

Synchronization of Deterministic Visibly Push-Down Automata

Henning Fernau 

Universität Trier, Fachbereich IV, Informatikwissenschaften, Germany
fernau@uni-trier.de

Petra Wolf 

Universität Trier, Fachbereich IV, Informatikwissenschaften, Germany
<https://www.wolfp.net/>
wolfp@uni-trier.de

Abstract

We generalize the concept of synchronizing words for finite automata, which map all states of the automata to the same state, to deterministic visibly push-down automata. Here, a synchronizing word w does not only map all states to the same state but also fulfills some conditions on the stack content of each run after reading w . We consider three types of these stack constraints: after reading w , the stack (1) is empty in each run, (2) contains the same sequence of stack symbols in each run, or (3) contains an arbitrary sequence which is independent of the other runs. We show that in contrast to general deterministic push-down automata, it is decidable for deterministic visibly push-down automata whether there exists a synchronizing word with each of these stack constraints, i.e., the problems are in EXPTIME. Under the constraint (1) the problem is even in P. For the sub-classes of deterministic very visibly push-down automata the problem is in P for all three types of constraints. We further study variants of the synchronization problem where the number of turns in the stack height behavior caused by a synchronizing word is restricted, as well as the problem of synchronizing a variant of a sequential transducer, which shows some visibly behavior, by a word that synchronizes the states and produces the same output on all runs.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Grammars and context-free languages; Theory of computation → Automata extensions; Theory of computation → Transducers

Keywords and phrases Synchronizing word, Deterministic visibly push-down automata, Deterministic finite automata, Finite-turn push-down automata, Sequential transducer, Computational complexity

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Funding Petra Wolf: DFG project FE 560/9-1

1 Introduction

The classical *synchronization problem* asks for a deterministic finite automaton (DFA) whether there exists a *synchronizing word* that brings all states of the automaton to a single state. While this problem is solvable in polynomial time [10, 35, 27], many variants, such as synchronizing only a subset of states [27], or synchronizing a partial automaton without taking an undefined transition (called carefully synchronizing) [18], are PSPACE-complete. Restricting the length of a potential synchronizing word by a parameter in the input also yields a harder problem, namely the NP-complete short synchronizing word problem [24, 14]. The field of synchronizing automata has been intensively studied over the last years among others in attempt to verify the famous Černý conjecture claiming that every synchronizable DFA admits a synchronizing word of quadratic length in the number of states [10, 11, 31, 32]. The currently best upper bound on this length is cubic, and only very little progress has been made, basically improving on the multiplicative constant factor in front of the cubic term, see [30, 33]. More information on synchronization of DFA and the Černý conjecture



© Henning Fernau, Petra Wolf;
licensed under Creative Commons License CC-BY

42nd Conference on Very Important Topics (CVIT 2016).

Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

can be found in [35, 6, 1]. In this work, we want to move away from deterministic finite automata to more general deterministic visibly push-down automata.¹

The synchronization problem has been generalized in the literature to other automata models including infinite-state systems with infinite branching such as weighted and timed automata [13, 29] or register automata [5]. For instance, register automata are infinite state systems where a state consists of a control state and register contents.

Another automaton model, where the state set is enhanced with a potential infinite memory structure, namely a stack, is the class of *nested word automata* (NWAs were introduced in [3]), where an input word is enhanced with a matching relation determining at which pair of positions in a word a symbol is pushed to and popped from the stack. The class of languages accepted by NWAs is identical to the class of *visibly push-down languages* (VPL) accepted by *visibly push-down automata* (VPDA) and form a proper sub-class of the deterministic context-free languages. VPDAs have first been studied by Mehlhorn [20] under the name *input-driven pushdown automata* and became quite popular more recently due to the work by Alur and Madhusudan [2], showing that VPLs share several nice properties with regular languages. For more on VPLs we refer to the survey [23]. In [12], the synchronization problem for NWAs was studied. There, the concept of synchronization was generalized to bringing all states to one single state such that for all runs the stack is empty (or in its start configuration) after reading the synchronizing word. In this setting, the synchronization problem is solvable in polynomial time (again indicating similarities of VPLs with regular languages), while the short synchronizing word problem (with length bound given in binary) is PSPACE-complete; the question of synchronizing from or into a subset is EXPTIME-complete. Also, matching exponential upper bounds on the length of a synchronizing word are given.

Our attempt in this work is to study the synchronization problem for real-time deterministic visibly push-down automata (DVPDA) and several sub-classes thereof, like real-time deterministic very visibly push-down automata (DVVPDA), real-time deterministic visibly counter automata (DVCA) and finite turn variants thereof. We want to point out that, despite the equivalence of the accepted language class, the automata models of nested word automata and visibly push-down automata still differ and the results from [12] do not immediately transfer to VPDAs. In general the complexity of the synchronization problem can differ for different automata models accepting the same language class, for instance in contrast to the polynomial time solvable synchronization problem for DFAs, the generalized synchronization problem for finite automata with one ambiguous transition is PSPACE-complete as well as the problem of carefully synchronizing a DFA with one undefined transition [19]. We will not only consider the synchronization model introduced in [12], where reading a synchronizing word results in an empty stack on all runs; but we will also consider a synchronization model where not only the final state on every run must be the same but also the stack content needs to be identical, as well as a model where only the states needs to be synchronized and the stack content might be arbitrary. These three models of synchronization have been introduced in [21], where length bounds on a synchronizing word for general DPDAs have been studied dependent on the stack height. The complexity of these three concepts of synchronization for general DPDAs are considered in [15] where it is shown that synchronizability is undecidable for general DPDAs and deterministic counter automata (DCA). It becomes decidable for deterministic partially blind counter automata and is PSPACE-complete for some types of

¹ The term *synchronization of push-down automata* already occurs in the literature, i.e., in [9, 4], but there the term *synchronization* refers to some relation of the input symbols to the stack behavior [9] or to reading different words in parallel [4]; do not to confuse it with our notion of synchronizing states.

finite turn DPDAs, while it is still undecidable for other types of finite turn DPDAs.

In contrast, we will show in the following that for DVPDAs and considered sub-classes hereof, the synchronization problem for all three stack models, with restricted or unrestricted number of turns, is in EXPTIME and hence decidable. For DVVPDAs and DVCAs, the synchronization problems for all three stack models (with unbounded number of turns) are even in P. Like the synchronization problem for NWA's in the empty stack model considered in [12], we observe that the synchronization problem for DVPDAs in the empty stack model is solvable in polynomial time, whereas synchronization of DVPDAs in the same and arbitrary stack models is at least PSPACE-hard. If the number of turns caused by a synchronizing word on each run is restricted, the synchronization problem becomes PSPACE-hard for all considered automata models for $n > 0$ and is only in P for $n = 0$ in the empty stack model. We will further introduce variants of synchronization problems distinguishing the same and arbitrary stack models by showing complementary complexities in these models. For problems considered in [15], these two stack models have always shared their complexity status.

Missing details can be found in the appendix.

2 Fixing Notations

We refer to the empty word as ϵ . For a finite alphabet Σ we denote with Σ^* the set of all words over Σ and with $\Sigma^+ = \Sigma\Sigma^*$ the set of all non-empty words. For $i \in \mathbb{N}$ we set $[i] = \{1, 2, \dots, i\}$. For $w \in \Sigma^*$ we denote with $|w|$ the length of w , with $w[i]$ for $i \in [|w|]$ the i 'th symbol of w , and with $w[i..j]$ for $i, j \in [|w|]$ the factor $w[i]w[i+1] \dots w[j]$ of w . We call $w[1..i]$ a prefix and $w[i..|w|]$ a suffix of w .

We call $A = (Q, \Sigma, \delta, q_0, F)$ a *deterministic finite automaton* (DFA for short) if Q is a finite set of states, Σ is a finite input alphabet, δ is a transition function $Q \times \Sigma \rightarrow Q$, q_0 is the initial state and $F \subseteq Q$ is the set of final states. The transition function δ is generalized to words by $\delta(q, w) = \delta(\delta(q, w[1]), w[2..|w|])$ for $w \in \Sigma^*$. A word $w \in \Sigma^*$ is accepted by A if $\delta(q_0, w) \in F$ and the language accepted by A is defined by $\mathcal{L}(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \in F\}$. We extend δ to sets of states $Q' \subseteq Q$ or to sets of letters $\Sigma' \subseteq \Sigma$, letting $\delta(Q', \Sigma') = \{\delta(q', \sigma') \mid (q', \sigma') \in Q' \times \Sigma'\}$. Similarly, we may write $\delta(Q', \Sigma') = p$ to define $\delta(q', \sigma') = p$ for each $(q', \sigma') \in Q' \times \Sigma'$. The synchronization problem for DFAs (called DFA-SYNC) asks for a given DFA A whether there exists a synchronizing word for A . A word w is called a *synchronizing word* for a DFA A , if it brings all states of the automaton to one single state, i.e., $|\delta(Q, w)| = 1$.

