Computational Complexity of Synchronization under Regular Constraints

Henning Fernau
Fachbereich 4 - Abteilung Informatikwissenschaften, Universität Trier, Germany
fernau@uni-trier.de

Vladimir V. Gusev
Leverhulme Research Centre for Functional Materials Design, University of Liverpool, UK
https://www.liverpool.ac.uk/chemistry/staff/vladimir-gusev/
vladimir.gusev@liverpool.ac.uk

Stefan Hoffmann
Fachbereich 4 - Abteilung Informatikwissenschaften, Universität Trier, Germany
hoffmanns@uni-trier.de

Markus Holzer
Institut für Informatik, Universität Gießen, Germany
holzer@informatik.uni-giessen.de

Mikhail V. Volkov
Institute of Natural Sciences and Mathematics, Ural Federal University, Yekaterinburg, Russia
http://csseminar.imkn.urfu.ru/volkov/
m.v.volkov@urfu.ru

Petra Wolf
Fachbereich 4 - Abteilung Informatikwissenschaften, Universität Trier, Germany
wolfp@uni-trier.de

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.

2012 ACM Subject Classification Theory of computation → Regular languages; Theory of computation → Problems, reductions and completeness

Keywords and phrases Finite automata, synchronization, computational complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.63

Funding The financial support of the last two authors by the DFG-funded project FE560/9-1 is gratefully acknowledged. V. V. Gusev is supported by the Leverhulme Trust. M. V. Volkov is supported by the Ministry of Science and Higher Education of the Russian Federation, project no. 1.580.2016, and the Competitiveness Enhancement Program of Ural Federal University.

Acknowledgements This project started during the workshop "Modern Complexity Aspects of Formal Languages" that took place at Trier University on February 11–15, 2019.

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 [24, 28], as well as to Chapter 13 in [17], 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 [8]. 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 [25, 27].

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 [23]. Instead of considering deterministic finite automata (DFAs), one could alternatively study the notion of synchronizability for nondeterministic finite automata [12, 21]. The notion of synchronizability naturally transfers to partially defined transition functions where a synchronizing automata avoiding undefined transitions is called *carefully synchronizing*, see [9, 20, 21]. To capture more adaptive variants of synchronizing words, synchronizing strategies have been introduced in [19]. Recall that the question of synchronizability (without length bounds) is solvable in polynomial time for complete DFAs [28]. 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 [13] 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., [24, 28]. Also, it contrasts the complexity of synchronizability for partial DFAs, which is PSPACE-complete; see [21].

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.

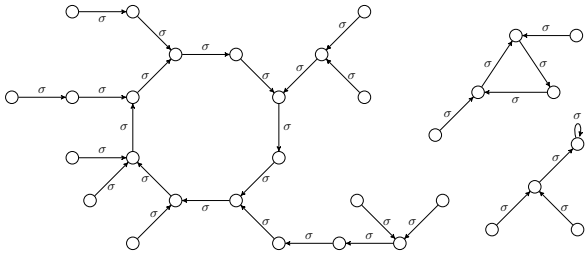
Rystsov [23] considered a problem that he called Global Inclusion Problem for Non-initial Automata. As we will see below, this problem (together with a variation) will be the key problem in our reductions. Looking at the proof of [23, Theorem 2.1], we can observe the following refined result. We consider 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 2.** *$\mathcal{P}_\Sigma$ is NP-hard if $|\Sigma| = 2$, and PSPACE-hard if $|\Sigma| > 2$.*

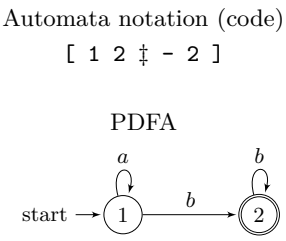
In particular, the case distinction between binary input alphabets and larger input alphabets (concerning hardness results) comes from the fact that the reduction of Rystsov uses DFA-Intersection Nonemptiness, the non-emptiness of intersection problem for deterministic finite automata on the alphabet $\Sigma \setminus \{a\}$, using Theorem 6.1 in [26] (more details in [11, 18]). 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 19.

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),



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



(b) Automata notation and PDFA using its standard interpretation.

