On a Class of Constrained Synchronization Problems in NP*

Stefan Hoffmann

Informatikwissenschaften, FB IV, Universität Trier, Universitätsring 15, 54296 Trier, Germany, hoffmanns@informatik.uni-trier.de

Abstract. We characterize a class of constraint automata that gives constrained problems in NP, which encompasses all known constrained synchronization problems in NP so far. We call these automata polycyclic automata. The corresponding language class of polycyclic languages is introduced. We show various characterizations and closure properties for this new language class. We then give a criterion for NP-completeness and a criterion for polynomial time solvability for polycyclic constraint languages.

Keywords: finite automata · synchronization · computational complexity · polycyclic automata

1 Introduction

A deterministic semi-automaton is synchronizing if it admits a reset word, i.e., a word which leads to some definite state, regardless of the starting state. This notion has a wide range of applications, from software testing, circuit synthesis, communication engineering and the like, see [19, 21]. The famous Černý conjecture [3] states that a minimal synchronizing word has at most quadratic length. We refer to the mentioned survey articles for details. Due to its importance, the notion of synchronization has undergone a range of generalizations and variations for other automata models. It was noted in [14] that in some generalizations only certain paths, or input words, are allowed (namely those for which the input automaton is defined). In [9] the notion of constrained synchronization was introduced in connection with a reduction procedure for synchronizing automata. The paper [8] introduced the computational problem of constrained synchronization. In this problem, we search for a synchronizing word coming from some specific subset of allowed input sequences. For further motivation and applications we refer to the aforementioned paper [8]. Let us mention that restricting the solution space by a regular language has also been applied in other areas, for example to topological sorting [1], solving word equations [4, 5], constraint programming [15], or shortest path problems [17]. In [8] it was shown that the smallest partial constraint automaton for which the problem becomes PSPACE-complete has two states and a ternary alphabet. Also, the smallest constraint automaton for which the problem is NP-complete needs three states and a binary alphabet. A complete classification of the complexity landscape for constraint automata with two states and a binary or ternary alphabet was given in [8]. In [11] the result for two-state automata was generalized to arbitrary alphabets, and a complexity classification for special three-state constraint automata over a binary alphabet was given. As shown in [10], for regular commutative constraint languages we only find constrained problems that are NP-complete, PSPACE-complete, or solvable in polynomial time. In all the mentioned work [8, 10, 11], it was noted that the constraint automata for which the corresponding constrained synchronization problem is NP-complete admit a special form, which we generalize in this work.

Our contribution: Here, we generalize a theorem from [8] to give a wider class of constrained synchronization problems in NP. As noted in [11], the constraint automata that yield problems in NP admit a special form and our class encompasses all known cases of constrained problems in NP. We also give a characterization that this class is given precisely by those constraint automata whose strongly connected components are single cycles. We call automata of this type *polycyclic*. Then we introduce the language class of polycyclic languages. We show that this class is closed under union, quotients, concatenation and also admits certain robustness properties with respect to different definitions by partial or nondeterministic automata. Lastly, we also give a criterion for our class that yields constrained synchronization problems that are NP-complete and a criterion for problems in P.

2 Preliminaries and Definitions

By $\mathbb{N} = \{0, 1, 2, \ldots\}$ we denote the natural numbers, including zero. Throughout the paper, we consider *deterministic finite automata (DFAs)*. Recall that a DFA $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$, 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 call $\mathcal{A}$ *complete* if δ is defined for every $(q, a) \in Q \times \Sigma$; if δ is undefined for some (q, a), the automaton $\mathcal{A}$ is called *partial*. If $|\Sigma| = 1$, we call $\mathcal{A}$ a *unary* automaton. The set $L(\mathcal{A}) = \{\, w \in \Sigma^* \mid \delta(q_0, w) \in F \,\}$ denotes the language accepted by $\mathcal{A}$. A *semi-automaton* is a finite automaton without a specified start state and with no specified set of final states. 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 $\mathcal{A}$ or change them in a DFA $\mathcal{A}$, we use the notation $\mathcal{A}_{r,S}$. A *nondeterministic finite automaton (NFA)* $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, Q, \delta, s_0, F)$ where $\delta \subseteq Q \times \Sigma \times Q$ is an arbitrary relation. Hence, they generalize deterministic automata. With a nondeterministic automaton $\mathcal{A}$ we also associate the set of accepted words $L(\mathcal{A}) = \{w \in \Sigma^* \mid w \text{ labels a path from } s_0 \text{ to some state in } F\}$. We refer to [12] for a more formal treatment. In this work, when we only use the word automaton without any adjective, we always mean a deterministic automaton. An automaton $\mathcal{A}$ 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 $\mathcal{A}$. 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 $\delta(Q, w) = S$. We call an automaton *initially connected*, if every state is reachable from the start state.

Fact 1 *[21] For any DCSA, we can decide if it is synchronizing in polynomial time $O(|\Sigma||Q|^2)$. Additionally, we can compute a synchronizing word of length at most $O(|Q|^3)$ in time $O(|Q|^3 + |Q|^2|\Sigma|))$.*

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

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

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

Decision Problem 1: [8] $L(\mathcal{B})$-Constr-Sync
Input: Deterministic complete semi-automaton $\mathcal{A} = (\Sigma, Q, \delta)$.
Question: Is there a synchronizing word $w \in \Sigma^*$ for $\mathcal{A}$ with $w \in L(\mathcal{B})$?

