tendency can also be observed in the generalization that we introduce in this paper, which we call *regular constraints*. These constraints are defined by some (fixed) finite automaton describing a regular language R, and the question is, given some DFA A, if A has some synchronizing word from R. This notion explicitly appeared in [Gusev, 2012] as an auxiliary tool: it was shown that the synchronization problem of every automaton $A = (\Sigma, Q, \delta)$ whose letters σ have ranks at most r, i.e., $|\delta(Q, \sigma)| \leq r$, is equivalent to the synchronization of an r-state automaton A' under some regular constraints.

The main research question that we look into is to understand for which regular constraints the question of synchronizability is solvable in polynomial time (as it is for $R = \Sigma^*$), or for which it is hard. Furthermore, it would be interesting to see complexity classes different from P and PSPACE to show up (depending on R). In our paper, we give a complete description of the complexity status for constraints that can be described by partial 2-state deterministic automata on alphabets with at most three letters. In this case, indeed, we only observe P and PSPACE situations. However, we also find 3-state automata (on binary input alphabets) that exhibit an NP-complete synchronization problem when considered as constraints. We describe several ways how to generalize our results to larger constraint automata. Moreover, we identify several classes of constraint automata that imply feasible synchronization problems. We motivate our study of synchronization under regular constraints by the following example.

A motivating result. In the theory of synchronizing automata, one normally allows the directing instruction to be an arbitrary word over the input language of the corresponding automaton. In reality, however, available commands might be subject to certain restrictions; for instance, it is quite natural to assume that a directing instruction should always start and end with a specific command that first switches the automaton to a 'directive' mode and then returns the automaton to its usual mode. In its simplest form, the switching between 'normal mode' and 'directive' (synchronization) mode can be modeled as ab^*a. This scenario produces an NP-complete synchronization problem. In order to state our first result formally, we make use of some (standard) notions defined in Section 2.

Proposition 1. *The following problem is NP-complete: Given a deterministic finite complete automaton A with $L(A) \subseteq \{a, b\}^*$, is there a synchronizing word $w \in ab^*a$ for A?*

Notice that this contrasts with the complexity of the synchronizability question for complete DFAs, which can be solved in quadratic time; see, e.g., [Sandberg, 2005, Volkov, 2008]. Also, it contrasts the complexity of synchronizability for partial DFAs, which is PSPACE-complete; see [Martyugin, 2014].

The constraint automaton describing ab^*a has three states. As we will see below, only P- and PSPACE-results can be observed for two-state constraint automata over binary and ternary input alphabets. Hence, in a sense, Proposition 1 is a minimal example of a complexity status inbetween P and PSPACE. A proof sketch of Proposition 1 is given below.

Rystsov [Rystsov, 1983] considered a problem that he called Global Inclusion Problem for Non-initial Automata. As we will see below, this problem (together with

a variation) will be the key problem in our reductions. Looking at the proof of [Rystsov, 1983, Theorem 2.1], we can observe the following refined result. We consider the next problem that we call $\mathcal{P}_\Sigma$ for brevity. Given a complete DFA A with state set Q and input alphabet Σ, with $a \in \Sigma$, as well as a designated state subset S, is there some word $w \in \{a\}(\Sigma \setminus \{a\})^*$ such that w drives A into S, irrespectively of where A starts processing w? Trivially, $\mathcal{P}_\Sigma$ is in P if $|\Sigma| = 1$.

Theorem 1. *$\mathcal{P}_\Sigma$ is NP-hard if $|\Sigma| = 2$, and PSPACE-hard if $|\Sigma| > 2$.*

In particular, the case distinction between binary input alphabets and larger input alphabets (concerning hardness results) comes from the fact that the reduction of Rystsov uses DFA-Intersection Nonemptiness, the non-emptiness of intersection problem for deterministic finite automata on the alphabet $\Sigma \setminus \{a\}$. According to Theorem 6.1 in [Stockmeyer and Meyer, 1973] (more details in [Fernau and Krebs, 2017, Kozen, 1977]) this problem is NP-complete on unary alphabets. In Rystsov's reduction, the state set of the automaton A consists of a part $Q_\cap$, which just copies the n automata A_i (over alphabet $\Sigma \setminus \{a\}$) of a DFA-Intersection Nonemptiness instance, together with n new states $t_i \in Q_\to$ that move on input a into the initial state s_i of A_i. Likewise, from any state q_i of A_i, letter a leads to s_i. All transitions not yet defined are self-loops. Set S collects all final states of all A_i. Hence, a word $w \in (\Sigma \setminus \{a\})^*$ is accepted by all of the A_i iff aw drives A into S, starting out from any state. The promised proof sketch follows. Modify A to obtain an automaton A' such that A' has a synchronizing word awa, with $w \in (\Sigma \setminus \{a\})^*$ iff aw drives A into S as follows: add a new state s where all letters loop; for all $q \in S$, replace the a-transitions leading from q into s_i by a-transitions leading into s. For more details (membership in NP for $\Sigma = \{a, b\}$ is non-trivial), see Theorem 6.

2 Preliminaries and Definitions

Throughout the paper, we consider deterministic finite automata (DFAs). Recall that a DFA A is a tuple $A = (\Sigma, Q, \delta, q_0, F)$, where the alphabet Σ is a finite set of input symbols, Q is the finite state set, with start state $q_0 \in Q$, and final state set $F \subseteq Q$. The transition function $\delta : Q \times \Sigma \to Q$ extends to words from Σ^* in the usual way. The function δ can be further extended to sets of states in the following way. For every set $S \subseteq Q$ with $S \neq \emptyset$ and $w \in \Sigma^*$, we set $\delta(S, w) := \{\, \delta(q, w) \mid q \in S \,\}$. We sometimes refer to the function δ as a relation and we identify a transition $\delta(q, \sigma) = q'$ with the tuple (q, σ, q'). We call A *complete* if δ is defined for every $(q, a) \in Q \times \Sigma$; if δ is undefined for some (q, a), the automaton A is called *partial*. If $|\Sigma| = 1$, we call A a *unary* automaton. The set $L(A) = \{\, w \in \Sigma^* \mid \delta(q_0, w) \in F \,\}$ denotes the language accepted by A. A semi-

automaton is a finite automaton without a specified start state and with no specified set of final states. Notice that $(\Sigma, \binom{Q}{\leq k}, \delta)$ can be viewed as a semi-automaton for each $k \leq |Q|$, when $\binom{Q}{\leq k}$ is the set formed by all subsets of Q of cardinality at most k. The properties of being *deterministic*, *partial*, and *complete* of semi-automata are defined as for DFA. When the context is clear, we call both deterministic finite automata and semi-automata simply *automata*. We call a deterministic complete semi-automaton a DCSA and a partial deterministic finite automaton a PDFA for short. If we want to add an explicit initial state r and an explicit set of final states S to a DCSA A or change them in a DFA A, we use the notation $A_{r,S}$.

An automaton A is called *synchronizing* if there exists a word $w \in \Sigma^*$ with $|\delta(Q, w)| = 1$. In this case, we call w a *synchronizing word* for A. For a word w, we call a state in $\delta(Q, w)$ an *active* state. We call a state $q \in Q$ with $\delta(Q, w) = \{q\}$ for some $w \in \Sigma^*$ a *synchronizing state*. A state from which some final state is reachable is called *co-accessible*. For a set $S \subseteq Q$, we say S is *reachable* from Q or Q is synchronizable to S if there exists a word $w \in \Sigma^*$ such that $\delta(Q, w) = S$. An automaton A is called *returning*, if for every state $q \in Q$, there exists a word $w \in \Sigma^*$ such that $\delta(q, w) = q_0$, where q_0 is the start state of A.

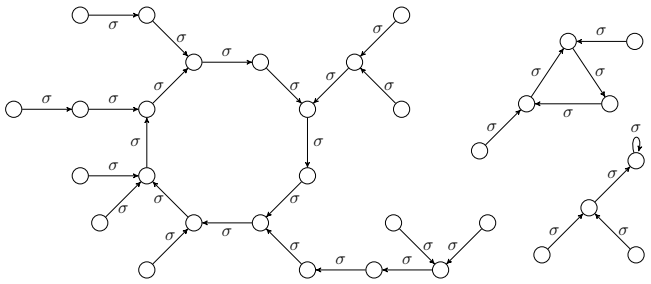
Fact 1. [Volkov, 2008] For any DCSA, we can decide if it is synchronizing in polynomial time $O(|\Sigma||Q|^2)$. Additionally, if we want to compute a synchronizing word w, then we need time $O(|Q|^3 + |Q|^2|\Sigma|))$ and the length of w will be $O(|Q|^3)$.

The following obvious remark will be used frequently without further mentioning.

Lemma 1. *Let $A = (\Sigma, Q, \delta)$ be a DCSA and $w \in \Sigma^*$ be a synchronizing word for A. Then for every $u, v \in \Sigma^*$, the word uwv is also synchronizing for A.*

For an automaton A over the alphabet Σ, we denote by $A_{\Sigma'}$ for every $\Sigma' \subset \Sigma$ the restriction of A to the alphabet Σ'. Automaton $A_{\Sigma'}$ is obtained from A by deleting all transitions with labels in $\Sigma \setminus \Sigma'$. We will identify $A_{\{\sigma\}}$ with A_σ for every $\sigma \in \Sigma$. For a complete deterministic automaton A_σ, each connected component of A_σ consists of exactly one cycle and some tails leading into the cycle (see Figure 1a). A cycle is a sequence of states $q_1, q_2, \ldots, q_k$, for $k \in \mathbb{N}$ such that $\delta(q_i, \sigma) = q_{i+1}$ and $\delta(q_k, \sigma) = q_1$. In particular, a cycle may consist of one single state only. The tails are only leading into the cycle since A is deterministic. We call components of this form *sun-structures* as illustrated in Figure 1a.

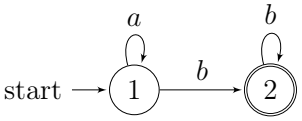
We call two automata A and A' isomorphic if one automaton can be obtained from the other one by renaming states and alphabet-symbols. Notice that the number of non-isomorphic automata can be quite huge even for small number of states and alphabet



(a) Unary automaton A_σ consisting of 3 sun-structures.

Automaton notation (code)
`[ 1 2 ‡ - 2 ]`

PDFA



(b) Automaton notation and PDFA using its standard interpretation (see Section 4).

Figure 1: Illustration of sun-structures and of the notation of PDFAs.

sizes; see [Bassino and Nicaud, 2007, Domaratzki et al., 2002, Harrison, 1965]. In order to address the presented automata in a compact way, we introduce a short notation motivated by [Almeida et al., 2007, Ananichev et al., 2013, Reis et al., 2009], where we assume some order is given on the alphabet and on the state set. Each automaton is denoted by a tuple of size $|Q|\cdot|\Sigma|$ where for each state the mapping of this state with each alphabet symbol is listed. The states themselves are separated by ‡-signs. For example, the first entry of the tuple denotes the transition of the first state under the first symbol (and "`-`" for an undefined transition), while the second entry denotes the transition of the first state by the second symbol, and so on. We will always assume the first state in the ordering of the states to be the start state of the automaton. See Figure 1b for an example. Final states are not part of this coding. We will further summarize notions of automata by a set notation in the description of a transition. For instance, the tuple `[ 1 2 ‡ {-,2} - ]` denotes the set of automata `[ 1 2 ‡ - - ]` and `[ 1 2 ‡ 2 - ]`. Further, a star `*` in the tuple notation denotes all possible realizations of a transition.

For a fixed PDFA $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$, we define the *constrained synchronization problem*:

Definition 1. $L(\mathcal{B})$-Constr-Sync
Input: DCSA $A = (\Sigma, Q, \delta)$.
Question: Is there a synchronizing word w for A with $w \in L(\mathcal{B})$?

The automaton $\mathcal{B}$ will be called the *constraint automaton*. If an automaton A is a yes-instance of $L(\mathcal{B})$-Constr-Sync we call A *synchronizing with respect to* $\mathcal{B}$. Occasionally, we do not specify $\mathcal{B}$ and rather talk about L-Constr-Sync. We are going to inspect the complexity of this problem for different (small) constraint automata. We assume the reader to have some basic knowledge in computational complexity theory and formal language theory, as contained, e.g., in [Hopcroft et al., 2001]. For instance, we make use of regular expressions to describe languages. We also identify singleton sets with its elements. We make use of complexity classes like P, NP, or PSPACE. At one point,

we also mention the parameterized complexity class XP. By $\leq_m^{\log}$ we denote a logspace many-one reduction. If for two problems L_1, L_2 it holds that $L_1 \leq_m^{\log} L_2$ and $L_2 \leq_m^{\log} L_1$, then we write $L_1 \equiv_m^{\log} L_2$.

For establishing some of our results, we need the following computational problems taken from [Berlinkov et al., 2018], which are PSPACE-complete problems for at least binary alphabets, also see [Rystsov, 1983, Sandberg, 2005].

Definition 2. Sync-From-Subset
Input: DCSA $A = (\Sigma, Q, \delta)$ and $S \subseteq Q$.
Question: Is there a word w with $|\delta(S, w)| = 1$?

Definition 3. Sync-Into-Subset
Input: DCSA $A = (\Sigma, Q, \delta)$ and $S \subseteq Q$.
Question: Is there a word w with $\delta(Q, w) \subseteq S$?

Remark 1. The terminology is not homogeneous in the literature. For instance, Sync-Into-Subset has different names in [Berlinkov et al., 2018] and in [Rystsov, 1983].

3 Placing Constrained Problems Within Complexity Classes

In this section we present several criteria for L which lead to the membership of L-Constr-Sync in different complexity classes, starting by studying unary languages.

Lemma 2. *Let $A = (\{\sigma\}, Q, \delta)$ be a unary synchronizing DCSA. For all $i \geq |Q| - 1$, we have $\delta(Q, \sigma^i) = \delta(Q, \sigma^{i+1})$. A shortest word w synchronizing $S \subseteq Q$ obeys $|w| \leq |Q| - 1$.*

Proof. Since A is a unary DCSA, it consists of sun-structures; see Fig. 1a. Each sun-structure of A consists of exactly one cycle and some tails leading into the cycle. There exists a cycle in A, since A is complete. No transition is leading *out of* the cycle, since A is deterministic and unary. Let $n = |Q|$. Then, for each $q \in Q$, the state $\delta(q, \sigma^{n-1})$ is part of a cycle. Hence, $\delta(Q, \sigma^{n-1}) = \delta(Q, \sigma^n) = \delta(Q, \sigma^{n+j})$ for $j \geq 0$.

If S contains states from several sun-structures of A, then there is no word w with $|\delta(S, w)| = 1$. Therefore, consider only the sun-structure containing S. Let c denote the number of states in its cycle. For each state $q \in Q$, after at most $|Q| - 1 - c$ transitions, we are in the cycle. Hence, for $k \geq |Q| - 1 - c$ we have $\delta(q, \sigma^k) = \delta(q, \sigma^{k+c})$. So for $k \geq |Q| - 1$, using the equation $\delta(S, \sigma^k) = \delta(S, \sigma^{k-c})$ iteratively gives our claim. □

Corollary 1. *If $L(\mathcal{B}) \subseteq \{\sigma\}^*$ for a PDFA $\mathcal{B}$, then $L(\mathcal{B})$-Constr-Sync $\in$ P.*

Proof. A synchronizing word for an DCSA A with respect to L can only contain σ's. Hence, A can be reduced to A_σ. According to Lemma 2 the length of a shortest synchronizing word w for A is linearly bounded in the size of A. Extending w to a word in $L(\mathcal{B})$ only adds linearly in $\mathcal{B}$ many letters. Since $\mathcal{B}$ is constant, the number of candidates for synchronizing words for A with respect to $\mathcal{B}$ is linearly bounded in A. □

Theorem 2. *If L is regular, then L-Constr-Sync is contained in PSPACE.*

Proof. As L is regular, there is some DFA $\mathcal{B}$ accepting L. The NPSPACE-machine that checks if the DCSA $A = (\Sigma, Q, \delta)$ has a synchronizing word $w \in L$ guesses w letter-by-letter and keeps track both of the set of active states of A and of the current state of $\mathcal{B}$. This information can be stored in linear space (in the size of A). As NPSPACE = PSPACE by [Savitch, 1970], the claim follows. □

We continue with 1-state constraint automata and unions of constraint languages.

Lemma 3. *Let $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$ be a PDFA. If $L(\mathcal{B}) = L(\mathcal{B}_{\Sigma\setminus\{\sigma\}})$ for some $\sigma \in \Sigma$, then $L(\mathcal{B})$-Constr-Sync $\equiv_m^{\log}$ $L(\mathcal{B}_{\Sigma\setminus\{\sigma\}})$-Constr-Sync.*

Proof. Let $A = (\Sigma, Q, \delta)$ be a DCSA. Since every synchronizing word for A with respect to $\mathcal{B}$ must not contain any σ-letter, A is synchronizing with respect to $\mathcal{B}$ if and only if $A_{\Sigma\setminus\{\sigma\}}$ is synchronizing with respect to $\mathcal{B}_{\Sigma\setminus\{\sigma\}}$. The other reduction is simply the identity. □

Corollary 2. *$L(\mathcal{B})$-Constr-Sync $\in$ P for every one-state constraint automaton $\mathcal{B}$.*

Proof. Let $A = (\Sigma, Q, \delta)$ be a DCSA. We can reduce the alphabet Σ of $\mathcal{B}$ to Σ' consisting of those letters which actually appear in $L(\mathcal{B})$ by using Lemma 3 repeatedly. Since $\mathcal{B}$ is a one-state automaton, $L(\mathcal{B}')$ is either empty or Σ'^*. The former case is trivial. For the latter case, it is sufficient to test if $A_{\Sigma'}$ is synchronizing at all. □

Lemma 4. *If L is a finite union of languages $L_1, L_2, \ldots, L_n$ such that for each $1 \leq i \leq n$ the problem L_i-Constr-Sync $\in$ P, then L-Constr-Sync $\in$ P.*

Proof. Checking synchronizability of a DCSA $A = (\Sigma, Q, \delta)$ with respect to language L can be done by checking synchronizability of A with respect to each L_i. Since the problem L_i-Constr-Sync is decidable in polynomial time for each $1 \leq i \leq n$, computing $\bigvee_{i=1}^{n} A \in L_i$-Constr-Sync yields a polynomial-time algorithm for the problem L-Constr-Sync. □

Theorem 3. *Let $L \subseteq \Sigma^*$. If $\{ v \in \Sigma^* \mid \exists u, w \in \Sigma^* : uvw \in L \} = \Sigma^*$, then L-Constr-Sync is solvable in polynomial time.*

Proof. Let $A = (\Sigma, Q, \delta)$ be a DCSA. To decide if A has a synchronizing word from L, simply test if A is synchronizing at all, confer Fact 1. Assume $v \in \Sigma^*$ is a synchronizing word for A. By assumption, $uvw \in L$ for some $u, w \in \Sigma^*$. Moreover, the word uvw also synchronizes A. □

With the same type of reasoning, one can show:

Theorem 4. *Let $L \subseteq L' \subseteq \Sigma^*$. If $L' \subseteq \{ v \in \Sigma^* \mid \exists u, w \in \Sigma^* : uvw \in L \}$, then L-Constr-Sync $\equiv_m^{\log}$ L'-Constr-Sync.*

Remark 2. Considering a PDFA $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$, we conclude: (a) If $\mathcal{B}$ is complete and each state is co-accessible, then $L(\mathcal{B})$-Constr-Sync $\in \mathsf{P}$. (b) If P' is the set of reachable states and if $L(\mathcal{B}_{P',F}) = \Sigma^*$, then $L(\mathcal{B})$-Constr-Sync $\in \mathsf{P}$. (c) If $p \in P$ is co-accessible, then $L(\mathcal{B})$-Constr-Sync $\equiv_m^{\log}$ $L(\mathcal{B}_{p_0,F\cup\{p\}})$-Constr-Sync.

In [Volkov, 2008], the unconstrained synchronization problem can be decided in polynomial time by verifying that every pair of states can be synchronized. In essence, we generalize this algorithm here for returning constraint automata.

Lemma 5. *Let $A = (\Sigma, Q, \delta)$ be a DCSA. If the constraint automaton $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$ is returning, then A is synchronizing with respect to $\mathcal{B}$ if and only if for all $q, q' \in Q$ we find $w \in \Sigma^*$ with $\mu(p_0, w) = p_0$ such that $\delta(q, w) = \delta(q', w)$.*

Proof. Let $\mathcal{B}$ be a returning automata. Assume A is synchronizing with respect to $\mathcal{B}$. Let $u \in \Sigma^*$ be a synchronizing word for A with $u \in L(\mathcal{B})$. Therefore $|\delta(Q, u)| = 1$ and $\delta(q, u) = \delta(q', u)$ for every pair of states $q, q' \in Q$. Assume $\delta(p_0, u) = p_f \in F$. Since $\mathcal{B}$ is returning, there exists a word $v \in \Sigma^*$ with $\delta(p_f, v) = p_0$. Hence $w = uv$ fulfills the condition.

For the other direction, the reasoning is similar to the synchronization problem in the unconstrained case. We start with Q and choose a pair $q, q' \in Q$. By assumption, there exists a word $u \in \Sigma^*$ with $\mu(p_0, u) = p_0$ such that $\delta(q, u) = \delta(q', u)$ and hence $|\delta(Q, u)| < |Q|$. Since $\mu(p_0, u) = p_0$, the constraint automaton is in the start state after reading u and we can simply choose another pair of states in $\delta(Q, u)$ which we will synchronize next until we end up with a single state. Note that we have to synchronize at most $|Q| - 1$ pairs. The concatenation of the synchronizing words for the chosen pairs of states yields a word w with $\mu(p_0, w) = p_0$. Let $v \in L(\mathcal{B})$, then $wv \in L(\mathcal{B})$ and wv is a synchronizing word for A. □

As we just have to check pairs of states we can devise a polynomial-time algorithm to decide $L(\mathcal{B})$-Constr-Sync of a returning automaton $\mathcal{B}$.

Theorem 5. *If $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$ is returning, then $L(\mathcal{B})$-Constr-Sync $\in$ P.*

Proof. Let $A = (\Sigma, Q, \delta)$ be an DCSA with $n = |Q|$. Let $m = |P|$. From A, we construct the DCSA $A^{\leq 2} = (\Sigma, \binom{Q}{\leq 2}, \delta')$. Then, for each two-element set $\{q_1, q_2\} \in \binom{Q}{\leq 2}$, define $A' = A^{\leq 2}_{\{q_1,q_2\},Q}$, identifying Q with all 1-element state sets. We check for each two-element set $\{q_1, q_2\} \in \binom{Q}{\leq 2}$ if $L(A') \cap L(\mathcal{B}_{p_0,\{p_0\}}) \neq \emptyset$. By Lemma 5, the DCSA A is synchronizing with respect to $\mathcal{B}$ if and only if each intersection is non-empty. Each of the $\binom{n}{2}$ intersections can be checked by using the product-automaton construction in time $\mathcal{O}\left((n + \binom{n}{2})m\right)$. □

Our considerations can be turned into a polynomial-time algorithm for computing a synchronizing word for A with respect to $\mathcal{B}$, which implies the following result.

Corollary 3. *If the constraint automaton $\mathcal{B}$ is returning, a shortest synchronizing word with respect to $\mathcal{B}$ is polynomially bounded in the size of the input automaton.*

With the next theorem we generalize our introductory example of ab^*a as a constraint language. We show that if all cycles in the constraint automaton are labeled with words from a single unary language, then the Constr-Sync problem is solvable in NP.

Theorem 6. *Let $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$ be a PDFA. Then, $L(\mathcal{B})$-Constr-Sync $\in$ NP if there is a $\sigma \in \Sigma$ such that for all states $p \in P$, it holds that if $L(\mathcal{B}_{p,\{p\}})$ is infinite, then $L(\mathcal{B}_{p,\{p\}}) \subseteq \{\sigma\}^*$.*

Proof. By assumption, the letters in $\Sigma \setminus \{\sigma\}$ do not appear in any pumpable substring. Hence, their number in a word in $L(\mathcal{B})$ is bounded by $|P| = m$. Therefore, any word $w \in L(\mathcal{B})$ can be partitioned into at most $2m - 1$ substrings $w = u_1 v_1 \dots u_{m-1} v_{m-1} u_m$ with $u_i \in \{\sigma\}^*$ and $v_i \in (\Sigma \setminus \{\sigma\})^*$ for all $i \leq m$. Note that $|v_i| \leq m - 1$, for all $i < m$. Let $A = (\Sigma, Q, \delta)$ be a yes-instance of $L(\mathcal{B})$-Constr-Sync with $|Q| = n$. Let k be the number of sun-structures in A_σ. Let $w \in L(\mathcal{B})$ be a synchronizing word for A partitioned as mentioned above.

Claim 1: If for some $i \leq m$, $|u_i| > (mn)^k + n$, then we can replace u_i by some $u_i' \in \{\sigma\}^*$ with $|u_i'| \leq (mn)^k + n$, yielding a word $w' \in L(\mathcal{B})$ that synchronizes A.

Proof of Claim 1: Since $u_i \in \{\sigma\}^*$, the automaton A acts like A_σ on u_i. For each u_i, after at most n σ-transitions, every active state q in A (i.e., $q \in \delta(Q, u_1 v_1 \dots v_{i-1} \sigma^n)$) reached some cycle. After at most n further σ-transitions, each active state in A will have been

through some cycle at least once and has been back to its original position in the cycle at least once. The size of the cycle in $\mathcal{B}$ which is repeated in u_i might cause A to skip some states in the cycle. Therefore, we need to multiply the length of a considered cycle in A with the length of the biggest cycle in $\mathcal{B}$, trivially upper-bounded by m, in order to get an upper bound on the number of σ's needed to reach every combination of active states in one sun-structure of A. To get all possible combinations of active states over all sun-structures, we must multiply this expression for each sun-structure, yielding a total upper bound on the number of σ-transitions and hence the size of any u_i of $(mn)^k + n$ when avoiding cycles in configuration sequences. Therefore, each u_i can be replaced by an u_i' with $|u_i'| \leq (mn)^k + n$, yielding a synchronizing word for A, as well. $\lhd$

We will now show that we can decide whether A is synchronizing with respect to $\mathcal{B}$ in polynomial time using nondeterminism despite the fact that an actual synchronizing word might be exponentially large. This problem is circumvented by some preprocessing based on modulo arithmetics. This allows us to guess a binary representation $\mathrm{bin}(u_i)$ of $|u_i|$ instead of u_i itself. Hence, we guess $w_{\mathrm{bin}} = \mathrm{bin}(u_1)v_1 \dots \mathrm{bin}(u_{m-1})v_{m-1}\,\mathrm{bin}(u_m)$. Since all u_i are single exponential in size, the length of w_{bin} is polynomially bounded in the size of A.

Claim 2: For each $q \in Q$, one can compute in polynomial time numbers $\ell(q), \tau(q) \leq n$ such that, given some number x in binary, based on $\ell(q), \tau(q)$, one can compute in polynomial time a number $y \leq n$ such that $\delta(q, \sigma^x) = \delta(q, \sigma^y)$.

Proof of Claim 2: For each state $q \in Q$, we calculate its *σ-orbit* $\mathrm{Orb}_\sigma(q)$, that is, the set $\mathrm{Orb}_\sigma(q) = \{q, \delta(q,\sigma), \delta(q,\sigma^2), \dots, \delta(q,\sigma^\tau), \delta(q,\sigma^{\tau+1}), \dots, \delta(q,\sigma^{\tau+\ell-1})\}$ such that all states in $\mathrm{Orb}_\sigma(q)$ are distinct but $\delta(q,\sigma^{\tau+\ell}) = \delta(q,\sigma^\tau)$. Let $\tau(q) := \tau$ and $\ell(q) := \ell$ be the lengths of the tail and the cycle, respectively; these are nonnegative integers that do not exceed n. Observe that $\mathrm{Orb}_\sigma(q)$ includes the cycle $\{\delta(q,\sigma^\tau), \dots, \delta(q,\sigma^{\tau+\ell-1})\}$. We can use this information to calculate $\delta(q,\sigma^x)$, given a nonnegative integer x and a state $q \in Q$, as follows: (a) If $x \leq \tau(q)$, we can find $\delta(q,\sigma^x) \in \mathrm{Orb}_\sigma(q)$. (b) If $x > \tau(q)$, then $\delta(q,\sigma^x)$ lies on the cycle of the sun-structure. Compute $y := \tau(q) + (x - \tau(q)) \pmod{\ell(q)}$. Clearly, $\delta(q,\sigma^x) = \delta(q,\sigma^y) \in \mathrm{Orb}_\sigma(q)$. The crucial observation is that this computation can be done in time polynomial in $|Q|$ and in $|\,\mathrm{bin}(x)|$ using a square-and-multiply approach. $\lhd$

As a consequence, given $S \subseteq Q$ and $x \geq 0$ (in binary), we can compute $\delta(S,\sigma^x)$ in polynomial time. As $\mathcal{B}$ is even fixed, we can do a similar preprocessing also for $\mathcal{B}$ in polynomial time.

The NP-machine guesses w_{bin} part-by-part, keeping track of the set S of active states of A and of the current state p of $\mathcal{B}$. Initially, $S = Q$ and $p = p_0$. When guessing the number

$x_i = |u_i|$ in binary, by Claim 1 we guess $\log(|u_i|) \leq \log((mn)^k + n) \in O(n \log n)$ many bits. By Claim 2, we can update $S := \delta(S, \sigma^{x_i})$ and $p := \mu(p, \sigma^{x_i})$ in polynomial time. After guessing v_i, we can simply update $S := \delta(S, v_i)$ and $p := \mu(p, v_i)$ by simulating this input, as $|v_i| \leq m = |P|$, which is a constant in our setting. Finally, check if $|S| = 1$ and if $p \in F$. □

Let $b(n, k)$ be the number of bits guessed by our NP-algorithm. Observe that $b(n, k) \in O((k + 1) \log(nm))$ with $b(n, k) \leq m^2 \log(|\Sigma|) + (m - 1)(k + 1)(\log(m) + \log(n))$. As m and $|\Sigma|$ are constants and as $n, k < n$ depend on A, we can determinize this algorithm by testing $b(n, k)$ many bits.

Corollary 4. *Under the assumptions of Theorem 6, $L(\mathcal{B})$-Constr-Sync is in XP with parameter k counting the number of sun-structures in A_σ for an input DCSA A.*

When $k = 1$, we face a one-cluster automaton, see [Béal et al., 2011]. Ideas how to further reduce the amount of nondeterminism by using more algebraic tools are described in Appendix A.

After these more general thoughts, we now focus on two-state constraint automata $\mathcal{B}$, giving a complete picture of the complexity of $L(\mathcal{B})$-Constr-Sync over alphabets Σ with $|\Sigma| \leq 3$.

4 Constraint Automata with Two States and Two or Three Letters

In the following we will give a complete classification of the complexity of $L(\mathcal{B})$-Constr-Sync for two-state constraint automata $\mathcal{B}$ over a binary and a ternary alphabet. The number of two-state PDFAs is already quite high. We explain why we need to consider only one automaton for each automaton code listed in Tables 1 and 3. Here, we consider 1 as the start state and $\{2\}$ as the set of final states and call this the *standard interpretation* of a code.

Lemma 6. *Let $\mathcal{B} = (\Sigma, P, \mu)$ be some partial deterministic semi-automaton with two states, i.e., $P = \{1, 2\}$. Then, for each $p_0 \in P$ and each $F \subseteq P$, we either have $L(\mathcal{B}_{p_0,F})$-Constr-Sync $\in$ P, or $L(\mathcal{B}_{p_0,F})$-Constr-Sync $\equiv_m^{\log}$ $L(\mathcal{B}')$-Constr-Sync for a PDFA $\mathcal{B}' = (\Sigma, P', \mu', 1, \{2\})$.*

Proof. We can clearly exclude the trivial case $F = \emptyset$. If $p_0 = 2$, then we can easily change the roles of states 1 and 2 in our discussions, i.e., we find an isomorphic automaton

that is taken care of. Therefore, we can assume $p_0 = 1$. If $F = \{1\}$, then two cases may arise. (1) Either the automaton $\mathcal{B}_{p_0,F}$ is returning if there exists some transition form state 1 to 2 and vice versa, in which case $L(\mathcal{B}_{p_0,F})$-Constr-Sync $\in$ P according to Theorem 5. (2) Or $\mathcal{B}_{p_0,F}$ is equivalent to a partial one state automaton which is captured by Corollary 2. If $F = \{1, 2\}$, additionally a third case can appear. (3) There is a transition from state 1 to 2 but no transition from state 2 to 1. In this case the accepted language equals the set of prefixes of $L(\mathcal{B}_{1,\{2\}})$, which is a case captured by Theorem 4. □

Hence, we only need to specify $\mathcal{B} = (\Sigma, \{1, 2\}, \mu)$ in the following without start and final state. Let $\Sigma_{ij} := \{\, a \in \Sigma \mid \mu(i, a) = j \,\}$ for $1 \leq i, j \leq 2$. As $\mathcal{B}$ is deterministic, $\Sigma_{i1} \cap \Sigma_{i2} = \emptyset$. Consider easy cases first.

Proposition 2. *If one of the following conditions hold, then $L(\mathcal{B}_{1,\{2\}})$-Constr-Sync is in P: (1) $\Sigma_{1,2} = \emptyset$, (2) $\Sigma_{2,1} \neq \emptyset$, (3) $\Sigma_{1,1} \cup \Sigma_{1,2} \subseteq \Sigma_{2,2}$, or (4) $\Sigma_{1,1} \cup \Sigma_{2,2} = \emptyset$.*

Proof. (1) If $\Sigma_{1,2} = \emptyset$ means $L(\mathcal{B}_{1,\{2\}}) = \emptyset$. (2) If $\Sigma_{2,1} \neq \emptyset$, then $\mathcal{B}$ is returning (Theorem 5). (3) Lemma 3 and Theorem 3 cover this case. (4) Now, $L(\mathcal{B}_{1,\{2\}})$ is finite. □

For $\mathcal{B} = (\Sigma, \{1, 2\}, \mu)$ and $x \in \Sigma_{1,2}$, let $\mathcal{B}^x$ denote the variation with transition function μ^x defined by $\mu^x = \mu \cap (\{\, (p, y, p) \mid p \in \{1, 2\}, y \in \Sigma \,\} \cup \{(1, x, 2)\})$. Then Lemma 4 implies:

Lemma 7. *If $L(\mathcal{B}^x_{1,\{2\}})$-Constr-Sync $\in$ P for each $x \in \Sigma_{1,2}$ and if $\Sigma_{2,1} = \emptyset$, then $L(\mathcal{B}_{1,\{2\}})$-Constr-Sync $\in$ P.*

Lemma 7 gives some final arguments why we only study the standard interpretation.

For 2-state constraint automata with alphabet $\Sigma = \{a, b\}$, in order to avoid isomorphic automata and by Proposition 2, we can assume that either (1) $a \in \Sigma_{1,1}$ and $b \in \Sigma_{1,2}$ and $|\Sigma_{2,2}| \leq 1$ or (2) $a \in \Sigma_{1,2}$ but $b \notin \Sigma_{1,1}$ and $|\Sigma_{2,2}| > 0$. See Table 1.

The constrained synchronization problem for constraint automata with a binary alphabet is not easy in general, as we have seen already in Proposition 1 for 3-state constraint PDFA.

Theorem 7. *For any two-state binary PDFA $\mathcal{B}$, $L(\mathcal{B})$-Constr-Sync $\in$ P.*

Proof. By Table 1, we only need to show the claim for $\mathcal{B}_1$ = `[ 1 2 ‡ 2 - ]`, $\mathcal{B}_2$ = `[ 1 2 ‡ - 2 ]`, $\mathcal{B}_3$ = `[ 1 2 ‡ - - ]`, $\mathcal{B}_4$ = `[ - 2 ‡ 2 - ]`, and $\mathcal{B}_5$ = `[ 2 2 ‡ 2 - ]`. Let $A = (\Sigma, Q, \delta)$ be a DCSA with $n := |Q| - 1$. Consider the first PDFA $\mathcal{B}_1$ with $L(\mathcal{B}_1) =$

Automaton code	Why in P?	Automaton code	Why in P?
`[ * 2 ‡ 1 * ]`	$a \in \Sigma_{2,1}$ Propos. 2, (2)	`[ 2 2 ‡ 2 - ]`	Theorem 7
`[ * 2 ‡ * 1 ]`	$b \in \Sigma_{2,1}$ Propos. 2, (2)	`[ 2 2 ‡ - 2 ]`	Isomorphic to `[ 2 2 ‡ 2 - ]`
`[ * 2 ‡ 2 2 ]`	$\Sigma_{1,1} \cup \Sigma_{1,2} = \Sigma_{2,2}$ Propos. 2, (3)	`[ {2,-} 2 ‡ - - ]`	$\Sigma_{1,1} \cup \Sigma_{2,2} = \emptyset$ Propos. 2, (4)
`[ 1 2 ‡ {-,2} - ]`	Theorem 7	`[ - 2 ‡ 2 - ]`	Theorem 7
`[ 1 2 ‡ - 2 ]`	Theorem 7	`[ - 2 ‡ - 2 ]`	$\Sigma_{1,1} \cup \Sigma_{1,2} = \Sigma_{2,2}$ Propos. 2, (3)

Table 1: List of all PDFAs with two states and a binary alphabet, with $\Sigma_{1,2} = \{a, b\}$ or $\Sigma_{1,2} = \{b\}$.

a^*ba^*. Let $a^\ell ba^m$ be some synchronizing word for A, then by Lemma 2, applied to A_a, we have $\delta(Q, a^\ell) = \delta(Q, a^j)$ for some $j \leq n$, and moreover, by a similar argument, we find $k \leq n$ with $\delta(\delta(Q, a^j b), a^m) = \delta(\delta(Q, a^j b), a^k)$. So, the word $a^j ba^k$ is synchronizing and according to Lemma 1 the word $a^n ba^n$ is also synchronizing. In order to decide synchronizability with respect to $\mathcal{B}_1$, we simply have to check this last word. With the same argument, for $\mathcal{B}_2$ we only have to test the word $a^n b^n$, for $\mathcal{B}_3$ the word $a^n b$, and for $\mathcal{B}_4$ the word ba^n. As $\mathcal{B}_5$ accepts the union of $L(\mathcal{B}_4)$ and a unary regular language, the claim follows with Corollary 1 and Lemma 4. □

Next, we give a full classification on the complexity of the constrained synchronization problem for constraint automata with two states and a ternary alphabet. As can be verified by a case-by-case analysis, inspecting Table 2, the only automaton with a constrained synchronization problem in P not covered by the generalization results in Section 3 is `[ 1 2 - ‡ - - 2 ]`.

Theorem 8. *Let $\mathcal{B} =$ `[ 1 2 - ‡ - - 2 ]`. Then $L(\mathcal{B})$-Constr-Sync is in P.*

Proof. The language accepted by the constraint automaton $\mathcal{B} =$ `[ 1 2 - ‡ - - 2 ]` is a^*bc^*. Let $A = (\Sigma, Q, \delta)$ be a DCSA, $n = |Q|$. By arguments along the lines of the proof of Theorem 7, one can show that there is a synchronizing word for A with respect to $\mathcal{B}$ if and only if $a^n bc^n$ synchronizes A. This condition is easy to check. □

The leftover two-state automata over a ternary alphabet are listed in Table 3. For all of them the corresponding constrained synchronization problem is PSPACE-complete (see Theorem 9). We want to point out that there is no constraint automata with two states and a ternary alphabet for which the $L(\mathcal{B})$-Constr-Sync is not either PSPACE-complete or contained in P, as we covered all possible automata of this kind.

Theorem 9. *For each constraint automaton $\mathcal{B}$ in Table 3 the problem $L(\mathcal{B})$-Constr-Sync is PSPACE-hard.*

Automaton code	Complexity	Explanation
`[ * * * ‡ 1 * * ]`	P	$\Sigma_{2,1} \neq \emptyset$ Propos. 2, (2)
`[ * * * ‡ * 1 * ]`	P	$\Sigma_{2,1} \neq \emptyset$ Propos. 2, (2)
`[ * * * ‡ * * 1 ]`	P	$\Sigma_{2,1} \neq \emptyset$ Propos. 2, (2)
`[ * * * ‡ 2 2 2 ]`	P	$\Sigma_{1,1} \cup \Sigma_{1,2} \subseteq \Sigma_{2,2}$ Propos. 2, (3)
`[ {-,1} {-,1} {-,1} ‡ * * * ]`	P	$\Sigma_{1,2} = \emptyset$ Propos. 2, (1)
`[ 1 1 2 ‡ - - - ]`	PSPACE-complete	Theorem 9 Case 2
`[ 1 1 2 ‡ 2 - - ]`	PSPACE-complete	Theorem 9 Case 2
`[ 1 1 2 ‡ - 2 - ]`	PSPACE-complete	Isomorphic to `[ 1 1 2 ‡ 2 - - ]`
`[ 1 1 2 ‡ - - 2 ]`	PSPACE-complete	Theorem 9 Case 2
`[ 1 1 2 ‡ 2 2 - ]`	PSPACE-complete	Theorem 9 Case 2
`[ 1 1 2 ‡ - 2 2 ]`	PSPACE-complete	Theorem 9 Case 4
`[ 1 1 2 ‡ 2 - 2 ]`	PSPACE-complete	Isomorphic to `[ 1 1 2 ‡ - 2 2 ]`
`[ 1 2 2 ‡ - - - ]`	P	Lemma 7
`[ 1 2 2 ‡ 2 - - ]`	P	Lemma 7
`[ 1 2 2 ‡ - 2 - ]`	P	Lemma 7
`[ 1 2 2 ‡ - - 2 ]`	P	Lemma 7
`[ 1 2 2 ‡ 2 2 - ]`	PSPACE-complete	Theorem 9 Case 3
`[ 1 2 2 ‡ - 2 2 ]`	PSPACE-complete	Theorem 9 Case 4
`[ 1 2 2 ‡ 2 - 2 ]`	PSPACE-complete	Isomorphic to `[ 1 2 2 ‡ 2 2 - ]`
`[ 2 2 2 ‡ - - - ]`	P	$\Sigma_{1,2} = \emptyset$ Propos. 2,(1)
`[ 2 2 2 ‡ 2 - - ]`	P	Lemma 7
`[ 2 2 2 ‡ - 2 - ]`	P	Lemma 7
`[ 2 2 2 ‡ - - 2 ]`	P	Lemma 7
`[ 2 2 2 ‡ 2 2 - ]`	PSPACE-complete	Theorem 9 Case 1
`[ 2 2 2 ‡ - 2 2 ]`	PSPACE-complete	Isomorphic to `[ 2 2 2 ‡ 2 2 - ]`
`[ 2 2 2 ‡ 2 - 2 ]`	PSPACE-complete	Isomorphic to `[ 2 2 2 ‡ 2 2 - ]`
`[ 1 2 - ‡ - - - ]`	P	$\lvert\bigcup_{1\leq i,j\leq 2}\Sigma_{ij}\rvert \leq 2$, Lemma 3
`[ 1 2 - ‡ 2 - - ]`	P	$\lvert\bigcup_{1\leq i,j\leq 2}\Sigma_{ij}\rvert \leq 2$, Lemma 3
`[ 1 2 - ‡ - 2 - ]`	P	$\lvert\bigcup_{1\leq i,j\leq 2}\Sigma_{ij}\rvert \leq 2$, Lemma 3
`[ 1 2 - ‡ - - 2 ]`	P	Theorem 8
`[ 1 2 - ‡ 2 2 - ]`	P	$\lvert\bigcup_{1\leq i,j\leq 2}\Sigma_{ij}\rvert \leq 2$, Lemma 3
`[ 1 2 - ‡ - 2 2 ]`	PSPACE-complete	Theorem 9 Case 4
`[ 1 2 - ‡ 2 - 2 ]`	PSPACE-complete	Theorem 9 Case 3
`[ 2 2 - ‡ - - - ]`	P	Lemma 7
`[ 2 2 - ‡ 2 - - ]`	P	Lemma 7
`[ 2 2 - ‡ - 2 - ]`	P	Lemma 7
`[ 2 2 - ‡ - - 2 ]`	P	Lemma 7
`[ 2 2 - ‡ 2 2 - ]`	P	Lemma 7
`[ 2 2 - ‡ - 2 2 ]`	PSPACE-complete	Isomorphic to `[ 2 2 - ‡ 2 - 2 ]`
`[ 2 2 - ‡ 2 - 2 ]`	PSPACE-complete	Theorem 9 Case 1
`[ 2 - - ‡ - - - ]`	P	$\lvert\bigcup_{1\leq i,j\leq 2}\Sigma_{ij}\rvert \leq 2$, Lemma 3
`[ 2 - - ‡ 2 - - ]`	P	$\lvert\bigcup_{1\leq i,j\leq 2}\Sigma_{ij}\rvert \leq 2$, Lemma 3
`[ 2 - - ‡ - 2 - ]`	P	$\lvert\bigcup_{1\leq i,j\leq 2}\Sigma_{ij}\rvert \leq 2$, Lemma 3
`[ 2 - - ‡ - - 2 ]`	P	$\lvert\bigcup_{1\leq i,j\leq 2}\Sigma_{ij}\rvert \leq 2$, Lemma 3
`[ 2 - - ‡ 2 2 - ]`	P	$\lvert\bigcup_{1\leq i,j\leq 2}\Sigma_{ij}\rvert \leq 2$, Lemma 3
`[ 2 - - ‡ - 2 2 ]`	PSPACE-complete	Theorem 9 Case 1

Table 2: List of all PDFAs with two states and a ternary alphabet together with the complexity of the corresponding constrained synchronization problem and the explanation thereof.

Case	Automaton code	Language
1	[2 - - ‡ - 2 2]	$a(b+c)^*$
	[2 2 2 ‡ 2 2 -]	$(a+b+c)(a+b)^*$
	[2 2 - ‡ 2 - 2]	$(a+b)(a+c)^*$
2	[1 1 2 ‡ - - -]	$(a+b)^*c$
	[1 1 2 ‡ 2 - -]	$(a+b)^*ca^*$
	[1 1 2 ‡ 2 2 -]	$(a+b)^*c(a+b)^*$
	[1 1 2 ‡ - - 2]	$(a+b)^*cc^*$
3	[1 2 - ‡ 2 - 2]	$a^*b(a+c)^*$
	[1 2 2 ‡ 2 2 -]	$a^*(b+c)(a+b)^*$
4	[1 2 - ‡ - 2 2]	$a^*b(b+c)^*$
	[1 1 2 ‡ - 2 2]	$(a+b)^*c(b+c)^*$
	[1 2 2 ‡ - 2 2]	$a^*(b+c)(b+c)^*$

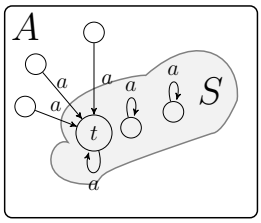
Table 3: Constraint automata (2 states, 3 letters) with a PSPACE-hard Constr-Sync problem.

Proof. We prove each case separately by giving an explicit reduction for one of the automata, the statement for the other automata of that case follows by the same argument. Our reductions are illustrated in Figure 2. Each reduction is starting out from (A, S), with $A = (\Sigma, Q, \delta)$ being a DCSA and $S \subseteq Q$. Depending on the considered case, (A, S) is either an instance of Sync-From-Subset, or of Sync-Into-Subset, respectively. We construct from A a DCSA $A' = (\Sigma', Q', \delta')$, with $\Sigma' = \Sigma \cup \{\sigma\}$ for an appropriately chosen letter $\sigma \notin \Sigma$, such that there exists $w \in \Sigma^*$ with $|\delta(S, w)| = 1$, or $\delta(Q, w) \subseteq S$, respectively, if and only if A' is synchronizing with respect to $\mathcal{B}$. This construction is described and illustrated in Figure 2.

Case 1: Consider the first automaton $\mathcal{B} =$ [2 - - ‡ - 2 2]. Then, $L(\mathcal{B}) = a(b+c)^*$. We reduce from the PSPACE-complete problem Sync-From-Subset for the binary alphabet $\Sigma = \{b, c\}$. Since the constraint automaton forces us to read an a as the first letter, we start synchronizing A' with $\delta'(Q, a) = S$. After the first a, we are allowed to read any letter from Σ. Hence, if $|\delta(S, w)| = 1$ by a word $w \in \Sigma^*$, then $aw \in L(\mathcal{B})$ synchronizes A'. Conversely, if there exists a word v that synchronizes A' with respect to $\mathcal{B}$, then v must be of the form $v = au$ with $u \in \{b, c\}^*$. By the definition of δ', we have $|\delta(S, u)| = 1$.

The PSPACE-hardness of constrained synchronization with respect to the PDFA [2 2 2 ‡ 2 2 -] with the language $(a+b+c)(a+b)^*$ follows with the same reduction with the letters a and c interchanged. The same idea applies to [2 2 - ‡ 2 - 2].

SYNC-FROM-SUBSET (Case 1)

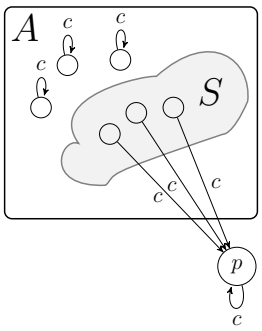


$$\delta'(q,a) := \begin{cases} q & \text{if } q \in S \\ t & \text{otherwise.} \end{cases}$$

$$\delta'(q,b) := \delta(q,b)$$

$$\delta'(q,c) := \delta(q,c)$$

SYNC-INTO-SUBSET (Case 2)

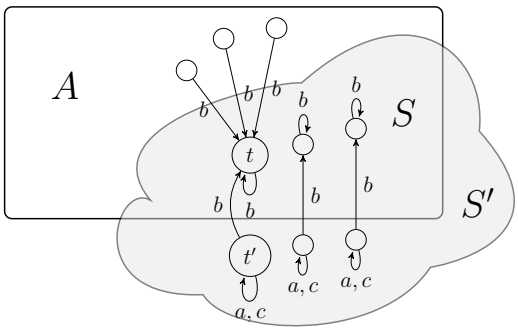


$$\delta'(q,a) := \begin{cases} p & \text{if } q = p, \\ \delta(q,a) & \text{otherwise.} \end{cases}$$

$$\delta'(q,b) := \begin{cases} p & \text{if } q = p, \\ \delta(q,b) & \text{otherwise.} \end{cases}$$

$$\delta'(q,c) := \begin{cases} p & \text{if } q \in S \cup \{p\} \\ q & \text{otherwise.} \end{cases}$$

SYNC-FROM-SUBSET (Case 3)

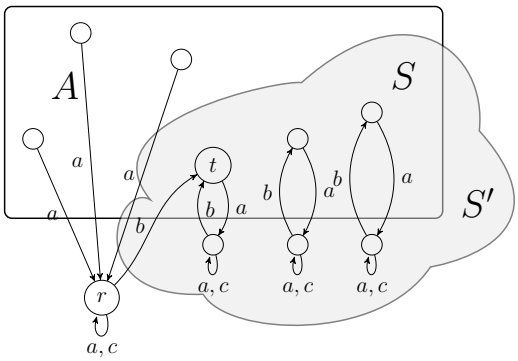


$$\delta'(p,a) := \begin{cases} \delta(p,a) & \text{if } p \in Q \\ p & \text{if } p \in S' \end{cases}$$

$$\delta'(p,b) := \begin{cases} t & \text{if } p \in Q \setminus S \\ p & \text{if } p \in S \\ q & \text{if } p \in S' \text{ and } p = q' \end{cases},$$

$$\delta'(p,c) := \begin{cases} \delta(p,c) & \text{if } p \in Q \\ p & \text{if } p \in S' \end{cases}$$

SYNC-FROM-SUBSET (Case 4)



$$\delta'(p,a) := \begin{cases} r & \text{if } p \in (Q \setminus S) \cup \{r\} \\ p' & \text{if } p \in S \\ p & \text{if } p \in S' \end{cases}$$

$$\delta'(p,b) := \begin{cases} \delta(p,b) & \text{if } p \in Q \\ t & \text{if } p = r \\ q & \text{if } p \in S' \text{ and } p = q' \end{cases}$$

$$\delta'(p,c) := \begin{cases} \delta(p,c) & \text{if } p \in Q \\ r & \text{if } p = r \\ p & \text{if } p \in S' \end{cases}$$

Figure 2: Schematic illustration of our reductions. Transitions inherited from A are not shown.

Case 2: The language accepted by $\mathcal{B} =$ `[ 1 1 2 ‡ - - - ]` is $L(\mathcal{B}) = (a+b)^*c$. We reduce from Sync-Into-Subset. Note that by construction if A' is synchronizing, p must be the unique synchronization state. The state p can only be reached by a transition with the letter c, but the constraint automaton only allows us to read one single c as the last letter of the synchronizing word. Hence, if there exists a synchronizing word w for A' with respect to $\mathcal{B}$, it is of the form uc with $u \in \{a,b\}^*$. Since $\delta'(Q,w) = p$, $\delta'(Q,u) \subseteq \{\, q \in Q \mid \delta'(q,c) = p \,\}$; by definition of δ', this equals the set $S \cup \{p\}$. Hence, u synchronizes the automaton A into a subset of S. Conversely, if w is a word that synchronizes A to a subset of S, by the construction of δ', the word wc synchronizes A' to $\{p\}$ and since $w \in \{a,b\}^*$, $wc \in L(\mathcal{B})$.

We can only reach the synchronizing state by reading the letter c and for each automaton of this case we are only allowed to read one single letter c. Therefore, allowing additional letters a and b in the synchronizing word after reading the letter c does not change the synchronizability of A' and hence the same construction works for the constraint automata `[ 1 1 2 ‡ 2 - - ]` and `[ 1 1 2 ‡ 2 2 - ]`. The same holds if we allow only additional letters c (and no a or b) after the first c. In A', c only leads in the synchronization state from states in S and is the identity on other states. Therefore, $\delta(q,cc) = \delta(q,c)$ for any state q and the construction of Case 2 also works for the constraint automaton `[ 1 1 2 ‡ - - 2 ]`.

Case 3: The language accepted by $\mathcal{B} =$ `[ 1 2 - ‡ 2 - 2 ]` is $L(\mathcal{B}) = a^*b(a+c)^*$. We reduce from Sync-From-Subset for $\Sigma = \{a,c\}$ similar to the one in Case 1, but we have to ensure that the whole set S is active after reading the letter b, since a preceding a might already merge some states in S. The idea is to add for each state $q \in S$ a new state q' for which we stay in q' with the letters a, c and go to q with the letter b. Therefore, we ensure that $\delta(Q, a^ib) = S$ for every integer i. Since, starting from the whole state set, A' is precisely in the state set S after the first and only b letter, the rest of the argument follows as in Case 1. For the constraint automaton `[ 1 2 2 ‡ 2 2 - ]`, accepting the language $a^*(b+c)(a+b)^*$, the same idea applies.

Case 4: The language accepted by $\mathcal{B} =$ `[ 1 2 - ‡ - 2 2 ]` is $L(\mathcal{B}) = a^*b(b+c)^*$. Here, we do not have a special letter which appears exactly once in a word from $L(\mathcal{B})$. We will use the optional a letters in order to jump into S, since a word from $L(\mathcal{B})$ that does not contain any a must synchronize the whole state set and therefore also the subset S. We reduce from the problem Sync-From-Subset for the alphabet $\Sigma = \{b,c\}$. We decompose the state set Q with the letter a in S and $Q \setminus S$. The states in S are stored in an annotated copy S' of S. The other states are gathered up in a new state r. With the first b, we restore the set S as the set of active states. As we stored the states in S in the set S' before, we ensure that the first letter b (after a letter a) does not cause any

transition in the initial automaton A which could already map an active state outside of S. The remainder of a synchronizing word then synchronizes S.

If the set S in A is synchronizable to a single state by a word $u \in \{b, c\}^*$, then the word $abu \in L(\mathcal{B})$ synchronizes A' since $\delta'(Q', ab) = S$. For the other direction, assume A' is synchronizing with respect to $\mathcal{B}$ by a word w. Then w is of the form ubv with $u \in a^*$, $v \in \{b, c\}^*$. If $u \neq \epsilon$, then $\delta(Q', ub) = S$ and v synchronizes S to a single state. If $u = \epsilon$, then $S \subseteq \delta'(Q', b) \subseteq Q$ since we never synchronized any states into r and we leave all states in the set $S' \cup \{r\}$ with b and are not able to reach them again. In particular $\delta'(S', b) = S$. Therefore, bv synchronizes S to a single state without ever leaving Q. The same idea can be applied to the constraing automata `[ 1 1 2 ‡ - 2 2 ]` and `[ 1 2 2 ‡ - 2 2 ]`. □

5 Generalizations to Lift Results

In this section we aim for more general results, either by lifting existing cases by homomorphic images, or by identifying common patterns. For a map $\varphi : \Sigma \to \Gamma^*$ we identify it with its natural homomorphism extension $\varphi : \Sigma^* \to \Gamma^*$ without further mentioning.

Theorem 10. *Let $L \subseteq \Gamma^*$ and let $\varphi : \Sigma^* \to \Gamma^*$ be an homomorphism such that $\varphi(\varphi^{-1}(L)) = L$. Then L-Constr-Sync $\leq_m^{\log}$ $\varphi^{-1}(L)$-Constr-Sync.*

Proof. Let $A = (\Gamma, Q, \delta)$ be some DCSA. We want to know if it is synchronizing with respect to L. Therefore, we build the automaton $A' = (\Sigma, Q, \delta')$ according to the rule

$$\delta'(p, x) = q \quad \text{if and only if} \quad \delta(p, \varphi(x)) = q,$$

for $x \in \Sigma$. As φ is a homomorphism δ' generalizes to words as expected. As φ is a mapping, A' is indeed deterministic and complete, since A is a DCSA. As the homomorphism φ is independent of A, automaton A' can be constructed from A in logarithmic space. Next we prove that the translation is indeed a reduction.

If $u \in L$ is some synchronizing word for A, then there is some $s \in Q$ such that $\delta(r, u) = s$, for all $r \in Q$. By our assumption $\varphi(\varphi^{-1}(L)) = L$, we find some word $w \in \varphi^{-1}(L)$ with $\varphi(w) = u$. As with $\delta(r, \varphi(w)) = s$, it follows $\delta'(r, w) = s$, hence w is a synchronizing word for A'. Conversely, if $w \in \varphi^{-1}(L)$ is a synchronizing word for A', then there is some $s \in Q$ such that $\delta'(r, w) = s$, for all $r \in Q$. Further, $\varphi(w)$ is a synchronizing word for A, as by definition for all $r \in Q$, we have $\delta(r, \varphi(w)) = s$. Moreover, because $w \in \varphi^{-1}(L)$, we have $\varphi(w) \in L$, as $\varphi(\varphi^{-1}(L)) = L$. □

A typical application of the preceding theorem is to lift hardness results from smaller to bigger alphabets; e.g., knowing PSPACE-hardness for the constraint language $a(b+c)^*$ lifts to PSPACE-hardness for the constraint language $a(b+c+d)^*$ via $\varphi : a \mapsto a$, $b \mapsto b, c \mapsto c, d \mapsto c$.

Remark 3. It is impossible to further generalize the previous result from homomorphisms to mappings induced by deterministic generalized sequential machines. Such a machine allows to map $(a+b)(a+b)^*$ to $a(b+c)^*$, but the constraint $(a+b)(a+b)^*$ yields a synchronization problem in P.

Theorem 11. *Let $L \subseteq \Sigma^*$. Let $\varphi : \Sigma^* \to \Gamma^*$ be an homomorphism such that $\varphi(\Sigma)$ is a prefix code. Let $c \in \Gamma$ with $\{c\}\Gamma^* \cap \varphi(\Sigma) = \emptyset$. Let $k := \max\{\, \ell \geq 0 \mid \exists u, v \in \Gamma^* \colon uc^\ell v \in \varphi(\Sigma) \,\}$. Then L-Constr-Sync $\leq_m^{\log}$ $\{c^{k+1}\}\varphi(L)$-Constr-Sync.*

Proof. Let $A = (\Sigma, Q, \delta)$ be some DCSA. We want to know if it has a synchronizing word from L. For each state $q \in Q$, introduce states

$$Q_q := \{\, q_u \mid u \text{ is a proper prefix of some word in } \varphi(\Sigma) \,\}.$$

Set $Q' := \bigcup_{q \in Q} Q_q$. We identify all states q_ε with $q \in Q$, hence we have $Q \subseteq Q'$. Then, build $A' = (\Gamma, Q', \delta')$. For $x \in \Sigma$ and states $q_u \in Q_q$ and $q \in Q$, set

$$\delta'(q_u, x) = \begin{cases} q_{ux} & \text{if } ux \text{ is a proper prefix of some word in } \varphi(\Sigma), \\ \delta(q, a) & \text{if } ux = \varphi(a), \text{ for some } a \in \Sigma, \\ q & \text{otherwise.} \end{cases}$$

As $\varphi(\Sigma)$ is a prefix code, the transition function δ' is well-defined, because the first two cases of its definition are mutually exclusive and there could not be two symbols a and a' with $ux = \varphi(a) = \varphi(a')$. The third case in the definition catches left-over cases, including the transition for c where c can not appear in the next position of a code-word, so that A' is indeed a DCSA. Also, it should be clear that the translation of a table for δ into a table for δ' can be implemented with a logspace machine by a proper pointer management.

The idea of the definition of δ' is to replace letter $x \in \Sigma$ with its image $\varphi(x)$, hence, in A' we 'branch' along the words $\varphi(\Sigma)$. With an undefined c-branch A' will fall back to the states of A (recall that $Q \subseteq Q'$). Observe that $\delta'(Q', c^{k+1}) = Q$. For $q \in Q$, we have $\delta'(q, \varphi(x)) = \delta(q, x)$ by construction. So if A has a synchronizing word $w \in L$, leading into the state $s \in Q$, then for all $r \in Q$ we have $\delta'(r, \varphi(w)) = \delta(r, w) = s$ and so $c^{k+1}\varphi(w) \in \{c\}^{k+1}\varphi(L)$ is a synchronizing word for A'.

Conversely, let $c^{k+1}u \in \{c\}^{k+1}\varphi(L)$ be synchronizing for A'. Then $u \in \varphi(L)$. Hence, we can find some $w \in L$ such that $\varphi(w) = u$. Moreover, by construction of A', the transition function $\delta'(t, c) = t$ for any $t \in Q$ and since there are at most k consecutive appearances of c in any code-word from $\varphi(\Sigma)$, we have $\delta'(t, c^{k+1}) \in Q$ for any $t \in Q'$ and hence $\delta'(t, c^{k+1}u) =: s \in Q$. Using $\delta(r, x) = \delta'(r, \varphi(x))$ for $r \in Q$ and $x \in \Sigma$, we find that $\delta(r, w) = \delta'(r, \varphi(w)) = s$ for all $r \in Q$, because $\delta'(Q', c^{k+1}) = Q$ by construction of A'. Hence, $w \in L$ is a synchronizing word for A. □

In the special case where c does not occur in $\varphi(\Sigma)$ at all, it is sufficient to choose $k = 0$, i.e., to consider the language $\{c\}\varphi(L)$ as constraint language.

With Theorem 11, we can transfer hardness results with constraint language L over arbitrary alphabets to hardness results with constraint languages over a binary alphabet.

Remark 4. With the construction presented in Corollary 1 in [Berlinkov, 2014] (see pp. 220-221) we can lift our hardness results for constrained synchronization with constraint automata with two states and a ternary alphabet to constrained synchronization problems with 6 states and a binary alphabet. More generally we can reduce the alphabet size of a constraint automata from $k = |\Sigma|$ to 2 by enlarging the size of its state set from $n = |Q|$ to $k \cdot n$ without affecting the hardness of the associated constrained synchronization problem.

Up to this point, we did not make use of the fact that constraint languages considered in this paper are given by finite automata. This changes from here onward.

It is hardness-preserving to plug sub-automata of some kind in front of an automata with a hard constrained synchronization problem. A partial automaton A is called *carefully synchronizing* if there exists a synchronizing word w for A such that the transition function of A is defined for w on every state of A.

Theorem 12. *Let $\mathcal{B} = (\Sigma^{\mathcal{B}}, P^{\mathcal{B}}, \mu^{\mathcal{B}}, p_0^{\mathcal{B}}, F)$ and $\mathcal{C} = (\Sigma^{\mathcal{C}}, P^{\mathcal{C}}, \mu^{\mathcal{C}}, p_0^{\mathcal{C}}, \emptyset)$ be PDFAs with $P^{\mathcal{B}} \cap P^{\mathcal{C}} = \emptyset$. For $p_x \in P^{\mathcal{C}}$ let $\nu \subseteq \{p_x\} \times (\Sigma^{\mathcal{B}} \cup \Sigma^{\mathcal{C}}) \times \{p_0^{\mathcal{B}}\}$ define the automaton $\mathcal{B}' = (\Sigma^{\mathcal{B}} \cup \Sigma^{\mathcal{C}}, P^{\mathcal{B}} \cup P^{\mathcal{C}}, \mu^{\mathcal{B}} \cup \mu^{\mathcal{C}} \cup \nu, p_0^{\mathcal{C}}, F)$. If the following three conditions are satisfied:*

1. *automaton $\mathcal{B}'$ is deterministic,*
2. *automaton $\mathcal{C}' = (\Sigma^{\mathcal{B}} \cup \Sigma^{\mathcal{C}}, P^{\mathcal{C}} \cup \{p_0^{\mathcal{B}}\}, \mu^{\mathcal{C}} \cup \nu \cup \{p_0^{\mathcal{B}}\} \times (\Sigma^{\mathcal{B}} \cup \Sigma^{\mathcal{C}}) \times \{p_0^{\mathcal{B}}\})$ is carefully synchronizing, and*
3. *there exists a synchronizing word $v = v_1 \dots v_n$ for $\mathcal{C}'$ such that $v_1 \dots v_{n-1} \in L(\mathcal{C}_{p_x})$, where $\mathcal{C}_{p_x}$ results from $\mathcal{C}$ by adding p_x to the set of final states,*

then $L(\mathcal{B})$-Constr-Sync $\leq L(\mathcal{B}')$-Constr-Sync.