We call $M = (Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$ a *deterministic push-down automaton* (DPDA for short) if Q is a finite set of states; the finite sets Σ and Γ are the input and stack alphabet, respectively; δ is a transition function $Q \times \Sigma \times \Gamma \rightarrow Q \times \Gamma^*$; q_0 is the initial state; $\perp \in \Gamma$ is the stack bottom symbol which is only allowed as the first (lowest) symbol in the stack, i.e., if $\delta(q, a, \gamma) = (q', \gamma')$ and γ' contains \perp , then \perp only occurs in γ' as its prefix and moreover, $\gamma = \perp$; and F is the set of final states. We will only consider real-time push-down automata and forbid ϵ -transitions, as can be seen in the definition. Notice that the bottom symbol can be removed, but then the computation gets stuck.

Following [12], a *configuration* of M is a tuple $(q, v) \in Q \times \Gamma^*$. For a letter $\sigma \in \Sigma$ and a stack content v with $|v| = n$, we write $(q, v) \xrightarrow{\sigma} (q', v[1..(n-1)]\gamma)$ if $\delta(q, \sigma, v[n]) = (q', \gamma)$. This means that the top of the stack v is the right end of v . We also denote with $\xrightarrow{\sigma}$ the reflexive transitive closure of the union of $\xrightarrow{\sigma}$ over all letters in Σ . The input words on top of $\xrightarrow{\sigma}$ are concatenated accordingly, so that $\xrightarrow{\sigma} = \bigcup_{w \in \Sigma^*} \xrightarrow{w}$. The language $\mathcal{L}(M)$ accepted by a DPDA M is $\mathcal{L}(M) = \{w \in \Sigma^* \mid (q_0, \perp) \xrightarrow{w} (q_f, \gamma), q_f \in F\}$. We call the sequence of configurations $(q, \perp) \xrightarrow{w} (q', \gamma)$ the *run* induced by w , starting in q , and ending in q' . We

might also call q' the *final state* of the run.

We will discuss three different concepts of synchronizing DPDAs. For all concepts we demand that a synchronizing word $w \in \Sigma^*$ maps all states, starting with an empty stack, to the same synchronizing state, i.e., for all $q, q' \in Q$: $(q, \perp) \xrightarrow{w} (\bar{q}, v)$, $(q', \perp) \xrightarrow{w} (\bar{q}, v')$. In other words, for a synchronizing word all runs started on some states in Q end up in the same final state. In addition to synchronizing the states of a DPDA we will consider the following two conditions for the stack content: (1) $v = v' = \perp$, (2) $v = v'$. We will call (1) the *empty stack model* and (2) the *same stack model*. In the third case, we do not put any restrictions on the stack content and call this the *arbitrary stack model*.

As we are only interested in synchronizing a DPDA we can neglect the start and final states.

Starting from DPDAs we define the following sub-classes thereof:

- A *deterministic visibly push-down automaton* (DVPDA) is a DPDA where the input alphabet Σ can be partitioned into $\Sigma = \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}$ such that the change in the stack height is determined by the partition of the alphabet. To be more precise, the transition function δ is modified such that it can be partitioned accordingly into $\delta = \delta_{\text{call}} \cup \delta_{\text{int}} \cup \delta_{\text{ret}}$ such that $\delta_{\text{call}}: Q \times \Sigma \rightarrow Q \times (\Gamma \setminus \{\perp\})$ puts a symbol on the stack, $\delta_{\text{int}}: Q \times \Sigma \rightarrow Q$ leaves the stack unchanged, and $\delta_{\text{ret}}: Q \times \Sigma \times \Gamma \rightarrow Q$ reads and pops a symbol from the stack [2]. If \perp is the symbol on top of the stack, then \perp is only read and not popped. We call letters in Σ_{call} *call* or *push* letters; letter in Σ_{int} *internal* letters; and letters in Σ_{ret} *return* or *pop* letters. The language class accepted by DVPDA is equivalent to the class of languages accepted by deterministic nested word automata (see [12]).
- A *deterministic very visibly push-down automaton* (DVVPA) is a DVPDA where not only the stack height but also the stack content is completely determined by the input alphabet, i.e., for a letter $\sigma \in \Sigma$ and all states $p, q \in Q$ for $\delta_{\text{call}}(p, \sigma) = (p', \gamma_p)$ and $\delta_{\text{call}}(q, \sigma) = (q', \gamma_q)$ it holds that $\gamma_p = \gamma_q$.
- A *deterministic visibly (one) counter automaton* (DVCA) is a DVPDA where $|\Gamma \setminus \{\perp\}| = 1$; note that every DVCA is also a DVVPA.

We are now ready to define a family of synchronization problems, the complexity of which will be our subject of study in the following chapters.

► **Definition 1** SYNC-DVPDA-EMPTY.

Given: DPDA $M = (Q, \Sigma, \Gamma, \delta, \perp)$.

Question: Does there exist a word $w \in \Sigma^*$ that synchronizes M in the empty stack model?

For the same stack model, we refer to the synchronization problem above as SYNC-DVPDA-SAME and as SYNC-DVPDA-ARB in the arbitrary stack model. Variants of these problems are defined by replacing the DVPDA in the definition above by a DVVPA, and DVCA. If results holds for several stack models or automata models, then we summarize the problems by using set notations in the corresponding statements. For the problems SYNC-DVPDA-SAME and SYNC-DVPDA-ARB we introduce two further refined variants of these problems, denoted by the extension -RETURN and -NORETURN, where for all input DVPDA in the former variant $\Sigma_{\text{ret}} \neq \emptyset$ holds, whereas in the latter variant $\Sigma_{\text{ret}} = \emptyset$ holds. In the following these variants reveal insights in the differences between synchronization in the same stack and arbitrary stack models, as well as connections to a concept of trace-synchronizing a sequential transducer showing some visibly behavior.

We will further consider synchronization of these automata classes in a finite-turn setting. Finite-turn push-down automata are introduced in [17]. We adopt the definition in [34]. For a DVPDA M an *upstroke* of M is a sequence of configurations induced by an input word w such that no transition decreases the stack-height. Accordingly, a *downstroke* of M

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

■ **Table 1** Complexity status of the synchronization problem for different classes of deterministic real-time visibly push-down automata in different stack synchronization modes. For the n -turn synchronization variants, n takes all values not explicitly listed. All our problems are in EXPTIME.

is a sequence of configurations in which no transition increases the stack-height. A stroke is either an upstroke or downstroke. A DVPDA M is an n -turn DVPDA if for all $w \in \mathcal{L}(M)$ the sequence of configurations induced by w can be split into at most $n + 1$ strokes. Especially, for 1-turn DVPDAs each sequence of configurations induced by an accepting word consists of one upstroke followed by a most one downstroke. Two subtleties arise when translating this concept to synchronization: (a) there is no initial state so that there is no way to associate a stroke counter to a state, and (b) there is no language of accepted words that restricts the set of words on which the number of strokes should be limited. We therefore generalize the concept of finite-turn DVPDAs to finite-turn synchronization for DVPDAs as follows.

► **Definition 2.** n -TURN-SYNC-DVPDA-EMPTY

Given: DVPDA $M = (Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$.

Question: Is there a synchronizing word $w \in \Sigma^*$ in the empty stack model, such that for all states $q \in Q$, the sequence of configurations $(q, \perp) \xrightarrow{w} (\bar{q}, \perp)$ consists of at most $n + 1$ strokes?

We call such a synchronizing word w an n -turn synchronizing word for M . We define n -TURN-SYNC-DVPDA-SAME and n -TURN-SYNC-DVPDA-ARB accordingly for the same stack and arbitrary stack model. Further we extend the problem definition to other classes of automata such as real-time DVVPDAs, and DVCAs. Table 1 summarizes our results, obtained in the next sections, on the complexity status of these problems together with the above introduced synchronization problems.

Finally, we introduce two PSPACE-complete problems for DFAs to reduce from later.

► **Definition 3 DFA-Sync-Into-Subset (PSPACE-complete [25]).**

Given: DFA $A = (Q, \Sigma, \delta)$, subset $S \subseteq Q$.

Question: Is there a word $w \in \Sigma^*$ such that $\delta(Q, w) \subseteq S$?

► **Definition 4 DFA-Sync-From-Subset (PSPACE-complete [27]).**

Given: DFA $A = (Q, \Sigma, \delta)$ with $S \subseteq Q$.