The automaton $\mathcal{B}$ will be called the *constraint automaton*. If an automaton $\mathcal{A}$ is a yes-instance of $L(\mathcal{B})$-Constr-Sync we call $\mathcal{A}$ *synchronizing with respect to* $\mathcal{B}$. Occasionally, we do not specify $\mathcal{B}$ and rather talk about L-Constr-Sync. We assume the reader to have some basic knowledge in computational complexity theory and formal language theory, as contained, e.g., in [12]. For instance, we make use of regular expressions to describe languages, or use many-one polynomial time reductions. We write ε for the empty word, and for $w \in \Sigma^*$ we denote by $|w|$ the length of w. For some language $L \subseteq \Sigma^*$, we denote by $\mathrm{Pref}(L) = \{w \mid \exists u \in \Sigma^* : wu \in L\}$, $\mathrm{Suff}(L) = \{w \mid \exists u \in \Sigma^* : uw \in L\}$ and $\mathrm{Fact}(L) = \{w \mid \exists u, v \in \Sigma^* : uwv \in L\}$ the set of *prefixes*, *suffixes* and *factors* of words in L. The language L is called *prefix-free* if for each $w \in L$ we have $\mathrm{Pref}(w) \cap L = \{w\}$. If $u, w \in \Sigma^*$, a prefix $u \in \mathrm{Pref}(w)$ is called a *proper prefix* if $u \neq w$. For $L \subseteq \Sigma^*$ and $u \in \Sigma^*$, the language $u^{-1}L = \{w \in \Sigma \mid uw \in L\}$ is called a *quotient* (of L by u). We identify singleton sets with its elements. And we make use of complexity classes like P, NP, or PSPACE. A trap (or sink) state in a (semi-)automaton $\mathcal{A} = (\Sigma, Q, \delta)$ is a state $q \in Q$ such that $\delta(q, x) = q$ for each $x \in \Sigma$. If a synchronizable automaton admits a sink state, then this is the only state to which we could synchronize every other state, as it could only map to itself. For an automaton $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$, we say that two states $q, q' \in Q$ are connected, if one is reachable from the other, i.e., we have a word $u \in \Sigma^*$ such that $\delta(q, u) = q'$. A subset $S \subseteq Q$ of states is called *strongly connected*, if all pairs from S are connected. A maximal strongly connected subset is called a *strongly connected component*. By combining Proposition 3.2 and Proposition 5.1 from [7], we get the next result.

Lemma 2. *For any automaton $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$ and any $p \in P$, we have $L(\mathcal{B}_{p,\{p\}}) = C^*$ for some regular prefix-free set $C \subseteq \Sigma^*$.*

We will also need the following combinatorial lemma from [20].

Lemma 3. *[20] Let $u, v \in \Sigma^*$. If $u^m = v^n$ and $m \geqslant 1$, then u and v are powers of a common word.*

In [2,13] the decision problem SetTransporter was introduced. In general it is PSPACE-complete.

Decision Problem 2: [2,13] SetTransporter
Input: DCSA $\mathcal{A} = (\Sigma, Q, \delta)$ and two subsets $S, T \subseteq Q$.
Question: Is there a word $w \in \Sigma^*$ such that $\delta(S, w) \subseteq T$?

We will only use the following variant, which has the same complexity.

Decision Problem 3: DisjointSetTransporter
Input: DCSA $\mathcal{A} = (\Sigma, Q, \delta)$ and two subsets $S, T \subseteq Q$ with $S \cap T = \emptyset$.
Question: Is there a word $w \in \Sigma^*$ such that $\delta(S, w) \subseteq T$?

Proposition 1. *The problems* SetTransporter *and* DisjointSetTransporter *are equivalent under polynomial time many-one reductions.*

We will use Problem 3 for unary input DCSAs.

Proposition 2. *For unary DCSAs problem* SetTransporter *is NP-complete.*

In [8], with Theorem 1, a sufficient criterion was given when the constrained synchronization problem is in NP.

Theorem 1. *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 $L(\mathcal{B}_{p,\{p\}}) \subseteq \{\sigma\}^*$.*

3 Results

First, in Section 3.1, we introduce polycyclic automata and generalize Theorem 1, thus widening the class for which the problem is contained in NP. Then, in Section 3.2, we take a closer look at polycyclic automata. We determine their form, show that they admit definitions by partial and by nondeterministic automata and prove some closure properties. In Section 3.3 we state a general criterion that gives a polynomial time solvable problem. Then, in Section 3.4, we give a sufficient criterion for constraint languages that give NP-complete problems, which could be used to construct polycyclic constraint languages that give NP-complete problems.

3.1 A Sufficient Criterion for Containment in NP

The main result of this section is Theorem 2. But first, let us introduce the class of polycyclic partial automata.

Definition 1 (polycyclic PDFA). *A PDFA $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$ is called* polycyclic, *if for all states $p \in P$ we have $L(\mathcal{B}_{p,\{p\}}) \subseteq \{u_p\}^*$ for some $u_p \in \Sigma^*$.*

The results from Section 3.2 will give some justification why we call these automata polycyclic. In Definition 1, languages that are given by automata with a single final state, which equals the start state, occur. Our first Lemma 4 determines the form of these languages, under the restriction in question, more precisely. Note that in any PDFA $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$ we have either that $L(\mathcal{B}_{p,\{p\}})$ is infinite or $L(\mathcal{B}_{p,\{p\}}) = \{\varepsilon\}$.