Proof. Note that the start state of $\mathcal{B}'$ is the start state of $\mathcal{C}$, but the final states of $\mathcal{B}'$ are the ones from $\mathcal{B}$.

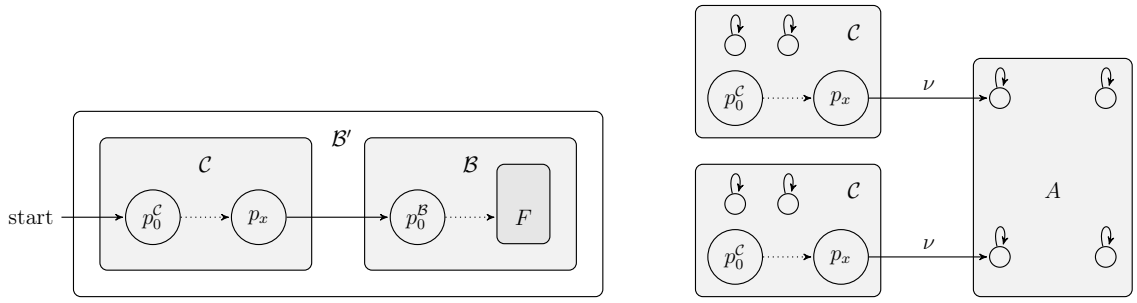


Figure 3: Schematic illustration of the constraint automaton $\mathcal{B}'$ and the construction of A' in Theorem 12. Transitions inherited from A are not shown.

Let $A = (\Sigma^{\mathcal{B}}, Q, \delta)$ be a DCSA. We extend A to a DCSA $A' = (\Sigma^{\mathcal{B}} \cup \Sigma^{\mathcal{C}}, Q', \delta')$ in the following way. For every state $q \in Q$ we add a copy of $\mathcal{C}$ to A', where a self-loop is added for every yet undefined transition in A'. The $\mathcal{C}$-copy is connected to q with the transitions in ν where the target $p_0^{\mathcal{B}}$ is replaced by q. Since the automaton $\mathcal{B}'$ is deterministic by condition (1), the $\mathcal{C}$-copies, which are added to A, are also deterministic and so is A'.

It remains to show that A is synchronizing with respect to $\mathcal{B}$ if and only if A' is synchronizing with respect to $\mathcal{B}'$. For the only if direction, assume $w \in L(\mathcal{B})$ is a synchronizing word for A. Considering A', condition (2) states that there exists a word v that, applied to all states of a copy of $\mathcal{C}$, leads every state of this copy through the exit state p_x into the original states of A. Further, condition (3) specifies that the last state leaves through p_x with the last letter of v and that this last state is the image of the start state. Hence, v is the label of a path from $p_0^{\mathcal{C}}$ to $p_0^{\mathcal{B}}$ in $\mathcal{B}'$ and $vw \in L(\mathcal{B}')$. Starting in all states of A', the active states in each $\mathcal{C}$-copy act synchronously. Hence, $\delta'(Q', v) = Q$. Note that no state of a $\mathcal{C}$-copy is reachable by a state of Q. Since A' acts like A on Q, $|\delta'(Q, w)| = 1$ and vw is a synchronizing word for A' with respect to $\mathcal{B}'$.

We now prove the converse implication of the reduction property. Let $w \in L(\mathcal{B}')$ be a synchronizing word for A'. Since the $\mathcal{C}$-copies are not reachable from any state in Q and A' acts on all $\mathcal{C}$-copies simultaneously, we can identity a position in w at which the last bunch of states leaves the $\mathcal{C}$-copies. Let $w = vx$ be a partition of w with $v = v_1 \dots v_n$, $v_j \in \Sigma^{\mathcal{B}} \cup \Sigma^{\mathcal{C}}$ for $j \leq n$, such that $\delta'(Q', v) = Q$ and $\delta'(Q', v_1 \dots v_j) \cap (Q' \setminus Q) \neq \emptyset$ for all $j < n$. Clearly, $|\delta'(Q, x)| = 1$. Hence, x is a synchronizing word for A. The automaton $\mathcal{B}'$ is in the $\mathcal{B}$ component after reading v, since the start state of $\mathcal{B}'$ works in the same way

as the state $p_0^{\mathcal{B}}$ in a $\mathcal{C}$-copy in A and after reading v every state has left the $\mathcal{C}$ components according to condition (2). Hence, x is a suffix of a word in $L(\mathcal{B})$. Let i be an index such that $\mathcal{B}'$ is in the state $p_0^{\mathcal{B}}$ after reading $v_1 v_2 \dots v_i$. Note that this i exists since $p_0^{\mathcal{B}}$ must be part of any accepting path in $\mathcal{B}'$. Then, $v_{i+1} v_{i+2} \dots v_n x$ is a word of $L(\mathcal{B})$ and also synchronizing for A, since it contains x as a suffix. □

As an illustration, we apply Theorem 12 to a family of languages.

Corollary 5. *Let the language-family $\mathcal{L}$ consists of languages $L_i := (b^*a)^i$ with $i \geq 2$. The constrained synchronization problem for all languages in $\mathcal{L}$ is NP-complete.*

Proof. The membership in NP follows from Theorem 6 since all pumpable substrings only contain the letter b. The NP-hardness of the constrained synchronization problem for b^*ab^*a follows directly from the proof sketch of Proposition 1 since we can allow an arbitrary number of b's before the first a without changing the reduction. The NP-hardness of the constrained synchronization problem for $(b^*a)^i$ with $i > 2$ follows from the fact that there exists an automaton $\mathcal{B}'$ with $L(\mathcal{B}') = (b^*a)^i$ that consists of the two automata $\mathcal{B}$ (accepting b^*ab^*a) and $\mathcal{C}$ (describing $(b^*a)^{i-3}b^*$) which are connected by an a-transition and fulfill the conditions of Theorem 12. □

6 Conclusions and Prospects

We have commenced a study of synchronization under regular constraints. The complexity landscape of 2-state constraint automata with at most ternary input alphabets is completely understood. In particular, binary alphabets yield polynomial-time solvable synchronization problems, while ternary alphabets split the constrained synchronization problems into polynomial-time solvable and PSPACE-complete cases. As already seen in the introduction, this picture changes with 3-state automata, giving an NP-complete scenario with binary alphabets. Our general results also imply PSPACE-complete synchronization problems for binary constraint automata with at least six states. In the following theorem, we present a three state constraint automaton with a binary alphabet for which the associated constrained synchronization problem is PSPACE-complete, also because of Theorem 2.

Theorem 13. *Let $\mathcal{B} =$* `[ - 2 ‡ 3 3 ‡ 2 - ]` *be a three state PDFA over the alphabet $\{0, 1\}$ with start state 1 and final state 3. Then the problem $L(\mathcal{B})$-*Constr-Sync *is PSPACE-hard.*

Proof. We give a reduction from the PSPACE-complete problem Sync-From-Subset. Let $A = (\{a, b\}, Q, \delta)$ be a DCSA and $S \subseteq Q$. We construct a DCSA $A' = (\{0, 1\}, Q', \delta')$ which is synchronizing with respect to $\mathcal{B}$ if and only if the state set S is synchronizable in A. The set Q' contains the set Q, an annotated copy $\hat{q}$ for each state $q \in Q$, and the new states q_T and q_{T_1}. We encode the actions of a on A as the actions of 00 in A'. The actions of b are transformed to 01. For the string 11 it holds that $\delta'(Q', 11) = S$. The component consisting of q_T and q_{T_1} ensures that any synchronizing word for A' starts with 11. For $t \in Q'$, we define the transition function δ' as

$$\delta'(t,0) := \begin{cases} \hat{q} & \text{if } t = q, \text{for } q \in Q \\ \delta(q,a) & \text{if } t = \hat{q}, \text{for } q \in Q \end{cases},$$

$$\delta'(t,1) := \begin{cases} s & \text{if } t = q, \text{for } q \in Q \setminus S \\ t & \text{if } t = q, \text{for } q \in S \\ \delta(q,b) & \text{if } t = \hat{q}, \text{for } q \in Q \end{cases},$$

and $\delta(q_T, 0) = \delta(q_{T_1}, 0) = q_T, \delta(q_T, 1) = q_{T_1}, \delta(q_{T_1}, 1) = s$ for some fixed $s \in S$. Note that A' is complete and deterministic. Assume $w \in \{a, b\}^*$ synchronizes S in A. Let $\varphi : \{a, b\} \to \{0, 1\}^*$ be an homomorphism with $a \mapsto 00, b \mapsto 01$. Then, $w' = 11\varphi(w)$ synchronizes A' with respect to $\mathcal{B}$. Starting from a state $q \in Q$ in A', it holds that $\delta'(q, 11) = s$ if $q \notin S$ and $\delta'(q, 11) = q$ otherwise. Also, the states q_T and q_{T_1} reach s with the string 11. Further for $\hat{q}, q \in Q$, it holds that $\delta'(\hat{q}, 11) = s$ if $\delta'(\hat{q}, 1) \notin S$ and $\delta'(\hat{q}, 11) = \delta'(\hat{q}, 1) \in S$, otherwise. Therefore, $\delta'(Q', 11) = S$. It is easy to see that $\varphi(v)$ with $v \in \{a, b\}$ acts on A' in the same as v acts on A. Hence, $11\varphi(w) \in L(\mathcal{B})$ is a synchronizing word for A'.

For the other direction, assume w synchronizes A' with respect to $\mathcal{B}$. The state q_T only transitions into the state set Q with two consecutive 1s but 11 can only appear as a prefix of any word in $L(\mathcal{B})$. Since q_T is not reachable from any state in $Q' \setminus \{q_T, q_{T_1}\}$, w must start with 11. As mentioned above, $\delta'(Q', 11) = S$. Since the states $\hat{q} \in Q'$ all have just one incident edge, a merging of states can only happen with a transition that reaches a state from Q. Hence, we can partition w into $11w_1w_2 \dots w_n$ such that after reading w up to 11 or to the end of some w_i, only states from Q are active in A', while after reading w up to some proper factor of any w_i, only states in $Q' \setminus Q$ are active in A'. Any state merging after the first 11 can only happen with the last letter of some w_i. Hence, $\varphi^{-1}(w_i)$ are single letters and $\varphi^{-1}(w_1w_2 \dots w_n)$ synchronizes S in A. □

To summarize, binary 3-state constraint automata offer easy synchronization problems as well as problems complete for NP and for PSPACE. However, we have no complete complexity picture here, giving a natural research question. Motivated by a remark of Rystsov [Rystsov, 1983] in a related setting, one could also ask if there are regular

language constraints that define synchronization problems that are complete for other levels of the polynomial-time hierarchy. We presented several criteria for a regular language L such that L-Constr-Sync $\in P$ as well as generalization-results to transfer the obtained hardness results for fixed L to larger classes of constraint languages, but a full classification of the complexity of L-Constr-Sync for regular constraint languages L is still an open research problem.

Acknowledgement. This project started during the workshop 'Modern Complexity Aspects of Formal Languages' that took place at Trier University 11.–15. February, 2019. The financial support of this workshop and in particular of the last two authors by the DFG-funded project FE560/9-1 is gratefully acknowledged. V. Gusev is supported by the Leverhulme Trust.

References

[Almeida et al., 2009] Almeida, J., Margolis, S., Steinberg, B., and Volkov, M. (2009). Representation theory of finite semigroups, semigroup radicals and formal language theory. *Transactions of the American Mathematical Society*, 361(3):1429–1461.

[Almeida et al., 2007] Almeida, M., Moreira, N., and Reis, R. (2007). Enumeration and generation with a string automata representation. *Theoretical Computer Science*, 387(2):93–102.

[Ananichev et al., 2013] Ananichev, D. S., Volkov, M. V., and Gusev, V. V. (2013). Primitive digraphs with large exponents and slowly synchronizing automata. *Journal of Mathematical Sciences*, 192(3):263–278.

[Bassino and Nicaud, 2007] Bassino, F. and Nicaud, C. (2007). Enumeration and random generation of accessible automata. *Theoretical Computer Science*, 381(1-3):86–104.

[Béal et al., 2011] Béal, M., Berlinkov, M. V., and Perrin, D. (2011). A quadratic upper bound on the size of a synchronizing word in one-cluster automata. *International Journal of Foundations of Computer Science*, 22(2):277–288.

[Berlinkov, 2014] Berlinkov, M. V. (2014). Approximating the minimum length of synchronizing words is hard. *Theory of Computing Systems*, 54(2):211–223.

[Berlinkov et al., 2018] Berlinkov, M. V., Ferens, R., and a, M. S. (2018). Complexity of preimage problems for deterministic finite automata. In Potapov, I., Spirakis, P. G., and Worrell, J., editors, *43rd International Symposium on Mathematical Foundations*

of Computer Science, MFCS, volume 117 of *LIPIcs*, pages 32:1–32:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Blondel and Portier, 2002] Blondel, V. D. and Portier, N. (2002). The presence of a zero in an integer linear recurrent sequence is NP-hard to decide. *Linear Algebra and its Applications*, 351-352:91–98.

[Černý, 1964] Černý, J. (1964). Poznámka k homogénnym experimentom s konečnými automatmi. *Matematicko-fyzikálny časopis*, 14(3):208–216.

[de Bondt et al., 2019] de Bondt, M., Don, H., and Zantema, H. (2019). Lower bounds for synchronizing word lengths in partial automata. *International Journal of Foundations of Computer Science*, 30(1):29–60.

[Domaratzki et al., 2002] Domaratzki, M., Kisman, D., and Shallit, J. (2002). On the number of distinct languages accepted by finite automata with n states. *Journal of Automata, Languages and Combinatorics*, 7(4):469–486.

[Fernau and Krebs, 2017] Fernau, H. and Krebs, A. (2017). Problems on finite automata and the exponential time hypothesis. *Algorithms*, 10(1):24.

[Gazdag et al., 2009] Gazdag, Z., Iván, S., and Nagy-György, J. (2009). Improved upper bounds on synchronizing nondeterministic automata. *Information Processing Letters*, 109(17):986–990.

[Gusev, 2012] Gusev, V. V. (2012). Synchronizing automata of bounded rank. In Moreira, N. and Reis, R., editors, *Implementation and Application of Automata - 17th International Conference, CIAA*, volume 7381 of *LNCS*, pages 171–179. Springer.

[Harrison, 1965] Harrison, M. A. (1965). A census of finite automata. *Canadian Journal of Mathematics*, 17:100–113.

[Hopcroft et al., 2001] Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2001). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2nd edition.

[Kannan and Lipton, 1986] Kannan, R. and Lipton, R. J. (1986). Polynomial-time algorithm for the orbit problem. *Journal of the ACM*, 33(4):808–821.

[Kohavi and Jha, 2009] Kohavi, Z. and Jha, N. K. (2009). *Switching and Finite Automata Theory*. Cambridge University Press, 3rd edition.

[Kozen, 1977] Kozen, D. (1977). Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science, FOCS*, pages 254–266. IEEE Computer Society.

[Larsen et al., 2014] Larsen, K. G., Laursen, S., and Srba, J. (2014). Synchronizing strategies under partial observability. In Baldan, P. and Gorla, D., editors, *Concurrency Theory - 25th International Conference, CONCUR*, volume 8704 of *LNCS*, pages 188–202. Springer.

[Martyugin, 2009] Martyugin, P. (2009). Complexity of problems concerning reset words for some partial cases of automata. *Acta Cybernetica*, 19(2):517–536.

[Martyugin, 2014] Martyugin, P. V. (2014). Computational complexity of certain problems related to carefully synchronizing words for partial automata and directing words for nondeterministic automata. *Theory of Computing Systems*, 54(2):293–304.

[Morawietz et al., 2020] Morawietz, N., Rehs, C., and Weller, M. (2020). A timecop's work is harder than you think. In Esparza, J. and Král', D., editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPIcs*, pages 71:1–71:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Reis et al., 2009] Reis, R., Moreira, N., and Almeida, M. (2009). On the representation of finite automata. *CoRR*, abs/0906.2477.

[Rystsov, 1983] Rystsov, I. K. (1983). Polynomial complete problems in automata theory. *Information Processing Letters*, 16(3):147–151.

[Sandberg, 2005] Sandberg, S. (2005). Homing and synchronizing sequences. In Broy, M., Jonsson, B., Katoen, J., Leucker, M., and Pretschner, A., editors, *Model-Based Testing of Reactive Systems, Advanced Lectures*, volume 3472 of *LNCS*, pages 5–33. Springer.

[Savitch, 1970] Savitch, W. J. (1970). Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192.

[Shitov, 2019] Shitov, Y. (2019). An improvement to a recent upper bound for synchronizing words of finite automata. *Journal of Automata, Languages and Combinatorics*, 24(2-4):367–373.

[Stockmeyer and Meyer, 1973] Stockmeyer, L. J. and Meyer, A. R. (1973). Word problems requiring exponential time (preliminary report). In *Proceedings of the fifth annual ACM Symposium on Theory of Computing, STOC*, pages 1–9. ACM.

[Szykuła, 2018] Szykuła, M. (2018). Improving the upper bound on the length of the shortest reset word. In Niedermeier, R. and Vallée, B., editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS*, volume 96 of *LIPIcs*, pages 56:1–56:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Trakhtman, 2007] Trakhtman, A. (2007). The černý conjecture for aperiodic automata. *Discrete Mathematics and Theoretical Computer Science*, 9(2).

[Volkov, 2008] Volkov, M. V. (2008). Synchronizing automata and the Černý conjecture. In Martín-Vide, C., Otto, F., and Fernau, H., editors, *Language and Automata Theory and Applications, Second International Conference, LATA*, volume 5196 of *LNCS*, pages 11–27. Springer.

A A More Algebraic Approach to NP

Let us first present a more direct proof of NP-hardness as claimed in Proposition 1. Occasionally, we make use of the notation $\delta^{-1}(q,a)$ for a transition function δ, referring to the set of those states p for which $\delta(p,a)=q$.

The reduction is from the NP-complete DFA-Intersection Nonemptiness problem for unary input languages [Stockmeyer and Meyer, 1973, Morawietz et al., 2020] and works as follows.

Take an arbitrary instance $A_1, A_2, \ldots, A_n$ of the DFA-Intersection Nonemptiness problem for unary input languages, or IE-UA for short, where the automata A_i are specified by $A_i = (\{b\}, Q^i, \delta^i, q_0^i, F^i)$. We modify each A_i by adding a fresh initial state p_0^i to the automaton and extending the transition function δ^i to the set $P^i := Q^i \cup \{p_0^i\}$ as follows: $\delta^i(p_0^i, b) := q_0^i$. Clearly, a word $w \in b^*$ is accepted by the DFA A_i if and only if the word bw is accepted by the DFA $A_i' := (\{b\}, P^i, \delta^i, p_0^i, F^i)$. Therefore, the answers to the instances $A_1, A_2, \ldots, A_n$ and $A_1', A_2', \ldots, A_n'$ of IE-UA coincide. W.l.o.g., we may assume that the state sets P^i for $1 \leq i \leq n$ are disjoint. Now consider the DFA A with the state set $Q := \{s\} \cup \bigcup_{i=1}^n P^i$, where s is a new state, and the input letters a and b whose actions at a generic state $q \in Q$ are defined as follows:

$$\delta(q,a) := \begin{cases} p_0^i & \text{if } q \in P^i \setminus F^i, \\ s & \text{otherwise;} \end{cases} \qquad \delta(q,b) := \begin{cases} \delta^i(q,b) & \text{if } q \in P^i, \\ s & \text{if } q = s. \end{cases} \tag{1}$$

Observe that by construction s is a sink state in A, so that it will be the synchronizing state.

We claim that A has a synchronizing word from the language ab^*a if and only if there exists a word in b^* which is accepted by all DFAs $A_1, A_2, \ldots, A_n$. Indeed, if $w \in b^*$ is accepted by all DFAs $A_1, A_2, \ldots, A_n$, then the word bw is accepted by all DFAs $A_1', A_2', \ldots, A_n'$. Let us verify that the word $abwa$ sends all states of A to the sink s. From (1) we see that $\delta(Q,a) = \{p_0^1, p_0^2, \ldots, p_0^n, s\}$. Since A_i accepts w, the DFA A_i' accepts bw, whence $\delta(p_0^i, bw) \in F^i$ for each $i = 1, 2, \ldots, n$. Thus, $\delta(Q, abw) \subseteq \{s\} \cup \bigcup_{i=1}^n F^i$, and from (1) we conclude that $\delta(Q, abwa) = \{s\}$.

Conversely, suppose that some $v \in ab^*a$ is a synchronizing word for A. Since s is a sink of A, $\delta(s,v) = s$ whence $\delta(Q,v) = \{s\}$. Decompose $v \in ab^*a$ as $v = aua$ with $u \in b^*$. The equality $\delta(Q, aua) = \{s\}$ forces the inclusion $\delta(Q, au) \subseteq \delta^{-1}(s,a) = \{s\} \cup \bigcup_{i=1}^n F^i$. On the other hand, $\delta(Q,a) \supseteq \{p_0^1, \ldots, p_0^n\}$, since $\delta(p_0^i, a) = p_0^i$ for each $i = 1, 2, \ldots, n$. Recall that $p_0^i \notin F^i$ by the construction of the DFA A_i'. Thus, the word $u \in b^*$ must bring

each state p_0^i to the state set $\{s\} \cup \bigcup_{i=1}^n F^i$. From (1) we see that this is only possible if $\delta(p_0^i, u) \in F^i$ for each $i = 1, 2, \ldots, n$, that is, $\delta^i(p_0^i, u) \in F^i$ in each DFA A_i'. Thus, the word u is accepted by all DFAs $A_1', A_2', \ldots, A_n'$. As already mentioned, $p_0^i \notin F^i$, whence u cannot be empty. Thus, $u = bw$ for some $w \in b^*$, and w is accepted by all DFAs $A_1, A_2, \ldots, A_n$.

It remains to show that synchronization subject to the language ab^*a is in NP. This is not completely trivial since there are synchronizing automata A for which the least N such that ab^Na is a synchronizing word for A is exponentially big with respect to the number of states in A. The number N can be guessed in binary, but a direct verification that ab^Na synchronizes automaton A may take exponential time. We are going to show how this difficulty can be bypassed.

Given a DFA $A = (\{a, b\}, Q, \delta)$ with $|Q| := n$, we first calculate the set $\delta(Q, a)$. Then for each $q \in \delta(Q, a)$, we calculate its *b-orbit* $\mathrm{Orb}_b(q)$, that is, the set

$$\mathrm{Orb}_b(q) = \{q, \delta(q, b), \delta(q, b^2), \ldots, \delta(q, b^k), \delta(q, b^{k+1}), \ldots, \delta(q, b^{k+\ell-1})\}$$

such that all states in $\mathrm{Orb}_b(q)$ are distinct but $\delta(q, b^{k+\ell}) = \delta(q, b^k)$. Let $k(q) := k$ and $\ell(q) := \ell$; these are nonnegative integers that do not exceed n. Observe that $\mathrm{Orb}_b(q)$ includes the *cycle* $\{\delta(q, b^k), \ldots, \delta(q, b^{k+\ell-1})\}$ of length $\ell(q)$; the rest $k(q)$ states in $\mathrm{Orb}_b(q)$ forms the *tail*. For each state $f \in Q$, we build a system $\Sigma_f := \bigwedge_{q \in \delta(Q,a)} \Sigma_{f,q}$ of number-theoretic conditions, where each $\Sigma_{f,q}$ is a disjunction of equalities $x = m$ and/or "threshold congruences", that is, conjunctions of the form

$$x \equiv m \pmod{d} \wedge x \geq t,$$

for certain nonnegative integers m and t and a positive integer d. Consider the following procedure.

1. Construct the set $T_f = \delta^{-1}(f, a)$.

2. For each $q \in \delta(Q, a)$, consider the intersection $\mathrm{Orb}_b(q) \cap T_f$. If for some q, the intersection is empty, the construction of Σ_f fails, and we proceed to the next state in Q.

3. Suppose that $\mathrm{Orb}_b(q) \cap T_f \neq \emptyset$ for each $q \in \delta(Q, a)$. For each $p \in \mathrm{Orb}_b(q) \cap T_f$, let k_p be the least number such that $p = \delta(q, b^{k_p})$. If $k_p \geq k(q)$, that is, p lies on the cycle of $\mathrm{Orb}_b(q)$, we append to $\Sigma_{f,q}$ as a disjunct the threshold congruence

$$x \equiv k_p \pmod{\ell(q)} \wedge x \geq k(q);$$

if p lies in the tail of $\mathrm{Orb}_b(q)$, we append to $\Sigma_{f,q}$ as a disjunct the equality $x = k_p$.

Clearly, constructing the system Σ_f requires polynomial in n time. Here is the key property of this gadget.

Lemma 8. *A number N satisfies the system Σ_f if and only if $\delta(Q, ab^N a) = \{f\}$.*

Proof. The equality $\delta(Q, ab^N a) = \{f\}$ amounts to saying that for each $q \in \delta(Q, a)$, the state $\delta(q, b^N)$ belongs to T_f, and the inclusion $\delta(q, b^N) \in T_f$ holds exactly when N satisfies one of the conditions whose disjunction is $\Sigma_{f,q}$. □

Now the non-deterministic polynomial algorithm for checking whether a given DFA $A = (\{a, b\}, Q, \delta)$ has a synchronizing word in ab^*a is clear: we build the systems Σ_f for each $f \in Q$, guess the binary representation of a non-negative integer N and check if this number satisfies one of these systems; the latter check takes polynomial time in the length of the binary representation of N which in turn is polynomial in the size of A. Lemma 8 ensures that the algorithm has a chance to succeed if and only if A has a synchronizing word of the form $ab^N a$.

Connections to the Matrix Orbit Problem

We will rely on the matrix orbit problem to present another proof that synchronization subject to ab^*a belongs to NP. Let $n := |Q|$ and $x \in \mathbb{N}_0^n$ be the characteristic vector of $\delta(Q, a)$, i.e., $x[i] = 1$ if $i \in \delta(Q, a)$ and $x[i] = 0$, otherwise. Further, the $(0, 1)$-adjacency matrix B of letter b is defined in such a manner that $B[i, j] = 1$ if and only if $\delta(i, b) = j$ for all $i, j \in Q$.

The k^{th} power of the matrix B would be denoted by B^k. Observe that $B^k[i, j]$ is equal to the number of walks of length k from i to j, which is at most 1 in our case as our automata are deterministic. Thus, $xB^k[j] = \sum_{i=1}^{n} x[i]B^k[i, j]$ is equal to the number of states mapped under the action of b^k from $\delta(Q, a)$ to j.

Observe now that for every synchronizing word $w = ua$ subject to $w \in ab^*a$ there is a vector y satisfying: *(i)* $y[j]$ is equal to the cardinality of $\delta^{-1}(j, u)$; *(ii)* the subset $\{j : y[j] > 0\}$ is synchronized by a; *(iii)* $xB^k = y$ holds true for some k. Moreover, every $y \in \mathbb{N}_0^n$ satisfying *(ii)* and *(iii)* gives rise to a synchronizing word subject to ab^*a.

Clearly, condition *(iii)* implies that the sum of entries of y is equal to $|\delta(Q, a)|$, as our input automaton is complete and deterministic and hence B^k has exactly a single one in each row. Every such y can be encoded in polynomial space and one can check condition *(ii)* in polynomial time. By the celebrated result of Kannan and Lipton [Kannan and

Lipton, 1986], condition *(iii)* can be verified in polynomial time as well, which confirms our initial claim that the problem belongs to NP.

Similarly to the preceding considerations, the problem whether an automaton is synchronized to a given state subject to ab^*a can be seen as an instance of the following equation in k: $xB^ky = c$, where $c \in \mathbb{Q}$ and $x, y \in \mathbb{Q}^n$ are given. It is not known whether the latter problem is decidable, but it is at least NP-hard [Blondel and Portier, 2002]. Our reduction can be seen as an alternative proof of this fact.

Chapter 8

Synchronization under Dynamic Constraints

Petra Wolf.

An extended abstract appeared in the proceedings of FSTTCS 2020:
Leibniz International Proceedings in Informatics (LIPIcs) 182 (2020) pp. 58:1 – 58:14.
DOI: 10.4230/LIPIcs.FSTTCS.2020.58.

This research paper was awarded with the 2021 publication prize of the Graduate Center of the University of Trier, Faculty IV (consisting of Maths, Computer Science, Economics, and Social Sciences).

Synchronization under Dynamic Constraints

Petra Wolf*

Universität Trier, Germany

Abstract

We introduce a new natural variant of the synchronization problem. Our aim is to model different constraints on the order in which a potential synchronizing word might traverse through the states. We discuss how a word can induce a state-order and examine the computational complexity of different variants of the problem whether an automaton can be synchronized with a word of which the induced order agrees with a given relation. While most of the problems are PSPACE-complete we also observe NP-complete variants and variants solvable in polynomial time. One of them is the careful synchronization problem for partial weakly acyclic automata (which are partial automata whose states can be ordered such that no transition leads to a smaller state), which is shown to be solvable in time $\mathcal{O}(k^2n^2)$ where n is the size of the state set and k is the alphabet-size. The algorithm even computes a synchronizing word as a witness. This is quite surprising as the careful synchronization problem uses to be a hard problem for most classes of automata. We will also observe a drop in the complexity if we track the orders of states on several paths simultaneously instead of tracking the set of active states. Further, we give upper bounds on the length of a synchronizing word depending on the size of the input relation and show that (despite the partiality) the bound of the Černý conjecture also holds for partial weakly acyclic automata.

1 Introduction

We call $A = (Q, \Sigma, \delta)$ a deterministic partial (semi-) automaton (DPA) if Q is a finite set of states, Σ is a finite alphabet, and $\delta\colon Q \times \Sigma \to Q$ is a (potentially partial) transition

*The author was supported by DFG-funded project FE560/9-1

function. If δ is defined for every element in $Q \times \Sigma$, we call A a deterministic complete (semi-) automaton (DCA). Clearly, every DCA is also a DPA. We do not specify any start and final states as we are only interested in the transition of states. A DCA $A = (Q, \Sigma, \delta)$ is *synchronizing* if there exists a word $w \in \Sigma^*$ such that w takes every state to the same state. In that case, we call w a *synchronizing word* for A. If we are only interested in synchronizing a subset of states $S \subseteq Q$ we refer to the problem as *subset synchronization*.

One of the oldest applications of the intensively studied topic of synchronizing automata is the problem of designing parts orienters, which are robots or machines that get an object in an (due to a lack of expensive sensors) unknown orientation and transform it into a defined orientation [Ananichev and Volkov, 2004]. In his pioneering work, Natarajan [Natarajan, 1986] modeled the parts orienters as deterministic complete automata where a state corresponds to a possible orientation of a part and a transition of some letter a from state q corresponds to applying the modifier corresponding to a to a part in orientation q. He proved that the synchronization problem is solvable in polynomial time for – what is later called – the class of *orientable automata* [Ryzhikov, 2019] if the cyclic order respected by the automaton is part of the input. Many different classes of automata have since been studied regarding their synchronization behavior. We refer to [Volkov, 2008, Béal and Perrin, 2016, Truthe and Volkov, 2019] for an overview. The original motivation of designing a parts orienter was revisited in [Türker and Yenigün, 2015] where Türker and Yenigün modeled the design of an assembly line, which again brings a part from an unknown orientation into a known orientation, where different modifiers have different costs. What has not been considered so far is that different modifiers can have different impact on the parts and as we do not know the current orientation we might want to restrict the chronology of applied modifiers. For example, if the part is a box with a fold-out lid, turning it upside-down will cause the lid to open. In order to close the lid one might need another modifier such as a low bar which brushes the lid and closes it again. To specify that a parts orienter should deliver the box facing upward with a closed lid one needs to encode something like: "When the box is in the state *facing down*, it later needs to be in the state *lid closed*". But this does not stop us from opening the lid again, so we need to be more precise and encode: "After **the last time** the box was in the state *facing down*, it needs to visit the state *lid closed* at least once". We will implement these conditions in our model of a parts orienter by enhancing a given DCA with a relation R. We will then consider different ways of how a synchronizing word implies an order on the states and ask whether there exists a synchronizing word whose implied state-order agrees with the input-relation R. The case-example above will be covered by the first two introduced orders. The third considered order relates to the following scenario: Let us again picture the box with the lid

in mind, but this time the box initially contains some water. We would like to have the box in a specific orientation with the lid open but the water should not be shed during orientating. We have a modifier that opens the lid and a modifier which rotates the box. Clearly we do not want the box to face downwards after the lid has been opened. So, we encode: "As soon as the state *lid open* has been reached, the state *facing downwards* should never be entered again".

For every type of dynamic constraint (which we will also call *order*), we investigate the computational complexity of the problem whether a given automaton admits a synchronizing word that transitions the states of the automaton in an order that is conform with a given relation. Thereby, we distinguish between tracking all active states simultaneously and tracking each state individually. We observe different complexities for different ordering concepts and get a good understanding of which ordering constraints yield tractable synchronization problems and which do not. The complexity of the problem also depends on how detailed we describe the allowed sequence of states.

2 Related Work

The problem of checking whether a synchronizing word exists for a given DCA $A = (Q, \Sigma, \delta)$ can be solved in time $\mathcal{O}(|Q|^2|\Sigma|)$, when no synchronizing word is computed, and in time $\mathcal{O}(|Q|^3)$ when a witnessing synchronizing word is demanded [Eppstein, 1990, Volkov, 2008]. In comparison, if we only ask for a subset of states $S \subseteq Q$ to be synchronized, the problem becomes PSPACE-complete for general DCAs [Sandberg, 2004]. These two problems have been investigated for several smaller classes of automata involving orders on states. Here, we want to mention the class of *oriented* automata whose states can be arranged in a cyclic order which is preserved by all transitions. This model has been studied among others in [Natarajan, 1986, Eppstein, 1990, Ananichev and Volkov, 2004, Ryzhikov and Shemyakov, 2018, Volkov, 2008]. If the order on the states is linear instead of cyclic, we get the class of *monotone* automata which has been studied in [Ananichev and Volkov, 2004, Ryzhikov and Shemyakov, 2018]. An automaton is called *aperiodic* [Béal and Perrin, 2016] if there is a non-negative integer k such that for any word w and any state q it holds that $\delta(q, w^k) = \delta(q, w^{k+1})$. An automaton is called *weakly acyclic* [Ryzhikov, 2019] if there exists an ordering of the states $q_1, q_2, \ldots, q_n$ such that if $\delta(q_i, a) = q_j$ for some letter $a \in \Sigma$, then $i \leq j$. In other words, all cycles in a WAA are self-loops. In Section 4 we will consider partial WAAs. The class of WAAs forms a proper subclass of the class of aperiodic automata. Each synchronizing aperiodic automaton admits a synchronizing word of length at most $n(n-1)/2$ [Trakhtman, 2007], whereas synchronizing WAAs admit synchronizing words

of linear lengths [Ryzhikov, 2019]. Asking whether an aperiodic automaton admits a synchronizing word of length at most k is an NP-complete task [Volkov, 2008] as it is for general DCAs [Rystsov, 1980, Eppstein, 1990]. The subset synchronization problem for WAAs, and hence for aperiodic automata, is NP-complete [Ryzhikov, 2019].

Going from complete automata to partial automata normally brings a jump in complexity. For example, the so called *careful synchronization* problem for DPAs asks for synchronizing a partial automata such that the synchronizing word w is defined on all states. The problem is PSPACE-complete for DPAs with a binary alphabet [Martyugin, 2014]. It is even PSPACE-complete for DPAs with a binary alphabet if δ is undefined for only one pair in $Q \times \Sigma$ [Martyugin, 2012]. The length of a shortest carefully synchronizing word $c(n)$, for a DPA with $|Q| = n$, differs with $\Omega(3^{\frac{n}{3}}) \leq c(n) \leq \mathcal{O}(4^{\frac{n}{3}} \cdot n^2)$ [Martyugin, 2012] significantly from the cubic upper-bound for complete automata. Also for the smaller class of monotone partial automata with an unbounded alphabet size, an exponential lower bound on the length of a shortest carefully synchronizing word is known, while for fixed alphabet sizes of 2 and 3 only a polynomial lower bound is obtained [Ryzhikov and Shemyakov, 2018]. The careful synchronization problem is NP-hard for partial monotone automata over a four-letter alphabet [Türker and Yenigün, 2015, Ryzhikov and Shemyakov, 2018]. It is also NP-hard for aperiodic partial automata over a three-letter alphabet [Ryzhikov, 2019]. In contrast we show in Section 4 that the careful synchronization problem is decidable in polynomial time for partial WAAs.

In [Ryzhikov, 2019, Ryzhikov and Shemyakov, 2018] several hardness and inapproximability results are obtained for WAAs, which can be transferred into our setting as depicted in Section 4. We will also observe W[1]-hardness results from the reductions given in [Ryzhikov, 2019]. So far, only little is known (see for example [Fernau et al., 2015, Vorel and Roman, 2015, Bruchertseifer and Fernau, 2019]) about the parameterized complexity of all the different synchronization variants considered in the literature.

While synchronizing an automaton under a given order, the set of available (or allowed) transitions per state may depend on the previously visited states on all paths. This dynamic of allowed transitions of a state depending on the history of chosen transition can also be observed in weighted and timed automata [Doyen et al., 2014]. More static constraints given by a second automaton have been discussed in [Fernau et al., 2019].

3 Problem Definitions

A deterministic semi-automaton $A = (Q, \Sigma, \delta)$ that might either be partial or complete is called an *automaton*. The transition function δ is generalized to words in the usual way. It is further generalized to sets of states $S \subseteq Q$ as $\delta(S, w) := \{\delta(q, w) \mid q \in S\}$. We sometimes refer to $\delta(S, w)$ as $S.w$. We call a state q *active* regarding a word w if $q \in Q.w$. If for some $w \in \Sigma^*$, $|Q.w| = 1$ we call $q \in Q.w$ a *synchronizing state*. We denote by $|S|$ the size of the set S. With $[i..j]$ we refer to the set $\{k \in \mathbb{N} \mid i \leq k \leq j\}$. For a word w over some alphabet Σ, we denote by $|w|$ the length of w, by $w[i]$ the i^{th} symbol of w (or the empty word ϵ if $i = 0$) and by $w[i..j]$ the factor of w from symbol i to symbol j. For each state q, we call the sequence of active states $q.w[i]$ for $0 \leq i \leq |w|$ the *path* induced by w starting at q. We expect the reader to be familiar with basic concepts in complexity theory, approximation theory and parameterized complexity theory. We refer to the textbooks [Cygan et al., 2015, Sipser, 1997, Ausiello et al., 1999]. as a reference.

We are now presenting different orders $\preccurlyeq_w$ which describe how a word traverses an automaton. We describe how a word implies each of the three presented orders. The first two orders relate the last visits of the states to each other, while the third type of order relates the first visits. We will then combine the order with an automaton A and a relation $R \subseteq Q^2$ given in the input and ask whether there exists a synchronizing word for A such that the implied order of the word agrees with the relation R. An order $\preccurlyeq_w$ agrees with a relation $R \subseteq Q^2$ if and only if for all pairs $(p, q) \in R$ it holds that $p \preccurlyeq_w q$, i.e., $R \subseteq \preccurlyeq_w$.

For any of the below defined orders $\preccurlyeq_w \subseteq Q \times Q$, we define the problem of *synchronization under order* and *subset synchronization under order* as:

Definition 1 (Sync-Under-$\preccurlyeq_w$)**.** Given a DCA $A = (Q, \Sigma, \delta)$ and a relation $R \subseteq Q^2$. Does there exist a word $w \in \Sigma^*$ such that $|Q.w| = 1$ and $R \subseteq \preccurlyeq_w$?

Definition 2 (Subset-Sync-Under-$\preccurlyeq_w$)**.** Given a DCA $A = (Q, \Sigma, \delta)$, $S \subseteq Q$, and a relation $R \subseteq Q^2$. Is there a word $w \in \Sigma^*$ with $|S.w| = 1$ and $R \subseteq \preccurlyeq_w$?

It is reasonable to distinguish whether the order should include the initial configuration of the automaton or if it should only describe the consequences of the chosen transitions. In the former case, we refer to the problem as Sync-Under-*0*-$\preccurlyeq_w$ (starting at $w[0]$), in the latter case as Sync-Under-*1*-$\preccurlyeq_w$ (starting at $w[1]$), and if the result holds for both variants, we simply refer to is as Sync-Under-$\preccurlyeq_w$. Examples for positive and negative instances of the problem synchronization under order for some discussed variants are illustrated in Figure 1. Let $\text{first}(q, w, S)$ be the function returning the minimum of

positions at which the state q appears as an active state over all paths induced by w starting at some state in S. Accordingly, let $\text{last}(q, w, S)$ return the maximum of those positions. Note that $\text{first}(q, w, S) = 0$ for all states $q \in S$ and > 0 for $q \in Q \backslash S$. If q does not appear on a path induced by w on S, then set $\text{first}(q, w, S) := |w| + 1$ and $\text{last}(q, w, S) := -1$. In the SYNC-UNDER-*1*-$\preccurlyeq_w$ problem variant, the occurrence of a state at position 0 is ignored (i.e., if q occurs only at position 0 while reading w on S, then $\text{last}(q, w, S) = -1$). In the following definitions let $A = (Q, \Sigma, \delta)$ be a DCA and let $p, q \in Q$. The following relations $\preccurlyeq_w$ are defined for every word $w \in \Sigma^*$.

Definition 3 (Order $l < l$ on sets)**.** $p \propto^{l<l}_{w@s} q :\Leftrightarrow \text{last}(p, w, Q) < \text{last}(q, w, Q)$.

Definition 4 (Order $l \leq l$ on sets)**.** $p \propto^{l\leq l}_{w@s} q :\Leftrightarrow \text{last}(p, w, Q) \leq \text{last}(q, w, Q)$.

The second order differs from the first in the sense that q does not have to appear finally without p, instead they can disappear simultaneously. Further, note that in comparison with order $\propto^{l<l}_{w@s}$, for a pair (p, q) in order $\propto^{l\leq l}_{w@s}$ it is not demanded that q is active after reading w up to some position $i > 0$. This will make a difference when we later consider the orders on isolated paths rather than on the transition of the whole state set. It can easily be verified that for any word $w \in \Sigma^*$ and any automaton $A = (Q, \Sigma, \delta)$ the order $\propto^{l<l}_{w@s}$ is a proper subset of $\propto^{l\leq l}_{w@s}$. For the order $\propto^{l\leq l}_{w@s}$, it makes no difference whether we take the initial configuration into account since states can disappear simultaneously.

So far, we only introduced orders which consider the set of active states as a whole. It did not matter which active state belongs to which path and a state on a path τ could stand in a relation with a state on some other path ρ. But, in most scenarios the fact that we start with the active state set Q only models the lack of knowledge about the *actual* current state. In practice only one state q is active and hence any constraints on the ordering of transitioned states should apply to the path starting at q. Therefore, we are introducing variants of order 1 and 2 which are defined on paths rather than on series of state sets.

Definition 5 (Order $l < l$ on paths)**.**

$$p \propto^{l<l}_{w@p} q :\Leftrightarrow \forall r \in Q\colon\ \text{last}(p, w, \{r\}) < \text{last}(q, w, \{r\}).$$

Definition 6 (Order $l \leq l$ on paths)**.**

$$p \propto^{l\leq l}_{w@p} q :\Leftrightarrow \forall r \in Q\colon\ \text{last}(p, w, \{r\}) \leq \text{last}(q, w, \{r\}).$$

The orders $\propto^{l<l}_{w@p}$ and $\propto^{l\leq l}_{w@p}$ significantly differ since the synchronization problem (starting at position 1) for $\propto^{l<l}_{w@p}$ is in NP while it is PSPACE-complete for $\propto^{l\leq l}_{w@p}$.

While the previously defined orders are bringing "positive" constraints to the future transitions of a word, in the sense that the visit of a state p will demand for a later visit of the state q (as opening the lid demands closing the lid later in our introductory example), we will now introduce an order which yields "negative" constraints. The third kind of order demands for a pair of states (p, q) that the (first) visit of the state q forbids any future visits of the state p (like do not turn the box after opening the lid). This stands in contrast to the previous orders where we could made up for a "forbidden" visit of the state p by visiting q again. The order $l < f$ will only be considered on paths since when we consider the state set Q, every pair in R would already be violated in position 0.

Definition 7 (Order $l < f$ on paths)**.**

$$p \propto_{w@p}^{l<f} q :\Leftrightarrow \forall r \in Q : \text{last}(p, w, \{r\}) < \text{first}(q, w, \{r\}).$$

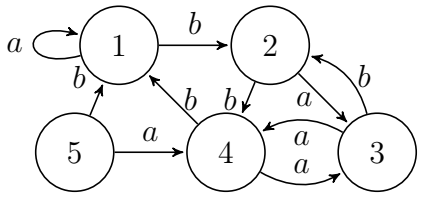
Note that $\propto_{w@p}^{l<f}$ is not transitive; e.g., for $R = \{(1, 2), (2, 3)\}$ we are allowed to go from 3 to 1 as long as we have not transitioned from 1 to 2 yet. For the order $l < f$, we will also consider the special case of R being a strict total order (irreflexive, asymmetric, transitive, and total).

Definition 8 (Sync-Under-Total-$\propto_{w@p}^{l<f}$)**.** Given a DCA $A = (Q, \Sigma, \delta)$, a strict and total order $R \subseteq Q^2$. Is there a word $w \in \Sigma^*$ with $|Q.w| = 1$ and $R \subseteq \propto_{w@p}^{l<f}$?

The orders on path could also be stated as LTL formulas of some kind which need to be satisfied on every path induced by a synchronizing word w and our hardness results transfer to the more general problem whether a given DCA can be synchronized by a word such that every path induced by w satisfies a given LTL formula. The orders on sets could be translated into LTL formulas which need to be satisfied on the path in the powerset-automaton starting in the state representing Q.

Using the temporal operators globally $\Box$, finally $\Diamond$, and until $\mathcal{U}$, we can express the orders $\propto^{l\leq l}$, $\propto^{l<l}$, and $\propto^{l<f}$ as follows, see [Manna and Pnueli, 1990, Chang et al., 1992] for details on these operators. For instance, the order $p \propto^{l\leq l} q$ can be expressed as $\Box(\Diamond q \vee \Box\neg p)$, meaning that globally it holds that finally q holds or globally p does not hold; the order $p \propto^{l<l} q$ can be expressed as $\Diamond(q \wedge \Box\neg p)$, meaning that finally q holds and from there on p does not hold anymore; and the order $p \propto^{l<f} q$ can be expressed as $\neg q \mathcal{U}(\Box\neg p)$, meaning that there is a position from which on p does not hold anymore and before that q does not hold.

Despite the similarity of the chosen orders and their translated LTL formulas we need different constructions for the considered orders as the presented attempts mostly do



	1	2	3	4	5
b	2	4	2	1	1
a	3	3	3	1	1
a	4	4	4	1	1
b	1	1	1	2	2
b	2	2	2	4	4
a	3	3	3	3	3

	✓	✗
$\propto_{w@s}^{l<l}$	$(1,2)$	$(2,4)$
$\propto_{w@s}^{l\leq l}$	$(2,4)$	$(2,1)$
$\propto_{w@p}^{l<l}$	$(1,2)$	$(4,5)$
$\propto_{w@p}^{l\leq l}$	$(5,5)$	$(2,4)$
$\propto_{w@p}^{l<f}$	$(5,2)$	$(4,3)$

Figure 1: DCA A (left) with all paths induced by $w = baabba$ (middle) and relations R consisting of single pairs forming a positive, resp. negative, instance for versions of Sync-Under-$\preccurlyeq_w$ (right).

		Synchronization				*Subset Synchronization*		
Order		$l < l$	$l \leq l$	$l < f$	$l<f$-tot	$l \not< / \leq l$	$l < f$	$l<f$-tot
Set	*0*	PSPACE-c	PSPACE-c	–	–	PSPACE-c	–	–
	1	PSPACE-c	PSPACE-c	–	–	PSPACE-c	–	–
Path	*0*	in NP	NP-hard	PSPACE-c	P	PSPACE-c	PSPACE-c	NP-c
	1	in NP	PSPACE-c	PSPACE-c	NP-c	PSPACE-c	PSPACE-c	NP-c

Table 1: Overview of the complexity for synchronization (on the left), and subset synchronization under order (on the right) for relations $\propto_{w@s}^{l<l}$, $\propto_{w@p}^{l<l}$, $\propto_{w@s}^{l\leq l}$, $\propto_{w@p}^{l\leq l}$, and $\propto_{w@p}^{l<f}$ (tot. is short for total).

not transfer to the other problems. Therefore, it is not to be expected that a general construction for restricted LTL formulas can be obtained. Our aim is to focus on restricting the order in which states appear and disappear on a path in the automaton or on a path in the powerset-automaton (remember the introductory example). Hence, we have chosen the stated definitions in order to investigate the complexity of problems where the LTL formula is always of the same type, i.e., comparing only the last or first appearances of states on a path. We leave it to future research to investigate other types of LTL formulas. In order to express synchronizability of Kripke structures, an extension to CTL has been introduced in [Chatterjee and Doyen, 2016]. Note that synchronization of Kripke structures is more similar to D3-directing words [Imreh and Steinby, 1999] for unary NFAs as in contrast to general DFAs the labels of the transitions are omitted in Kripke structures.

Finally, we introduce two problems from which we will reduce from in the next section.

Definition 9 (Careful Sync (PSPACE-complete [Martyugin, 2014]))**.** Given a DPA $A = (Q, \Sigma, \delta)$. Is there a word $w \in \Sigma^*$, s.t. $|Q.w| = 1$ and w is defined on all $q \in Q$?

Definition 10 (Vertex Cover (NP-complete [Sipser, 1997]))**.** Given a graph $G = (V, E)$ and an integer $k \leq |V|$. Is there a vertex cover $V' \subseteq V$ of size $|V'| \leq k$? A vertex cover is a set of states that contains at least one vertex incident to every edge.

4 Main Results

We now investigate the complexity of the introduced problems. An overview on the obtained results is given in Table 1.

Theorem 1. *For all orders* $\lessdot \in \{\propto_{w@s}^{l<l}, \propto_{w@p}^{l<l}, \propto_{w@s}^{l\leq l}, \propto_{w@p}^{l\leq l}, \propto_{w@p}^{l<f}\}$*, the problem* Sync-Under-$\lessdot$ *is contained in* PSPACE*. Further, it is* FPT *with parameter* $|Q|$*.*

Proof. Let $A = (Q, \Sigma, \delta)$ be a DCA and $R \subseteq Q^2$. We decide if there exists a synchronizing word $w \in \Sigma^*$ for A with $R \subseteq \lessdot_w$ for $\lessdot_w \in \{\propto_{w@s}^{l<l}, \propto_{w@p}^{l<l}, \propto_{w@s}^{l\leq l}, \propto_{w@p}^{l\leq l}, \propto_{w@p}^{l<f}\}$ by performing reachability tests in an enhanced version of the powerset-automaton $\mathcal{P}(A) = (\mathcal{P}(Q), \Sigma, \delta^{\mathcal{P}})$ for A. Therefore, we enhance $\mathcal{P}(A)$ with the information about the set of active pairs in R in every state. Here, a pair in R is active during the transition of a word if it constrains which states might be, or need to be visited in the future. For instance, in the example in Figure 1 concerning the order $\propto_{w@s}^{l<l}$, the pair $(2, 4)$ is active while reading the prefix ba, since the state 2 has appeared as an active state while the state 4 has not appeared without 2 as an active state yet. It is not active after reading baa, since now 4 is active without 2 and hence the pair $(2, 4)$ is satisfied and does not demand for further state visits. The pair becomes active again after reading $baab$ since again 2 became active demanding for the state 4 to become active without 2 again.

For the orders $\propto_{w@s}^{l<l}$ and $\propto_{w@s}^{l\leq l}$, we will enhance each state $\hat{q} \subseteq Q$ in $\mathcal{P}(A)$ with the information on the set of *active pairs* in R in the configuration represented by $\hat{q}$. For that, the state $\hat{q}$ will be copied $2^{|R|}$ times. For the orders $\propto_{w@p}^{l<l}$, $\propto_{w@p}^{l\leq l}$, $\propto_{w@p}^{l<f}$, we enhance every state $\hat{q} \subseteq Q$ in $\mathcal{P}(A)$ by a set S_t of active pairs in R *for each state* $t \in \hat{q}$. Here, the state $\hat{q}$ will be copied up to $|Q| \cdot 2^{|R|}$ times, and the size of the automaton $\mathcal{P}(A)$ is bounded by $2^{|Q|}|Q|2^{|R|} = 2^{\mathcal{O}(|Q|^2)}$. Hence, for every pair of start and end state the length of a shortest path connecting them is bounded by $2^{\mathcal{O}(|Q|^2)}$.

First, we clarify when a pair $(p, q) \in R$ is called active in a set state $\hat{q} \subseteq Q$, respectively in a state $t \in \hat{q}$, of the automaton $\mathcal{P}(A)$ by defining the transition function $\delta^{\mathcal{P}}$: For each $\hat{q} \subseteq Q$, $\sigma \in \Sigma$, we set $E := \{\delta(r, \sigma) \mid r \in \hat{q}\}$ and,

for each $S \subseteq R$ we set:

- $\propto_{w@s}^{l<l}$**:** $\delta^{\mathcal{P}}((\hat{q}, S), \sigma) = (E, (S \cup \{(p, q) \in R \mid p \in E\}) \setminus \{(p, q) \in S \mid q \in E \wedge p \notin E\})$,
- $\propto_{w@s}^{l\leq l}$**:** $\delta^{\mathcal{P}}((\hat{q}, S), \sigma) = (E, (S \cup \{(p, q) \in R \mid p \in E\}) \setminus \{(p, q) \in S \mid q \in E\})$,

for each $\hat{q} = \{q_1, q_2, \ldots, q_k\}$ and $S = \{(q_1, S_1), (q_2, S_2), \ldots, (q_k, S_k)\}$ we set:

- $\propto^{l<l}_{w@p}$: $\delta^{\mathcal{P}}((\hat{q},S),\sigma) = (E, \{(\delta(q_1,\sigma), S'_{\delta(q_1,\sigma)}), (\delta(q_2,\sigma), S'_{\delta(q_2,\sigma)}), \ldots, (\delta(q_k,\sigma), S'_{\delta(q_k,\sigma)})\})$ with $S'_{\delta(q_i,\sigma)} := \bigcup_{\{S_j \mid (q_j,S_j)\in S \wedge \delta(q_j,\sigma)=\delta(q_i,\sigma)\}} (S_j \cup \{(p,q) \in R \mid p = \delta(q_i,\sigma)\}) \setminus \{(p,q) \in R \mid q = \delta(q_i,\sigma)\}$,
- $\propto^{l\leq l}_{w@p}$: $\delta^{\mathcal{P}}((\hat{q},S),\sigma) = (E, \{(\delta(q_1,\sigma), S'_{\delta(q_1,\sigma)}), (\delta(q_2,\sigma), S'_{\delta(q_2,\sigma)}), \ldots, (\delta(q_k,\sigma), S'_{\delta(q_k,\sigma)})\})$ with $S'_{\delta(q_i,\sigma)} := \bigcup_{\{S_j \mid (q_j,S_j)\in S \wedge \delta(q_j,\sigma)=\delta(q_i,\sigma)\}} (S_j \cup \{(p,q) \in R \mid p = \delta(q_i,\sigma)\}) \setminus \{(p,q) \in R \mid q = \delta(q_i,\sigma)\}$,
- $\propto^{l<f}_{w@p}$: $\delta^{\mathcal{P}}((\hat{q},S),\sigma) = (E, \{(\delta(q_1,\sigma), S'_{\delta(q_1,\sigma)}), (\delta(q_2,\sigma), S'_{\delta(q_2,\sigma)}), \ldots, (\delta(q_k,\sigma), S'_{\delta(q_k,\sigma)})\})$ with $S'_{\delta(q_i,\sigma)} := \bigcup_{\{S_j \mid (q_j,S_j)\in S \wedge \delta(q_j,\sigma)=\delta(q_i,\sigma)\}} (S_j) \cup \{(p,q) \mid q = \delta(q_i,\sigma)\}$, if $\{(\delta(q_i,\sigma),q) \mid q \in Q\} \cap S'_{\delta(q_i,\sigma)} = \emptyset$ for all $(\delta(q_i,\sigma), S'_{\delta(q_i,\sigma)})$. Otherwise the transition yields to the error state $(\emptyset,\emptyset)$.

Generally speaking, the transition function $\delta^{\mathcal{P}}$ updates the set of active states according to δ and further updates the set of active pairs S according to the newly visited states. Thereby, regarding the orders 1 and 2, the visit of a new state can activate additional pairs or satisfy some pairs (p,q) and hence remove them from S while regarding to the third order a state visit can only activate more pairs. Here, a transition yields to the error state $(\emptyset,\emptyset)$ if it would violate any active pair.

We set Singl $:= \{(\{p\},\emptyset) \mid p \in Q\}$. Depending on the order, we define different start and final states for the automaton $\mathcal{P}(A)$.:

- $\propto^{l<l}_{w@s}$: Start state: (Q,R), final states: Singl.
- $\propto^{l\leq l}_{w@s}$: Start state: $(Q,\emptyset)$, final states: Singl.
- $\propto^{l<l}_{w@p}$: Start state: $(Q, \{(q_1, R\setminus\{(p,q) \in R \mid q = q_1\}), \ldots, (q_{|Q|}, R\setminus\{(p,q) \in R \mid q = q_{|Q|}\})\})$ for the case including $i = 0$, and $(Q, \{(q_1,R), (q_2,R), \ldots, (q_{|Q|},R)\})$ otherwise; final states: Singl.
- $\propto^{l\leq l}_{w@p}$: Start state: $(Q, \{(q_1,S_1), \ldots, (q_{|Q|},S_{|Q|})\})$ with $S_i := \{(p,q) \in R \mid p = q_i\}$ for the case including $i = 0$, and $(Q,\emptyset)$ otherwise; final states: Singl.
- $\propto^{l<f}_{w@p}$: Start state: $(Q, \{(q_1,S_1), \ldots, (q_{|Q|},S_{|Q|})\})$ with $S_i := \{(p,q) \in R \mid q = q_i\}$ for the case including $i = 0$, and $(Q,\emptyset)$ otherwise; final states: $(\{p\},*)$, every state with a singleton state set and an arbitrary set of active pairs.

In each case, the automaton A is synchronizable by a word w with $R \subseteq \lessdot_w$ if and only if the language accepted by $\mathcal{P}(A)$ is non-empty. This can be checked by non-deterministically stepwise guessing a path from the start state to some final state. Since each state contains only up to $|Q|+1$ bit-strings of length up to $|Q|^2$ a state of $\mathcal{P}(A)$ can

be stored in polynomial space. Hence, we can decide non-emptiness of $L(\mathcal{P}(A))$ in non-deterministic polynomial space and according to Savitch theorem [Savitch, 1970] we can also do this using deterministic polynomial space. Furthermore, since the size of $\mathcal{P}(A)$ is bounded by $2^{\mathcal{O}(|Q|^2)}$ a recursive search for a path from the start state to any final state can be done in time $2^{\mathcal{O}(|Q|^2)}$ which gives us an FPT algorithm in the parameter $|Q|$. □

After giving a general PSPACEupper bound, we now focus on lower bounds. First, we focus on the problem Sync-Under-$\propto_{w@s}^{l\leq l}$ and present a reduction from the PSPACE-complete problem of Careful Sync for DPAs. Since this problem is already PSPACE-complete for binary DPAs with one undefined transition [Martyugin, 2012], and the number of undefined transitions directly correlates to the size of the relation R, we get the following result:

Theorem 2. Sync-Under-$\propto_{w@s}^{l\leq l}$ *is PSPACE-complete, even for* $|R| = 1$ *and* $|\Sigma| = 2$.

Proof. We reduce from the PSPACE-complete Careful Sync problem for DPAs, see [Martyugin, 2012, Martyugin, 2014]. Let $A = (Q, \Sigma, \delta)$ be a DPA. We construct from A a DCA $A' = (Q' = Q \cup \{q_\ominus, r\}, \Sigma, \delta')$ with $q_\ominus, r \notin Q$. For every pair $q \in Q, \sigma \in \Sigma$ for which $\delta(q, \sigma)$ is undefined, we define the transition $\delta'(q, \sigma) = q_\ominus$. On all other pairs δ' agrees with δ. Further, for some arbitrary state $t \in Q$ and for all $\gamma \in \Sigma$ we set $\delta'(q_\ominus, \gamma) = \delta'(t, \gamma)$ (note that this can be $q_\ominus$ itself) and $\delta'(r, \gamma) = \delta'(t, \gamma)$. We set the relation R to $R := \{(q_\ominus, r)\}$.

Assume there exists a word $w \in \Sigma^*$, $|w| = n$ that synchronizes A without using an undefined transition. Then, $\delta(q, w[1])$ is defined for all states $q \in Q$. The letter $w[1]$ acts on A' in the following way: (1) $\delta'(r, w[1]) = \delta'(q_\ominus, w[1]) = \delta(t, w[1])$ which is defined by assumption; (2) $\delta'(Q, w[1]) \subseteq Q$ since $\delta(q, w[1])$ is defined for all states $q \in Q$. The combination of (1)-(2) yields $\delta'(Q', w[1]) \subseteq Q$. We further constructed δ' such that $\delta'(Q', w[1]) = \delta(Q, w[1])$. Since $\delta(q, w[2..n])$ is defined by assumption for every $q \in \delta(Q, w[1])$, δ' agrees with δ on $w[2..n]$ for every $q \in \delta(Q, w[1])$. This means especially that while reading $w[2..n]$ in A' on the states in $\delta'(Q', w[1])$ the state $q_\ominus$ is not reached and that $\delta'(Q', w) = \delta(Q, w)$. Therefore, w also synchronizes the automaton A'. The state $q_\ominus$ is only active in the start configuration where no letter of w is read yet and is not active anymore while reading w. The same holds for r, hence $R = \{(q_\ominus, r)\} \subseteq \propto_{w@s}^{l\leq l}$.

For the other direction, assume there exists a word $w \in \Sigma^*$, $|w| = n$ that synchronizes A' with $(q_\ominus, r) \in \propto_{w@s}^{l\leq l}$. Then, w can be partitioned into $w = uv$ with $u, v \in \Sigma^*$ where r is not active while reading the factor v in w. The only position of w in which r is active due to the definition of δ' is before any letter of w is read. Hence, we can set $u = \epsilon$ and $v = w$. As $(q_\ominus, r) \in \propto_{w@s}^{l\leq l}$ it holds for all $i \in [1..n]$ that $q_\ominus \notin \delta'(Q', v[1..i])$. Hence,

$\delta'(q, v)$ is defined for every state $q \in Q$. Since δ' and δ agree on the definition range of δ it follows that v also synchronizes the state set Q in A without using an undefined transition. □

Remark 1. The construction works for both variants (with and without 0) of the problem. It can further be adapted for the order $\propto_{w@s}^{l<l}$ (both variants) by introducing a copy $\hat{q}$ of every state in $Q \cup \{r\}$ and setting $\delta'(\hat{q}, \sigma) = q$ for every $\sigma \in \Sigma$, $q \in Q \cup \{r\}$. For all other transitions, we follow the above construction. We keep $R := \{(q_\ominus, r)\}$. Since r is left after $w[2]$ for any word $w \in \Sigma^*$ with $|w| \geq 2$ in order to satisfy R the state $q_\ominus$ needs to be left with $w[1]$ such that afterwards r is active without $q_\ominus$. Note that $q_\ominus$ has not been copied.

Corollary 1. Sync-Under-$\propto_{w@s}^{l<l}$ *is* PSPACE*-complete even for* $|R| = 1$ *and* $|\Sigma| = 2$.

Remark 2. The reduction presented in the proof of Theorem 2 can also be applied to show the PSPACE-completeness of Sync-Under-*1*-$\propto_{w@p}^{l\leq l}$. Since the state r cannot be reached from any other state, the state $q_\ominus$ needs to be left with the first letter of any synchronizing word and must not become active again on any path. The rest of the argument follows the proof of Theorem 2. Note that the construction only works for Sync-Under-*1*-$\propto_{w@p}^{l\leq l}$. If we consider Sync-Under-*0*-$\propto_{w@p}^{l\leq l}$ the problem might become easier. But it is at least NP-hard.