Question: Is there a word $w \in \Sigma^*$ that synchronizes S , i.e., for which $|\delta(S, w)| = 1$ is true?

3 DVPDAs – Distinguishing the Stack Models

We start with some positive result showing that we come down from the undecidability of the synchronization problem for general DPDA in the empty set model to a polynomial time solvable version by considering visibly DPDA.

► **Theorem 5.** *The problems SYNC-DVPDA-EMPTY, SYNC-DVCA-EMPTY, and SYNC-DVVPDA-EMPTY are decidable in polynomial time.*

Proof. We prove the claim for SYNC-DVPDA-EMPTY as the other automata classes are sub-classes of DVPDAs. Let $M = (Q, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, \Gamma, \delta, \perp)$ be a DVPDA. First, observe that if Σ_{ret} is empty, then any synchronizing word w for M in the empty stack model cannot contain any letter from Σ_{call} . Hence, M is basically a DFA and for DFAs the synchronization problem is in P [10, 35, 27]. From now on, assume $\Sigma_{\text{ret}} \neq \emptyset$. We show that a pair argument similar to the one for DFAs can be applied, namely that M is synchronizable in the empty stack model if and only if every pair of states $p, q \in Q$ can be synchronized in the empty stack model. The only if direction is clear as every synchronizing word for Q also synchronizes each pair of states. For the other direction observe that since M is a DVPDA, the stack height of each path starting in any state of M is predefined by the sequence of input symbols. Hence, if we focus on the two runs starting in p, q and ensure that their stacks are empty after reading a word w , then also the stacks of all other runs starting in other states in parallel are empty after reading w . Therefore, we can successively concatenate words that synchronize some pair of active states in the empty stack model and end up with a word that synchronizes all states of M in the empty stack model. Further formal algorithmic details can be found in the appendix. ◀

Does this mean everything is easy and we are done? Interestingly, the picture is not that simple, as considering the same and arbitrary stack models shows.

► **Theorem 6.** *The problem SYNC-DVPDA-SAME is PSPACE-hard.*

Proof. We reduce from DFA-SYNC-INTO-SUBSET. Let $A = (Q, \Sigma, \delta)$ be a DFA and $S \subseteq Q$. We construct from A a DVPDA $M = (Q \cup \{q_S\}, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, \{\odot, \ominus, \perp\}, \delta', \perp)$ with $q_S \notin Q$, $\Sigma_{\text{call}} = \{a\}$, $\Sigma_{\text{int}} = \Sigma$, $\Sigma_{\text{ret}} = \emptyset$ and $\Sigma_{\text{call}} \cap \Sigma_{\text{int}} = \emptyset$. The transition function δ' agrees with δ on all letters in Σ_{int} . For q_S we set $\delta'(q_S, a) = (q_S, \odot)$ and $\delta'(q_S, \sigma) = q_S$ for all $\sigma \in \Sigma_{\text{int}}$. For $q \in S$, we set $\delta'(q, a) = (q_S, \odot)$, and for $q \notin S$, $\delta'(q, a) = (q, \odot)$.

Note that q_S is a sink-state and can only be reached from states in S with a transition by the call-letter a . For states not in S , the input letter a pushes an \odot on the stack which cannot be pushed to the stack by any letter on a path starting in q_S . Hence, in order to synchronize M in the same stack model, a letter a might only and must be read in a configuration where only states in $S \cup \{q_S\}$ are active. Every word $w \in \Sigma_{\text{int}}^*$ that brings M in such a configuration also synchronizes Q in A into the set S . ◀

From the proof of Theorem 6, we can conclude the next results by observing that a DVPDA without any return letter cannot make any turn.

► **Corollary 7.** *SYNC-DVPDA-SAME-NORETURN and 0-TURN-SYNC-DVPDA-SAME are PSPACE-hard.*

By a rather straight reduction to SYNC-DVPDA-EMPTY, we can show:

► **Theorem 8.** *SYNC-DVPDA-SAME-RETURN is in P.*

The arbitrary stack model requires the most interesting construction in the following proof.

► **Theorem 9.** SYNC-DVPDA-ARB is PSPACE-hard.

Proof. We give a reduction from the PSPACE-complete problem DFA-SYNC-INTO-SUBSET. Let $A = (Q, \Sigma, \delta)$ be a DFA with $S \subseteq Q$. If A is synchronizing, then (A, S) is a positive instance of DFA-SYNC-INTO-SUBSET. As synchronizability can be tested in polynomial time, the reduction can deliver a fixed positive instance of SYNC-DVPDA-ARB. Hence, assume A to be non-synchronizing. We construct a DVPDA $M = (Q \cup Q' \cup Q'' \cup \{q_{\text{fin}}, q_{\text{stall}}\}, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, Q \cup \{F, \perp\}, \delta', \perp)$, where Q' and Q'' are two annotated copies of Q . Further let all unions in the definition of M be disjoint. We set $\Sigma_{\text{int}} = \Sigma$, $\Sigma_{\text{call}} = \{a\}$ and $\Sigma_{\text{ret}} = \{b\}$. An illustration of the construction is depicted in Figure 1.

For states in Q and internal letters, δ' agrees with δ . For all other states, internal letters act as the identity. For all states $q'' \in Q''$, $q' \in Q'$, $q \in Q$, we set $\delta'(q'', a) = (q', q)$ and $\delta'(q', a) = \delta'(q, a) = (q, q)$.

For the remaining states, we set $\delta'(q_{\text{fin}}, a) = (q_{\text{fin}}, F)$ and $\delta'(q_{\text{stall}}, a) = (q_{\text{stall}}, F)$. Further for the return letter b , we set $\delta'(q_{\text{fin}}, b, F) = q_{\text{stall}}$ and $\delta'(q_{\text{stall}}, b, F) = q_{\text{fin}}$. For all states $p \in Q \cup Q' \cup Q'' \cup \{q_{\text{fin}}, q_{\text{stall}}\}$ and stack symbols $q \in Q$, we set $\delta'(p, b, q) = q''$. For the stack bottom symbol \perp , we set $\delta'(q_{\text{fin}}, b, \perp) = \delta'(q_{\text{stall}}, b, \perp) = r''$, where r'' is an arbitrary but fixed state in Q'' . For all other states $p \in Q \cup Q' \cup Q''$, let $\delta'(p, b, \perp) = p$.

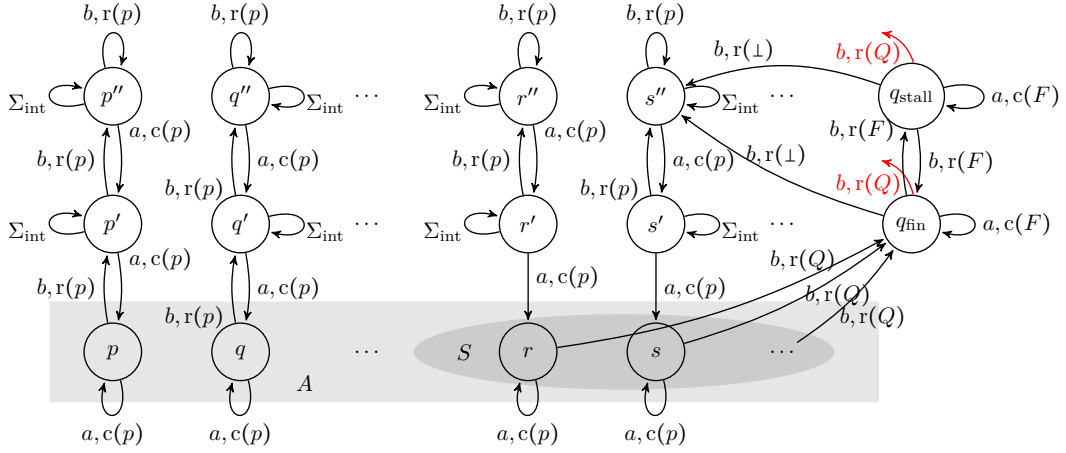
First, assume there exists some word $w \in \Sigma^*$ that brings all states from Q into S in the DFA A . Then, $baawb$ clearly is a synchronizing word for M in the arbitrary stack model.

Now, assume there exists a word $w \in (\Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}})^*$ that synchronizes M in the arbitrary stack model. We make the following observations:

- (1) The runs starting in q_{fin} and q_{stall} can only be synchronized if they are merged with an empty stack. This is only possible before the first call symbol is read.
- (2) States in Q'' and Q' can only be synchronized if they first transition to states in Q .
- (3) For each run starting in a state in Q'' , if the run is currently in a state in Q then there are necessarily at least two symbols from Q on the stack corresponding to this run.
- (4) Further, for these runs it holds that if they are currently in a state in $Q \setminus S$, then after applying the return symbol b they are in a state in Q' with at least one symbol from Q on the stack and every internal letter acts as the identity now. Note that runs which have been in the same state before are now in different states. In order to continue synchronizing them, the runs must be brought back on the layer of states in Q . In order to do so at least one call letter a needs to be read.