Lemma 4. *Let $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$ be a PDFA. Suppose $p \in P$ such that $L(\mathcal{B}_{p,\{p\}}) \subseteq \{u_p\}^*$ for some $u_p \in \Sigma^*$. Then $L(\mathcal{B}_{p,\{p\}}) = \{u_p^n\}^*$ for some $n \geqslant 1$.*

Now, we are ready to state the main result of this section.

Theorem 2. *Let $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$ be a polycyclic partial automaton. Then $L(\mathcal{B})$-*Constr-Sync *$\in$ NP.*

Proof. By Lemma 4, we can assume $L(\mathcal{B}_{p,\{p\}}) = \{u_p\}^*$ for some $u_p \neq \varepsilon$ for each $p \in P$ such that $L(\mathcal{B}_{p,\{p\}}) \neq \{\varepsilon\}$. Let $U = \{u_p \mid L(\mathcal{B}_{p,\{p\}}) = \{u_p\}^* \text{ with } u_p \neq \varepsilon \text{ for some } p \in P\}$ be all such words. Set $m = |P|$. Every word of length greater than $m-1$ must traverse some cycle. Therefore, any word $w \in L(\mathcal{B})$ can be partitioned into at most $2m-1$ substrings $w = u_{p_1}^{n_1} v_1 \cdots u_{p_{m-1}}^{n_{m-1}} v_{m-1} u_{p_m}^{n_m}$ for numbers $n_1, \ldots, n_m \geqslant 0$, $p_1, \ldots, p_m \in P$ and words $v_1, \ldots, v_{m-1}$. Note that $|v_i| \leqslant m-1$ for all $i \leqslant m-1$. Let $\mathcal{A} = (\Sigma, Q, \delta)$ be a yes-instance of $L(\mathcal{B})$-Constr-Sync. Let $w \in L(\mathcal{B})$ be a synchronizing word for $\mathcal{A}$ partitioned as mentioned above.

Claim 1: If for some $i \leqslant m$ we have $n_i \geqslant 2^{|Q|}$, then we can replace it by some $n_i' < 2^{|Q|}$, yielding a word $w' \in L(\mathcal{B})$ that synchronizes $\mathcal{A}$. This could be seen by considering the non-empty subsets

$$\delta(Q, u_{p_1}^{n_1} v_1 \cdots u_{p_{j-1}}^{n_{j-1}} v_{j-1} u_{p_j}^{k})$$

for $k = 0, 1, \ldots, n_i$. If $n_i \geqslant 2^{|Q|}$, then some such subsets appears at least twice, but then we can delete the power of u_{p_i} between those appearances.

We will now show that we can decide whether $\mathcal{A}$ 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 arithmetic, and by using a more compact representation for a synchronizing word. We will assume we have some numbering of the states, hence the p_i are numbers. Then, instead of the above form, we will represent a synchronizing word in the form $w_{code} = 1^{p_1} \# \operatorname{bin}(n_1) v_1 1^{p_2} \# \operatorname{bin}(n_2) v_2 \ldots v_{m-1} 1^{p_m} \# \operatorname{bin}(n_m)$, where # is some new symbol that works as a separator, and similarly $\{0, 1\} \cap \Sigma = \emptyset$ are new symbols to write down the binary number, or the unary presentation of p_i, indicating which word u_{p_i} is to be repeated. As $\operatorname{bin}(n_i) \leqslant |Q|$ by the above claim and m is fixed by the problem specification, the length of w_{code} is polynomially bounded, and we use nondeterminism to guess such a code for a synchronizing word.

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

Proof (Proof of Claim 2 of Theorem 1). For each state $q \in Q$ and $u \in \Sigma^*$, we calculate its *u-orbit* $\operatorname{Orb}_u(q)$, that is, the set

$$\operatorname{Orb}_u(q) = \{q, \delta(q, u), \delta(q, u^2), \ldots, \delta(q, u^\tau), \delta(q, u^{\tau+1}), \ldots, \delta(q, u^{\tau+\ell-1})\}$$

such that all states in $\operatorname{Orb}_u(q)$ are distinct but $\delta(q, u^{\tau+\ell}) = \delta(q, u^\tau)$. 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 $|Q|$. Observe that $\operatorname{Orb}_u(q)$ includes the cycle $\{\delta(q, u^\tau), \ldots, \delta(q, u^{\tau+\ell-1})\}$. We can use this information to calculate $\delta(q, u^x)$, given a nonnegative integer x and a state $q \in Q$, as follows: (a) If $x \leqslant \tau(q)$, we can find $\delta(q, u^x) \in \operatorname{Orb}_\sigma(q)$. (b) If $x > \tau(q)$, then $\delta(q, u^x)$ lies on the cycle. Compute $y := \tau(q) + (x - \tau(q)) \pmod{\ell(q)}$. Clearly, $\delta(q, u^x) = \delta(q, u^y) \in \operatorname{Orb}_\sigma(q)$. The crucial observation is that this computation can be done in time polynomial in $|Q|$ and in $|\operatorname{bin}(x)|$. As a consequence, given $S \subseteq Q$ and $x \geqslant 0$ (in binary), we can compute $\delta(S, u^x)$ in polynomial time.

