The Synchronization Game on Subclasses of Automata

Henning Fernau
Fachbereich IV, Informatikwissenschaften, Universität Trier, Germany

Carolina Haase
Informatik, Hochschule Trier, Germany

Stefan Hoffmann
Fachbereich IV, Informatikwissenschaften, Universität Trier, Germany

Abstract

The notion of synchronization of finite automata is connected to one of the long-standing open problems in combinatorial automata theory, which is Černý's Conjecture. In this paper, we focus on so-called synchronization games. We will discuss how to present synchronization questions in a playful way. This leads us to study related complexity questions on certain classes of finite automata. More precisely, we consider weakly acyclic, commutative and k-simple idempotent automata. We encounter a number of complexity classes, ranging from L up to PSPACE.

2012 ACM Subject Classification Theory of computation → Regular languages; Theory of computation → Complexity classes

Keywords and phrases Synchronization of finite automata, computational complexity

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.14

1 Introduction

"Gamification" is a catchword that describes, in a broader sense, how to teach or to explain possibly complicated concepts via games. In a narrower sense, we are discussing *educational games*. It is tempting to use this idea to explain ideas from combinatorics or complexity theory. For instance, one can play The Hamiltonian Circuit on one's smartphone[1]. As a predecessor, clearly Hamilton's Icosian Game is to be mentioned[2]. This has been done (not necessarily with didactical motivations) when combining ideas from game theory with graph-theoretic models of real-world phenomena, with *vertex cover games* [15] serving as one example. An alternative (but complementing) route is to analyze the combinatorial or computational complexity nature of games, which has been one of the driving themes of the FUN conference series; a certain overview on this ever-growing area can be found in [9, 16].

One drawback of many notions from combinatorics, in particular from graph theory, is that they have a rather static nature. This makes it rather difficult to define meaningful games based on these notions. The picture changes if dynamics enters the scene. One of the easiest ways to induce dynamics into graphs is when interpreting edge-labeled directed graphs as finite automata, because reading a word by a finite automaton infuses action into the otherwise static graph objects.

[1] See `https://apps.apple.com/us/app/the-hamiltonian-circuit/id1484345208`. In fact, there are also game variations of the Hamiltonian Circuit ("Maker-Breaker" games, see [24]) that are close to the ideas followed in this paper concerning combinatorial automata theory.
[2] See `https://www.puzzlemuseum.com/month/picm02/200207icosian.htm`, this has also interesting connections to quaternions, as explained in [3].

This then allows to explain important notions like synchronization, which is linked to an intriguing long-standing open combinatorial question in automata theory, namely, whether or not each synchronizable deterministic finite-state automaton possesses a so-called synchronizing word of a length quadratic in its number of states. This is (basically) also known as Černý's Conjecture.[3] A 2-person game called *synchronization game*, where Alice plays against Bob, was introduced in [13]. This game was indeed motivated by a gamification approach to software testing; see [4]. Fominykh, Martyugin and Volkov showed that whether or not Alice or Bob have a winning strategy is indeed a computationally hard question in general. In this paper, we look into this question, restricted to several classes of automata.[4]

Coming back to the idea of gamification, this game (and its analysis) can be seen as a playful doorway into a number of concepts that are prominent in Computer Science:

- graphs and automata are the most obvious concepts, as this is how a typical game would look like, although one could also think of designs that look more like classical board-games (see below) and hence obfuscate this possibly "too abstract" presentation by graphs;
- combinatorial questions and combinatorial thinking are at the heart of many mathematical reasonings in Computer Science and are also at the source of this game;
- computational complexity is another area that is touched and about which a player might get a feel, as the "game complexity" is somehow related to this concept.

The paper is structured as follows. In Section 2, we briefly describe the necessary background for the technical parts of the paper. We also shortly discuss the class of weakly acyclic finite automata as a simple example of a subclass of finite automata and also to illustrate the questions considered throughout the paper. In Section 3, we return to the question of board designs for automata games in general, which also motivates our major technical theme of this paper, which is to look at synchronization and synchronization games from the perspective of their complexity on special classes of automata. In Section 4, we give an overview over the situation in terms of complexity for general deterministic finite automata. Already here, a certain span of complexity classes from NL to PSPACE shows up. Then, in Section 5, we consider commutative automata. The span of complexity classes now contains L, P, NP, and Π_2^P. Again another class of automata is considered in Section 6: k-simple idempotent ones. Here, as a main result, we also show a full understanding about when Alice can win the synchronization game. In Section 7, we briefly discuss monotonic automata that are different from the previously considered classes in terms of complexity insofar as synronizability is different from Alice having a winning strategy on a given automaton. In the last section, we discuss several other aspects, including the question of level design for synchronization games.

2 Preliminaries

An *alphabet* is a finite, non-empty set whose elements are called *letters*. As usual, Σ^* denotes the set of all words over the alphabet Σ, including the *empty word* ε. Let $u \in \Sigma^*$ be a word and $a \in \Sigma$. By $|u|_a$ we denote the number of times the symbol a occurs in u. Let

[3] As it is often the case with famous mathematical conjectures, this question has quite some history. We are not diving into this here, but only mention that a few years ago, a special issue of a journal was dedicated to this question, see [38]; there, also English translations of the first papers of this topic can be found, which were written in Slovak and in German back in the 1960s.

[4] It should be noted that two different games have been suggested in the literature in connection with synchronization: a 1-person game [2] and a stochastic 2-person game [20], but we only discuss the synchronization game introduced in [13] in this paper.

$w = w_1 w_2 \cdots w_n$ be a word consisting of n subsequent letters. Then, n is also called the *length* of w. In this paper, a *deterministic finite automaton*, or DFA for short, is specified as $\mathscr{A} = (Q, \Sigma, \delta)$, where Σ is the input alphabet, Q is the state alphabet, and $\delta : Q \times \Sigma \to Q$ is the (total) transition function;[5] δ is then extended to $\delta : Q \times \Sigma^* \to Q$ by $\delta(q, \varepsilon) = q$ and $\delta(q, ua) = \delta(\delta(q, u), a)$ for $a \in \Sigma$. Likewise, δ is also extended to $\delta : 2^Q \times \Sigma^* \to 2^Q$. A word $w \in \Sigma^*$ is called *synchronizing* for $\mathscr{A}$ if there exists a *synchronizing state* q_{sync} so that, for each $q \in Q$, $\delta(q, w) = q_{\text{sync}}$. A DFA is called *synchronizing* if it admits a synchronizing word. A state σ is called a *sink state* if, for all input letters a, $\delta(\sigma, a) = \sigma$. Whether or not a DFA is synchronizing can be checked by considering a restricted variation of the power-set automaton. For a DFA $\mathscr{A} = (Q, \Sigma, \delta)$, its *pair automaton* is defined as $\mathcal{P}_2(\mathscr{A}) = (Q_{\mathcal{P}_2}, \Sigma, \delta_{\mathcal{P}_2})$ with state set $Q_{\mathcal{P}_2} = \{\{s, t\} | s, t \in Q \wedge s \neq t\} \cup \{\bot\} = \binom{Q}{2} \cup \{\bot\}$ and transition function

$$\delta_{\mathcal{P}_2}(\{s, t\}, a) = \begin{cases} \{\delta(s, a), \delta(t, a)\} & \text{if } \delta(s, a) \neq \delta(t, a) \\ \bot & \text{if } \delta(s, a) = \delta(t, a) \end{cases}$$

Moreover, $\delta_{\mathcal{P}_2}(\bot, a) = \bot$. In other words, $\bot$ is a sink state that is entered via symbol a whenever the two states s, t of the original automaton are synchronized when reading a. Apart from $\bot$, a pair automaton can be viewed as a sub-automaton of the well-known powerset automaton. The mentioned check is based on the following result that is well-known in the area of synchronizing automata, compare the surveys [32, 37]:

▶ **Lemma 2.1.** *$\mathscr{A} = (Q, \Sigma, \delta)$ is synchronizing if and only if for each unordered pair of states $\{s, t\} \in \binom{Q}{2}$, there exists a word $w \in \Sigma^*$ such that $\delta_{\mathcal{P}_2}(\{s, t\}, w) = \bot$.*

States $\{s, t\} \in Q_{\mathcal{P}_2}$ with $\delta_{\mathcal{P}_2}(\{s, t\}, a) = \bot$ are therefore also called *synchronizing pairs*.