Theorem 3. *The problem* Sync-Under-*0*-$\propto_{w@p}^{l\leq l}$ *is* NP*-hard.*

Proof. We give a reduction from Vertex Cover. We refer to Figure 2 for a schematic illustration. Let $G(V, E)$ be a graph and let $k \in \mathbb{N}$. We construct from G a DCA $A = (Q, \Sigma, \delta)$ in the following way. We set $\Sigma = V \cup \{p\}$ for some $p \notin V$. We start with $Q = \{f, r, s\}$ where s is a sink state, meaning $\delta(s, \sigma) = s$ for all $\sigma \in \Sigma$, f will be the "false way" and r will be the "right way". We set $\delta(r, p) = \delta(f, p) = s$ and $\delta(r, v) = r$, $\delta(f, v) = f$ for all other $v \in \Sigma$. For every edge $e_{ij} \in E$ connecting some vertices $v_i, v_j \in V$, we create two states e_{ij} and $\hat{e}_{ij}$ and set $\delta(e_{ij}, v_i) = \delta(e_{ij}, v_j) = \hat{e}_{ij}$, $\delta(e_{ij}, p) = f$. For all other letters, we stay in e_{ij}. For the state $\hat{e}_{ij}$, we stay in $\hat{e}_{ij}$ for all letters except p. For p, we set $\delta(\hat{e}_{ij}, p) = s$. We further create for $1 \leq i \leq k + 2$ the states q_i with the transitions $\delta(q_i, v) = q_{i+1}$ for $i \leq k + 1$ and $v \in V$, $\delta(q_i, p) = q_i$ for $i \leq k$, and $\delta(q_{k+1}, p) = r$, $\delta(q_{k+2}, p) = s$, $\delta(q_{k+2}, v) = q_{k+2}$ for $v \in V$. We set $R := \{(q_1, r)\} \cup \{(e_{ij}, \hat{e}_{ij}) \mid e_{ij} \in E\}$.

If there exists a vertex cover of size $k' < k$ for G, then there also exists a vertex cover of size k for G. Therefore, assume V' is a vertex cover for G of size k. Then, the word wpp where w is any non-repeating listing of the vertices in V' is a synchronizing word for A with $R \subseteq \propto_{wpp@p}^{l\leq l}$. Since q_1 cannot be reached from any other state, the pair $(q_1, r) \in R$ is

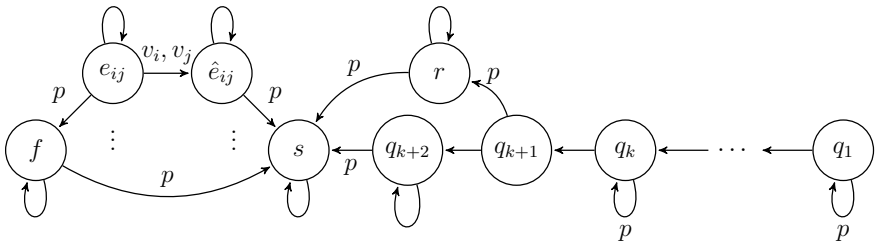


Figure 2: Schematic illustration of the reduction from VERTEX COVER (see Theorem 3). For each state, the transition without a label represents all letters which are not explicitly listed as an outgoing transition from that state.

trivially satisfied for each path starting in any state other than q_1. Hence, we only have to track the appearances of q_1 and r on the path starting in q_1. Since w lists the states in the vertex cover V' it holds that $|w| = k$ and hence $q_1.w = q_{k+1}$. Further, $q_1.wp = r$ and $q_1.wpp = s$. Hence, the pair (q_1, r) is satisfied on the path starting in q_1 as well as on all paths. It remains to show that wpp is indeed a synchronizing word and that all pairs in R of the form $(e_{ij}, \hat{e}_{ij})$ are satisfied. For every state e_{ij} representing an edge e_{ij}, the state $\hat{e}_{ij}$ is reached if we read a letter corresponding to a vertex incident to it. Since V' is a vertex cover, the word w contains for each edge e_{ij} at least one vertex incident to it. Hence, for each edge $e_{ij}.w = \hat{e}_{ij}$ and $e_{ij}.wpp = s$. Since each state e_{ij} is not reachable from any other state it follows that all pairs $(e_{ij}, \hat{e}_{ij})$ are satisfied by wpp on all paths. It is easy to see that for all other states $q \in Q$ it holds that $q.wpp = s$.

For the other direction, assume there exists a synchronizing word w for A with $R \subseteq \propto_{w@p}^{l \leq l}$. By the construction of A the word w must contain some letters p. Partition w into $w = upv$ where p does not appear in u. Since $R \subseteq \propto_{w@p}^{l \leq l}$ the pair (q_1, r) in R enforces $|u| \leq k$ since otherwise the only path on which q_1 appears (namely the one starting in q_1) will not contain the state r as for any longer prefix u it holds that $q_1.u = q_{k+2}$ and r is not reachable from q_{k+2}. The other pairs of the form $(e_{ij}, \hat{e}_{ij}) \in R$ enforces that u encodes a vertex cover for G. Assume this is not the case, then there is some state e_{ij} for which $e_{ij}.u = e_{ij}$. But then, $e_{ij}.up = f$ and from f the state $\hat{e}_{ij}$ is not reachable, hence the pair $(e_{ij}, \hat{e}_{ij})$ is not satisfied on the path starting in e_{ij}. Therefore, u encodes a vertex cover of size at most k. □

If we consider $\propto_{w@p}^{l<l}$, the two variants of the order (with and without position $i = 0$) do not differ since for a pair (p, q), regardless of whether p is reached, the state q must be reached on every path. Hence, whenever we leave q we must be able to return to it, so it does not matter if we consider starting in q or not. In comparison with SYNC-UNDER-*1*-$\propto_{w@p}^{l \leq l}$, the problem SYNC-UNDER-$\propto_{w@p}^{l<l}$ is solvable in polynomial time using non-determinism.

Theorem 4. *The problem* Sync-Under-$\propto_{w@p}^{l<l}$ *is in NP.*

Proof. Recall that in the problem Sync-Under-$\propto_{w@p}^{l<l}$, for every pair of states $(p, q) \in R$ and every state $r \in Q$, it is demanded that q appears somewhere on a path induced by the sought synchronizing word w, starting in r. Hence, a precondition for the existence of w is that for every pair $(p_i, q_i) \in R$ the states q_i must be reachable *from any state* in Q. More precisely, under the order $\propto_{w@p}^{l<l}$ only the last appearance of each state on a path is taken into account. Hence, a prohibited visit of a state can later be compensated by revisiting all related states in the correct order. Thus, it is sufficient to first synchronize all pairs of states and then transition the remaining state through all related states in the demanded order. The next Lemma 1 proves this claim and shows that these properties can be checked in non-deterministic polynomial time. □

Lemma 1. *Let $A = (Q, \Sigma, \delta)$ be a DCA and let $R \subseteq Q^2$. The automaton A is synchronizable by a word $w \in \Sigma^*$ with $R \subseteq \propto_{w@p}^{l<l}$ if and only if the following holds:*
(1) For every pair of states $q_i, q_j \in Q$, there exists a word w_{ij} such that $q_i.w_{ij} = q_j.w_{ij}$.
(2) For every state $r \in Q$, there exists a word w_r such that if we consider $\propto_{w@p}^{l<l}$ only on the path induced by w_r, which starts in r, it holds that $p \propto_{w_r@p}^{l<l} q$ for every pair $(p, q) \in R$.

Conditions (1) and (2) can be proved in polynomial time using non-determinism.

Proof. Assume A can be synchronized by a word $w \in \Sigma^*$ such that $R \subseteq \propto_{w@p}^{l<l}$. Then, $|Q.w| = 1$ and hence also $|\{q_i, q_j\}.w| = 1$ for every $q_i, q_j \in Q$. Since $R \subseteq \propto_{w@p}^{l<l}$ the condition (2) already holds by definition on every path induced by w.

For the other direction, assume condition (1) and (2) hold. Then, we can construct a synchronizing word $w = w_p w_r$ with $R \subseteq \propto_{w@p}^{l<l}$ in the following way: Start with $w_p = \epsilon$ and the set of active states $Q_{\text{act}} := Q$.

Step 1: If $\lvert Q_{\text{act}}\rvert = 1$ continue with Step 2, otherwise pick two arbitrary states $q_i, q_j \in Q_{\text{act}}$. Set $w_p := w_p w_{ij}$ and update $Q_{\text{act}} := Q_{\text{act}}.w_{ij}$. Repeat this step.
Step 2: We now have $Q_{\text{act}} = \{r\}$ for some state $r \in Q$. Return $w_p w_r$.

The algorithm terminates after at most $n = |Q|$ repetitions of Step 1 since by condition (1) in each iteration at least two states are merged. Obviously $w_p w_r$ is synchronizing for A. Further, condition (2) gives us that for every pair of states $(p, q) \in R$ the last appearance of q is on the path τ induced by w_r starting in r. This path appears on every path starting in any state $s \in Q$ as a suffix, because of $s.w_p = r$. Since for $p \propto_{w@p}^{l<l} q$ the order $\propto_{w@p}^{l<l}$ only considers the last appearance of q it follows that $R \subseteq \propto_{w_r@p}^{l<l}$ since every pair in R holds on the path τ.

The words w_{ij} in (1) can be found by determining reachability in the squared automaton $A \times A$ from the state (q_i, q_j) to any singleton state[1]. The words w_r in (2) can be computed in polynomial time using non-determinism. Let $B := \{q \in Q \mid \exists p \in Q\colon (p, q) \in R\}$ be the set of all second components of pairs in R with $m = |B|$. We guess an ordering $q_{i_1}, q_{i_2}, \ldots, q_{i_m}$ of the states in B corresponding to the last appearances of them on the path starting in r, induced by w_r. We compute the word w_r in the following way, starting with $w_r = \epsilon$:

Step 1: Check whether q_{i_1} is reachable from r by some word v using breadth-first search. If so, delete all states p with $(p, q_{i_1}) \in R$ from A and set $w_r := v$, otherwise return false.
Step 2: For each k with $1 \leq k < m$, check whether $q_{i_{k+1}}$ is reachable from q_{i_k} by some word v_k using breadth-first search. If so, delete all states p with $(p, q_{i_{k+1}}) \in R$ from A and set $w_r := w_r v_k$, otherwise return false.
Step 3: Return w_r.

If we guessed correctly, the algorithm returns a word w_r that satisfies condition (2). □

Remark 3. The NP-hardness proof for Sync-Under-*0*-$\propto_{w@p}^{l\leq l}$ in Theorem 3 and the NP-membership proof for Sync-Under-$\propto_{w@p}^{l<l}$ in Theorem 4 do not work for the respectively other problem since concerning $\propto_{w@p}^{l<l}$ the larger states need to be reached on *every* path and not only on a path containing the corresponding smaller state as it is the case concerning $\propto_{w@p}^{l\leq l}$.

Theorem 5. *The problem* Sync-Under-*0*-$\propto_{w@p}^{l<f}$ *is PSPACE-complete.*

Proof sketch. We reduce from Careful Sync. As in the proof of Theorem 2 we take every undefined transition $\delta(q, \sigma)$ to the new state $q_\ominus$. We further enrich the alphabet by a letter c and use c to take $q_\ominus$ into Q. We use the relation R and extra states r, s to enforce that c is the first letter of any synchronizing word, and that afterwards $q_\ominus$ is not reached again. □

Proof. We give a reduction from the problem Careful Sync. Let $A = (Q, \Sigma, \delta)$ be a DPA with $c \notin \Sigma$. We construct from A the DCA $A' = (Q', \Sigma \cup \{c\}, \delta')$ where $Q' = Q \cup \{q_\ominus, r, s\}$ with $Q \cap \{q_\ominus, r, s\} = \emptyset$. For every undefined transition $\delta(q, \sigma)$ with $q \in Q$, $\sigma \in \Sigma$ in A, we define the transition $\delta'(q, \sigma) = q_\ominus$ in A'. We set $\delta'(q, c) = q$ for $q \in Q$ and for some state t in Q we set $\delta(q_\ominus, c) = \delta'(r, c) = \delta'(s, c) = t$. For all other letters $\gamma \in \Sigma$, we set $\delta'(q_\ominus, \gamma) = q_\ominus$ and $\delta'(r, \gamma) = \delta'(s, \gamma) = s$. On all other transitions δ' agrees with δ. We set the relation R to $R := \{(s, r)\} \cup \{q_\ominus\} \times Q$.

[1]For more details see the algorithm in [Eppstein, 1990] which solves general synchronizability of a DCA in polynomial time.

Assume there exists a word $w \in \Sigma^*$, $|w| = n$ that synchronizes A without using an undefined transition, then cw synchronizes A' and $R \subseteq \propto_{cw@p}^{l<f}$. In the automaton A' the letter c transitions the state set $\{q_\ominus, r, s\}$ into Q. As c is the identity on the states in Q, we have $\delta'(Q', c) = Q$. Since δ' agrees with δ on all defined transitions of δ and $\delta(q, w)$ is by assumption defined for all states $q \in Q$ we have $\delta'(Q', cw) = \delta(Q, w) = \{p\}$ for some state $p \in Q$ and $\delta'(Q', cw[1..i]) \subseteq Q$ for all $i \leq |w|$. It remains to show that R is consistent with $\propto_{cw@p}^{l<f}$. As $q_\ominus$ is left with the prefix c and is not reached while reading w the subset $\{q_\ominus\} \times Q$ of R is fulfilled. The prefix c also causes the states r and s to transition into the state t (instead of s), and since s is not reachable from Q it is not the case that s appears after r on any path induced by cw.

For the other direction, assume there exists a word w that synchronizes A' with $R \subseteq \propto_{w@p}^{l<f}$. As $(s, r) \in R$ the path induced by w which starts in r must not contain the state s. Hence, the first symbol of w must be the letter c as otherwise r transitions into s. As $\delta'(Q', c) = Q$ all path labeled with c, starting in a state in Q', end in a state in Q. For every state $q \in Q$, $(q_\ominus, q)$ is contained in R. Hence, $q_\ominus$ must not appear on a paths labeled with w starting in a state in Q' after reading the first letter c of w. This means that the word $w[2..|w|]$ synchronizes the state set Q in the automaton A' without leaving the state set Q or using a transition which is undefined in A. Since δ' agrees with δ on all defined δ-transitions, $w[2..|w|]$ carefully synchronizes the state set Q in A. □

Remark 4. In the presented way, the reduction relies on taking the initial configuration at position $i = 0$ into account but we can adapt the construction to prove PSPACE-completeness of Sync-Under-*1*-$\propto_{w@p}^{l<f}$ by copying every state in Q and the state r. Denote a copy of a state q with q'. Then, for each letter σ, we set $\delta'(q', \sigma) = q$, for any copied state including r'. Note that the copied states are not reachable from any state. Now, after the first transition $w[1]$ (which can be arbitrary), we have a similar situation as previously considered for $w[0]$. The state r is active and forces the next letter to be the letter c; all states in Q are active; reading the letter c will cause all states q_σ to be left and never be reached again.

In the above reduction from Careful Sync the size of R depends on $|Q|$. Hence, the question whether Sync-Under-$\propto_{w@p}^{l<f}$ is PSPACE-hard for $|R| = 1$ is an interesting topic for further research. We will now see that when R is a strict and total order on Q, the problem of synchronizing under $\propto_{w@p}^{l<f}$ (a.k.a. Sync-Under-Total-$\propto_{w@p}^{l<f}$) becomes tractable.

Theorem 6. *Let $A = (Q, \Sigma, \delta)$, R be an instance of* Sync-Under-Total-$\propto_{w@p}^{l<f}$. *A shortest synchronizing word w for A with $R \subseteq \propto_{w@p}^{l<f}$ has length $|w| \leq \frac{|Q|(|Q|-1)}{2} + 1$.*

Proof. The relation R implies a unique ordering σ of the states in Q. We put a token

in every state which will be moved by applications of letters and think of active states as states containing a token. In the problem variant SYNC-UNDER-TOTAL-*1*-$\propto_{w@p}^{l<f}$ the tokens can be moved anywhere in the first step but afterwards - and in the variant SYNC-UNDER-TOTAL-*0*-$\propto_{w@p}^{l<f}$ - the tokens can only move to bigger states concerning σ. Each letter should move at least one token and the tokens in the $|Q|$ states can only be moved $0, 1, 2, \ldots, |Q| - 1$ times, giving a total length bound of $\frac{|Q|(|Q|-1)}{2} + 1$ for SYNC-UNDER-TOTAL-*1*-$\propto_{w@p}^{l<f}$ and $\frac{|Q|(|Q|-1)}{2}$ for SYNC-UNDER-TOTAL-*0*-$\propto_{w@p}^{l<f}$. □

Note that this length bound is smaller than the bound of the Černý conjecture for $|Q| > 3$ [Černý, 1964, Cerný, 2019]. The same bound can be obtained for SUBSET-SYNC-UNDER-TOTAL-$\propto_{w@p}^{l<f}$. We will now prove that the problem SYNC-UNDER-TOTAL-*0*-$\propto_{w@p}^{l<f}$ is equivalent – concerning polynomial time many-one-reductions (depicted by $\equiv_p$) – to the problem of carefully synchronizing a partial weakly acyclic automaton (PWAA) (a PWAA is a WAA where δ might be only partially defined). The obtained length bound also holds for PWAAs, which is only a quadratic increase w.r.t. the linear length bound in the complete case [Ryzhikov, 2019].

Theorem 7. SYNC-UNDER-TOTAL-*0*-$\propto_{w@p}^{l<f} \equiv_p$ CAREFUL SYNC *of PWAAs.*

Proof. We prove this statement by reducing the two problems to each other. Let $A = (Q, \Sigma, \delta)$, $R \subseteq Q^2$ be an instance of SYNC-UNDER-TOTAL-*0*-$\propto_{w@p}^{l<f}$. Since R is a strict total order on Q, we can order the states according to R. We construct from A the PWAA $A' = (Q, \Sigma, \delta')$ by removing all transitions in δ which are leading backwards in the order. Clearly, A' is carefully synchronizable if and only if A is synchronizable with respect to R.

For the other reduction, assume $A = (Q, \Sigma, \delta)$ is a PWAA. Then, we can order the states in Q such that no transition leads to a smaller state. We are constructing from A the DCA $A' = (Q \cup \{q_<\}, \Sigma, \delta')$ and insert $q_<$ as the smallest state in the state ordering. Then, we define in δ' all transitions (q, σ) for $q \in Q, \sigma \in \Sigma$ which are undefined in δ as $\delta'(q, \sigma) = q_<$. We take the state $q_<$ with every symbol to the maximal state in the order. Note that the maximal state needs to be the synchronizing state if one exists. We set $R = \{(p, q) \mid p < q \text{ in the state ordering of } Q \text{ in } A\} \cup \{(q_<, q) \mid q \in Q\}$. Every undefined transition (p, σ) in A is not allowed in A' at any time, since otherwise the pair $(q_<, p) \in R$ would be violated. The state $q_<$ itself can reach the synchronizing state with any transition. Hence, A' is synchronizable with respect to R if and only if A is carefully synchronizable. □

Remarks on the length bound of synchronizing words for PWAAs: In the reduction from CAREFUL SYNC of PWAAs to SYNC-UNDER-TOTAL-*0*-$\propto_{w@p}^{l<f}$ the state set is only

increased by one additional state $q_<$. As this state is not reachable from any other state (as otherwise the order would be violated) and is left into the largest state, w.r.t. the constructed order, with every letter, this state does not contribute to the length of a potential synchronizing word if the number of states is >1. As for all other states, the allowed transitions in the DCA act in the same way as they do in the PWAA, the length bound of a synchronizing word for DCAs w.r.t. Sync-Under-Total-*0*-$\propto_{w@p}^{l<f}$ translates to a length bound of a carefully synchronizing word for PWAAs. This is quite surprising as in general shortest carefully synchronizing words have an exponential lower bound [Martyugin, 2012]. Further, we show that careful synchronization for PWAAs is in P while the problem is PSPACE-complete for general DPAs even if only one transition is undefined [Martyugin, 2012].

Corollary 2. *For every PWAA $A = (Q, \Sigma, \delta)$, a shortest word w carefully synchronizing A has length $|w| \leq \frac{|Q|(|Q|-1)}{2}$.*

We can generalize the length bound obtained in the case when R is a total order on the whole state set to the case that R is only total for a subset of states.

Theorem 8. *Let $A = (Q, \Sigma, \delta)$, $R \subseteq Q^2$ with $n = |Q|$. Let $Q_1 \subseteq Q$ be such that R restricted to $Q_1 \times Q_1$ is a strict and total order. Let $p = |Q| - |Q_1|$. For* Sync-Under-$\propto_{w@p}^{l<f}$*: If A is synchronizable by a shortest word w with $R \subseteq \propto_{w@p}^{l<f}$, then: $|w| \leq (\frac{n(n-1)}{2} + 1) \cdot 2^p$.*

Proof. As before, the states in the set Q_1 can be ordered according to R and might only be traversed in this order. For every transition of a state in Q_1, in the worst case all possible combinations of active states in $Q \backslash Q_1$ might be traversed (once) yielding 2^p transitions with identical active states in Q_1 between two transitions of any state in Q_1. □

We now present an $\mathcal{O}(|\Sigma|^2|Q|^2)$ algorithm for Sync-Under-Total-*0*-$\propto_{w@p}^{l<f}$. The idea is the following: First, we delete all transitions that violate the state order. Then, we start on all states as the set of active states and pick a letter, which is defined on all active states and maps at least one active state to a larger state in the order R. We collect the sequence u of applied letters and after each step, we apply the whole sequence u on the set of active states. This is possible as we already know that u is defined on Q. We thereby ensure that a state which has become inactive after some iteration never becomes active again after an iteration step and hence Σ_{def} grows in each step and never shrinks. While a greedy algorithm which does not store u runs in $\mathcal{O}(|\Sigma||Q|^3)$, with this trick we get a running time of $\mathcal{O}(|\Sigma|^2|Q|^2)$. As in practice $|Q| \gg |\Sigma|$ this is a remarkable improvement. Note that we can store u compactly by only keeping the map induced by

the current u and storing the sequence of letters σ from which we can restore the value of u in each iteration.

Greedy algorithm for Sync-Under-Total-0-$\propto_{w@p}^{l<f}$ Let $A = (Q, \Sigma, \delta)$ be a DCA with $|Q| = n$ and $|\Sigma| = m$, and let $R \subseteq Q \times Q$ be a total order. We sketch an $\mathcal{O}(mn^3)$ greedy algorithm which computes a synchronizing word w for A with $R \subseteq \propto_{w@p}^{l<f}$.

First, order the set Q according to R. Delete all transitions in A which are leading backwards in the state-ordering obtaining the DPA A'. Set $Q_1 = Q$, $w_1 = \epsilon$.

At each step i: Check if $|Q_i| = 1$, if so return yes and the word w_i. Otherwise, compute $\Sigma_i = \{\sigma \in \Sigma \mid q.\sigma \text{ is defined for all } q \in Q_i\}$. Test if there is at least one letter $\sigma \in \Sigma_i$ that maps a state in Q_i to a larger state. If so, apply this letter to Q_i, obtaining $Q_{i+1} = Q_i.\sigma$, set $w_{i+1} = w_i\sigma$, and continue with the next step. If there is no such letter σ, return no.

By Theorem 7 there exists a synchronizing word w for A with $R \subseteq \propto_{w@p}^{l<f}$ if and only if there exists a carefully synchronizing word w for A'. Observe that if A' is carefully synchronizing, then every subset of Q can be synchronized. Hence, if A' is carefully synchronizing, then for every subset $S \subseteq Q$ there exists a letter σ which is defined on all states in S and maps at least one state in S to a larger state. Hence, the algorithm will find a carefully synchronizing word and terminate.

Conversely, if the algorithm returns no, the set Q_i of active states at the last step is a witness proving that A' is not carefully synchronizing, since no letter can map this subset to a different one, and thus Q cannot be synchronized.

The preprocessing of the algorithm takes time $\mathcal{O}(n \log n + mn)$. Each step of the algorithm takes time $\mathcal{O}(mn)$. The maximum number of steps is $\mathcal{O}(n^2)$, since at each step we move a token on the active states according to a total order by at least one. Hence, the total running time is $\mathcal{O}(mn^3)$.

Theorem 9. Sync-Under-Total-0-$\propto_{w@p}^{l<f}$ *is solvable in quadratic time.*

Proof. Let $A = (Q, \Sigma, \delta)$ be a DCA, and let $R \subseteq Q^2$ be a strict and total order on Q. Figure 3 describes an algorithm that decides in time $\mathcal{O}(|\Sigma|^2|Q|^2)$ whether A is synchronizable with respect to R under the order $\propto_{w@p}^{l<f}$ (including position $i = 0$) on paths. Despite the simplicity of the algorithm its correctness is not trivial and is proven in the following lemmas. □

Lemma 2. *The algorithm in Figure 3 terminates on every input $A = (Q, \Sigma, \delta)$ with $m = |\Sigma|$, $n = |Q|$, strict and total order $R \subseteq |Q|^2$ in time $\mathcal{O}(m^2n^2)$.*

Step 1: Order all states in Q according to the order R. Since R is strict and total the states can be ordered in an array $\{q_1, q_2, \ldots, q_n\}$.
Step 2: Delete in the automaton A all transitions which are leading backwards in the state-ordering. If this produces a state with no outgoing arc, abort; return false.
Step 3: Let q_n be the maximal state according to the order R. Delete all transitions in A which are labeled with letters $\sigma \in \Sigma$ for which $q_n.\sigma$ is undefined. If this produces a state with no outgoing transition, abort and return false.
Step 4: Partition the alphabet Σ into Σ_{def}, consisting of all letters $\sigma \in \Sigma$ for which $q.\sigma$ is defined for all states $q \in Q$, and $\Sigma_{\text{par}} := \Sigma \backslash \Sigma_{\text{def}}$. If $\Sigma_{\text{def}} = \emptyset$ abort; return false.
Step 5: Compute `explore`$(Q, Q, \Sigma_{\text{def}}, \epsilon)$ which returns Q_{act} and $u \in \Sigma_{\text{def}}^*$. The returned set of active states will equate $Q_{\text{trap}} = \{q \in Q \mid q.\Sigma_{\text{def}} = q\}$.
Step 6: Set $\Sigma_{\text{def}} := \Sigma_{\text{def}} \cup \{\sigma \in \Sigma_{\text{par}} \mid q.\sigma \text{ is defined for all } q \in Q_{\text{act}}\}$.
Compute `explore`$(Q, Q_{\text{act}}, \Sigma_{\text{def}}, u)$ which returns Q'_{act} and $u' \in \Sigma_{\text{def}}^*$.
Set $Q_{\text{act}} := Q'_{\text{act}}$, $u := u'$, $\Sigma_{\text{par}} := \Sigma \backslash \Sigma_{\text{def}}$.
Repeat this step until Q_{act} does not change anymore ($\equiv$ to Σ_{def} does not change anymore).
Then, if $Q_{\text{act}} = \{q_n\}$ return true, otherwise return false.
Procedure `explore`**:** Input: Ordered state set Q, set of active states Q_{act}, alphabet Σ_{exp} to be explored, word u with $Q.u = Q_{\text{act}}$.
Initialize a new word $u' := u$.
Go through the active states in order. For the current state q, test if any $\sigma \in \Sigma_{\text{def}}$ leads to a larger state, if so, perform the transition σu on all active states and update the set of active states Q_{act}. Concatenate u' with σu. Continue with the next larger active state (not that this can be $q.\sigma u$). If q_n is reached, return u', and the current set of active states Q_{act}.

Figure 3: Polynomial time algorithm for Sync-Under-Total-0-$\propto_{w@p}^{l<f}$ on the input $A = (Q, \Sigma, \delta)$, $R \subseteq Q^2$.

Proof. Step 1 can be performed in time $\mathcal{O}(n \log n)$ using the Quicksort-algorithm. Step 2 to Step 5 take time $\mathcal{O}(mn)$ each. The procedure `explore` takes time $\mathcal{O}(mn^2)$. The number of iterations in Step 6 is bounded by $|\Sigma_{\text{part}}|$ as Σ_{def} is applied exhaustively on Q_{act} and by invariant (2) of Lemma 3 we have $Q'_{\text{act}} \subseteq Q_{\text{act}}$, This yields a total run-time of $\mathcal{O}(m^2n^2)$. □

Lemma 3. *If the algorithm in Figure 3 returns true on the input $A = (Q, \Sigma, \delta)$, strict and total order $R \subseteq |Q|^2$, then A, R is a yes instance of* Sync-Under-Total-0-$\propto_{w@p}^{l<f}$.

Proof. For the procedure `explore`, the following invariant holds: Let u_{old} be the word u before the execution of `explore` and let u_{new} be the one after the execution of `explore`. Then, it holds for all executions of `explore` that (1) $Q.u_{\text{new}}$ is defined, (2) $Q.u_{\text{new}} \subseteq Q.u_{\text{old}}$, and (3) $Q.u_{\text{new}}u_{\text{new}} = Q.u_{\text{new}}$. We prove the invariant by induction. First, note that the word u computed by `explore` in Step 5 is defined on all states in Q since it only consists of letters which are defined on all states. Since we go through the states in order during the execution of `explore` and we only proceed with the next larger state if (1)

we where able to leave the current one towards a larger state or if (2) the current state cannot be left with any of the letters in Σ_{def}, it holds that $Q.uu = Q.u$. Also, trivially $Q.u \subseteq Q$.

Next, consider some later execution of `explore`. The new word computed by `explore` is of the form $u_{\text{new}} := u_{\text{old}}\sigma_1 u_{\text{old}}\sigma_2 u_{\text{old}} \dots \sigma_i u_{\text{old}}$ for some $0 \leq i \leq |Q|$. The induction hypothesis tells us that (1) $Q.u_{\text{old}}$ is defined. Since $Q.u_{\text{old}}\sigma_1$ is defined (since $\sigma_1 \in \Sigma_{\text{def}}$) and $Q.u_{\text{old}}\sigma_1 \subseteq Q$ it holds that $Q.u_{\text{old}}\sigma_1 u_{\text{old}}$ is defined. Further, since u_{old} brings all states to the set $Q.u_{\text{old}}$ it also brings a subset of Q to a subset of $Q.u_{\text{old}}$. Using the induction hypothesis (3) we get by an induction on i that $Q.u_{\text{new}}$ is defined and $Q.u_{\text{new}} \subseteq Q.u_{\text{old}}$. Since in the execution of `explore` we only proceed with the next larger state if we exhaustively checked all possible transitions for the current state and since $Q.u_{\text{old}}u_{\text{old}} = Q.u_{\text{old}}$ it follows that $Q.u_{\text{new}}u_{\text{new}} = Q.u_{\text{new}}$.

If the algorithm in the proof of Theorem 9 terminates and returns yes, it also returns a synchronizing word u. By the invariant proven above, we know that $Q.u$ is defined. This means that u never causes a transition of a larger state to a smaller state and hence $\propto_{u@p}^{l<f}$ agrees with R. During the execution of the algorithm we track the set of active states Q_{act} (starting with Q) and only return true if Q_{act} contains only the in R largest state q_n. Since R is a total order, every $q \in Q$ is smaller than q_n and hence q_n cannot be left. Therefore, q_n needs to be the single synchronizing state of A and u is a synchronizing word for A. □

Lemma 4. *If the algorithm in Figure 3 returns false on the input $A = (Q, \Sigma, \delta)$ and a strict and total order $R \subseteq |Q|^2$, then A is not synchronizable under the order $\propto_{w@p}^{l<f}$ with respect to the input order R .*

Proof. The algorithm returns false in the following cases.

(1) All outgoing transitions of some state q are deleted in Step 2. In that case, every transition of q leads to a smaller state. As this would violate the order R, we cannot perform any of those transitions. Hence, q cannot be left. (The case that $q = q_n$ is treated in (2).)

(2) Since q_n is the largest state, it cannot be left. Hence, q_n will be active the whole time. Therefore, any transition which is not defined for q_n cannot be taken at all since q_n is active during the whole synchronizing process. Hence, we can delete these transitions globally. If this creates a state which cannot be left anymore, this state cannot be synchronized.

(3) The execution of `explore` returns two identical sets of active states Q_{act} in a row.

Let Σ_{def} be the explored alphabet of the last execution of `explore`. Then, Σ_{def} contains all letters σ from Σ for which $q.\sigma$ is defined on all states $q \in Q_{\text{act}}$ and none of them leads some state in Q_{act} to a larger state. Since the relation R forbids cycles, for all $\sigma \in \Sigma_{\text{def}}$ and all $q \in Q_{\text{act}}$ $q.\sigma = q$ and hence this set cannot be left when all states of the set are active simultaneously. Since all states are active at the beginning of the algorithm, also all states in Q_{act} are active and since this set cannot be left with any transition which does not cause an undefined transition for all states in the set, the state set cannot be synchronized at all. □

Corollary 3. *The careful synchronization problem for PWAA is in P.*

If we allow one unrestricted transition first (Sync-Under-Total-*1*-$\propto_{w@p}^{l<f}$) the problem is related to the subset synchronization problem of complete WAAs which is NP-complete [Ryzhikov, 2019]. Together with the quadratic length bound of a synchronizing word of Sync-Under-Total-*1*-$\propto_{w@p}^{l<f}$ (which implies membership of Sync-Under-Total-*1*-$\propto_{w@p}^{l<f}$ in NP), we get:

Theorem 10. *The problem* Sync-Under-Total-*1*-$\propto_{w@p}^{l<f}$ *is NP-complete.*

Proof. We reduce from the NP-complete problem: Given a complete weakly acyclic automaton $A = (Q, \Sigma, \delta)$ and a subset $S \subseteq Q$, does there exist word $w \in \Sigma^*$ such that $|S.w| = 1$. We construct from A an automaton $A' = (Q', \Sigma \cup \{c\}, \delta')$ with $c \notin \Sigma$ in the following way. A schematic illustration of the construction is depicted in Figure 4. We start with $Q' = Q$. W.l.o.g., assume $|S| \geq 2$. For each state $q \in S$, we add a copy $\hat{q}$ to Q'. Further, we add the states $q_<$ and $q_>$. Let $q_1, q_2, \ldots, q_n$ be an ordering of the states in Q such that δ follows this ordering. The transition function δ' agrees with δ on all states in Q and letters in Σ. For a copied state $\hat{q}$, we set $\delta'(\hat{q}, \sigma) = \hat{q}$ for all $\sigma \in \Sigma$ and $\delta'(\hat{q}, c) = q$. For every state $q \in Q$, we set $\delta'(q, c) = q_<$. Let q_s be some state in S. Then for all $\sigma \in \Sigma$ we set $\delta'(q_<, \sigma) = \delta(q_s, \sigma)$, $\delta'(q_<, c) = q_s$ and $\delta'(q_>, \sigma) = q_>$, $\delta'(q_>, c) = q_s$. Then, we set $R = \{(q_i, q_j) \mid i < j\}$ for all states in Q. Further, for every copied state $\hat{q_k}$ we extend R by the sets: $\{(\hat{q_k}, q_k)\}$, $\{(q_i, \hat{q_k}), (\hat{q_k}, q_j) \mid i < k, k < j\}$, and $\{(\hat{q_i}, \hat{q_k}), (\hat{q_k}, \hat{q_j}) \mid i < k < j\}$ for all copied states $\hat{q_i}, \hat{q_j}$. For the states $q_<, q_>$, we add $\{(q_<, q) \mid q \neq q_< \in Q'\}$ and $\{(q, q_>) \mid q \neq q_> \in Q'\}$ to R.

Assume, $w \in \Sigma^*$ synchronizes the set S in A. W.l.o.g., assume $w \neq \epsilon$. Then, cw synchronizes the automaton A' such that $R \subseteq \propto_{w@p}^{l<f}$ (where position $i = 0$ is not taken into account). We have in A' that $Q'.c = S \cup \{q_<\}$. Since the initial configuration is not taken into account all transitions are allowed as the first letter of a synchronizing word and hence $R \subseteq \propto_{c@p}^{l<f}$. From now on, no transition which leads backwards in the order is allowed. We constructed R such that for the states in Q all transitions inherited from δ

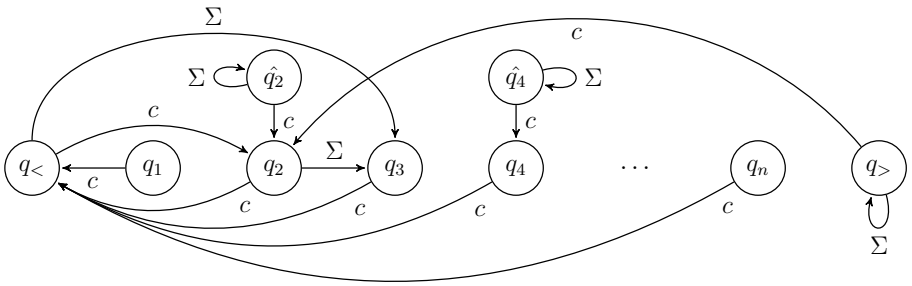


Figure 4: Schematic illustration of the reduction from the subset synchronization problem for complete weakly acyclic automata (see Theorem 10). In this example, the subset S contains the states q_2 and q_4, we picked $q_s = q_2$. Transitions inherited from the original automaton A are not drawn except for the transitions from q_2, for illustration they were assumed to lead to q_3.

are valid at any time. Since $w \in \Sigma^*$, all transitions induced by w are valid for states in Q in A'. The only active state outside of Q is $q_<$ which mimics transitions of the active state q_s with the next letter $w[1] \in \Sigma$ and hence q_s and $q_<$ are synchronized in the next step. Note that with any word from Σ^* no state outside of Q is reachable from a state in Q. Hence, $Q'.cw = S.w$.

For the other direction, assume $w \in \Sigma'^*$ synchronizes A' and $R \subseteq \propto_{w@p}^{l<f}$ (where position $i = 0$ is not taken into account). Then, the first letter of w needs to be the letter c. Otherwise, the state $q_>$ stays in $q_>$. This state cannot be left later anymore since after the first transition the pair $(q_s, q_>)$ in R is active and forbids a transition out of $q_>$. As the state $q_>$ cannot be reached from any other state we caused an active non-synchronizing trap-state.

For the letter c, we have $Q'.c = S \cup \{q_<\}$. After the first transition for all active states in Q, the transition by letter c is not allowed anymore as it would yield the states to reach the state $q_<$ which is smaller than any state in Q. For all letters $\sigma \neq c$, the state $q_<$ simulates the active state q_s and hence synchronizes with it with the letter $w[2]$. Starting from Q we stay in Q with all $\sigma \in \Sigma$ and simulate the automaton A. Hence, any word that synchronizes the set $S \cup \{q_<\}$ in A' also synchronizes the set S in A. □

Theorem 11. Subset-Sync-Under-Total-$\propto_{w@p}^{l<f}$ *is NP-complete.*

Proof. By Theorem 7 we know that the automata – which are synchronizable under the constraint formulated in Sync-Under-Total-*0*-$\propto_{w@p}^{l<f}$ – are precisely the synchronizable weakly acyclic automata. Since every complete weakly acyclic automaton (CWAA) is also a PWAA the NP-hardness of the subset synchronization problem for CWAAs transfers to the problem Subset-Sync-Under-Total-*0*-$\propto_{w@p}^{l<f}$ in our setting. Since the

problem Sync-Under-Total-*1*-$\propto_{w@p}^{l<f}$ is already NP-hard it follows by setting $S := Q$ that Subset-Sync-Under-Total-*1*-$\propto_{w@p}^{l<f}$ is also NP-hard. The length bound obtained in Theorem 6 also holds for a shortest word w synchronizing a subset S with $w \in \propto_{w@p}^{l<f}$ if R is a strict and total order. This gives membership in NP as we can guess the synchronizing word. □

Corollary 4. *Let $A = (Q, \Sigma, \delta)$ with $n = |Q|$, $S \subseteq Q$, and $R \subseteq Q^2$ be a strict and total order on Q. If S is synchronizable in A by a shortest word w such that $R \subseteq \propto_{w@p}^{l<f}$, then $|w| \leq \frac{n(n-1)}{2} + 1$ for* Subset-Sync-Under-Total-$\propto_{w@p}^{l<f}$.

Theorem 12. *The following subset synchronization problems are PSPACE-complete for both -0- and -1-:* Subset-Sync-Under-$\propto_{w@s}^{l<l}$, -$\propto_{w@p}^{l<l}$, -$\propto_{w@s}^{l\leq l}$, -$\propto_{w@p}^{l\leq l}$, -$\propto_{w@p}^{l<f}$.

Proof. For the mentioned orders, the subset synchronization problem is trivially PSPACE-hard which can be observed by setting $R = \emptyset$. In order to show membership in PSPACE, the start states in the considered powerset-construction in Theorem 1 can be adapted to check reachability from the start configuration where exactly the subset S is active to some final state using polynomial space. □

Several other results can be transferred from [Ryzhikov, 2019] to the corresponding version of the Sync-Under-Total-*0*-$\propto_{w@p}^{l<f}$ problem, such as inapproximability of the problems of finding a shortest synchronizing word; a synchronizing set of maximal size (here also W[1]-hardness can be observed); or determining the rank of a given set. Further, by the observation (in [Ryzhikov, 2019]) that, in the construction given in [Rystsov, 1980, Eppstein, 1990] the automata are WAAs, we immediately get NP-hardness for finding a shortest synchronizing word for all of our orders (for order $l < l$ and $l \leq l$ set $R = \emptyset$).

Corollary 5. *For all considered orders $\lessdot_w$, the problem given a DCA $A = (Q, \Sigma, \delta)$, $k \in \mathbb{N}$, $R \in Q^2$, if there exist a synchronizing word $w \in \Sigma^*$ with $|w| \leq k$ and $R \subseteq \lessdot_w$ is NP-hard.*

5 Transferred Results

In [Ryzhikov, 2019] weakly acyclic automata are considered, which are complete deterministic automata for which the states can be ordered such that no transitions leads to a smaller state in the order. We proved in Theorem 7 that the class of automata considered in Sync-Under-Total-*0*-$\propto_{w@p}^{l<f}$ is equivalent to the class of partial weakly acyclic automata (PWAA). In [Ryzhikov, 2019] the corresponding class of complete weakly acyclic

automata is investigated and several hardness results are obtained for different synchronization problems concerning this class of automata. Since complete weakly acyclic automata are a subclass of partial weakly acyclic automata the obtained hardness results easily transfer into our setting. Note that in [Ryzhikov, 2019] the approximation results are measured in $n = |Q|$ and not in the size of the input. Hence, the results can be directly transferred despite the fact that in the problem SYNC-UNDER-TOTAL-*0*-$\propto_{w@p}^{l<f}$ the input is extended to include the set R of size $|Q|^2$. We refer to the decision variant of an optimization problem by the extension *-D* in its name. The following results transfer from [Ryzhikov, 2019]:

Definition 11 (SHORT-SYNC-WORD-TOTAL-*0*-$\propto_{w@p}^{l<f}$)**.** Given a DCA $A = (Q, \Sigma, \delta)$, and a strict and total order $R \subseteq Q^2$. Output the length of a shortest word w such that $|Q.w| = 1$ and $R \subseteq \propto_{w@p}^{l<f}$.

Corollary 6. *The problem* SHORT-SYNC-WORD-TOTAL-*0*-$\propto_{w@p}^{l<f}$ *for n-state binary automata cannot be approximated in polynomial time within a factor of $\mathcal{O}(n^{\frac{1}{2}-\epsilon})$ for any $\epsilon > 0$ unless $P = NP$.*

Definition 12 (MAX-SYNC-SET-TOTAL-*0*-$\propto_{w@p}^{l<f}$)**.** Given a DCA $A = (Q, \Sigma, \delta)$, and a strict and total order $R \subseteq Q^2$. Output a set $S \subseteq Q$ of maximum size such that $|S.w| = 1$ and $R \subseteq \propto_{w@p}^{l<f}$.

Corollary 7. *The problem* MAX-SYNC-SET-TOTAL-*0*-$\propto_{w@p}^{l<f}$ *for n-state automata over an alphabet of cardinality $\mathcal{O}(n)$ cannot be approximated in polynomial time within a factor of $\mathcal{O}(n^{1-\epsilon})$ for any $\epsilon > 0$ unless $P = NP$.*

Corollary 8. *The problem* MAX-SYNC-SET-TOTAL-*0*-$\propto_{w@p}^{l<f}$ *for binary n-state automata cannot be approximated in polynomial time within a factor of $\mathcal{O}(n^{\frac{1}{3}-\epsilon})$ for any $\epsilon > 0$ unless $P = NP$.*

Observing the reductions given in [Ryzhikov, 2019] to obtain the above transferred inapproximability results, we also conclude the following hardness results concerning the parameterized complexity class W[1].

Corollary 9. *The problem* MAX-SYNC-SET-TOTAL-*0*-$\propto_{w@p}^{l<f}$*-D is W[1]-hard with the parameter k being the given size bound on the set S in the decision variant of the problem.*

Definition 13 (SET-RANK-TOTAL-*0*-$\propto_{w@p}^{l<f}$)**.** Given a DCA $A = (Q, \Sigma, \delta)$, a subset $S \subseteq Q$ and a strict and total order $R \subseteq Q^2$. Output the rank of S in A under $\propto_{w@p}^{l<f}$, that is the size of a smallest set S' such that there exists a word w with $S.w = S'$ and $R \subseteq \propto_{w@p}^{l<f}$.

Corollary 10. *The problem* Set-Rank-Total-*0*-$\propto^{l<f}_{w@p}$ *for n-state automata with alphabet of size* $\mathcal{O}(\sqrt{n})$ *cannot be approximated within a factor of* $\mathcal{O}(n^{\frac{1}{2}-\epsilon})$ *for any* $\epsilon > 0$ *unless* $P = NP$.

Corollary 11. *The problem* Set-Rank-Total-*0*-$\propto^{l<f}_{w@p}$ *for n-state binary automata cannot be approximated within a factor of* $\mathcal{O}(n^{\frac{1}{3}-\epsilon})$ *for any* $\epsilon > 0$ *unless* $P = NP$.

Definition 14 (Sync-Into-Subset-*0*-$\propto^{l<f}_{w@p}$)**.** Given a DCA $A = (Q, \Sigma, \delta)$, a subset $S \subseteq Q$ and a strict and total order $R \subseteq Q^2$. Does there exist a word w with $Q.w = S$ and $R \subseteq \propto^{l<f}_{w@p}$.

Corollary 12. *The problem* Sync-Into-Subset-*0*-$\propto^{l<f}_{w@p}$ *is NP-hard.*

6 Conclusion

We discussed ideas how constraints for the design of assembly lines caused by the physical deformation of a part can be described in terms of synchronization problems. For that, we considered several ways how a word can imply an order of states in Q. We considered the complexity of synchronizing an automaton under different variants of orders and observed that the complexity of considering an order on the set of active states may differ from considering the order on each single path. Although we were able to get a good understanding of the complexity of synchronization under the considered orders, some questions remained open: We only know that Sync-Under-$\propto^{l<l}_{w@p}$ is contained in NP but it is open whether the problem is NP-complete or if it can be solved in polynomial time. Conversely, for Sync-Under-*0*-$\propto^{l\leq l}_{w@p}$ the problem is NP-hard but its precise complexity is unknown. It would be quite surprising to observe membership in NP here since it would separate the complexity of this problem from the closely related problem Sync-Under-*1*-$\propto^{l\leq l}_{w@p}$. Further, it remains open whether for the other orders a drop in the complexity can be observed, when R is strict and total, as it is the case for $\propto^{l<f}_{w@p}$.

References

[Ananichev and Volkov, 2004] Ananichev, D. S. and Volkov, M. V. (2004). Synchronizing monotonic automata. *Theoretical Computer Science*, 327(3):225–239.

[Ausiello et al., 1999] Ausiello, G., Protasi, M., Marchetti-Spaccamela, A., Gambosi, G., Crescenzi, P., and Kann, V. (1999). *Complexity and Approximation: Combinatorial*

Optimization Problems and Their Approximability Properties. Springer-Verlag, Berlin, Heidelberg, 1st edition.

[Béal and Perrin, 2016] Béal, M.-P. and Perrin, D. (2016). *Synchronised Automata*, page 213–240. Encyclopedia of Mathematics and its Applications. Cambridge University Press.

[Bruchertseifer and Fernau, 2019] Bruchertseifer, J. and Fernau, H. (2019). Synchronizing series-parallel automata with loops. In Freund, R., Holzer, M., and Sempere, J. M., editors, *Eleventh Workshop on Non-Classical Models of Automata and Applications, NCMA 2019, Valencia, Spain, July 2-3, 2019.*, pages 63–78. Österreichische Computer Gesellschaft.

[Černý, 1964] Černý, J. (1964). Poznámka k homogénnym eksperimentom s konečnými automatami. *Matematicko-fyzikalny Časopis Slovensk*, 14(3):208–215.

[Cerný, 2019] Cerný, J. (2019). A note on homogeneous experiments with finite automata. *Journal of Automata, Languages and Combinatorics*, 24(2-4):123–132.

[Chang et al., 1992] Chang, E. Y., Manna, Z., and Pnueli, A. (1992). Characterization of temporal property classes. In Kuich, W., editor, *Automata, Languages and Programming, 19th International Colloquium, ICALP92, Vienna, Austria, July 13-17, 1992, Proceedings*, volume 623 of *Lecture Notes in Computer Science*, pages 474–486. Springer.

[Chatterjee and Doyen, 2016] Chatterjee, K. and Doyen, L. (2016). Computation tree logic for synchronization properties. In Chatzigiannakis, I., Mitzenmacher, M., Rabani, Y., and Sangiorgi, D., editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 98:1–98:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Cygan et al., 2015] Cygan, M., Fomin, F. V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., and Saurabh, S. (2015). *Parameterized Algorithms*. Springer.

[Doyen et al., 2014] Doyen, L., Juhl, L., Larsen, K. G., Markey, N., and Shirmohammadi, M. (2014). Synchronizing words for weighted and timed automata. In Raman, V. and Suresh, S. P., editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, volume 29 of *LIPIcs*, pages 121–132. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

[Eppstein, 1990] Eppstein, D. (1990). Reset sequences for monotonic automata. *SIAM Journal on Computing*, 19(3):500–510.

[Fernau et al., 2019] Fernau, H., Gusev, V. V., Hoffmann, S., Holzer, M., Volkov, M. V., and Wolf, P. (2019). Computational complexity of synchronization under regular constraints. In Rossmanith, P., Heggernes, P., and Katoen, J., editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPIcs*, pages 63:1–63:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Fernau et al., 2015] Fernau, H., Heggernes, P., and Villanger, Y. (2015). A multi-parameter analysis of hard problems on deterministic finite automata. *Journal of Computer and System Sciences*, 81(4):747–765.

[Imreh and Steinby, 1999] Imreh, B. and Steinby, M. (1999). Directable nondeterministic automata. *Acta Cybernetica*, 14(1):105–115.

[Manna and Pnueli, 1990] Manna, Z. and Pnueli, A. (1990). A hierarchy of temporal properties. In Dwork, C., editor, *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing, Quebec City, Quebec, Canada, August 22-24, 1990*, pages 377–410. ACM.

[Martyugin, 2014] Martyugin, P. (2014). Computational complexity of certain problems related to carefully synchronizing words for partial automata and directing words for nondeterministic automata. *Theory of Computing Systems*, 54(2):293–304.

[Martyugin, 2012] Martyugin, P. V. (2012). Synchronization of automata with one undefined or ambiguous transition. In Moreira, N. and Reis, R., editors, *Implementation and Application of Automata - 17th International Conference, CIAA 2012, Porto, Portugal, July 17-20, 2012. Proceedings*, volume 7381 of *Lecture Notes in Computer Science*, pages 278–288. Springer.

[Natarajan, 1986] Natarajan, B. K. (1986). An algorithmic approach to the automated design of parts orienters. In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 132–142. IEEE Computer Society.

[Rystsov, 1980] Rystsov, I. K. (1980). On minimizing the length of synchronizing words for finite automata. In *Theory of Designing of Computing Systems*, pages 75–82. Institute of Cybernetics of Ukrainian Acad. Sci. (in Russian).

[Rystsov, 1983] Rystsov, I. K. (1983). Polynomial complete problems in automata theory. *Information Processing Letters*, 16(3):147–151.

[Ryzhikov, 2019] Ryzhikov, A. (2019). Synchronization problems in automata without non-trivial cycles. *Theoretical Computer Science*, 787:77–88.

[Ryzhikov and Shemyakov, 2018] Ryzhikov, A. and Shemyakov, A. (2018). Subset synchronization in monotonic automata. *Fundamenta Informaticae*, 162(2-3):205–221.

[Sandberg, 2004] Sandberg, S. (2004). Homing and synchronizing sequences. In Broy, M., Jonsson, B., Katoen, J., Leucker, M., and Pretschner, A., editors, *Model-Based Testing of Reactive Systems, Advanced Lectures [The volume is the outcome of a research seminar that was held in Schloss Dagstuhl in January 2004]*, volume 3472 of *Lecture Notes in Computer Science*, pages 5–33. Springer.

[Savitch, 1970] Savitch, W. J. (1970). Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192.

[Sipser, 1997] Sipser, M. (1997). *Introduction to the Theory of Computation*. PWS Publishing Company.

[Trakhtman, 2007] Trakhtman, A. (2007). The Černý conjecture for aperiodic automata. *Discrete Mathematics and Theoretical Computer Science*, 9(2).

[Truthe and Volkov, 2019] Truthe, B. and Volkov, M. V. (2019). Journal of Automata, Languages and Combinatorics – Essays on the Černý Conjecture. `https://www.jalc.de/issues/2019/issue_24_2-4/content.html`. Accessed: 10/1/2020.

[Türker and Yenigün, 2015] Türker, U. C. and Yenigün, H. (2015). Complexities of some problems related to synchronizing, non-synchronizing and monotonic automata. *International Journal of Foundations of Computer Science*, 26(1):99–122.

[Volkov, 2008] Volkov, M. V. (2008). Synchronizing automata and the Černý conjecture. In Martín-Vide, C., Otto, F., and Fernau, H., editors, *Language and Automata Theory and Applications, Second International Conference, LATA 2008, Tarragona, Spain, March 13-19, 2008. Revised Papers*, volume 5196 of *Lecture Notes in Computer Science*, pages 11–27. Springer.

[Vorel and Roman, 2015] Vorel, V. and Roman, A. (2015). Parameterized complexity of synchronization and road coloring. *Discrete Mathematics and Theoretical Computer Science*, 17(1):283–306.

Chapter 9

Synchronizing Deterministic Push-Down Automata Can Be Really Hard

Henning Fernau, Petra Wolf, and Tomoyuki Yamakami.

An extended abstract appeared in the proceedings of MFCS 2020:
Leibniz International Proceedings in Informatics (LIPIcs) 170 (2020) pp. 33:1 – 33:15.
DOI 10.4230/LIPIcs.MFCS.2020.33.

Synchronizing Deterministic Push-Down Automata Can Be Really Hard

Henning Fernau[1], Petra Wolf*[1], and Tomoyuki Yamakami[2]

[1]Universität Trier, Germany
[2]University of Fukui, Japan

Abstract

The question if a deterministic finite automaton admits a software reset in the form of a so-called synchronizing word can be answered in polynomial time. In this paper, we extend this algorithmic question to deterministic automata beyond finite automata. We prove that the question of synchronizability becomes undecidable even when looking at deterministic one-counter automata. This is also true for another classical mild extension of regularity, namely, that of deterministic one-turn push-down automata. However, when we combine both restrictions, we arrive at scenarios with a PSPACE-complete (and hence decidable) synchronizability problem. Likewise, we arrive at a decidable synchronizability problem for (partially) blind deterministic counter automata.

There are several interpretations of what *synchronizability* should mean for deterministic push-down automata. This is depending on the role of the stack: should it be empty on synchronization, should it be always the same or is it arbitrary? For the automata classes studied in this paper, the complexity or decidability status of the synchronizability problem is mostly independent of this technicality, but we also discuss one class of automata where this makes a difference.

1 Introduction

The classical *synchronization problem* asks, for a given deterministic finite automaton (DFA), if there exists a *synchronizing word*, i.e., an input that brings all states of the

*The author was supported by DFG-funded project FE560/9-1

automaton to a single state. While this problem is solvable in polynomial time [Černý, 1964, Volkov, 2008, Sandberg, 2005], many variants, such as synchronizing only a subset of states [Sandberg, 2005], or synchronizing into a specified subset of states [Rystsov, 1983], or synchronizing a partial automaton without taking any undefined transition on some path [Martyugin, 2014], are PSPACE-complete. Restricting the length of a potential synchronizing word of some DFA by an integer parameter in the input also yields a harder problem, namely the NP-complete short synchronizing word problem [Rystsov, 1980, Eppstein, 1990]. The field of synchronizing automata has been intensively studied over the last years also in attempt to verify the famous Černý conjecture, claiming that every synchronizable DFA admits a synchronizing word of quadratic length in the number of states [Černý, 1964, Černý, 2019, Starke, 1966, Starke, 2019]. We are far from solving this combinatorial question, as the currently best upper bound on this length is only cubic [Shitov, 2019, Szykuła, 2018]. For more on synchronization of DFAs and the Černý conjecture, we refer to the surveys [Volkov, 2008, Béal and Perrin, 2016, Truthe and Volkov, 2019].

The idea of bringing an automaton to a well-defined state by reading a word, starting from any state, can be seen as implementing a software reset. This is why a synchronizing word is also sometimes called a *reset word*. But this very idea is obviously not restricted to finite automata. In this work, we want to move away from deterministic finite automata to more general deterministic push-down automata. What should a synchronizing word mean in this context? Mikami and Yamakami first studied in [Mikami and Yamakami, 2020] three different models, depending on requirements of the stack contents when a word w drives the automaton into a synchronizing state, irrespectively of the state where processing w started: we could require at the end (a) that the stack is always empty; or (b) that the stack contents is always the same (but not necessarily empty); or (c) that the stack contents is completely irrelevant upon entering the synchronizing state. They demonstrated in [Mikami and Yamakami, 2020] some upper and lower bounds on the maximum length of the shortest synchronizing word for those three models of push-down automata, dependent on the stack height. Here, we study these three models from a complexity-theoretic perspective. However, as we show in our first main result, synchronizability becomes undecidable when asking about synchronizability in any of the stack models. Clearly, by restricting the length of a potential synchronizing word of some DPDA by an integer parameter (given in unary), we can observe that the corresponding synchronization problems all become NP-complete, as the hardness is trivially inherited from what is known about DFA synchronizability. Therefore, we will not consider such length-bounded problem variants any further in this paper. Yet, it remains interesting to observe that with DFAs, introducing a length bound on the synchronizing word means an increase of complexity, while for DPDAs, this introduction

means dropping from undecidability close to feasibility. Beside general DPDAs, we will study these stack model variants of synchronization for sub-classes of DPDAs such as deterministic counter automata (DCA), deterministic (partially) blind automata and finite-turn variants of DPDAs and DCAs. In [Fernau and Wolf, 2020], further restricted sub-classes of DPDAs, such as visibly and very visibly deterministic push-down and counter automata are considered. There, all considered cases are in EXPTIME and even membership in P and PSPACE is observed, contrasting our undecidability results here.

Closest to the problems studied in our paper comes the work of Chistikov et al., see [Chistikov et al., 2019], reviewed in the following, as their automaton model could be viewed as a special case of push-down automata, related to *input-driven pushdown automata* [Mehlhorn, 1980] which later became popular as *visibly push-down automata* [Alur and Madhusudan, 2004]. In [Chistikov et al., 2019], the synchronization problem for so-called *nested word automata* (NWA) has been studied, where the concept of synchronization has been generalized to bringing all states to one single state such that for all runs the stack is empty (or in its start configuration) after reading the synchronizing word. In this setting the synchronization problem is solvable in polynomial time, whereas the short synchronizing word problem is PSPACE-complete (here, the length bound is given in binary) and the question of synchronizing from or into a subset is EXPTIME-complete.

The DFA synchronization problem has been generalized in the literature to other automata models including infinite-state systems with infinite branching such as weighted and timed automata [Doyen et al., 2014, Shirmohammadi, 2014] or register automata [Babari et al., 2016]. For instance, register automata are infinite state systems where a state consists of a control state and register contents. A synchronizing word for a register automaton brings all (infinitely many) states to the same state (and same register content). The synchronization problem for deterministic register automata (DRA) is PSPACE-complete and NLOGSPACE-complete for DRAs with only one register.

Finally, we want to mention that the term *synchronization of push-down automata* has already some occurrences in the literature, i.e., in [Caucal, 2006, Arenas et al., 2011], but here the term *synchronization* refers to some relation of the input symbols to the stack behavior [Caucal, 2006] or to reading different words in parallel [Arenas et al., 2011] and is not to be confused with our notion of synchronizing states.

We are presenting an overview on our results at the end of the next section, where we introduce our notions more formally.

2 Definitions

We refer to the empty word as ϵ. For a finite alphabet Σ we denote by Σ^* the set of all words over Σ and by $\Sigma^+ = \Sigma\Sigma^*$ the set of all non-empty words. For $i \in \mathbb{N}$ we set $[i] = \{1, 2, \ldots, i\}$. For $w \in \Sigma^*$ we denote by $|w|$ the length of w, by $w[i]$ for $i \in [|w|]$ the i'th symbol of w, and by $w[i..j]$ for $i, j \in [|w|]$ the factor $w[i]w[i+1]\ldots w[j]$ of w. We call $w[1..i]$ a prefix and $w[i..|w|]$ a suffix of w. The reversal of w is denoted by w^R, i.e., for $|w| = n$, $w^R = w[n]w[n-1]\ldots w[1]$.

We call $A = (Q, \Sigma, \delta, q_0, F)$ a *deterministic finite automaton* (DFA for short) if Q is a finite set of states, Σ is a finite input alphabet, δ is a transition function $Q \times \Sigma \to Q$, q_0 is the initial state and $F \subseteq Q$ is the set of final states. The transition function δ is generalized to words by $\delta(q, w) = \delta(\delta(q, w[1]), w[2..|w|])$ for $w \in \Sigma^*$. A word $w \in \Sigma^*$ is accepted by A if $\delta(q_0, w) \in F$ and the language accepted by A is defined by $\mathcal{L}(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \in F\}$. We extend δ to sets of states $Q' \subseteq Q$ or to sets of letters $\Sigma' \subseteq \Sigma$, letting $\delta(Q', \Sigma') = \{\delta(q', \sigma') \mid (q', \sigma') \in Q' \times \Sigma'\}$. Similarly, we may write $\delta(Q', \Sigma') = p$ to define $\delta(q', \sigma') = p$ for each $(q', \sigma') \in Q' \times \Sigma'$. The synchronization problem for DFAs (called DFA-Sync) asks for a given DFA A whether there exists a synchronizing word for A. A word w is called a *synchronizing word* for a DFA A if it brings all states of the automaton to one single state, i.e., $|\delta(Q, w)| = 1$.

We call $M = (Q, \Sigma, \Gamma, \delta, q_0, \bot, F)$ a *deterministic push-down automaton* (DPDA for short) if Q is a finite set of states; the finite sets Σ and Γ are the input and stack alphabet, respectively; δ is a transition function $Q \times \Sigma \times \Gamma \to Q \times \Gamma^*$; q_0 is the initial state; $\bot \in \Gamma$ is the stack bottom symbol which is only allowed as the first (lowest) symbol in the stack, i.e., if $\delta(q, a, \gamma) = (q', \gamma')$ and γ' contains $\bot$, then $\bot$ only occurs in γ' as its prefix and moreover, $\gamma = \bot$; and F is the set of final states. We will only consider real-time push-down automata and forbid ϵ-transitions, as can be seen in the definition. Notice that the bottom symbol can be removed, but then the computation gets stuck.

Following [Chistikov et al., 2019], a *configuration* of M is a tuple $(q, \upsilon) \in Q \times \Gamma^*$. For a letter $\sigma \in \Sigma$ and a stack content υ with $|\upsilon| = n$ we write $(q, \upsilon) \xrightarrow{\sigma} (q', \upsilon[1..(n-1)]\gamma)$ if $\delta(q, \sigma, \upsilon[n]) = (q', \gamma)$. This means that the top of the stack υ is the right end of υ. We also denote by $\longrightarrow$ the reflexive transitive closure of the union of $\xrightarrow{\sigma}$ over all letters in Σ. The input words on top of $\longrightarrow$ are concatenated accordingly, so that $\longrightarrow = \bigcup_{w \in \Sigma^*} \xrightarrow{w}$. The language $\mathcal{L}(M)$ accepted by a DPDA M is $\mathcal{L}(M) = \{w \in \Sigma^* \mid (q_0, \bot) \xrightarrow{w} (q_f, \gamma), q_f \in F\}$. We call the sequence of configurations $(q, \bot) \xrightarrow{w} (q', \gamma)$ the *run* induced by w, starting in q, and ending in q'.

We will discuss three different concepts of synchronizing DPDAs. For all concepts we

demand that a synchronizing word $w \in \Sigma^*$ maps all states, starting with an empty stack, to the same synchronizing state, i.e., for all $q, q' \in Q$: $(q, \bot) \xrightarrow{w} (\overline{q}, v), (q', \bot) \xrightarrow{w} (\overline{q}, v')$. In other words, for a synchronizing word all runs started on some states in Q end up in the same state. In addition to synchronizing the states of a DPDA we will consider the following two conditions for the stack content: (1) $v = v' = \bot$, (2) $v = v'$. We will call (1) the *empty stack model* and (2) the *same stack model*. In the third case, we do not put any restrictions on the stack content and call this the *arbitrary stack model*.