The NP-machine guesses w_{code} part-by-part, keeping track of the set S of active states of $\mathcal{A}$ and of the current state p of $\mathcal{B}$. Initially, $S = Q$ and $p = p_0$. For $i \in \{1, \ldots, m\}$, when guessing the number n_i in binary, by Claim 1 we guess $\log(n_i) \leqslant n$ many bits. By Claim 2, we can update $S := \delta(S, u_{p_i}^{n_i})$ and $p := \mu(p, u_{p_i}^{n_i})$ 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 $|v_i| \leqslant m = |P|$, which is a constant in our setting. Finally, check if $|S| = 1$ and if $p \in F$. □

Comparing Theorem 2 with Theorem 1 shows that our generalization allows entire words as a restriction instead of powers of a single letter for languages of the form $L(\mathcal{B}_{p,\{p\}})$, and these words could be different for each state.

3.2 Properties of Polycyclic Automata

Here, we look closer at polycyclic automata. We find that every strongly connected component of a polycyclic PDFA essentially consists of a single cycle, i.e, for each strongly connected component $S \subseteq P$ and $p \in S$ we have $|\{\mu(p,x) \mid x \in \Sigma,\ \mu(p,x) \text{ is defined } \} \cap S| \leqslant 1$. Hence, these automata admit a notable simple structure. We then introduce the class of polycyclic languages. In Proposition 4 we show that these languages could be characterized with accepting nondeterministic automata. This result yields closure under union.

Proposition 3. *Let $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$ be a PDFA. Then every strongly connected component of $\mathcal{B}$ is a single cycle if and only if $\mathcal{B}$ is polycyclic.*

We transfer our definition from automata to languages.

Definition 2. *A language $L \subseteq \Sigma^*$ is called* polycyclic*, if there exists a polycylic PDFA accepting it.*

Hence, we have the result that the constrained synchronization problem is in NP if the constraint language is polycyclic. By our results, if L gives a constrained synchronization problem outside of NP, then L could not be polycyclic. But we also state a simpler necessary criterion.

Lemma 5. *Let $L \subseteq \Sigma^*$ and let $a, b \in \Sigma$ be distinct letters. If we find $u \in \Sigma^*$ and $a, b \in \Sigma^+$ such that $u(a+b)^* \subseteq L$, then L is not polycyclic.*

By adding a trap state, we can convert every PDFA into a complete DFA accepting the same language. But the resulting complete DFA is not polycyclic anymore for $|\Sigma| > 1$, as the trap state has a cycle for every letter. The language $L = ab^*$ is polycyclic, but its complement $b(a+b)^* \cup a(a+b)^*a(a+b)^*$ is not polycyclic by Lemma 5. Hence, the polycyclic languages are not closed under complement, which implies that we could not have a structural characterization in terms of complete DFA without reference to the set of final states. However, we can use nondeterministic automata in the definition of polycyclic languages. We need the next lemma to prove this claim.

Lemma 6. *Let $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$ be a PDFA such that for some state $p \in P$ we have $L(\mathcal{B}_{p,\{p\}}) \subseteq v^* \cup w^*$. Then $L(\mathcal{B}_{p,\{p\}}) \subseteq u^*$ for some word $u \in \Sigma^*$.*

With Lemma 6 we can prove the next characterization by NFAs.

Proposition 4. *A language $L \subseteq \Sigma^*$ is polycyclic if and only if it is accepted by a nondeterministic automaton $\mathcal{A} = (\Sigma, Q, \delta, s_0, F)$ such that for all states $p \in Q$ we have $L(\mathcal{A}_{p,\{p\}}) \subseteq \{u_p\}^*$ for some $u_p \in \Sigma^*$.*

A useful property, which will be used in Section 3.4 for constructing examples that yield NP-complete problems, is that the class of polycyclic languages is closed under concatenation. We need the next lemma to prove this claim, which gives a certain normal form.

Lemma 7. *Let $L \subseteq \Sigma^*$ be a polycyclic language. Then, there exists an accepting polycyclic PDFA $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$ such that p_0 is not contained in any cycle, i.e., $L(\mathcal{B}_{p_0,\{p_0\}}) = \{\varepsilon\}$.*

Intuitively, for an automaton $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$ that has the form as stated in Lemma 7, we can compute its concatenation $L \cdot L(\mathcal{B})$ with another regular language $L \subseteq \Sigma^*$ by identifying p_0 with every final state of an automaton for L.

Proposition 5. *If $U, V \subseteq \Sigma^*$ are polycyclic, then $U \cdot V$ is polycyclic.*

We also have further closure properties.

Proposition 6. *The polycyclic languages are closed under union and quotients.*

Without proof, we note that polycyclic automata are a special case of solvable automata as introduced in [18]. Solvable automata are constructed out of commutative automata, and here polycyclic automata are constructed out of cycles in the same manner[1]. Without getting to technical, let us note that in abstract algebra and the theory of groups, a polycyclic group is a group constructed out of cyclic groups in the same manner as a solvable group is constructed out of commutative groups [16]. Hence, the naming supports the analogy to group theory quite well. Also, let us note that polycyclic automata have cycle rank [6] at most one, hence they have star height at most one. But they are properly contained in the languages of star height one, as shown for example by $(a + b)^*$.