(5) M can only be synchronized in the same stack model if it reaches a configuration where only states in S are active and the symbol on top of each stack is a symbol from Q . Here, reading an additional b synchronizes M in the arbitrary stack model.

Observations (1) and (2) give us that there must be at least a configuration where all active states are in Q . So, assume M is in a configuration where both states in $Q \setminus S$ and states in S are active at the same time with a symbol from Q on top of each stack (the stacks cannot be empty or contain an F in this configuration). Then, reading a b brings the states in S into q_{fin} and states in $Q \setminus S$ into states in Q' , whereby merged runs might split again. Reading a sequence baa would bring M into a configuration where *all* states in Q are active with at least two symbols from Q on the stack. We cannot avoid this configuration as an intermediate configuration to a synchronizing configuration from this point on. For instance, if we read an a in order to bring the states in Q' back into their counter-parts in Q all runs currently in q_{fin} would add an F on top of their states. In order to merge the remaining active states, the remaining states in Q need to be brought to q_{fin} by reading a b , but as the



■ **Figure 1** Illustration of the DVPDA M in the proof of Theorem 9. In a transition $c(q)$ means that the stack symbol q is pushed on top of the stack with this transition. Similarly, $r(q)$ means that the stack symbol q is popped from the stack with this transition. The transitions drawn in red lead to the annotated state q'' if the letter q is popped from the stack. For reasons of clarity and comprehensibility, the following transitions are *not* drawn: For $q \in Q, q' \in Q', q'' \in Q''$ $\delta'(\{q, q', q''\}, b, p)$ for $q \neq p$; $\delta'(\{q_{fin}, q_{stall}\}, b, Q)$; $\delta'(Q \cup Q' \cup Q'', b, \perp)$; $\delta'(Q, \Sigma_{int})$.

runs which are currently in q_{fin} have an F on top of their stacks they would elude into q_{stall} . Now the states q_{fin} and q_{stall} are active at the same time and the only configuration reachable from here where both of these states are left (note that they cannot be merged pairwise without transitioning to other states first) is the one where all states in Q'' are active; in order to synchronize any of them, they need to transition into their counter-parts in Q first.

Observation (5) finishes the proof that from a synchronizing word for M in the arbitrary stack model a word that synchronizes Q into S in the DFA A can be extracted. Namely, the sub-word consisting only of internal letters which brings the last configuration in which all states in Q are active into the configuration from which an additional b synchronizes M . ◀

Observe, that in the above construction $\Sigma_{ret} \neq \emptyset$ for all input DFAs. The next corollary follows from Theorem 9 and should be observed together with the next theorem in contrast to Theorem 8 and Corollary 7.

► **Corollary 10.** SYNC-DVPDA-ARB-RETURN is PSPACE-hard.

► **Theorem 11.** SYNC-DVPDA-ARB-NORETURN \equiv DFA-SYNC.

Proof. Let M be a DVPDA with empty set of return symbols. As there is no return-symbol, the transitions of M cannot depend on the stack content. Hence, we can redistribute the symbols in Σ_{call} into Σ_{int} and obtain a DFA. The converse is trivial. ◀

If we move from deterministic visibly push-down automata to even more restricted classes, like deterministic very visibly push-down automata or deterministic visibly counter automata, the three stack models do no longer yield synchronization problems with different complexities. Instead, all three models are equivalent, as stated next. Hence, their synchronization problems can be solved by the pair-argument presented in Theorem 5 in polynomial time.

► **Theorem 12.** SYNC-DVCA-EMPTY \equiv SYNC-DVCA-SAME \equiv SYNC-DVCA-ARB.
 SYNC-DVVPDA-EMPTY \equiv SYNC-DVVPDA-SAME \equiv SYNC-DVVPDA-ARB.

4 Restricting the Number of Turns Makes Synchronization Harder

We are now restricting the number of turns a synchronizing word may cause on any run. Despite the fact that we are hereby restricting the considered model even further, the synchronization problem becomes even harder, as the following results show, in contrast to the previous section.

► **Theorem 13.** *For every fixed $n \in \mathbb{N}$ with $n > 0$, the problems n -TURN-SYNC-DVCA-SAME and n -TURN-SYNC-DVCA-ARB are PSPACE-hard.*

Proof. We give a reduction from the PSPACE-complete problem DFA-SYNC-INTO-SUBSET. Let $A = (Q, \Sigma, \delta)$ be a DFA with $S \subseteq Q$. We construct from A a DVCA $M = (Q \cup \{q_{\text{sync}}\} \cup \{q_{\text{stall}_i} \mid 0 \leq i \leq n\}, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, \{1, \perp\}, \delta', \perp)$, where all unions are disjoint. We set $\Sigma_{\text{int}} = \Sigma$, $\Sigma_{\text{call}} = \{a\}$ and $\Sigma_{\text{ret}} = \{b\}$. For all internal letters, δ' agrees with δ on all states in Q . For the letter a , we set for all $q \in S$, $\delta'(q, a) = (q_{\text{stall}_0}, 1)$ and for all $q \in Q \setminus S$, we set $\delta'(q, a) = (q, 1)$. For b we loop in every state in Q . For q_{sync} , we loop with every letter in q_{sync} (incrementing the counter with a and decrementing it with b).

Let r be an arbitrary but fixed state in Q . For the states q_{stall_i} we set for $i < n$, $\delta'(q_{\text{stall}_i}, a) = (q_{\text{stall}_i}, 1)$. Further, for even index $i < n$, we set $\delta'(q_{\text{stall}_i}, b, 1) = q_{\text{stall}_{i+1}}$ and $\delta'(q_{\text{stall}_i}, b, \perp) = r$. For odd index $i < n$, we set $\delta'(q_{\text{stall}_i}, b, 1) = r$, and $\delta'(q_{\text{stall}_i}, b, \perp) = q_{\text{stall}_{i+1}}$. For even n , let $\delta'(q_{\text{stall}_n}, a) = (q_{\text{sync}}, 1)$, $\delta'(q_{\text{stall}_n}, b, 1) = r$, and $\delta'(q_{\text{stall}_n}, b, \perp) = r$. For odd n , let $\delta'(q_{\text{stall}_n}, a) = (q_{\text{stall}_n}, 1)$, $\delta'(q_{\text{stall}_n}, b, 1) = r$, and $\delta'(q_{\text{stall}_n}, b, \perp) = q_{\text{sync}}$. All other transitions (on internal letters) act as the identity.

Observe that the state q_{sync} must be the synchronizing state of M , since it is a sink state. In order to reach q_{sync} from any state in Q , the automaton must pass through all the states q_{stall_i} for all $0 \leq i \leq n$ by construction. Since we can only transition from a state q_{stall_i} to $q_{\text{stall}_{i+1}}$ with an empty or non-empty stack in alternation, passing the q_{stall_i} gadget forces M to make n turns. For even n , the last upstroke is enforced by passing from q_{stall_n} to q_{sync} by explicitly increasing the stack. As M is only allowed to make n turns while reading the n -turn synchronizing word this implies that any of the states q_{stall_i} might be visited at most once, as branching back into Q by taking a transition that maps to r would force M to go through all states q_{stall_i} again, which exceeds the allowed number of states. Note that only counter values of at most one are allowed in any run which is currently in a state in q_{stall_i} as otherwise the run will necessarily branch back into Q later on.² Especially, this is the case for q_{stall_0} which ensures that each n -turn synchronizing word has first synchronized Q into S before the first letter a is read, as otherwise q_{stall_0} is reached with a counter value greater than 1, or M has already made a turn in Q and hence cannot reach q_{sync} anymore.

In the construction above, for odd n each run enters the synchronizing state with an empty stack (*). For even n each run enters the synchronizing state with a counter value of 1. The visibly condition, or more precisely very visibly condition as we are considering DVCA's, tells us that at each time while reading a synchronizing word, the stack content of every run is identical. In particular, this is the case at the point when the last state enters the synchronizing state and hence, any n -turn synchronizing word for M is a synchronizing word in both the arbitrary and the same stack models. ◀

By observing that in the empty stack model allowing n even turns is as good as allowing $(n - 1)$ turns, essentially (*) from the previous proof yields the next result.

² In some states, such as q_{stall_n} for even n , it is simply impossible to have a higher counter value.