As we are only interested in synchronizing a DPDA, we can neglect the start and final states.

As mentioned above, we will show that synchronizability of DPDAs is undecidable, which is in stark contrast to the situation with DFAs, where this problem is solvable in polynomial time. Hence, it is interesting to discuss deterministic variants of classical sub-classes of context-free languages. In this paper, we focus on one-counter languages and on linear languages and related classes. A *deterministic (one) counter automaton* (DCA) is a DPDA where $|\Gamma \backslash \{\bot\}| = 1$. Note that our DCAs can perform zero-tests by checking if the bottom-of-stack symbol is on top of the stack. As we will see that also with this restriction, synchronizability is still undecidable, we further restrict them to the *partially blind* setting [Greibach, 1978]. This means in our formalization that a transition $\delta(q, \sigma, x) = (q', \gamma)$ either satisfies $\gamma = \epsilon$ for both $x = 1$ and $x = \bot$, or x is a prefix of γ, i.e., $\gamma = x\gamma'$, and then both $\delta(q, \sigma, 1) = (q', 1\gamma')$ (for $\Gamma = \{1, \bot\}$) and $\delta(q, \sigma, \bot) = (q', \bot\gamma')$. The situation is even more delicate with one-turn or, more general, finite-turn DPDAs, whose further discussion and formal definition we defer to the specific section below.

We are now ready to define a family of synchronization problems, the complexity of which will be our subject of study in the following chapters.

Definition 1 (Sync-DPDA-Empty)**.**
Given: DPDA $M = (Q, \Sigma, \Gamma, \delta, \bot)$.
Question: Does there exist a word $w \in \Sigma^*$ that synchronizes M in the empty stack model?

For the same stack model, we refer to the synchronization problem above as Sync-DPDA-Same and as Sync-DPDA-Arb in the arbitrary stack model. Variants of these problems are defined by replacing the DPDA in the definition above by a DCA, a deterministic partially blind counter automaton (DPBCA), or by adding turn restrictions, in particular, whether the automaton is allowed to make zero or one turns of its stack movement.

class of automata	empty stack model	same stack model	arbitrary stack model
DPDA	undecidable	undecidable	undecidable
1-Turn-Sync-DPDA	undecidable	undecidable	undecidable
0-Turn-Sync-DPDA	PSPACE-complete	undecidable	PSPACE-complete
DCA	undecidable	undecidable	undecidable
1-Turn-Sync-DCA	PSPACE-complete	PSPACE-complete	PSPACE-complete
0-Turn-Sync-DCA	PSPACE-complete	PSPACE-complete	PSPACE-complete
DPBCA	decidable	decidable	decidable

Table 1: Complexity status of the synchronization problem for different classes of deterministic real-time push-down automata in different stack synchronization modes as well as finite-turn variants of the respective synchronization problem.

Outlook and summary of the paper

We summarize our results in Table 1. In short, while already seemingly innocuous extensions of finite automata (with counters or with 1-turn push-downs) result in an undecidable synchronizability problem, some extensions do offer some algorithmic synchronizability checks, although nothing efficient. At the end, we show how to apply some of our techniques to synchronizability questions concerning sequential transducers.

As an auxiliary result for proving undecidability of finding 1-turn synchronizing words for real-time deterministic push-down automata, we also prove undecidability of the inclusion and intersection non-emptiness problems for these automata, which could be an interesting result on its own.

3 General DCAs and DPDAs: When Synchronizability is Really Hard

The inclusion problem for deterministic real-time one counter automata that can perform zero-tests is undecidable [Böhm and Göller, 2011, Minsky, 1961]. This result is used to prove undecidability of synchronization in any general setting as the main result of this section. However, there are special cases of DPDAs and DCAs that have a decidable inclusion problem (see [Higuchi et al., 1995] as an example) so that this argument does not apply to these sub-classes. We will have a closer look at some of these sub-classes in the following sections.

Theorem 1. *The problems* Sync-DCA-Empty, Sync-DCA-Same, *and* Sync-DCA-Arb *are undecidable.*

Proof. We give a reduction from the undecidable intersection non-emptiness problem for real-time DCAs [Böhm and Göller, 2011]. Let $M_1 = (Q_1, \Sigma, \{1, \bot\}, \delta_1, q_0^1, \bot, F_1)$ and $M_2 = (Q_2, \Sigma, \{1, \bot\}, \delta_2, q_0^2, \bot, F_2)$ be two DCAs over the same input alphabet with disjoint state sets. We construct a DCA $M_S = (Q_1 \cup Q_2 \cup \{q_f^1, q_f^2, q_s\}, \Sigma \cup \{a, b\}, \{1, \bot\}, \delta, \bot)$, where we neglect start and final states, which is synchronizable in the empty stack model if and only if the DCAs M_1 and M_2 accept a common word. The same construction also works for the same stack and arbitrary stack models. We assume $\{q_f^1, q_f^2, q_s\} \cap (Q_1 \cup Q_2) = \varnothing$ and $\{a, b\} \cap \Sigma = \varnothing$. For the states in Q_1 and Q_2, the transition function δ agrees with δ_1 and δ_2 for all letters in Σ. In the following, let $i \in \{1, 2\}$. For $q \in Q_i$, we set $\delta(q, a, \bot) = (q_0^i, \bot)$ and $\delta(q, a, 1) = (q_f^i, 1)$. Further, for $q \in Q_i \backslash F_i$ we set $\delta(q, b, 1) = (q_f^i, 1)$ and $\delta(q, b, \bot) = (q_f^i, \bot 1)$. For $q \in F_i$, we set $\delta(q, b, 1) = (q_s, 1)$ and $\delta(q, b, \bot) = (q_s, \bot)$. For q_f^i, we set $\delta(q_f^i, a, \bot) = (q_0^i, \bot)$ with all other transitions we stay in q_f^i and increase the counter. Hence, the state q_f^i can only be left with an empty counter and this is only the case if no letter other than a has been read before. For the state q_s, we set $\delta(q_s, \Sigma \cup \{a, b\}, \bot) = (q_s, \bot)$, and $\delta(q_s, \Sigma \cup \{a, b\}, 1) = (q_s, \epsilon)$.

First, assume there is a word $w \in \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$. Then, the word awb synchronizes all states of the DCA M_S into the state q_s. Let l_1, l_2 be stack contents such that $(q_0^1, \bot) \xrightarrow{awb} (q_s, l_1)$ and $(q_0^2, \bot) \xrightarrow{awb} (q_s, l_2)$. Let $l = \max(|l_1|, |l_2|)$. Then $awbb^l$ synchronizes M_S in the empty stack model.

For the other direction assume there exists a word $w \in (\Sigma \cup \{a, b\})^*$ that synchronizes M_S in the empty stack model. The states q_f^1 and q_f^2 forces $w[1] = a$ since otherwise these states cannot be left. Since the state q_s has no outgoing transition, it must be our synchronizing state. In order to reach it, w must contain at least one letter b. Let $m \in [|w|]$ be an index such that $w[m] = b$ and for $j < b$, $w[j] \neq b$. With a letter b we move from all final states to the state q_s and from all non-final states of M_1 and M_2 we go to a state q_f^i and increase the counter. As we cannot leave the states q_f^i if we reach them once with a non-empty counter, reading a b from a non-final state causes the automaton to reach a configuration from which we no longer can synchronize the automaton. Hence, we know that after reading $w[1..m]$ all active states are in the set $F_1 \cup F_2 \cup \{q_s\}$. Let $\ell \in [|w|]$ be an index with $\ell < m$ with $w[\ell] = a$ such that for $\ell < i < m$, $w[i] \neq a$. Then $w[\ell + 1 \,..\, m - 1]$ is a word which is accepted by both DCAs M_1 and M_2.

It is easy to see that clearing the stacks in state q_s is not crucial and hence the reduction also works for the same stack and arbitrary stack models. □

Corollary 1. *The problems* Sync-DPDA-Empty*,* Sync-DPDA-Same*, and* Sync-DPDA-Arb *are undecidable.* □

How can we overcome the problem that, even for deterministic one-counter languages,

the synchronizability problem is undecidable? One of the famous further restrictions are (partially) blind counters, to which we turn our attention next.

4 Partially Blind Deterministic Counter Automata

The blind and partially blind variations of counter automata have been introduced by Greibach in [Greibach, 1978]. She already noticed that the emptiness problem for such automata (even with multiple counters) is decidable should the reachability problem for vector addition systems, also known as Petri nets, be decidable, which has been proven some years later [Mayr, 1981, Kosaraju, 1982]; its non-elementary complexity has only been recently fully understood [Czerwinski et al., 2021], for a survey see [Schmitz, 2016]. Although we will stick to the models introduced so far in the following statements and proofs, we want to make explicit that our decidability results also hold for deterministic multi-counter automata. But as we focus on discussing families of automata describing languages between regular and context-free, we refrain from giving further details here.

Because partially blind counters can simulate blind counters, our results hold for blind counters as well, but we make them explicit only for the partially blind case. One formal reason is that we want to preserve our stack model, while it becomes awkward to formalize blind counters in this stack model (see next section).

Recall that a partially blind counter automaton will get blocked when its counter gets below zero. The blindness refers to the fact that such a machine can never explicitly test its counter for zero. This translates into our formalization by requiring that a transition $\delta(q,\sigma,x) = (q',\gamma)$ either satisfies $\gamma = \epsilon$ for both $x = 1$ and $x = \bot$, or x is a prefix of γ, i.e., $\gamma = x\gamma'$, and then both $\delta(q,\sigma,1) = (q',1\gamma')$ (for $\Gamma = \{1,\bot\}$) and $\delta(q,\sigma,\bot) = (q',\bot\gamma')$. In other words, the processing of a letter will somehow perform the same action, irrespectively of the stack contents, whenever this is possible; however, the machine will stop if it is trying to pop the bottom-of-stack symbol. As a specialty, such automata accept when having arrived in a final state together with having zero in its counter. We call a deterministic partially blind (one-)counter automaton a DPBCA.

Theorem 2. *The problems* Sync-DPBCA-Empty*,* Sync-DPBCA-Same*, and* Sync-DPBCA-Arb *are decidable.*

We are not specifying the complexity here, but only mention that we are using, in the end, the reachability problem for Petri nets, which is known to be decidable, but only with a non-elementary complexity; see [Mayr, 1981, Kosaraju, 1982, Czerwinski et al., 2021]. However, we leave it as an open question if the synchronization complexity of DPBCAs

is non-elementary. When looking into this question more in details, the number of states of the counter automaton could be a useful parameter to be discussed, as it influences the number of counters of the partially blind multi-counter automaton that we construct in our proof in order to show the claimed decidability result.

Proof. Let $M = (Q, \Sigma, \{1, \bot\}, \delta, q_0, \bot, F)$ be some DPBCA. Let us first describe the case Sync-DPBCA-Empty. Here, we can first produce the multi-counter $|Q|$-fold product automaton $M^{|Q|}$ from M that starts, assuming $Q = \{q_0, \ldots, q_{|Q|-1}\}$, in the state $(q_0, \ldots, q_{|Q|-1})$. Notice that $M^{|Q|}$ has $|Q|^{|Q|}$ many states and operates $|Q|$ many counters. We could take as the set of final states $F^{|Q|} = \{(q, \ldots, q) \mid q \in Q\}$. This mimicks state synchronization of M: any word that synchronizes all states of M drives $M^{|Q|}$ into F. As mentioned above, partially blind multi-counter automata accept with final states and empty stacks, so that M is synchronizable in the empty stack model if and only if $M^{|Q|}$ accepts any word.

For the arbitrary stack model, we have to count down (removing 1 from any of the stacks sequentially) until the bottom-of-stack symbol appears on all stacks on top (at the same time), leading to the variant $M^{|Q|}_{\text{Arb}}$. These are moves without reading the input (or reading arbitrary symbols at the end, this way only prolonging a possibly synchronizing word), but this does not matter, as the emptiness problem is decidable for partially blind nondeterministic multi-counter automata. It should be clear that $M^{|Q|}_{\text{Arb}}$ accepts any word if and only if M is synchronizable. For the case Sync-DPBCA-Same, the counting down at the end should be performed in parallel for all counters instead. □

5 Discussing Blind Counter Automata

Recall that a counter automaton is blind if it has a counter that stores some integer (that can also be negative), but it can test emptiness only at the very end of its computation, being hence part of the definition of the accepted language. If we want to model this behavior similar to our previous definitions, we would therefore consider an element from $Q \times \mathbb{Z}$ as a blind counter automaton configuration. The transition function δ would map $Q \times \Sigma$ to $Q \times \{-1, 0, 1\}$, meaning that if $\delta(q, \sigma) = (q', z)$, then $(q, c) \xrightarrow{\sigma} (q', c + z)$. The remainder of this formalization is standard and hence omitted. Alternatively, we could try to formalize this behavior also as a pushdown automaton. But then, the automaton must store in an additional bit if the counter stores a positive or a negative number. This also means that depending on this bit of information, incrementing the counter could either mean pushing 1 onto the stack or popping 1 from the stack, and (in a reversed fashion), this is also true when decrementing the counter. Clearly, this additional bit

of information cannot be tested by the machine itself upon processing, as this would destroy the blindness condition; yet, this bit influences the formal processing, so that any formalization of blind counter automata (and their synchronizability) as special push-down automata looks somewhat strange.

How can we use this model for synchronization purposes? First, observe that as the counter contents never influences the run of a blind automaton, synchronization with the arbitrary stack model would mean just synchronization of the underlying DFA.

This looks different for the empty stack model. Here, we (have to) use a product automaton construction as in the proof of Theorem 2 to reduce this synchronization problem to the emptiness problem of blind (real-time) deterministic multi-counter machines. By adding the possibility to decrement all counters upon reading a special symbol at the end, the emptiness problem of blind (real-time) deterministic multi-counter machines can also be used to show the decidability of the synchronization problem for deterministic blind counter automata. We summarize our observations as follows.

Proposition 1. *The problems* Sync-DBCA-Empty*,* Sync-DBCA-Same*, and* Sync-DBCA-Arb *are decidable, the latter even in polynomial time.*

As a final remark, let us mention that already Greibach [Greibach, 1978] observed the relations between (quasi-realtime) blind counter automata and reversal-bounded counter automata (reversals is just another name for turns), which also links to our discussion of finite-turn automata. However, when making a blind counter automaton reversal-bounded, the number of counters is increased, so that we cannot compare these models for a fixed number of counters, which is of course our focus when studying language classes (and automata models) between regular and context-free.

6 Finite-Turn DPDAs

Finite-turn PDAs are introduced in [Ginsburg and Spanier, 1966]. From the formal language side, it is known that one-turn PDAs characterize the rather familiar family of linear context-free languages, usually defined via grammars. In our setting, the automata view is more interesting. We adopt the definition in [Valiant, 1973]. For a DPDA M an *upstroke* of M is a sequence of configurations induced by an input word w such that no transition decreases the stack-height. Accordingly a *downstroke* of M is a sequence of configurations in which no transition increases the stack-height. A stroke is either an upstroke or downstroke. Note that exchanging the top symbol of the stack is allowed in both an up- and downstroke. A DPDA M is an n-turn DPDA if for all $w \in \mathcal{L}(M)$ the

sequence of configurations induced by w can be split into at most $n+1$ strokes. Especially, for 1-turn DPDAs each sequence of configurations induced by an accepting word consists of one upstroke followed by at most one downstroke. There are two subtleties when translating this concept to synchronization: (a) there is no initial state so that there is no way to associate a stroke counter to a state, and (b) there is no language of accepted words that restricts the set of words on which the number of strokes should be limited. We therefore generalize the concept of finite-turn DPDAs to finite-turn synchronization for DPDAs in the following way, which offers quite an interesting complexity landscape.

Definition 2. n-Turn-Sync-DPDA-Empty
Given: DPDA $M = (Q, \Sigma, \Gamma, \delta, q_0, \bot, F)$.
Question: Is there a synchronizing word $w \in \Sigma^*$ in the empty stack model such that for all states $q \in Q$, the sequence of configurations $(q, \bot) \xrightarrow{w} (\overline{q}, \bot)$ consists of at most $n+1$ strokes?

We call such a synchronizing word w an *n-turn synchronizing word* for M. We define n-Turn-Sync-DPDA-Same and n-Turn-Sync-DPDA-Arb accordingly for the same stack and arbitrary stack models. Further, we extend the problem definition to real-time DCAs.

Motivated by the proof of Theorem 1, we are first reviewing the status of the inclusion problem for 1-turn DPDAs in the literature.

Remark 1. The inclusion problem for 1-turn DPDAs with ϵ-transitions is undecidable [Friedman, 1976, Valiant, 1973]. The intersection non-emptiness problem for real-time 1-turn non-deterministic push-down automata is also undecidable [Kim, 2011]. The decidability of the inclusion and intersection non-emptiness problems for *real-time* 1-turn *deterministic* push-down automata have not been settled in the literature; we will do so below by proving undecidability for both problems.

We will present a reduction from the undecidable Post Correspondence Problem (PCP for short) [Post, 1946] to the intersection non-emptiness for real-time 1-turn DPDAs which also implies undecidability of the inclusion problem for this class since it is closed under complement.

Definition 3 (PCP)**.**
Given: Two lists of input words over $\{0,1\}$: $A = (a_1, a_2, \ldots, a_n)$, and $B = (b_1, b_2, \ldots, b_n)$.
Question: Is there a sequence of indices $i_1, i_2, \ldots, i_k$ with $i_j \in [n]$ for $1 \leq j \leq k$ such that $a_{i_1} a_{i_2} \ldots a_{i_k} = b_{i_1} b_{i_2} \ldots b_{i_k}$?

Observe that already Post stated this problem over binary alphabets. Much later, Matyasevich and Sénizergues [Matiyasevich and Sénizergues, 2005] showed that indeed lists of length seven are sufficient for undecidability. This was recently lowered to lists of length five by Neary [Neary, 2015].

Theorem 3. *Let M_1 and M_2 be two real-time 1-turn DPDAs. Then the following problems are undecidable: Is $\mathcal{L}(M_1) \cap \mathcal{L}(M_2) = \varnothing$? Is $\mathcal{L}(M_1) \subseteq \mathcal{L}(M_2)$?*

Proof. Let $A = (a_1, a_2, \ldots, a_n)$, and $B = (b_1, b_2, \ldots, b_n)$ be an instance of PCP. We construct from A a real-time 1-turn DPDA $M_A = (Q, \{0, 1, \#, \$\} \cup [\overline{n}], \{0, 1, \bot\}, \delta, q_0, \bot, \{q_f\})$ where $[\overline{n}] = \{\overline{1}, \overline{2}, \ldots, \overline{n}\}$ are marked numbers from 1 to n. The set Q contains the start state q_0, the states $\overline{q_0}$, q_{check} and q_{fail}, and the single final state q_f. The rest of Q is a partition into state sets $Q_1, Q_2, \ldots, Q_n$ such that the (deterministic partial) sub-automaton induced by Q_i reads the string $a_i \# b_i$ and thereby pushes each symbol of a_i on the stack, whereas symbols of the string b_i leave the stack content unchanged. The sub-automaton induced by Q_i is embedded into M_A, and thereby completed, by taking the state after reading $a_i \# b_i$ to $\overline{q_0}$ with the symbol $\#$. We go from q_0 and $\overline{q_0}$ to the initial state of the sub-automaton induced by Q_i by the letter $\overline{i}$. With the letter \$ the state $\overline{q_0}$ maps to q_{check}. Here, if the input symbol equals the symbol on top of the stack, we pop the stack and stay in q_{check} until we reach the bottom symbol $\bot$, in which case an additional letter \$ brings us to the final state q_f. If the input symbol does not equal the symbol on top of the state we go to q_{fail} which is a trap state for all letters. Every other not yet defined transition on Q maps to the state q_{fail}. If not stated otherwise, every transition leaves the stack content unchanged.

For the list B we construct a real-time 1-turn DPDA M_B in a similar way except that here we push the strings b_i on the stack and symbols of strings a_i leave the stack unchanged.

The languages accepted by M_A and M_B are the following ones:

$$\mathcal{L}(M_A) = \{\, \overline{i_1} a_{i_1} \# b_{i_1} \# \overline{i_2} a_{i_2} \# b_{i_2} \# \cdots \overline{i_m} a_{i_m} \# b_{i_m} \# \$ a_{i_m}^R \cdots a_{i_1}^R \$ \mid m \geq 1, i_j \in [n], j \leq m \}$$

$$\mathcal{L}(M_B) = \{\, \overline{i_1} a_{i_1} \# b_{i_1} \# \overline{i_2} a_{i_2} \# b_{i_2} \# \cdots \overline{i_m} a_{i_m} \# b_{i_m} \# \$ b_{i_m}^R \cdots b_{i_1}^R \$ \mid m \geq 1, i_j \in [n], j \leq m \}$$

Obviously, the given PCP has a solution if and only if $\mathcal{L}(M_A) \cap \mathcal{L}(M_B) \neq \varnothing$.

By complementing the set of final states, from M_A one would arrive at a real-time 1-turn DPDA M'_A such that $\mathcal{L}(M'_A)$ is the complement of $\mathcal{L}(M_A)$, so that the given PCP has no solution if and only if $\mathcal{L}(M'_A) \supseteq \mathcal{L}(M_B)$. □

We will now adapt the aforementioned construction to show that the synchronization problem for real-time one-turn DPDAs is undecidable in all three synchronization models.

Theorem 4. *The problems* 1-Turn-Sync-DPDA-Empty*,* 1-Turn-Sync-DPDA-Same*, and* 1-Turn-Sync-DPDA-Arb *are undecidable.*

Proof. Let M_A and M_B be the real-time 1-turn DPDAs from the proof in Theorem 3. We take these machines as inputs for the construction in the proof of Theorem 1 to obtain the DPDA M. Therefore, observe that the construction also works if the input machines are not only DCAs but general DPDAs. Further, observe that for M_A and M_B, if for both machines the only active state is the final state, then the stack of all runs is empty and hence the stack does not need to be altered in the synchronizing state and all transitions here can act as the identity and leave the stack unchanged.

If w is a word in $\mathcal{L}(M_A) \cap \mathcal{L}(M_B)$, then awb synchronizes M in the empty, same, and arbitrary stack models; furthermore, awb is a 1-turn synchronizing word for M. Conversely, if w is a 1-turn synchronizing word for M, then w must be of the form avb or $auavb$ where $v \in \mathcal{L}(M_A) \cap \mathcal{L}(M_B)$ and u is a word that does not change the stack because, otherwise, either M could not be synchronized, or the 1-turn condition is violated. To be more precise, a must be the first letter of w since, otherwise, we get stuck in q_f^1 and q_f^2. The letter a resets the machines M_A and M_B to their initial state and can only be read when the stack is empty since, otherwise, the machine gets stuck. In order to reach final states, both machines M_A and M_B must increase and decrease the stack by reading some word v, but as soon as we increased the stack once, we are not allowed to reset the machine anymore due to the 1-turn condition. Hence, the letter a can only be read while the stack has not been changed yet. Note that for all three stack models, the construction enforces that any 1-turn synchronizing word brings M into a configuration where the stack is empty. □

When considering automata as language accepting devices, there is no good use of 0-turn PDAs, as they cannot exploit their stack. This becomes different if synchronization requires to end in the same configuration, which means that in particular the stack contents are identical.

Theorem 5. *The problem* 0-Turn-Sync-DPDA-Same *is undecidable.*

Proof sketch. The proof is by a straightforward adaption of the previously presented constructions by getting rid of the check phase; instead, check with the same stack condition that the two words of the PCP coincide. As we never pop the stack, a 0-turn DPDA is sufficient in the construction. □

Proof. Let $A = (a_1, a_2, \ldots, a_n)$, $B = (b_1, b_2, \ldots, b_n)$ be an instance of PCP. We construct from A a real-time 0-turn partial DPDA $M_A = (Q^A, \{0, 1, \#\} \cup [\overline{n}], \{0, 1, \bot\}, \delta^A, \bot)$ where

$[\overline{n}] = \{\overline{1}, \overline{2}, \ldots, \overline{n}\}$ are marked numbers from 1 to n. The set Q^A contains the states q_0^A and $\underline{q_0^A}$. The rest of Q^A is a partition into states $Q_1^A, Q_2^A, \ldots, Q_n^A$ such that the (deterministic partial) sub-automaton induced by Q_i^A reads the string $a_i \# b_i$ and thereby pushes each symbol of a_i on the stack whereas symbols of b_i leave the stack unchanged. The sub-automaton induced by Q_i^A is embedded into M_A, by taking the state after reading $a_i \# b_i$ to $\underline{q_0^A}$ with the symbol $\#$. We go from q_0^A and $\underline{q_0^A}$ to the initial state of the sub-automaton induced by Q_i^A by the letter $\overline{i}$. If not stated otherwise, every transition leaves the stack content unchanged.

For the list B, we construct a real-time 0-turn partial DPDA $M_B = (Q^B, \{0,1,\#\} \cup [\overline{n}], \{0,1,\bot\}, \delta^B, \bot)$ in a similar way, except that here we push the strings b_i on the stack and symbols of strings a_i leave the stack unchanged.

We combine the two machines M_A and M_B into a real-time 0-turn complete DPDA $M = (Q^A \cup Q^B \cup \{q_{\text{fail}}^A, q_{\text{fail}}^B, q_{\text{sync}}\}, \{0,1,\#,a\} \cup [\overline{n}], \{0,1,\bot\}, \delta, \bot)$ where all unions are assumed to be disjoint. The transition function δ agrees with δ^A and δ^B on all states in $Q^A \cup Q^B$ and letters in $\{0,1,\#\} \cup [\overline{n}]$. For the letter a we set for all states $q^j \in Q^j$ with $j \in \{A,B\}$, $\delta(q^j, a, \bot) = (q_0^j, \bot)$ and for $\gamma \neq \bot$, $\delta(q^j, a, \gamma) = (q_{\text{fail}}^j, \gamma)$. For $\underline{q_0^j}$ and $\gamma \in \{0,1\}$ we set $\delta(\underline{q_0^j}, \#, \gamma) = (q_{\text{sync}}, \gamma)$. For q_{fail}^j we set $\delta(q_{\text{fail}}^j, a, \bot) = (q_0^j, \bot)$ and for all other transitions we stay in q_{fail}^j and add a 1 on top of the stack. For q_{sync} we set $\delta(q_{\text{sync}}, a, \bot) = (q_0^A, \bot)$ and for all other transitions, we stay in q_{sync} and leave the stack unchanged. With all other not yet defined transition in Q^j, we go to q_{fail}^j and add a 1 on top of the stack.

Following previous arguments, it is clear that any synchronizing word w for M in the same stack model must start with the letter a and contain at at least one sequence $\#\#$. Further, let $j \in w[|w|]$ with $w[j] = \#$ be the position of the first occurrence of a sequence $\#\#$ and let $i \in w[|w|]$ with $w[i] = a$ be the position of the last occurrence of a before position j. Then, the sub-word $w[i+1\,..\,j-1]$ describes a solution for the PCP instance. Conversely, a solution of the PCP can be embedded in a synchronizing word for M. □

The picture changes again for other 0-turn stack models, but remains intractable.

Theorem 6. *The problems* 0-Turn-Sync-DCA-Empty*,* 0-Turn-Sync-DCA-Same *and* 0-Turn-Sync-DCA-Arb *are PSPACE-hard.*

Proof. We give a reduction from the problem DFA-Sync-Into-Subset, also called Global Inclusion Problem for Non-Initial Automata in [Rystsov, 1983], proven to be PSPACE-complete in Theorem 2.1 in [Rystsov, 1983]:

Definition 4 (DFA-Sync-Into-Subset)**.**

Given: DFA $A = (Q, \Sigma, \delta)$, subset $S \subseteq Q$.
Question: Is there a word $w \in \Sigma^*$ such that $\delta(Q, w) \subseteq S$?

Let $A = (Q, \Sigma, \delta)$ be a DFA with $S \subseteq Q$. We construct from A a DCA $M = (Q \cup \{q_{\text{stall}}, q_{\text{sync}}\}, \Sigma \cup \{a\}, \{N, \bot\}, \delta', \bot)$ where all unions are disjoint. For $q \in Q$, $\sigma \in \Sigma$ and $\gamma \in \{N, \bot\}$, set $\delta'(q, \sigma, \gamma) = (\delta(q, \sigma), \gamma)$. For the letter a, we set for states $q \in S$, $\delta'(q, a, \bot) = (q_{\text{stall}}, \bot)$ and for states $q \in Q \backslash S$, we set $\delta'(q, a, \bot) = (q_{\text{stall}}, \bot N)$. For q_{stall} we set $\delta'(q_{\text{stall}}, a, \bot) = (q_{\text{sync}}, \bot)$ and $\delta'(q_{\text{stall}}, a, N) = (q_{\text{stall}}, N)$. All transitions not yet defined act as the identity and leave the stack unchanged.

First, assume there exists a word $w \in \Sigma^*$ that synchronizes Q into S in the DFA A. Then clearly waa synchronizes M in the arbitrary stack model. Now, assume there exists a word $w \in (\Sigma \cup \{a\})^*$ that synchronizes M in the arbitrary stack model. Then, w must contain at least two occurrences of a to bring all states into the sink state q_{sync}. In order to reach q_{sync} the states in Q need to pass through the state q_{stall} but by doing so, it is noted on the stack if an active state transitions from $Q \backslash S$ into q_{stall} and only the active states coming from S are allowed to pass on to q_{sync}. In q_{stall} the stack content cannot be changed and hence the prefix of w up to the first occurrence of the letter a must have already synchronized Q into S in the DFA A. Note that M can only be synchronized by a word that leaves all stacks empty. Hence, the result follows for all three stack models. □

Corollary 2. *The problems* 0-Turn-Sync-DPDA-Empty *and* 0-Turn-Sync-DPDA-Arb *are* PSPACE*-hard.*

Proof. The claim follows from Theorem 6 by inclusion of automata classes. □

Theorem 7. 0-Turn-Sync-DPDA-Empty, 0-Turn-Sync-DPDA-Arb ∈ *PSPACE.*

Proof. Let $M = (Q, \Sigma, \Gamma, \delta, \bot)$ be a DPDA. We first focus on 0-Turn-Sync-DPDA-Empty. As we need to synchronize with an empty stack, the 0-turn condition forbids us to write anything on the stack at all. Hence, we only keep in M transitions of the form $\delta(q, \sigma, \bot) = (q', \bot)$ for $q, q' \in Q$, $\sigma \in \Sigma$ and delete all other transitions from M. We call the obtained automaton M'. We may observe that M' is basically a partial DFA and the problem of synchronizing M with a 0-turn synchronizing word in the empty stack model has been reduced to synchronizing M' without using an undefined transition. The latter problem is called the Careful Synchronization problem and is solvable in PSPACE [Martyugin, 2014].

For the problem 0-Turn-Sync-DPDA-Arb it is sufficient for each run to only keep the symbol on top of the stack in memory, as we are not allowed to decrease the height of the

stack at any time, but transitions might still depend on the symbol on top of the stack. As we have no restriction on the stack content for synchronization, we can safely forget all other symbols on the stack. Hence, we can construct from M a partial DFA M' with state set $Q \times \Gamma$ by re-interpreting transitions $\delta(q, \sigma, \gamma) = (q', \gamma')$ for $q, q' \in Q, \sigma \in \Sigma, \gamma \in \Gamma, \gamma' \in \Gamma^*$ as $\delta((q, \gamma), \sigma) = (q', \gamma'[|\gamma'|])$ and deleting transitions of the form $\delta(q, \sigma, \gamma) = (q', \epsilon)$. The problem of synchronizing M with a 0-turn synchronizing word in the arbitrary stack model has now been reduced to finding a word that brings all states $(q', \bot)$ of M' with $q' \in Q$ into one of the sets $S_q = \{(q, \gamma) \mid \gamma \in \Gamma\}$ for $q \in Q$ without using an undefined transition. We can solve this problem using polynomial space by guessing a path in the $|Q|$-fold product automaton $M'^{|Q|}$. The automaton $M'^{|Q|}$ with state set $(Q \times \Gamma)^{|Q|}$, consisting of $|Q|$-tuples of states in $Q \times \Gamma$, and alphabet Σ, is defined based on M' by simulating the transition function δ of M' on every single state in a $|Q|$-tuple state of $M'^{|Q|}$ independently, yielding the transition function $\delta^{|Q|}$ of $M'^{|Q|}$. Here, $\delta^{|Q|}$ is only defined on a $|Q|$-tuple if and only if δ is defined on every state of this tuple. Clearly, the size of $M^{|Q|}$ is $\mathcal{O}((|Q||\Gamma|)^{|Q|})$ and we can guess a path from the state $((q_1, \bot), (q_2, \bot), \ldots, (q_n, \bot))$ for $Q = \{q_1, q_2, \ldots, q_n\}$ to one of the sets S_q for $q \in Q$ using space $\mathcal{O}(\log(|Q||\Gamma|)|Q|)$. We conclude the proof with Savitch's famous theorem proving PSPACE = NPSPACE [Savitch, 1970]. □

Theorem 8. 1-Turn-Sync-DCA-Empty, 1-Turn-Sync-DCA-Same, *and* 1-Turn-Sync-DCA-Arb *are in PSPACE.*

Notice that PSPACE-hardness is inherited from corresponding results for visibly counter automata, as obtained in [Fernau and Wolf, 2020].

Proof. Let $M = (Q, \Sigma, \Gamma, \delta, \bot)$ be a DCA. As we are looking into 1-turn behavior, any computation, in which we are interested, would split into two phases: in the first upstroke phase, the counter is incremented or stays constant, while in the second downstroke phase, the counter is decremented or stays constant. In particular, because the counter is 1-turn, after the first counter increment, any zero-test will always return 'false', while in the downstroke phase, when zero-tests return 'true', then all future computations cannot decrement the counter any further, so that at the end, the counter will also contain zero. We are formalizing this intuition to create a machine that has an awareness about its phase stored in its states and that is behaving very similar. In the rest of the proof, a *spread-out variant* of a word $a_1 a_2 \cdots a_n$ of length n, with symbols a_i from Σ, is any word in $a_1 \Sigma a_2 \Sigma \cdots a_n \Sigma$ of length $2n$.

We will now construct from M and $q \in Q$ a deterministic 1-turn counter automaton M_q that accepts precisely all spread-out variants of words that M would accept when starting in state q and finishing its 1-turn computation (in any state) with the empty stack, but

that keeps track of a basic property of managing the counter in so-called stages. M_q has the state set $Q\times\{1,2,3,4\}\times\{0,1\}$, $(q,1,0)$ as its initial state, and as its set of final states, take $Q\times\{1,4\}\times\{0\}$. The transitions of δ_q can be defined with the following semantics in mind (details are given below): (a) the last bit always alternates, (b) the spread-out is used to enable a *deterministic* work and to make sure that the simulated machine has counter content zero if the simulating automaton M_q is in one of the states from $Q\times\{1,4\}\times\{0\}$, (c) M_q changes from $Q\times\{1\}\times\{0,1\}$ to $Q\times\{2\}\times\{0,1\}$ if the counter is no longer zero, so that the simulated machine has "properly" entered the upstroke phase, (d) M_q changes from $Q\times\{2\}\times\{0,1\}$ to $Q\times\{3\}\times\{0,1\}$ if the simulated machine made its first pop, i.e., it "properly" entered the downstroke phase, (e) M_q changes from $Q\times\{3\}\times\{0,1\}$ to $Q\times\{4\}\times\{0,1\}$ if the counter has become zero again.

Now, we build the $|Q|$-fold product automaton $M_{\text{Empty}}^{|Q|}$ from all automata M_q with the start state $((q_1,1,0),(q_2,1,0),\ldots,(q_{|Q|},1,0))$, assuming $Q=\{q_1,\ldots,q_{|Q|}\}$. This means that $M_{\text{Empty}}^{|Q|}$ has $(8|Q|)^{|Q|}$ many states and $|Q|$ many counters, each of which makes at most one turn. Now observe that a word w synchronizes M with empty stacks, say, in state p if and only if any spread-out variant of w drives $M_{\text{Empty}}^{|Q|}$ into a state $((p,i_1,0),(p,i_2,0),\ldots,(p,i_{|Q|},0))$ for some $i_j\in\{1,4\}$ for all $1\le j\le|Q|$. Now, define $\{((p,i_1,0),(p,i_2,0),\ldots,(p,i_{|Q|},0)) \mid p\in Q, i_j\in\{1,4\} \text{ for } 1\le j\le|Q|\}$ as the final states of $M_{\text{Empty}}^{|Q|}$. We see that M is synchronizable with empty stacks if and only if $M_{\text{Empty}}^{|Q|}$ accepts any word. As Gurari and Ibarra have shown in [Gurari and Ibarra, 1981, Lemma 2], $M_{\text{Empty}}^{|Q|}$ accepts any word if and only if it accepts any word up to length $(|Q|(8|Q|)^{|Q|}|\Sigma|)^{O(|Q|)}\le(|Q|(8|Q||\Sigma|)^{O(|Q|)^2})$,[1] within the same time bounds. Now, testing all these words for membership basically needs two counters that are able to capture numbers of size $(|Q|(8|Q||\Sigma|)^{O(|Q|)^2})$, which means we need polynomial space in $|Q|$ and $|\Sigma|$ to check if M is synchronizable with empty stacks.

By considering $M_{\text{Empty}}^{|Q|}$ with the final states $((p,i_1,0),(p,i_2,0),\ldots,(p,i_{|Q|},0))$ for arbitrary $p\in Q$ and $i_j\in\{1,2,3,4\}$, this way defining an automaton $M_{\text{Arb}}^{|Q|}$, we can check if M is synchronizable in the arbitrary stack model also in polynomial space with the same argument. From $M_{\text{Empty}}^{|Q|}$, we can also construct a nondeterministic 1-turn $|Q|$-counter machine $M_{\text{Same}}^{|Q|}$ by adding a nondeterministic move from $((p,i_1,0),(p,i_2,0),\ldots,(p,i_{|Q|},0))$ for arbitrary $p\in Q$ and $i_j\in\{1,2,3,4\}$ upon reading some arbitrary but fixed $\sigma_{\text{sync}}\in\Sigma$ to a special state q_{dec} in which state we loop upon reading $\sigma_{\text{sync}}\in\Sigma$, decrementing all counters at the same time; finally, there is the possibility to move to q_f (that is the only finaly state now) upon reading $\sigma_{\text{sync}}\in\Sigma$ if all counters are empty. Notice that also this automaton $M_{\text{Same}}^{|Q|}$ has $(\mathcal{O}(|Q|))^{|Q|}|\Sigma|$ many transitions. Thus, using [Gurari and Ibarra, 1981, Lemma 2] we can again conclude that synchronizability in the same stack model

[1] In the cited lemma, the number of steps of the checking machine is upper-bounded by $(ms)^{O(m)}$, where m is the number of 1-turn counters and s is the number of transitions of the machine.

can be checked in polynomial space for M.

Finally, we explain in detail how to build the transition function of M_q from the specification of M.

- Whenever there is a transition $\delta(p, a, \bot) = (p', \bot)$ in M, M_q can transition from p to p' upon reading a, leaving its counter untouched. Formally, this means that $\delta_q((p, 1, 0), a, \bot) = ((p', 1, 1), \bot)$ in M_q. Moreover, for any state $r \in Q$ and any letter $b \in \Sigma$, $\delta_q((r, 1, 1), b, \bot) = ((r, 1, 0), \bot)$ in M_q. Notice that in this stage one, M_q knows that the counter is zero. In particular, any word read in this stage could be accepted.

- If there is a transition $\delta(p, a, \bot) = (p', \bot 1^\ell)$ with $\ell > 0$ in M, M_q moves into stage two. Hence, there is a transition $\delta_q((p, 1, 0), a, \bot) = ((p', 2, 1), \bot 1^\ell)$ in M_q. This certifies that we have truly entered a proper upstroke phase.

- We can check that we are in the proper upstroke phase by defining the transitions as $\delta_q((r, 2, 1), b, 1) = ((r, 2, 0), 1)$ in M_q for any $r \in Q$ and any $b \in \Sigma$.

- Transitions of the form $\delta(p, a, 1) = (p', 1^\ell)$ with $\ell > 0$ in M let M_q stay in stage two. Hence, there is a transition $\delta_q((p, 1, 0), a, 1) = ((p', 2, 1), 1^\ell)$ in M_q.

- However, transitions of the form $\delta(p, a, 1) = (p', \varepsilon)$ in M let M_q move in stage three. We have now truly entered a proper downstroke phase. Hence, there is a transition $\delta_q((p, 2, 0), a, 1) = ((p', 3, 1), \varepsilon)$ in M_q.

- We can check that we are in the proper downstroke phase by setting the transition function as $\delta_q((r, 3, 1), b, 1) = ((r, 3, 0), 1)$ in M_q for any $r \in Q$ and any $b \in \Sigma$.

- We stay within stage three by introducing $\delta_q((p, 3, 0), a, 1) = ((p', 3, 1), 1^\ell)$ in M_q for rules $\delta(p, a, 1) = (p', 1^\ell)$ with $\ell \in \{0, 1\}$.

- We will move into stage four once we have arrived at an empty stack again. This is realized by having transitions $\delta_q((r, 3, 1), b, \bot) = ((r, 4, 0), \bot)$ in M_q for any $r \in Q$ and any $b \in \Sigma$.

- Whenever there is a transition $\delta(p, a, \bot) = (p', \bot)$ in M, then M_q can transition from p to p' upon reading a, leaving its counter untouched, i.e., $\delta_q((p, 4, 0), a, \bot) = ((p', 4, 1), \bot)$ in M_q. Moreover, for any state $r \in Q$ and any letter $b \in \Sigma$, we have $\delta_q((r, 4, 1), b, \bot) = ((r, 4, 0), \bot)$ in M_q. Notice that in this stage four, M_q knows again that the counter is zero. In particular, any word read in this stage could be accepted.

This finally concludes the proof. □

7 Sequential Transducers

We will now introduce a new concept of synchronization of sequential transducers.[2] We call $T = (Q, \Sigma, \Gamma, q_0, \delta, F)$ a *sequential transducer* (ST for short) if Q is a finite set of states, Σ is a finite input alphabet, Γ is a finite output alphabet, q_0 is the start state, $\delta: Q \times \Sigma \to Q \times \Gamma^*$ is a total transition function, and F is a set of final states. We generalize δ from input letters to words by concatenating the produced outputs, i.e., for $q, q', q'' \in Q$, $\sigma_1, \sigma_2 \in \Sigma$, $\gamma_1, \gamma_2 \in \Gamma^*$ and transitions $\delta(q, \sigma_1) = (q', \gamma_1), \delta(q', \sigma_2) = (q'', \gamma_2)$ we define $\delta(q, \sigma_1\sigma_2) = (q'', \gamma_1\gamma_2)$. We say that a word w *trace-synchronizes* a sequential transducer T if for all states $p, q \in Q$ it holds that $\delta(p, w) = \delta(q, w)$. Intuitively, w brings all states of T to the same state and produces the same output on all states. Again, we might neglect start and final states.

Definition 5 (Trace-Sync-Transducer)**.**
Given: Sequential transducer $T = (Q, \Sigma, \Gamma, \delta)$.
Question: Does there exist a word $w \in \Sigma^*$ that trace-synchronizes T?

Theorem 9. *The problem* Trace-Sync-Transducer *is undecidable.*

Proof. We adapt the construction of M in Theorem 5 to obtain a sequential transducer T in the following way: Each time, we push a letter to the stack in Q_i^A or Q_i^B, instead we now output this letter. Whenever we leave the stack unchanged, we now simply do not produce any output. For the letter a, we output the special letter r on all states in order to indicate that the machine has been reset. For the state q_{fail}^A we output the special letter A and for q_{fail}^B the special letter B, for all input symbols expect a.

We make the following observations for any potential trace-synchronizing word w for T: (1) w needs to start with a. (2) w needs to contain the sub-word ##. (3) The sub-word between the first occurrence of ## and the respective last occurrence of a before ## describes a solution of the PCP instance. (4) If the PCP instance has a solution, one can construct a trace-synchronizing word for T from that solution. □

8 Prospects

It would be interesting to look into the synchronization problem for further automata models. In view of the undecidability results that we obtained in this paper, a special

[2]The definitions in the literature are not very clear for finite automata with outputs. We follow here the name used by Berstel in [Berstel, 1979]; Ginsburg [Ginsburg, 1966] called Berstel's sequential transducers *generalized machines*, but used the term *sequential transducer* for the nondeterministic counterpart.

focus should be to look into deterministic automata classes with a known decidable inclusion problem, as otherwise it should be possible to adapt our undecidability proofs for synchronizability to these automata models. To make this research direction more clear: (a) There are quite efficient algorithms for the inclusion problem for so-called *very simple deterministic pushdown automata*, see [Wakatsuki and Tomita, 1993]; (b) a proper super-class of these languages are so-called *NTS languages* that also have a deterministic automaton characterization[3] but their inclusion problem is undecidable, see [Sénizergues, 1985, Boasson and Sénizergues, 1985]. The overall aim of this research would be to find the borderline between decidable and undecidable synchronizability and, moreover, within the decidable part, to determine the complexity of this problem. A step in this research direction has been made in direction of (sub-classes of) visibly deterministic pushdown automata in [Fernau and Wolf, 2020]. Interestingly enough, that research line also revealed some cases where synchronizability can be decided in polynomial time, quite in contrast to the situation found in the present study.

Another approach is to look into variants of synchronization problems for DPDAs, such as restricting the length of a potential synchronizing word. It follows from the NP-hardness of this problem for DFAs [Rystsov, 1980, Eppstein, 1990] and the polynomial-time solvability of the membership problem for DPDAs that for unary encoded length bounds this problem is NP-complete for DPDAs as well, and contained in EXPTIME for binary encoded length bounds. The precise complexity of this problem for binary encoded length bounds will be a topic of future research.

References

[Alur and Madhusudan, 2004] Alur, R. and Madhusudan, P. (2004). Visibly pushdown languages. In Babai, L., editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 202–211. ACM.

[Arenas et al., 2011] Arenas, M., Barceló, P., and Libkin, L. (2011). Regular languages of nested words: Fixed points, automata, and synchronization. *Theory of Computing Systems*, 49(3):639–670.

[Babari et al., 2016] Babari, P., Quaas, K., and Shirmohammadi, M. (2016). Synchronizing data words for register automata. In Faliszewski, P., Muscholl, A., and Niedermeier, R., editors, *41st International Symposium on Mathematical Foundations of*

[3]This is rather implicit in the literature, which is one of the reasons why we do not present more details here; one would have to first define the automaton model properly.

Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland, volume 58 of *LIPIcs*, pages 15:1–15:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Béal and Perrin, 2016] Béal, M.-P. and Perrin, D. (2016). *Synchronised Automata*, page 213–240. Encyclopedia of Mathematics and its Applications. Cambridge University Press.

[Berstel, 1979] Berstel, J. (1979). *Transductions and Context-Free Languages*, volume 38 of *Teubner Studienbücher: Informatik*. Teubner.

[Boasson and Sénizergues, 1985] Boasson, L. and Sénizergues, G. (1985). NTS languages are deterministic and congruential. *Journal of Computer and System Sciences*, 31(3):332–342.

[Böhm and Göller, 2011] Böhm, S. and Göller, S. (2011). Language equivalence of deterministic real-time one-counter automata is nl-complete. In Murlak, F. and Sankowski, P., editors, *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*, volume 6907 of *Lecture Notes in Computer Science*, pages 194–205. Springer.

[Caucal, 2006] Caucal, D. (2006). Synchronization of pushdown automata. In Ibarra, O. H. and Dang, Z., editors, *Developments in Language Theory, 10th International Conference, DLT 2006, Santa Barbara, CA, USA, June 26-29, 2006, Proceedings*, volume 4036 of *Lecture Notes in Computer Science*, pages 120–132. Springer.

[Černý, 1964] Černý, J. (1964). Poznámka k homogénnym eksperimentom s konečnými automatami. *Matematicko-fyzikalny Časopis Slovensk*, 14(3):208–215.

[Černý, 2019] Černý, J. (2019). A note on homogeneous experiments with finite automata. *Journal of Automata, Languages and Combinatorics*, 24(2-4):123–132.

[Chistikov et al., 2019] Chistikov, D., Martyugin, P., and Shirmohammadi, M. (2019). Synchronizing automata over nested words. *Journal of Automata, Languages and Combinatorics*, 24(2-4):219–251.

[Czerwinski et al., 2021] Czerwinski, W., Lasota, S., Lazic, R., Leroux, J., and Mazowiecki, F. (2021). The reachability problem for Petri nets is not elementary. *Journal of the ACM*, 68(1):7:1–7:28.

[Doyen et al., 2014] Doyen, L., Juhl, L., Larsen, K. G., Markey, N., and Shirmohammadi, M. (2014). Synchronizing words for weighted and timed automata. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, pages 121–132.

[Eppstein, 1990] Eppstein, D. (1990). Reset sequences for monotonic automata. *SIAM Journal on Computing*, 19(3):500–510.

[Fernau and Wolf, 2020] Fernau, H. and Wolf, P. (2020). Synchronization of deterministic visibly push-down automata. In Saxena, N. and Simon, S., editors, *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020, December 14-18, 2020, BITS Pilani, K K Birla Goa Campus, Goa, India (Virtual Conference)*, volume 182 of *LIPIcs*, pages 45:1–45:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

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

[Ginsburg, 1966] Ginsburg, S. (1966). *The mathematical Theory of Context-Free Languages*. McGraw-Hill.

[Ginsburg and Spanier, 1966] Ginsburg, S. and Spanier, E. H. (1966). Finite-turn pushdown automata. *SIAM Journal on Control*, 4(3):429–453.

[Greibach, 1978] Greibach, S. A. (1978). Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7:311–324.

[Gurari and Ibarra, 1981] Gurari, E. M. and Ibarra, O. H. (1981). The complexity of decision problems for finite-turn multicounter machines. *Journal of Computer and System Sciences*, 22(2):220–229.

[Higuchi et al., 1995] Higuchi, K., Wakatsuki, M., and Tomita, E. (1995). A polynomial-time algorithm for checking the inclusion for real-time deterministic restricted one-counter automata which accept by final state. *IEICE Transactions on Information and Systems*, 78-D(8):939–950.

[Kim, 2011] Kim, C. (2011). Quasi-rocking real-time pushdown automata. *Theoretical Computer Science*, 412(48):6720–6735.

[Kosaraju, 1982] Kosaraju, S. R. (1982). Decidability of reachability in vector addition systems (preliminary version). In Lewis, H. R., Simons, B. B., Burkhard, W. A., and Landweber, L. H., editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 267–281. ACM.

[Martyugin, 2014] Martyugin, P. (2014). Computational complexity of certain problems related to carefully synchronizing words for partial automata and directing words for nondeterministic automata. *Theory of Computing Systems*, 54(2):293–304.

[Matiyasevich and Sénizergues, 2005] Matiyasevich, Y. V. and Sénizergues, G. (2005). Decision problems for Semi-Thue systems with a few rules. *Theoretical Computer Science*, 330(1):145–169.

[Mayr, 1981] Mayr, E. W. (1981). An algorithm for the general Petri net reachability problem. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing, May 11-13, 1981, Milwaukee, Wisconsin, USA*, pages 238–246. ACM.

[Mehlhorn, 1980] Mehlhorn, K. (1980). Pebbling moutain ranges and its application of DCFL-recognition. In de Bakker, J. W. and van Leeuwen, J., editors, *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 422–435. Springer.

[Mikami and Yamakami, 2020] Mikami, E. and Yamakami, T. (2020). Synchronizing pushdown automata and reset words. An article appeared in Japanese as Technical Report of The Institute of Electonics, Information and Communication Engineers, COMP2019-54(2020-03), pp. 57–63.

[Minsky, 1961] Minsky, M. L. (1961). Recursive unsolvability of Post's problem of "Tag" and other topics in theory of Turing machines. *Annals of Mathematics*, pages 437–455.

[Neary, 2015] Neary, T. (2015). Undecidability in binary tag systems and the Post correspondence problem for five pairs of words. In Mayr, E. W. and Ollinger, N., editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 649–661. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Post, 1946] Post, E. L. (1946). A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52(4):264–268.

[Rystsov, 1980] Rystsov, I. K. (1980). On minimizing the length of synchronizing words for finite automata. In *Theory of Designing of Computing Systems*, pages 75–82. Institute of Cybernetics of Ukrainian Acad. Sci. (in Russian).

[Rystsov, 1983] Rystsov, I. K. (1983). Polynomial complete problems in automata theory. *Information Processing Letters*, 16(3):147–151.

[Sandberg, 2005] Sandberg, S. (2005). Homing and synchronizing sequences. In Broy, M., Jonsson, B., Katoen, J., Leucker, M., and Pretschner, A., editors, *Model-Based Testing of Reactive Systems, Advanced Lectures*, volume 3472 of *LNCS*, pages 5–33. Springer.

[Savitch, 1970] Savitch, W. J. (1970). Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192.

[Schmitz, 2016] Schmitz, S. (2016). The complexity of reachability in vector addition systems. *ACM SIGLOG News*, 3(1):4–21.

[Sénizergues, 1985] Sénizergues, G. (1985). The equivalence and inclusion problems for NTS languages. *Journal of Computer and System Sciences*, 31(3):303–331.

[Shirmohammadi, 2014] Shirmohammadi, M. (2014). *Qualitative Analysis of Synchronizing Probabilistic Systems. (Analyse qualitative des systèmes probabilistes synchronisants)*. PhD thesis, École normale supérieure de Cachan, France.

[Shitov, 2019] Shitov, Y. (2019). An improvement to a recent upper bound for synchronizing words of finite automata. *Journal of Automata, Languages and Combinatorics*, 24(2-4):367–373.

[Starke, 1966] Starke, P. H. (1966). Eine Bemerkung über homogene Experimente. *Elektronische Informationsverarbeitung und Kybernetik (Journal of Information Processing and Cybernetics)*, 2(4):257–259.

[Starke, 2019] Starke, P. H. (2019). A remark about homogeneous experiments. *Journal of Automata, Languages and Combinatorics*, 24(2-4):133–137.

[Szykuła, 2018] Szykuła, M. (2018). Improving the upper bound on the length of the shortest reset word. In Niedermeier, R. and Vallée, B., editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPIcs*, pages 56:1–56:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Truthe and Volkov, 2019] Truthe, B. and Volkov, M. V. (2019). Journal of Automata, Languages and Combinatorics – Essays on the Černý Conjecture. `https://www.jalc.de/issues/2019/issue_24_2-4/content.html`. Accessed: 10/1/2020.

[Valiant, 1973] Valiant, L. G. (1973). *Decision Procedures for Families of Deterministic Pushdown Automata*. PhD thesis, University of Warwick, Coventry, UK.

[Volkov, 2008] Volkov, M. V. (2008). Synchronizing automata and the Černý conjecture. In Martín-Vide, C., Otto, F., and Fernau, H., editors, *Language and Automata Theory and Applications, Second International Conference, LATA*, volume 5196 of *LNCS*, pages 11–27. Springer.

[Wakatsuki and Tomita, 1993] Wakatsuki, M. and Tomita, E. (1993). A fast algorithm for checking the inclusion for very simple deterministic pushdown automata. *IEICE Transactions on Information and Systems*, 76-D(10):1224–1233.

Chapter 10

Synchronization of Deterministic Visibly Push-Down Automata

Henning Fernau and Petra Wolf.

An extended abstract appeared in the proceedings of FSTTCS 2020:
Leibniz International Proceedings in Informatics (LIPIcs) 182 (2020) pp. 45:1 – 45:15.
DOI: 10.4230/LIPIcs.FSTTCS.2020.45.

Synchronization of Deterministic Visibly Push-Down Automata

Henning Fernau and Petra Wolf*

Universität Trier, Germany

Abstract

We generalize the concept of synchronizing words for finite automata, which map all states of the automata to the same state, to deterministic visibly push-down automata. Here, a synchronizing word w does not only map all states to the same state but also fulfills some conditions on the stack content of each run after reading w. We consider three types of these stack constraints: after reading w, the stack (1) is empty in each run, (2) contains the same sequence of stack symbols in each run, or (3) contains an arbitrary sequence which is independent of the other runs. We show that in contrast to general deterministic push-down automata, it is decidable for deterministic visibly push-down automata whether there exists a synchronizing word with each of these stack constraints, more precisely, the problems are in EXPTIME. Under the constraint (1), the problem is even in P. For the sub-classes of deterministic very visibly push-down automata, the problem is in P for all three types of constraints. We further study variants of the synchronization problem where the number of turns in the stack height behavior caused by a synchronizing word is restricted, as well as the problem of synchronizing a variant of a sequential transducer, which shows some visibly behavior, by a word that synchronizes the states and produces the same output on all runs.

1 Introduction

The classical *synchronization problem* asks, given a deterministic finite automaton (DFA), whether there exists a *synchronizing word* that brings all states of the automaton to a single state. While this problem is solvable in polynomial time, see [Černý, 1964, Sandberg,

*The author was supported by DFG-funded project FE560/9-1

2005, Volkov, 2008], many variants, such as synchronizing only a subset of states [Sandberg, 2005], or synchronizing a partial automaton without taking an undefined transition (called carefully synchronizing) [Martyugin, 2014], are PSPACE-complete. Restricting the length of a potential synchronizing word by a parameter in the input also yields a harder problem, namely the NP-complete short synchronizing word problem [Rystsov, 1980, Eppstein, 1990]. The field of synchronizing automata has been intensively studied over the last years, among others in attempt to verify the famous Černý conjecture claiming that every synchronizable DFA admits a synchronizing word of quadratic length in the number of states [Černý, 1964, Černý, 2019, Starke, 1966, Starke, 2019]. The currently best upper bound on this length is cubic, and only very little progress has been made, basically improving on the multiplicative constant factor in front of the cubic term, see [Shitov, 2019, Szykuła, 2018]. More information on synchronization of DFA and the Černý conjecture can be found in [Volkov, 2008, Béal and Perrin, 2016, Truthe and Volkov, 2019]. In this work, we want to move away from deterministic finite automata to more general deterministic visibly push-down automata.[1]

The synchronization problem has been generalized in the literature to other automata models including infinite-state systems with infinite branching such as weighted and timed automata [Doyen et al., 2014, Shirmohammadi, 2014] or register automata [Babari et al., 2016]. Here, register automata are infinite state systems where a state consists of a control state and register contents.

Another automaton model, where the state set is enhanced with a possibly infinite memory structure, namely a stack, is the class of *nested word automata* (NWAs were introduced in [Alur and Madhusudan, 2009]), where an input word is enhanced with a matching relation determining at which pair of positions in a word a symbol is pushed to and popped from the stack. The class of languages accepted by NWAs is identical to the class of *visibly push-down languages* (VPL) accepted by *visibly push-down automata* (VPDA) and form a proper sub-class of the deterministic context-free languages. VPDAs have first been studied by Mehlhorn [Mehlhorn, 1980] under the name *input-driven pushdown automata* and became quite popular more recently due to the work by Alur and Madhusudan [Alur and Madhusudan, 2004], showing that VPLs share several nice properties with regular languages. For more on VPLs we refer to the survey [Okhotin and Salomaa, 2014]. In [Chistikov et al., 2019], the synchronization problem for NWAs was studied. There, the concept of synchronization was generalized to bringing all states to one single state such that for all runs the stack is empty (or in its start configuration) after

[1] The term *synchronization of push-down automata* already occurs in the literature, i.e., in [Caucal, 2006, Arenas et al., 2011], but there the term *synchronization* refers to some relation of the input symbols to the stack behavior [Caucal, 2006] or to reading different words in parallel [Arenas et al., 2011] and is not to be confused with our notion of synchronizing states.

reading the synchronizing word. In this setting, the synchronization problem is solvable in polynomial time (again indicating similarities of VPLs with regular languages), while the short synchronizing word problem (with length bound given in binary) is PSPACE-complete; the question of synchronizing from or into a subset is EXPTIME-complete. Also, matching exponential upper bounds on the length of a synchronizing word are given.

Our attempt in this work is to study the synchronization problem for real-time (no ϵ-transitions) deterministic visibly push-down automata (DVPDA) and several sub-classes thereof, like real-time deterministic very visibly push-down automata (DVVPDA for short; this model was introduced in [Ludwig, 2019]), real-time deterministic visibly counter automata (DVCA for short; this model appeared a.o. in [Bárány et al., 2006, Srba, 2009, Bollig, 2016, Hahn et al., 2015, Krebs et al., 2015a, Krebs et al., 2015b]) and finite turn variants thereof. We want to point out that, despite the equivalence of the accepted language class, the automata models of nested word automata and visibly push-down automata still differ and the results from [Chistikov et al., 2019] do not immediately transfer to VPDAs, as for NWAs an input word is equipped with a matching relation, which VPDAs lack of. In general, the complexity of the synchronization problem can differ for different automata models accepting the same language class. For instance, in contrast to the polynomial time solvable synchronization problem for DFAs, the generalized synchronization problem for finite automata with one ambiguous transition is PSPACE-complete, as well as the problem of carefully synchronizing a DFA with one undefined transition [Martyugin, 2012]. We will not only consider the synchronization model introduced in [Chistikov et al., 2019], where reading a synchronizing word results in an empty stack on all runs; but we will also consider a synchronization model where not only the last state of every run must be the same but also the stack content needs to be identical, as well as a model where only the states needs to be synchronized and the stack content might be arbitrary. These three models of synchronization have been introduced in [Mikami and Yamakami, 2020], where length bounds on a synchronizing word for general DPDAs have been studied dependent on the stack height. The complexity of these three concepts of synchronization for general DPDAs are considered in [Fernau et al., 2020], where it is shown that synchronizability is undecidable for general DPDAs and deterministic counter automata (DCA). It becomes decidable for deterministic partially blind counter automata and is PSPACE-complete for some types of finite turn DPDAs, while it is still undecidable for other types of finite turn DPDAs.

In contrast, we will show in the following that for DVPDAs and considered sub-classes hereof, the synchronization problem for all three stack models, with restricted or unrestricted number of turns, is in EXPTIME and hence decidable. For DVVPDAs and DVCAs, the synchronization problems for all three stack models (with unbounded num-

ber of turns) are even in P. Like the synchronization problem for NWAs in the empty stack model considered in [Chistikov et al., 2019], we observe that the synchronization problem for DVPDAs in the empty stack model is solvable in polynomial time, whereas synchronization of DVPDAs in the same and arbitrary stack models is at least PSPACE-hard. If the number of turns caused by a synchronizing word on each run is restricted, the synchronization problem becomes PSPACE-hard for all considered automata models for $n > 0$ and is only in P for $n = 0$ in the empty stack model. We will further introduce variants of synchronization problems distinguishing the same and arbitrary stack models by showing complementary complexities in these models. For problems considered in [Fernau et al., 2020], these two stack models have always shared their complexity status.

2 Fixing Notations

We refer to the empty word as ϵ. For a finite alphabet Σ, we denote by Σ^* the set of all words over Σ and by $\Sigma^+ = \Sigma\Sigma^*$ the set of all non-empty words. For $i \in \mathbb{N}$, we set $[i] = \{1, 2, \ldots, i\}$. For $w \in \Sigma^*$, we denote by $|w|$ the length of w, by $w[i]$ for $i \in [|w|]$ the i'th symbol of w, and by $w[i..j]$ for $i, j \in [|w|]$ the subword $w[i]w[i+1]\ldots w[j]$ of w. We call $w[1..i]$ a prefix and $w[i..|w|]$ a suffix of w. If $i < j$, then $w[j, i] = \epsilon$.

We call $A = (Q, \Sigma, \delta, q_0, F)$ a *deterministic finite automaton* (DFA for short) if Q is a finite set of states, Σ is a finite input alphabet, δ is a transition function $Q \times \Sigma \to Q$, q_0 is the initial state, and $F \subseteq Q$ is the set of final states. The transition function δ is generalized to words by $\delta(q, w) = \delta(\delta(q, w[1]), w[2..|w|])$ for $w \in \Sigma^*$. A word $w \in \Sigma^*$ is accepted by A if $\delta(q_0, w) \in F$ and the language accepted by A is defined by $\mathcal{L}(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \in F\}$. We extend δ to sets of states $Q' \subseteq Q$ or to sets of letters $\Sigma' \subseteq \Sigma$, letting $\delta(Q', \Sigma') = \{\delta(q', \sigma') \mid (q', \sigma') \in Q' \times \Sigma'\}$. Similarly, we may write $\delta(Q', \Sigma') = p$ to define $\delta(q', \sigma') = p$ for each $(q', \sigma') \in Q' \times \Sigma'$. The synchronization problem for DFAs (called DFA-Sync) asks for a given DFA A whether there exists a synchronizing word for A. A word w is called a *synchronizing word* for a DFA A if it brings all states of the automaton to one single state, i.e., $|\delta(Q, w)| = 1$.