3.3 Polynomial Time Solvable Cases

Here, with Proposition 7, we state a sufficient criterion for a polycyclic constraint automaton that gives constrained synchronization problems that are solvable in polynomial time. Please see Figure 1 for an example constraint automaton whose constrained synchronization problem is in P according to Proposition 7.

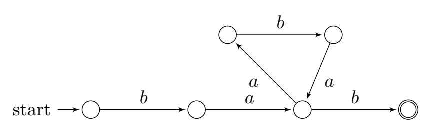

Fig. 1. An example constraint automaton with $L(\mathcal{B})$-Constr-Sync $\in$ P.

Proposition 7. *Let $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$ be a polycyclic PDFA. If for any reachable $p \in P$ with $L(\mathcal{B}_{p,\{p\}}) \neq \{\varepsilon\}$ we have $L(\mathcal{B}_{p_0,\{p\}}) \subseteq \mathrm{Suff}(L(\mathcal{B}_{p,\{p\}}))$, then the problem $L(\mathcal{B})$-Constr-Sync is solvable in polynomial time.*

[1] Solvable automata according to Rystsov [18] always have a trap state and are complete. If our partial automata are not complete, then we can make them complete by adding a trap state and the analogy is meant in this ways, where special attention has to be paid to the trap state as it is in general not a single cycle. If the polycyclic automaton happens to be complete, Rystsov's [18] definition has to be altered slightly by not demanding the lowest automaton in a composition chain to be a single state complete automaton.

Proof. By Lemma 4, we can assume $L(\mathcal{B}_{p,\{p\}}) = \{u_p\}^*$ for some $u_p \neq \varepsilon$ for each $p \in P$ such that $L(\mathcal{B}_{p,\{p\}}) \neq \{\varepsilon\}$. Let $U = \{u_p \mid L(\mathcal{B}_{p,\{p\}}) = \{u_p\}^*$ with $u_p \neq \varepsilon$ for some $p \in P\}$ be all such words. Set $m = |P|$. Every word of length greater than $m-1$ must traverse some cycle. Therefore, any word $w \in L(\mathcal{B})$ can be partitioned into at most $2m-1$ substrings $w = u_{p_1}^{n_1} v_1 \cdots u_{p_{m-1}}^{n_{m-1}} v_{m-1} u_{p_m}^{n_m}$ for numbers $n_1, \ldots, n_m \geqslant 0$, $p_1, \ldots, p_m \in P$ and words $v_1, \ldots, v_{m-1}$. Note that $|v_i| \leqslant m-1$ for all $i \leqslant m-1$. Let $\mathcal{A} = (\Sigma, Q, \delta)$ be a yes-instance of $L(\mathcal{B})$-Constr-Sync. Let $w \in L(\mathcal{B})$ be a synchronizing word for $\mathcal{A}$ partitioned as mentioned above. We show that by our assumptions we could choose the numbers $n_1, \ldots, n_m$ to be strictly smaller than $|Q|$.

<u>Claim 1</u>: If for some $i \leqslant m$ we have $n_i \geqslant |Q|$, then we can replace it by $n_i' = |Q| - 1$, yielding a word $w' \in L(\mathcal{B})$ that synchronizes $\mathcal{A}$.

Proof (Proof of Claim 1 of Proposition 7). Let $j \in \{1, \ldots, m\}$ be arbitrary with $n_j \geqslant |Q|$ and $u_{p_j} \neq \varepsilon$ (otherwise we have nothing to prove). Set $u = u_{p_1}^{n_1} v_1 u_{p_2}^{n_2} v_2 \cdots u_{p_{j-1}}^{n_{j-1}} v_{j-1}$ and $S = \delta(Q, u)$. By choice of the decomposition of w, if $n_j > 0$, then $\mu(p_0, u) = p_j$ and $L(\mathcal{B}_{p_j,\{p_j\}}) = \{u_{p_j}\}^*$. Write $p = p_j$. As by assumption u appears as a suffix of some word from $L(\mathcal{B}_{p,\{p\}})$, we have $u_p^l = vu$ for some $v \in \Sigma^*$ and $l \geqslant 0$. Hence, for each $k \in \mathbb{N}_0$ we have $\delta(S, u_p^{lk}) \subseteq S$ as every word that has u as a suffix maps any state to a state in S. Let us assume $l = 1$ in the following argument, as this is a fixed parameter of $L(\mathcal{B})$-Constr-Sync only depending on u. Hence, the conclusion would be the same if we replace u_p by u_p^l in the following arguments.
First, we show $\delta(S, u_p^{|S|}) = \delta(S, u_p^{|S|-1})$. For $q \in S$ we have $\delta(q, u_p^{|S|}) = \delta(\delta(q, u_p), u_p^{|S|-1})$ and as $\delta(q, u_p) \in S$ this gives $\delta(S, u_p^{|S|}) \subseteq \delta(S, u_p^{|S|-1})$. Now let us show the other inclusion $\delta(S, u_p^{|S|-1}) \subseteq \delta(S, u_p^{|S|})$. Let $q \in S$. By the pigeonhole principle

$$\delta(q, u_p^{|S|}) \in \{q, \delta(q, u_p), \ldots, \delta(q, u_p^{|S|-1})\}.$$