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

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.** *[28] 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 3.** *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 with $A_{\Sigma'}$ 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 sizes;

see [3, 10, 14]. In order to address the presented automata in a compact way, we introduce a short notation motivated by [1, 2, 22], 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.
For a fixed PDFA $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$, we define the *constrained synchronization problem*:

▶ **Definition 4.** $L(\mathcal{B})$-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 [15]. For instance, we make use of regular expressions to describe languages. We also identify singleton sets with its elements. We also make use of complexity classes like P, NP, or PSPACE. At one point, we also mention the parameterized complexity class XP. With $\leq_m^{\log}$ 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 [6], which are PSPACE-complete problems for at least binary alphabets, also see [23, 24].

▶ **Definition 5.** 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 6.** 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 7. The terminology is not homogeneous in the literature. For instance, Sync-Into-Subset has different names in [6] and in [23].

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 8.** *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* $|w| \leq |Q| - 1$.

▶ **Corollary 9.** *If* $L(\mathcal{B}) \subseteq \{\sigma\}^*$ *for a PDFA* $\mathcal{B}$*, then* $L(\mathcal{B})$-Constr-Sync $\in P$.

▶ **Theorem 10.** *If* L *is regular, then* L-Constr-Sync *is contained in PSPACE.*

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

▶ **Lemma 11.** *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.*

▶ **Corollary 12.** *$L(\mathcal{B})$-Constr-Sync $\in$ P for every one-state constraint automaton $\mathcal{B}$.*

▶ **Lemma 13.** *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.*

▶ **Theorem 14.** *Let $L \subseteq \Sigma^*$. If $\{\, v \in \Sigma^* \mid \exists u, w \in \Sigma^* : uvw \in L \,\} = \Sigma^*$, L-Constr-Sync $\in$ P.*

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, cf. Fact 1. Assume $v \in \Sigma^*$ is a synchronizing word for A. By assumption, $uvw \in L$ for some $u, w \in \Sigma^*$. Moreover, uvw also synchronizes A. ◀

With the same type of reasoning, one can show:

▶ **Theorem 15.** *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.*

In [28], 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 16.** *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 sketch. If $u \in L(\mathcal{B})$ is a synchronizing word for A, u collapses any pair of states, but this is also true for $w = uv$ with $\mu(p_0, w) = p_0$ that must exist as $\mathcal{B}$ is returning. Repeatedly using this idea to prolong collapsing words, the somewhat more involved argument for proving the converse implication can be adapted from the unconstrained setting. ◀

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 17.** *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 $A' = A^{\leq 2}_{\widetilde{\{q_1,q_2\},Q}}$, 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 16, 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 18.** *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.*

▶ **Theorem 19.** *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 $p \in P$, 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.

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 $\delta(q, a^x) = \delta(q, a^y)$.

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 $p \in F$. ◀

Observe that our NP-algorithm was guessing at most $b(n,k) \in O((k+1)\log(nm))$ many bits 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 20.** *Under the assumptions of Theorem 19, $L(\mathcal{B})$-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 [4].

After these more general thoughts, we 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

There are already very many 2-state PDFA. We explain why we need to consider only one automaton for each automaton code listed in Tables 1 and 2. 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 21.** *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 $F \subseteq P$, either $L(\mathcal{B}_{p_0,F})$-Constr-Sync $\in$ P, or $L(\mathcal{B}_{p_0,F})$-Constr-Sync $\equiv_m^{\log} L(\mathcal{B}')$-Constr-Sync for a PDFA $\mathcal{B}' = (\Sigma, P', \mu', 1, \{2\})$.*

Hence, we only need to specify $\mathcal{B} = (\Sigma, \{1,2\}, \mu)$ in the following. 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.