Next, we describe a related 2-persons game, the *synchronization game*. There are two players, Alice (Synchronizer) and Bob (Desynchronizer), whose moves alternate. They play on a DFA $\mathscr{A} = (Q, \Sigma, \delta)$. Alice who plays first wants to synchronize $\mathscr{A}$, while Bob aims to prevent synchronization or, if synchronization is unavoidable, to delay it as long as possible. More formally, at the beginning, all states are *active*, or, more pictorially, all states hold coins. Alice starts by playing the first letter w_1. After her move, only the states in $Q_1 = \delta(Q, w_1)$ hold coins. Then, Bob plays the second letter w_2, so that then, the states in $Q_2 = \delta(Q, w_1 w_2) = \delta(Q_1, w_2)$ hold coins, etc. Alice wins if there is some t such that $|Q_t| = 1$. Then, $w_1 w_2 \cdots w_t$ is a synchronizing word, and q_t with $\{q_t\} = Q_t$ is the corresponding synchronizing state. Provided that both players play optimally, the outcome of such a game depends only on $\mathscr{A}$. But who can enforce a win, and if so, how does a winning strategy look like? This game was introduced in [13], where also several complexity results have been established, see Section 4. The following result is of particular importance when analyzing winning strategies. It clearly bears some similarities with Lemma 2.1.

▶ **Lemma 2.2** ([13]). *Alice has a winning strategy in the synchronization game on a DFA iff she has a winning strategy in every position in which only two states of the DFA hold coins.*

Therefore, we could alternatively start a synchronization game by having Bob choose two states on which he places one coin each. Alice would win if she can synchronize these two states, although Bob will try to prevent this from happening. In particular, we could say that Bob has won when he manages to have two coins on the same two states twice when he

[5] We do not need to speak about initial or final states in this paper. Sometimes, this variant of automata is also called semi-automata.

is about to move. In this paper, we are looking for further conditions under which one can tell if Alice or Bob will win the synchronization game. Therefore, we will look into special cases of the synchronization game that can be described by special classes of finite automata. Therefore, we are introducing DFAs with several special properties in the following.

Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a DFA. Extending the definition of simple idempotent letters given in [29], we call a letter $a \in \Sigma$ *k-simple idempotent* if $|\delta(Q, a)| = |Q| - k$ and $\delta(q, a) = q$ for all $q \in \delta(Q, a)$. We say that $\mathscr{A}$ is a *cyclic automaton with a k-simple idempotent* over a binary alphabet Σ if there exists a k-simple idempotent letter $a \in \Sigma$ and $b \in \Sigma$ permutes all states cyclically. States are numbered in clockwise order in relation to b. We say that two states $q, p \in Q$ are neighbors if $\delta(q, b) = p$. The *b-distance* of two states q, p is the number k of b-transitions between them, that is $k = \min\{\ell \in \mathbb{N} \mid \delta(q, b^\ell) = p \vee \delta(p, b^\ell) = q\}$.

We call $\mathscr{A} = (Q, \Sigma, \delta)$ *connected* if the undirected simple graph $G = (V, E)$ with vertex set $V = Q$ and edge set $E = \{\{p, q\} \mid p \neq q \wedge \exists a \in \Sigma : \delta(p, a) = q \vee \delta(q, a) = p\}$ is connected (note that this is different from the way connectedness is defined for automata in [5]).

Let $\mathscr{A} = (Q, \Sigma, \delta)$ and $p, q \in Q$. We say that the state q is *reachable* from p, written $p \precsim q$, if there exists a word $u \in \Sigma^*$ such that $q = \delta(p, u)$. Note that this relation yields a quasiorder on the states and it induces a partial order on the equivalence classes where two states $p, q \in Q$ are equivalent if $p \precsim q$ and $q \precsim p$. The equivalence classes are precisely the strongly connected components in the automaton graph. A DFA is called *weakly acyclic* if each $\precsim$-equivalence class contains only one element. Hence, the quasi-order $\precsim$ is in fact a partial order. This is why these automata are sometimes called *partially ordered.* In general, a DFA with a sink state σ is synchronizing if and only if the sink state is reachable from each other state; in this case, the sink state is also the synchronizing state.

We consider weakly acyclic automata as a warm-up for this paper. Synchronization problems on this class of DFA have been investigated in [18, 30]. The following result shows that the question whether or not Alice has a winning strategy is particularly easy for this type of DFA. Compare this with Theorem 5.2, which gives the same for commutative automata; we will refer to this theorem here also in the proof, as this theorem serves rather as an appetizer for the technical parts of this paper.

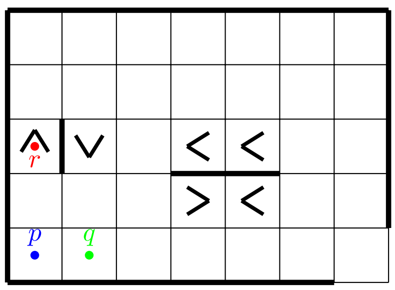
▶ **Theorem 2.3.** *For a weakly acyclic automaton $\mathscr{A} = (Q, \Sigma, \delta)$, the following are equivalent:*
1. *$\mathscr{A}$ has a unique sink state,*
2. *$\mathscr{A}$ is synchronizing,*
3. *Alice has a winning strategy on $\mathscr{A}$.*

Proof. (1) implies (2): As every maximal state is a sink state, by uniqueness, there exists a unique maximal state (with respect to $\precsim$). By definition of the reachability relation, there exists a word mapping every state to this unique maximal sink state. Then the automaton is synchronizing with the same argument as in the proof of Theorem 5.2.

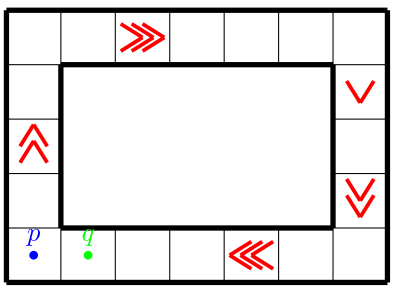
(2) implies (3): Let $\mathscr{A} = (Q, \Sigma, \delta)$ be synchronizing and weakly acyclic. By definition, there is a partial ordering on the state set $Q = \{q_1, \ldots, q_n\}$ such that $q_i \precsim q_j$ implies $i \leq j$. Then, q_n is a sink state and hence the synchronizing state. Also, there is a letter a_i and some $j_i > i$ with $\delta(q_i, a_i) = q_{j_i}$ for $i = 1, \ldots, n - 1$. Therefore, Alice can win after she made at most $n - 1$ moves, as she can shorten the distance to the sink state in each move.