► **Corollary 14.** *For every fixed $n \in \mathbb{N}$ with $n > 0$, the problem n -TURN-SYNC-DVCA-EMPTY is PSPACE-hard.*

► **Corollary 15.** *For every fixed $n \in \mathbb{N}$ with $n > 0$, the problems n -TURN-SYNC-DVPDA and n -TURN-SYNC-DVVPDA in the empty, same, and arbitrary stack models are PSPACE-hard.*

► **Theorem 16.** 0 -TURN-SYNC-DVPDA-EMPTY \equiv DFA-SYNC.

Proof. The visibly condition and the fact that we can only synchronize with an empty stack means that we cannot read any letter from Σ_{call} , hence we cannot use the stack at all. Delete (a) all transitions with a symbol from Σ_{call} and (b) all transitions with a symbol from Σ_{ret} and a non-empty stack. Then, assigning the elements in Σ_{ret} to Σ_{int} gives us a DFA. ◀

The next result is obtained by a reduction from DFA-SYNC-FROM-SUBSET.

► **Theorem 17.** *The problems 0 -TURN-SYNC-DVCA- $\{\text{SAME}, \text{ARB}\}$ are PSPACE-hard.*

► **Corollary 18.** *The problems 0 -TURN-SYNC-DVVPDA- $\{\text{SAME}, \text{ARB}\}$, and 0 -TURN-SYNC-DVPDA- $\{\text{SAME}, \text{ARB}\}$ are PSPACE-hard.*

5 (Non-)Tight Upper Bounds

In this section we will prove that at least all considered problems are decidable (in contrast to non-visibly DPDAs and DCAs, see [15]) by giving exponential time upper bounds. We will also give some tight PSPACE upper bounds for some PSPACE-hard problems discussed in previous section, but for other problems previously discussed a gap between PSPACE-hardness and membership in EXPTIME remains.

► **Theorem 19.** *All problems listed in Table 1 are in EXPTIME.*

Proof. We show the claim explicitly for SYNC-DVPDA-SAME, SYNC-DVPDA-ARB, n -TURN-SYNC-DVPDA-EMPTY, n -TURN-SYNC-DVPDA-SAME, and n -TURN-SYNC-DVPDA-ARB. The other results follow by inclusion of automata classes.

Let $M = (Q, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, \Gamma, \delta, \perp)$ be a DVPDA. We construct from M the $|Q|$ -fold product DVPDA $M^{|Q|}$ with state set $Q^{|Q|}$, consisting of $|Q|$ -tuples of states, and alphabet $\Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}$. Since M is a DVPDA, for every word $w \in (\Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}})^*$, the stack heights on runs starting in different states in Q is equal at every position in w . Hence, we can multiply the stacks to obtain the stack alphabet $\Gamma^{|Q|}$ for $M^{|Q|}$. For the transition function $\delta^{|Q|}$ (split up into $\delta_{\text{call}}^{|Q|} \cup \delta_{\text{int}}^{|Q|} \cup \delta_{\text{ret}}^{|Q|}$) of $M^{|Q|}$ we simulate δ independently on every state in an $|Q|$ -tuple, i.e., for $(q_1, q_2, \dots, q_n) \in Q^{|Q|}$ and letters $\sigma_c \in \Sigma_{\text{call}}, \sigma_i \in \Sigma_{\text{int}}, \sigma_r \in \Sigma_{\text{ret}}$, we set

- $\delta_{\text{call}}^{|Q|}((q_1, q_2, \dots, q_n), \sigma_c) = ((q'_1, q'_2, \dots, q'_n), (\gamma_1, \gamma_2, \dots, \gamma_n))$ if $\delta(q_j, \sigma_c) = (q'_j, \gamma_j)$ for $j \in [n]$;
- $\delta_{\text{int}}^{|Q|}((q_1, q_2, \dots, q_n), \sigma_i) = (\delta(q_1, \sigma_i), \delta(q_2, \sigma_i), \dots, \delta(q_n, \sigma_i))$;
- $\delta_{\text{ret}}^{|Q|}((q_1, q_2, \dots, q_n), \sigma_r, (\gamma_1, \gamma_2, \dots, \gamma_n)) = (\delta(q_1, \sigma_r, \gamma_1), \delta(q_2, \sigma_r, \gamma_2), \dots, \delta(q_n, \sigma_r, \gamma_n))$.

The bottom symbol of the stack is the $|Q|$ -tuple $(\perp, \perp, \dots, \perp)$. Let p_1, p_2, \dots, p_n be an enumeration of the states in Q and set (p_1, p_2, \dots, p_n) as the start state of $M^{|Q|}$.

For SYNC-DVPDA-ARB, set $\{(q, q, \dots, q) \in Q^{|Q|} \mid q \in Q\}$ as the final states for $M^{|Q|}$. Clearly, for SYNC-DVPDA-ARB, $M^{|Q|}$ is a DVPDA and the words accepted by $M^{|Q|}$ are precisely the synchronizing words for M in the arbitrary stack model. As the emptiness problem can be decided for visibly push-down automata in time polynomial in the size of the automaton [2], the claim follows, observing that $M^{|Q|}$ is exponentially larger than M .

For SYNC-DVPDA-SAME, we produce a DVPDA $M_{\text{same}}^{|Q|}$ by enhancing the automaton $M^{|Q|}$ with three additional states $q_{\text{check}}, q_{\text{fin}}$, and q_{fail} and an additional new return letter r and

set q_{fin} as the single accepting state of $M_{\text{same}}^{|Q|}$, while the start state coincides with the one of $M^{|Q|}$. For states $(q_1, q_2, \dots, q_n) \in Q^{|Q|}$ we set $\delta_{\text{ret}}^{|Q|}((q_1, q_2, \dots, q_n), r, (\gamma_1, \gamma_2, \dots, \gamma_n)) = q_{\text{check}}$ if $q_i = q_j$ and $\gamma_i = \gamma_j$, $\gamma_i \neq \perp$ for all $i, j \in [n]$. We set $\delta_{\text{ret}}^{|Q|}((q_1, q_2, \dots, q_n), r, (\perp, \perp, \dots, \perp)) = q_{\text{fin}}$ if $q_i = q_j$ for all $i, j \in [n]$. For all other cases, we map with r to q_{fail} . We let the transitions for q_{fail} be defined such that q_{fail} is a non-accepting trap state for all alphabet symbols. For q_{check} we set $\delta_{\text{ret}}^{|Q|}(q_{\text{check}}, r, (\gamma_1, \gamma_2, \dots, \gamma_n)) = q_{\text{check}}$ if $\gamma_i = \gamma_j$ for $i, j \in [n]$. Further, we set $\delta_{\text{ret}}^{|Q|}(q_{\text{check}}, r, (\perp, \perp, \dots, \perp)) = q_{\text{fin}}$ and map with r to q_{fail} in all other cases. The state q_{check} also maps to q_{fail} with all input symbols other than r . We let the transitions for q_{fin} be defined such that q_{fin} is an accepting trap state for all alphabet symbols.

Clearly, for SYNC-DVPDA-SAME $M_{\text{same}}^{|Q|}$ is a DVPDA and the words accepted by $M_{\text{same}}^{|Q|}$ are precisely the synchronizing words for M in the same stack model, potentially prolonged by a sequence of r 's, as the single accepting state q_{fin} can only be reached from a state in $Q^{|Q|}$ where the states are synchronized and the stack content is identical for each run (which is checked in the state q_{check}). As the size of $M_{\text{same}}^{|Q|}$ is exponential in the size of M , we get the claimed result as in the previous case.

For the n -TURN synchronization problems, we have to modify the previous construction by adding a stroke counter similar as in the proof of Theorem 13 (see Appendix C). ◀

► **Remark 20.** It cannot be expected to show PSPACE-membership of synchronization problems concerning DVPDAs using a $|Q|$ -fold product DVPDA, as the resulting automata is exponentially large in the size of the DVPDA that is to be synchronized, as the emptiness problem for DVPDAs is P-complete [23]. Rather, one would need a separate membership proof. We conjecture that a PSPACE-membership proof similar to the one for the short synchronizing word problem presented in [12] can be obtained if exponential upper bounds for the length of synchronizing words for DVPDAs in the respective models can be obtained.