We call $M = (Q, \Sigma, \Gamma, \delta, q_0, \bot, F)$ a *deterministic push-down automaton* (DPDA for short) if Q is a finite set of states; the finite sets Σ and Γ are the input and stack alphabet, respectively; δ is a transition function $Q \times \Sigma \times \Gamma \to Q \times \Gamma^*$; q_0 is the initial state; $\bot \in \Gamma$ is the stack bottom symbol which is only allowed as the first (lowest) symbol in the stack, i.e., if $\delta(q, a, \gamma) = (q', \gamma')$ and γ' contains $\bot$, then $\bot$ only occurs in γ' as its prefix and moreover, $\gamma = \bot$; and F is the set of final states. We will only consider *real-time*

push-down automata and forbid ϵ-transitions, as can be seen in the definition. Notice that the bottom symbol can be removed, but then the computation gets stuck.

Following [Chistikov et al., 2019], a *configuration* of M is a tuple $(q, v) \in Q \times \Gamma^*$. For a letter $\sigma \in \Sigma$ and a stack content v, with $|v| = n$, we write $(q, v) \xrightarrow{\sigma} (q', v[1..(n-1)]\gamma)$ if $\delta(q, \sigma, v[n]) = (q', \gamma)$. This means that the top of the stack v is the right end of v. We also denote by $\longrightarrow$ the reflexive transitive closure of the union of $\xrightarrow{\sigma}$ over all letters in Σ. The input words on top of $\longrightarrow$ are concatenated accordingly, so that $\longrightarrow = \bigcup_{w \in \Sigma^*} \xrightarrow{w}$. The language $\mathcal{L}(M)$ accepted by a DPDA M is $\mathcal{L}(M) = \{w \in \Sigma^* \mid (q_0, \bot) \xrightarrow{w} (q_f, \gamma), q_f \in F\}$. We call the sequence of configurations $(q, \bot) \xrightarrow{w} (q', \gamma)$ the *run* induced by w, starting in q, and ending in q'.

We will discuss three different concepts of synchronizing DPDAs. For all concepts, we require that a synchronizing word $w \in \Sigma^*$ maps all states, starting with an empty stack, to the same synchronizing state, i.e., for all $q, q' \in Q$: $(q, \bot) \xrightarrow{w} (\overline{q}, v), (q', \bot) \xrightarrow{w} (\overline{q}, v')$. In other words, for a synchronizing word all runs started on some states in Q end up in the same state. In addition to synchronizing the states of a DPDA, we will consider the following two conditions for the stack content: (1) $v = v' = \bot$, (2) $v = v'$. We will call (1) the *empty stack model* and (2) the *same stack model*. In the third case, we do not put any restrictions on the stack content and call this the *arbitrary stack model*.

As we are only interested in synchronizing a DPDA, we can neglect the start and final states.

Starting from DPDAs, we define the following sub-classes thereof:

- A *deterministic visibly push-down automaton* (DVPDA) is a DPDA where the input alphabet Σ can be partitioned into $\Sigma = \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}$ such that the change in the stack height is determined by the partition of the alphabet. To be more precise, the transition function δ is modified such that it can be partitioned accordingly into $\delta = \delta_{\text{c}} \cup \delta_{\text{i}} \cup \delta_{\text{r}}$ such that $\delta_{\text{c}}: Q \times \Sigma \to Q \times (\Gamma \backslash \{\bot\})$ puts a symbol on the stack, $\delta_{\text{i}}: Q \times \Sigma \to Q$ leaves the stack unchanged, and $\delta_{\text{r}}: Q \times \Sigma \times \Gamma \to Q$ reads and pops a symbol from the stack [Alur and Madhusudan, 2004]. If $\bot$ is the symbol on top of the stack, then $\bot$ is only read and not popped. We call letters in Σ_{call} *call* or *push* letters; letter in Σ_{int} *internal* letters; and letters in Σ_{ret} *return* or *pop* letters. The language class accepted by DVPDA is equivalent to the class of languages accepted by deterministic nested word automata (see [Chistikov et al., 2019]).
- A *deterministic very visibly push-down automaton* (DVVPA) is a DVPDA where not only the stack height but also the stack content is completely determined by the input alphabet, i.e., for a letter $\sigma \in \Sigma$ and all states $p, q \in Q$ for $\delta_{\text{c}}(p, \sigma) = (p', \gamma_p)$

and $\delta_c(q, \sigma) = (q', \gamma_q)$ it holds that $\gamma_p = \gamma_q$.

- A *deterministic visibly (one) counter automaton* (DVCA) is a DVPDA where $|\Gamma \backslash \{\bot\}| = 1$; note that every DVCA is also a DVVPDA.

We are now ready to define a family of synchronization problems, the complexity of which will be our subject of study in the following sections.

Definition 1 (Sync-DVPDA-Empty)**.**
Given: DPDA $M = (Q, \Sigma, \Gamma, \delta, \bot)$.
Question: Does there exist a word $w \in \Sigma^*$ that synchronizes M in the empty stack model?

For the same stack model, we refer to the synchronization problem above as Sync-DVPDA-Same and as Sync-DVPDA-Arb in the arbitrary stack model. Variants of these problems are defined by replacing the DVPDA in the definition above by a DVVPDA, and DVCA. If results hold for several stack models or automata models, then we summarize the problems by using set notations in the corresponding statements. For the problems Sync-DVPDA-Same and Sync-DVPDA-Arb, we introduce two further refined variants of these problems, denoted by the extension -Return and -NoReturn, where for all input DVPDA in the former variant $\Sigma_{ret} \neq \varnothing$ holds, whereas in the latter variant $\Sigma_{ret} = \varnothing$ holds. In the following, these variants reveal insights in the differences between synchronization in the same stack and arbitrary stack models, as well as connections to a concept of trace-synchronizing a sequential transducer showing some visibly behavior.

We will further consider synchronization of these automata classes in a finite-turn setting. Finite-turn push-down automata were introduced in [Ginsburg and Spanier, 1966]. We adopt the definition in [Valiant, 1973]. For a DVPDA M, an *upstroke* of M is a sequence of configurations induced by an input word w such that no transition decreases the stack-height. Accordingly, a *downstroke* of M is a sequence of configurations in which no transition increases the stack-height. A stroke is either an upstroke or downstroke. A DVPDA M is an n-turn DVPDA if for all $w \in \mathcal{L}(M)$ the sequence of configurations induced by w can be split into at most $n + 1$ strokes. Especially, for 1-turn DVPDAs, each sequence of configurations induced by an accepting word consists of one upstroke followed by at most one downstroke. Two subtleties arise when translating this concept to synchronization: (a) there is no initial state so that there is no way to associate a stroke counter with a state, and (b) there is no language of accepted words that restricts the set of words on which the number of strokes should be limited. Hence, in the synchronization setting the finite turn property is not a property of the push-down automaton but rather of the word applied to all states in parallel. We therefore generalize the concept of finite-turn DVPDAs to finite-turn synchronization for DVPDAs as follows.

class of automata	empty stack model	same stack model	arbitrary stack model
DVPDA	P	PSPACE-compl	PSPACE-hard
DVPDA-NoReturn	P	PSPACE-compl	P
DVPDA-Return	P	P	PSPACE-hard
n-Turn-Sync-DVPDA	PSPACE-hard	PSPACE-hard	PSPACE-hard
0-Turn-Sync-DVPDA	P	PSPACE-compl	PSPACE-compl
DVVPDA	P	P	P
n-Turn-Sync-DVVPDA	PSPACE-hard	PSPACE-hard	PSPACE-hard
0-Turn-Sync-DVVPDA	P	PSPACE-compl	PSPACE-compl
DVCA	P	P	P
n-Turn-Sync-DVCA	PSPACE-hard	PSPACE-hard	PSPACE-hard
1-Turn-Sync-DVCA	PSPACE-compl	PSPACE-compl	PSPACE-compl
0-Turn-Sync-DVCA	P	PSPACE-compl	PSPACE-compl

Table 1: Complexity status of the synchronization problem for different classes of deterministic real-time visibly push-down automata in different stack synchronization modes. For the n-turn synchronization variants, n takes all values not explicitly listed. All our problems are in EXPTIME.

Definition 2. n-Turn-Sync-DVPDA-Empty
Given: DVPDA $M = (Q, \Sigma, \Gamma, \delta, q_0, \bot, F)$.
Question: Is there a synchronizing word $w \in \Sigma^*$ in the empty stack model, such that for all states $q \in Q$, the sequence of configurations $(q, \bot) \xrightarrow{w} (\overline{q}, \bot)$ consists of at most $n+1$ strokes?

We call such a synchronizing word w an *n-turn synchronizing word* for M. We define n-Turn-Sync-DVPDA-Same and n-Turn-Sync-DVPDA-Arb accordingly for the same stack and arbitrary stack model. Further, we extend the problem definition to other classes of automata such as real-time DVVPDAs, and DVCAs. Table 1 summarizes our results, obtained in the next sections, on the complexity status of these problems together with the above introduced synchronization problems.

Finally, we introduce two PSPACE-complete problems for DFAs to reduce from later.

Definition 3 (DFA-Sync-Into-Subset (PSPACE-complete [Rystsov, 1983]))**.**
Given: DFA $A = (Q, \Sigma, \delta)$, subset $S \subseteq Q$.
Question: Is there a word $w \in \Sigma^*$ such that $\delta(Q, w) \subseteq S$?

Definition 4 (DFA-Sync-From-Subset (PSPACE-complete [Sandberg, 2005]))**.**
Given: DFA $A = (Q, \Sigma, \delta)$ with $S \subseteq Q$.
Question: Is there a word $w \in \Sigma^*$ that synchronizes S, i.e., for which $|\delta(S, w)| = 1$ is true?

3 DVPDAs – Distinguishing the Stack Models

We start with some positive result showing that we come down from the undecidability of the synchronization problem for general DPDAs in the empty set model to a polynomial time solvable version by considering visibly DPDAs.

Theorem 1. *The problems* Sync-DVPDA-Empty*,* Sync-DVCA-Empty*, and* Sync-DVVPDA-Empty *are decidable in polynomial time.*

Proof. We prove the claim for Sync-DVPDA-Empty as the other automata classes are sub-classes of DVPDAs. Let $M = (Q, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, \Gamma, \delta, \bot)$ be a DVPDA. First, observe that if Σ_{ret} is empty, then any synchronizing word w for M in the empty stack model cannot contain any letter from Σ_{call}. Hence, M is basically a DFA and for DFAs the synchronization problem is in P [Černý, 1964, Volkov, 2008, Sandberg, 2005]. From now on, assume $\Sigma_{\text{ret}} \neq \varnothing$. We show that a pair argument similar to the one for DFAs can be applied, namely that M is synchronizable in the empty stack model if and only if every pair of states $p, q \in Q$ can be synchronized in the empty stack model. The only if direction is clear as every synchronizing word for Q also synchronizes each pair of states. For the other direction, observe that since M is a DVPDA, the stack height of each path starting in any state of M is predefined by the sequence of input symbols. Hence, if we focus on the two runs starting in p, q and ensure that their stacks are empty after reading a word w, then also the stacks of all other runs starting in other states in parallel are empty after reading w. Therefore, we can successively concatenate words that synchronize some pair of active states in the empty stack model and end up with a word that synchronizes all states of M in the empty stack model.

In order to determine if a pair of states $p, q \in Q$ can be synchronized in the empty stack model, we build the following product automaton $M \times M[p, q] = (Q \times Q \cup Q, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, \Gamma, \delta^2, (p, q), \bot, Q)$. For all states in $(r, s) \in Q \times Q$ for which $r \neq s$, δ^2 simulates the actions of δ on r in the first component and the actions of δ on s in the second component. For states $(r, r) \in Q \times Q$, this is also the case for all transitions except for zero-tests of the stack, as here we map to the corresponding state $r \in Q$. For Q, δ^2, restricted to Q, is the same as δ. Clearly, $M \times M[p, q]$ accepts all words that have $w\sigma_r$ as a prefix, for which w synchronizes the states p and q in M in the empty stack model and σ_r is any return letter in Σ_{ret} that checks the empty stack condition. Further, for all pairs of states p, q, $M \times M[p, q]$ is a DVPDA. As the emptiness problem for DVPDAs is in P [Alur and Madhusudan, 2004], we can build and test all product automata $M \times M[p, q]$ for non-emptiness in polynomial time. □

Does this mean everything is easy and we are done? Interestingly, the picture is not that

simple, as considering the same and arbitrary stack models shows.

Theorem 2. *The problem* Sync-DVPDA-Same *is PSPACE-hard.*

Proof. We reduce from DFA-Sync-Into-Subset. Let $A = (Q, \Sigma, \delta)$ be a DFA and $S \subseteq Q$. We construct from A a DVPDA $M = (Q \cup \{q_S\}, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, \{☺, ☹, \bot\}, \delta' = \delta'_{\text{c}} \cup \delta'_{\text{i}} \cup \delta'_{\text{r}}, \bot)$ with $q_S \notin Q$, $\Sigma_{\text{call}} = \{a\}$, $\Sigma_{\text{int}} = \Sigma$, $\Sigma_{\text{ret}} = \varnothing$ and $\Sigma_{\text{call}} \cap \Sigma_{\text{int}} = \varnothing$. The transition function δ'_{i} agrees with δ on all letters in Σ_{int}. For q_S, we set $\delta'_{\text{c}}(q_S, a) = (q_S, ☺)$ and $\delta'_{\text{i}}(q_S, \sigma) = q_S$ for all $\sigma \in \Sigma_{\text{int}}$. For $q \in S$, we set $\delta'_{\text{c}}(q, a) = (q_S, ☺)$, and for $q \notin S$, $\delta'_{\text{c}}(q, a) = (q, ☹)$.

Note that q_S is a sink-state and can only be reached from states in S with a transition by the call-letter a. For states not in S, the input letter a pushes an ☹ on the stack which cannot be pushed to the stack by any letter on a path starting in q_S. Hence, in order to synchronize M in the same stack model, a letter a might only and must be read in a configuration where only states in $S \cup \{q_S\}$ are active. Every word $w \in \Sigma^*_{\text{int}}$ that brings M in such a configuration also synchronizes Q in A into the set S. □

From the proof of Theorem 2, we can conclude the next results by observing that a DVPDA without any return letter cannot make any turn.

Corollary 1. Sync-DVPDA-Same-NoReturn *and* 0-Turn-Sync-DVPDA-Same *are PSPACE-hard.*

In contrast with the two previous results, Sync-DVPDA-Same is solvable in polynomial time if we have the promise that $\Sigma_{\text{ret}} \neq \varnothing$.

Theorem 3. Sync-DVPDA-Same-Return *is in P.*

Proof. We prove the claim by straight reducing to Sync-DVPDA-Empty with the identity function. If a DVPDA M with $\Sigma_{\text{ret}} \neq \varnothing$ can be synchronized in the same stack model with a synchronizing word w, then w can be extended to ww' where $w' \in \Sigma^*_{\text{ret}}$ empties the stack. As M is deterministic and complete, w' is defined on all states. As after reading w, the stack content on all paths is the same, reading w' extends all paths with the same sequence of states. Conversely, a word w synchronizing a DVPDA M with $\Sigma_{\text{ret}} \neq \varnothing$ in the empty stack model also synchronizes M in the same stack model. □

The arbitrary stack model requires the most interesting construction in the following proof.

Theorem 4. Sync-DVPDA-Arb *is PSPACE-hard.*

Proof. We give a reduction from the PSPACE-complete problem DFA-Sync-From-Subset. Let $A = (Q, \Sigma, \delta)$ be a DFA with $S \subseteq Q$. We construct from A a DVPDA $M = (Q, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, Q \cup \{\bot\}, \delta' = \delta'_{\text{c}} \cup \delta'_{\text{i}} \cup \delta'_{\text{r}}, \bot)$ where all unions in the definition of M are disjoint. Let $\Sigma_{\text{call}} = \Sigma$, $\Sigma_{\text{int}} = \varnothing$, and $\Sigma_{\text{ret}} = \{r\}$ with $r \notin \Sigma$.

For states $s \in S$, we set $\delta'_{\text{r}}(s, r, \bot) = s$ and for states $q \in Q \backslash S$, we set $\delta'_{\text{r}}(q, r, \bot) = t$ for some arbitrary but fixed $t \in S$. For states $p, q \in Q$, we set $\delta'_{\text{r}}(q, r, p) = p$.

For each call letter $\sigma \in \Sigma_{\text{call}}$, we set for $q \in Q$, $\delta'_{\text{c}}(q, \sigma) = (\delta(q, \sigma), q)$.

First, assume w is a word that synchronizes the set S in the DFA A. Then, it can easily be observed that rw is a synchronizing word for M in the arbitrary stack model.

Now, assume w is a synchronizing word for M in the arbitrary stack model. If $w \in \Sigma^*_{\text{call}}$, then w is also a synchronizing word for A and especially synchronizes the set S in A. (*) Next, assume w contains some letters r. The action of r is designed such that it maps Q to the set S if applied to an empty stack and otherwise gradually undoes the transitions performed by letters from Σ_{call}. This is possible as each letter $\sigma \in \Sigma_{\text{call}}$ stores its pre-image on the stack when σ is applied. Further, r acts as the identity on the states in S if applied to an empty stack. Hence, whenever the stacks are empty while reading some word, all states in S are active.

Hence, if σr is a subword of a synchronizing word $w = u\sigma r v$ of M, with $\sigma \in \Sigma_{\text{call}}$, then $w' = uv$ is also a synchronizing word of M. This justifies the set of rewriting rules $R = \{\sigma r \to \varepsilon \mid \sigma \in \Sigma_{\text{call}}\}$. Now, consider a synchronizing word w of M where none of the rewriting rules from R applies, and, which by (*) contains some letter r. Hence, $w \in \{r\}^* \Sigma^*_{\text{call}}$. By (*), $w = r^k v$, with $k > 0$, and $v \in \Sigma^*_{\text{call}}$. Then, $w' = rv$ is also a synchronizing word of M, because for all states $q \in Q$, M is in the same configuration after reading r, starting in configuration $(q, \bot)$, as after reading rr. But as only (and all) states from S are active after reading r, v is also a word in Σ^* that synchronizes the set S in A. □

Observe that in the construction above, $\Sigma_{\text{ret}} \neq \varnothing$ for all input DFAs. The next corollary follows from Theorem 4 and should be observed together with the next theorem in contrast to Theorem 3 and Corollary 1.

Corollary 2. Sync-DVPDA-Arb-Return *is PSPACE-hard.*

Theorem 5. Sync-DVPDA-Arb-NoReturn ≡ DFA-Sync.

Proof. Let M be a DVPDA with empty set of return symbols. As there is no return-symbol, the transitions of M cannot depend on the stack content. Hence, we can redis-

tribute the symbols in Σ_{call} into Σ_{int} and obtain a DFA. The converse is trivial. □

If we move from deterministic visibly push-down automata to even more restricted classes, like deterministic very visibly push-down automata or deterministic visibly counter automata, the three stack models do no longer yield synchronization problems with different complexities. Instead, all three models are equivalent, as stated next. Hence, their synchronization problems can be solved by the pair-argument presented in Theorem 1 in polynomial time.

Theorem 6. Sync-DVCA-Empty ≡ Sync-DVCA-Same ≡ Sync-DVCA-Arb. Sync-DVVPDA-Empty ≡ Sync-DVVPDA-Same ≡ Sync-DVVPDA-Arb.

Proof. First, note that every DVCA is also a DVVPDA. If for a DVVPDA $\Sigma_{\text{ret}} \neq \varnothing$, then we can empty the stack after synchronizing the state set, as the very visibly conditions ensures that the contents of the stacks on all runs coincide. As the automaton is deterministic, all transitions for letters in Σ_{ret} are defined on each state. As the stack content on all runs coincides in every step, the arbitrary stack model is identical to the same stack model and hence equivalent to the empty stack model. If $\Sigma_{\text{ret}} = \varnothing$, then we can reassign Σ_{call} to Σ_{int} in order to reduce from the same-stack and arbitrary stack to the empty stack variant, as transitions cannot depend on the stack content which is again the same on all runs due to the very visibly condition. □

4 Restricting the Number of Turns Makes Synchronization Harder

Let us now restrict the number of turns a synchronizing word may cause on any run. Despite the fact that we are hereby restricting the considered model even further, the synchronization problem becomes even harder, in contrast to the previous section.

Theorem 7. *For every fixed $n \in \mathbb{N}$ with $n > 0$, the problems* n-Turn-Sync-DVCA-Same *and* n-Turn-Sync-DVCA-Arb *are PSPACE-hard.*

Proof. We reduce from the PSPACE-complete problem DFA-Sync-Into-Subset. Let $A = (Q, \Sigma, \delta)$ be a DFA with $S \subseteq Q$. We construct from A a DVCA $M = (Q \cup \{q_{\text{sync}}\} \cup \{q_{\text{stall}_i} \mid 0 \leq i \leq n\}, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, \{1, \bot\}, \delta' = \delta'_{\text{c}} \cup \delta'_{\text{i}} \cup \delta'_{\text{r}}, \bot\})$, where all unions are disjoint. We set $\Sigma_{\text{int}} = \Sigma$, $\Sigma_{\text{call}} = \{a\}$ and $\Sigma_{\text{ret}} = \{b\}$. For all internal letters, δ'_{i} agrees with δ on all states in Q. For the letter a, we set for all $q \in S$, $\delta'_{\text{c}}(q, a) = (q_{\text{stall}_0}, 1)$ and for all $q \in Q \backslash S$,

we set $\delta'_c(q,a) = (q,1)$. For b, we loop in every state in Q. For q_{sync}, we loop with every letter in q_{sync} (incrementing the counter with a and decrementing it with b).

Let r be an arbitrary but fixed state in Q. For the states q_{stall_i}, we set for $i < n$, $\delta'_c(q_{\text{stall}_i}, a) = (q_{\text{stall}_i}, 1)$. Further, for even index $i < n$, we set $\delta'_r(q_{\text{stall}_i}, b, 1) = q_{\text{stall}_{i+1}}$ and $\delta'_r(q_{\text{stall}_i}, b, \bot) = r$. For odd index $i < n$, we set $\delta'_r(q_{\text{stall}_i}, b, 1) = r$, and $\delta'_r(q_{\text{stall}_i}, b, \bot) = q_{\text{stall}_{i+1}}$. For even n, let $\delta'_c(q_{\text{stall}_n}, a) = (q_{\text{sync}}, 1)$, $\delta'_r(q_{\text{stall}_n}, b, 1) = r$, and $\delta'_r(q_{\text{stall}_n}, b, \bot) = r$. For odd n, let $\delta'_c(q_{\text{stall}_n}, a) = (q_{\text{stall}_n}, 1)$, $\delta'_r(q_{\text{stall}_n}, b, 1) = r$, and $\delta'_r(q_{\text{stall}_n}, b, \bot) = q_{\text{sync}}$. All other transitions (on internal letters) act as the identity.

Observe that the state q_{sync} must be the synchronizing state of M, since it is a sink state. In order to reach q_{sync} from any state in Q, the automaton must pass through all the states q_{stall_i} for all $0 \le i \le n$, by construction. Since we can only pass from a state q_{stall_i} to $q_{\text{stall}_{i+1}}$ with an empty or non-empty stack in alternation, passing through all states q_{stall_i}, for $0 \le i \le n$, forces M to make n turns. For even n, the last upstroke is enforced by passing from q_{stall_n} to q_{sync} by explicitly increasing the stack. As M is only allowed to make n turns while reading the n-turn synchronizing word it follows that any of the states q_{stall_i} might be visited at most once, as branching back into Q by taking a transition that maps to r would force M to go through all states q_{stall_i} again, which exceeds the allowed number of strokes. Note that only counter values of at most one are allowed in any run which is currently in a state in q_{stall_i} as otherwise the run will necessarily branch back into Q later on.[2] Especially, this is the case for q_{stall_0} which ensures that each n-turn synchronizing word has first synchronized Q into S before the first letter a is read, as otherwise q_{stall_0} is reached with a counter value greater than 1, or M has already made a turn in Q and hence cannot reach q_{sync} anymore.

In the construction above, for odd n, each run enters the synchronizing state with an empty stack (*). For even n, each run enters the synchronizing state with a counter value of 1. The visibly condition, or more precisely very visibly condition as we are considering DVCAs, tells us that at each time while reading a synchronizing word, the stack content of every run is identical. In particular, this is the case at the point when the last state enters the synchronizing state and, hence, any n-turn synchronizing word for M is a synchronizing word in both the arbitrary and the same stack models. □

By observing that in the empty stack model allowing n even turns is as good as allowing $(n-1)$ turns, essentially (*) from the previous proof yields the next result.

Corollary 3. *For every fixed $n \in \mathbb{N}$, with $n > 0$, the problem* n-Turn-Sync-DVCA-Empty *is PSPACE-hard.*

[2] In some states, such as q_{stall_n} for even n, it is simply impossible to have a higher counter value.

Proof. Since we need to synchronize with an empty stack, for even n, the last upstroke cannot be performed. Hence, for even n, every DVCA M can be synchronized by an n-turn synchronizing word if and only if M can be synchronized by an $(n-1)$-turn synchronizing word. As for odd n in the construction above, every n-turn synchronizing word synchronized M in the empty stack model, the claim follows from the proof in Theorem 7. □

Corollary 4. *For every fixed $n \in \mathbb{N}$, with $n > 0$, the problems* n-Turn-Sync-DVPDA *and* n-Turn-Sync-DVVPDA *in the empty, same, and arbitrary stack models are PSPACE-hard.*

Theorem 8. 0-Turn-Sync-DVPDA-Empty ≡ DFA-Sync.

Proof. The visibly condition and the fact that we can only synchronize with an empty stack mean that we cannot read any letter from Σ_{call}, hence we cannot use the stack at all. Delete (a) all transitions with a symbol from Σ_{call} and (b) all transitions with a symbol from Σ_{ret} and a non-empty stack. Then, assigning the elements in Σ_{ret} to Σ_{int} gives us a DFA. □

In contrast to the empty stack model, for the same and arbitrary stack model the finite turns variant of the synchronization problem remains PSPACE-hard also for zero turns.

Theorem 9. *The problems* 0-Turn-Sync-DVCA-{Same, Arb} *are PSPACE-hard.*

Proof. We give a reduction from the PSPACE-complete problem DFA-Sync-From-Subset. Let $A = (Q, \Sigma, \delta)$ be a DFA with $S \subseteq Q$. We construct from A a DVCA $M = (Q, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, \{1, \bot\}, \delta' = \delta'_{\text{c}} \cup \delta'_{\text{i}} \cup \delta'_{\text{r}}, \bot)$. We set $\Sigma_{\text{call}} = \Sigma$, $\Sigma_{\text{int}} = \varnothing$, and $\Sigma_{\text{ret}} = \{b\}$. For all $q \in Q$ and $\sigma \in \Sigma_{\text{call}}$, we set $\delta'_{\text{c}}(q, \sigma) = (\delta(q, \sigma), 1)$. For all states $q \in Q \backslash S$, we set $\delta'_{\text{r}}(q, b, \bot) = s$ for some arbitrary but fixed state $s \in S$. All other transitions act as the identity.

Note that the 0-turn condition only allows us to read the letter b before any letter in Σ_{call} has been read, as afterwards b would decrease the stack after it has been increased. Therefore, every synchronizing word for M in the same and arbitrary stack models also synchronizes S in Q by either synchronizing the whole set Q without using any b transition, or it brings Q in exactly the set S with the first letter b and continues to synchronize S. □

Corollary 5. *The problems* 0-Turn-Sync-DVVPDA-{Same, Arb}, *and* 0-Turn-Sync-DVPDA-{Same, Arb} *are PSPACE-hard.*

5 (Non-)Tight Upper Bounds

In this section, we will prove that at least all considered problems are decidable (in contrast to non-visibly DPDAs and DCAs, see [Fernau et al., 2020]) by giving exponential time upper bounds. We will also give some tight PSPACE upper bounds for some PSPACE-hard problems discussed in the previous section, but for other previously discussed problems, a gap between PSPACE-hardness and membership in EXPTIME remains.

Theorem 10. *All problems listed in Table 1 are in EXPTIME.*

Proof. We show the claim explicitly for Sync-DVPDA-Same, Sync-DVPDA-Arb, n-Turn-Sync-DVPDA-Empty, n-Turn-Sync-DVPDA-Same, and n-Turn-Sync-DVPDA-Arb. The other results follow by inclusion of automata classes.

Let $M = (Q, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, \Gamma, \delta, \bot)$ be a DVPDA. We construct from M the $|Q|$-fold product DVPDA $M^{|Q|}$ with state set $Q^{|Q|}$, consisting of $|Q|$-tuples of states, and alphabet $\Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}$. Since M is a DVPDA, for every word $w \in (\Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}})^*$, the stack heights on runs starting in different states in Q is equal at every position in w. Hence, we can multiply the stacks to obtain the stack alphabet $\Gamma^{|Q|}$ for $M^{|Q|}$. For the transition function $\delta^{|Q|}$ (split up into $\delta_{\text{c}}^{|Q|} \cup \delta_{\text{i}}^{|Q|} \cup \delta_{\text{r}}^{|Q|}$) of $M^{|Q|}$, we simulate δ independently on every state in an $|Q|$-tuple, i.e., for $(q_1, q_2, \ldots, q_n) \in Q^{|Q|}$ and letters $\sigma_c \in \Sigma_{\text{call}}, \sigma_i \in \Sigma_{\text{int}}, \sigma_r \in \Sigma_{\text{ret}}$, we set

- $\delta_{\text{c}}^{|Q|}((q_1, q_2, \ldots, q_n), \sigma_c) = ((q_1', q_2', \ldots, q_n'), (\gamma_1, \gamma_2, \ldots, \gamma_n))$ if $\delta(q_j, \sigma_c) = (q_j', \gamma_j)$ for $j \in [n]$;
- $\delta_{\text{i}}^{|Q|}((q_1, q_2, \ldots, q_n), \sigma_i) = (\delta(q_1, \sigma_i), \delta(q_2, \sigma_i), \ldots, \delta(q_n, \sigma_i))$;
- $\delta_{\text{r}}^{|Q|}((q_1, q_2, \ldots, q_n), \sigma_r, (\gamma_1, \gamma_2, \ldots, \gamma_n)) = (\delta(q_1, \sigma_r, \gamma_1), \delta(q_2, \sigma_r, \gamma_2), \ldots, \delta(q_n, \sigma_r, \gamma_n))$.

The bottom symbol of the stack is the $|Q|$-tuple $(\bot, \bot, \ldots, \bot)$. Let $p_1, p_2, \ldots, p_n$ be an enumeration of the states in Q and set $(p_1, p_2, \ldots, p_n)$ as the start state of $M^{|Q|}$.

For Sync-DVPDA-Arb, set $\{(q, q, \ldots, q) \in Q^{|Q|} \mid q \in Q\}$ as the final states for $M^{|Q|}$. Clearly, for Sync-DVPDA-Arb, $M^{|Q|}$ is a DVPDA and the words accepted by $M^{|Q|}$ are precisely the synchronizing words for M in the arbitrary stack model. As the emptiness problem can be decided for visibly push-down automata in time polynomial in the size of the automaton [Alur and Madhusudan, 2004], the claim follows observing that $M^{|Q|}$ is exponentially larger than M.

For Sync-DVPDA-Same, we produce a DVPDA $M^{|Q|}_{\text{same}}$ by enhancing the automaton $M^{|Q|}$ with three additional states q_{check}, q_{fin}, and q_{fail} and an additional new return letter r and set q_{fin} as the single accepting state of $M^{|Q|}_{\text{same}}$, while the start state coincides with the the start state of $M^{|Q|}$. For states $(q_1, q_2, \ldots, q_n) \in Q^{|Q|}$, we set $\delta^{|Q|}_{\text{r}}((q_1, q_2, \ldots, q_n), r, (\gamma_1, \gamma_2, \ldots, \gamma_n)) = q_{\text{check}}$ if $q_i = q_j$ and $\gamma_i = \gamma_j$, $\gamma_i \neq \bot$ for all $i, j \in [n]$. We set $\delta^{|Q|}_{\text{r}}((q_1, q_2, \ldots, q_n), r, (\bot, \bot, \ldots, \bot)) = q_{\text{fin}}$ if $q_i = q_j$ for all $i, j \in [n]$. For all other cases, we map with r to q_{fail}. We let the transitions for q_{fail} be defined such that q_{fail} is a non-accepting trap state for all alphabet symbols. For q_{check}, we set $\delta^{|Q|}_{\text{r}}(q_{\text{check}}, r, (\gamma_1, \gamma_2, \ldots, \gamma_n)) = q_{\text{check}}$ if $\gamma_i = \gamma_j \neq \bot$ for all $i, j \in [n]$. Further, we set $\delta^{|Q|}_{\text{r}}(q_{\text{check}}, r, (\bot, \bot, \ldots, \bot)) = q_{\text{fin}}$ and map with r to q_{fail} in all other cases. The state q_{check} also maps to q_{fail} with all input symbols other than r. We let the transitions for q_{fin} be defined such that q_{fin} is an accepting trap state for all alphabet symbols.

Clearly, for Sync-DVPDA-Same $M^{|Q|}_{\text{same}}$ is a DVPDA and the words accepted by $M^{|Q|}_{\text{same}}$ are precisely the synchronizing words for M in the same stack model, potentially prolonged by a sequence of r's, as the single accepting state q_{fin} can only be reached from a state in $Q^{|Q|}$ where the states are synchronized and the stack content is identical for each run (which is checked in the state q_{check}). As the size of $M^{|Q|}_{\text{same}}$ is exponential in the size of M, we get the claimed result as in the previous case.

For the n-Turn synchronization problems, we have to modify the previous construction by adding a stroke counter similar as in the proof of Theorem 7.

For the problems n-Turn-Sync-DVPDA in the empty, same, and arbitrary stack models, we enhance in $M^{|Q|}$ each $|Q|$-tuple with an additional index $I \in \{0, 1, \ldots, n+1\}$, i.e., the basic set of states is now $Q^{|Q|} \times \{0, 1, \ldots, n+1\}$. We further add for all three models the non-accepting trap state q_{fail} to the set of states. For each $(|Q|+1)$-tuple, we implement the transition function $\delta^{|Q|}_{\text{i}}$ of $M^{|Q|}$ for internal letters in Σ_{int} as before by keeping the value of the index I in each transition. For call letters in Σ_{call}, we realize $\delta^{|Q|}_{\text{c}}$ as before for state-tuples with index $I < n+1$ by simulation δ on the individual states and setting in every image $I = I + 1$ if I is even, and keeping the value of I if I is odd. For tuples with index $I = n+1$, we proceed as before for smaller index if $n+1$ is odd, while for even $n+1$ we map with a call letter to the state q_{fail}. For the return letters in Σ_{ret}, we realize $\delta^{|Q|}_{\text{r}}$ for pairs of states in $Q^{|Q|} \times \{1, 2, \ldots, n+1\}$ and bottom of stack symbol $(\bot, \bot, \ldots, \bot)$ as before by simulating δ on the individual states and keeping the value of I. For all other stack symbols, we realize $\delta^{|Q|}_{\text{r}}$ as before for state-tuples with index $0 < I < n+1$ by simulation δ on the individual states and keeping the value of I if I is even, and setting in every image $I = I + 1$ if I is odd. For tuples with $I = n+1$, we proceed as before if $n+1$ is even. For states with index $I = 0$ or $I = n+1$ for odd $n+1$, we map with each return letter to q_{fail} for stack symbols other than the bottom of stack

symbol. In all three models we set $(p_1, p_2, \ldots, p_n, 0)$ as the start state, with $p_1, p_2, \ldots, p_n$ being an enumeration of the states in Q.

For n-Turn-Sync-DVPDA-Arb, we set $\{(q, q, \ldots, q, I) \mid q \in Q, I \in \{0, 1, \ldots, n+1\}\} \subset Q^{|Q|} \times \{0, 1, \ldots, n+1\}$ as the set of final states.

For n-Turn-Sync-DVPDA-Empty, we set the additional trap state q_{fin} as the single accepting state and add a new return letter r with which we map to q_{fin} for states $(q, q, \ldots, q, I) \in Q^{|Q|} \times \{0, 1, \ldots, n+1\}$ with the bottom-of-stack symbol and to q_{fail} for all other stack symbols or states.

For n-Turn-Sync-DVPDA-Same, we add the two states q_{check} and q_{fin} to $M^{|Q|}$ and set q_{fin} as the single accepting state. Again, we add a new return letter r. For states $(q, q, \ldots, q, I)$ with $I \in \{0, 1, \ldots, n+1\}$ and symbol $(\gamma_1, \gamma_2, \ldots, \gamma_n)$ on top of the stack, which is not the bottom-of-stack symbol, we map with r to q_{check} if all entries γ_i in the stack symbol tuple are identical. If instead the bottom-of-stack symbol is on top of the stack, we map with r for states $(q, q, \ldots, q, I)$ directly to q_{fin}. For all other states and stack symbols, r maps to q_{fail}. For q_{check}, we stay in q_{check} with the letter r if we see a symbol $(\gamma, \gamma, \ldots, \gamma)$ on the stack with $\gamma \in \Gamma \backslash \{\bot\}$ and map with r to q_{fin} if we see the bottom-of-stack symbol. For all other stack symbols, r maps q_{check} to q_{fail}. Also, all input letter other than r maps q_{check} to q_{fail}. For q_{fin}, we define all transitions such that q_{fin} is a trap state.

In all three cases, the constructed automaton is a DVPDA that accepts precisely the n-turn synchronizing words for M (potentially prolonged by a sequence of r's) in the respective stack model. As the constructed automaton is of size $\mathcal{O}((|Q||\Gamma|)^{|Q|})$ in all three cases, we can decide whether the constructed automaton accepts at least one word in time exponential in the description of M. □

Remark 1. It cannot be expected to show PSPACE-membership of synchronization problems concerning DVPDAs using a $|Q|$-fold product DVPDA as the resulting automata is exponentially large in the size of the DVPDA that is to be synchronized and the emptiness problem for DVPDAs is P-complete [Okhotin and Salomaa, 2014]. Rather, one would need a separate membership proof. We conjecture that a PSPACE-membership proof similar to the one presented in [Chistikov et al., 2019] for the short synchronizing word problem can be obtained if exponential upper bounds for the length of shortest synchronizing words for DVPDAs in the respective models can be obtained. For the empty stack model, an exponential upper bound on the length of a shortest synchronizing word should follow by applying analogous arguments as in [Chistikov et al., 2019, Theorem 6]. For the same and arbitrary stack model, the question is open as we cannot reduce the problem to considering pairs like in the empty stack model.

Theorem 11. *The problems* 0-Turn-Sync-{DVPDA, DVVPDA, DVCA}-Same *are in PSPACE.*

Proof. Let $M = (Q, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, \Gamma, \delta = \delta_{\text{c}} \cup \delta_{\text{i}} \cup \delta_{\text{r}}, \bot)$ be a DVPDA. For the same stack model, the 0-turn condition forbids us to put in simultaneous runs different letters on the stack at any time while reading a synchronizing word, as we cannot exchange symbols on the stack with visible PDAs. Note that this is a dynamic runtime-behavior and does not imply that M is necessarily very visibly. Further, the 0-turn and visibility condition enforces that at each step the next transition does not depend on the stack content if the symbol on top of the stack is not $\bot$. We construct from M a partial $|Q|$-fold product DFA $M^{|Q|}$ with state set $Q^{|Q|} \times \{0,1\}$, consisting of $|Q|$-tuples of states with an additional bit of information which will indicate whether the stack is still empty, and alphabet $\Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}$. For the transition function $\delta^{|Q|}$ of $M^{|Q|}$, we simulate for a state $(q_1, q_2, \ldots, q_n, b)$ with $q_1, q_2, \ldots, q_n \in Q$, $b \in \{0,1\}$ and letter σ, δ (by restricting the image to the first component in Q for call letters) on the individual states q_i, q_j with $i, j \in [n]$ in the tuple if (1) $\sigma \in \Sigma_{\text{ret}}$ and $b = 0$, (2) $\sigma \in \Sigma_{\text{int}}$, or (3) $\sigma \in \Sigma_{\text{call}}$ and for $\delta_{\text{c}}(q_i, \sigma) = (q_i', \gamma_i)$, $\delta_{\text{c}}(q_j, \sigma) = (q_j', \gamma_j)$ it holds that $\gamma_i = \gamma_j$. In case (1) and (2), we keep the value of b in the transition and in case (3), we ensure $b = 1$ in the image of the transition. The size of the state graph of $M^{|Q|}$ is bounded by $\mathcal{O}(|Q|^{|Q|})$. Clearly, the DVPDA M can be synchronized by a 0-turn synchronizing word in the same stack model if and only if there is a path in the state graph of $M^{|Q|}$ from the state $(q_1, q_2, \ldots, q_n, 0)$ for $Q = \{q_1, q_2, \ldots, q_n\}$ to some state in $\{(q_i, q_i, \ldots, q_i, b) \mid i \in [n], b \in \{0,1\}\}$. These $2|Q|$ reachability tests can be performed in NPSPACE = PSPACE [Savitch, 1970]. The claim for the other problems follows by inclusion of automata classes. □

Corollary 6. *The problems* Sync-DVPDA-Same-NoReturn *and* Sync-DVPDA-Same *are in PSPACE.*

Proof. Consider first Sync-DVPDA-Same-NoReturn. Let $M = (Q, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, \Gamma, \delta, \bot)$ be a DVPDA with $\Sigma_{\text{ret}} = \varnothing$. As we have no return letter, any synchronizing word for M is also a 0-turn synchronizing word and hence, the claim follows with Theorem 11.

Consider now Sync-DVPDA-Same. For a DVPDA M with return alphabet Σ_{ret}, either $\Sigma_{\text{ret}} = \varnothing$, in which case the previous argument applies, or $\Sigma_{\text{ret}} \neq \varnothing$, which allows us to even conclude membership in P by Theorem 3. □

Theorem 12. *The problems* 0-Turn-Sync-{DVPDA, DVVPDA, DVCA}-Arb*, and* 1-Turn-Sync-DVCA-{Empty, Same, Arb} *are in PSPACE.*

Proof. The claim follows from [Fernau et al., 2020, Theorem 16 & 17] by inclusion of automata classes. □

6 Sequential Transducers

In [Fernau et al., 2020], the concept of trace-synchronizing a sequential transducer has been introduced. We want to extend this concept to sequential transducers showing some kind of *visible behavior* regarding their output, inspired by the predetermined stack height behavior of DVPDAs. We call $T = (Q, \Sigma, \Gamma, q_0, \delta, F)$ a *sequential transducer* (ST for short) if Q is a finite set of states, Σ is an input alphabet, Γ is an output alphabet, q_0 is the start state, $\delta: Q \times \Sigma \to Q \times \Gamma^*$ is a total transition function, and F collects the final states. We generalize δ from input letters to words by concatenating the produced outputs. T is called a *visibly sequential transducer* (VST for short) [or *very visibly sequential transducer* (VVST for short)] if for each $\sigma \in \Sigma$ and for all $q_1, q_2 \in Q$ and $\gamma_1, \gamma_2 \in \Gamma^*$, it holds that $\delta(q_1, \sigma) = (q_1', \gamma_1)$ and $\delta(q_2, \sigma) = (q_2', \gamma_2)$ implies that $|\gamma_1| = |\gamma_2|$ [or that $\gamma_1 = \gamma_2$, respectively]. A VVST T is thereby computing the same homomorphism h_T, regardless of which states are chosen as start and final states (*). Hence, if A_T is the underlying DFA (ignoring any outputs), then $h_T(\mathcal{L}(A_T)) \subseteq \Gamma^*$ describes the language of all possible output of T. By Nivat's theorem [Nivat, 1968], a language family is a full trio iff it is closed under VVST and inverse homomorphisms. Our considerations also show that a language family is a full trio iff it is closed under VVST and inverse VVST mappings.

We say that a word w *trace-synchronizes* a sequential transducer T if, for all states $p, q \in Q$, $\delta(p, w) = \delta(q, w)$, i.e., a synchronizing state is reached, producing identical output. Notice that from the viewpoint of trace-synchronization, we do not assume that a VVST has only one state.

Definition 5 (Trace-Sync-Transducer)**.**
Given: Sequential transducer $T = (Q, \Sigma, \Gamma, \delta)$.
Question: Does there exist a word $w \in \Sigma^*$ that trace-synchronizes T?

Remarks on sequential transducers. The definitions in the literature are not very clear for finite automata with outputs. We follow here the name used by Berstel in [Berstel, 1979]; Ginsburg [Ginsburg, 1966] called Berstel's sequential transducers *generalized machines*, but used the term *sequential transducer* for the nondeterministic counterpart. For non-deterministic transducers which allow to read multiple letters at once, the concept of fixing the ratio between the length of the produced output and the length of the

input was already studied in [Carton, 2007] and was even mentioned in [Sakarovitch, 2003]. Here, the ratio is fixed for every transition independent of the input letter(s) and a transducer admitting such a fixed ratio α is called *α-synchronous*. The term 'synchronization' again appears here but refers to finding an α-synchronous transducer to a given rational relation.

We define Trace-Sync-VST and Trace-Sync-VVST by considering a VST, respectively VVST. In contrast to the undecidability of the problem Trace-Sync-Transducer [Fernau et al., 2020], we get the following results for trace-synchronizing VST and VVST from previous results.

Theorem 13. Trace-Sync-VST *is PSPACE-complete.*

Proof. First, observe that there is a straight reduction from the problem Sync-DVPDA-Same-NoReturn to Trace-Sync-VST as the input DVPDAs to the problem Sync-DVPDA-Same-NoReturn have no return letters and, hence, the stack is basically a write only tape. Further, as the remaining alphabet is partitioned into letters in Σ_{call}, which write precisely one symbol on the stack, and into letters in Σ_{int}, writing nothing on the stack, the visibly condition is satisfied when interpreting the DVPDA with $\Sigma_{\text{ret}} = \varnothing$ as a VST.

There is also a straight reduction from Trace-Sync-VST to Sync-DVPDA-Same-NoReturn as follows. For a VST $T = (Q, \Sigma, \Gamma, \delta)$, we construct a DVPDA $M = (Q, \Sigma_{\text{call}} \cup \Sigma_{\text{int}}, \Gamma', \delta)$ with $\Sigma_{\text{ret}} = \varnothing$ by introducing for each $\sigma \in \Sigma$ a new alphabet $\Sigma_\sigma = \{w \in \Gamma^* \mid \exists q, q' \in Q \colon \delta(q, \sigma) = (q', w)\}$. Observe that Σ_σ is either $\{\epsilon\}$ or contains only words of the same length. By setting $\Sigma_{\text{int}} = \{\sigma \in \Sigma \mid \Sigma_\sigma = \{\epsilon\}\}$, $\Sigma_{\text{call}} = \{\sigma \in \Sigma \mid \Sigma_\sigma \neq \{\epsilon\}\}$, $\Gamma' = \bigcup_{\sigma \in \Sigma}(\Sigma_\sigma \backslash \{\epsilon\})$, and interpreting the output sequence $w \in \Gamma^*$ produced by δ as the single stack symbol in Γ'. □

Yet, by Observation (*), we inherit from Sync-DFA the following algorithmic result.

Theorem 14. Trace-Sync-VVST *is in P.*

Proof. For each VVST $T = (Q, \Sigma, \Gamma, \delta)$ and $w \in \Sigma^*$, the same output is already produced in $\delta(q, w)$ for all $q \in Q$. Hence, we can ignore the output and test T for trace-synchronization by the polynomial time pair-algorithm for DFAs [Sandberg, 2005]. □

7 Discussion

Our results concerning DVPDAs and sub-classes thereof are summarized in Table 1. While all problems listed in the table are contained in EXPTIME, the table lists several problems for which their known complexity status still contains a gap between PSPACE-hardness lower bounds and EXPTIME upper bounds. Presumably, their precise complexity status is closely related to upper bounds on the length of synchronizing words which we want to consider in the near future. One of the questions which could be solved in this work is if there is a difference between the complexity of synchronization in the same stack model and synchronization in the arbitrary stack model. While for general DPDA, DCA, and sub-classes thereof, see [Fernau et al., 2020], these two models admitted synchronization problems with the same complexity, here we observed that these models can differ significantly. While the focus of this work is on determining the complexity status of synchronizability for different models of automata, an obvious question for future research is the complexity status of closely related, and well understood questions in the realm of DFAs, such as the problem of shortest synchronizing word, subset synchronization, synchronization into a subset, and careful synchronization.

Here is one subtlety that comes with shortest synchronizing words: While for finding synchronizing words of length at most k for DFAs, it does not matter if the number k is given in unary or in binary due to the known cubic upper bounds on the lengths of shortest synchronizing words, this will make a difference in other models where such polynomial length bounds are unknown. More precisely, for instance with DVPDAs, it is rather obvious that with a unary length bound k, the problem becomes NP-complete, while the status is unclear for binary length bounds. As there is no general polynomial upper bound on the length of shortest synchronizing words for VPDAs, they might be of exponential length. Hence, we do not get membership in PSPACE easily, not even for synchronization models concerning DVPDA for which general synchronizability is solvable in P, as it might be necessary to store the whole word on the stack in order to test its synchronization effects.

References

[Alur and Madhusudan, 2004] Alur, R. and Madhusudan, P. (2004). Visibly pushdown languages. In Babai, L., editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 202–211. ACM.

[Alur and Madhusudan, 2009] Alur, R. and Madhusudan, P. (2009). Adding nesting

structure to words. *Journal of the ACM*, 56(3):16:1–16:43.

[Arenas et al., 2011] Arenas, M., Barceló, P., and Libkin, L. (2011). Regular languages of nested words: Fixed points, automata, and synchronization. *Theory of Computing Systems*, 49(3):639–670.

[Babari et al., 2016] Babari, P., Quaas, K., and Shirmohammadi, M. (2016). Synchronizing data words for register automata. In Faliszewski, P., Muscholl, A., and Niedermeier, R., editors, *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, volume 58 of *LIPIcs*, pages 15:1–15:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Bárány et al., 2006] Bárány, V., Löding, C., and Serre, O. (2006). Regularity problems for visibly pushdown languages. In Durand, B. and Thomas, W., editors, *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science*, volume 3884 of *Lecture Notes in Computer Science*, pages 420–431. Springer.

[Béal and Perrin, 2016] Béal, M.-P. and Perrin, D. (2016). *Synchronised Automata*, page 213–240. Encyclopedia of Mathematics and its Applications. Cambridge University Press.

[Berstel, 1979] Berstel, J. (1979). *Transductions and Context-Free Languages*, volume 38 of *Teubner Studienbücher: Informatik*. Teubner.

[Bollig, 2016] Bollig, B. (2016). One-counter automata with counter observability. In Lal, A., Akshay, S., Saurabh, S., and Sen, S., editors, *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016, Proceedings*, volume 65 of *LIPIcs*, pages 20:1–20:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Carton, 2007] Carton, O. (2007). The growth ratio of synchronous rational relations is unique. *Theoretical Computer Science*, 376(1-2):52–59.

[Caucal, 2006] Caucal, D. (2006). Synchronization of pushdown automata. In Ibarra, O. H. and Dang, Z., editors, *Developments in Language Theory, 10th International Conference, DLT 2006, Santa Barbara, CA, USA, June 26-29, 2006, Proceedings*, volume 4036 of *Lecture Notes in Computer Science*, pages 120–132. Springer.

[Černý, 1964] Černý, J. (1964). Poznámka k homogénnym eksperimentom s konečnými automatami. *Matematicko-fyzikalny Časopis Slovensk*, 14(3):208–215.

[Černý, 2019] Černý, J. (2019). A note on homogeneous experiments with finite automata. *Journal of Automata, Languages and Combinatorics*, 24(2-4):123–132.

[Chistikov et al., 2019] Chistikov, D., Martyugin, P., and Shirmohammadi, M. (2019). Synchronizing automata over nested words. *Journal of Automata, Languages and Combinatorics*, 24(2-4):219–251.

[Doyen et al., 2014] Doyen, L., Juhl, L., Larsen, K. G., Markey, N., and Shirmohammadi, M. (2014). Synchronizing words for weighted and timed automata. In Raman, V. and Suresh, S. P., editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, volume 29 of *LIPIcs*, pages 121–132. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Eppstein, 1990] Eppstein, D. (1990). Reset sequences for monotonic automata. *SIAM Journal on Computing*, 19(3):500–510.

[Fernau et al., 2020] Fernau, H., Wolf, P., and Yamakami, T. (2020). Synchronizing deterministic push-down automata can be really hard. In Esparza, J. and Král', D., editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPIcs*, pages 33:1–33:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Ginsburg, 1966] Ginsburg, S. (1966). *The mathematical Theory of Context-Free Languages*. McGraw-Hill.

[Ginsburg and Spanier, 1966] Ginsburg, S. and Spanier, E. H. (1966). Finite-turn push-down automata. *SIAM Journal on Control*, 4(3):429–453.

[Hahn et al., 2015] Hahn, M., Krebs, A., Lange, K., and Ludwig, M. (2015). Visibly counter languages and the structure of NC^1. In Italiano, G. F., Pighizzini, G., and Sannella, D., editors, *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015*, volume 9235 of *Lecture Notes in Computer Science*, pages 384–394. Springer.

[Krebs et al., 2015a] Krebs, A., Lange, K., and Ludwig, M. (2015a). On distinguishing NC^1 and NL. In Potapov, I., editor, *Developments in Language Theory - 19th International Conference, DLT 2015*, volume 9168 of *Lecture Notes in Computer Science*, pages 340–351. Springer.

[Krebs et al., 2015b] Krebs, A., Lange, K., and Ludwig, M. (2015b). Visibly counter languages and constant depth circuits. In Mayr, E. W. and Ollinger, N., editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015*, volume 30 of *LIPIcs*, pages 594–607. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Ludwig, 2019] Ludwig, M. (2019). *Tree-Structured Problems and Parallel Computation.* PhD thesis, University of Tübingen, Germany.

[Martyugin, 2014] Martyugin, P. (2014). Computational complexity of certain problems related to carefully synchronizing words for partial automata and directing words for nondeterministic automata. *Theory of Computing Systems*, 54(2):293–304.

[Martyugin, 2012] Martyugin, P. V. (2012). Synchronization of automata with one undefined or ambiguous transition. In Moreira, N. and Reis, R., editors, *Implementation and Application of Automata - 17th International Conference, CIAA 2012, Porto, Portugal, July 17-20, 2012. Proceedings*, volume 7381 of *Lecture Notes in Computer Science*, pages 278–288. Springer.

[Mehlhorn, 1980] Mehlhorn, K. (1980). Pebbling moutain ranges and its application of DCFL-recognition. In de Bakker, J. W. and van Leeuwen, J., editors, *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 422–435. Springer.

[Mikami and Yamakami, 2020] Mikami, E. and Yamakami, T. (2020). Synchronizing pushdown automata and reset words. An article appeared in Japanese as Technical Report of The Institute of Electonics, Information and Communication Engineers, COMP2019-54(2020-03), pp. 57–63.

[Nivat, 1968] Nivat, M. (1968). Transductions des langages de Chomsky. *Ann. Inst. Fourier, Grenoble*, 18:339–456.

[Okhotin and Salomaa, 2014] Okhotin, A. and Salomaa, K. (2014). Complexity of input-driven pushdown automata. *SIGACT News*, 45(2):47–67.

[Rystsov, 1980] Rystsov, I. K. (1980). On minimizing the length of synchronizing words for finite automata. In *Theory of Designing of Computing Systems*, pages 75–82. Institute of Cybernetics of Ukrainian Acad. Sci. (in Russian).

[Rystsov, 1983] Rystsov, I. K. (1983). Polynomial complete problems in automata theory. *Information Processing Letters*, 16(3):147–151.

[Sakarovitch, 2003] Sakarovitch, J. (2003). *Eléments de Théorie des Automates.* Vuibert informatique.

[Sandberg, 2005] Sandberg, S. (2005). Homing and synchronizing sequences. In Broy, M., Jonsson, B., Katoen, J., Leucker, M., and Pretschner, A., editors, *Model-Based Testing of Reactive Systems, Advanced Lectures*, volume 3472 of *LNCS*, pages 5–33. Springer.

[Savitch, 1970] Savitch, W. J. (1970). Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192.

[Shirmohammadi, 2014] Shirmohammadi, M. (2014). *Qualitative Analysis of Synchronizing Probabilistic Systems. (Analyse qualitative des systèmes probabilistes synchronisants)*. PhD thesis, École normale supérieure de Cachan, France.

[Shitov, 2019] Shitov, Y. (2019). An improvement to a recent upper bound for synchronizing words of finite automata. *Journal of Automata, Languages and Combinatorics*, 24(2-4):367–373.

[Srba, 2009] Srba, J. (2009). Beyond language equivalence on visibly pushdown automata. *Logical Methods in Computer Science*, 5(1).

[Starke, 1966] Starke, P. H. (1966). Eine Bemerkung über homogene Experimente. *Elektronische Informationsverarbeitung und Kybernetik (Journal of Information Processing and Cybernetics)*, 2(4):257–259.

[Starke, 2019] Starke, P. H. (2019). A remark about homogeneous experiments. *Journal of Automata, Languages and Combinatorics*, 24(2-4):133–137.

[Szykuła, 2018] Szykuła, M. (2018). Improving the upper bound on the length of the shortest reset word. In Niedermeier, R. and Vallée, B., editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPIcs*, pages 56:1–56:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Truthe and Volkov, 2019] Truthe, B. and Volkov, M. V. (2019). Journal of Automata, Languages and Combinatorics – Essays on the Černý Conjecture. `https://www.jalc.de/issues/2019/issue_24_2-4/content.html`. Accessed: 10/1/2020.

[Valiant, 1973] Valiant, L. G. (1973). *Decision Procedures for Families of Deterministic Pushdown Automata*. PhD thesis, University of Warwick, Coventry, UK.

[Volkov, 2008] Volkov, M. V. (2008). Synchronizing automata and the černý conjecture. In Martín-Vide, C., Otto, F., and Fernau, H., editors, *Language and Automata Theory and Applications, Second International Conference, LATA*, volume 5196 of *LNCS*, pages 11–27. Springer.

Chapter 11

On the Complexity of Intersection Non-Emptiness for Star-Free Language Classes

Emmanuel Arrighi, Henning Fernau, Stefan Hoffmann, Markus Holzer, Ismaël Jecker, Mateus de Oliveira Oliveira, and Petra Wolf.

An extended abstract appeared in the proceedings of FSTTCS 2021:
Leibniz International Proceedings in Informatics (LIPIcs) 213 (2021) pp. 34:1 – 34:15.
DOI: 10.4230/LIPIcs.FSTTCS.2021.34

This work started at the Schloss Dagstuhl Event 20483 *Moderne Aspekte der Komplexitätstheorie in der Automatentheorie* which was co-organized by the author of this thesis, see `https://www.dagstuhl.de/20483`.

On the Complexity of Intersection Non-Emptiness for Star-Free Language Classes

Emmanuel Arrighi*[1], Henning Fernau†[2], Stefan Hoffmann[2], Markus Holzer[3], Ismaël Jecker‡[4], Mateus de Oliveira Oliveira§[1], and Petra Wolf¶[2]

[1]University of Bergen, Norway
[2]Universität Trier, Germany
[3]Universität Gießen, Germany
[4]Institute of Science and Technology, Klosterneuburg, Austria

Abstract

In the Intersection Non-emptiness problem, we are given a list of finite automata $A_1, A_2, \ldots, A_m$ over a common alphabet Σ as input, and the goal is to determine whether some string $w \in \Sigma^*$ lies in the intersection of the languages accepted by the automata in the list. We analyze the complexity of the Intersection Non-emptiness problem under the promise that all input automata accept a language in some level of the dot-depth hierarchy, or some level of the Straubing-Thérien hierarchy. Automata accepting languages from the lowest levels of these hierarchies arise naturally in the context of model checking. We identify a dichotomy in the dot-depth hierarchy by showing that the problem is already NP-complete when all input automata accept languages of the levels $\mathcal{B}_0$ or $\mathcal{B}_{1/2}$ and already PSPACE-hard when all automata accept a language from the level $\mathcal{B}_1$. Conversely, we identify a tetrachotomy in the Straubing-Thérien hierarchy. More

*The author was supported by research Council of Norway (no. 274526), and part of a collaboration supported by IS-DAAD (no. 309319)
†The author was part of a collaboration supported by DAAD PPP (no. 57525246)
‡The author was supported by Marie Skłodowska-Curie Grant Agreement no. 754411
§The author was supported by research Council of Norway (no. 288761) and part of a collaboration supported by IS-DAAD (no. 309319)
¶The author was supported by DFG-funded project FE560/9-1 and part of a collaboration supported by DAAD PPP (no. 57525246)

precisely, we show that the problem is in AC^0 when restricted to level $\mathcal{L}_0$; complete for L or NL, depending on the input representation, when restricted to languages in the level $\mathcal{L}_{1/2}$; NP-complete when the input is given as DFAs accepting a language in $\mathcal{L}_1$ or $\mathcal{L}_{3/2}$; and finally, PSPACE-complete when the input automata accept languages in level $\mathcal{L}_2$ or higher. Moreover, we show that the proof technique used to show containment in NP for DFAs accepting languages in $\mathcal{L}_1$ or $\mathcal{L}_{3/2}$ does not generalize to the context of NFAs. To prove this, we identify a family of languages that provide an exponential separation between the state complexity of general NFAs and that of partially ordered NFAs. To the best of our knowledge, this is the first superpolynomial separation between these two models of computation.

1 Introduction

The Intersection Non-emptiness problem for finite automata is one of the most fundamental and well studied problems in the interplay between algorithms, complexity theory, and automata theory [Kozen, 1977,Kasai and Iwata, 1985,Lange and Rossmanith, 1992, Wareham, 2000, Karakostas et al., 2003, Wehar, 2014, Fernau and Krebs, 2017, Wehar, 2016]. Given a list $A_1, A_2, \ldots, A_m$ of finite automata over a common alphabet Σ, the goal is to determine whether there is a string $w \in \Sigma^*$ that is accepted by each of the automata in the list. This problem is PSPACE-complete when no restrictions are imposed [Kozen, 1977], and becomes NP-complete when the input automata accept unary languages (implicitly contained already in [Stockmeyer and Meyer, 1973] and studied in detail in [Morawietz et al., 2020]) or finite languages [Rampersad and Shallit, 2010].

In this work, we analyze the complexity of the Intersection Non-emptiness problem under the assumption that the languages accepted by the input automata belong to a given level of the Straubing-Thérien hierarchy [Place and Zeitoun, 2019, Straubing, 1981, Straubing, 1985, Thérien, 1981] or to some level of the Cohen-Brzozowski dot-depth hierarchy [Brzozowski, 1976, Cohen and Brzozowski, 1971, Place and Zeitoun, 2019]. Somehow, these languages are severely restricted, in the sense that both hierarchies, which are infinite, are entirely contained in the class of star-free languages, a class of languages that can be represented by expressions that use union, concatenation, and complementation, but *no* Kleene star operation [Brzozowski, 1976, Brzozowski and Knast, 1978, Place and Zeitoun, 2019]. Yet, languages belonging to fixed levels of either hierarchy may already be very difficult to characterize, in the sense that the very problem of deciding whether the language accepted by a given finite automaton belongs to a given full level or half-level k of either hierarchy is open, except for a few values of k [Almeida and Klíma, 2010, Glaßer and Schmitz, 2001, Glaßer and Schmitz,

2000, Place and Zeitoun, 2019]. It is worth noting that while the problem of determining whether a given automaton accepts a language in a certain level of either the dot-depth or of the Straubing-Thérien hierarchy is computationally hard (Theorem 1), automata accepting languages in lower levels of these hierarchies arise naturally in a variety of applications such as model checking where the Intersection Non-emptiness problem is of fundamental relevance [Abdulla, 2012, Bouajjani et al., 2000, Bouajjani et al., 2007].

An interesting question to consider is how the complexity of the Intersection Non-emptiness problem changes as we move up in the levels of the Straubing-Thérien hierarchy or in the levels of the dot-depth hierarchy. In particular, does the complexity of this problem changes gradually, as we increase the complexity of the input languages? In this work, we show that this is actually not the case, and that the complexity landscape for the Intersection Non-emptiness problem is already determined by the very first levels of either hierarchy (see Figure 1). Our first main result states that the Intersection Non-emptiness problem for NFAs and DFAs accepting languages from the level 1/2 of the Straubing-Thérien hierarchy are NL-complete and L-complete, respectively, under AC^0 reductions (Theorem 3). Additionally, this completeness result holds even in the case of unary languages. To prove hardness for NL and L, respectively, we will use a simple reduction from the reachability problem for DAGs and for directed trees, respectively. Nevertheless, the proof of containment in NL and in L, respectively, will require a new insight that may be of independent interest. More precisely, we will use a characterization of languages in the level 1/2 of the Straubing-Thérien hierarchy as shuffle ideals to show that the Intersection Non-emptiness problem can be reduced to concatenation non-emptiness (Lemma 2). This allows us to decide Intersection Non-emptiness by analyzing each finite automaton given at the input individually. It is worth mentioning that this result is optimal in the sense that the problem becomes NP-hard even if we allow a single DFA to accept a language from $\mathcal{L}_1$, and require all the others to accept languages from $\mathcal{L}_{1/2}$ (Theorem 5).