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

Automaton code	Why in P?	Automaton code	Why in P?
`[ * 2 ‡ 1 * ]`	$a \in \Sigma_{2,1}$ Propos. 22, (2)	`[ 2 2 ‡ 2 - ]`	Theorem 24
`[ * 2 ‡ * 1 ]`	$b \in \Sigma_{2,1}$ Propos. 22, (2)	`[ 2 2 ‡ - 2 ]`	Isomorphic to `[ 2 2 ‡ 2 - ]`
`[ * 2 ‡ 2 2 ]`	$\Sigma_{1,1} \cup \Sigma_{1,2} = \Sigma_{2,2}$ Propos. 22, (3)	`[ {2,-} 2 ‡ - - ]`	$\Sigma_{1,1} \cup \Sigma_{2,2} = \emptyset$ Propos. 22, (4)
`[ 1 2 ‡ {-,2} - ]`	Theorem 24	`[ - 2 ‡ 2 - ]`	Theorem 24
`[ 1 2 ‡ - 2 ]`	Theorem 24	`[ - 2 ‡ - 2 ]`	$\Sigma_{1,1} \cup \Sigma_{1,2} = \Sigma_{2,2}$ Propos. 22, (3)

▶ **Proposition 22.** *If one of the following conditions hold, then* $L(\mathcal{B}_{1,\{2\}})$-Constr-Sync $\in$ *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 17). (3) Lemma 11 and Theorem 14 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 13 implies:

▶ **Lemma 23.** *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 23 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 22, 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 24.** *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 $L(\mathcal{B}_1) = a^*ba^*$. Let $a^\ell ba^m$ be some synchronizing word for A, then by Lemma 8, 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 3 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 9 and Lemma 13. ◀

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, the only automaton with a constrained synchronization problem in P not covered by the generalization results in Section 3 is `[ 1 2 - ‡ - - 2 ]`.

▶ **Theorem 25.** *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 24, 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. ◀

Table 2 Constraint automata (2 states, 3 letters) causing PSPACE-hard synchronization.

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 ]`	$(a+b)^*cc^*$

Case	Automaton code	Language
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 ]`	$a^*(b+c)(b+c)^*$

The leftover two-state automata over a ternary alphabet are listed in Table 2. For all of them, the corresponding constrained synchronization problem is PSPACE-complete (see Theorem 26). We want to point out that there is no constraint automaton 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 26.** *For each constraint automaton $\mathcal{B}$ in Table 2 the problem $L(\mathcal{B})$-Constr-Sync is PSPACE-hard.*

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, resp.). 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 $\delta(Q, w) \subseteq S$, resp.) 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', $|\delta(S, u)| = 1$.
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 ]`.

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 $w \in \{a, b\}^*$, $wc \in L(\mathcal{B})$.