► **Theorem 21.** *The problems 0-TURN-SYNC-{DVPDA, DVVPDA, DVCA}-SAME are in PSPACE.*

Proof sketch. Let $M = (Q, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, \Gamma, \delta, \perp)$ be a DVPDA. For the same stack model, the 0-turn condition forbids us to put in simultaneous runs different letters on the stack at any time while reading a synchronizing word, as we cannot exchange symbols on the stack with visible PDAs. Note that this is a dynamic runtime-behavior and does not imply that M is necessarily very visibly. Further, the 0-turn and visibility condition enforces that at each step the next transition does not depend on the stack content if the symbol on top of the stack is not \perp . Hence, we can construct from M a $|Q|$ -fold DFA (with a state set exponential in the size of $|Q|$) in a similar way as in the proof of Theorem 19 by neglecting the stack as nothing is ever popped from the stack. Details on the construction can be found in the appendix. As the emptiness problem for DFAs can be solved in NLOGSPACE, the claim follows with Savitch's famous theorem stating that $\text{NPSPACE} = \text{PSPACE}$ [28].³ ◀

► **Corollary 22.** SYNC-DVPDA-SAME-NORETURN is in PSPACE.

► **Theorem 23.** *The problems 0-TURN-SYNC-{DVPDA, DVVPDA, DVCA}-ARB, and 1-TURN-SYNC-DVCA-{EMPTY, SAME, ARB} are in PSPACE.*

Proof. The claim follows from [15, Theorem 16 & 17] by inclusion of automata classes. ◀

³ Here, a smaller powerset-construction would also work but for simplicity, we stuck with the introduced $|Q|$ -fold product construction.

6 Sequential Transducers

In [15], the concept of trace-synchronizing a sequential transducer has been introduced. We want to extend this concept to sequential transducers showing some kind of *visible behavior* regarding their output, inspired by the predetermined stack height behavior of DVPDAs. We call $T = (Q, \Sigma, \Gamma, q_0, \delta, F)$ a *sequential transducer* (ST for short) if Q is a finite set of states, Σ is an input alphabet, Γ is an output alphabet, q_0 is the start state, $\delta: Q \times \Sigma \rightarrow Q \times \Gamma^*$ is a total transition function, and F collects the final states. We generalize δ from input letters to words by concatenating the produced outputs. T is called a *visibly sequential transducer* (VST for short) [or *very visibly sequential transducer* (VVST for short)] if for each $\sigma \in \Sigma$ and for all $q_1, q_2 \in Q$ and $\gamma_1, \gamma_2 \in \Gamma^*$, it holds that $\delta(q_1, \sigma) = (q'_1, \gamma_1)$ and $\delta(q_2, \sigma) = (q'_2, \gamma_2)$ implies that $|\gamma_1| = |\gamma_2|$ [or that $\gamma_1 = \gamma_2$, respectively]. A VVST T is thereby computing the same homomorphism h_T , regardless of which states are chosen as start and final states (*). Hence, if A_T is the underlying DFA (ignoring any outputs), then $h_T(\mathcal{L}(A_T)) \subseteq \Gamma^*$ describes the language of all possible outputs of T . By Nivat's theorem [22], a language family is a full trio iff it is closed under VVST and inverse homomorphisms.

We say that a word w *trace-synchronizes* a sequential transducer T if, for all states $p, q \in Q$, $\delta(p, w) = \delta(q, w)$, i.e., a synchronizing state is reached, producing identical output.

► **Definition 24** TRACE-SYNC-TRANSDUCER.

Given: Sequential transducer $T = (Q, \Sigma, \Gamma, \delta)$.

Question: Does there exist a word $w \in \Sigma^$ that trace-synchronizes T ?*

We define TRACE-SYNC-VST and TRACE-SYNC-VVST by considering a VST, respectively VVST, instead. In contrast to the undecidability of TRACE-SYNC-TRANSDUCER [15], we get the following results for trace-synchronizing VST and VVST from previous results. The next theorem follows by a reduction from and to SYNC-DVPDA-SAME-NORETURN.

► **Theorem 25.** TRACE-SYNC-VST is PSPACE-complete.

Yet, by Observation (*), we inherit from SYNC-DFA the following algorithmic result.

► **Theorem 26.** TRACE-SYNC-VVST is in P.

7 Discussion

Our results concerning DVPDAs and sub-classes thereof, are subsumed in Table 1. While all problems listed in the table are contained in EXPTIME, the table lists several problems for which their known complexity status still contains a gap between PSPACE-hardness lower bounds and EXPTIME upper bounds. Presumably, their precise complexity status is closely related to upper bounds on the length of synchronizing words which we want to consider in the near future. One of the questions which could be solved in this work is if there is a difference between the complexity of synchronization in the same stack model and synchronization in the arbitrary stack model. While for general DPDA, DCA, and sub-classes thereof, see [15], these two models admitted synchronization problems with the same complexity, here we observed that these models can differ significantly. While the focus of this work is on determining the complexity status of synchronizability for different models of automata, an obvious question for future research is the complexity status of closely related, and well understood questions in the realm of DFAs, such as the problem of shortest synchronizing word, subset synchronization, synchronization into a subset, and careful synchronization.

References

- 1 Journal of Automata, Languages and Combinatorics – Essays on the Černý Conjecture. https://www.jalc.de/issues/2019/issue_24_2-4/content.html. Accessed: 10/1/2020.
- 2 Rajeev Alur and P. Madhusudan. Visibly Pushdown Languages. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 202–211. ACM, 2004.
- 3 Rajeev Alur and P. Madhusudan. Adding Nesting Structure to Words. *J. ACM*, 56(3):16:1–16:43, 2009.
- 4 Marcelo Arenas, Pablo Barceló, and Leonid Libkin. Regular Languages of Nested Words: Fixed Points, Automata, and Synchronization. *Theory of Computing Systems*, 49(3):639–670, 2011.
- 5 Parvaneh Babari, Karin Quaas, and Mahsa Shirmohammadi. Synchronizing Data Words for Register Automata. In *41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
- 6 Marie-Pierre Béal and Dominique Perrin. *Synchronised Automata*, page 213–240. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2016.
- 7 Jean Berstel. *Transductions and Context-Free Languages*, volume 38 of *Teubner Studienbücher: Informatik*. Teubner, 1979.
- 8 Olivier Carton. The Growth Ratio of Synchronous Rational Relations is Unique. *Theoretical Computer Science*, 376(1-2):52–59, 2007.
- 9 Didier Caucal. Synchronization of Pushdown Automata. In Oscar H. Ibarra and Zhe Dang, editors, *Developments in Language Theory, 10th International Conference, DLT 2006, Santa Barbara, CA, USA, June 26-29, 2006, Proceedings*, volume 4036 of *Lecture Notes in Computer Science*, pages 120–132. Springer, 2006.
- 10 Ján Černý. Poznámka k homogénnym eksperimentom s konečnými automatami. *Matematicko-fyzikálny Časopis Slovensk*, 14(3):208–215, 1964.
- 11 Ján Černý. A Note on Homogeneous Experiments with Finite Automata. *Journal of Automata, Languages and Combinatorics*, 24(2-4):123–132, 2019.
- 12 Dmitry Chistikov, Pavel Martyugin, and Mahsa Shirmohammadi. Synchronizing Automata over Nested Words. *Journal of Automata, Languages and Combinatorics*, 24(2-4):219–251, 2019.
- 13 Laurent Doyen, Line Juhl, Kim Guldstrand Larsen, Nicolas Markey, and Mahsa Shirmohammadi. Synchronizing Words for Weighted and Timed Automata. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, pages 121–132, 2014.
- 14 David Eppstein. Reset Sequences for Monotonic Automata. *SIAM Journal on Computing*, 19(3):500–510, 1990.
- 15 Henning Fernau, Petra Wolf, and Tomoyuki Yamakami. Synchronizing Deterministic Push-Down Automata Can Be Really Hard. *CoRR*, abs/2005.01381, 2020. URL: <https://arxiv.org/abs/2005.01381>, [arXiv:2005.01381](https://arxiv.org/abs/2005.01381).
- 16 Seymour Ginsburg. *The mathematical Theory of Context-Free Languages*. McGraw-Hill, 1966.
- 17 Seymour Ginsburg and Edwin H Spanier. Finite-Turn Pushdown Automata. *SIAM Journal on Control*, 4(3):429–453, 1966.
- 18 Pavel Martyugin. Computational Complexity of Certain Problems Related to Carefully Synchronizing Words for Partial Automata and Directing Words for Nondeterministic Automata. *Theory of Computing Systems*, 54(2):293–304, 2014.
- 19 Pavel V. Martyugin. Synchronization of Automata with One Undefined or Ambiguous Transition. In Nelma Moreira and Rogério Reis, editors, *Implementation and Application of Automata - 17th International Conference, CIAA 2012, Porto, Portugal, July 17-20, 2012. Proceedings*, volume 7381 of *Lecture Notes in Computer Science*, pages 278–288. Springer, 2012.