(3) implies (1): If Alice has a winning strategy, there exists a synchronizing word u. Every maximal state q is a sink state, so there exists at least one. However, if $\mathscr{A}$ is synchronizing, then only one sink state can exist, as one cannot reach a sink state from another one. ◀

Hence, checking if Alice has a winning strategy is the same as checking if the DFA is synchronizing, which is in NL, while both complexities differ in general, see Theorem 4.1.



(1) A board design with $\delta(p,e) = q$ and $\delta(q,w) = p$.



(2) A racing track design

Figure 1 Designing boards for the synchronization game: two different proposals.

3 On Board Designs

How can we present a synchronization game in a compact and attractive fashion? This is the question we like to discuss in this section. Clearly, every finite automaton can be presented by a transition table or by its automaton graph. However, we claim that more traditional designs like those known from traditional board games like chess or like games with racing tracks that are also popular children's games may be more appealing than the ones usually applied in automata theory. Notice that we cannot represent every automaton this way, but notice that the proposed board designs do relate to the classes of automata studied throughout this paper.

Referring to Example (1) in Figure 1, we can interpret such a board design as a finite automaton over the input alphabet $\{e, n, s, w\}$ (with the letters denoting the east, north, south and west directions of movement) as follows:

- Each cell represents a state of the automaton.
- The input letters let us move through the board as expected: e moves one step "east", i.e., to the right, etc.
- If the move would hit a wall, indicated by a thicker drawn line, then the direction indicated by the arrow in the current state should be "executed." Therefore, in our picture, $\delta(r, wsss) = p = \delta(r, esss) = \delta(r, nsss)$.
- If a wall is hit but no arrow is drawn, then the direction is inverted, i.e., if one bumps against the wall by moving east, a west move is executed, as $\delta(p, w) = q$ and $\delta(p, sn) = r$.[6]
- An exception is the cell in the right lower corner: here the open walls indicate that this is a "way out." More formally, there is one more state to be reached this way (by moving either south or east), a sink state σ, which must be synchronizing. Hence, $\delta(q, eeeeee) = \sigma$.
- Notice that apart from the exceptions that have been worked out above, the automaton behaves as a commutative automaton, e.g., $\delta(q, nw) = \delta(q, wn)$. This is also partially true for our rules at the walls, e.g., $\delta(p, se) = \delta(p, es) = \delta(q, n)$. However, $\delta(p, sn) = \delta(p, nn) \neq \delta(p, ns) = p$. But without the walls and in particular the "special walls" built in the middle of the board, the game might appear to be a bit boring.
- Also notice that board designs often (but not necessarily) lead to automata whose graphs are planar. Also such automata have been previously studied in our setting; see [25].

[6] An alternative might be to get stuck when bumping against a wall; this would correspond to considering incomplete deterministic automata. Then, we arrive at different synchronization problems, and we like to avoid discussions in this direction in this paper.

The goal of the game is as in the general case and can also be played with just two pieces (or coins) on the board. The two players move both of them at the same time according to one of the letters a, e, n, w. While player one will try to move both coins on the same cell of the board, player two will try to prevent this from happening. To make the game more interesting, it is suitable to introduce the additional rule that is it prohibited to enter the same configuration twice.

Such a board design is quite compact: The example above denotes a 36-state DFA, whose formal transition function is clearly less intuitive than this board presentation. Also when comparing it to a classical automaton graph representation, the suggested presentation has an edge, mainly because of the implicit transition arcs and implicit arc labels. Additionally, if one tries to draw an automaton graph as small as the board can be drawn without losing readability, the letters that need to be written on the arcs will be hard to decipher.

One can also use such a board to define a single-player game as follows. First, specify three positions p, q, r on the board (an example is shown in Figure 1 on the left side, but in general you can think of special tasks that a player draws from a pile of tasks at the beginning of the game), plus a number n. The question to solve is to find a sequence of movements, i.e., a word x of length at most n over the alphabet $\{e, n, s, w\}$, such that for the transition function δ that can be associated to the board, we find that $\delta(p, x) = \delta(q, x) = r$. With the positions shown in Figure 1, it is not possible to find any such word. This is due to the fact that one can show that, assuming that the two pieces stay on the board, the Manhattan distance between the two pieces on the board (ignoring walls) will always be an odd number. In general, this specific form of a synchronizability question could be interesting as a single-player game, although it is polynomial-time solvable (without the length restriction), mainly because of facts: (a) as discussed above, boards can be used as quite compact representations of DFA; (b) although words that synchronize two states are of quadratic size only, see Lemma 2.1, this might mean that the player might have to look for a synchronizing word of a length like 100 even for the small board displayed in Figure 1. This makes this question quite challenging for a human player.

A second example is shown in Figure 1 (2). Here, the underlying automaton is a cyclic one over a binary alphabet with a k-simple idempotent. The b-cycle of such an automaton is interpreted as kind of a racing track, where each state of the automaton coresponds to a square. There are two coins in the game. Players can only move both of them at the same time and in clockwise order. The goal for player one is for both coins to end up on the same square, while the opponent tries to prevent exactly that. Resembling a game of tag. In each round players can choose between either moving both coins forward by one or, if they are positioned on a field with one or more red arrows, move the number of steps indicated by the number of arrows on the square. In this case, if there are no arrows on a square, a coin positioned on this square does not move. Obviously, this is equivalent to the synchronization game by Lemma 2.2.

Apart from being more similar to board games, this perspective on finite automata is also motivated by robot navigation problems; see [26]. Conceptually, it is interesting to note that this particular paper talks about homing sequences, a notion quite akin to that of synchronizing words, cf. the exposition of Sandberg [32]. Of course, one could think of different puzzles played on such a board: the traditional Short Synchro problem would lead to a one-person game, while the synchronization game itself is a two-persons game. Further variations (or possibly levels) can be designed by introducing further special "effects" (or board cell symbols). For instance, one could introduce cells where the directions are rather interpreted as knight moves, etc.

4 General Complexity Considerations

The concept of synchronization has been studied quite intensively also in the light of complexity theory. In the following, we assume acquaintance with the basic complexity concepts behind on side of the reader. Our main questions are the following ones:

- Synchro: Given a DFA $\mathscr{A}$, is $\mathscr{A}$ synchronizing?
- Short Synchro: Given a DFA $\mathscr{A}$ and an integer $k \geq 0$, does $\mathscr{A}$ possess a synchronizing word of length at most k?
- SynchroGame: Given a DFA $\mathscr{A}$, does Alice possess a winning strategy on $\mathscr{A}$?
- Short SynchroGame: Given a DFA $\mathscr{A}$ and an integer $k \geq 0$, can Alice enforce a win on $\mathscr{A}$ in at most k moves?

It is possible to explain four important classical complexity classes with these problems.