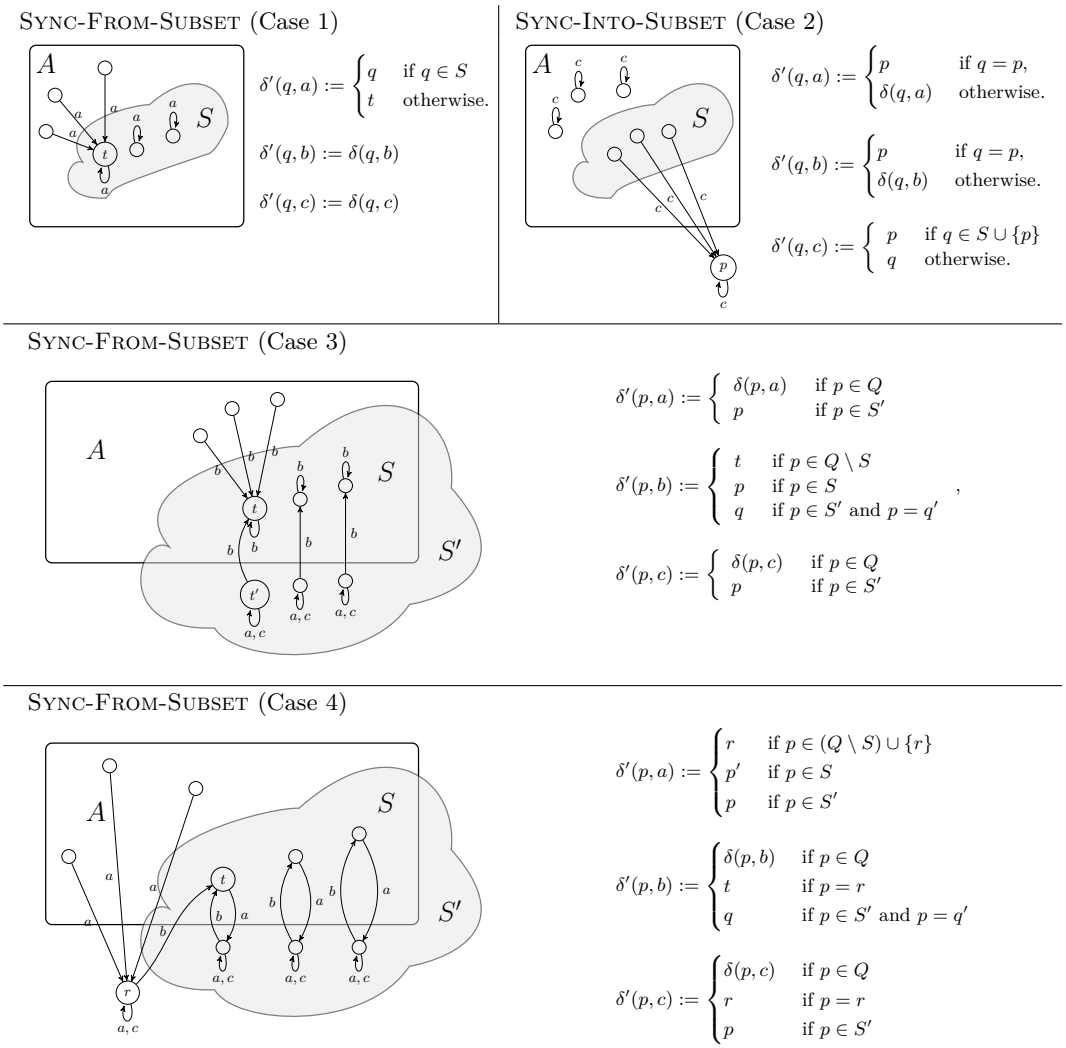


Figure 2 Schematic illustration of our reductions. Transitions inherited from A are not shown.

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

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 28. It is impossible to further generalize the previous result from homomorphisms to mappings induced by deterministic gsm. 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 29.** *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 $k := \max\{\, \ell \geq 0 \mid \exists u, v \in \Gamma^* : uc^\ell v \in \varphi(\Sigma) \,\}$. Then L-Constr-Sync $\leq_m^{\log}$ $\{c^{k+1}\}\varphi(L)$-Constr-Sync.*

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

▶ Remark 30. With the construction presented in Corollary 1 (see pp. 220-221) in [5] 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 31.** *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 $\mathcal{B}$. 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 $\mathcal{B}'$. For the other direction, we refer to the long version of this paper. ◀

As an illustration, we apply Theorem 31 to a family of languages.

▶ **Corollary 32.** *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.*

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 10.

▶ **Theorem 33.** *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. *The problem* $L(\mathcal{B})$-Constr-Sync *is PSPACE-hard.*

Hence, 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 [23] 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.

References

1 Marco Almeida, Nelma Moreira, and Rogério Reis. Enumeration and generation with a string automata representation. *Theoretical Computer Science*, 387(2):93–102, 2007. doi:10.1016/j.tcs.2007.07.029.

2 Dmitry S. Ananichev, Mikhail V. Volkov, and Vladimir V. Gusev. Primitive digraphs with large exponents and slowly synchronizing automata. *Journal of Mathematical Sciences*, 192(3):263–278, 2013. doi:10.1007/s10958-013-1392-8.

3 Frédérique Bassino and Cyril Nicaud. Enumeration and random generation of accessible automata. *Theoretical Computer Science*, 381(1-3):86–104, 2007. doi:10.1016/j.tcs.2007.04.001.

4 Marie-Pierre Béal, Mikhail V. Berlinkov, and Dominique Perrin. 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, 2011. doi:10.1142/S0129054111008039.

5 Mikhail V. Berlinkov. Approximating the minimum length of synchronizing words is hard. *Theory of Computing Systems*, 54(2):211–223, 2014. doi:10.1007/s00224-013-9511-y.

6 Mikhail V. Berlinkov, Robert Ferens, and Marek Szykuła. Complexity of preimage problems for deterministic finite automata. In Igor Potapov, Paul G. Spirakis, and James Worrell, 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, 2018. doi:10.4230/LIPIcs.MFCS.2018.32.