- 20 Kurt Mehlhorn. Pebbling Mountain Ranges and its Application of DCFL-Recognition. In J. W. de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordwijkerhout, The Netherlands, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 422–435. Springer, 1980.
- 21 Eitatsu Mikami and Tomoyuki Yamakami. Synchronizing Pushdown Automata and Reset Words. 2020. An article appeared in Japanese as Technical Report of The Institute of Electronics, Information and Communication Engineers, COMP2019-54(2020-03), pp. 57–63.
- 22 Maurice Nivat. Transductions des langages de Chomsky. *Ann. Inst. Fourier, Grenoble*, 18:339–456, 1968.
- 23 Alexander Okhotin and Kai Salomaa. Complexity of Input-Driven Pushdown Automata. *SIGACT News*, 45(2):47–67, 2014.
- 24 I. K. 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).
- 25 I. K. Rystsov. Polynomial Complete Problems in Automata Theory. *Information Processing Letters*, 16(3):147–151, 1983.
- 26 Jacques Sakarovitch. *Éléments de Théorie des Automates*. Vuibert informatique, 2003.
- 27 Sven Sandberg. Homing and Synchronizing Sequences. In Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner, editors, *Model-Based Testing of Reactive Systems, Advanced Lectures*, volume 3472 of *LNCS*, pages 5–33. Springer, 2005.
- 28 Walter J. Savitch. Relationships Between Nondeterministic and Deterministic Tape Complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.
- 29 Mahsa Shirmohammadi. *Qualitative Analysis of Synchronizing Probabilistic Systems. (Analyse qualitative des systèmes probabilistes synchronisants)*. PhD thesis, École normale supérieure de Cachan, France, 2014. URL: <https://tel.archives-ouvertes.fr/tel-01153942>.
- 30 Yaroslav Shitov. An Improvement to a Recent Upper Bound for Synchronizing Words of Finite Automata. *Journal of Automata, Languages and Combinatorics*, 24(2-4):367–373, 2019.
- 31 Peter H. Starke. Eine Bemerkung über homogene Experimente. *Elektronische Informationsverarbeitung und Kybernetik (J. Inf. Process. Cybern.)*, 2(4):257–259, 1966.
- 32 Peter H. Starke. A Remark About Homogeneous Experiments. *Journal of Automata, Languages and Combinatorics*, 24(2-4):133–137, 2019.
- 33 Marek Szykuła. Improving the Upper Bound on the Length of the Shortest Reset Word. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPICs*, pages 56:1–56:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
- 34 Leslie G. Valiant. *Decision Procedures for Families of Deterministic Pushdown Automata*. PhD thesis, University of Warwick, Coventry, UK, 1973. URL: <http://wrap.warwick.ac.uk/34701/>.
- 35 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.

A Proofs for Section 3 (DVPDAs – Distinguishing the Stack Models)

Formal and algorithmic proof details of Theorem 5. In order to determine if a pair of states $p, q \in Q$ can be synchronized in the empty stack model, we build the following product automaton $M \times M[p, q] = (Q \times Q \cup Q, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, \Gamma, \delta^2, (p, q), \perp, Q)$. For all states in $(r, s) \in Q \times Q$ for which $r \neq s$, δ^2 simulates the actions of δ on r in the first component and the actions of δ on s in the second component. For states $(r, r) \in Q \times Q$, this is also the case for all transitions except for zero-tests of the stack, as here we transition to the corresponding state $r \in Q$. For Q , δ^2 , restricted to Q , is the same as δ . Clearly, $M \times M[p, q]$ accepts all words that have $w\sigma_r$ as a prefix, for which w synchronizes the states p and q in M in the empty stack model and σ_r is any return letter in Σ_{ret} that checks the empty stack condition. Further for all pairs of states p, q , $M \times M[p, q]$ is a DVPDA. As the emptiness problem for DVPDAs is in P [2], we can build and test all product automata $M \times M[p, q]$ for non-emptiness in polynomial time. ◀

Proof of Theorem 8. We prove the claim by straight reducing to SYNC-DVPDA-EMPTY with the identity function. If a DVPDA M with $\Sigma_{\text{ret}} \neq \emptyset$ can be synchronized in the same stack model with a synchronizing word w , then w can be extended to ww' where $w' \in \Sigma_{\text{ret}}^*$ empties the stack. As M is deterministic and complete, w' is defined on all states. As after reading w , the stack content on all paths is the same, reading w' extends all paths with the same sequence of states. Conversely, a word w synchronizing a DVPDA M with $\Sigma_{\text{ret}} \neq \emptyset$ in the empty stack model also synchronizes M in the same stack model. ◀

Proof of Theorem 12. First, note that every DVCA is also a DVVPDA. If for a DVVPDA $\Sigma_{\text{ret}} \neq \emptyset$, then we can empty the stack after synchronizing the state set, as the very visibly conditions ensures that the contents of the stacks on all runs coincide. As the automaton is deterministic, all transitions for letters in Σ_{ret} are defined on each state. As the stack content on all runs coincides in every step, the arbitrary stack model is identical to the same stack model and hence equivalent to the empty stack model. If $\Sigma_{\text{ret}} = \emptyset$, then we can reassign Σ_{call} to Σ_{int} in order to reduce from the same-stack and arbitrary stack to the empty stack variant, as transitions cannot depend on the stack content which is again the same on all runs due to the very visibly condition. ◀

B Proofs for Section 4 (Restricting the Number of Turns Makes Synchronization Harder)

Proof of Cor. 14. Since we need to synchronize with an empty stack, for even n , the last upstroke cannot be performed. Hence, for even n , every DVCA M can be synchronized by an n -turn synchronizing word if and only if M can be synchronized by an $(n-1)$ -turn synchronizing word. As for odd n in the construction above, every n -turn synchronizing word synchronized M in the empty stack model, the claim follows from the proof in Theorem 13. ◀

Proof of Theorem 17. We give a reduction from the PSPACE-complete problem DFA-SYNC-FROM-SUBSET. Let $A = (Q, \Sigma, \delta)$ be a DFA with $S \subseteq Q$. We construct from A a DVCA $M = (Q, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, \{1, \perp\}, \delta', \perp)$. We set $\Sigma_{\text{call}} = \Sigma$, $\Sigma_{\text{int}} = \emptyset$, and $\Sigma_{\text{ret}} = \{b\}$. For all $q \in Q$ and $\sigma \in \Sigma_{\text{call}}$, we set $\delta'(q, \sigma) = (\delta(q, \sigma), 1)$. For all states $q \in Q \setminus S$, we set $\delta(q, b, \perp) = s$ for some arbitrary but fixed state $s \in S$. All other transitions act as the identity.

Note that the 0-turn condition only allows us to read the letter b before any letter in Σ_{call} has been read, as afterwards b would decrease the stack after it has been increased.

Therefore, every synchronizing word for M in the same and arbitrary stack models also synchronizes S in Q by either synchronizing the whole set Q without using any b transition, or it brings Q in exactly the set S with the first letter b and continues to synchronize S . ◀

C Proofs for Section 5 ((Non-)Tight Upper Bounds)

Further proof details for the n -TURN cases in Theorem 19. For the problems n -TURN-SYNC-DVPDA in the empty, same, and arbitrary stack models, we enhance in $M^{|Q|}$ each $|Q|$ -tuple with an additional index $I \in \{0, 1, \dots, n+1\}$, i.e., the basic set of states is now $Q^{|Q|} \times \{0, 1, \dots, n+1\}$. We further add for all three models the non-accepting trap state q_{fail} to the set of states. For each $(|Q|+1)$ -tuple, we implement the transition function $\delta_{\text{int}}^{|Q|}$ of $M^{|Q|}$ for internal letters in Σ_{int} as before by keeping the value of the index I in each transition. For call letters in Σ_{call} we realize $\delta_{\text{call}}^{|Q|}$ as before for state-tuples with index $I < n+1$ by simulation δ on the individual states and setting in every image $I = I + 1$ if I is even, and keeping the value of I if I is odd. For tuples with index $I = n+1$, we proceed as before for smaller index if $n+1$ is odd, while for even $n+1$ we map with a call letter to the state q_{fail} . For the return letters in Σ_{ret} , we realize $\delta_{\text{ret}}^{|Q|}$ for pairs of states in $Q^{|Q|} \times \{1, 2, \dots, n+1\}$ and bottom of stack symbol $(\perp, \perp, \dots, \perp)$ as before by simulating δ on the individual states and keeping the value of I . For all other stack symbols, we realize $\delta_{\text{ret}}^{|Q|}$ as before for state-tuples with index $0 < I < n+1$ by simulation δ on the individual states and keeping the value of I if I is even, and setting in every image $I = I + 1$ if I is odd. For tuples with $I = n+1$ we proceed as before if $n+1$ is even. For states with index $I = 0$ or $I = n+1$ for odd $n+1$, we map with each return letter to q_{fail} for stack symbols other than the bottom of stack symbol. In all three models we set $(p_1, p_2, \dots, p_n, 0)$ as the start state, with p_1, p_2, \dots, p_n being an enumeration of the states in Q .