▶ **Theorem 4.1.** *Under logspace-reductions, we can state the following:*

- *Synchro is* NL*-complete.*
- *Short Synchro is* NP*-complete.*
- *SynchroGame is* P*-complete.*
- *Short SynchroGame is* PSPACE*-complete.*

In the following proof, we reduce from the P-complete And/or Graph Accessibility Problem, or AGAP for short, that was first introduced in [19]; we follow the presentation of Sudborough [35] as a pebble game, as it makes the connection to SynchroGame more transparent. The input is a directed acyclic graph $G = (V, E)$ and a labelling function $f : V \to \{\vee, \wedge\}$ such that there are no edges in between $\vee$-labeled and no edges in between $\wedge$-labeled vertices.[7] The rules of this pebble game are simple: (1) one can always place a pebble on a node which has no outgoing edges, and (2a) one can place a pebble on node x, when $f(x) = \wedge$ and all nodes to which an edge is directed from node x contain pebbles, and (2b) one can place a pebble on node x, when $f(x) = \vee$ and at least one node to which an edge is directed from node x contains a pebble. The question is whether a pebble can be placed on one of the nodes of indegree zero by playing the pebble game. It is possible to restrict one's attention to directed acyclic graphs that have exactly one vertex t of outdegree zero and one vertex s of indegree zero. Then, we also say that t is *reachable* from s in G. Notice that then conditions (1) and (2a) mean the same if $f(t) = \wedge$. Moreover, the labelled directed acyclic graph G can be transformed into a labelled directed acyclic bipartite graph G' that has outdegree at most two by replacing a larger number of outgoing edges by a tree. Notice that AGAP is basically equivalent to the non-emptiness problem of alternating finite automata introduced in [7, 8]

Proof. The NL-completeness of Synchro is somewhat folklore, it is based on the similarity of the condition of Lemma 2.1 to reachability in graphs. We also refer to [17, Theorem 5]. The NP-completeness of Short Synchro was shown at least three times independently from each other in the literature, with nearly identical proofs; see [27, 10, 14] in chronological order. The PSPACE-completeness of Short SynchroGame was proved in [13].

We are now proving that SynchroGame is P-complete, a result that nicely complements earlier findings, because it shows two effects that can be observed already from the previous list of results: (a) the "game variant" of the synchronization problem is harder than the classical variant; (b) the length-bounded variant is harder than the unbounded variation.

[7] The classical exposition does not require this bipartiteness condition, but it is not hard to see that bipartiteness can be enforced with a logspace machine.

The following reduction is inspired by that of [17, Theorem 5]. We start from a labelled directed acyclic bipartite graph $G = (V, E)$ that has outdegree at most two, with one vertex t of outdegree zero, labeled like $f(t) = \wedge$, and one vertex s of indegree zero. We construct a DFA $\mathscr{A} = (Q, \Sigma, \delta)$ such that Alice can win on $\mathscr{A}$ if and only if t is reachable from s in G. Let $\Sigma = \{a, b, c\}$. We can assume that s is an $\wedge$-vertex (otherwise, put a new vertex before s that is a $\wedge$-vertex and has only one transition to s). Set $Q = V \cup \{s'\}$, where $s' \notin V$. Now, for each vertex, if two edges go out, label one with a and the other one with b, and if only a single edge emanates, label this one with a and b, i.e., add two transitions, and if no edge leaves, add two self-loops for a and b. Furthermore, set $\delta(s', a) = \delta(s', b) = \delta(s', c) = s$. For every other vertex $q \in Q \setminus \{s'\}$, set $\delta(q, c) = t$ if q corresponds to a $\wedge$-vertex (where Bob has a choice later) and $\delta(q, c) = s$ if q corresponds to a $\vee$-vertex. By the choice of c, if it is Alice's turn (where she lands on a $\vee$-vertex) she never chooses c (except at the beginning to set everything to either s or t), because this would "destroy" her progress, i.e., reset to her starting position where only s and t are "active" states. If it is Bob's turn (on the $\wedge$-vertices) he also never chooses c (or at least we do not have to consider those moves in the following), as this would instantly map everything to t and Alice wins.

Recall that due to Lemma 2.2, we only need to discuss what happens if there are only two coins left in the game. In fact, this led to the discussion of a variation of the pair automaton in [13, Theorem 4] that keeps (additionally) track of whether it is Alice's turn or Bob's turn. On this automaton, a marking algorithm is presented that works exactly like the pebble game described above. The discussion in the previous paragraph shows that we need not consider this whole pair automaton for the synchronization game that we constructed, only the state pairs reachable from $\{s, t\}$ matter. However, now it can be observed that the automaton $\mathscr{A}$ described above is isomorphic to the pair automaton of $\mathscr{A}$, restricted to state pairs reachable from $\{s, t\}$. The reasoning of [13, Theorem 4] hence shows that Alice has a winning strategy on $\mathscr{A}$ if and only if t is reachable from s in G. ◀

▶ **Corollary 4.2.** *On weakly acyclic DFAs,* SYNCHRO *and* SYNCHROGAME *are NL-complete.*

Proof. By the previous theorem, checking if Alice has a winning strategy is the same as checking if the automaton is synchronizing, which is in NL. Hardness can be shown by a reduction similar as the one used in Theorem 4.1 to show P-completeness, but using the ordinary acyclic graph reachability (GAP), which is NL-complete. ◀

5 Commutative Automata

The DFA $\mathscr{A} = (Q, \Sigma, \delta)$ is *commutative*, if for each $q \in Q$ and $a, b \in \Sigma$, we have $\delta(q, ab) = \delta(q, ba)$. Every unary DFA, i.e., if $|\Sigma| = 1$, is commutative. See Figure 2 for two examples.

5.1 Combinatorial and Structural Results

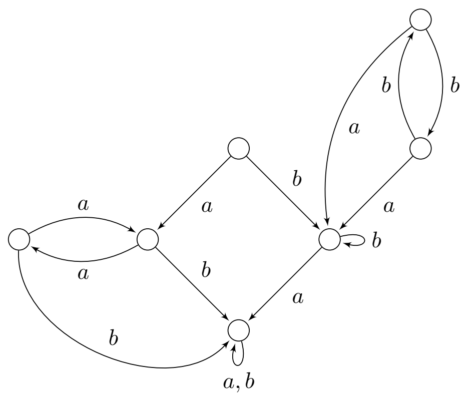
It is easy to construct connected automata with several sink states. However, for a commutative automaton, we can only have at most one sink state.

▶ **Lemma 5.1.** *Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a commutative DFA. If the DFA $\mathscr{A}$ is connected and has a sink state, then this sink state is unique and reachable from every other state.*

With Lemma 5.1, we can prove the following for synchronizing commutative DFAs.

▶ **Theorem 5.2.** *For a commutative automaton $\mathscr{A}$, the following are equivalent:*

1. *$\mathscr{A}$ is connected and contains a sink state (which must be unique),*
2. *$\mathscr{A}$ is synchronizing (and the synchronizing state is the unique sink state),*
3. *Alice has a winning strategy when playing on $\mathscr{A}$.*



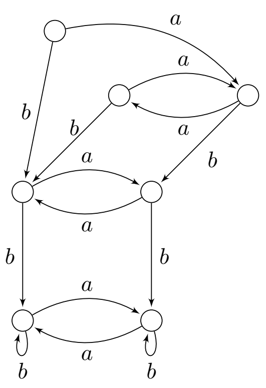


Figure 2 Left: A commutative synchronizing automaton with synchronizing word aab. Right: A commutative automaton that is not synchronizing.