7 Vincent D. Blondel and Natacha Portier. 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, 2002. doi:10.1016/S0024-3795(01)00466-9.

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

9 Michiel de Bondt, Henk Don, and Hans Zantema. Lower bounds for synchronizing word lengths in partial automata. *International Journal of Foundations of Computer Science*, 30(1):29–60, 2019. doi:10.1142/S0129054119400021.

10 Michael Domaratzki, Derek Kisman, and Jeffrey Shallit. On the number of distinct languages accepted by finite automata with n states. *Journal of Automata, Languages and Combinatorics*, 7(4):469–486, 2002. doi:10.25596/jalc-2002-469.

11 Henning Fernau and Andreas Krebs. Problems on finite automata and the exponential time hypothesis. *Algorithms*, 10(1):24, 2017.

12 Zsolt Gazdag, Szabolcs Iván, and Judit Nagy-György. Improved upper bounds on synchronizing nondeterministic automata. *Information Processing Letters*, 109(17):986–990, 2009. doi:10.1016/j.ipl.2009.05.007.

13 Vladimir V. Gusev. Synchronizing automata of bounded rank. In Nelma Moreira and Rogério Reis, editors, *Implementation and Application of Automata - 17th International Conference, CIAA*, volume 7381 of *LNCS*, pages 171–179. Springer, 2012. doi:10.1007/978-3-642-31606-7_15.

14 Michael A. Harrison. A census of finite automata. *Canadian Journal of Mathematics*, 17:100–113, 1965. doi:10.4153/CJM-1965-010-9.

15 John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2nd edition, 2001.

16 Ravindran Kannan and Richard J. Lipton. Polynomial-time algorithm for the orbit problem. *Journal of the ACM*, 33(4):808–821, August 1986. doi:10.1145/6490.6496.

17 Zvi Kohavi and Niraj K. Jha. *Switching and Finite Automata Theory*. Cambridge University Press, 3rd edition, 2009.

18 Dexter Kozen. Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science, FOCS*, pages 254–266. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.16.

19 Kim Guldstrand Larsen, Simon Laursen, and Jirí Srba. Synchronizing strategies under partial observability. In Paolo Baldan and Daniele Gorla, editors, *Concurrency Theory - 25th International Conference, CONCUR*, volume 8704 of *LNCS*, pages 188–202. Springer, 2014. doi:10.1007/978-3-662-44584-6_14.

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

21 Pavel V. Martyugin. 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, 2014. doi:10.1007/s00224-013-9516-6.

22 Rogério Reis, Nelma Moreira, and Marco Almeida. On the representation of finite automata. *CoRR*, abs/0906.2477, 2009. URL: http://arxiv.org/abs/0906.2477.

23 Igor K. Rystsov. Polynomial complete problems in automata theory. *Information Processing Letters*, 16(3):147–151, 1983. doi:10.1016/0020-0190(83)90067-4.

24 Sven Sandberg. Homing and synchronizing sequences. In Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner, editors, *Model-Based Testing of Reactive Systems, Advanced Lectures*, volume 3472 of *LNCS*, pages 5–33. Springer, 2005. doi:10.1007/11498490_2.

25 Yaroslav Shitov. An improvement to a recent upper bound for synchronizing words of finite automata. Technical Report arXiv:1901.06542, Cornell University, 2019.

26 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time (preliminary report). In *Proceedings of the fifth annual ACM Symposium on Theory of Computing, STOC*, pages 1–9. ACM, 1973. doi:10.1145/800125.804029.

27 Marek Szykuła. Improving the upper bound on the length of the shortest reset word. In Rolf Niedermeier and Brigitte Vallée, 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, 2018. doi:10.4230/LIPIcs.STACS.2018.56.

28 Mikhail V. Volkov. Synchronizing automata and the Černý conjecture. In Carlos Martín-Vide, Friedrich Otto, and Henning Fernau, editors, *Language and Automata Theory and Applications, Second International Conference, LATA*, volume 5196 of *LNCS*, pages 11–27. Springer, 2008. doi:10.1007/978-3-540-88282-4_4.