Subsequently, we analyze the complexity of Intersection Non-emptiness when all input automata are assumed to accept languages from one of the levels of $\mathcal{B}_0$ or $\mathcal{B}_{1/2}$ of the dot-depth hierarchy, or from the levels $\mathcal{L}_1$ or $\mathcal{L}_{3/2}$ of the Straubing-Thérien hierarchy. It is worth noting that NP-hardness follows straightforwardly from the fact that Intersection Non-emptiness for DFAs accepting finite languages is already NP-hard [Rampersad and Shallit, 2010]. Containment in NP, on the other hand, is a more delicate issue, and here the representation of the input automaton plays an important role. A characterization of languages in $\mathcal{L}_{3/2}$ in terms of languages accepted by partially ordered NFAs [Schwentick et al., 2001] is crucial for us, combined with the fact that Intersection Non-emptiness when the input is given by such automata is NP-complete [Masopust and Thomazo, 2015]. Intuitively, the proof in [Masopust and

Thomazo, 2015] follows by showing that the minimum length of a word in the intersection of languages in the level 3/2 of the Straubing-Thérien hierarchy is bounded by a polynomial on the sizes of the minimum partially ordered NFAs accepting these languages. To prove that Intersection Non-emptiness is in NP when the input automata are given as DFAs, we prove a new result establishing that the number of Myhill-Nerode equivalence classes in a language in the level $\mathcal{L}_{3/2}$ is at least as large as the number of states in a minimum partially ordered automaton representing the same language (Lemma 5).

Interestingly, we show that the proof technique used to prove this last result does not generalize to the context of NFAs. To prove this, we carefully design a sequence $(L_n)_{n\in\mathbb{N}_{\geq 1}}$ of languages over a binary alphabet such that for every $n \in \mathbb{N}_{\geq 1}$, the language L_n can be accepted by an NFA of size n, but any partially ordered NFA accepting L_n has size $2^{\Omega(\sqrt{n})}$. This lower bound is ensured by the fact that the syntactic monoid of L_n has many $\mathcal{J}$-factors. Our construction is inspired by a technique introduced by Klein and Zimmermann, in a completely different context, to prove lower bounds on the amount of look-ahead necessary to win infinite games with delay [Klein and Zimmermann, 2016]. To the best of our knowledge, this is the first exponential separation between the state complexity of general NFAs and that of partially ordered NFAs. While this result does not exclude the possibility that Intersection Non-emptiness for languages in $\mathcal{L}_{3/2}$ represented by general NFAs is in NP, it gives some indication that proving such a containment requires substantially new techniques.

Finally, we show that Intersection Non-emptiness for both DFAs and for NFAs is already PSPACE-complete if all accepting languages are from the level $\mathcal{B}_1$ of the dot-depth hierarchy or from the level $\mathcal{L}_2$ of the Straubing-Thérien hierarchy. We can adapt Kozen's classical PSPACE-completeness proof by using the complement of languages introduced in [Masopust and Krötzsch, 2021] in the study of partially ordered automata. Since the languages in [Masopust and Krötzsch, 2021] belong to $\mathcal{L}_{3/2}$, their complement belong to $\mathcal{L}_2$ (and to $\mathcal{B}_1$), and therefore, the proof follows.

Acknowledgment We like to thank Lukas Fleischer and Michael Wehar for our discussions. This work started at the Schloss Dagstuhl Event 20483 *Moderne Aspekte der Komplexitätstheorie in der Automatentheorie* `https://www.dagstuhl.de/20483`.

2 Preliminaries

We let $\mathbb{N}_{\geq k}$ denote the set of natural numbers greater or equal than k.

We assume the reader to be familiar with the basics in computational complexity theory [Papadimitriou, 1994]. In particular, we recall the inclusion chain: $\mathsf{AC}^0 \subset \mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE}$. Let AC^0 (NC^1, respectively) refer to the class of problems accepted by Turing machines with a bounded (unbounded, respectively) number of alternations in logarithmic time; alternatively one can define these classes by uniform Boolean circuits. Here, L (NL, respectively) refers to the class of problems that are accepted by deterministic (non-deterministic, respectively) Turing machines with logarithmic space, P (NP, respectively) denotes the class of problems solvable by deterministic (non-deterministic, respectively) Turing machines in polynomial time, and PSPACE refers to the class of languages accepted by deterministic or non-deterministic Turing machines in polynomial space [Savitch, 1970]. Completeness and hardness are always meant with respect to deterministic logspace many-one reductions unless otherwise stated. We will also consider the parameterized class XP of problems that can be solved in time $n^{f(k)}$, where n is the size of the input, k is a parameter, and f is a computable function [Flum and Grohe, 2006].

We mostly consider *non-deterministic finite automata* (NFAs). An NFA A is a tuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q is the finite *state set* with the *start state* $q_0 \in Q$, the *alphabet* Σ is a finite set of input symbols, and $F \subseteq Q$ is the *final state set*. The *transition function* $\delta : Q \times \Sigma \to 2^Q$ extends to words from Σ^* as usual. Here, 2^Q denotes the powerset of Q. By $L(A) = \{\, w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset \,\}$, we denote the *language accepted by* A. The NFA A is a *deterministic finite automaton* (DFA) if $|\delta(q, a)| = 1$ for every $q \in Q$ and $a \in \Sigma$. Then, we simply write $\delta(q, a) = p$ instead of $\delta(q, a) = \{p\}$. If $|\Sigma| = 1$, we call A a *unary* automaton.

We study Intersection Non-emptiness problems and their complexity. For finite automata, this problem is defined as follows:

- *Input*: Finite automata $A_i = (Q_i, \Sigma, \delta_i, q_{(0,i)}, F_i)$, for $1 \leq i \leq m$.
- *Question*: Is there a word w that is accepted by all A_i, i.e., is $\bigcap_{i=1}^{m} L(A_i) \neq \emptyset$?

Observe that the automata have a common input alphabet. Note that the complexity of the non-emptiness problem for finite automata of a certain type is a lower bound for the Intersection Non-emptiness for this particular type of automata. Throughout the paper we are mostly interested in the complexity of the Intersection Non-emptiness problem for finite state devices whose languages are contained in a particular language class.

We study the computational complexity of the intersection non-emptiness problem for languages from the classes of the Straubing-Thérien [Straubing, 1981, Thérien, 1981] and

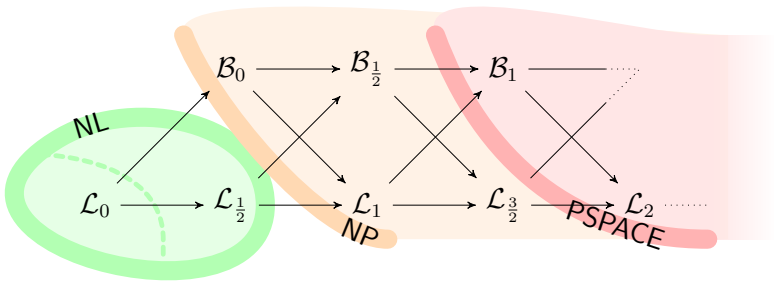


Figure 1: Straubing-Thérien and dot-depth hierarchies: the Intersection Non-emptiness status.

Cohen-Brzozowski's dot-depth hierarchy [Cohen and Brzozowski, 1971]. Both hierarchies are concatenation hierarchies that are defined by alternating the use of polynomial and Boolean closures. Let's be more specific. Let Σ be a finite alphabet. A language $L \subseteq \Sigma^*$ is a *marked product* of the languages $L_0, L_1, \ldots, L_k$, if $L = L_0 a_1 L_1 \cdots a_k L_k$, where the a_i's are letters. For a class of languages $\mathcal{M}$, the *polynomial closure* of $\mathcal{M}$ is the set of languages that are finite unions of marked product of languages from $\mathcal{M}$.

The concatenation hierarchy of basis $\mathcal{M}$ (a class of languages) is defined as follows (also refer to [Pin, 1998]): Level 0 is $\mathcal{M}$, i.e., $\mathcal{M}_0 = \mathcal{M}$ and, for each $n \geq 0$,

1. $\mathcal{M}_{n+1/2}$, that is, level $n + 1/2$, is the polynomial closure of level n and
2. $\mathcal{M}_{n+1}$, that is, level $n + 1$, is the Boolean closure of level $n + 1/2$.

The basis of the dot-depth hierarchy is the class of all finite and co-finite languages[1] and their classes are referred to as $\mathcal{B}_n$ ($\mathcal{B}_{n+1/2}$, respectively), while the basis of the Straubing-Thérien hierarchy is the class of languages that contains only the empty set and Σ^* and their classes are denoted by $\mathcal{L}_n$ ($\mathcal{L}_{n+1/2}$, respectively). Their inclusion relation is given by

$$\mathcal{B}_{n+1/2} \subseteq \mathcal{B}_{n+1} \subseteq \mathcal{B}_{n+3/2} \quad \text{and} \quad \mathcal{L}_{n+1/2} \subseteq \mathcal{L}_{n+1} \subseteq \mathcal{L}_{n+3/2},$$

for $n \geq 0$, and

$$\mathcal{L}_{n-1/2} \subseteq \mathcal{B}_{n-1/2} \subseteq \mathcal{L}_{n+1/2} \quad \text{and} \quad \mathcal{L}_n \subseteq \mathcal{B}_n \subseteq \mathcal{L}_{n+1},$$

for $n \geq 1$. In particular, $\mathcal{L}_0 \subseteq \mathcal{B}_0$, $\mathcal{B}_0 \subseteq \mathcal{B}_{1/2}$, and $\mathcal{L}_0 \subseteq \mathcal{L}_{1/2}$. Both hierarchies are infinite for alphabets of at least two letters and completely exhaust the class of star-free languages, which can be described by expressions that use union, concatenation, and complementation, but *no* Kleene star operation. For singleton letter alphabets, both hierarchies collapse to $\mathcal{B}_0$ and $\mathcal{L}_1$, respectively. Next, we describe the first few levels of each of these hierarchies:

[1]The dot-depth hierarchy, apart from level $\mathcal{B}_0$, coincides with the concatenation hierarchy starting with the language class $\{\emptyset, \{\lambda\}, \Sigma^+, \Sigma^*\}$.

Straubing-Thérien hierarchy: A language of Σ^* is of level 0 if and only if it is empty or equal to Σ^*. The languages of level $1/2$ are exactly those languages that are a finite (possibly empty) union of languages of the form $\Sigma^* a_1 \Sigma^* a_2 \cdots a_k \Sigma^*$, where the a_i's are letters from Σ. The languages of level 1 are finite Boolean combinations of languages of the form $\Sigma^* a_1 \Sigma^* a_2 \cdots a_k \Sigma^*$, where the a_i's are letters. These languages are also called *piecewise* testable languages. In particular, all finite and co-finite languages are of level 1. Finally, the languages of level $3/2$ of Σ^* are the finite unions of languages of the form $\Sigma_0^* a_1 \Sigma_1^* a_2 \cdots a_k \Sigma_k^*$, where the a_i's are letters from Σ and the Σ_i are subsets of Σ.

Dot-depth hierarchy: A language of Σ^* is of dot-depth (level) 0 if and only if it is finite or co-finite. The languages of dot-depth $1/2$ are exactly those languages that are a finite union of languages of the form $u_0 \Sigma^* u_1 \Sigma^* u_2 \cdots u_{k-1} \Sigma^* u_k$, where $k \geq 0$ and the u_i's are words from Σ^*. The languages of dot-depth 1 are finite Boolean combinations of languages of the form $u_0 \Sigma^* u_1 \Sigma^* u_2 \cdots u_{k-1} \Sigma^* u_k$, where $k \geq 0$ and the u_i's are words from Σ^*.

It is worth mentioning that in [Schwentick et al., 2001] it was shown that partially ordered NFAs (with multiple initial states) characterize the class $\mathcal{L}_{3/2}$, while partially ordered DFAs characterize the class of $\mathcal{R}$-trivial languages [Brzozowski and Fich, 1980], a class that is strictly in between $\mathcal{L}_1$ and $\mathcal{L}_{3/2}$. For an automaton A with input alphabet Σ, a state q is reachable from a state p, written $p \leq q$, if there is a word $w \in \Sigma^*$ such that $q \in \delta(p, w)$. An automaton is partially ordered if $\leq$ is a partial order. Partially ordered automata are sometimes also called acyclic or weakly acyclic automata. We refer to a partially ordered NFA (DFA, respectively) as poNFA (poDFA, respectively).

The fact that some of our results have a promise looks a bit technical, but the following result implies that we cannot get rid of this condition in general. To this end, we study, for a language class $\mathcal{L}$, the following question of $\mathcal{L}$-Membership.

- *Input*: A finite automaton A.
- *Question*: Is $L(A) \in \mathcal{L}$?

Theorem 1. *For each level $\mathcal{L}$ of the Straubing-Thérien or the dot-depth hierarchies, the $\mathcal{L}$-*Membership *problem for NFAs is* PSPACE*-hard, even when restricted to binary alphabets.*

Proof. For the PSPACE-hardness, note that each of the classes contains $\{0,1\}^*$ and is closed under quotients, since each class is a positive variety. As Non-universality

is PSPACE-hard for NFAs, we can apply Theorem 3.1.1 of [Hunt III and Rosenkrantz, 1978], first reducing regular expressions to NFAs. □

For some of the lower levels of the hierarchies, we also have containment in PSPACE, but in general, this is unknown, as it connects to the famous open problem if, for instance, $\mathcal{L}$-Membership is decidable for $\mathcal{L} = \mathcal{L}_3$; see [Masopust, 2018, Place and Zeitoun, 2019] for an overview on the decidability status of these questions. Checking for $\mathcal{L}_0$ up to $\mathcal{L}_2$ and $\mathcal{B}_0$ up to $\mathcal{B}_1$ containment for DFAs can be done in NL and is also complete for this class by ideas similar to the ones used in [Cho and Huynh, 1991].

3 Inside Logspace

A language of Σ^* belongs to level 0 of the Straubing-Thérien hierarchy if and only if it is empty or Σ^*. The Intersection Non-emptiness problem for language from this language family is not entirely trivial, because we have to check for emptiness. Since by our problem definition the property of a language being a member of level 0 is a promise, we can do the emptiness check within AC^0, since we only have to verify whether the empty word belongs to the language L specified by the automaton. In case $\varepsilon \in L$, then $L = \Sigma^*$; otherwise $L = \emptyset$. Since in the definition of finite state devices we do not allow for ε-transitions, we thus only have to check whether the initial state is also an accepting one. Therefore, we obtain:

Theorem 2. *The* Intersection Non-emptiness *problem for DFAs or NFAs accepting languages from $\mathcal{L}_0$ belongs to AC^0.*

For the languages of level $\mathcal{L}_{1/2}$ we find the following completeness result.

Theorem 3. *The* Intersection Non-emptiness *problem for NFAs accepting languages from $\mathcal{L}_{1/2}$ is* NL*-complete. Moreover, the problem remains* NL*-hard even if we restrict the input to NFAs over a unary alphabet. If the input instance contains only DFAs, the problem becomes* L*-complete (under weak reductions[2]).*

Hardness is shown by standard reductions from variants of graph accessibility [Hartmanis et al., 1978, Sudborough, 1975].

Lemma 1. *The* Intersection Non-emptiness *problem for NFAs over unary alphabet accepting languages from $\mathcal{L}_{1/2}$ is* NL*-hard. If the input instance contains only DFAs, the problem becomes* L*-hard under weak reductions.*

[2]Some form of AC^0 reducibility can be employed.

Proof. The NL-complete graph accessibility problem 2-GAP [Sudborough, 1975] is defined as follows: given a directed graph $G = (V, E)$ with outdegree (at most) two and two vertices s and t. Is there a path linking s and t in G? The problem remains NL-complete if the outdegree of every vertex of G is exactly two and if the graph is ordered, that is, if $(i, j) \in E$, then $i < j$ must be satisfied. The complexity of the reachability problem drops to L-completeness, if one considers the restriction that the outdegree is at most one. In this case the problem is referred to as 1-GAP [Hartmanis et al., 1978].

First we consider the Intersection Non-emptiness problem for NFAs. The NL-hardness is seen as follows: let $G = (V, E)$ and $s, t \in V$ be an ordered 2-GAP instance. Without loss of generality, we assume that $V = \{1, 2, \ldots, n\}$, the source vertex $s = 1$, and the target vertex $t = n$. From G we construct a unary NFA $A = (V, \{a\}, \delta, 1, n)$, where $\delta(i, a) = \{\, j \mid (i, j) \in E \,\} \cup \{i\}$. The 2-GAP instance has a solution if and only if the language accepted by A is non-empty. Moreover, by construction the automaton accepts a language of level 1/2, because (i) the NFA without a-self-loops is acyclic, since G is ordered, and thus does not contain any large cycles and (ii) all states do have self-loops.

Finally, we concentrate on the L-hardness of the Intersection Non-emptiness problem for DFAs. Here we use the 1-GAP variant to prove our result. Let $G = (V, E)$ and $s, t \in V$ be a 1-GAP instance, where we can assume that $V = \{1, 2, \ldots, n\}$, $s = 1$, and $t = n$. From G we construct a unary DFA $A = (V, \{a\}, \delta, 1, n)$ with $\delta(i, a) = j$, for $(i, j) \in E$ and $1 \leq i < n$, and $\delta(n, a) = n$. By construction the DFA A accepts either the empty language or a unary language where all words are at least of a certain length. In both cases $L(A)$ is a language from level 1/2 of the Straubing-Thérien hierarchy. Moreover, it is easy to see that there is a path in G linking s and t if and only if $L(A) \neq \emptyset$. □

It remains to show containment in logspace. To this end, we utilize an alternative characterization of the languages of level 1/2 of the Straubing-Thérien hierarchy as exactly those languages that are shuffle ideals. A language L is a *shuffle ideal* if, for every word $w \in L$ and $v \in \Sigma^*$, the set $w \sqcup\!\sqcup v$ is contained in L, where $w \sqcup\!\sqcup v := \{\, w_0 v_0 w_1 v_1 \ldots w_k v_k \mid w = w_0 w_1 \ldots w_k \text{ and } v = v_0 v_1 \ldots v_k \text{ with } w_i, v_i \in \Sigma^*, \text{ for } 0 \leq i \leq k \,\}$. The operation $\sqcup\!\sqcup$ naturally generalizes to sets. For the level $\mathcal{L}_{1/2}$, we find the following situation.

Lemma 2. *Let $m \geq 1$ and languages $L_i \subseteq \Sigma^*$, for $1 \leq i \leq m$, be shuffle ideals, i.e., they belong to $\mathcal{L}_{1/2}$. Then, $\bigcap_{i=1}^{m} L_i \neq \emptyset$ iff the shuffle ideal $L_1 L_2 \cdots L_m \neq \emptyset$ iff $L_i \neq \emptyset$ for every i with $1 \leq i \leq m$. Finally, $L_i \neq \emptyset$, for $1 \leq i \leq m$, iff $(a_1 a_2 \ldots a_k)^{\ell_i} \in L_i$, where $\Sigma = \{a_1, a_2, \ldots a_k\}$ and the shortest word in L_i is of length ℓ_i.*

Proof. The implication from left to right holds, because if $\bigcap_{i=1}^{m} L_i \neq \emptyset$, then there is a word w that belongs to all L_i, and hence the concatenation $L_1 L_2 \cdots L_m$ is nonempty, too. Since this argument has not used the prerequisite that the L_i's belong to the first half level of the Straubing-Thérien hierarchy, this implication does hold in general.

For the converse implication, recall that a language L of the first half level is a finite (possibly empty) union of languages of the form $\Sigma^* a_1 \Sigma^* a_2 \cdots a_k \Sigma^*$, where the a_i's are letters. Hence, whenever a word w belongs to L, any word of the form uwv with $u, v \in \Sigma^*$ is a member of L, too. Now assume that $L_1 L_2 \cdots L_m \neq \emptyset$, which can be witnessed by words $w_i \in L_i$, for $1 \leq i \leq m$. But then the word $w_1 w_2 \ldots w_m$ belongs to every L_i, by setting $u = w_1 w_2 \ldots w_{i-1}$ and $v = w_{i+1} w_{i+2} \ldots w_m$ and using the argument above. Therefore, the intersection of all L_i, i.e., the set $\bigcap_{i=1}^{m} L_i$, is nonempty, because of the word $w_1 w_2 \ldots w_m$.

The statement that $L_1 L_2 \cdots L_m$ is an ideal and that $L_1 L_2 \cdots L_m \neq \emptyset$ if and only if $L_i \neq \emptyset$, for every i with $1 \leq i \leq m$, is obvious.

For the last statement, assume $\Sigma = \{a_1, a_2 \ldots, a_k\}$. The implication from right to left is immediate, because if $(a_1 a_2 \ldots a_k)^{\ell_i} \in L_i$, for ℓ_i as specified above, then L_i is non-empty. Conversely, if L_i is non-empty, then there is a shortest word w of length ℓ_i that is contained in L_i. But then $(a_1 a_2 \ldots a_k)^{\ell_i}$ belongs to $w \sqcup\!\sqcup \Sigma^*$, which by assumption is a subset of the language L_i, since L_i is an ideal. Therefore, $L_i \neq \emptyset$ implies $(a_1 a_2 \ldots a_k)^{\ell_i} \in L_i$, which proves the stated claim. □

Now, we are ready to prove containment in logspace and thereby conclude the proof of Theorem 3.

Lemma 3. *The* Intersection Non-emptiness *problem for NFAs accepting languages from $\mathcal{L}_{1/2}$ belongs to* NL*. If the input instance contains only DFAs, the problem is in* L*.*

Proof. In order to solve the Intersection Non-emptiness problem for given finite automata $A_1, A_2, \ldots, A_m$ with a common input alphabet Σ, regardless of whether they are deterministic or non-deterministic, it suffices to check non-emptiness for all languages $L(A_i)$, for $1 \leq i \leq m$, in sequence, because of Lemma 2. To this end, membership of the words $(a_1 a_2 \ldots a_k)^{\ell_i}$ in L_i has to be tested, where ℓ_i is the length of the shortest word in L_i. Obviously, all ℓ_i are linearly bounded in the number of states of the appropriate finite automaton that accepts L_i. Hence, for NFAs as input instance, the test can be done on a non-deterministic logspace-bounded Turing machine, guessing the computations in the individual NFAs on the input word $(a_1 a_2 \ldots a_k)^{\ell_i}$. For DFAs as input instance, non-determinism is not needed, so that the procedure can be implemented on a deterministic Turing machine. □

4 NP-Completeness

In contrast to the Straubing-Thérien hierarchy, the Intersection Non-emptiness problem for languages from the dot-depth hierarchy is already NP-hard in the lowest level $\mathcal{B}_0$. More precisely, Intersection Non-emptiness for finite languages is NP-hard [Rampersad and Shallit, 2010, Theorem 1] and $\mathcal{B}_0$ already contains all finite languages. Hence, the Intersection Non-emptiness problem for languages from the Straubing-Thérien hierarchy of level $\mathcal{L}_1$ and above is NP-hard, too. For the levels $\mathcal{B}_0$, $\mathcal{B}_{1/2}$, $\mathcal{L}_1$, or $\mathcal{L}_{3/2}$, we give matching complexity upper bounds if the input are DFAs, yielding the first main result of this section proven in subsection 4.1.

Theorem 4. *The* Intersection Non-emptiness *problem for DFAs accepting languages from either $\mathcal{B}_0$, $\mathcal{B}_{1/2}$, $\mathcal{L}_1$, or $\mathcal{L}_{3/2}$ is* NP*-complete. The same holds for poNFAs instead of DFAs. The results hold even for a binary alphabet.*

For the level $\mathcal{L}_1$ of the Straubing-Thérien hierarchy, we obtain with the next main theorem a stronger result. Recall that if all input DFAs accept languages from $\mathcal{L}_{1/2}$, the Intersection Non-emptiness problem is L-complete due to Lemmata 1 and 3.

Theorem 5. *The* Intersection Non-emptiness *problem for DFAs is* NP*-complete even if only one DFA accepts a language from $\mathcal{L}_1$ and all other DFAs accept languages from $\mathcal{L}_{1/2}$ and the alphabet is binary.*

The proof of this theorem will be given in subsection 4.2.

For the level $\mathcal{B}_0$, we obtain a complete picture of the complexity of the Intersection Non-emptiness problem, independent of structural properties of the input finite automata, i.e., we show that here the problem is NP-complete for general NFAs.

For the level $\mathcal{L}_{3/2}$, if the input NFA are from the class of poNFA, which characterize level $\mathcal{L}_{3/2}$, then the Intersection Non-emptiness problem is known to be NP-complete [Masopust and Krötzsch, 2021]. Recall that $\mathcal{L}_{3/2}$ contains the levels $\mathcal{B}_{1/2}$, and $\mathcal{L}_1$ and hence also languages from these classes can be represented by poNFAs. But if the input automata are given as NFAs without any structural property, then the precise complexity of Intersection Non-emptiness for $\mathcal{B}_{1/2}$, $\mathcal{L}_1$, and $\mathcal{L}_{3/2}$ is an open problem and narrowed by NP-hardness and membership in PSPACE. We present a "No-Go-Theorem" by proving that for an NFA accepting a co-finite language, the smallest equivalent poNFA is exponentially larger in Subsection 4.3.

Theorem 6. *For every $n \in \mathbb{N}_{\geq 1}$, there exists a language $L_n \in \mathcal{B}_0$ on a binary alphabet such that L_n is recognized by an NFA of size $O(n^2)$, but the minimal poNFA recognizing L_n has more than 2^{n-1} states.*

While for NFAs the precise complexity for Intersection Non-emptiness of languages from $\mathcal{L}_1$ remains open, we can tackle this gap by narrowing the considered language class to *commutative* languages in level $\mathcal{L}_1$; recall that a language $L \subseteq \Sigma^*$ is *commutative* if, for any $a, b \in \Sigma$ and words $u, v \in \Sigma^*$, we have that $uabv \in L$ implies $ubav \in L$. We show that for DFAs, this restricted Intersection Non-emptiness problem remains NP-hard, in case the alphabet is unbounded. Concerning membership in NP, we show that even for NFAs, the Intersection Non-emptiness problem for *commutative* languages is contained in NP in general and in particular for commutative languages on each level. This generalizes the case of unary NFAs. Note that for commutative languages, the Straubing-Thérien hierarchy collapses at level $\mathcal{L}_{3/2}$. See Subsection 4.4 for the proofs.

Theorem 7. *The* Intersection Non-emptiness *problem*

- *is* NP*-hard for DFAs accepting* commutative *languages in* $\mathcal{L}_1$*, but*
- *is contained in* NP *for NFAs accepting* commutative *languages that might not be star-free.*

The proof of NP-hardness for commutative star-free languages in $\mathcal{L}_1$ requires an arbitrary alphabet. However, we show that Intersection Non-emptiness is contained in XP, with the size of the alphabet as the parameter, for specific forms of NFAs accepting commutative star-free languages, i.e., for fixed input alphabets, the Intersection Non-emptiness problem is solvable in polynomial time for this class of NFAs.

4.1 NP-Membership

Next, we focus on the NP-membership part of Theorem 4 and begin by proving that for $\mathcal{B}_0$, regardless of whether the input automata are NFAs or DFAs, the Intersection Non-emptiness problem is contained in NP and therefore NP-complete in combination with [Rampersad and Shallit, 2010].

Lemma 4. *The* Intersection Non-emptiness *problem for DFAs or NFAs all accepting languages from* $\mathcal{B}_0$ *is contained in* NP*.*

Proof. Let $A_1, A_2, \ldots, A_m$ be NFAs accepting languages from $\mathcal{B}_0$. If all NFAs accept co-finite languages, the intersection $\bigcap_{i=1}^{m} L(A_i)$ is non-empty. We can check deterministically if an NFA A_i, accepting a language from $\mathcal{B}_0$, is co-finite in polynomial time by using the promise of the language class, stating that $L(A_i)$ is either finite or co-finite. For that, we check whether A_i contains a cycle on an accepting path from the initial to some final state. If this is the case, we know that $L(A_i)$ is co-finite as it is not finite.

Otherwise, there is at least one NFA accepting a finite language, where the longest word is bounded by the number of states of this device. Hence, if $\bigcap_{i=1}^{m} L(A_i) \neq \emptyset$, there is a word w of length polynomial in the length of the input that witnesses this fact. Such a w can be non-deterministically guessed by a Turing machine checking membership of w in $L(A_i)$, for all NFAs A_i, in sequence. This shows containment in NP as desired. □

Notice that Masopust and Krötzsch have shown in [Masopust and Krötzsch, 2021] that Intersection Non-emptiness for poDFAs and for poNFAs is NP-complete. Also the unary case is discussed there, which can be solved in polynomial time. We cannot directly make use of these results, as we consider arbitrary NFAs or DFAs as inputs, only with the promise that they accept languages from a certain level of the studied hierarchies. In order to prove that for the levels $\mathcal{B}_0$, $\mathcal{B}_{1/2}$, $\mathcal{L}_1$, and $\mathcal{L}_{3/2}$, the Intersection Non-emptiness problem for DFAs is contained in NP, it is sufficient to prove the claim for $\mathcal{L}_{3/2}$ as all other stated levels are contained in $\mathcal{L}_{3/2}$. We prove the latter statement by obtaining a bound, polynomial in the size of the largest DFA, on the length of a shortest word accepted by all DFAs. Therefore, we show that for a minimal poNFA A, the size of an equivalent DFA is lower-bounded by the size of A and use a result of [Masopust and Krötzsch, 2021] for poNFAs. They have shown that given poNFAs $A_1, A_2, \ldots, A_m$, if the intersection of these automata is non-empty, then there exists a word of size at most $\sum_{i\in\{1,\ldots,m\}} d_i$, where d_i is the *depth* of A_i [Masopust and Krötzsch, 2021, Theorem 3.3]. Here, the depth of A_i is the length of the longest path (without self-loops) in the state graph of A_i. This result implies that the Intersection Non-emptiness problem for poNFAs accepting languages from $\mathcal{L}_{3/2}$ is contained in NP. We will further use this result to show that the Intersection Non-emptiness problem for DFAs accepting languages from $\mathcal{L}_{3/2}$ is NP-complete. First, we show that the number of states in a minimal poNFA is at most the number of classes in the Myhill-Nerode equivalence relation.

Lemma 5. *Let $A = (Q, \Sigma, \delta, q_0, F)$ be a minimal poNFA. Then, $L({}_{q_1}A) \neq L({}_{q_2}A)$ for all states $q_1, q_2 \in Q$, where ${}_qA$ is defined as $(Q, \Sigma, \delta, q, F)$.*

Proof. Let $A = (Q, \Sigma, \delta, q_0, F)$ be a minimal poNFA and $q_1, q_2 \in Q$ be two states. Suppose that $L({}_{q_1}A) = L({}_{q_2}A)$. We have two cases.

1. If q_1 and q_2 are pairwise not reachable from each other, then let $A' = (Q', \Sigma, \delta', q_0, F')$ be the NFA obtained from A, where q_1 and q_2 are merged into a new state $q_{1,2}$, so that $Q' = (Q \setminus \{q_1, q_2\}) \cup \{q_{1,2}\}$, $\delta'(q_{1,2}, a) = \delta(q_1, a) \cup \delta(q_2, a)$, for all $q \in Q'$, $q_{1,2} \in \delta'(q, a)$ if and only if $q_1 \in \delta(q, a)$ or $q_2 \in \delta(q, a)$, and $q_{1,2} \in F'$ if and only if $q_1 \in F$ or $q_2 \in F$. Automata A' is a partially ordered NFA. As q_1 and q_2 are not reachable one from the other, they are incomparable in the partial order relation

defined by A. Therefore, there is no state q such that $q_1 < q$ and $q < q_2$. One can check that $L(A') = L(A)$, which contradicts the minimality of A.

2. Otherwise, q_1 is reachable from q_2, or q_2 is reachable from q_1. Without loss of generality, we assume that q_2 is reachable from q_1. Let $A' = (Q', \Sigma, \delta', q_0, F')$ be the NFA obtained from A in two steps as described next. First, we remove all outgoing transitions from q_1 and then we merge q_1 and q_2 into a new state $q_{1,2}$ as done before. After removing all outgoing transitions from q_1, state q_2 is no longer reachable from q_1, therefore, as before, A' is a partially ordered NFA. Now we will prove that $L(A) = L(A')$.

   - Let $w \in L(A)$. Let ρ be an accepting run in A. If ρ does not contain q_1, then the run obtained by replacing every q_2 by $q_{1,2}$ is an accepting run in A'. If ρ contains q_1, then we split w into w_1 and w_2 such that $w = w_1 w_2$ and w_1 is the shortest prefix of w such that, after reading w_1, we reach q_1 in ρ. Because we merged q_1 and q_2 into $q_{1,2}$, we have that $q_{1,2} \in \delta'(q_0, w_1)$ in A'. Because $L({}_{q_1}A) = L({}_{q_2}A)$, we have that $L({}_{q_1}A) = L({}_{q_2}A) = L({}_{q_{1,2}}A')$ and therefore $\delta'(q_{1,2}, w_2) \cap F' \neq \emptyset$. So, w is accepted by A'.
   - Conversely, let $w \in L(A')$. Let ρ be an accepting run in A'. If ρ does not contain $q_{1,2}$, then the same run is accepting in A, too. If ρ contains $q_{1,2}$, we split w into w_1 and w_2 such that $w = w_1 w_2$, where w_1 is the shortest prefix of w such that, after reading w_1, we reach $q_{1,2}$ in ρ. Then, by definition of $q_{1,2}$, $\delta(q_0, w_1) \cap \{q_1, q_2\} \neq \emptyset$, and $\delta(q_1, w_2) \cap F \neq \emptyset$ iff $\delta(q_2, w_2) \cap F \neq \emptyset$ iff $\delta'(q_{1,2}, w_2) \neq \emptyset$. Therefore, $w \in L(A)$.

   This contradicts the minimality of A. □

Now, we can use the result from Masopust and Krötzsch to prove that the Intersection Non-emptiness problem for DFAs accepting languages in $\mathcal{L}_{3/2}$ is in NP.

Lemma 6. *The* Intersection Non-emptiness *problem for DFAs accepting languages from* $\mathcal{L}_{3/2}$ *belongs to* NP.

Proof. By Lemma 5, we have that the number of states in a minimal poNFA is at most the number of classes of the Myhill-Nerode equivalence relation. Hence, given a DFA accepting a language $L \in \mathcal{L}_{3/2}$, there exists a potentially smaller poNFA that recognizes L. By [Masopust and Krötzsch, 2021, Theorem 3.3], if the intersection is not empty, then there is a certificate of size polynomial in the sizes of the poNFAs . □

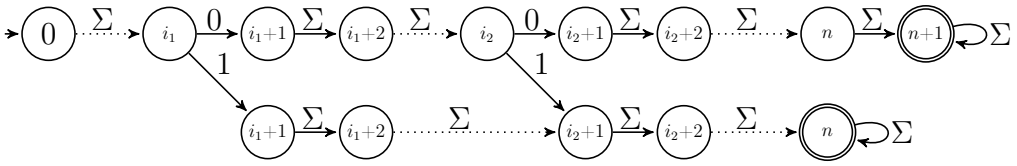


Figure 2: DFA A_{e_i} with $L(A_{e_i}) = \Sigma^{i_1} \cdot 1 \cdot \Sigma^{n-i_1-1} \cup \Sigma^{i_2} \cdot 1 \cdot \Sigma^{n-i_2-1} \cup \Sigma^{\geq n+1}$. A dotted arrow between some states j and j' represents a chain of length $j' - j$ with the same transition labels.

4.2 NP-Hardness

Recall that Intersection Non-emptiness for finite languages accepted by DFAs is already NP-complete by [Rampersad and Shallit, 2010, Theorem 1]. As the level $\mathcal{B}_0$ of the dot-depth hierarchy contains all finite language, the NP-hardness part of Theorem 4 follows directly from inclusion of language classes. Combining Lemma 6, and [Masopust and Krötzsch, 2021, Theorem 3.3] with the inclusion between levels in the Straubing-Thérien and the dot-depth hierarchy, we conclude the proof of Theorem 4.

Remark 1. Recall that the dot-depth hierarchy, apart form $\mathcal{B}_0$, coincides with the concatenation hierarchy starting with the language class $\{\emptyset, \{\lambda\}, \Sigma^+, \Sigma^*\}$. The Intersection Non-emptiness problem for DFAs or NFAs accepting only languages from $\{\emptyset, \{\lambda\}, \Sigma^+, \Sigma^*\}$ belongs to AC^0, by similar arguments as in the proof of Theorem 2.

We showed in Section 3 that Intersection Non-emptiness for DFAs, all accepting languages from $\mathcal{L}_{1/2}$, belongs to L. If we allow only one DFA to accept a language from $\mathcal{L}_1$, the problem becomes NP-hard. The statement also holds if the common alphabet is binary.

Theorem 5. *The* Intersection Non-emptiness *problem for DFAs is* NP*-complete even if only one DFA accepts a language from $\mathcal{L}_1$ and all other DFAs accept languages from $\mathcal{L}_{1/2}$ and the alphabet is binary.*

Proof sketch. The reduction is from Vertex Cover. Let $k \in \mathbb{N}_{\geq 0}$ and let $G = (V, E)$ be a graph with vertex set $V = \{v_0, v_1, \ldots, v_{n-1}\}$ and edge set $E = \{e_0, e_1, \ldots, e_{m-1}\}$. The only words $w = a_0 a_1 \ldots a_\ell$ accepted by all DFAs will be of length exactly $n = \ell + 1$ and encode a vertex cover by: v_j is in the vertex cover if and only if $a_j = 1$. Therefore, we construct for each edge $e_i = \{v_{i_1}, v_{i_2}\} \in E$, with $i_1 < i_2$, a DFA A_{e_i}, as depicted in Figure 2, that accepts the language $L(A_{e_i}) = \Sigma^{i_1} \cdot 1 \cdot \Sigma^{n-i_1-1} \cup \Sigma^{i_2} \cdot 1 \cdot \Sigma^{n-i_2-1} \cup \Sigma^{\geq n+1}$. We show that $L(A_{e_i})$ is from $\mathcal{L}_{1/2}$, as it also accepts all words of length at least $n+1$. We further construct a DFA $A_{=n,\leq k}$ that accepts all words of length exactly n that contain at most k letters 1. The finite language $L(A_{=n,\leq k})$ is the only language from $\mathcal{L}_1$ in the instance. □

Proof. The NP-membership follows from Lemma 6 by inclusion of language classes. For the hardness, we give a reduction from the VERTEX COVER problem: given an undirected graph $G = (V, E)$ with vertex set V and edge set $E \subseteq V \times V$ and integer k. Is there a subset $S \subseteq V$ with $|S| \leq k$ and for all $e \in E$, $S \cap e \neq \emptyset$? If yes, we call S a *vertex cover* of G of size at most k.

Let $k \in \mathbb{N}_{\geq 0}$ and let $G = (V, E)$ be an undirected graph with vertex set $V = \{v_0, v_1, \ldots, v_{n-1}\}$ and edge set $E = \{e_0, e_1, \ldots, e_{m-1}\}$. From (G, k) we construct $m + 1$ DFAs over the common alphabet $\Sigma = \{0, 1\}$. The input word for these automata will encode which vertices are in the vertex cover. Therefore, we assume a linear order on V indicated by the indices of the vertices. More precisely, a word accepted by all automata will have a 1 at position j if and only if the vertex v_j will be contained in the vertex cover S. For a word $w = a_0 a_1 \ldots a_\ell$ with $a_j \in \Sigma$ for $0 \leq j \leq \ell$ we denote $w[j] = a_j$. We may call a word w of length n a *vertex cover* and say that the vertex cover covers an edge $e = \{v_{j_1}, v_{j_2}\}$ if $w[j_1] = 1$ or $w[j_2] = 1$.

For every edge $e_i = \{v_{i_1}, v_{i_2}\}$ in E with $i_1 < i_2$, we construct a DFA A_{e_i} as depicted in Figure 2 consisting of two chains, one of length $n + 1$ and one of length $n - (i_1 + 1)$ (The length of a chain is the number of transitions in the chain). The DFA is defined as $A_{e_i} = (Q, \Sigma, \delta, q^0, F)$ with state set $Q = \{\, q^j \mid 0 \leq j \leq n + 1 \,\} \cup \{\, q'^j \mid i_1 + 1 \leq j \leq n \,\}$ and final states $F = \{\, q^{n+1}, q'^n \,\}$. We first focus on the states $\{\, q^j \mid 0 \leq j \leq n + 1 \,\}$. The idea is that there, the first $n + 1$ states correspond to the sequence of vertices and reading a 1 at position j for which $v_j \in e_i$ will cause the automaton to switch to the chain consisting of states $\{\, q'^j \mid i_1 + 1 \leq j \leq n \,\}$. There, only one state is accepting namely the state that we reach after reading a vertex cover of length exactly n that satisfies the edge e_i. Note that the paths from q_0 to q'^n are one transition shorten than the path from q_0 to q^{n+1}. To be more formal, we define $\delta(q^{i_1}, 1) = q'^{i_1+1}$ and $\delta(q^{i_2}, 1) = q'^{i_2+1}$. All other transitions are leading to the next state in the corresponding chain. Formally, we define $\delta(q^{i_1}, 0) = q^{i+1}$ and $\delta(q^{i_2}, 0) = q^{i_2+1}$, and for all $0 \leq j \leq n$ with $j \notin \{i_1, i_2\}$, we define $\delta(q^j, \sigma) = q^{j+1}$, for both $\sigma \in \Sigma$, and for all $i + 1 \leq j \leq n - 1$, we define $\delta(q'^j, \sigma) = q'^{j+1}$. We conclude the definition of δ by defining self-loops for the two accepting states, i.e., we define $\delta(q^{n+1}, \sigma) = q^{n+1}$ and $\delta(q'^n, \sigma) = q'^n$ for both $\sigma \in \Sigma$. Clearly, A_{e_i} is deterministic and of size $\mathcal{O}(n)$.

Note that the only words of length *exactly* n that are accepted by A_{e_i} contain a 1 at position i_1 or position i_2 and therefore cover the edge e_i. All other words accepted by A_{e_i} are of length at least $n + 1$. More precisely A_{e_i} accepts *all* words which are of size at least $n + 1$. Hence, we can describe the language accepted by A_{e_i} as

$$L(A_{e_i}) = \Sigma^{i_1} \cdot 1 \cdot \Sigma^{n-i_1-1} \cup \Sigma^{i_2} \cdot 1 \cdot \Sigma^{n-i_2-1} \cup \Sigma^{\geq n+1}.$$

Consider a word $w \in L(A_{e_i})$ of length n. W.l.o.g., assume $w[i_1] = 1$. If we insert into w one letter somewhere before or after position i_1, then the size of w increases by 1 and hence w falls into the subset $\Sigma^{\geq n+1}$ of $L(A_{e_i})$. Hence, we can rewrite the language $L(A_{e_i})$ by the following equivalent expression.

$$L(A_{e_i}) = \Sigma^{i_1} \cdot \Sigma^* \cdot 1 \cdot \Sigma^{n-i_1-1} \cdot \Sigma^* \cup \Sigma^{i_2} \cdot \Sigma^* \cdot 1 \cdot \Sigma^{n-i_2-1} \cdot \Sigma^* \cup \Sigma^{n+1}\Sigma^*.$$

As we can rewrite a language of the form $\Sigma^\ell\Sigma^*$ equivalently as a union of languages of the form $\Sigma^* w_1 \Sigma^* w_2 \dots w_\ell \Sigma^*$ for $w_i \in \Sigma$, for $1 \leq i \leq \ell$, it is clear that $L(A_{e_i})$ is a language of level $\mathcal{L}_{1/2}$.

Next, we define a DFA $A_{=n,\leq k}$ which accepts the finite language of all binary words of length n which contain at most k appearances of the letter 1. We define $A_{=n,\leq k} = (\{q_i^j \mid 0 \leq i \leq n+1, 0 \leq j \leq k+1\}, \Sigma, \delta, q_0^0, \{q_n^j \mid j \leq k\})$. The state graph of $A_{=n,\leq k}$ is a (n,k)-grid graph, where each letter increases the x dimension represented by the subscript i up to the value $n+1$, and each letter that is a 1 increases the y dimension represented by the superscript j up to the value $k+1$. More formally, we define $\delta(q_i^j, 0) = q_{i+1}^j$, and $\delta(q_i^j, 1) = q_{i+1}^{j+1}$ for $0 \leq i \leq n$ and $0 \leq j \leq k$; and $\delta(q_i^j, \sigma) = q_i^j$ for $i = n+1$ or $j = k+1$. The size of $A_{=n,\leq k}$ is bounded by $\mathcal{O}(nk)$. For readability, we defined $A_{=n,\leq k}$ as a non-minimal DFA. As $L(A_{=n,\leq k})$ is finite, it is of level $\mathcal{B}_0 \subseteq \mathcal{L}_1$.

By the arguments discussed above, the set of words accepted by all of the automata $(A_{e_i})_{e_i \in E}$ and $A_{=n,\leq k}$ are of size exactly n and encode a vertex cover for G of size at most k. □

4.3 Large Partially Ordered NFAs

The results obtained in the last subsection left open the precise complexity membership of Intersection Non-emptiness in the case of input automata being NFAs without any structural properties for the levels $\mathcal{B}_{1/2}$, $\mathcal{L}_1$, and $\mathcal{L}_{3/2}$. We devote this subsection to the proof of Theorem 6, showing that already for languages of $\mathcal{B}_0$ being accepted by an NFA, the size of an equivalent minimal poNFA can be exponential in the size of the NFA.

Theorem 6. *For every $n \in \mathbb{N}_{\geq 1}$, there exists a language $L_n \in \mathcal{B}_0$ on a binary alphabet such that L_n is recognized by an NFA of size $O(n^2)$, but the minimal poNFA recognizing L_n has more than 2^{n-1} states.*

Proof. While the statement requires languages over a binary alphabet, we begin by constructing an auxiliary family $(M_n)_{n \in \mathbb{N}_{\geq 1}}$ of languages over an unbounded alphabet.

For all $n \in \mathbb{N}_{\geq 1}$ we then define L_n by encoding M_n with a binary alphabet, and we prove three properties of these languages that directly imply the statement of the Theorem.

For every $n \in \mathbb{N}_{\geq 1}$, we define the languages M'_n and M''_n over the alphabet $\{1, 2, \ldots, n\}$ as follows. The language M'_n contains all the words of odd length, and M''_n contains all the words in which there are two occurrences of some letter $i \in \{1, 2, \ldots, n\}$ with only letters smaller than i appearing in between.[3] Formally,

$$M'_n = \{\, x \in \{1, 2, \ldots, n\}^* \mid |x| \text{ is odd} \,\},$$
$$M''_n = \{\, xiyiz \in \{1, 2, \ldots, n\}^* \mid i \in \{1, 2, \ldots, n\}, y \in \{1, 2, \ldots, i-1\}^* \,\}.$$

We then define M_n as the union $M'_n \cup M''_n$. Moreover, we define L_n by encoding M_n with the binary alphabet $\{a, b\}$: Let us consider the function $\phi_n : \{1, 2, \ldots, n\}^* \to \{a, b\}^*$ defined by $\phi(i_1 i_2 \ldots i_m) = a^{i_1} b^{n-i_1} a^{i_2} b^{n-i_2} \ldots a^{i_m} b^{n-i_m}$. We set $L_n \subseteq \{a, b\}^*$ as the union of $\phi_n(M_n)$ with the language $\{a, b\}^* \setminus \phi(\{1, 2, \ldots, n\}^*)$ containing all the words that are not a proper encoding of some word in $\{1, 2, \ldots, n\}^*$.

The statement of the theorem immediately follows from the following claims.

1. The languages M_n and L_n are cofinite, thus they are in $\mathcal{B}_0$.
2. The languages M_n and L_n are recognized by NFAs of size $n + 4$, resp. $O(n^2)$.
3. Every poNFA recognizing either M_n or L_n has a size greater than 2^{n-1}.

Proof of Item 1. We begin by proving that M_n is cofinite. Note that, by itself, the language M'_n is not in $\mathcal{B}_0$, as it is not even star-free. We show that M''_n is cofinite, which directly implies that $M_n = M'_n \cup M''_n$ is also cofinite. This follows from the fact that every word $u \in \{1, 2, \ldots, n\}^*$ satisfying $|u| \geq 2^n$ is in M''_n [Klein and Zimmermann, 2016]. This is easily proved by induction on n: If $n = 1$, we immediately get that $1^j \in M''_1$ for every $j \geq 2 = 2^1$: such a word contains two adjacent occurrences of 1. Now suppose that $n > 1$, and that the property holds for $n-1$. Every word $u \in \{1, 2, \ldots, n\}^*$ satisfying $|u| \geq 2^n$ can be split into two parts u_0, u_1 such that $|u_0|, |u_1| \geq 2^{n-1}$. We consider two possible cases, and prove that $u \in M''_n$ in both of them.

1. If either u_0 or u_1 contains no occurrence of the letter n, then by the induction hypothesis, either $u_0 \in M''_{n-1}$ or $u_1 \in M''_{n-1}$, which directly implies that $u \in M''_n$.

[3] The languages $(M''_n)_{n \in \mathbb{N}_{\geq 1}}$ were previously studied in [Klein and Zimmermann, 2016] with a game-theoretic background. We also refer to [Naylor, 2011] for similar "fractal languages."

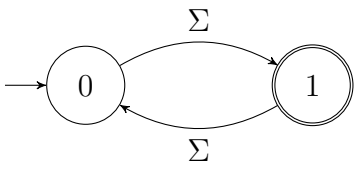


Figure 3: Automaton A' recognizing M'_n.

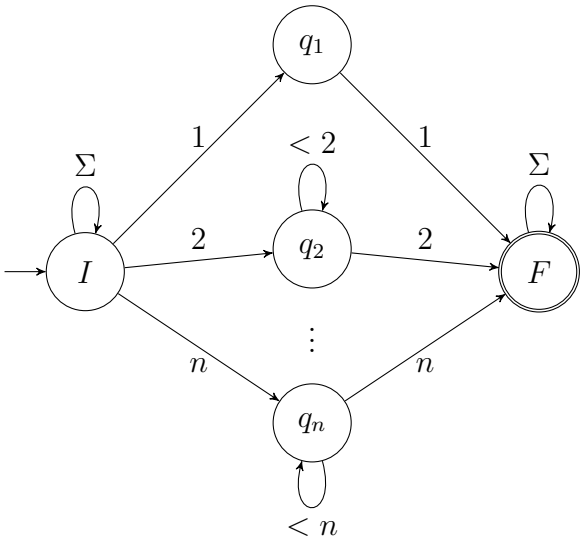


Figure 4: Automaton A'' recognizing M''_n.

2. If both u_0 and u_1 contain (at least) one occurrence of the letter n, then $u \in M''_n$ since it contains two occurrences of the letter n with only letters smaller than n appearing in between (the latter part trivially holds, as n is the largest letter).

Finally, we also get that L_n is cofinite: for all $u \in \{a, b\}^*$ satisfying $|u| \geq 2^n \cdot n$, either u is not a proper encoding of a word of $\{1, 2, \ldots, n\}^*$, thus $u \in L_n$, or u encodes a word $v \in \{1, 2, \ldots, n\}^*$ satisfying $|v| \geq 2^n$, hence $v \in M_n$, which again implies that $u \in L_n$. $\lhd$

Proof of Item 2. We first construct an NFA A of size $n + 4$ recognizing $M_n = M'_n \cup M''_n$ as the disjoint union of an NFA A' (Figure 3) of size 2 recognizing M'_n and an NFA A'' (Figure 4) of size $n + 2$ recognizing M''_n. The language M'_n of words of odd length is trivially recognized by an NFA of size 2, thus we only need to build an NFA $A'' = (Q, \{1, 2, \ldots, n\}, \delta, q_I, \{q_F\})$ of size $n + 2$ that recognizes M''_n. The state space Q is composed of the start state q_I, the single final state q_F, and n intermediate states $\{q_1, q_2, \ldots, q_n\}$. The NFA A'' behaves in three phases:

1. First, A'' loops over its start state until it non-deterministically guesses that it will read two copies of some $i \in \Sigma$ with smaller letters in between: $\delta(q_I, i) = \{q_I, q_i\}$ for all $i \in \Sigma$.

2. To check its guess, A'' loops in q_i while reading letters smaller than i until it reads a second i: $\delta(q_i, j) = \{q_i\}$ for all $j \in \{1, 2, \ldots, i-1\}$ and $\delta(q_i, i) = \{q_F\}$.

3. The final state q_F is an accepting sink: $\delta(q_F, j) = \{q_F\}$ for all $j \in \Sigma$.

This definition guarantees that A'' accepts the language M''_n.

Finally, we build an NFA B of size $O(n^2)$ that recognizes L_n by following similar ideas. Once again, B is defined as the disjoint union of two NFAs B' and B'': The NFA B' uses $4n$ states to check that either the input is *not* a proper encoding, or the input encodes a word $u \in \{1, 2, \ldots, n\}^*$ of odd length. Then, the NFA B'' with $O(n^2)$ states is obtained by adapting the NFA A'' to the encoding of the letters $\{1, 2, \ldots, n\}$: we split each of the $2n$ intermediate transitions of A'' into n parts by adding $n-1$ states, and we add $2(n-1)$ states to each self-loop of A'' in order to check that the encoding of an adequate letter is read. ◁

Proof of Item 3. It is sufficient to prove the result for M_n, as we can transform each poNFA $A = (Q, \{a, b\}, \delta_A, q_I, F)$ recognizing L_n into a poNFA $B = (Q, \{1, 2, \ldots, n\}, \delta_B, q_I, F)$ recognizing M_n with the same set of states by setting $\delta_B(q, i) = \delta_A(q, a^i b^{n-i})$.

Note that, by itself, the language M''_n is recognized by the poNFA A of size $n+2$ defined in the proof of Item 2. Let A' be a poNFA recognizing M_n. To show that A' has more than 2^{n-1} states, we study its behavior on the *Zimin words*, defined as follows:

$$\text{Let } u_1 = 1 \text{ and } u_j = u_{j-1} j u_{j-1} \text{ for all } 1 < j \leq n.$$

For instance, $u_4 = 121312141213121$. It is known that $|u_j| = 2^j - 1$ and $u_j \notin M''_n$ for every $1 \leq j \leq n$ [Klein and Zimmermann, 2016]. These two properties are easily proved by induction on j: Trivially, u_1 is not in M''_1 and its size is $1 = 2^1 - 1$. Now suppose that $j > 1$ and that u_{j-1} satisfies both properties: $|u_{j-1}| = 2^{j-1} - 1$ and $u_{j-1} \notin M''_n$. The first property follows immediately from the induction hypothesis.

$$|u_j| = |u_{j-1} j u_{j-1}| = 2 \cdot |u_{j-1}| + 1 = 2 \cdot (2^{j-1} - 1) + 1 = 2^j - 1;$$

To prove the induction step for the second property, we suppose, towards building a contradiction, that $u_j \in M''_n$. Then u_j contains two occurrences of some letter $i \in \{1, 2, \ldots, n\}$ with only letters smaller than i appearing in between. Since u_j contains only one occurrence of the letter j and no letter is greater than j, i is strictly smaller than j. Moreover, as only letters smaller than i (thus no j) can appear between these two occurrences, they both need to appear in one of the copies of u_{j-1}. Therefore u_{j-1}

is also in M_n'', which contradicts the induction hypothesis.

To conclude, remark that the word u_n is not in M_n'', but since $|u_n| = 2^n - 1$ is odd, it is in $M_n = L(A')$. Consider a sequence $\rho \in Q^*$ of states leading A' from its start state to a final state over the input u_n. Observe that the word u_n contains 2^{n-1} occurrences of the letter 1, and deleting (any) one of these occurrences results in a word of even length that is still not in M_n'', thus it is also not in $M_n = L(A')$. This proves that the sequence ρ cannot loop over any of the 1's in u_n. Moreover, as A' is partially ordered by assumption, once it leaves a state, it can never return to it. Therefore, ρ contains at least $2^{n-1} + 1$ distinct states while processing the 2^{n-1} occurrences of 1 in u_n, which shows that the automaton A' has more than 2^{n-1} many states. $\triangleleft$ $\square$

4.4 Commutative Star-Free Languages

In the case of commutative languages, we have a complete picture of the complexities for both hierarchies, even for arbitrary input NFAs. Observe, that commutative languages generalize unary languages, where it is known that for unary star-free languages both hierarchies collapse. For commutative star-free languages, a similar result holds, employing [Hoffmann, 2021, Prop. 28].

Theorem 8. *For* commutative star-free languages *the levels $\mathcal{L}_n$ of the Straubing-Thérien and $\mathcal{B}_n$ of the dot-depth hierarchy coincide for all full and half levels, except for $\mathcal{L}_0$ and $\mathcal{B}_0$. Moreover, the hierarchy collapses at level one.*

Proof. The strict inclusion $\mathcal{L}_0 \subset \mathcal{B}_0$ even in the commutative case is obvious. Since $\mathcal{L}_{1/2} \subseteq \mathcal{B}_{1/2}$ we only need to show the converse inclusion in the case of commutative languages. For the sake of notational simplicity, we shall give the proof only in a special case. Observe that, by commutativity, if $\Sigma^* ab \Sigma^* \subseteq L$, then $\Sigma^* a \Sigma^* b \Sigma^* \subseteq L$; moreover, $\Sigma^* ab \Sigma^* \subseteq \Sigma^* a \Sigma^* b \Sigma^*$. Using this idea repeatedly for marked products, as they describe languages from $\mathcal{B}_{1/2}$, we can write them as equivalent polynomials used for defining languages from $\mathcal{L}_{1/2}$.

It remains to show that every commutative star-free language is contained in $\mathcal{L}_1$. As shown in [Hoffmann, 2021, Prop. 28], every star-free commutative language can be written as a finite union of languages of the form $L = \mathtt{perm}(u) \sqcup\!\sqcup \Gamma^*$ for some $u \in \Sigma^*$ and $\Gamma \subseteq \Sigma$. Here $\mathtt{perm}(u) = \{\, w \in \Sigma^* \mid |u|_a = |w|_a \text{ for every } a \in \Sigma \,\}$, where $|w|_a$ is equal to the number of occurrences of a in w. Since $\mathtt{perm}(u)$ is a finite language, clearly, language L is equal to the finite union of all $v \sqcup\!\sqcup \Gamma^*$ for $v \in \mathtt{perm}(u)$, and thus belongs to $\mathcal{L}_{3/2}$, since $\Gamma \subseteq \Sigma$.

Now, note that $v \sqcup\!\sqcup \Sigma^* = \Sigma^* v_1 \Sigma^* \cdots \Sigma^* v_{|v|} \Sigma^*$, where $v = v_1 \cdots v_{|u|}$ with $v_i \in \Sigma$, is in level $1/2$ of the Straubing-Thérien hierarchy. Further,

$$v \sqcup\!\sqcup \Gamma^* = (v \sqcup\!\sqcup \Sigma^*) \cap \overline{\bigcup_{a \in \Sigma \setminus \Gamma} \mathtt{perm}(va) \sqcup\!\sqcup \Sigma^*}.$$

Hence, we can conclude containment in $\mathcal{L}_1$. As we have shown earlier that for commutative languages $\mathcal{B}_{1/2} \subseteq \mathcal{L}_{1/2}$, we get $\mathcal{B}_1 \subseteq \mathcal{L}_1$ and hence L is also contained in $\mathcal{B}_1$, and both hierarchies collapse at level one for commutative languages. □

Next we will give the results, summarized in Theorem 7, for the case of the commutative (star-free) languages. The NP-hardness follows by a reduction from 3-CNF-SAT.

Lemma 7. *The* Intersection Non-emptiness *problem is* NP*-hard for DFAs accepting* commutative *languages in* $\mathcal{L}_1$.

Proof. The NP-complete 3-CNF-SAT problem is defined as follows: given a Boolean formula φ as a set of clauses $C = \{c_1, c_2, \ldots, c_m\}$ over a set of variables $V = \{x_1, x_2, \ldots, x_n\}$ such that $|c_i| \leq 3$ for $i \leq m$. Is there a variable assignment $\beta : V \to \{0, 1\}$ such that φ evaluates to true under β?

Let φ be a Boolean formula in 3-CNF with clause set $C = \{c_1, c_2, \ldots, c_m\}$ and variable set $V = \{x_1, x_2, \ldots, x_n\}$. Let $\Sigma = \{x_1, x_2, \ldots, x_n, \overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n\}$. It is straightforward to construct polynomial-size DFAs for the following languages from $\mathcal{L}_1$:

$$L_{c_i} = \bigcup_{x \in c_i} \Sigma^* x \Sigma^* \quad \text{and} \quad L_{x_j} = \Sigma^* \setminus (\Sigma^* x_j \Sigma^* \overline{x}_j \Sigma^* \cup \Sigma^* \overline{x}_j \Sigma^* x_j \Sigma^*),$$

where $1 \leq i \leq m$ and $1 \leq j \leq n$. Then, the intersection of all L_{c_i} and all L_{x_j} is non-empty if and only if the 3-CNF-SAT instance φ is satisfiable. □

The upper bound shown next also holds for arbitrary commutative languages.

Theorem 9. *The* Intersection Non-emptiness *problem for NFAs accepting arbitrary, i.e., not necessarily star-free,* commutative *languages is in* NP.

Proof. It was shown in [Stockmeyer and Meyer, 1973] that the problem Intersection Non-emptiness is NP-complete for unary NFAs as input. Fix some order $\Sigma = \{a_1, a_2, \ldots, a_r\}$ of the input alphabet. Let $A_1, A_2, \ldots, A_m$ be the NFAs accepting commutative languages with $A_i = (Q_i, \Sigma, \delta_i, q_{0,i}, F_i)$ for $1 \leq i \leq m$. Without loss of generality, we may assume that every F_i is a singleton set, namely $F_i = \{q_{f,i}\}$. For each $1 \leq i \leq m$ and $1 \leq j \leq r$, let $B_{i,j}$ be the automaton over the unary alphabet $\{a_j\}$

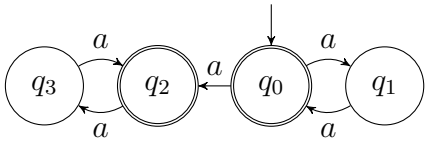


Figure 5: An example of a non-totally star-free NFA that accepts a star-free language.

obtained from A_i by deleting all transitions labeled with letters different from a_j and only retaining those labeled with a_j. Each $B_{i,j}$ will have one initial and one final state. Let $\vec{q_0} = (q_{0,1}, q_{0,2}, \ldots, q_{0,m})$ be the tuple of initial states of the NFAs; they are the initial states of $B_{1,1}, B_{2,1}, \ldots, B_{m,1}$, respectively. Then, non-deterministically guess further tuples $\vec{q_j}$ from $Q_1 \times Q_2 \times \ldots \times Q_m$ for $1 \leq j \leq r-1$. The jth tuple is considered as collecting the final states of the $B_{i,j}$ but also as the start states for the $B_{i,j+1}$. Finally, let $\vec{q_f} = (q_{f,1}, q_{f,2}, \ldots, q_{f,m})$ and consider this as the final states of $B_{1,r}, B_{2,r}, \ldots, B_{m,r}$. Then, for each $1 \leq j \leq r$ solve Intersection Non-emptiness for the unary automata $B_{1,j}, B_{2,j}, \ldots, B_{m,j}$. If there exist words w_j in the intersection of $L(B_{1,j}), L(B_{2,j}), \ldots, L(B_{m,j})$, for each $1 \leq j \leq r$, then, by commutativity, there exists one in $a_1^* a_2^* \cdots a_r^*$, namely, $w_1 w_2 \cdots w_m$, and so the above procedure finds it. Conversely, if the above procedure finds a word, this is contained in the intersection of the languages induced by the A_i's. □

For fixed alphabets, we have a polynomial-time algorithm, showing that the Intersection Non-emptiness problem, with the alphabet size as a parameter, is in XP, for totally star-free NFAs accepting star-free commutative languages. We say that an NFA $A = (Q, \Sigma, \delta, q_0, F)$ is *totally star-free*, if the language accepted by ${}_qA_p = (Q, \Sigma, \delta, q, \{p\})$ is star-free for any states $q, p \in Q$. For instance, poNFAs are totally star-free.

An example of a non-totally star-free NFA accepting a star-free language is given next. Consider the following NFA $A = (\{q_0, q_1, q_2, q_3\}, \delta, q_0, \{q_0, q_2\})$ with $\delta(q_0, a) = \{q_1, q_2\}$, $\delta(q_1, a) = \{q_0\}$, $\delta(q_2, a) = \{q_3\}$, and $\delta(q_3, a) = \{q_2\}$ that accepts the language $\{a\}^*$. The automaton is depicted in Figure 5. Yet, neither $L({}_{q_0}A_{q_0}) = \{aa\}^*$ nor $L({}_{q_0}A_{q_2}) = \{a\}\{aa\}^* \cup \{\varepsilon\}$ are star-free.

The proof of the following theorem uses classical results of Chrobak and Schützenberger [Chrobak, 1986, Schützenberger, 1965].

Theorem 10. *The* Intersection Non-emptiness *problem for totally star-free NFAs accepting* star-free commutative *languages, i.e., commutative languages in* $\mathcal{L}_1$*, is contained in* XP *with the size of the alphabet as the parameter.*

The proof of Theorem 10 is based on a combinatorial result that might be of independent interest.