Proof. Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a commutative automaton.

(1) implies (2): Assume $\mathscr{A}$ is connected and let s_f be the sink state. By Lemma 5.1, it is unique and reachable from every other state. Write $Q = \{q_0, q_1, \ldots, q_n\}$. Then, for each number $i \in \{0, 1, \ldots, n\}$ there exists $u_i \in \Sigma^*$ such that $\delta(q_i, u_i) = s_f$. Hence, the word $u = u_0 u_1 \cdots u_n$ synchronizes $\mathscr{A}$. Note that this simple arguments gives a quadratic upper bound for a synchronizing word. However, for commutative DFA the better bound $n - 1$ is known [28]. That the synchronizing state is a unique sink state was shown in [12, Lemma 20].

(2) implies (3): Let $u = u_1 \cdots u_n \in \Sigma^*$ be a synchronizing word with $u_i \in \Sigma$ for each $i \in \{1, \ldots, n\}$. Then, if Alice plays u_i in her i-th move, the will win after at most n moves. For if Bob chooses the letter $b_1, \ldots, b_{n-1}$ after the i-th move of Alice, we have, by commutativity,

$$\delta(Q, u_1 b_1 u_2 b_2 \cdots u_{n-1} b_{n-1} u_n) = \delta(Q, u_1 u_2 \cdots u_n b_1 b_2 \cdots b_{n-1}) = \delta(\delta(Q, u), b_1 b_2 \cdots b_{n-1})$$

and the latter set is a singleton set, as $\delta(Q, u)$ is a singleton set.

(3) implies (1): Similar to Theorem 2.3. ◀

In Theorem 5.2 it was shown that if $\mathscr{A}$ is synchronizing, then Alice has a winning strategy by using a synchronizing word directly as a winning strategy. Combined with the fact that a shortest synchronizing word in a commutative n-state automaton has length $n - 1$, first proven in [28] and reproven by a combinatorial analysis in [12], we find the next bound for the shortest number of moves that Alice has to perform in a winning strategy.

▶ **Proposition 5.3.** *Let $\mathscr{A}$ be a commutative automaton with n states on which Alice can win. Then she can win performing at most $n - 1$ moves and this bound is best possible.*

5.2 Complexity-Theoretical Results

With Theorem 5.2, we can show that testing synchronizability of commutative automata is L-complete. Furthermore, contrary to Theorem 4.1, the "gamified" variant has the same complexity on commutative automata, i.e., is L-complete.

▶ **Theorem 5.4.** *For commutative automata $\mathscr{A}$, the problems* SYNCHRO *and* SYNCHROGAME *are L-complete (even for a fixed unary alphabet).*

In [11, Theorem 3] a reduction from HittingSet to a DFA is given that, in fact, yields a commutative automaton (with an unbounded alphabet) and shows that this problem is NP-hard on this class. Hence, Short Synchro is NP-complete on commutative automata.

Given $\mathscr{A} = (Q, \Sigma, \delta)$, a *permutational letter* $a \in \Sigma$ is a letter such that $q \mapsto \delta(q, a)$ is a permutation, i.e., a bijective mapping. The problems are Short SynchroCommPermChar and Short SynchroGameCommPermChar are defined as Short Synchro and Short SynchroGame but with the input restricted to commutative automata having at least one permutational letter. This letter can be used by Bob to "delay" an eventual win of Alice on a synchronizing commutative automaton, as mapping at least two states to a single state might help Alice. Note that commutativity is crucial here, as then we can move Bob's choices to the beginning, and they do not interfere with the moves from Alice, i.e., do not make the word longer (as is possible in the non-commutative setting by evading states that might get synchronized, i.e., mapped to a single state, by Alice). With this idea, we can show the next theorem.

▶ **Theorem 5.5.** *The following problems are equivalent under logspace-reductions:*
1. Short Synchro *on commutative automata,*
2. Short SynchroCommPermChar*,*
3. Short SynchroGameCommPermChar*.*

and so they are all NP-complete (with the remarks preceding this statement).

Proof. The first two problems are obviously reducible to each other: The second is a restriction of the first, and if we have an instance of the first, we can add an "idle"-letter, i.e., a letter a such that $\delta(q, a) = q$ for each state q and this letter never appears in a shortest synchronizing word and it is a permutational letter.

Now for the equivalence of the latter two problems. Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a commutative automaton with permutational letter $b \in \Sigma$. Then $\mathscr{A}$ has a synchronizing word of length $\leq k$ if and only if Alice can win in at most k moves on $\mathscr{A}$. Suppose Alice can win in at most k moves. In particular, she can win in case Bob always chooses the letter b. More specifically, consider a game with k rounds where Bob always chooses the letter b and Alice wins, i.e., a game

$$a_1 b a_2 b \cdots a_{k-1} b a_k.$$

Observe that we have $\delta(Q, b^k) = Q$, as b is a permutational letter. Hence, we have $\delta(Q, a_1 b a_2 b \cdots a_{k-1} b a_k) = \delta(Q, b^{k-1} a_1 a_2 \cdots a_{k-1} a_k) = \delta(Q, a_1 a_2 \cdots a_{k-1} a_k)$ and so the word $a_1 a_2 \cdots a_{k-1} a_k$ synchronizes $\mathscr{A}$.

Conversely, if $u \in \Sigma^*$ is synchronizing with $u = u_1 \cdots u_k$, $u_i \in \Sigma$, then Alice simply has to choose the letter u_i at her i-th move. This works because when the sequence of moves is $u_1 b_1 u_2 b_2 \cdots b_n u_n$ after Alice has following this strategy, where the b_i indicate Bob's moves, then, by commutativity, $\delta(Q, u_1 b_1 u_2 b_2 \cdots b_n u_n) = \delta(Q, b_1 b_2 \cdots b_n u_1 u_2 \cdots u_n) \subseteq \delta(Q, u_1 u_2 \cdots u_n)$ and the latter set is a singleton set as u is a synchronizing word. So Alice has won. ◀

As Short SynchroCommPermChar is a special case of Short SynchroGame on commutative automata, we get the next corollary to Theorem 5.5.

▶ **Corollary 5.6.** *The problem* Short SynchroGame *is NP-hard on commutative automata.*

For general commutative input automata, we do not know if this problem is also in NP. The best we can show here is that the problem is in $\prod_2^P$, a complexity class from the second level of the polynomial-time hierarchy (see [34]). This poses the natural (open) question if Short SynchroGame is complete for any well-known class between NP and $\prod_2^P$.

▶ **Theorem 5.7.** Short SynchroGame *is in* $\prod_2^P$ *for commutative input automata.*

Proof. Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a commutative automaton and $k \geq 0$. We claim that Alice has a winning strategy within at most k moves if and only if, for every word u with $|u| \leq k-1$, there exists a word v with $|v| = k$ such that

$$|\delta(Q, uv)| = 1.$$

First, suppose Alice has a winning strategy of length at most k and let $u \in \Sigma^*$ with $u = u_1 \cdots u_{k-1}$ and $u_i \in \Sigma$. Then, there exists $v = v_1 \cdots v_k$ such that, when Bob plays u, Alice reacts to u_i with v_{i+1}, and

$$|\delta(Q, v_1 u_1 v_2 \cdots u_{k-1} v_k)| = 1.$$

By commutativity, $\delta(Q, uv) = \delta(Q, v_1 u_1 v_2 \cdots u_{k-1} v_k)$.