Hence $\delta(q, u_p^{|S|})$ equals $\delta(q, u_p^k)$ for some $0 \leqslant k < |S|$. Choose $a, b \geqslant 0$ such that $|S| = a(|S| - k) + b$ with $0 \leqslant b < |S| - k$. Note that $\delta(q, u_p^{k+l+a(|S|-k)}) = \delta(q, u_p^{k+l})$ for each $l \geqslant 0$ because $\delta(q, u_p^{k+|S|-k}) = \delta(q, u_p^{|S|}) = \delta(q, u_p^k)$. Set $q' = \delta(\delta(q, u_p^{|S|-1}), u_p^{|S|-k-b})$. By assumption $q' \in S$. Then

$$\begin{aligned}\delta(q', u_p^{|S|}) &= \delta(q, u_p^{|S|-1+|S|-k-b+|S|}) \\ &= \delta(q, u_p^{|S|-1+|S|-k+a(|S|-k)}) \\ &= \delta(q, u_p^{|S|-1}).\end{aligned}$$

So $\delta(q, u_p^{|S|-1}) \in \delta(S, u_p^{|S|})$. Hence, regarding our original problem, if $n_j \geqslant |Q| \geqslant |S|$, we have $\delta(Q, uu_{p_j}^{n_j}) = \delta(Q, uu_{p_j}^{|Q|-1})$, as inductively $\delta(Q, uu_{p_j}^{n_j}) = \delta(S, u_{p_j}^{n_j}) = \delta(S, u_{p_j}^{|S|-1}) = \delta(S, u_{p_j}^{|Q|-1}) = \delta(Q, uu_{p_j}^{|Q|-1})$.

So, to find out if we have any synchronizing word in $L(\mathcal{B})$, we only have to test the finitely many words

$$u_{p_1}^{n_1} v_1 \cdots u_{p_{m-1}}^{n_{m-1}} v_{m-1} u_{p_m}^{n_m}$$

for $n_1, \ldots, n_{m-1}, n_m \in \{0, 1, \ldots, |Q| - 1\}$, $u_{p_1}, \ldots, u_{p_{m-1}}, u_{p_m} \in U_p$ and words $v_1, \ldots, v_{m-1}$ of length at most m. As $m = |P|$ and U_p is fixed, we have to test $O(|Q|^m)$ many words. For each word $w = u_{p_1}^{n_1} v_1 \cdots u_{p_{m-1}}^{n_{m-1}} v_{m-1} u_{p_m}^{n_m}$, we have to read in this word starting from any state in Q and check if a unique state results, i.e., check if $\delta(q, w) = \delta(q', w)$ for $q, q' \in Q$. All these operations could be performed in polynomial time with parameter $|Q|$. □

3.4 NP-complete Cases

In [8] it was shown that for the constraint language $L = ba^*b$ and for the languages $L_i = (b^*a)^i$ with $i \geqslant 2$ the corresponding constrained synchronization problems are NP-complete. All NP-complete problems with a 3-state constraint automaton and a binary alphabet where determined in [11]. Here, with Proposition 8, we state a general scheme, involving the concatenation operator, to construct NP-hard problems. As, by Proposition 5, the polycyclic languages are closed under concatenation, this gives us a method to construct NP-complete constrained synchronization problems with polycyclic constraint languages.

Proposition 8. *Suppose we find $u, v \in \Sigma^*$ such that we can write $L = uv^*U$ for some non-empty language $U \subseteq \Sigma^*$ with*

$$u \notin \mathrm{Fact}(v^*), \quad v \notin \mathrm{Fact}(U), \quad \mathrm{Pref}(v^*) \cap U = \emptyset.$$

*Then L-*Constr-Sync *is NP-hard.*

Proof. Note that $u \notin \mathrm{Fact}(v^*)$ implies $u \neq \varepsilon$, $v \notin \mathrm{Fact}(U)$ implies $v \neq \varepsilon$ and $\mathrm{Pref}(v^*) \cap U = \emptyset$ with $U \neq \emptyset$ implies $U \cap \Sigma^+ \neq \emptyset$. We show NP-hardness by reduction from DisjointSetTransporter for unary automata, which is NP-complete by Proposition 1 and Proposition 2. Let $(\mathcal{A}, S, T)$ be an instance of DisjointSetTransporter with unary semi-automaton $\mathcal{A} = (\{c\}, Q, \delta)$. Write $v^{|u|} = x_1 \cdots x_n$ with $x_i \in \Sigma$ for $i \in \{1, \ldots, n\}$. We construct a new semi-automaton $\mathcal{A}' = (\Sigma, Q', \delta')$ with $Q' = Q \cup Q_1 \cup \ldots \cup Q_{n-1} \cup \{t\}$, where $Q_i = \{q_i \mid q \in Q\}$ are disjoint copies of Q and t is a new state that will work as a trap state in $\mathcal{A}'$. Assume $\varphi_i : Q \to Q_i$ for $i \in \{1, \ldots, n-1\}$ are bijections with $\varphi_i(q) = q_i$. Also, to simplify the formulas, set $Q = Q_0$ and $\varphi_0 : Q \to Q$ the identity map. Choose some $\hat{s} \in S$. Then, for $r \in Q'$ and $x \in \Sigma$ we define