For n -TURN-SYNC-DVPDA-ARB, we set $\{(q, q, \dots, q, I) \mid q \in Q, I \in \{0, 1, \dots, n+1\}\} \subset Q^{|Q|} \times \{0, 1, \dots, n+1\}$ as the set of final states.

For n -TURN-SYNC-DVPDA-EMPTY, we set the additional trap state q_{fin} as the single accepting state and add a new return letter r with which we map to q_{fin} for states $(q, q, \dots, q, I) \in Q^{|Q|} \times \{0, 1, \dots, n+1\}$ with the bottom-of-stack symbol and to q_{fail} for all other stack symbols or states.

For n -TURN-SYNC-DVPDA-SAME, we add the two states q_{check} and q_{fin} to $M^{|Q|}$ and set q_{fin} as the single accepting state. Again, we add a new return letter r . For states (q, q, \dots, q, I) with $I \in \{0, 1, \dots, n+1\}$ and symbol $(\gamma_1, \gamma_2, \dots, \gamma_n)$ on top of the stack, which is not the bottom-of-stack symbol, we map with r to q_{check} if all entries γ_i in the stack symbol tuple are identical. If instead the bottom-of-stack symbol is on top of the stack, we map with r for states (q, q, \dots, q, I) directly to q_{fin} . For all other states and stack symbols, r maps to q_{fail} . For q_{check} , we stay in q_{check} with the letter r if we see a symbol $(\gamma, \gamma, \dots, \gamma)$ on the stack with $\gamma \in \Gamma \setminus \{\perp\}$ and map with r to q_{fin} if we see the bottom-of-stack symbol. For all other stack symbols, r maps q_{check} to q_{fail} . Also, all input letter other than r maps q_{check} to q_{fail} . For q_{fin} we define all transitions such that q_{fin} is a trap state.

In all three cases, the constructed automaton is a DVPDA that accepts precisely the n -turn synchronizing words for M (potentially prolonged by a sequence of r 's) in the respective stack model. As the constructed automaton is of size $\mathcal{O}((|Q||\Gamma|)^{|Q|})$ in all three cases, we can decide whether the constructed automaton accepts at least one word in time exponential in the description of M . ◀

Proof of Theorem 21. Let $M = (Q, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, \Gamma, \delta, \perp)$ be a DVPDA. For the same stack model, the 0-turn condition forbids us to put in simultaneous runs different letters on

the stack at any time while reading a synchronizing word, as we cannot exchange symbols on the stack with visible PDAs. Note that this is a dynamic runtime-behavior and does not imply that M is necessarily very visibly. Further, the 0-turn and visibility condition enforces that at each step the next transition does not depend on the stack content if the symbol on top of the stack is not \perp . We construct from M a partial $|Q|$ -fold product DFA $M^{|Q|}$ with state set $Q^{|Q|} \times \{0, 1\}$, consisting of $|Q|$ -tuples of states with an additional bit of information which will indicate whether the stack is still empty, and alphabet $\Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}$. For the transition function $\delta^{|Q|}$ of $M^{|Q|}$, we simulate for a state $(q_1, q_2, \dots, q_n, b)$ with $q_1, q_2, \dots, q_n \in Q$, $b \in \{0, 1\}$ and letter σ , δ (by restricting the image to the first component in Q for call letters) on the individual states q_i, q_j with $i, j \in [n]$ in the tuple if (1) $\sigma \in \Sigma_{\text{ret}}$ and $b = 0$, (2) $\sigma \in \Sigma_{\text{int}}$, or (3) $\sigma \in \Sigma_{\text{call}}$ and for $\delta(q_i, \sigma) = (q'_i, \gamma_i)$, $\delta(q_j, \sigma) = (q'_j, \gamma_j)$ it holds that $\gamma_i = \gamma_j$. In case (1) and (2), we keep the value of b in the transition and in case (3), we ensure $b = 1$ in the image of the transition. The size of the state graph of $M^{|Q|}$ is bounded by $\mathcal{O}(|Q|^{|Q|})$. Clearly, the DVPDA M can be synchronized by a 0-turn synchronizing word in the same stack model if and only if there is a path in the state graph of $M^{|Q|}$ from the state $(q_1, q_2, \dots, q_n, 0)$ for $Q = \{q_1, q_2, \dots, q_n\}$ to some state in $\{(q_i, q_i, \dots, q_i, b) \mid i \in [n], b \in \{0, 1\}\}$. These $2|Q|$ reachability tests can be performed in $\text{NPSpace} = \text{PSPACE}$ [28]. The claim for the other problems follows by inclusion of automata classes. \blacktriangleleft

Proof of Cor. 22. Let $M = (Q, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, \Gamma, \delta, \perp)$ be a DVPDA with $\Sigma_{\text{ret}} = \emptyset$. As we have no return letter, any synchronizing word for M is also a 0-turn synchronizing word and hence, the claim follows with Theorem 21. \blacktriangleleft

D Proofs for Section 6 (Sequential Transducers)

► **Remark 27.** The definitions in the literature are not very clear for finite automata with outputs. We follow here the name used by Berstel in [7]; Ginsburg [16] called Berstel's sequential transducers *generalized machines*, but used the term *sequential transducer* for the nondeterministic counterpart. For non-deterministic transducers which allow to read multiple letters at once the concept of fixing the ratio between the length of the produced output and the length of the input was already studied in [8] and was even mentioned in [26]. Here, the ratio is fixed for every transition independent of the input letter(s) and a transducer admitting such a fixed ratio α is called α -synchronous. The term 'synchronization' again appears here but refers to finding an α -synchronous transducer to a given rational relation.

Proof of Theorem 25. First, observe that there is a straight reduction from the problem SYNC-DVPDA-SAME-NORETURN to TRACE-SYNC-VST as the input DVPDAs to the problem SYNC-DVPDA-SAME-NORETURN have no return letters and hence, the stack is basically a write only tape. Further, as the remaining alphabet is partitioned into letters in Σ_{call} , which write precisely one symbol on the stack, and into letters in Σ_{int} , writing nothing on the stack, the visibly condition is satisfied when interpreting the DVPDA with $\Sigma_{\text{ret}} = \emptyset$ as a VST.

There is also a straight reduction from TRACE-SYNC-VST to SYNC-DVPDA-SAME-NORETURN as follows. For a VST $T = (Q, \Sigma, \Gamma, \delta)$ we construct a DVPDA $M = (Q, \Sigma_{\text{call}} \cup \Sigma_{\text{int}}, \Gamma', \delta)$ with $\Sigma_{\text{ret}} = \emptyset$ by introducing for each $\sigma \in \Sigma$ a new alphabet $\Sigma_\sigma = \{w \in \Gamma^* \mid \exists q, q' \in Q: \delta(q, \sigma) = (q', w)\}$. Observe that Σ_σ is either $\{\epsilon\}$ or contains only words of the same length. By setting $\Sigma_{\text{int}} = \{\sigma \in \Sigma \mid \Sigma_\sigma = \{\epsilon\}\}$, $\Sigma_{\text{call}} = \{\sigma \in \Sigma \mid \Sigma_\sigma \neq \{\epsilon\}\}$, $\Gamma' = \bigcup_{\sigma \in \Sigma} (\Sigma_\sigma \setminus \{\epsilon\})$, and interpreting the output sequence $w \in \Gamma^*$ produced by δ as the single stack symbol in Γ' . \blacktriangleleft

Proof of Theorem 26. For each VVST $T = (Q, \Sigma, \Gamma, \delta)$ and $w \in \Sigma^*$ the same output is already produced in $\delta(q, w)$ for all $q \in Q$. Hence, we can ignore the output and test T for trace-synchronization by the polynomial time pair-algorithm for DFAs [27]. ◀

E Remarks on Section 7 (Discussion)

► **Remark 28.** Here is one subtlety that comes with shortest synchronizing words: While for finding synchronizing words of length at most k for DFAs, it does not matter if the number k is given in unary or in binary due to the known cubic upper bounds on the lengths of shortest synchronizing words, this will make a difference in other models where such polynomial length bounds are unknown. More precisely, for instance with DVPDAs, it is rather obvious that with a unary length bound k , the problem becomes NP-complete, while the status is unclear for binary length bounds.