Conversely, consider an arbitrary game, played till the i-th round with $i \leq k$, having a sequence of moves $a_1 b_1 a_2 b_2 \cdots b_{i-2} a_{i-1} b_{i-1}$. Then there exists a letter $a_i \in \Sigma$ such that after having played for k rounds, the word

$$a_1 b_1 a_2 b_2 \cdots a_{k-1} b_{k-1} a_k$$

synchronizes $\mathscr{A}$, as for $u = b_1 b_2 \cdots b_{k-1}$ there exists a word $v = a_1 a_2 \cdots a_k$ such that $|\delta(Q, uv)| = 1$ and by commutativity $\delta(Q, a_1 b_1 a_2 b_2 \cdots a_{k-1} b_{k-1} a_k) = \delta(Q, uv)$.

Note that we can assume $|u| = k-1$ by commutativity.

Lastly, given a word and an automaton, it can be checked in logarithmic space [17, Theorem 4] if this word synchronizes a given automaton. Hence, the formula

$$\forall u_1 \cdots u_{k-1} \in \Sigma \; \exists v_1, \ldots, v_k \in \Sigma : |\delta(Q, u_1 \cdots u_{k-1} v_1 \cdots v_k)| = 1$$

can be checked by an alternating non-deterministic polynomial-time Turing machine that starts in a universal state, guesses the sequence $u_1, \ldots, u_{k-1}$ and then switches to a existential state, from which it guesses the remaining words and finally checks if the guessed word synchronizes $\mathscr{A}$. This shows containment in $\prod_2^P$. ◀

6 Cyclic Automata with a k-Simple Idempotent over a Binary Alphabet

Recall the definition of cyclic DFAs with a k-simple idempotent over a binary alphabet $\Sigma = \{a, b\}$: There is one k-simple idempotent letter $a \in \Sigma$, while the letter $b \in \Sigma$ permutes all states cyclically. Notice that there is always a race-track design for cyclic DFAs with a k-simple idempotent as introduced in Section 3.

6.1 Combinatorial and Structural Results

The idempotency immediately leads to our first result. For simplicity, we call synchronizing pairs $\{q, q'\}, \{p, p'\} \in Q_{\mathcal{P}_2}$ with $\delta_{\mathcal{P}_2}(\{q, q'\}, b) = \{p, p'\}$ a *double synchronizing pair.*

▶ **Theorem 6.1.** *Let $\mathscr{A}$ be a cyclic DFA with a k-simple idempotent. Bob has a winning strategy for $\mathscr{A}$ if there is no double synchronizing pair in its pair automaton $\mathcal{P}_2(\mathscr{A})$.*

Proof. As a consequence of Lemma 2.2 it is sufficient to consider only the last two coins left in the synchronization game. On the respective pair automaton, this means that there is one coin left which Bob wants to keep from reaching $\perp$. Whenever it is Bob's turn on a non-synchronizing pair $\{s, t\}$ where choosing b would lead into a synchronizing pair, choosing a will lead to a non-synchronizing pair. Otherwise, $\mathscr{A}$ would not be cyclic with a k-simple idempotent. Furthermore, whenever it is Bob's turn on a synchronizing pair, choosing b will lead to a non-synchronizing state. Bob can therefore always prevent that Alice's turn starts in a synchronizing pair, keeping her from winning. ◀

Since there can only ever be one synchronizing pair in the pair automaton of a 1-simple idempotent automaton, the next corollary follows immediately.

▶ **Corollary 6.2.** *Bob has a winning strategy on every cyclic DFA with a* 1*-simple idempotent.*

To get a better feel for this newly introduced class of DFAs, we continue by looking at cyclic DFAs with a 2-simple idempotent before getting back to the general case. The proof of the following theorem (which is based on some case distinction) leads to a nice result, as it allows us to determine who has a winning strategy on a cyclic DFA with a 2-simple idempotent by simply looking at the DFA in question.

▶ **Theorem 6.3.** *Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a cyclic DFA with a* 2*-simple idempotent with $\Sigma = \{a, b\}$ and a single b-cycle. Alice has a winning strategy on the synchronizing game on $\mathscr{A}$ if and only if $\delta(q_{i+1 \bmod n}, a) = q_i$ and $\delta(q_{i+2 \bmod n}, a) = q_i$ for any $i \in \{0, 1, ..., n-1\}$ and $n \geq 4$.*

▶ **Corollary 6.4.** *Let $\mathscr{A}$ be a cyclic automaton with a k-simple idempotent. Bob has a winning strategy for $\mathscr{A}$ if there is no double synchronizing pair in its pair automaton $\mathcal{P}_2(\mathscr{A})$ which is reachable from every state $q \in Q_{\mathcal{P}_2}$.*

By a similar argument as in the proof of Theorem 6.1, Bob can prevent Alice from leaving a cycle in the pair automaton. A cycle can consist of both a- and b-transitions. To leave a cycle, Alice needs two states $\{s, t\}, \{s', t'\} \in \mathcal{P}_2(\mathscr{A})$ that both have an a-transition leaving the cycle. Bob can also not stop her from leaving if there exists a synchronizing pair preceded or followed by a state with an a-transition leaving the cycle. Otherwise, Alice could synchronize.

▶ **Corollary 6.5.** *Let $\mathscr{A}$ be a cyclic DFA with a k-simple idempotent. Bob has a winning strategy in the synchronization game on $\mathscr{A}$ if there is a cycle in its pair automaton that does not include a double synchronizing pair and which Bob can prevent Alice from leaving.*

Combining the previous results, we conclude with the following theorem.

▶ **Theorem 6.6.** *Alice has a winning strategy in the synchronization game on a cyclic DFA $\mathscr{A}$ with a k-simple idempotent if and only if*

1. *There is at least one double synchronizing pair in its pair automaton $\mathcal{P}_2(\mathscr{A})$ that is reachable from every state $\{q, q'\} \in \mathcal{P}_2(\mathscr{A})$,*
2. *There is no cycle in $\mathcal{P}_2(\mathscr{A})$ that does not contain a double synchronizing pair and which Bob can prevent Alice from leaving.*

Proof. If there is no double synchronizing pair in the pair automaton that is reachable from every $\{q, q'\} \in \mathcal{P}_2(\mathscr{A})$ or there is a cycle without a double synchronizing pair which Bob can prevent Alice from leaving, Bob wins by Theorem 6.1 and Corollaries 6.4 and 6.5.

If there is a double synchronizing pair that is reachable from every $\{q, q'\} \in \mathcal{P}_2(\mathscr{A})$, Bob needs an a-transition either skipping or leading away from the double synchronizing pair to keep Alice from synchronizing. Since there is a path from every state in the pair automaton to the synchronizing pair in question, this always creates a cycle. If Alice cannot leave this cycle and there is no other synchronizing pair within this cycle, then Bob wins. If there is a synchronizing pair that Bob cannot avoid, then Alice wins. If there is a synchronizing pair which Bob can avoid, then the argument repeats itself. ◀

6.2 Complexity-Theoretical Results

A *k-simple idempotent DFA* is a DFA such that every letter either is a k-simple idempotent letter or permutes the states (called a permutational letter in Section 5). Next, we show that for k-simple idempotent automata with an unbounded alphabet, the problem SHORT SYNCHRO is NP-complete. However, we do not know if the problem remains hard for a fixed alphabet, or for cyclic automata with a k-simple idempotent over a binary alphabet. More precisely, we prove the following result.