Lemma 8. *Let $n \geq 1$ and $t_i, p_i \in \mathbb{N}_{\geq 0}$ for $1 \leq i \leq n$. Set $X = \bigcup_{i=1}^{n} (t_i + \mathbb{N}_{\geq 0} \cdot p_i)$, where $\mathbb{N}_{\geq 0} \cdot p_i = \{ x \cdot p_i \mid x \in \mathbb{N}_{\geq 0} \}$. If there exists a threshold $T \geq 0$ such that $\mathbb{N}_{\geq T} \subseteq X$, then already for $T_{\max} = \max\{ t_i \mid 1 \leq i \leq n \}$, we find $\mathbb{N}_{\geq T_{\max}} \subseteq X$.*

Proof. The assumption basically says that every integer y greater than $T-1$ is congruent to t_ℓ modulo p_ℓ for some $1 \leq \ell \leq n$. More specifically, if x is an arbitrary number with $x \geq T_{\max}$, then $y = x + T \cdot \operatorname{lcm}\{p_1, p_2, \ldots, p_n\}$ is congruent to t_ℓ modulo p_ℓ for some $1 \leq \ell \leq n$. But this implies that x itself is congruent to t_ℓ modulo p_ℓ, and so, as $x \geq t_\ell$, we can write $x = t_\ell + k_\ell \cdot p_\ell$ for some $k_\ell \geq 0$, i.e., $x \in X$. □

This result can be used to prove a polynomial bound for star-free unary languages on an equivalence resembling Schützenberger's characterization of star-freeness [Schützenberger, 1965].

Lemma 9. *Let L be a unary star-free language specified by an NFA A with n states. Then, there is a number N of order $\mathcal{O}(n^2)$ such that $a^N \in L$ if and only if for all $k \in \mathbb{N}_{\geq 0}$, $a^{N+k} \in L$.*

Proof. By a classical result of Chrobak [Chrobak, 1986], the given NFA A on n states can be transformed into a normal form where we have an initial tail with length at most $\mathcal{O}(n^2)$ that branches at a common endpoint into several cycles, where every cycle is of size at most n, see [Chrobak, 1986, Lemma 4.3]. Moreover, this transformation can be performed in polynomial time [Gawrychowski, 2011]. Note that a unary star-free language is either finite or co-finite [Brzozowski, 1976]. If L is finite, then there are no final states on the cycles and we can set N to be equal to the length of the tail, plus one. Otherwise, if $L \subseteq \{a\}^*$ is co-finite, then it can be expressed as a union of a finite language corresponding to the final states on the tail and finitely many languages of the form $\{ a^\ell \mid \ell \in (t + \mathbb{N}_{\geq 0} \cdot p) \}$, where the numbers t and p are induced by the Chrobak normal form. Then we can apply Lemma 8, where the set X is built from the t's and p's, and where the t's are bounded by $T_{\max}$, the sum of the longest tail and the largest cycle, plus one. Note that $T_{\max}$ is in $\mathcal{O}(n^2)$ and that the threshold from Lemma 8 guarantees that every word a^ℓ with $\ell \geq T_{\max}$ is a member of L, as desired. □

Theorem 10. *The* Intersection Non-emptiness *problem for totally star-free NFAs accepting* star-free commutative *languages, i.e., commutative languages in $\mathcal{L}_1$, is contained in* XP *with the size of the alphabet as the parameter.*

Proof. Let $A_i = (Q_i, \Sigma, \delta_i, q_i, F_i)$, for $i \in \{1, 2, \ldots, m\}$, be totally star-free NFAs accepting commutative languages. Let $n_i = |Q_i|$ be the number of states of A_i. Fix some order $\Sigma = \{a_1, a_2, \ldots, a_r\}$.

For $1 \leq i \leq m$ and $1 \leq j \leq r$, as well as $q, p \in Q_i$, let the automaton $B_{i,j,q,p} = (Q_i, \{a_j\}, \delta_i, q, \{p\})$ be obtained from A_i by deleting all transitions not labeled with the letter a_j and only retaining those labeled with a_j. Further, let $A_{i,q,p}$ be obtained from A_i by taking q as (new) initial state and p as the new (and only) final state. As A_i is totally star-free, $L(A_{i,q,p})$ is also star-free. By Schützenberger's Theorem characterizing star-freeness [Schützenberger, 1965], it is immediate that $L(A_{i,q,p}) \cap \Gamma^*$ is also star-free for each $\Gamma \subseteq \Sigma$. In particular, $L(B_{i,j,q,p}) = L(A_{i,q,p}) \cap \{a_j\}^*$ is star-free and commutative.

Recall that $\texttt{perm}(u) = \{\, w \in \Sigma^* \mid |u|_a = |w|_a \text{ for every } a \in \Sigma \,\}$, where $|w|_a$ is equal to the number of letters a in w. Moreover, $\texttt{perm}(L) = \bigcup_{v \in L} \texttt{perm}(v)$. By commutativity, the following property is clear:

$$L(A_i) = \texttt{perm}\left(\bigcup_{p_1,p_2,\ldots,p_{r-1} \in Q_i} \bigcup_{p_r \in F_i} L(B_{i,1,q_i,p_1}) \cdot L(B_{i,2,p_1,p_2}) \cdots L(B_{i,r,p_{r-1},p_r}) \right).$$

As A_i accepts a commutative language, by ordering the letters, we find that $w \in L(A_i)$ if and only if $a_1^{\ell_1} a_2^{\ell_2} \cdots a_r^{\ell_r} \in L(A_i)$, for ℓ_j being the number of occurrences of a_j in w, with $1 \leq j \leq r$. Furthermore, the word $a_1^{\ell_1} a_2^{\ell_2} \cdots a_r^{\ell_r}$ is in $L(A_i)$ if and only if for all j with $1 \leq j \leq r$, there is a state $p_j \in Q_i$ such that $a_j^{\ell_j} \in L(B_{i,j,p_{j-1},p_j})$, where $p_0 = q_i$ and $p_r \in F_i$. We can apply Lemma 9 to get constants $N_{i,1}, N_{i,2}, \ldots, N_{i,r} \in \mathcal{O}(n_i^2)$ such that checking membership of $a_j^{\ell_j}$ in $L(B_{i,j,p_{j-1},p_j})$ can be restricted to checking membership for a word of length at most N_j. Now, we describe a polynomial-time procedure to solve Intersection Non-emptiness for fixed alphabets. Set $N_i = \max\{N_{i,1}, N_{i,2}, \ldots, N_{i,r}\}$ with the numbers $N_{i,j}$ from above. Then, we know that a word $a_1^{\ell_1} a_2^{\ell_2} \cdots a_r^{\ell_r}$ is accepted by an input automaton A_i if and only if the word $a_1^{\min\{\ell_1, N_i\}} a_2^{\min\{\ell_2, N_i\}} \cdots a_r^{\min\{\ell_r, N_i\}}$ is accepted by it. If we let $N = \max\{N_1, N_2, \ldots, N_r\}$, we only need to test the $(N+1)^r$ many words $a_1^{i_1} a_2^{i_2} \cdots a_r^{i_r}$ with $0 \leq i_j \leq N$ and $1 \leq j \leq r$ if we can find a word among them that is accepted by all automata A_i for $1 \leq i \leq m$. Altogether, ignoring polynomial factors, this leads to a running time of the form $\mathcal{O}^*(N^r)$. □

Remark 2. Note that Theorem 10 does not hold for arbitrary commutative languages concerning a fixed alphabet, since in the general case, the problem is NP-complete even for languages over a common unary alphabet [Stockmeyer and Meyer, 1973].

5 PSPACE-Completeness

Here, we prove that even when restricted to languages from $\mathcal{B}_1$ or $\mathcal{L}_2$, Intersection Non-emptiness is PSPACE-complete, as it is for unrestricted DFAs or NFAs. We will profit from the close relations of Intersection Non-emptiness to the Non-universality problem for NFAs: Given an NFA A with input alphabet Σ, decide if $L(A) \neq \Sigma^*$. Conversely, we can also observe that Non-universality for NFAs is PSPACE-complete for languages from $\mathcal{B}_1$.

Theorem 11. *The* Intersection Non-emptiness *problem for DFAs or NFAs accepting languages from $\mathcal{B}_1$ or $\mathcal{L}_2$ is* PSPACE*-complete, even for binary input alphabets.*

As $\mathcal{B}_1 \subseteq \mathcal{L}_2$, it is sufficient to show that the problem is PSPACE-hard for $\mathcal{B}_1$. While without paying attention to the size of the input alphabet, this result can be readily obtained by re-analyzing Kozen's original proof in [Kozen, 1977], the restriction to binary input alphabets needs some more care. We modify the proof of Theorem 3 in [Krötsch et al., 2017] that showed PSPACE-completeness for Non-universality for poNFAs (that characterize the level 3/2 of the Straubing-Thérien hierarchy). Also, it can be observed that the languages involved in the intersection are actually locally testable languages. The class of locally testable languages is a sub-class of $\mathcal{B}_1$ and consists of the Boolean closure of languages of the form $u\Sigma^*$, Σ^*v, and $\Sigma^*w\Sigma^*$ where u, v, w are words from Σ^*, see [Pin, 2017].

Corollary 1. *The* Intersection Non-emptiness *problem for DFAs or NFAs accepting locally testable languages is* PSPACE*-complete, even for binary input alphabets.*

Proof. To see our claims, we re-analyze the proof of Theorem 3 in [Krötsch et al., 2017] that shows PSPACE-completeness for the closely related Non-universality problem for NFAs. Similar to Kozen's original proof, this gives a reduction from the general word problem of deterministic polynomial-space bounded Turing Machines. In the proof of Theorem 3 in [Krötsch et al., 2017] that showed PSPACE-completeness for Non-universality for poNFAs (that characterize the level 3/2 of the Straubing-Thérien hierarchy), a polynomial number of binary languages L_i was constructed such that $\bigcup_i L_i \neq \{0,1\}^*$ if and only if the p-space-bounded Turing machine M, where p is some polynomial, accepts a word $x \in \{0,1\}^*$ using space $p(|x|)$. Observe that each of the languages L_i is a polynomial union of languages of the forms $E\{0,1\}^*$, $\{0,1\}^*E$, $\{0,1\}^*E\{0,1\}^*$, or E for finite binary languages E. This means that each L_i belongs to $\mathcal{B}_{1/2}$. Now, observe that $\bigcup_i L_i \neq \{0,1\}^*$ if and only if $\bigcap_i \overline{L_i} \neq \emptyset$. As $\overline{L_i} \in \mathcal{B}_1$ and each L_i (and hence its complement $\overline{L_i}$) can be described by a polynomial size DFA, the claims follow. □

By the proof of Theorem 3 in [Krötsch et al., 2017], also $\bigcup_i L_i$ belongs to $\mathcal{B}_1$, so that we can conclude:

Corollary 2. *The* NON-UNIVERSALITY *problem for NFAs accepting languages from* $\mathcal{B}_1$ *is* PSPACE*-complete, even for binary input alphabets.*

We now present all proof details, because the construction is somewhat subtle.

The proof is based on simulating a p-space-bounded Turing machine M. We are interested in simulating a run of M on a string x. Its configurations are encoded as words over an alphabet Δ, so that with the help of the enhanced alphabet $\Delta_\# = \Delta \cup \{\#\}$, runs of M can be encoded, with # serving as a separator between configurations. More precisely, if Σ_M is the input alphabet of M, Γ_M (containing a special *blank* symbol $\llcorner\!\lrcorner$) is the tape alphabet, and Q_M is the state alphabet, then transitions take the form $f_M : Q_M \times \Gamma_M \to Q_M \times \Gamma_M \times \{L, R\}$, where L, R indicate the movements of the head. For simplicity, define $\Delta = \Gamma_M \times (Q_M \cup \{\$\})$. A configuration $\gamma \in \Delta^+$ has then the specific properties that it contains exactly one symbol from $\Gamma_M \times Q_M$ and that it has length $p(|x|)$ always, i.e., we are possibly filling up a string that is too short by the blank symbol $\llcorner\!\lrcorner$. Configuration sequences of M, or runs for short, can be encoded by words from $\#(\Delta^+\#)^*$, or more precisely, from $L_{simple-run} = \#((\Gamma_M \times \{\$\})^*(\Gamma_M \times Q_M)(\Gamma_M \times \{\$\})^*\#)^*$. The latter language can be encoded by a 3-state DFA. However, we will not make use of this language in the following, as it does not fit in the level of the dot-depth hierarchy that we are aiming at.

Let $\Sigma = \{0, 1\}$ be the binary target alphabet. A letter $a \in \Delta_\#$ is first encoded by a binary word $\hat{a}$ of length $K = \lceil \log_2(|\Delta_\#|) \rceil$, but this is only an auxiliary encoding, used to define the block-encoding

$$enc(a) = 001\hat{a}[1]1\hat{a}[2]1 \cdots \hat{a}[K]1$$

of length $L = 2K + 3$, where $\hat{a}[1]$ is the first symbol of the word $\hat{a}$, and so on. This block-encoding is extended to words and sets of words as usual. In order to avoid some case distinctions, we assume that $|\Delta_\#|$ is a power of two, so that $enc(\Delta_\#) = 001\Sigma1\Sigma1 \cdots \Sigma1$. Hence, $\overline{enc}(\Delta_\#) = 00\Sigma^{L-2} \setminus enc(\Delta_\#) = 00\{a_1b_1a_2b_2 \cdots a_Kb_K \mid a_1a_2 \cdots a_K \in \Sigma^K \wedge b_1b_2 \cdots b_K \in 1^*0\Sigma^*\}$. Clearly, there are DFAs with $\mathcal{O}(L)$ many states accepting $enc(\Delta_\#)$ and $\overline{enc}(\Delta_\#)$ (†). In this proof, we will call DFAs with $\mathcal{O}(L \cdot p(|x|))$ many states *small*. Any encoded word $enc(w)$, with $w \in \Delta_\#^*$, contains the factor 00 only at positions (minus one) that are multiples of L, more precisely: $enc(w)[i] = enc(w)[i + 1] = 0$ if and only if $i - 1$ is divisable by L. This observation allows us to construct small DFAs for $\Sigma^* enc(\Delta_\#)c\Sigma^*$ (for $c \in \{1, 01\}$) and for $\Sigma^* \overline{enc}(\Delta_\#)\Sigma^*$, based on (†). As shown in the proof of Theorem 3 in [Krötsch et al., 2017], the language of words that are <u>not</u> encodings

over $\Delta_\#$ at all is the union of the following languages:

1. $(1 \cup 01)\Sigma^*$,
2. $\Sigma^* \, \overline{enc}(\Delta_\#)\Sigma^*$,
3. $\Sigma^* \, enc(\Delta_\#)(1 \cup 01)\Sigma^*$, and
4. $\Sigma^* 00(\bigcup_{i=1}^{L-3} \Sigma^i) = \{w \in \Sigma^* \mid \text{The factor 00 is in the last } L-1 \text{ positions}\}$.

Each of these languages can be accepted by small DFAs A_1, A_2, A_3, A_4.

Then, we have to take care of the binary words that cannot be encodings of configuration sequences, because the first configuration is not initial. By our construction, the (unique) initial configuration γ is encoded by a binary string $enc(\gamma)$ of length $L \cdot p(|x|)$, i.e., we consider a language L' which is the complement of $enc(\#\gamma\#)\Sigma^*$, the language of all binary strings that do not start with the encoding of the initial configuration. Let $\#\gamma\# = a_1 a_2 \cdots a_{p(|x|)+2}$. As we already described non-encodings by automata A_1 through A_4, instead of L', we describe $\bigcup_{j=0}^{p(|x|)+2} L'_j$, where L'_0 is the set of all words of which their length is not divisible by L and bounded by $L \cdot (p(|x|) + 2)$. Intuitively, L'_0 is the set of strings that are too short and further contains the empty word. We further define $L'_j = \Sigma^{(j-1)L} \, \overline{enc}(a_j)\Sigma^*$ for $j = 1$ to $p(|x|) + 2$, describing a violation at symbol a_j of the initial configuration γ. Moreover, there are small DFAs $A_5, A_6, \ldots, A_{p(|x|)+7}$ that accept $L'_0, L'_1, \ldots, L_{p(|x|)+2}$.

Assuming a unique final state and also assuming that M cleans up the tape after processing, there is a unique final configuration γ_f that should be reached. Then, invalidity of a computation with respect to the final configuration can be checked as for the initial configuration, giving us small DFAs $A_{p(|x|)+8}, A_{p(|x|)+9}, \ldots, A_{2p(|x|)+10}$.

Finally, we want to check the (complement of the) following property of a valid configuration sequence $\rho \in \#(\Delta^+\#)^*$: any sequence of three letters a, b, c in ρ determines the letter $f(a, b, c)$ that should be present at a distance of $p(|x|) - 1$ to the right. More precisely, we are interested in any factor $abcvdf(a, b, c)e$ of ρ where $|vd| = p(|x|) - 1$. Different scenarios can occur; we only describe three typical situations in the following.

- If $a = (a', \$)$, $b = (b', \$)$, $c = (c', \$)$, then $f(a, b, c) = b$. For d, we know $d \in \{a'\} \times (Q \cup \{\$\})$ and similarly for e, we know $e \in \{c'\} \times (Q \cup \{\$\})$.
- If $a = (a', \$)$, $b = \#$, $c = (c', \$)$, then $f(a, b, c) = \#$. For d, we know $d \in \{a'\} \times (Q \cup \{\$\})$ and similarly for e, we know $e \in \{c'\} \times (Q \cup \{\$\})$.

- If $a = (a', \$)$, $b = (b', q)$, $c = (c', \$)$ and if $f_M(q, b') = (p, \hat{b}', L)$, then $d = (a', p)$, $f(a, b, c) = (\hat{b}, \$)$, and $e = c$.

We refrain from describing all such situations in detail. Yet with some more sloppiness, we write $\overline{enc}(d(f(a, b, c)e)$ for all situations that do not obey the rules for $df(a, b, c)e$ as tentatively formulated before. Now, for each triple $a, b, c \in \Delta_{\#}$, consider the binary language $L_{a,b,c} = \Sigma^* \cdot enc(abc) \cdot \Sigma^{L \cdot (p(|x|)-1)} \cdot \overline{enc}(d(f(a, b, c)e) \cdot \Sigma^*$. This language can be accepted by a small DFA $A_{a,b,c}$.

Altogether, we described $2p(|x|) + 10 + (|\Delta_{\#}|)^3$ many languages from $\mathcal{B}_1$ such that their union does not yield Σ^* if and only if M accepts x using $p(|x|)$ space. Moreover, for each of the languages, we can build small DFAs.

6 Conclusion and Open Problems

We have investigated how the increase in complexity within the dot-depth and the Straubing-Thérien hierarchies is reflected in the complexity of the Intersection Non-emptiness problem. We have shown the complexity of this problem is already completely determined by the very first levels of either hierarchy.

Our work leaves open some very interesting questions and directions of research. First, we were not able to prove containment in NP for the Intersection Non-emptiness problem when the input automata are allowed to be NFAs accepting a language in the level 3/2 or in the level 1 of the Straubing-Thérien hierarchy. Interestingly, we have shown that such containment holds in the case of DFAs, but have shown that the technique we have used to prove this containment does not carry over to the context of NFAs. In particular, to show this we have provided the first exponential separation between the state complexity of general NFAs and partially ordered NFAs. The most immediate open question is if Intersection Non-emptiness for NFAs accepting languages in $\mathcal{B}_{1/2}$, $\mathcal{L}_1$, or $\mathcal{L}_{3/2}$ is complete for some level higher up in the polynomial-time hierarchy (PH), or if this case is already PSPACE-complete. Another tantalizing open question is whether one can capture the levels of PH in terms of the Intersection Non-emptiness problem when the input automata are assumed to accept languages belonging to levels of a sub-hierarchy of $\mathcal{L}_2$. Such sub-hierarchies have been considered for instance in [Klíma and Polák, 2011].

It would also be interesting to have a systematic study of these two well-known sub-regular hierarchies for related problems like Non-universality for NFAs or Union Non-universality for DFAs. Notice the technicality that Union Non-universality

(similar to Intersection Non-emptiness) has an implicit Boolean operation (now union instead of intersection) within the problem statement, while Non-universality lacks this implicit Boolean operation. This might lead to a small "shift" in the discussions of the hierarchy levels that involve Boolean closure. Another interesting hierarchy is the group hierarchy [Pin, 1998], where we start with the group languages, i.e., languages acceptable by automata in which every letter induces a permutation of the state set, at level 0. Note that for group languages, Intersection Non-emptiness is NP-complete even for a unary alphabet [Stockmeyer and Meyer, 1973]. As Σ^* is a group language, the Straubing-Thérien hierarchy is contained in the corresponding levels of the group hierarchy, and hence, we get PSPACE-hardness for level 2 and above in this hierarchy. However, we do not know what happens in the levels in between.

References

[Abdulla, 2012] Abdulla, P. A. (2012). Regular model checking. *International Journal on Software Tools for Technology Transfer*, 14(2):109–118.

[Almeida and Klíma, 2010] Almeida, J. and Klíma, O. (2010). New decidable upper bound of the second level in the Straubing-Thérien concatenation hierarchy of star-free languages. *Discrete Mathematics & Theoretical Computer Science*, 12(4):41–58.

[Bouajjani et al., 2000] Bouajjani, A., Jonsson, B., Nilsson, M., and Touili, T. (2000). Regular model checking. In Emerson, E. A. and Sistla, A. P., editors, *Computer Aided Verification, 12th International Conference, CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 403–418. Springer.

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

[Brzozowski, 1976] Brzozowski, J. A. (1976). Hierarchies of aperiodic languages. *RAIRO Informatique théorique et Applications/Theoretical Informatics and Applications*, 10(2):33–49.

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

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

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

[Chrobak, 1986] Chrobak, M. (1986). Finite automata and unary languages. *Theoretical Computer Science*, 47(3):149–158.

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

[Fernau and Krebs, 2017] Fernau, H. and Krebs, A. (2017). Problems on finite automata and the exponential time hypothesis. *Algorithms*, 10(1):24.

[Flum and Grohe, 2006] Flum, J. and Grohe, M. (2006). *Parameterized Complexity Theory*. Springer.

[Gawrychowski, 2011] Gawrychowski, P. (2011). Chrobak normal form revisited, with applications. In Bouchou-Markhoff, B., Caron, P., Champarnaud, J., and Maurel, D., editors, *Implementation and Application of Automata - 16th International Conference, CIAA*, volume 6807 of *Lecture Notes in Computer Science*, pages 142–153. Springer.

[Glaßer and Schmitz, 2000] Glaßer, C. and Schmitz, H. (2000). Decidable hierarchies of starfree languages. In Kapoor, S. and Prasad, S., editors, *Foundations of Software Technology and Theoretical Computer Science, 20th Conference, FST TCS*, volume 1974 of *Lecture Notes in Computer Science*, pages 503–515. Springer.

[Glaßer and Schmitz, 2001] Glaßer, C. and Schmitz, H. (2001). Level 5/2 of the Straubing-Thérien hierarchy for two-letter alphabets. In Kuich, W., Rozenberg, G., and Salomaa, A., editors, *Developments in Language Theory, 5th International Conference, DLT*, volume 2295 of *Lecture Notes in Computer Science*, pages 251–261. Springer.

[Hartmanis et al., 1978] Hartmanis, J., Immerman, N., and Mahaney, S. R. (1978). One-way log-tape reductions. In *19th Annual Symposium on Foundations of Computer Science, FOCS*, pages 65–72. IEEE Computer Society.

[Hoffmann, 2021] Hoffmann, S. (2021). Regularity conditions for iterated shuffle on commutative regular languages. In Maneth, S., editor, *Implementation and Application of Automata - 25th International Conference, CIAA 2021, Virtual Event, July 19-22, 2021, Proceedings*, volume 12803 of *Lecture Notes in Computer Science*, pages 27–38. Springer.

[Hunt III and Rosenkrantz, 1978] Hunt III, H. B. and Rosenkrantz, D. J. (1978). Computational parallels between the regular and context-free languages. *SIAM Journal on Computing*, 7(1):99–114.

[Karakostas et al., 2003] Karakostas, G., Lipton, R. J., and Viglas, A. (2003). On the complexity of intersecting finite state automata and NL versus NP. *Theoretical Computer Science*, 302(1):257–274.

[Kasai and Iwata, 1985] Kasai, T. and Iwata, S. (1985). Gradually intractable problems and nondeterministic log-space lower bounds. *Mathematical Systems Theory*, 18(1):153–170.

[Klein and Zimmermann, 2016] Klein, F. and Zimmermann, M. (2016). How much lookahead is needed to win infinite games? *Logical Methods in Computer Science*, 12(3).

[Klíma and Polák, 2011] Klíma, O. and Polák, L. (2011). Subhierarchies of the second level in the Straubing-Thérien hierarchy. *International Journal of Algebra and Computation*, 21(7):1195–1215.

[Kozen, 1977] Kozen, D. (1977). Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science, FOCS*, pages 254–266. IEEE Computer Society.

[Krötsch et al., 2017] Krötsch, M., Masopust, T., and Thomazo, M. (2017). Complexity of universality and related problems for partially ordered NFAs. *Information and Computation*, 255:177–192.

[Lange and Rossmanith, 1992] Lange, K. and Rossmanith, P. (1992). The emptiness problem for intersections of regular languages. In Havel, I. M. and Koubek, V., editors, *Mathematical Foundations of Computer Science 1992, 17th International Symposium, MFCS*, volume 629 of *Lecture Notes in Computer Science*, pages 346–354. Springer.

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

[Masopust and Krötzsch, 2021] Masopust, T. and Krötzsch, M. (2021). Partially ordered automata and piecewise testability. *Logical Methods in Computer Science*, 17(2).

[Masopust and Thomazo, 2015] Masopust, T. and Thomazo, M. (2015). On the complexity of k-piecewise testability and the depth of automata. In Potapov, I., editor, *Developments in Language Theory - 19th International Conference, DLT*, number 9168 in Lecture Notes in Computer Science, pages 364–376. Springer.

[Morawietz et al., 2020] Morawietz, N., Rehs, C., and Weller, M. (2020). A timecop's work is harder than you think. In Esparza, J. and Král', D., editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August*

24-28, 2020, Prague, Czech Republic, volume 170 of *LIPIcs*, pages 71:1–71:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Naylor, 2011] Naylor, M. (2011). Abacaba! – using a mathematical pattern to connect art, music, poetry and literature. *Bridges*, pages 89–96.

[Papadimitriou, 1994] Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley.

[Pin, 1998] Pin, J. (1998). Bridges for concatenation hierarchies. In Larsen, K. G., Skyum, S., and Winskel, G., editors, *Automata, Languages and Programming, 25th International Colloquium, ICALP*, volume 1443 of *Lecture Notes in Computer Science*, pages 431–442. Springer.

[Pin, 2017] Pin, J. (2017). The dot-depth hierarchy, 45 years later. In Konstantinidis, S., Moreira, N., Reis, R., and Shallit, J. O., editors, *The Role of Theory in Computer Science - Essays Dedicated to Janusz Brzozowski*, pages 177–202. World Scientific.

[Place and Zeitoun, 2019] Place, T. and Zeitoun, M. (2019). Generic results for concatenation hierarchies. *Theory of Computing Systems*, 63(4):849–901.

[Rampersad and Shallit, 2010] Rampersad, N. and Shallit, J. (2010). Detecting patterns in finite regular and context-free languages. *Information Processing Letters*, 110(3):108–112.

[Savitch, 1970] Savitch, W. J. (1970). Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192.

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

[Schwentick et al., 2001] Schwentick, T., Thérien, D., and Vollmer, H. (2001). Partially-ordered two-way automata: A new characterization of DA. In Kuich, W., Rozenberg, G., and Salomaa, A., editors, *Developments in Language Theory, 5th International Conference, DLT*, volume 2295 of *Lecture Notes in Computer Science*, pages 239–250. Springer.

[Stockmeyer and Meyer, 1973] Stockmeyer, L. J. and Meyer, A. R. (1973). Word problems requiring exponential time: Preliminary report. In Aho, A. V., Borodin, A., Constable, R. L., Floyd, R. W., Harrison, M. A., Karp, R. M., and Strong, H. R., editors, *5th Annual Symposium on Theory of Computing, STOC*, pages 1–9. ACM.

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

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

[Sudborough, 1975] Sudborough, I. H. (1975). On tape-bounded complexity classes and multihead finite automata. *Journal of Computer and System Sciences*, 10(1):62–76.

[Thérien, 1981] Thérien, D. (1981). Classification of finite monoids: the language approach. *Theoretical Computer Science*, 14(2):195–208.

[Wareham, 2000] Wareham, T. (2000). The parameterized complexity of intersection and composition operations on sets of finite-state automata. In Yu, S. and Paun, A., editors, *Implementation and Application of Automata, 5th International Conference, CIAA*, volume 2088 of *Lecture Notes in Computer Science*, pages 302–310. Springer.

[Wehar, 2014] Wehar, M. (2014). Hardness results for intersection non-emptiness. In Esparza, J., Fraigniaud, P., Husfeldt, T., and Koutsoupias, E., editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 354–362. Springer.

[Wehar, 2016] Wehar, M. (2016). *On the Complexity of Intersection Non-Emptiness Problems*. PhD thesis, University at Buffalo.

Chapter 12

Decomposing Permutation Automata

Ismaël Jecker, Nicolas Mazzocchi, and Petra Wolf.

An extended abstract appeared in the proceedings of CONCUR 2021:
Leibniz International Proceedings in Informatics (LIPIcs) 203 (2021) pp. 18:1 – 18:19.
DOI: 10.4230/LIPIcs.CONCUR.2021.18.

Decomposing Permutation Automata

Ismaël Jecker*[1], Nicolas Mazzocchi[2], and Petra Wolf†[3]

[1]Institute of Science and Technology, Klosterneuburg, Austria
[2]IMDEA Software Institute, Madrid, Spain
[3]Universität Trier, Germany

Abstract

A deterministic finite automaton (DFA) $\mathcal{A}$ is composite if its language $L(\mathcal{A})$ can be decomposed into an intersection $\bigcap_{i=1}^{k} L(\mathcal{A}_i)$ of languages of smaller DFAs. Otherwise, $\mathcal{A}$ is prime. This notion of primality was introduced by Kupferman and Mosheiff in 2013, and while they proved that we can decide whether a DFA is composite, the precise complexity of this problem is still open, with a doubly-exponential gap between the upper and lower bounds. In this work, we focus on permutation DFAs, i.e., those for which the transition monoid is a group. We provide an NP algorithm to decide whether a permutation DFA is composite, and show that the difficulty of this problem comes from the number of non-accepting states of the instance: we give a fixed-parameter tractable algorithm with the number of rejecting states as the parameter. Moreover, we investigate the class of commutative permutation DFAs. Their structural properties allow us to decide compositionality in NL, and even in LOGSPACE if the alphabet size is fixed. Despite this low complexity, we show that complex behaviors still arise in this class: we provide a family of composite DFAs each requiring polynomially many factors with respect to its size. We also consider the variant of the problem that asks whether a DFA is *k-factor composite*, that is, decomposable into k smaller DFAs, for some given integer $k \in \mathbb{N}$. We show that, for commutative permutation DFAs, restricting the number of factors makes the decision computationally harder, and yields a problem with tight bounds: it is NP-complete. Finally, we show that in general, this problem is in PSPACE, and it is in LOGSPACE for DFAs with a singleton alphabet.

*The author was supported by Marie Skłodowska-Curie Grant Agreement No. 754411
†The author was supported by DFG-funded project FE560/9-1

1 Introduction

Compositionality is a fundamental notion in numerous fields of computer science [de Roever et al., 1998]. This principle can be summarized as follows: Every system should be designed by composing simple parts such that the meaning of the system can be deduced from the meaning of its parts, and how they are combined. For instance, this is a crucial aspect of modern software engineering: a program split into simple modules will be quicker to compile and easier to maintain. The use of compositionality is also essential in theoretical computer science: it is used to avoid the *state explosion* issues that usually happen when combining parallel processes together, and also to overcome the *scalability* issues of problems with a high theoretical complexity. In this work, we study compositionality in the setting of formal languages: we show how to make languages simpler by decomposing them into *intersections* of smaller languages. This is motivated by *model-checking* problems. For instance, the LTL model-checking problem asks, given a linear temporal logic formula φ and a finite state machine M, whether every execution of M satisfies φ. This problem is decidable, but has a high theoretical complexity (PSPACE) with respect to the size of φ [Baier and Katoen, 2008]. If φ is too long, it cannot be checked efficiently. This is where compositionality comes into play: if we can decompose the specification language into an intersection of simple languages, that is, decompose φ into a conjunction $\varphi = \varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_k$ of small specifications, it is sufficient to check whether all the φ_i are satisfied separately.

Our aim is to develop the theoretical foundations of the compositionality principle for formal languages by investigating how to decompose into simpler parts one of the most basic model of abstract machines: deterministic finite automata (DFAs). We say that a DFA $\mathcal{A}$ is *composite* if its language can be decomposed into the intersection of the languages of smaller DFAs. More precisely, we say that $\mathcal{A}$ is *k-factor composite* if there exist k DFAs $(\mathcal{A}_i)_{1 \leq i \leq k}$ with less states than $\mathcal{A}$ such that $L(\mathcal{A}) = \bigcap_{i=1}^{k} L(\mathcal{A}_i)$. We study the two following problems:

DFA Decomp	DFA Bound-Decomp
Given: DFA $\mathcal{A}$.	Given: DFA $\mathcal{A}$ and integer $k \in \mathbb{N}$.
Question: Is $\mathcal{A}$ composite?	Question: Is $\mathcal{A}$ k-factor composite?

The next example shows that decomposing DFAs can result in substantially smaller machines.

Example. Consider Figure 1. We simulate the interactions between a system and two clients by using finite words on the alphabet $\{r_1, r_2, g_1, g_2, i\}$: At each time step,

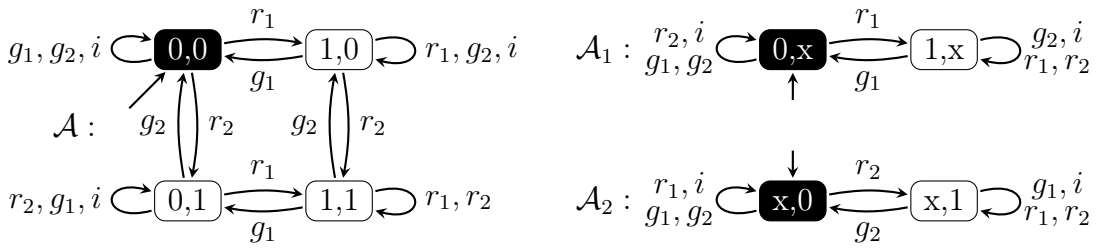


Figure 1: DFAs recognising specifications. Accepting states are drawn in black. The DFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ check that every request of the first, resp. second, client is eventually granted, $\mathcal{A}$ checks both.

the system either receives a request from a client (r_1, r_2), grants the open requests of a client (g_1, g_2), or stays idle (i). A basic property usually required is that every request is eventually granted. This specification is recognised by the DFA $\mathcal{A}$, which keeps track in its state of the current open requests, and only accepts if none is open when the input ends. Alternatively, this specification can be decomposed into the intersection of the languages defined by the DFAs $\mathcal{A}_1$ and $\mathcal{A}_2$: each one checks that the requests of the corresponding client are eventually granted. While in this precise example both ways of defining the specification are comparable, the latter scales drastically better than the former when the number of clients increases: Suppose that there are now $n \in \mathbb{N}$ clients. In order to check that all the requests are granted with a single DFA, we need 2^n states to keep track of all possible combinations of open requests, which is impractical when n gets too big. However, decomposing this specification into an intersection yields n DFAs of size two, one for each client. Note that, while in this specific example the decomposition is obvious, in general computing such a conjunctive form can be challenging: currently the best known algorithm needs exponential space.

DFAs in hardware. Our considered problems are of great interest in hardware implementations of finite state machines [Pedroni, 2013] where realizing large DFAs poses a challenge [Gould et al., 2007]. In [Clarke et al., 1991] the authors describe a state machine language for describing complex finite state hardware controllers, where the compiled state tables can automatically be input into a temporal logic model checker. If the control mechanism of the initial finite state machine can be split up into a conjunction of constraints, considering a decomposition instead could improve this work-flow substantially. Decomposing a complex DFA $\mathcal{A}$ can lead to a smaller representation of the DFA in total, as demonstrated in the previous example in Figure 1, and on top of that the individual smaller DFAs $\mathcal{A}_i$ in the decomposition $L(\mathcal{A}) = \bigcap_{i=1}^{k} L(\mathcal{A}_i)$ can be placed independently on a circuit board, as they do not have to interact with each other and only need to read their common input from a global bus and signal acceptance as a flag to the bus. This allows for a great flexibility in circuit designs, as huge DFAs can be

	DECOMP	BOUND-DECOMP
DFAs	EXPSPACE [Kupferman and Mosheiff, 2015]	**PSPACE**
Permutation DFAs	**NP/FPT**	**PSPACE**
Commutative permutation DFAs	**NL**	**NP-complete**
Unary DFAs	LOGSPACE [Jecker et al., 2020]	**LOGSPACE**

Figure 2: Complexity of studied problems with containing classes, with our contribution in **bold**.

broken down into smaller blocks which fit into niches giving space for inflexible modules such as CPU cores.

Reversible DFAs. We focus our study on *permutation* DFAs, which are DFAs whose transition monoids are groups: each letter induces a one-to-one map from the state set into itself. These DFAs are also called *reversible* DFAs [Kunc and Okhotin, 2013, Pin, 1992]. Reversibility is stronger than determinism: this powerful property allows to deterministically navigate *back and forth* between the steps of a computation. This is particularly relevant in the study of the physics of computation, since irreversibility causes energy dissipation [Landauer, 1961]. Remark that in the setting of DFAs, this power results in a loss of expressiveness: contrary to more powerful models (for instance Turing machines), reversible DFAs are less expressive than general DFAs.

Related work. The DFA DECOMP problem was first introduced in 2013 by Kupferman and Moscheiff [Kupferman and Mosheiff, 2015]. They proved that it is decidable in EXPSPACE, but left open the exact complexity: the best known lower bound is hardness for NL. They gave more efficient algorithms for restricted domains: a PSPACE algorithm for *permutation* DFAs, and a PTIME algorithm for *normal* permutation DFAs, a class of DFAs that contains all *commutative* permutation DFAs. Recently, the DECOMP problem was proved to be decidable in LOGSPACE for DFAs with a singleton alphabet [Jecker et al., 2020]. The trade-off between number and size of factors was studied in [Netser, 2018], where automata showing extreme behavior are presented, i.e., DFAs that can either be decomposed into a large number of small factors, or a small number of large factors.

Contribution. We expand the domain of instances over which the DECOMP problem is tractable. We focus on permutation DFAs, and we propose new techniques that improve the known complexities. Our results, summarised by Figure 2, are presented as follows.

Section 3: We give an NP algorithm for permutation DFAs, and we show that the complexity is directly linked to the number of non-accepting states. This allows us to obtain a fixed-parameter tractable algorithm with respect to the number of non-accepting states (Theorem 1). Moreover, we prove that permutation DFAs with a prime number of states cannot be decomposed (Theorem 2).

Section 4: We consider *commutative* permutation DFAs, where the DECOMP problem was already known to be tractable, and we lower the complexity from PTIME to NL, and even LOGSPACE if the size of the alphabet is fixed (Theorem 3). While it is easy to decide whether a commutative permutation DFA is composite, we show that rich and complex behaviors still appear in this class: there exist families of composite DFAs that require polynomially many factors to get a decomposition. More precisely, we construct a family $(\mathcal{A}_n^m)_{m,n\in\mathbb{N}}$ of composite commutative permutation DFAs such that $\mathcal{A}_n^m$ is a DFA of size n^m that is $(n-1)^{m-1}$-factor composite but not $(n-1)^{m-1}-1$-factor composite (Theorem 4). Note that, prior to this result, only families of composite DFAs with sub-logarithmic width were known [Jecker et al., 2020].

Section 5: Finally, we study the BOUND-DECOMP problem. High widths are undesirable for practical purposes: dealing with a huge number of small DFAs might end up being more complex than dealing with a single DFA of moderate size. The BOUND-DECOMP problem copes with this issue by limiting the number of factors allowed in the decompositions. We show that this flexibility comes at a cost: somewhat surprisingly, this problem is NP-complete for commutative permutation DFAs (Theorem 6), a setting where the DECOMP problem is easy. We also show that this problem is in PSPACE for the general setting (Theorem 5), and in LOGSPACE for unary DFAs, i.e., with a singleton alphabet (Theorem 7).

2 Definitions

We denote by $\mathbb{N}$ the set of non-negative integers $\{0, 1, 2, \ldots\}$. For a word $w = w_1 w_2 \ldots w_n$ with $w_i \in \Sigma$ for $1 \leq i \leq n$, we denote by $w^R = w_n \ldots w_2 w_1$ the *reverse* of w. Moreover, for every $\sigma \in \Sigma$, we denote by $\#_\sigma(w)$ the number of times the letter σ appears in w. A natural number $n > 1$ is called *composite* if it is the product of two smaller numbers, otherwise we say that n is *prime*. Two integers $m, n \in \mathbb{N}$ are called *co-prime* if their greatest common divisor is 1. We will use the following well known results [Hardy, 1929, Meher and Murty, 2013]:

Bertrand's Postulate: For all $n > 3$ there is a prime number p satisfying the condition $n < p < 2n - 2$.

Bézout's Identity: For every pair of integers $m, n \in \mathbb{N}$, the set $\{\lambda m - \mu n \mid \lambda, \mu \in \mathbb{N}\}$ contains exactly the multiples of the greatest common divisor of m and n.

Deterministic finite automata. A *deterministic finite automaton* (DFA hereafter) is a 5-tuple $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$, where Q is a finite set of states, Σ is a finite non-empty alphabet, $\delta \colon Q \times \Sigma \to Q$ is a transition function, $q_I \in Q$ is the initial state, and $F \subseteq Q$ is a set of accepting states. The states in $Q \setminus F$ are called *rejecting* states. We extend δ to words in the expected way, thus $\delta \colon Q \times \Sigma^* \to Q$ is defined recursively by $\delta(q, \varepsilon) = q$ and $\delta(q, w_1 w_2 \cdots w_n) = \delta(\delta(q, w_1 w_2 \cdots w_{n-1}), w_n)$. The *run* of $\mathcal{A}$ on a word $w = w_1 \dots w_n$ is the sequence of states $s_0, s_1, \dots, s_n$ such that $s_0 = q_I$ and for each $1 \leq i \leq n$ it holds that $\delta(s_{i-1}, w_i) = s_i$. Note that $s_n = \delta(q_I, w)$. The DFA $\mathcal{A}$ *accepts* w iff $\delta(q_I, w) \in F$. Otherwise, $\mathcal{A}$ *rejects* w. The set of words accepted by $\mathcal{A}$ is denoted $L(\mathcal{A})$ and is called the *language of* $\mathcal{A}$. A language accepted by some DFA is called a *regular language*.

We refer to the size of a DFA $\mathcal{A}$, denoted $|\mathcal{A}|$, as the number of states in $\mathcal{A}$. A DFA $\mathcal{A}$ is *minimal* if every DFA $\mathcal{B}$ such that $L(\mathcal{B}) = L(\mathcal{A})$ satisfies $|\mathcal{B}| \geq |\mathcal{A}|$.

Composite DFAs. We call a DFA $\mathcal{A}$ *composite* if there exists a family $(\mathcal{B}_i)_{1 \leq i \leq k}$ of DFAs with $|\mathcal{B}_i| < |\mathcal{A}|$ for all $1 \leq i \leq k$ such that $L(\mathcal{A}) = \bigcap_{1 \leq i \leq k} L(\mathcal{B}_i)$ and call the family $(\mathcal{B}_i)_{1 \leq i \leq k}$ a *decomposition* of $\mathcal{A}$. Note that, all $\mathcal{B}_i$ in the decomposition satisfy $|\mathcal{B}_i| < |\mathcal{A}|$ and $L(\mathcal{A}) \subseteq L(\mathcal{B}_i)$. Such DFAs are called *factors* of $\mathcal{A}$, and $(\mathcal{B}_i)_{1 \leq i \leq k}$ is also called a *k-factor decomposition* of $\mathcal{A}$. The *width* of $\mathcal{A}$ is the smallest k for which there is a k-factor decomposition of $\mathcal{A}$, and we say that $\mathcal{A}$ is *k-factor composite* iff $\mathit{width}(\mathcal{A}) \leq k$. We call a DFA $\mathcal{A}$ *prime* if it is not composite. We call a DFA $\mathcal{A}$ *trim* if all of its states are accessible from the initial state. As every non-trim DFA $\mathcal{A}$ is composite, we assume all given DFAs to be trim in the following.

We call a DFA a *permutation DFA* if for each letter $\sigma \in \Sigma$, the function mapping each state q to the state $\delta(q, \sigma)$ is a bijection. For permutation DFAs the transition monoid is a group. Further, we call a DFA $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ a *commutative DFA* if $\delta(q, uv) = \delta(q, vu)$ for every state q and every pair of words $u, v \in \Sigma^*$. In the next sections we discuss the problem of being composite for the classes of permutation DFA, and commutative permutation DFAs.

3 Decompositions of Permutation DFAs

In this section, we study permutation DFAs. Our main contribution is an algorithm for the DECOMP problem that is FPT with respect to the number of rejecting states:

Theorem 1. *The* DECOMP *problem for permutation DFAs is in NP. It is in FPT with parameter k, being the number of rejecting states of DFA $\mathcal{A}$, solvable in time $\mathcal{O}(2^k k^2 \cdot |\mathcal{A}|)$.*

We prove Theorem 1 by introducing the notion of *orbit-DFAs*: an orbit-DFA $\mathcal{A}^U$ of a DFA $\mathcal{A}$ is the DFA obtained by fixing a set of states U of $\mathcal{A}$ as the initial state, and letting the transition function of $\mathcal{A}$ act over it (thus the states of $\mathcal{A}^U$ are subsets of the state space of $\mathcal{A}$). We prove three key results:

- A permutation DFA is composite if and only if it can be decomposed into its orbit-DFAs (Corollary 1);
- A permutation DFA $\mathcal{A}$ can be decomposed into its orbit-DFAs if and only if for each of its rejecting states q, there exists an orbit-DFA $\mathcal{A}^U$ smaller than $\mathcal{A}$ that *covers* q, that is, one of the states of $\mathcal{A}^U$ contains q and no accepting states of $\mathcal{A}$ (Lemma 3);
- Given a permutation DFA $\mathcal{A}$ and a rejecting state q, we can determine the existence of an orbit-DFA covering q in non-deterministic time $\mathcal{O}(|\mathcal{A}|^2)$, and in deterministic time $\mathcal{O}(2^k k \cdot |\mathcal{A}|)$, where k is the number of rejecting states of $\mathcal{A}$ (Lemma 4, Algorithm 1).

These results directly imply Theorem 1. We also apply them to show that the DECOMP problem is trivial for permutation DFAs with a prime number of states.

Theorem 2. *Let $\mathcal{A}$ be a permutation DFA with at least one accepting state and one rejecting state. If the number of states of $\mathcal{A}$ is prime, then $\mathcal{A}$ is prime.*

3.1 Proof of Theorem 1

Consider a DFA $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$. We extend δ to subsets $U \subseteq Q$ in the expected way:

$$\delta(U, w) = \{q \in Q \mid q = \delta(p, w) \text{ for some } p \in U\} \text{ for every word } w \in \Sigma^*.$$

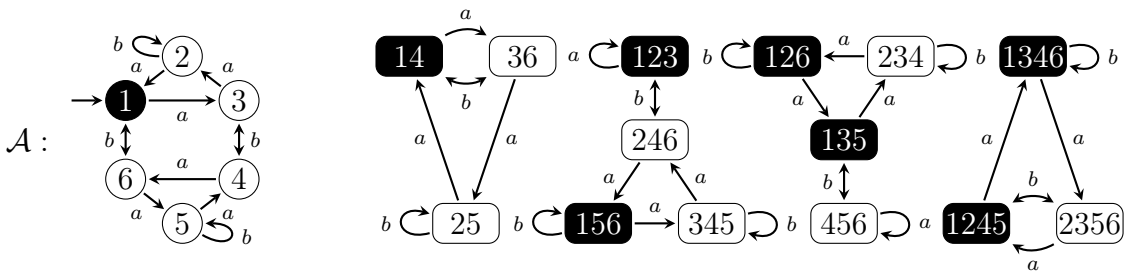


Figure 3: A DFA $\mathcal{A}$ together with some of its orbit-DFAs. Accepting states are depicted in black, an orbit-DFA can be obtained by setting a subset containing a 1 as an initial state. For instance the orbit-DFAs $\mathcal{A}^{\{1,2,3\}}$ and $\mathcal{A}^{\{1,5,6\}}$ form a decomposition of $\mathcal{A}$.

The *orbit* of U is the collection $\mathcal{C}_U = \{\delta(U, w) \subseteq Q \mid w \in \Sigma^*\}$ of subsets of Q that can be reached from U by the action of δ. If the subset $U \subseteq Q$ contains the initial state q_I of $\mathcal{A}$, we define the *orbit-DFA* $\mathcal{A}^U = \langle \Sigma, \mathcal{C}_U, U, \delta, \mathcal{C}' \rangle$, where the state space $\mathcal{C}_U$ is the orbit of U, and the set $\mathcal{C}'$ of accepting states is composed of the sets $U' \in \mathcal{C}_U$ that contain at least one of the accepting states of $\mathcal{A}$: $U' \cap F \neq \varnothing$. Note that $\mathcal{A}^U$ can alternatively be defined as the standard subset construction starting with the set $U \subseteq Q$ as initial state. The definition of the accepting states guarantees that $L(\mathcal{A}) \subseteq L(\mathcal{A}^U)$:

Proposition 1. *Every orbit-DFA $\mathcal{A}^U$ of a DFA $\mathcal{A}$ satisfies $L(\mathcal{A}) \subseteq L(\mathcal{A}^U)$.*

Proof. For every word w accepted by $\mathcal{A}$, the state $\delta(q_I, w)$ that $\mathcal{A}$ visits after reading w is accepting. Moreover, as the initial state q_I of $\mathcal{A}$ is in U, the state $\delta(U, w)$ that $\mathcal{A}^U$ visits after reading w contains the state $\delta(q_I, w) \in F$. Therefore, we get that $\delta(q_I, w) \in \delta(U, w) \cap F$, hence $\delta(U, w)$ is an accepting state of $\mathcal{A}^U$, which proves that $w \in L(\mathcal{A}^U)$. □

Example. Let us detail the orbits of the DFA $\mathcal{A}$ depicted in Figure 3. This DFA contains six states, and generates the following non-trivial orbits on its subsets of states:

- The 15 subsets of size 2 are split into two orbits: one of size 3, and one of size 12;
- The 20 subsets of size 3 are split into three orbits: two of size 4, and one of size 12;
- The 15 subsets of size 4 are split into two orbits, one of size 3, and one of size 12.

Figure 3 illustrates the four orbits smaller than $|\mathcal{A}|$: they induce seven orbit-DFAs, obtained by setting as initial state one of the depicted subsets containing the initial state 1 of $\mathcal{A}$.

In order to prove that a DFA is composite if and only if it can be decomposed into its orbit-DFAs, we prove that every factor $\mathcal{B}$ of a permutation DFA $\mathcal{A}$ can be turned into an

orbit-DFA $\mathcal{A}^U$ that is also a factor of $\mathcal{A}$, and satisfies $L(\mathcal{A}^U) \subseteq L(\mathcal{B})$. Our proof is based on a known result stating that factors can be turned into permutation DFAs:

Lemma 1 ([Kupferman and Mosheiff, 2015, Theorem 7.4])**.** *Let $\mathcal{A}$ be a permutation DFA. For every factor $\mathcal{B}$ of $\mathcal{A}$, there exists a permutation DFA $\mathcal{C}$ satisfying $|\mathcal{C}| \leq |\mathcal{B}|$ and $L(\mathcal{A}) \subseteq L(\mathcal{C}) \subseteq L(\mathcal{B})$.*

We strengthen this result by showing how to transform factors into orbit-DFAs:

Lemma 2. *Let $\mathcal{A}$ be a permutation DFA. For every factor $\mathcal{B}$ of $\mathcal{A}$, there exists an orbit-DFA $\mathcal{A}^U$ of A satisfying $|\mathcal{A}^U| \leq |\mathcal{B}|$ and $L(\mathcal{A}) \subseteq L(\mathcal{A}^U) \subseteq L(\mathcal{B})$.*

Proof. Let $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ be a permutation DFA, and let $\mathcal{B}$ be a factor of $\mathcal{A}$. By Lemma 1, there exists a *permutation* DFA $\mathcal{B}' = \langle \Sigma, S, s_I, \eta, G \rangle$ satisfying $|\mathcal{B}'| \leq |\mathcal{B}|$ and $L(\mathcal{A}) \subseteq L(\mathcal{B}') \subseteq L(\mathcal{B})$. We build, based on $\mathcal{B}'$, an orbit-DFA $\mathcal{A}^U$ of $\mathcal{A}$ satisfying the statement.

We say that a state $q \in Q$ of $\mathcal{A}$ is *linked* to a state $s \in S$ of $\mathcal{B}'$, denoted $q \sim s$, if there exists a word $u \in \Sigma^*$ satisfying $\delta(q_I, u) = q$ and $\eta(s_I, u) = s$. Let $f : S \to 2^Q$ be the function mapping every state $s \in S$ to the set $f(s) \subseteq Q$ containing all the states $q \in Q$ that are linked to s (i.e. satisfying $q \sim s$). We set $U = f(s_I)$. In particular, the initial state q_I of $\mathcal{A}$ is in U since $\delta(q_I, \varepsilon) = q_I$ and $\eta(s_I, \varepsilon) = s_I$. We show that the orbit-DFA $\mathcal{A}^U$ satisfies the desired conditions: $|\mathcal{A}^U| \leq |\mathcal{B}'|$ and $L(\mathcal{A}) \subseteq L(\mathcal{A}^U) \subseteq L(\mathcal{B}')$.

First, we show that $|\mathcal{A}^U| \leq |\mathcal{B}'|$ by proving that the function f defined earlier maps S surjectively into the orbit of U, which is the state space of $\mathcal{A}^U$. Since both $\mathcal{A}$ and $\mathcal{B}'$ are permutation DFAs, we get that for all $q \in Q$, $s \in S$ and $a \in \Sigma$, then $q \sim s$ if and only if $\delta(q, a) \sim \eta(s, a)$ holds.[1] Therefore, for every word $v \in \Sigma^*$, $f(\eta(s_I, v)) = \delta(f(s_I), v) = \delta(U, v)$. This shows that, as required, the image of the function f is the orbit of U, and f is surjective.

To conclude, we show that $L(\mathcal{A}) \subseteq L(\mathcal{A}^U) \subseteq L(\mathcal{B}')$. Proposition 1 immediately implies that $L(\mathcal{A}) \subseteq L(\mathcal{A}^U)$. Therefore it is enough to show that $L(\mathcal{A}^U) \subseteq L(\mathcal{B}')$. Let $v \in L(\mathcal{A}^U)$. By definition of an orbit-DFA, this means that the set $\delta(U, v)$ contains an accepting state q_F of $\mathcal{A}$. Since, as stated earlier, $f(\eta(s_I, v)) = \delta(U, v)$, this implies (by definition of the function f) that the accepting state q_F of $\mathcal{A}$ is linked to $\eta(s_I, v)$, i.e., there exists a word $v' \in \Sigma^*$ such that $\delta(q_I, v') = q_F$ and $\eta(s_I, v') = \eta(s_I, v)$. Then $\delta(q_I, v') = q_F$ implies that v' is in the language of $\mathcal{A}$. Moreover, since $L(\mathcal{A}) \subseteq L(\mathcal{B}')$ by supposition, v' is also accepted by $\mathcal{B}'$, i.e., $\eta(s_I, v')$ is an accepting state of $\mathcal{B}'$. Therefore, since $\eta(q_I, v') =$

[1] Remark that for general DFAs we only get that $q \sim s$ implies $\delta(q, a) \sim \eta(s, a)$ from the determinism. It is the *backward determinism* of the permutation DFAs $\mathcal{A}$ and $\mathcal{B}'$ that gives us the reverse implication.

$\eta(q_I, v)$, the word v is also in the language of $\mathcal{B}'$. This shows that $L(\mathcal{A}^U) \subseteq L(\mathcal{B}')$, which concludes the proof. □

As an immediate corollary, every decomposition of a permutation DFA can be transformed, factor after factor, into a decomposition into orbit-DFAs.

Corollary 1. *A permutation DFA is composite if and only if it can be decomposed into its orbit-DFAs.*

Orbit cover. Given a rejecting state $q \in Q \setminus F$ of $\mathcal{A}$, we say that the orbit-DFA $\mathcal{A}^U$ *covers* q if $|\mathcal{A}^U| < |\mathcal{A}|$, and $\mathcal{A}^U$ contains a rejecting state $U' \subseteq Q$ that contains q. Remember that, by definition, this means that U' contains no accepting state of $\mathcal{A}$, i.e., $U' \cap F = \varnothing$. We show that permutation DFAs that can be decomposed into their orbit-DFAs are characterized by the existence of orbit-DFAs covering each of their rejecting states.

Lemma 3. *A permutation DFA $\mathcal{A}$ is decomposable into its orbit-DFAs if and only if every rejecting state of $\mathcal{A}$ is covered by an orbit-DFA $\mathcal{A}'$ of $\mathcal{A}$ satisfying $|\mathcal{A}'| < |\mathcal{A}|$.*

Proof. Let $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ be a permutation DFA. We prove both implications.

Suppose that $\mathcal{A}$ can be decomposed into its orbit-DFAs $(\mathcal{A}^{U_i})_{1 \leq i \leq k}$, and let $q \in Q \setminus F$ be a rejecting state of $\mathcal{A}$. We show that q is covered by every orbit-DFA $\mathcal{A}^{U_i}$ that rejects a word $w \in \Sigma^*$ satisfying $\delta(q_I, w) = q$. Formally, let $w \in \Sigma^*$ be a word satisfying $\delta(q_I, w) = q$. Then $w \notin L(\mathcal{A}) = \bigcap_{i=1}^{n} L(\mathcal{A}^{U_i})$, hence there exists $1 \leq i \leq n$ such that $w \notin L(\mathcal{A}^{U_i})$. Let $U' \subseteq Q$ be the state visited by $\mathcal{A}^{U_i}$ after reading w. Then, by applying the definition of an orbit-DFA, we get that $q \in U'$ since $\delta(q_I, w) = q$, and $U' \cap F = \varnothing$ since U' is a rejecting state of $\mathcal{A}^{U_i}$ (as $w \notin L(\mathcal{A}^{U_i})$). Therefore, $\mathcal{A}^{U_i}$ covers q. Moreover, $|\mathcal{A}^{U_i}| < |\mathcal{A}|$ since $\mathcal{A}^{U_i}$ is a factor of $\mathcal{A}$.

Conversely, let us fix an enumeration $q_1, q_2, \ldots, q_m$ of the rejecting states of $\mathcal{A}$, and suppose that for all $1 \leq i \leq m$ there is an orbit-DFA $\mathcal{A}^{U_i}$ of $\mathcal{A}$ that covers q_i and satisfies $|\mathcal{A}^{U_i}| < |\mathcal{A}|$. Let $(U_{i.j})_{1 \leq j \leq n_i}$ be an enumeration of the subsets in the orbit of U_i that contain the initial state q_I of $\mathcal{A}$. We conclude the proof by showing that $S = \{\mathcal{A}^{U_{i.j}} \mid 1 \leq i \leq m, 1 \leq j \leq n_i\}$ is a decomposition of $\mathcal{A}$. Note that we immediately get $|\mathcal{A}^{U_{i.j}}| = |\mathcal{A}^{U_i}| < |\mathcal{A}|$ for all $1 \leq i \leq m$ and $1 \leq j \leq n_i$. Moreover, Proposition 1 implies $L(\mathcal{A}) \subseteq \bigcap_{\mathcal{A}' \in S} L(\mathcal{A}')$. To complete the proof, we show that $\bigcap_{\mathcal{A}' \in S} L(\mathcal{A}') \subseteq L(\mathcal{A})$. Let $w \in \Sigma^*$ be a word rejected by $\mathcal{A}$. To prove the desired inclusion, we show that there is a DFA $\mathcal{A}' \in S$ that rejects w. Since $w \notin L(\mathcal{A})$, the run of $\mathcal{A}$ on w starting from the

Function isComposite($\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$: permutation DFA)
  foreach $p \in Q \setminus F$ **do**
    guess U **with** $\{p\} \subseteq U \subseteq Q \setminus F$ /* guess rejecting state U of some orbit-DFA, such that U contains rejecting state p of $\mathcal{A}$ */
    if not cover($\mathcal{A}, p, U$) **then return** False
  return True

Function cover($\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$: permutation DFA, $p \in Q \setminus F$, $U \subseteq Q \setminus F$)
  $\mathcal{C}_U^{\text{old}} = \varnothing$
  $\mathcal{C}_U := \{U\}$
  while $\mathcal{C}_U \neq \mathcal{C}_U^{old}$ **and** $|\mathcal{C}_U| < |Q|$ **do**
    $\mathcal{C}_U^{\text{old}} := \mathcal{C}_U$
    $\mathcal{C}_U := \mathcal{C}_U \cup \{\delta(S, \sigma) \mid S \in \mathcal{C}_U, \sigma \in \Sigma\}$
  if $|\mathcal{C}_U| \geq |Q|$ **then return** False /* check that orbit-DFA is factor */
  foreach $S \in \mathcal{C}_U$ **do**
    if $q_I \in S$ **then return** True /* check that U is reachable from the inital state of the orbit-DFA */
  return False

Algorithm 1: NP-algorithm for the DECOMP problem for permutation DFAs.

initial state ends in a rejecting state q_i, for some $1 \leq i \leq m$. By supposition the orbit-DFA $\mathcal{A}^{U_i}$ covers q_i, hence the orbit of U_i contains a set $U' \subseteq Q$ that contains q_i and no accepting state. Note that there is no guarantee that $\mathcal{A}^{U_i}$ rejects w: while the set $\delta(U_i, w)$ contains q_i, it is not necessarily equal to U', and might contain accepting states. However, as $\mathcal{A}$ is a permutation DFA, we can reverse all of the transitions of $\mathcal{A}$ to get a path labeled by the reverse of w that starts from U' (that contains q_i), and ends in one of the sets $U_{i.j}$ (that contains q_I).[2] Therefore, by reversing this path back to normal, we get that $\delta(U_{i.j}, w) = U'$, hence the orbit-DFA $\mathcal{A}^{U_{i.j}} \in S$ rejects w. Therefore, every word rejected by $\mathcal{A}$ is rejected by an orbit-DFA $\mathcal{A}' \in S$, which shows that $\bigcap_{\mathcal{A}' \in S} L(\mathcal{A}') \subseteq L(\mathcal{A})$. □

This powerful lemma allows us to easily determine whether a permutation DFA is composite if we know its orbits. For instance, the DFA $\mathcal{A}$ depicted in Figure 3 is composite since the orbit-DFA $\mathcal{A}^{\{1,2,3\}}$ covers its five rejecting states. Following the proof of Lemma 3, we get that $(\mathcal{A}^{\{1,2,3\}}, \mathcal{A}^{\{1,5,6\}})$ is a decomposition of $\mathcal{A}$, and so is $(\mathcal{A}^{\{1,2,6\}}, \mathcal{A}^{\{1,3,5\}})$.

To conclude, we give an algorithm checking if a rejecting state is covered by an orbit-DFA. The whole NP-algorithm for checking whether a permutation DFA is composite is summarized in Algorithm 1.

Lemma 4. *Given a permutation DFA $\mathcal{A}$ and a rejecting state q, we can determine the existence of an orbit-DFA that covers q in nondeterministic time $\mathcal{O}(k \cdot |\mathcal{A}|^2)$, and in deterministic time $\mathcal{O}(2^k k \cdot |\mathcal{A}|^2)$, where k is the number of rejecting states of $\mathcal{A}$.*

[2] Remark that, if $\mathcal{A}$ is not a permutation DFA, then some states might not have incoming transitions for every letter. Thus, the reversal of w might not be defined.

Proof. We can decide in NP whether there exists an orbit-DFA $\mathcal{A}^U$ of $\mathcal{A}$ that covers p: we non-deterministically guess among the set of rejecting states of $\mathcal{A}$ a subset U' containing p. Then, we check in polynomial time that the orbit of U' is smaller than $|\mathcal{A}|$. This property can be checked in time $\mathcal{O}(|\mathcal{A}|^2)$. Since $\mathcal{A}$ is trim, in the orbit of U' there is a set U containing the initial state of $\mathcal{A}$. Moreover, since $\mathcal{A}$ is a permutation DFA, U and U' induce the same orbit. Hence, p is covered by the orbit-DFA $\mathcal{A}^U$. Finally, we can make this algorithm deterministic by searching through the 2^k possible subsets U' of the set of rejecting states of $\mathcal{A}$. □

3.2 Proof of Theorem 2

Thanks to the notion of orbit DFAs we are able to prove that a permutation DFA which has a prime number of states with at least one accepting and one rejecting, is prime.