$$\delta'(r, x) = \begin{cases} \varphi_{i+1}(q) & \exists i \in \{0, 1, \ldots, n-2\} : r \in Q_i, r = \varphi_i(q), x = x_{i+1}; \\ \delta(q, c) & r \in Q_{n-1}, r = \varphi_{n-1}(q), x = x_n; \\ \hat{s} & \exists i \in \{0, 1, \ldots, n-1\} : r \in Q_i, r = \varphi_i(q), q \notin T \cup S, x \neq x_{i+1}; \\ q & \exists i \in \{0, 1, \ldots, n-1\} : r \in Q_i, r = \varphi_i(q), q \in S, x \neq x_{i+1}; \\ t & \exists i \in \{0, 1, \ldots, n-1\} : r \in Q_i, r = \varphi_i(q), q \in T, x \neq x_{i+1}; \\ t & r = t. \end{cases}$$

Note that by construction of $\mathcal{A}'$, we have for $q \in Q \cup Q_1 \cup \ldots \cup Q_{n-1}$ and $w \in \Sigma^*$

$$\delta(q, w) = t \Leftrightarrow \exists x, y, z \in \Sigma^* : w = xyz, \delta(q, x) \in T, y \notin \operatorname{Pref}(v^{|u|}) \tag{1}$$

and for $q, q' \in Q$

$$\delta(q, c) = q' \text{ in } \mathcal{A} \ \Leftrightarrow \delta'(q, v^{|u|}) = q' \text{ in } \mathcal{A}' \ . \tag{2}$$

1. Suppose we have c^m such that $\delta(S, c^m) \subseteq T$ in $\mathcal{A}$. Because u is not a factor of $v^{2|u|}$, by construction of $\mathcal{A}'$, we have $\delta'(Q' \backslash (T \cup \{t\}), u) = S$, where S is reached as $|u| \leqslant |v^{|u|}|$. This yields $\delta'(Q \backslash (T \cup \{t\}), ua^m) \subseteq T$. By construction of $\mathcal{A}'$, $\delta'(T, x) = \{t\}$ for any $x \in U \cap \Sigma^+$, as $x \notin \operatorname{Pref}(v^*)$ and v is not a factor of x. Note that we need the last condition to ensure that we do not do a transition from a state in Q to a state in Q in $\mathcal{A}'$, for if a word does this, it must have $v^{|u|}$ as a prefix. Also $\delta'(T, u) = \{t\}$, as $0 < |u| \leqslant |v^{|u|}|$ and $u \notin \operatorname{Pref}(v^*)$ by the assumption $u \notin \operatorname{Fact}(v^*)$.
2. Suppose we have $w \in L(\mathcal{B})$ that synchronizes $\mathcal{A}'$. Then, as t is a trap state, $\delta'(Q', w) = \{t\}$. Write $uv^m x$ with $x \in U$. By construction of $\mathcal{A}'$, we have, as in the previous case, $\delta'(Q \backslash (T \cup \{t\}), u) = S$. Write $m = a|u| + b$ with $0 \leqslant b < |u|$. We argue that we must have $\delta'(S, v^{a|u|}) \subseteq T$. For assume we have $q \in S$ with $\delta'(q, v^{a|u|}) \notin T$, then $\delta'(q, v^{a|u|+b}) \in Q_{b \cdot |v|}$ by construction of $\mathcal{A}'$. As $S \cap T = \varnothing$, and hence $q \notin T$, by construction of $\mathcal{A}'$, this gives $\delta'(q, v^m x) \in S \cup Q_1 \cup \ldots \cup Q_{|v|-1}$, as x is not a prefix of v and does not contain v as a factor. More specifically, to go from $q' \in Q_{b|v|}$ to $Q_{(b+1)|v|}$, or Q in case $b + |v| = n$, we have to read v. But x does not contain v as a factor. By construction of $\mathcal{A}'$, for any $y \in \Sigma^{|v|} \backslash \{v\}$, we have $\delta'(q', y) \in S$ if $q' \in Q_{|v|b}$. This gives that if $x = x'x''x'''$ with $|x''| \leqslant |v|$ and $\delta'(q, v^m x') \in S$ we have $\delta'(q, v^m x' x'') \in S \cup Q_1 \cup \ldots \cup Q_{|v|-1}$, as after reading at most $|v|$ symbols of any factor of x, starting in a state from S, we must return to this state at least once while reading this factor. Note that by the above reasoning we find a prefix x' of x with $|x'| \leqslant |v|$ such that $\delta'(q, v^m x') \in S$ in case $|x| \geqslant |v|$. If $|x| < |v|$, then either $\delta'(q', x) \in Q_{|b|+|x|}$ or $\delta'(q', x) \in S$. So, in no case could we end up in the state t. Hence, we must have $\delta'(q, v^{a|u|}) \in T$ for each $q \in S$. By Equation (2) we get $\delta(S, c^a) \subseteq T$.

So, we have a synchronizing word for $\mathcal{A}'$ from $L(\mathcal{B})$ if and only if we can map the set S into T in $\mathcal{A}$. □

If $u, v \in \Sigma^*$ with $u \notin \operatorname{Fact}(v^*)$, by choosing $U = \{w\}$ with $w \notin \operatorname{Pref}(v) \cup \Sigma^* v \Sigma^*$ we get that uv^*w gives an NP-complete problem. Also our result shows that for example $aa(ba)^*aaa^*a$ yields an NP-complete problem.