▶ **Theorem 6.7.** *SHORT SYNCHRO is NP-complete for the class of k-simple idempotent DFAs, for each $k \geq 3$, even if no letter induces a permutation.*

In [22, Proposition 6.1], it was already shown that for 1-simple idempotent automata SHORT SYNCHRO is NP-complete. By adding ”dummy” states ending at the synchronizing state in the construction from [22], it can be adapted to k-simple idempotent automata. However, note that the used reduction from SAT uses an unbounded number of permutations (corresponding to variables) and an unbounded number of 1-simple idempotent letters (corresponding to the clauses). Contrary, our reduction, by using a special variant of SAT, uses no permutations at all (and adding permutations can be achieved easily) and only k-simple idempotent letters. As we were concerned with bounding the number of letters in the previous subsection, our new reduction that is presented in the long version of the paper is of interest in this context.

7 Monotonic Automata

An automaton $\mathscr{A} = (Q, \Sigma, \delta)$ is *monotonic* if there exists a linear ordering of the states $Q = \{q_0, q_1, \ldots, q_{n-1}\}$ such that if $q_i \leq q_j$, i.e., $i \leq j$, then $\delta(q_i, x) \leq \delta(q_j, x)$ for each $x \in \Sigma$.

Monotonic automata were introduced by Ananichev & Volkov [1] (the monotonic automata as introduced earlier by Eppstein [10] are more general, as they are only required to respect a cyclic order and were called *oriented* automata by Ananichev & Volkov). In [31], further problems related to (subset) synchronization problems on monotonic automata were investigated. An open problem from [1] was investigated in [33]. Checking if a given DFA is monotonic is NP-complete [36]. Note that monotonic DFAs also appeared in the context of games previously in [21], were they describe winning conditions on certain infinite games.

The problem SHORT SYNCHRO is solvable in polynomial time on monotonic automata. This follows from the more general result that this problem is solvable in polynomial time on oriented automata as shown in [10] (recall the remark above that what Eppstein calls monotonic was later renamed to oriented). However, we do not know the precise computational complexity for SHORT SYNCHROGAME on monotonic DFAs.

Note that if $u \in \Sigma^*$ and $q \in Q$, then either $q \leq \delta(q, u)$ or $\delta(q, u) \leq q$. The first case implies $q \leq \delta(q, u) \leq \delta(q, u^2) \leq \ldots$ and so, if $q = \delta(q, u^i)$ for some $i > 0$, then $q = \delta(q, u)$ and similarly in the second case. So, no word can permute a subset of states non-trivially and every monotonic automaton is counter-free in the sense of [23].

The DFA in Figure 3 is synchronizable by the word aaa. Yet, Alice has no winning strategy on this automaton. For if $\{q_0, q_1\} \subseteq \delta(Q, u)$ and it is Bob's turn, he can choose a, and if $\{q_2, q_3\} \subseteq \delta(Q, u)$, he can choose b. Doing so, when it is Alice's turn, we have $\{q_1, q_2\} \subseteq \delta(Q, ux)$ and in the next turn Bob is faced with one of the two situations outlined above. As $\{q_1, q_2\} \subseteq Q$, we can conclude inductively that Alice cannot synchronize this DFA.

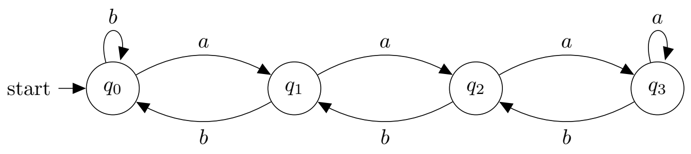


Figure 3 A synchronizable monotonic automaton for which Alice has no winning strategy.

In [25], the authors outlined a modeling of synchronization games by *accessiblity games* as used in software testing [4]. An accessiblity game is a tuple (G, v, A) where G is a finite directed graph, v a vertex and A a subset of vertices. At the start a token is placed on v and then the two players Alice and Bob successively move the token around among neighboring vertices and the goal of Alice is to move the token to a vertex in A, whereas Bob tries to prevent exactly this. Now, in [25] it was shown that every accessibility game can be modelled as a synchronization game started from a single subset containing two states (in [25], the authors generalize the synchronization game to arbitrary starting set instead of the whole state set, but this generalization is also implicit in the pair criterion from Lemma 2.2) and this conversion can be done in polynomial time. The authors also give a conversion of a synchronization game into a accessibility game. However, this conversion involves all subsets of the starting set (which, in our context, is always the full set of states Q) and is not doable in polynomial time. Next, we show that the synchronization game for every monotonic automaton can be converted into an equivalent accessibility game in polynomial time.

▶ **Proposition 7.1.** *For a given monotonic automaton, we can construct in polynomial time an accessiblity game equivalent to the synchronization game (even for the generalization that the starting set is not the full state set).*

Proof. Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a monotonic automaton with linear ordering of the states given by a numbering $Q = \{q_0, q_1, \ldots, q_{n-1}\}$. Then, as for each $q_i \in Q$, $i \in \{0, 1 \ldots, n-1\}$, and $u \in \Sigma^*$ we have $\delta(q_0, u) \leq \delta(q_i, u) \leq \delta(q_{n-1}, u)$, we find that $|\delta(Q, u)| = 1$ if and only if $\delta(q_0, u) = \delta(q_{n-1}, u)$. Construct the graph G with vertex set $\{\{q, q'\} \mid q, q' \in Q\}$ and edge set $\{(\{q, q'\}, \{q'', q'''\}) \mid \exists x \in \Sigma : \delta(\{q, q'\}, x) = \{q'', q'''\}\}$. Then Alice has a winning strategy in the synchronization game on $\mathscr{A}$ if and only if she has a winning strategy in the accessibility game on $(G, \{q_0, q_{n-1}\}, \{\{q\} \mid q \in Q\})$.

Lastly, note that the argument works with any subset $S \subseteq Q$ as a starting set, but instead of q_0, q_{n-1} we use the minimal state and the maximal state in S. ◀

8 Discussions

We have studied conditions that allow us to tell if Alice or Bob have a winning strategy in a *synchronization game*. Concludingly, we discuss some of the consequences that our studies may have on the design of an implementation of a synchronization game. In our eyes, the consequences are mostly concerning the design of different levels for this game. Namely, assume that the game is played between two persons (one of them could be replaced by a computer). Then, it seems to make a difference which DFAs are chosen to be synchronized. Hence, we propose to choose DFAs from the classes that we studied here in order to design lower levels of the game. In particular, commutative DFAs seem to lead to quite simple games. Also, they can be nicely visualized, as explained in Section 3. Then, with growing experience, automata can be proposed that are more complicated, not only in terms of their number of states, but also concerning the classes of automata that they belong to. Another possibility for level design comes with the aspect of time, here best measured in terms of an upper bound on the length of the synchronizing word: if less time is permitted, then Alice has a harder time to sychronize the DFA. Notice that the corresponding basic decision problem (given a DFA $\mathscr{A}$ and an integer k, is there a synchronizing word of length at most k?) is NP-complete even for seemingly easy classes of DFAs; we refer here to [27, 10, 22, 6], in chronological order. This step-bounded game variant is proven to be even PSPACE-complete in [13].