Proof. Let $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ be a trim permutation DFA with a state space Q of prime size that contains at least one accepting state and one rejecting state. We show that the only orbit of $\mathcal{A}$ smaller than $|Q|$ is the trivial orbit $\{Q\}$. This implies that $\mathcal{A}$ cannot be decomposed into its orbit-DFAs, which proves that $\mathcal{A}$ is prime by Lemma 2.

Let us consider a strict subset $U_1 \neq \varnothing$ of the state space Q, together with its orbit $\mathcal{C}_{U_1} = \{U_1, U_2, \ldots, U_m\}$. We prove that $m \geq |Q|$. First, we show that all the U_i have the same size: since U_i is an element of the orbit of U_1, there exists a word $u_i \in \Sigma^*$ satisfying $\delta(U_1, u_i) = U_i$, and, as every word in Σ^* induces via δ a permutation on the state space, $|U_i| = |\delta(U_1, u_i)| = |U_1|$. Second, for every $q \in Q$, we define the *multiplicity* of q in $\mathcal{C}_{U_1}$ as the number $\lambda(q) \in \mathbb{N}$ of distinct elements of $\mathcal{C}_{U_1}$ containing the state q. We show that all the states q have the same multiplicity: since $\mathcal{A}$ is trim, there exists a word $u_q \in \Sigma^*$ satisfying $\delta(q_I, u_q) = q$, hence u_q induces via δ a bijection between the elements of $\mathcal{C}_{U_1}$ containing q_I and those containing q, and $\lambda(q) = \lambda(\delta(q_I, u_q)) = \lambda(q_I)$. By combining these results, we obtain $m \cdot |U_1| = \Sigma_{i=1}^{m} |U_i| = \Sigma_{q \in Q} \lambda(q) = \lambda(q_I) \cdot |Q|$. Therefore, as $|Q|$ is prime by supposition, either m or $|U_1|$ is divisible by $|Q|$. However, $U_1 \subsetneq Q$, hence $|U_1| < |Q|$, which shows that m is divisible by $|Q|$. In particular, we get $m \geq |Q|$, which concludes the proof. □

4 Decompositions of Commutative Permutation DFAs

We now study *commutative* permutation DFAs: a DFA $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ is commutative if $\delta(q, uv) = \delta(q, vu)$ for every state q and every pair of words $u, v \in \Sigma^*$. Our

main contribution is an NL algorithm for the DECOMP problem for commutative permutation DFAs. Moreover, we show that the complexity goes down to LOGSPACE for alphabets of fixed size.

Theorem 3. *The* DECOMP *problem for commutative permutation DFAs is in NL, and in LOGSPACE when the size of the alphabet is fixed.*

The proof of Theorem 3 is based on the notion of *covering word*: a word $w \in \Sigma^*$ covers a rejecting state q of a DFA $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ if $\delta(q, w) \neq q$, and for every $\lambda \in \mathbb{N}$, the state $\delta(q, w^\lambda)$ is rejecting. We prove two related key results:

- A commutative permutation DFA is composite if and only if each of its rejecting states is covered by a word (Lemma 5).
- We can decide in NL (LOGSPACE when the size of the alphabet is fixed) if a given rejecting state of a DFA is covered by a word (Lemma 6, and Algorithm 2).

These results immediately imply Theorem 3. We conclude this section by showing an upper bound on the width of permutation DFAs and constructing a family of DFAs of polynomial width.

Theorem 4. *The width of every composite permutation DFA is smaller than its size. Moreover, for all $m, n \in \mathbb{N}$ such that n is prime, there exists a commutative permutation DFA of size n^m and width $(n-1)^{m-1}$.*

We show that the width of a commutative permutation DFA is bounded by its number of rejecting states (Lemma 5). Then, for each $m, n \in \mathbb{N}$ with n prime, we define a DFA $\mathcal{A}_n^m$ of size n^m that can be decomposed into $(n-1)^{m-1}$ factors (Proposition 3), but not into $(n-1)^{m-1} - 1$ (Proposition 4).

4.1 Proof of Theorem 3

The proof is based on the following key property of commutative permutation DFAs: In a permutation DFA $\mathcal{A}$, every input word acts as a permutation on the set of states, generating disjoint cycles, and if $\mathcal{A}$ is commutative these cycles form an orbit.

Proposition 2. *Let $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ be a commutative permutation DFA. For all $u \in \Sigma^*$, the sets $(\{\delta(q, u^\lambda) \mid \lambda \in \mathbb{N}\})_{q \in Q}$ partition Q and form an orbit of $\mathcal{A}$.*

Proof. Let $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ be a commutative permutation DFA. Given $u \in \Sigma^*$ and $q \in Q$, the sequence of states $\delta(q,u), \delta(q,u^2), \ldots, \delta(q,u^i)$ visited by applying δ on iterations of u eventually repeats i.e. $\delta(q,u^x) = \delta(q,u^y) = p$ for some $x, y \in \mathbb{N}$ and $p \in Q$. Since $\mathcal{A}$ is a permutation DFA, it is both forward and backward deterministic, thus the set of visited states $\{\delta(q,u^\lambda) \mid \lambda \in \mathbb{N}\}$ is a cycle that contain both p and q. The collection $(\{\delta(q,u^\lambda) \mid \lambda \in \mathbb{N}\})_{q \in Q}$ forms an orbit of $\mathcal{A}$ by commutativity. Formally, for all $u, v \in \Sigma^*$ and every $q \in Q$, we have: $\delta(\{\delta(q,u^\lambda) | \lambda \in \mathbb{N}\}, v) = \{\delta(q, u^\lambda v) | \lambda \in \mathbb{N}\} = \{\delta(q, vu^\lambda) | \lambda \in \mathbb{N}\} = \{\delta(\delta(q,v), u^\lambda) | \lambda \in \mathbb{N}\}$. □

We proved with Corollary 1 and Lemma 3 that a permutation DFA is composite if and only if each of its rejecting states is covered by an orbit-DFA. We now reinforce this result for *commutative* permutation DFAs. As stated before, we say that a word $u \in \Sigma^*$ *covers* a rejecting state q of a DFA $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ if u induces from q a non-trivial cycle composed of rejecting states: $\delta(q,u) \neq q$, and $\delta(q,u^\lambda)$ is rejecting for all $\lambda \in \mathbb{N}$. Note that the collection $(\{\delta(q,u^\lambda) \mid \lambda \in \mathbb{N}\})_{q \in Q}$ forms an orbit of $\mathcal{A}$ by Proposition 2. We show that we can determine if $\mathcal{A}$ is composite by looking for words covering its rejecting states.

Lemma 5. *For every $k \in \mathbb{N}$, a commutative permutation DFA $\mathcal{A}$ is k-factor composite if and only if there exist k words that, together, cover all the rejecting states of $\mathcal{A}$.*

Proof. Let $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ be a commutative permutation DFA and $k \in \mathbb{N}$. We start by constructing k factors based on k covering words. Suppose that there exist k words $u_1, u_2, \ldots, u_k$ such that every rejecting state $q \in Q \setminus F$ is covered by one of the u_i. Note that all the u_i covering at least one state q do not act as the identity on Q (since $\delta(q,u_i) \neq q$), therefore we suppose, without loss of generality, that none of the u_i acts as the identity on Q. For every $1 \leq i \leq k$, let $U_i = \{\delta(q_I, u_i^\lambda) \mid \lambda \in \mathbb{N}\}$. We show that $(\mathcal{A}^{U_i})_{1 \leq i \leq k}$ is a decomposition of $\mathcal{A}$. As none of the u_i acts as the identity on Q, Proposition 2 implies that every $\mathcal{A}^{U_i}$ is smaller than $\mathcal{A}$. Moreover, Proposition 1 implies that $L(\mathcal{A}) \subseteq L(\mathcal{A}^{U_i})$, hence $L(\mathcal{A}) \subseteq \bigcap_{j=1}^{k} L(\mathcal{A}^{U_j})$. To conclude, we show that $\bigcap_{j=1}^{k} L(\mathcal{A}^{U_j}) \subseteq L(\mathcal{A})$. Let $u \notin L(\mathcal{A})$. By supposition, there exists $1 \leq i \leq k$ such that u_i covers $\delta(q_I, u)$. As a consequence, the set

$$\begin{aligned} \delta(U_i, u) &= \delta(\{\delta(q_I, u_i^\lambda) \mid \lambda \in \mathbb{N}\}, u) = \{\delta(q_I, u_i^\lambda u) \mid \lambda \in \mathbb{N}\} = \{\delta(q_I, u u_i^\lambda) \mid \lambda \in \mathbb{N}\} \\ &= \{\delta(\delta(q_I, u), u_i^\lambda) \mid \lambda \in \mathbb{N}\} \end{aligned}$$

contains no accepting state of $\mathcal{A}$, hence it is a rejecting state of $\mathcal{A}^{U_i}$. As a consequence, we get $u \notin L(\mathcal{A}^{U_i}) \supseteq \bigcap_{j=1}^{k} L(\mathcal{A}^{U_j})$, which proves that $\bigcap_{j=1}^{k} L(\mathcal{A}^{U_j}) \subseteq L(\mathcal{A})$.

We now construct k covering words based on k factors. Suppose that $\mathcal{A}$ has a k-factor

decomposition $(\mathcal{B}_i)_{1\leq i\leq k}$. Lemma 1 directly implies that this decomposition can be transformed into a decomposition $(\mathcal{C}_i)_{1\leq i\leq k}$ of $\mathcal{A}$, where $\mathcal{C}_i = \langle \Sigma, S_i, s_I^i, \eta_i, G_i\rangle$ are permutation DFAs. For every $1 \leq i \leq k$, we build a word u_i based on $\mathcal{C}_i$, we prove that every rejecting state of $\mathcal{A}$ is covered by one of these u_i. Consider $1 \leq i \leq k$. Since $\mathcal{C}_i$ is a factor of $\mathcal{A}$, in particular $|\mathcal{C}_i| < |\mathcal{A}|$, hence there exist two input words $v_i, w_i \in \Sigma^*$ such that $\mathcal{A}$ reaches different states on v_i and w_i, but $\mathcal{C}_i$ reaches the same state: $\delta(q_I, v_i) \neq \delta(q_I, w_i)$ but $\eta_i(s_I^i, v_i) = \eta_i(s_I^i, w_i)$. Note that both $\mathcal{A}$ and $\mathcal{C}_i$ are permutation DFAs, hence there exists a power $v_i^{\kappa_i}$ of v_i that induces the identity function on both state spaces Q and S_i. We set $u_i = w_i v_i^{\kappa_i - 1}$, which guarantees that:

$$\delta(q_I, u_i) = \delta(\delta(q_I, w_i), v_i^{\kappa_i-1}) \neq \delta(\delta(q_I, v_i), v_i^{\kappa_i-1}) = \delta(q_I, v_i^{\kappa_i}) = q_I;$$
$$\eta_i(s_I^i, u_i) = \eta_i(\eta_i(s_I^i, w_i), v_i^{\kappa_i-1}) = \eta_i(\eta_i(s_I^i, v_i), v_i^{\kappa_i-1}) = \eta_i(s_I^i, v_i^{\kappa_i}) = s_I^i.$$

In other words, u_i moves the initial state q_I of $\mathcal{A}$, but fixes the initial state s_I^i of $\mathcal{C}_i$.

We now prove that each rejecting state of $\mathcal{A}$ is covered by one of the u_i. Let $q \in Q \setminus F$ be a rejecting state of $\mathcal{A}$. Since $\mathcal{A}$ is trim, there exists a word $u_q \in \Sigma^*$ such that $\delta(q_I, u_q) = q$. Then, as $u_q \notin L(\mathcal{A})$ and $(\mathcal{C}_i)_{1\leq i\leq k}$ is a decomposition of $\mathcal{A}$, there exists $1 \leq i \leq k$ such that $u_q \notin L(\mathcal{C}_i)$. We show that the word u_i covers the rejecting state q: we prove that $\delta(q, u_i) \neq q$, and that $\delta(q, u_i^\lambda)$ is rejecting for every $\lambda \in \mathbb{N}$. First, since $\mathcal{A}$ is a commutative permutation DFA and u_i moves q_I, we get that $\delta(q, u_i) = \delta(q_I, u_q u_i) = \delta(q_I, u_i u_q) \neq \delta(q_I, u_q) = q$. Moreover, for all $\lambda \in \mathbb{N}$, Since $u_q \notin L(\mathcal{C}_i)$ by supposition and u_i fixes s_I^i, the DFA $\mathcal{C}_i$ also rejects the word $u_i^\lambda u_q$. Therefore, as $L(\mathcal{A}) \subseteq L(\mathcal{C}_i)$, we finally get that $\delta(q, u_i^\lambda) = \delta(q_I, u_q u_i^\lambda) = \delta(q_I, u_i^\lambda u_q)$ is a rejecting state of $\mathcal{A}$. □

By Lemma 5, to conclude the proof of Theorem 3 we show that we can decide in NL (and in LOGSPACE when the size of the alphabet is fixed) whether a given rejecting state of a DFA is covered by a word (since in the Decomp problem we can afford to pick a covering word for each state). As we consider commutative permutation DFAs, we can represent a covering word by the number of occurrences of each letter, which are all bounded by $|Q|$.

Lemma 6. *Let $\mathcal{A}$ be a commutative permutation DFA and p a rejecting state.*

1. *We can determine the existence of a word covering p in space $\mathcal{O}(|\Sigma| \cdot \log |Q|)$;*
2. *We can determine the existence of a word covering p in NL;*

Proof of Item 1. Let $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F\rangle$ be a commutative permutation DFA with alphabet $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$. Note that (*) for every pair of states $p, q \in Q$, there exists a word $w_{p,q} = \sigma_1^{i_1}\sigma_2^{i_2}\ldots\sigma_m^{i_m} \in \Sigma^*$ with $i_1 + i_2 + \cdots + i_m < |Q|$ such that $\delta(p, w_{p,q}) = q$.

Further, note that for commutative permutation DFAs, $\delta(p, w^{|Q|}) = p$ for every state p and word $w \in \Sigma^*$. Let $p \in Q \setminus F$ be a rejecting state of $\mathcal{A}$. We can decide in logarithmic space whether there exists a word covering p as follows. Pick a state $q \in Q$ with $p \neq q$ and determine $w_{p,q}$ by iterating over all $|Q|^{|\Sigma|}$ words of the form described in (*) and check whether $\delta(p, w_{p,q}) = q$. Due to observation (*), we can represent each potential candidate for $w_{p,q}$ with $|\Sigma| \cdot \log(|Q|)$ bits.

Next, we have to ensure that all states in the cycle induced by $w_{p,q}$ on p are rejecting, i.e., we have to check that $\delta(p, w_{p,q}^{\lambda}) \notin F$ for all $\lambda \leq |Q|$. Therefore, we use two pointers, one for the state p and one for the image of p under $w_{p,q}^{\lambda}$ where λ denotes the current iteration step. As long as $\delta(p, w_{p,q}^{\lambda})$ is rejecting and $\neq p$ we continue to compute $\delta(p, w_{p,q}^{\lambda+1})$. If $\delta(p, w_{p,q}^{\lambda})$ is accepting, we abort the computation and repeat the computation with some other state q' instead of q. If we find $\delta(p, w_{p,q}^{\lambda}) = p$, we confirmed the existence of a word covering p. The current iteration step over the states p and q can be stored via two pointers. Note that the iteration step λ does not have to be stored due to the permutation property which ensures us to encounter p again, finally. □

Proof of Item 2. We adapt the algorithm presented in the proof of Item 1, and use the terminology introduced there. We adapt it in the sense that we do not store the word $w_{p,q}$ but instead guess it again every time we want to apply it to some state $\delta(p, w_{p,q}^{\lambda})$. Therefore, we store an extra copy of pointers to the states p and q. We guess w_{p,q_λ} applied in iteration step λ by successively guessing at most $|Q|$ letters $\sigma \in \Sigma$ and applying σ to both states p and the currently investigated state $\delta(p, w_{p,q_1} w_{p,q_2} \dots w_{p,q_{\lambda-1}})$. The counter on the number of guessed letters can be stored in $\log |Q|$ bits. We check that we actually reached $\delta(p, w_{p,q_1}^{\lambda})$ after guessing some w_{p,q_λ} by checking whether the state p is mapped to q. Note that the words w_{p,q_i} that are guessed in different iteration steps i of λ might differ, but they are all equivalent in the sense that they impose the same transitions in the cycle induced by w_{p,q_1} on p since $\mathcal{A}$ is a commutative permutation DFA. As the representation of $w_{p,q}$ was the only part using super-logarithmic space in the algorithm described in Lemma 6, the claim follows. Both variants of the algorithm (deterministic and non-deterministic) are depicted in Algorithm 2. □

4.2 Proof of Theorem 4

As a direct consequence of Lemma 5, the width of every commutative permutation DFA $\mathcal{A}$ is bounded by the number of rejecting states of $\mathcal{A}$, hence, it is smaller than $|\mathcal{A}|$. To conclude the proof of Theorem 4, for all $m, n \in \mathbb{N}$ with n prime, we define a DFA $\mathcal{A}_n^m$ of size n^m and width $(n-1)^{m-1}$ on the alphabet $\Sigma = \{a_1, a_2, \dots, a_m\}$. For all $\ell \in \mathbb{N}$, let $[\ell]$

Function isComposite($\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$: commutative permutation DFA)
  foreach $p \in Q \setminus F$ **do**
    cover_found:=False
    foreach $q \in Q \setminus F$ **with** $q \neq p$ **do**
      if cover($\mathcal{A}, p, q$) **then** cover_found:=True /* covering p with $w_{p,q}$ */
    if not cover_found **then return** False /* no cover found for p */
  return True /* all state p are covered */

Function cover($\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$: commutative permutation DFA, $p, q \in Q \setminus F$)
  $s := q$
  while $s \neq p$ **do** /* eventually $s = p$ $\mathcal{A}$ is a permuation DFA */
    $s :=$ mimic(p, q, s) /* thus $s := \delta(s, w_{p,q})$ */
    if $s \in F$ **then return** False /* contradiction of covering */
  return True /* encountered p again without hitting state in F */

Function mimic($\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$: commutative permutation DFA, $p, q, s \in Q \setminus F$)
  Assumption: $|\Sigma|$ is fixed, let $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$
  foreach $1 \leq x_1 + \cdots + x_{|\Sigma|} \leq |Q|$ **do** /* possible since $|\Sigma|$ is fixed */
    if $\delta(p, \sigma_1^{x_1}\sigma_2^{x_2}\ldots\sigma_m^{x_m}) = q$ **then return** $\delta(s, \sigma_1^{x_1}\sigma_2^{x_2}\ldots\sigma_m^{x_m})$

Function mimic($\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$: commutative permutation DFA, $p, q, s \in Q \setminus F$)
  Assumption: this algorithm is allowed to use non-determinism
  $p' := p, \ell := 0$
  while $p' \neq q$ **and** $\ell < |Q|$ **do**
    guess $\sigma \in \Sigma$ /* iteratively contruct $w_{p,q}$ */
    $p' := \delta(p', \sigma)$, $s := \delta(s, \sigma)$, $\ell := \ell + 1$
  if $\ell = |Q|$ **then return** *error* **else return** s /* check $q = \delta(p, w_{p,q})$ */

Algorithm 2: Deterministic and non-deterministic version of the algorithm solving the DECOMP problem for commutative permutation DFAs.

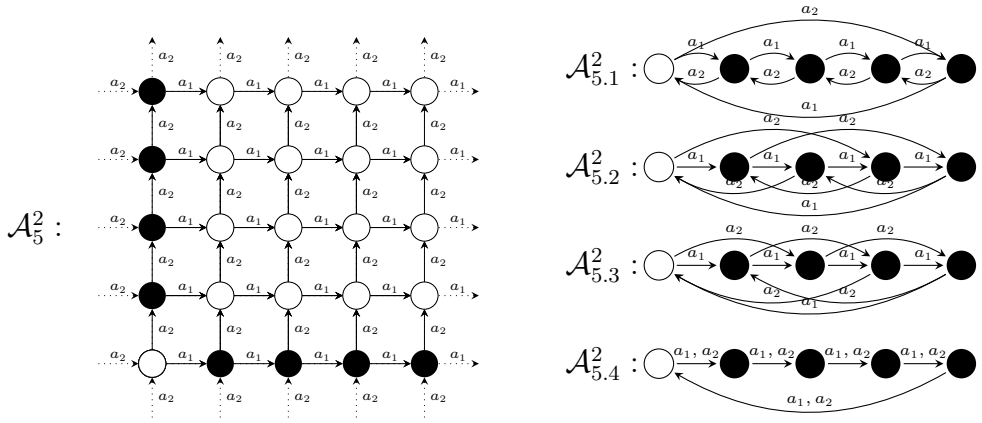


Figure 4: The DFA $\mathcal{A}_5^2$ recognising the language L_5^2, together with its decomposition into four non-trivial orbit-DFAs. Final states are depicted in black.

denote the equivalence class of ℓ modulo n. Let $L_n^m \subseteq \Sigma^*$ be the language composed of the words w such that for at least one letter $a_i \in \Sigma$ the number $\#_{a_i}(w)$ of a_i in w is a multiple of n, and for at least one (other) letter $a_j \in \Sigma$, the number $\#_{a_j}(w)$ of a_j in w is *not* a multiple of n:

$$L_n^m = \{w \in \Sigma^* \mid [\#_{a_i}(w)] = [0] \text{ and } [\#_{a_j}(w)] \neq [0] \text{ for some } 1 \leq i, j \leq m\}.$$

The language L_n^m is recognised by a DFA $\mathcal{A}_n^m$ of size n^m that keeps track of the value modulo n of the number of each a_i already processed. The state space of $\mathcal{A}_n^m$ is the direct product $(\mathbb{Z}/n\mathbb{Z})^m$ of m copies of the cyclic group $\mathbb{Z}/n\mathbb{Z} = ([0], [1], \ldots, [n-1])$; the initial state is $([0], [0], \ldots, [0])$; the final states are the ones containing at least one component equal to $[0]$ and one component distinct from $[0]$; and the transition function increments the i^{th} component when an a_i is read: $\delta(([j_1], [j_2], \ldots, [j_m]), a_i) = ([j_1], [j_2], \ldots, [j_{i-1}], [j_i + 1], [j_{i+1}], \ldots, [j_m])$. Figure 4 illustrates the particular case $n = 5$ and $m = 2$.

To prove that the width of $\mathcal{A}_n^m$ is $(n-1)^{m-1}$, we first show that the $(n-1)^{m-1}$ words $\{a_1 a_2^{\lambda_2} \ldots a_m^{\lambda_m} \mid 1 \leq \lambda_i \leq n-1\}$ cover all the rejecting states, thus by Lemma 5:

Proposition 3. *The DFA $\mathcal{A}_n^m$ is $(n-1)^{m-1}$-factor composite.*

Proof. For every $m-1$ tuple $\phi = (j_2, j_3, \ldots, j_m) \in \{1, 2, \ldots, n-1\}^{m-1}$, let u_ϕ be the word

$$u_\phi = a_1 a_2^{j_2} a_3^{j_3} a_4^{j_4} \ldots a_m^{j_m} \in \Sigma^*.$$

Note that there are $(n-1)^{m-1}$ distinct words u_ϕ. We show that every rejecting state of $\mathcal{A}_n^m$ is covered by one of the u_ϕ, which proves, by Lemma 5, that $\mathcal{A}_n^m$ is $(n-1)^{m-1}$-composite.

Let q be a rejecting state of $\mathcal{A}_n^m$. Remember that the rejecting states of $\mathcal{A}_n^m$ are precisely those for which either ($\blacklozenge$) all the components are $[0]$, or ($\star$) none of the component is $[0]$. We consider both possibilities.

($\blacklozenge$) If $q = ([0], [0], \ldots, [0])$ we show that every $u_\phi = a_1 a_2^{j_2} a_3^{j_3} \ldots a_m^{j_m}$ covers q: for all $\lambda \in \mathbb{N}$,

$$\begin{aligned} \delta(q, u_\phi^\lambda) &= \delta(([0], [0], \ldots, [0]), (a_1 a_2^{j_2} a_3^{j_3} a_4^{j_4} \ldots a_m^{j_m})^\lambda) \\ &= ([\lambda], [\lambda j_2], [\lambda j_3], [\lambda j_4], \ldots, [\lambda j_m]). \end{aligned}$$

Therefore, either $[\lambda] = [0]$ and all the components of $\delta(q, u_\phi^\lambda)$ are $[0]$, or $[\lambda] \neq [0]$ and none of the components of $\delta(q, u_\phi^\lambda)$ are $[0]$ (since n is prime and $1 < j_i < n-1$). In both cases, $\delta(q, u_\phi^\lambda)$ is rejecting. This proves that u_ϕ covers q.

($\star$) If $q = ([k_1], [k_2], \ldots, [k_m])$ such that none of the $[k_i]$ is equal to $[0]$, we build a specific u_ϕ that covers q. Since $[k_1] \neq 0$ and n is prime, there exists $\mu \in \mathbb{N}$ satisfying $[\mu \cdot k_1] = [1]$. Note that this implies that $[\mu] \neq [0]$, hence for every $2 \leq i \leq m$ we get that $[\mu k_i] = [j_i]$ for some $1 \leq j_i \leq n-1$. Let $\phi = (j_2, j_3, \ldots, j_m)$. Then for every $\lambda \in \mathbb{N}$

$$\begin{aligned} \delta(q, u_\phi^\lambda) &= \delta(([k_1], [k_2], \ldots, [k_m]), (a_1 a_2^{j_2} a_3^{j_3} a_4^{j_4} \ldots a_m^{j_m})^\lambda) \\ &= ([k_1 + \lambda], [k_2 + \lambda j_2], [k_3 + \lambda j_3], [k_4 + \lambda j_4], \ldots, [k_m + \lambda k_m]) \\ &= ([k_1 + \lambda\mu k_1], [k_2 + \lambda\mu k_2], [k_3 + \lambda\mu k_3], [k_4 + \lambda\mu k_4], \ldots, [k_m + \lambda\mu k_m]) \\ &= ([(1 + \lambda\mu)k_1], [(1 + \lambda\mu)k_2], [(1 + \lambda\mu)k_3], [(1 + \lambda\mu)k_4], \ldots, [(1 + \lambda\mu)k_m]). \end{aligned}$$

Remember that, by supposition, $[k_i] \neq [0]$ for all $1 \leq i \leq m$. Therefore, either it holds that $[\lambda\mu + 1] = [0]$ and all the components of $\delta(q, u_\phi^\lambda)$ are $[0]$, or $[\lambda\mu + 1] \neq [0]$ and none of the components of $\delta(q, u_\phi^\lambda)$ are $[0]$ (since n is prime). In both cases, $\delta(q, u_\phi^\lambda)$ is rejecting. This proves that u_ϕ covers q. □

Then, we prove that there exist no word that covers two states among the $(n-1)^{m-1}$ rejecting states $\{([1], [k_2], [k_3], \ldots, [k_m]) \mid 1 \leq k_i \leq m-1\}$. Therefore, we need at least $(n-1)^{m-1}$ words to cover all of the states, thus by Lemma 5:

Proposition 4. *The DFA $\mathcal{A}_n^m$ is not $((n-1)^{m-1} - 1)$-factor composite.*

Proof. For every $m-1$ tuple $\phi = (k_2, k_3, \ldots, k_m) \in \{1, 2, \ldots, n-1\}^{m-1}$, let q_ϕ denote the rejecting state $([1], [k_2], [k_3], \ldots, [k_m])$ of $\mathcal{A}_n^m$. Note that there are $(n-1)^{m-1}$ distinct q_ϕ. We show that there exists no word that covers two different q_ϕ, which proves, by Lemma 5, that $\mathcal{A}_n^m$ is not $((n-1)^{m-1} - 1)$-composite.

Let $\phi = (k_2, k_3, \ldots, k_m), \psi = (\ell_2, \ell_3, \ldots, \ell_m) \in \{1, 2, \ldots, n-1\}^{m-1}$, and let $u \in \Sigma^*$ be a word that covers both q_ϕ and q_ψ. We show that this implies $\phi = \psi$. Since $\mathcal{A}_n^m$ is

commutative, we can suppose without loss of generality that $u = a_1^{j_1} a_2^{j_2} \dots a_m^{j_m}$. Let $\lambda \in \mathbb{N}$ satisfying $[\lambda j_1] = [-1]$. Then,

$$\begin{aligned} \delta(q_\phi, u^\lambda) &= ([0], [k_2 + \lambda j_2], [k_3 + \lambda j_3], [k_4 + \lambda j_4], \dots, [k_m + \lambda j_m]) \\ \delta(q_\psi, u^\lambda) &= ([0], [\ell_2 + \lambda j_2], [\ell_3 + \lambda j_3], [\ell_4 + \lambda j_4], \dots, [\ell_m + \lambda j_m]). \end{aligned}$$

Since u covers both q_ϕ and q_ψ by supposition, both $\delta(q_\phi, u^\lambda)$ and $\delta(q_\psi, u^\lambda)$ are rejecting states of $\mathcal{A}_n^m$. Since the first component of both of these states is $[0]$, this implies that *all* of their components are $[0]$. In other words, for every $2 \leq i \leq m$ we get $[k_i + \lambda j_i] = [0] = [\ell_i + \lambda j_i]$, hence $k_i = \ell_i$. This proves that $\phi = \psi$. □

5 Bounded Decomposition

We finally study the Bound-Decomp problem: Given a DFA $\mathcal{A}$ and an integer $k \in \mathbb{N}$ encoded in unary, determine whether $\mathcal{A}$ is decomposable into k factors. For the general setting, we show that the problem is in PSPACE: it can be solved by non-deterministically guessing k factors, and checking that they form a decomposition.

Theorem 5. *The* Bound-Decomp *problem is in PSPACE.*

Proof. For a language $L \subseteq \Sigma^*$ we denote by $\overline{L}$ the complement $\Sigma^* \setminus L$ of L. Let $\mathcal{A} = \langle \Sigma, Q_\mathcal{A}, q_{I\mathcal{A}}, \delta_\mathcal{A}, F_\mathcal{A} \rangle$ be a DFA and let $k \in \mathbb{N}$ be encoded in unary. We non-deterministically guess $n \leq k$ DFAs $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n$ with $\mathcal{A}_i = \langle \Sigma, Q_{\mathcal{A}_i}, q_{I_{\mathcal{A}_i}}, \delta_{\mathcal{A}_i}, F_{\mathcal{A}_i} \rangle$ for $1 \leq i \leq n$, such that $|\mathcal{A}_i| < |\mathcal{A}|$. We implicitly build the product DFA $\Pi_1^n \mathcal{A}_i = \mathcal{A}_1 \times \mathcal{A}_2 \times \dots \mathcal{A}_n$ over the state space $Q_{\mathcal{A}_1} \times Q_{\mathcal{A}_2} \times \dots \times Q_{\mathcal{A}_n}$ with the start state $(q_{I_{\mathcal{A}_1}}, q_{I_{\mathcal{A}_2}}, \dots, q_{I_{\mathcal{A}_n}})$ and set of final states $F_{\mathcal{A}_1} \times F_{\mathcal{A}_2} \times \dots \times F_{\mathcal{A}_n}$, where in the i'th component the run of the DFA $\mathcal{A}_i$ on the input is simulated. We do not build this DFA explicitly as it is of exponential size in $|Q_\mathcal{A}|$. Note that $\Pi_1^n \mathcal{A}_i$ accepts $\bigcap_{1 \leq i \leq n} L(\mathcal{A}_i)$. In order to prove whether $L(\mathcal{A}) = L(\Pi_1^n \mathcal{A}_i)$ it is sufficient to verify that (1) $L(\mathcal{A}) \cap \overline{L(\Pi_1^n \mathcal{A}_i)} = \varnothing$ and (2) $\overline{L(\mathcal{A})} \cap L(\Pi_1^n \mathcal{A}_i) = \varnothing$. As $\Pi_1^n \mathcal{A}_i$ is a DFA, we can obtain a DFA for the complementary language $\overline{L(\Pi_1^n \mathcal{A}_i)}$ by complementing on the set of final states. We can test the complementary statement of both (1) and (2) by letter-wise guessing a word in the intersection and applying its map on the initial state of both DFAs. As we only need to store the active state of both DFAs, this can be done in NPSPACE. As NPSPACE is closed under complement and is equal to PSPACE, the claim follows. □

For commutative permutation DFAs, we obtain a better algorithm through the use of the results obtained in the previous sections, and we show a matching hardness result.

Theorem 6. *The* Bound-Decomp *problem for commutative permutation DFAs is NP-complete.*

Both parts of the proof of Theorem 6 are based on Lemma 5: a commutative permutation DFA is k-factor composite if and only if there exist k words covering all of its rejecting states. We prove the two following results:

- Bounded compositionnality is decidable in NP, as it is sufficient to non-deterministically guess a set of k words, and check whether they cover all rejecting states (Lemma 8);
- The NP-hardness is obtained by reducing the Hitting Set problem, a well known NP-complete decision problem. We show that searching for k words that cover the rejecting states of a DFA is as complicated as searching for a hitting set of size $k-1$ (Lemma 9).

We finally give a LOGSPACE algorithm based on known results for DFAs on unary alphabets [Jecker et al., 2020]. We begin with the case of unary DFAs.

Theorem 7. *The* Bound-Decomp *problem for unary DFAs is in LOGSPACE.*

Sketch. Recall that a unary DFA $\mathcal{A} = \langle \{a\}, Q, q_I, \delta, F \rangle$ consists of a chain of states leading into one cycle of states. The case where the chain is non-empty is considered in Lemmas 8-10 of [Jecker et al., 2020]. We prove that the criteria of these lemmas can be checked in LOGSPACE. If the chain of $\mathcal{A}$ is empty, then $\mathcal{A}$ is actually a commutative permutation DFA. In this case, by Proposition 2 for every word $u = a^i \in \{a\}^*$, the orbit of the set $\{\delta(q_I, u^\lambda) \mid \lambda \in \mathbb{N}\}$ is a partition ρ on Q, and every set in ρ has the same size s_ρ. Both s_ρ and $|\rho|$ divide $|Q|$. For $u = a^i$ where i and $|Q|$ are co-prime, the induced orbit DFA has a single state and thus cannot be a factor of $\mathcal{A}$. Further, if $i_1 < |Q|$ divides $i_2 < |Q|$, then all states covered by a^{i_1} are also covered by a^{i_2}. Hence, w.l.o.g., we only consider words of the form a^i where i is a *maximal divisor* of $|Q|$ in order to generate orbit-DFAs of $\mathcal{A}$ that are candidates for the decomposition. Now, let $p_1^{j_1} \cdot p_2^{j_2} \cdot \ldots \cdot p_m^{j_m} = |Q|$ be the prime factor decomposition of $|Q|$. Recall that $|Q|$ is given in unary and hence we can compute the prime factor decomposition of $|Q|$ in space logarithmic in $|Q|$. By Lemma 5 we have that $\mathcal{A}$ is k-factor composite if and only if a selection of k words from the set $\mathcal{W} = \{a^{|Q|/p_i} \mid 1 \leq i \leq m\}$ cover all the rejecting states of $\mathcal{A}$. As $|\mathcal{W}| = m$ is logarithmic in $|Q|$, we can iterate over all sets in $2^{\mathcal{W}}$ of size at most k in LOGSPACE using a binary string indicating the characteristic function. By Lemma 6, checking whether a state $q \in Q$ is covered by the current collection of k words can also be done in LOGSPACE. □

In order to prove the theorem, we first show that the result holds for unary permutation DFAs, and then we show how this extends to the general setting.

Lemma 7. *The* Bound-Decomp *problem for permutation unary DFAs is in LOGSPACE.*

Proof. Let $\mathcal{A} = \langle \{a\}, Q, q_I, \delta, F \rangle$ be a trim permutation DFA. Since the alphabet of $\mathcal{A}$ is unary, $\mathcal{A}$ is also commutative. Hence, by Proposition 2 for every word $u = a^i \in \{a\}^*$, the orbit of the set $\{\delta(q_I, u^\lambda) \mid \lambda \in \mathbb{N}\}$ is a partition ρ on Q. Since $\mathcal{A}$ is a permutation DFA, every word induces a permutation of Q and therefore, every set in the partition ρ has the same size s_ρ. Hence, both s_ρ and $|\rho|$ divide $|Q|$. Further, note that for every integer $i > 1$ there is at most one partition ρ of Q of size $|\rho| = i$ that is consistent with the transition relation of $\mathcal{A}$, and this partition (if existent) corresponds to the orbit-DFA generated by the word $a^{|Q|/i}$ where i evenly divides $|Q|$. Every word a^i where i and $|Q|$ are co-prime generates a trivial partition $\{\delta(q_I, u^\lambda) \mid \lambda \in \mathbb{N}\} = Q$ corresponding to the trivial orbit-DFA which has only one state. As trivial orbit-DFAs do not contribute to a decomposition, it is sufficient to consider only words a^i where i is a divisor of $|Q|$ to generate all orbit-DFAs of $\mathcal{A}$ that needs to be considered in order to obtain a k-factor decomposition of $\mathcal{A}$. Further, note that for two integers i_1 and i_2 it holds that if i_1 divides i_2, then all states covered by a^{i_1} are also covered by a^{i_2}. Therefore, while looking for covering words it is sufficient to look through the *maximal* divisors of $|Q|$, that is, the divisors that do not divide other divisors.

Now, let $p_1^{j_1} \cdot p_2^{j_2} \cdot \ldots \cdot p_m^{j_m} = |Q|$ be the prime factor decomposition of $|Q|$, i.e., p_i are prime numbers for $1 \leq i \leq m$. By the discussion above, and Lemma 5 we have that $\mathcal{A}$ is k-factor composite if and only if a selection of k words from the set $\mathcal{W} = \{a^{|Q|/p_i} \mid 1 \leq i \leq m\}$ cover all the rejecting states of $\mathcal{A}$. Note that $|\mathcal{W}| = m$ is logarithmic in $|Q|$ and hence, the size of the power set $2^{\mathcal{W}}$ of $\mathcal{W}$ is linear in $|Q|$. Further, we can represent a set in $2^{\mathcal{W}}$ as a binary string (with a 1 at position i iff the ith element is in the set represented by the string) of size m and iterate through $2^{\mathcal{W}}$ in logarithmic space. We can now check whether $\mathcal{A}$ is k-factor composite by iterating through $2^{\mathcal{W}}$ and testing whether for one set of words, each rejecting state of $\mathcal{A}$ is covered by one of the words in the considered set. How to verify that a rejecting state is covered by a word in logarithmic space is discussed in the proof of Lemma 6. Note that since $|Q|$ is given in unary, we can compute a prime divisor p_i of $|Q|$ in logarithmic space when needed and only need to store the currently considered word $a^{|Q|/p_i}$ in binary. The described LOGSPACE-algorithm is summarized in the first case of Algorithm 3. □

General unary DFAs consist of a cycle and a potentially empty chain of states from the initial state into the cycle. If this chain is empty, the DFA is actually a permutation DFA. If the tail is non-empty, then the DFA $\mathcal{A}$ is composite if and only if $\mathcal{A}$ is 2-composite, or

not minimal (and hence 1-composite) due to Lemma 8-10 in [Jecker et al., 2020]. The criteria in [Jecker et al., 2020, Lemma 8] and [Jecker et al., 2020, Lemma 9] can obviously be checked in LOGSPACE. The remaining criteria of [Jecker et al., 2020, Lemma 10] considers unary DFAs $\mathcal{A}$ where the two preimages of the state in the cycle, connecting the cycle with the chain, are separated by the set of final states. If the preimage from the chain is rejecting and the preimage q_c from the cycle is accepting, then we can simply decompose the automaton into an automaton where the cycle is collapsed to one accepting state and into one automaton where the chain is collapsed adjusting the initial state onto the cycle. If on the other hand, the preimage from the chain is accepting and the preimage q_c from the cycle is rejecting, then [Jecker et al., 2020, Lemma 10] states that $\mathcal{A}$ is composite if and only it is 2-composite if and only if the state q_c is covered by some word w in the commutative permutation sub-automaton consisting of the cycle only. The latter case can be checked in logarithmic space as a consequence of Lemma 7 yielding in summary a proof of Theorem 7. The complete algorithm solving the Bound-Decomp problem for unary DFAs in LOGSPACE is depicted in Algorithm 3.

Function isBoundedComposite($\mathcal{A} = \langle\{a\}, Q, q_I, \delta, F\rangle$: unary DFA, integer k)
  if $\mathcal{A}$ *is permutation DFA* **then**
    foreach binaryString wordCombination $\in \{0,1\}^{\log|Q|}$ *with* $\leq k$ *ones* **do**
      /* wordCombination represents current set in $2^{\mathcal{W}}$ */
      if testWordCombination($\mathcal{A}$, wordCombination) **then return** True
        /* Set of words covering all rejecting states found */
    return False /* No covering set found */
  else
    call [Jecker et al., 2020, Algorithm 1]

Function testWordCombination($\mathcal{A} = \langle\{a\}, Q, q_I, \delta, F\rangle$: unary DFA, wordCombination: **binaryString**)
  foreach $q \in Q \setminus F$ **do**
    if not cover $(\mathcal{A}, q,$ wordCombination$)$ **then return** False /* Found state not covered by current set */
  return True

Function coverBySet($\mathcal{A} = \langle\{a\}, Q, q_I, \delta, F\rangle$: unary DFA, $q \in Q \setminus F$, wordCombination: **binaryString**)
  foreach int i *with* wordCombination$[i] \stackrel{?}{=} 1$ **do** /* Go through all $\leq k$ words in the set and test if q is covered */
    compute $p_1 := i$'th prime divisor of $|Q|$
    if cover$(\mathcal{A}, q, \delta(q, a^{|Q|/p_i}))$ **then return** True /* Function cover from Algorithm 2 */
  return False

Algorithm 3: LOGSPACE-algorithm solving the Bound-Decomp problem for unary DFAs.

5.1 Proof of Theorem 6

By Lemma 5, a commutative permutation DFA $\mathcal{A}$ is k-factor composite if and only if its rejecting states can be covered by k words. As we can suppose that covering words have size linear in $|\mathcal{A}|$ (see proof of Lemma 6), the Bound-Decomp problem is decidable in NP: we guess a set of k covering words and check in polynomial time if they cover all rejecting states.

Lemma 8. *The* Bound-Decomp *problem for commutative permutation DFAs is in NP.*

Proof. Let $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ be a commutative permutation DFA. By Lemma 5 we have that $\mathcal{A}$ is k-factor composite if and only if there are k words that, together, cover all rejecting states of $\mathcal{A}$. Recall that a word w covers a state q if $\delta(q, w) \neq q$ and for every $\lambda \in \mathbb{N}$, the state $\delta(q, u^\lambda)$ is rejecting. Since $\mathcal{A}$ is a commutative permutation DFA, for each word $w \in \Sigma^*$ with $|w| > |Q|$ there exists a word $u \in \Sigma^*$ with $|u| < |Q|$ that induces the same mapping as w, in particular, $\delta(q, w^\lambda) = \delta(q, u^\lambda)$ for all $q \in Q$, $\lambda \in \mathbb{N}$. Hence, it is sufficient to test whether the rejecting states of $\mathcal{A}$ can be covered by k words of length up to $|Q|$. As we can guess these words u_i in polynomial time and check whether they cover all rejecting states q by computing the sets $\{\delta(q, u^\lambda) \mid \lambda \leq |Q|\}$ in polynomial time, the claim follows. The procedure is summarized in Algorithm 4. □

Function isBoundedComposite(commutative permutation DFA $\mathcal{A}$, integer k)
  guess $\mathcal{W} := \{w_i \in \Sigma^{\leq |Q|} \mid i \leq k\}$
  foreach $p \in Q \setminus F$ **do**
    if not cover$(\mathcal{A}, p, \mathcal{W})$ **then return** False /* Some p not covered? */
  return True /* all p are covered */
Function cover(commutative permutation DFA $\mathcal{A}$, state p, set of words $\mathcal{W}$)
  foreach $w_i \in \mathcal{W}$ **do**
    compute $Q_{q,w_i} := \{\delta(q, w_i^\lambda) \mid \lambda \leq |Q|\}$
    if $Q_{q,w_i} \cap F = \varnothing$ **then return** True
  return False

Algorithm 4: NP-algorithm solving the Bound-Decomp problem for commutative permutation DFAs.

We show that the problem is NP-hard by a reduction from the Hitting Set problem.

Lemma 9. *The* Bound-Decomp *problem is NP-hard for commutative permutation DFAs.*

Proof. The proof goes by a reduction from the Hitting Set problem (HIT for short), known to be NP-complete [Garey and Johnson, 1979]. The HIT problem asks, given a

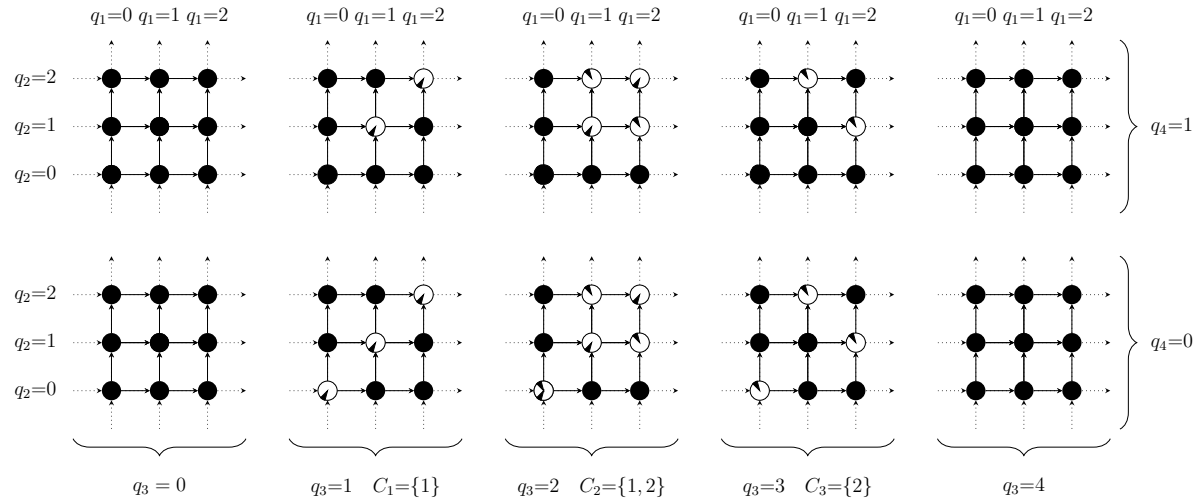


Figure 5: DFA representing the instance of HIT with $S = \{1, 2\}$ and $\mathcal{F} = \{\{1\}, \{1, 2\}, \{2\}\}$ using $\mu = 3$ and $\tau = 5$. Accepting states are filled black while rejecting states are sectored.

finite set $S = \{1, 2, \ldots, n\} \subseteq \mathbb{N}$, a finite collection of subsets $\mathcal{F} = \{C_1, C_2, \ldots, C_m\} \subseteq 2^S$, and an integer $k \in \mathbb{N}$, whether there is a subset $X \subseteq S$ with $|X| \leq k$ and $X \cap C_i \neq \varnothing$ for all $1 \leq i \leq m$. We describe how to construct a DFA $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ that is $(k+1)$-factor composite if and only if the HIT instance $\langle S, \mathcal{F}, k \rangle$ has a solution.

Automaton construction. To be constructed, the automaton $\mathcal{A}$ requires μ, τ defined as the smallest prime numbers that fulfill $n < \mu$ and $m < \tau$ and $2 < \mu < \tau$. By Bertrand's postulate [Meher and Murty, 2013], μ and τ have a value polynomial in $m + n$. The state space of $\mathcal{A}$ is defined as $Q = \{0, 1, \ldots, \mu - 1\} \times \{0, 1, \ldots, \mu - 1\} \times \{0, 1, \ldots, \tau - 1\} \times \{0, 1\}$ with $q_I = (0, 0, 0, 0)$ as initial state. Let us define the subset of states $Q_\perp = \{(q_1, q_2, q_3, q_4) \in Q \mid q_4 = 0\}$ to encode instances of HIT and the subset $Q_\top = \{(q_1, q_2, q_3, q_4) \in Q \mid q_4 = 1\}$ which is a copy of $Q_\perp$ with minor changes. The example in Figure 5 gives some intuition on the construction of $\mathcal{A}$. The DFA $\mathcal{A}$ is defined over the alphabet $\Sigma = \{a, b, c, d\}$ with the transition function defined for each state $q = (q_1, q_2, q_3, q_4)$ by $\delta(q, a) = (q_1 + 1 \mod \mu, q_2, q_3, q_4)$, $\delta(q, b) = (q_1, q_2 + 1 \mod \mu, q_3, q_4)$, $\delta(q, c) = (q_1, q_2, q_3 + 1 \mod \tau, q_4)$ and $\delta(q, d) = (q_1, q_2, q_3, q_4 + 1 \mod 2)$. Note that, $\mathcal{A}$ can be seen as a product of four prime finite fields. In particular, for every $q_3 \in \{0, \ldots, \tau - 1\}$ the subset of states $\{(x, y, q_3, 0) \in Q_\perp \mid 0 \leq x, y \leq \mu - 1\}$ can be seen as the direct product of two copies of the field of order μ (a.k.a. $\mathbb{F}_\mu$), thus inheriting the structure of a $\mathbb{F}_\mu$-vector space of origin $(0, 0, q_3, 0)$. We use these τ disjoint vector spaces to represent the collections of $\mathcal{F}$ via the acceptance of states. More precisely, each collection $C_i \in \mathcal{F}$ is encoded through the vector space $\{(x, y, i, 0) \in Q_\perp \mid 1 \leq i \leq m\}$ and each $v \in C_i$ is encoded by the non-acceptance of all states belonging to the line $\{(x, y, i, 0) \in Q_\perp \mid y = vx \mod \mu\}$. In Figure 5, each C_i is presented by an instance

of $\mathbb{F}_3 \times \mathbb{F}_3$ and each $v \in \mathcal{C}_i$ is depicted by rejecting states with the same emphasized sector. Since $\tau > m$, there are extra vector spaces for which all states are accepting i.e. $\{(q_1, q_2, q_3, 0) \in Q_\perp \mid q_3 \notin \{1, 2, \ldots, m\}\} \subseteq F$. The acceptance of states of $Q_\top$ is defined similarly as for $Q_\perp$ except that the origins of vector spaces are accepting in $Q_\top$ (see Figure 5). Formally, the rejecting states of $\mathcal{A}$ is defined by $\overline{F} = R_\perp \cup R_\top$ where $R_\perp = \{(q_1, q_2, q_3, 0) \in Q_\perp \mid q_2 = vq_1 \mod \mu, 1 \leq q_3 \leq m, v \in C_{q_3}\}$ and $R_\top = \{(q_1, q_2, q_3, 1) \in Q_\top \mid (q_1, q_2, q_3, 0) \in R_\perp, q_1 \neq 0, q_2 \neq 0\}$. All other states are accepting, i.e., we set $F = Q \setminus \overline{F}$. So, the acceptance of the subsets of states $Q_\perp$ and $Q_\top$ only differ by $O \cap Q_\perp \subseteq \overline{F}$ and $O \cap Q_\top \subseteq F$ where $O = \{(0, 0, q_3, q_4) \in Q \mid q_3 \in \{1, \ldots, m\}\}$.

The cornerstone which holds the connection between the two problems is the way the rejecting states of O can be covered. In fact, since $Q_\top$ mimics $Q_\perp$ for states in $Q \setminus O$, all rejecting states of $Q \setminus O$ can be covered by the single word $d \in \Sigma$. In addition, most words do not cover any rejecting states of $\mathcal{A}$, as stated by the following claim. Hereafter, we say that a word $w \in \Sigma^*$ is *concise* when it satisfies $\#_\sigma(w) < h_\sigma$ for all $\sigma \in \Sigma$, where $h_\sigma \in \{2, \mu, \tau\}$ is the size of the cycle induced by σ.

Claim 1. Let $u \in \Sigma^*$ be a concise word that covers some rejecting state of $\mathcal{A}$:

1. u must belong either to $\{d\}^*$ or to $\{a, b\}^* \setminus (\{a\}^* \cup \{b\}^*)$.
2. u covers some rejecting state of $Q_\top$ iff u covers *all* rejecting states of $Q_\top$ iff $u = d$.
3. u covers $(0, 0, i, 0) \in O$ iff $u \in \{a, b\}^*$ and $\#_b(u) \equiv v \cdot \#_a(u) \mod \mu$ for some $v \in C_i$.

Proof of Item 1. The statement is a direct consequence of the following:

i. Every concise word u satisfying $\#_c(u) > 0$ covers no rejecting state of $\mathcal{A}$;

ii. Every concise word $u \in \{a\}^* \cup \{b\}^*$ covers no rejecting state of $\mathcal{A}$;

iii. Every concise word u satisfying $\#_a(u) > 0$ and $\#_d(u) > 0$ covers no rejecting state of $\mathcal{A}$;

iv. Every concise word u satisfying $\#_b(u) > 0$ and $\#_d(u) > 0$ covers no rejecting state of $\mathcal{A}$.

In order to prove these four properties, we now fix a state $q = (q_1, q_2, q_3, q_4) \in Q$, and we show that, in each case, iterating a word of the corresponding form starting from q will eventually lead to an accepting state:

(i.) Let u be a concise word satisfying $\#_c(u) > 0$. Since u is concise we have $\#_c(u) < \tau$. Hence, as τ is prime, there exists $\lambda \in \mathbb{N}$ such that $\lambda \cdot \#_c(u) \equiv -q_3 \mod \tau$. Therefore the third component of $\delta(q, u^\lambda)$ is 0, thus it is an accepting state of $\mathcal{A}$.

(ii.) Let $u \in \{a\}^*$ be a concise word (if $u \in \{b\}^*$ instead, the same proof works by swapping the roles of q_1 and q_2). Since u is concise we have $0 < \#_a(u) < \mu$. Hence, as μ is prime there exists $\lambda_1, \lambda_2 \in \mathbb{N}$ satisfying $\lambda_1 \cdot \#_a(u) \equiv -q_1 \mod \mu$ and $\lambda_2 \cdot \#_a(u) \equiv -q_1 + 1 \mod \mu$. Therefore, if $q_2 \neq 0$, we get that $\delta(q, u^{\lambda_1}) = (0, q_2, q_3, q_4)$ is an accepting state of $\mathcal{A}$, and if $q_2 = 0$, we get that $\delta(q, u^{\lambda_2}) = (1, 0, q_3, q_4)$ is an accepting state of $\mathcal{A}$.

(iii.) Let u be a concise word satisfying $\#_a(u) > 0$ and $\#_d(u) > 0$. Since μ is a prime number greater than 2, there exist $\alpha \in \mathbb{N}$ such that $\mu - 2\alpha = 1$, thus $2\alpha \equiv -1 \mod \mu$. Moreover, since u is concise we have $\#_d(u) = 1$ and $\#_a(u) < \mu$. Hence there exists $\beta \in \mathbb{N}$ such that $\beta \cdot \#_a(u) \equiv 1 \mod \mu$. Therefore, if we let $\lambda = 2\alpha\beta q_1 + \mu(1 - p_4)$, we get

$$\begin{aligned} \#_a(u^\lambda) &= 2\alpha \cdot \beta \#_a(u) \cdot q_1 + \mu(1 - p_4) \cdot \#_a(u) \equiv -q_1 \mod \mu; \\ \#_d(u^\lambda) &= 2\alpha\beta q_1 + \mu \cdot (1 - p_4) \equiv p_4 + 1 \mod 2; \end{aligned}$$

As a consequence, the first component of $\delta(q, u^\lambda)$ is 0 and its fourth component is 1, hence it is an accepting state of $\mathcal{A}$.

(iv.) Let u be a concise word satisfying $\#_b(u) > 0$ and $\#_d(u) > 0$. Then we can prove that u does not cover q as in point (3), by swapping the roles of q_1 and q_2. □

Proof of Item 2. First, remark that d is the only concise word of $\{d\}^*$. By construction of $\mathcal{A}$, we have $(q_1, q_2, q_3, 0) \in F$ if and only if $(q_1, q_2, q_3, 1) \in F$ holds for all $(q_1, q_2, q_3, q_4) \in Q \setminus O$. Thus, for all $(q_1, q_2, q_3, q_4) \in \overline{F} \setminus O$ we have

$$\{\delta((q_1, q_2, q_3, q_4), d^\lambda) \mid \lambda \in \mathbb{N}\} = \{(q_1, q_2, q_3, x) \mid x \in \{0, 1\}\} \subseteq \overline{F}.$$

Hence, if $u = d$ then u covers all rejecting states of of $Q_\top$.

Now suppose that $u \in \Sigma^*$ covers some rejecting state $q = (q_1, q_2, q_3, 1) \in Q_\top$. By Item (1.), either $u \in \{d\}^*$ or $u \in \{a, b\}^* \setminus (\{a\}^* \cup \{b\}^*)$. We show that $u \in \{d\}^*$, by supposing that $\#_a(u) > 0$ and deriving a contradiction. Since μ is prime, there exists $\lambda \in \mathbb{N}$ satisfying $\lambda \cdot \#_a(u) \equiv -q_1 \mod \mu$. Therefore the first component of $\delta(q, u^\lambda)$ is 0 and its fourth component is 1, hence it is accepting, which contradicts the assumption that u covers q. □

Proof of Item 3. Consider a rejecting state $q = (0, 0, i, 0) \in O$. First, remark that no word in $\{d\}^*$ covers q since $(0, 0, i, 1)$ is accepting. Therefore, by Item (1.), the only

concise words that can cover q are the words $u \in \{a,b\}^* \setminus (\{a\}^* \cup \{b\}^*)$. For such a word u, since μ is prime, by Bezout's identity there exists $0 < v < \mu$ satisfying $\#_b(x) \equiv v \cdot \alpha\#_a(x) \mod \mu$, hence

$$\{\delta((0,0,i,0),u^\lambda) \mid \lambda \in \mathbb{N}\} = \{(q_1,q_2,i,0) \in Q \mid q_2 \equiv vq_1 \mod \mu\}.$$

If $v \in C_i$, all the states in this set are rejecting, thus u covers $(0,0,i,0)$, but if $v \notin C_i$, all these states except from $(0,0,i,0)$ are accepting, thus u does not cover $(0,0,i,0)$. □

We finally conclude the proof of Lemma 9 by proving that the sets of the initial instance of HIT are hitting if and only if the automaton $\mathcal{A}$ is composite.

If sets are hitting then the automaton is composite. Due to Lemma 5, we can show that $\mathcal{A}$ is $(k+1)$-factor composite by finding $(k+1)$ words, namely $w_\top, w_1, w_2, \ldots, w_k$, which all together cover all the rejecting states of $\mathcal{A}$. From the HIT solution $X = \{v_1, v_2, \ldots, v_k\} \subseteq S$, we define $w_j = ab^{v_j}$ for all $1 \leq j \leq k$. We prove now that for all $1 \leq i \leq m$, the rejecting state $(0,0,i,0) \in O$ is covered by some w_j. Since $X \cap C_i \neq \varnothing$, there exists $v_j \in X \cap C_i$. Moreover, by definition of w_j, we have $w_j \in \{a,b\}^*$ and $\#_b(w_j) \equiv v_j \cdot \#_a(w_j) \mod \mu$. Therefore, by Claim 1.3, $(0,0,i,0)$ is covered by w_j. Finally, we take $w_\top = d$ which covers all rejecting states $\overline{F} \setminus O$ by Claim 1.2.

If the automaton is composite then the sets are hitting. Suppose that $\mathcal{A}$ is $(k+1)$-factor composite. Hence, by Lemma 5, there exists a set W of at most $k+1$ words such that all rejecting states of $\mathcal{A}$ can be covered by some $w \in W$. In addition, we assume that each $w \in W$ is concise: if this is not the case, we can remove the superfluous letter to obtain a concise words that cover the same rejecting states. As a consequence of Claim 1.2, to cover the rejecting states of $Q_\top$, the set W needs the word d, thus W contains at most k words in $\{a,b\}^*$. Moreover, by Claim 1.3, for every $1 \leq i \leq m$, to cover $(0,0,i,0) \in O$ the set W needs a word $u_i \in \{a,b\}^*$ satisfying $\#_b(u_i) \equiv v_i \cdot \#_a(u_i) \mod \mu$ for some $v_i \in C_i$. To conclude, we construct $X = \{v_i \mid 1 \leq i \leq m\}$ which is a solution since $|X| \leq k$ due to $W \cap \{d\}^* \neq \varnothing$, and for each $C \in \mathcal{F}$ we have $X \cap C \neq \varnothing$. □

6 Discussion

We introduced in this work powerful techniques to treat the DECOMP problem for permutation DFAs. We now discuss how they could help solving the related questions that remain open:

- How do the insights obtained by our results translate to the general setting?
- How can we use our techniques to treat other variants of the DECOMP problem?

Solving the general setting. The techniques presented in this paper rely heavily on the group structure of transition monoids of permutation DFAs, thus cannot be used directly in the general setting. They still raise interesting questions: Can we also obtain an FPT algorithm with respect to the number of rejecting states in the general setting? Some known results point that bounding the number of states is not as useful in general as it is for permutation DFAs: while it is known that every permutation DFA with a single rejecting state is prime [Kupferman and Mosheiff, 2015], there exist (non-permutation) DFAs with a single rejecting state that are composite. However, we still have hope to find a way to adapt our techniques: maybe, instead of trying to cover rejecting *states*, we need to cover rejecting *behaviors* of the transition monoid. Another way to improve the complexity in the general setting would be to bound the *width* of DFAs: we defined here a family of DFAs with polynomial width, do there exist families with exponential width? If this is not the case (i.e., every composite DFA has polynomial width), we would immediately obtain a PSPACE algorithm for the general setting.

Variants of the DECOMP problem. In this work, we focused on the BOUND-DECOMP problem, that limits the *number* of factors in the decompositions. Numerous other restrictions can be considered. For instance, the FRAGMENTATION problem bounds the *size* of the factors: Given a DFA $\mathcal{A}$ and $k \in \mathbb{N}$, can we decompose $\mathcal{A}$ into DFAs of size smaller than k? Another interesting restriction is proposed by the COMPRESSION problem, that proposes a trade-off between limiting the size and the number of the factors: given a DFA $\mathcal{A}$, can we decompose $\mathcal{A}$ into DFAs $(\mathcal{A}_i)_{1 \leq i \leq k}$ satisfying $\Sigma_{i=1}^{n} |\mathcal{A}_i| < |\mathcal{A}|$? How do these problems compare to the ones we studied? We currently conjecture that the complexity of the FRAGMENTATION problem matches the DECOMP problem, while the complexity of the COMPRESSION problem matches the BOUND-DECOMP problem: for commutative permutation DFAs, the complexity seems to spike precisely when we limit the number of factors.

References

[Baier and Katoen, 2008] Baier, C. and Katoen, J. (2008). *Principles of Model Checking*. MIT Press.

[Clarke et al., 1991] Clarke, E. M., Long, D. E., and McMillan, K. L. (1991). A language for compositional specification and verification of finite state hardware controllers. *Proceedings of the IEEE*, 79(9):1283–1292.

[de Roever et al., 1998] de Roever, W. P., Langmaack, H., and Pnueli, A., editors (1998). *Compositionality: The Significant Difference, International Symposium, COMPOS'97, Bad Malente, Germany, September 8-12, 1997. Revised Lectures*, volume 1536 of *Lecture Notes in Computer Science*. Springer.

[Garey and Johnson, 1979] Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA.

[Gould et al., 2007] Gould, S., Peltzer, E., Barrie, R. M., Flanagan, M., and Williams, D. (2007). Apparatus and method for large hardware finite state machine with embedded equivalence classes. US Patent 7,180,328.

[Hardy, 1929] Hardy, G. H. (1929). An introduction to the theory of numbers. *Bulletin of the American Mathematical Society*, 35(6):778–818.

[Jecker et al., 2020] Jecker, I., Kupferman, O., and Mazzocchi, N. (2020). Unary prime languages. In Esparza, J. and Král, D., editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPIcs*, pages 51:1–51:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Kunc and Okhotin, 2013] Kunc, M. and Okhotin, A. (2013). Reversibility of computations in graph-walking automata. In Chatterjee, K. and Sgall, J., editors, *Mathematical Foundations of Computer Science 2013 - 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings*, volume 8087 of *Lecture Notes in Computer Science*, pages 595–606. Springer.

[Kupferman and Mosheiff, 2015] Kupferman, O. and Mosheiff, J. (2015). Prime languages. *Information and Computation*, 240:90–107.

[Landauer, 1961] Landauer, R. (1961). Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191.

[Meher and Murty, 2013] Meher, J. and Murty, M. R. (2013). Ramanujan's proof of Bertrand's postulate. *The American Mathematical Monthly*, 120(7):650–653.

[Netser, 2018] Netser, A. (2018). Decomposition of safe languages. Amirim Research Project report from the Hebrew University.

[Pedroni, 2013] Pedroni, V. A. (2013). *Finite State Machines in Hardware: Theory and Design (with VHDL and SystemVerilog)*. The MIT Press.

[Pin, 1992] Pin, J. (1992). On reversible automata. In Simon, I., editor, *LATIN '92, 1st Latin American Symposium on Theoretical Informatics, São Paulo, Brazil, April 6-10, 1992, Proceedings*, volume 583 of *Lecture Notes in Computer Science*, pages 401–416. Springer.

Index