4 Conclusion

We introduced the class of polycyclic automata and showed that for polycyclic constraint automata, the constrained synchronization problem is in NP. For these contraint automata, we have given a sufficient criterion that yields problems in P, and a criterion that yields problems that are NP-complete. However, both criteria do not cover all cases. Hence, there are still polycyclic constraint automata left for which we do not know the exact computational complexity in NP of the constrained synchronization problem. A dichotomy theorem for our class, i.e, that every problem is either NP-complete or in P, would be very interesting. However, much more interesting would be if we could find any candidate NP-intermediate problems. Lastly, we took a closer look at polycyclic automata, determined their form and also gave a characterization in terms of nondeterministic automata. We also introduced polycyclic languages and proved basic closure properties for this class.

Acknowledgement. I thank Prof. Dr. Mikhail V. Volkov for suggesting the problem of constrained synchronization 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 by the DFG-funded project FE560/9-1 is gratefully acknowledged. I also thank the anonymous reviewers for their suggestions which helped in improving the presentation.

References

1. Amarilli, A., Paperman, C.: Topological sorting with regular constraints. In: Chatzigiannakis, I., Kaklamanis, C., Marx, D., Sannella, D. (eds.) 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic. LIPIcs, vol. 107, pp. 115:1–115:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018)
2. Blondin, M., Krebs, A., McKenzie, P.: The complexity of intersecting finite automata having few final states. Comput. Complex. **25**(4), 775–814 (2016)
3. Černý, J.: Poznámka k homogénnym experimentom s konečnými automatmi. Matematicko-fyzikálny časopis **14**(3), 208–216 (1964)
4. Diekert, V.: Makanin's algorithm for solving word equations with regular constraints. Report, Fakultät Informatik, Universität Stuttgart (03 1998)
5. Diekert, V., Gutiérrez, C., Hagenah, C.: The existential theory of equations with rational constraints in free groups is PSPACE-complete. Inf. Comput. **202**(2), 105–140 (2005), `https://doi.org/10.1016/j.ic.2005.04.002`
6. Eggan, L.C.: Transition graphs and the star-height of regular events. The Michigan Mathematical Journal **10**(4), 385–397 (Dec 1963)
7. Eilenberg, S.: Automata, Languages, and Machines, Volume A. Academic Press, Inc., Orlando, FL, USA (1974)
8. Fernau, H., Gusev, V.V., Hoffmann, S., Holzer, M., Volkov, M.V., Wolf, P.: Computational complexity of synchronization under regular constraints. In: Rossmanith, P., Heggernes, P., Katoen, J. (eds.) 44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany. LIPIcs, vol. 138, pp. 63:1–63:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019), `https://doi.org/10.4230/LIPIcs.MFCS.2019.63`
9. Gusev, V.V.: Synchronizing automata of bounded rank. In: Moreira, N., Reis, R. (eds.) Implementation and Application of Automata - 17th International Conference, CIAA. LNCS, vol. 7381, pp. 171–179. Springer (2012)

10. Hoffmann, S.: Computational complexity of synchronization under regular commutative constraints (accepted for publication in COCOON 2020). CoRR **abs/2005.04042** (2020), https://arxiv.org/abs/2005.04042
11. Hoffmann, S.: Constraint synchronization with two or three state partial constraint automata. CoRR **abs/2005.05907** (2020), https://arxiv.org/abs/2005.05907
12. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 2nd edn. (2001)
13. Luks, E.M., McKenzie, P.: Parallel algorithms for solvable permutation groups. J. Comput. Syst. Sci. **37**(1), 39–62 (1988)
14. Martyugin, P.V.: Synchronization of automata with one undefined or ambiguous transition. In: Moreira, N., Reis, R. (eds.) Implementation and Application of Automata - 17th International Conference, CIAA. LNCS, vol. 7381, pp. 278–288. Springer (2012)
15. Pesant, G.: A regular language membership constraint for finite sequences of variables. In: Wallace, M. (ed.) Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3258, pp. 482–495. Springer (2004)
16. Robinson, D.J.: A Course in the Theory of Groups. Springer, 2 edn. (1995)
17. Romeuf, J.: Shortest path under rational constraint. Inf. Process. Lett. **28**(5), 245–248 (1988), https://doi.org/10.1016/0020-0190(88)90198-6
18. Rystsov, I.K.: Reset words for commutative and solvable automata. Theoretical Computer Science **172**(1-2), 273–279 (1997)
19. Sandberg, S.: Homing and synchronizing sequences. In: Broy, M., Jonsson, B., Katoen, J.P., Leucker, M., Pretschner, A. (eds.) Model-Based Testing of Reactive Systems. LNCS, vol. 3472, pp. 5–33. Springer (2005)
20. Schützenberger, M.P., Lyndon, R.C.: The equation aM=bNcP in a free group. Michigan Mathematical Journal **9**(4), 289–298 (1962)
21. Volkov, M.V.: Synchronizing automata and the Černý conjecture. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) Language and Automata Theory and Applications, 2nd Int. Conference, LATA. LNCS, vol. 5196, pp. 11–27. Springer (2008)