There is one further aspect concerning this level design: one could view the task of Alice in particular as that of extending a "current word" to some synchronizing word by appending letters. As mentioned above, the question if a word can be extended in such a way at all is a particular case of an *extension problem* as studied in [12]. As in general extension problems in this context depend on the chosen ordering on the words, we could increase the level of difficulty in the synchronization game by changing the partial order. For instance, instead of extending a word to the right (corresponding to a prefix ordering), one could also extend it to the left (i.e., using a suffix ordering), or to both sides (corresponding to an infix ordering) or anywhere in the word (corresponding to a subsequence ordering). One might even give different rules for Alice or Bob. This also opens room for further studies concerning winning strategies for these game variations.

References

1 Dimitry S. Ananichev and Mikhail V. Volkov. Synchronizing monotonic automata. *Theoretical Computer Science*, 327(3):225–239, 2004.

2 Dimitry S. Ananichev, Mikhail V. Volkov, and Yu. I. Zaks. Synchronizing automata with a letter of deficiency 2. *Theoretical Computer Science*, 376(1-2):30–41, 2007.

3 Janet H. Barnett. Early writings on graph theory: Hamiltonian circuits and the icosian game. In Brian Hopkins, editor, *Resources for Teaching Discrete Mathematics: Classroom Projects, History Modules, and Articles*, volume 74 of *MAA Notes*, pages 217–224. Mathematical Association of America, 2009.

4 Andreas Blass, Yuri Gurevich, Lev Nachmanson, and Margus Veanes. Play to test. In Wolfgang Grieskamp and Carsten Weise, editors, *Formal Approaches to Software Testing, 5th International Workshop, FATES*, volume 3997 of *LNCS*, pages 32–46. Springer, 2005.

5 Stojan Bogdanović, Balázs Imreh, Miroslav Cirić, and Tatjana Petković. Directable automata and their generalizations: A survey. *Novi Sad J. Math.*, 29(2), 1999.

6 Jens Bruchertseifer and Henning Fernau. Synchronizing series-parallel deterministic automata with loops and related problems. *RAIRO Informatique théorique et Applications/Theoretical Informatics and Applications*, 55(7):1–24, 2021.

7 Janusz A. Brzozowski and Ernst L. Leiss. On equations for regular languages, finite automata, and sequential networks. *Theoretical Computer Science*, 10:19–35, 1980. `doi:10.1016/0304-3975(80)90069-9`.

8 Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981. `doi:10.1145/322234.322243`.

9 Erik D. Demaine. Playing games with algorithms: Algorithmic combinatorial game theory. In Jirí Sgall, Ales Pultr, and Petr Kolman, editors, *Mathematical Foundations of Computer Science 2001, 26th International Symposium, MFCS*, volume 2136 of *LNCS*, pages 18–32. Springer, 2001.

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

11 Henning Fernau, Pinar Heggernes, and Yngve Villanger. A multi-parameter analysis of hard problems on deterministic finite automata. *Journal of Computer and System Sciences*, 81(4):747–765, 2015.

12 Henning Fernau and Stefan Hoffmann. Extensions to minimal synchronizing words. *Journal of Automata, Languages and Combinatorics*, 24(2-4):287–307, 2019.

13 Fedor M. Fominykh, Pavel V. Martyugin, and Mikhail V. Volkov. P(l)aying for synchronization. *International Journal of Foundations of Computer Science*, 24(6):765–780, 2013.

14 Pavel Goralčík and Václav Koubek. Rank problems for composite transformations. *International Journal of Algebra and Computation*, 5(3):309–316, 1995.

15 Vasily V. Gusev. The vertex cover game: Application to transport networks. *Omega*, 97:102102, 2020.

16 Robert A. Hearn and Erik D. Demaine. *Games, puzzles and computation.* A K Peters, 2009.

17 Stefan Hoffmann. Constrained synchronization and commutativity. *Theoretical Computer Science*, 890:147–170, 2021.

18 Stefan Hoffmann. Constrained synchronization and subset synchronization problems for weakly acyclic automata. In Nelma Moreira and Rogério Reis, editors, *Developments in Language Theory - 25th International Conference, DLT*, volume 12811 of *LNCS*, pages 204–216. Springer, 2021.

19 Neil Immerman. Length of predicate calculus formulas as a new complexity measure. In *20th Annual Symposium on Foundations of Computer Science, FOCS*, pages 337–347. IEEE Computer Society, 1979.

20 Raphael M. Jungers. The synchronizing probability function of an automaton. *SIAM Journal of Discrete Mathematics*, 26:177–192, 2012.

21 Eryk Kopczynski. Half-positional determinacy of infinite games. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP, Proceedings, Part II*, volume 4052 of *LNCS*, pages 336–347. Springer, 2006.

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

23 Robert McNaughton and Seymour A. Papert. *Counter-Free Automata (M.I.T. Research Monograph No. 65).* The MIT Press, 1971.

24 Mirjana Mikalački and Miloš Stojaković. Fast strategies in biased maker-breaker games. *Discrete Mathematics & Theoretical Computer Science*, 20(2), 2018.

25 Juan Andres Montoya and Christian Nolasco. Some remarks on synchronization, games and planar automata. *Electronic Notes in Theoretical Computer Science*, 339:85–97, 2018. The XLII Latin American Computing Conference.

26 Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103:299–347, 1993.

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

28 Igor Rystsov. Reset words for commutative and solvable automata. *Theoretical Computer Science*, 172(1-2):273–279, 1997.

29 Igor Rystsov. Estimation of the length of reset words for automata with simple idempotents. *Cybernetics and Systems Analysis*, 36(3):339–344, 2000.

30 Andrew Ryzhikov. Synchronization problems in automata without non-trivial cycles. *Theoretical Computer Science*, 787:77–88, 2019. Implementation and Application of Automata (CIAA 2017).

31 Andrew Ryzhikov and Anton Shemyakov. Subset synchronization in monotonic automata. *Fundamenta Informaticae*, 162(2-3):205–221, 2018.

32 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 [The volume is the outcome of a research seminar that was held in Schloss Dagstuhl in January 2004]*, volume 3472 of *LNCS*, pages 5–33. Springer, 2004.

33 Tamara Shcherbak. The interval rank of monotonic automata. In Jacques Farré, Igor Litovsky, and Sylvain Schmitz, editors, *Implementation and Application of Automata, 10th International Conference, CIAA*, volume 3845 of *LNCS*, pages 273–281. Springer, 2005.

34 Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

35 Ivan Hal Sudborough. The complexity of path problems in graphs and path systems of bounded bandwidth. In Hartmut Noltemeier, editor, *Graphtheoretic Concepts in Computer Science, Proceedings of the International Workshop WG '80*, volume 100 of *LNCS*, pages 293–305. Springer, 1981.

36 Marek Szykula. Checking whether an automaton is monotonic is NP-complete. In Frank Drewes, editor, *Implementation and Application of Automata - 20th International Conference, CIAA*, volume 9223 of *LNCS*, pages 279–291. Springer, 2015.

37 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.

38 Mikhail V. Volkov. Preface: Special issue on the Černý conjecture. *Journal of Automata, Languages and Combinatorics*, 24(2-4):119–121, 2019.