Synchronizing Deterministic Push-Down Automata Can Be Really Hard

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@informatik.uni-trier.de

Tomoyuki Yamakami
University of Fukui, Faculty of Engineering, Japan
tomoyukiyamakami@gmail.com

Abstract

The question if a deterministic finite automaton admits a software reset in the form of a so-called synchronizing word can be answered in polynomial time. In this paper, we extend this algorithmic question to deterministic automata beyond finite automata. We prove that the question of synchronizability becomes undecidable even when looking at deterministic one-counter automata. This is also true for another classical mild extension of regularity, namely that of deterministic one-turn push-down automata. However, when we combine both restrictions, we arrive at scenarios with a PSPACE-complete (and hence decidable) synchronizability problem. Likewise, we arrive at a decidable synchronizability problem for (partially) blind deterministic counter automata.

There are several interpretations of what *synchronizability* should mean for deterministic push-down automata. This is depending on the role of the stack: should it be empty on synchronization, should it be always the same or is it arbitrary? For the automata classes studied in this paper, the complexity or decidability status of the synchronizability problem is mostly independent of this technicality, but we also discuss one class of automata where this makes a difference.

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 automaton, Reset sequence, Real-time deterministic push-down automaton, Finite-turn push-down automaton, Computability, Computational complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2020.33

Related Version A full version is available at ArXiv https://arxiv.org/abs/2005.01381.

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

1 Introduction

The classical *synchronization problem* asks, for a given deterministic finite automaton (DFA), if there exists a *synchronizing word*, i.e., an input that brings all states of the automaton to a single state. While this problem is solvable in polynomial time [11, 30, 50, 42], many variants, such as synchronizing only a subset of states [42], or synchronizing only into a specified subset of states [41], or synchronizing a partial automaton without taking an undefined transition [32], are PSPACE-complete. Restricting the length of a potential synchronizing word of some DFA by an integer parameter in the input also yields a harder problem, namely the NP-complete short synchronizing word problem [40, 16]. The field of synchronizing automata has been intensively studied over the last years also in attempt to verify the famous Černý

conjecture, claiming that every synchronizable DFA admits a synchronizing word of quadratic length in the number of states [11, 12, 46, 47]. We are far from solving this combinatorial question, as the currently best upper bound on this length is only cubic [45, 48]. For more on synchronization of DFAs and the Černý conjecture, we refer to the surveys [50, 5, 1].

The idea of bringing an automaton to a well-defined state by reading a word, starting from any state, can be seen as implementing a software reset. This is why a synchronizing word is also sometimes called a *reset word*. But this very idea is obviously not restricted to finite automata. In this work, we want to move away from deterministic finite automata to more general deterministic push-down automata. What should a synchronizing word mean in this context? Mikami and Yamakami first studied in [36] three different models, depending on requirements of the stack contents when a word w drives the automaton into a synchronizing state, irrespectively of the state where processing w started: we could require at the end (a) that the stack is always empty; or (b) that the stack contents are always the same (but not necessarily empty); or (c) that the stack contents are completely irrelevant upon entering the synchronizing state. They demonstrated in [36] some upper and lower bounds on the maximum length of the shortest synchronizing word for those three models of push-down automata, dependent on the stack height. Here, we study these three models from a complexity-theoretic perspective. However, as we show in our first main result, synchronizability becomes undecidable when asking about synchronizability in any of the stack models. Clearly, by restricting the length of a potential synchronizing word of some DPDA by an integer parameter (given in unary), we can observe that the corresponding synchronization problems all become NP-complete, as the hardness is trivially inherited from what is known about DFA synchronizability. For binary length-bounds, only membership in EXPTIME can be easily observed, since in general the stack might need to store the whole input word of exponential length. We leave to future work to determine the precise complexity status. Yet, it remains interesting to observe that with DFAs, introducing a length bound on the synchronizing word means an increase of complexity, while for DPDAs, this introduction means dropping from undecidability close to feasibility in the unary case. Beside general DPDAs, we will study these stack model variants of synchronization for sub-classes of DPDAs such as deterministic counter automata (DCA), deterministic (partially) blind automata and finite-turn variants of DPDAs and DCAs, which are arguably the most classical restrictions of push-down automata, as also testified by the discussions in [6]. In [17], further restricted sub-classes of DPDAs, such as visibly and very visibly deterministic push-down and counter automata are considered. There, all considered cases are in EXPTIME and even membership in P and PSPACE is observed, contrasting our undecidability results here.

Closest to the problems studied in our paper comes the work of Chistikov et al. [13] reviewed in the following, as their automaton model could be viewed as a special case of push-down automata, related to *input-driven pushdown automata* [35] which later became popular as *visibly push-down automata* [2]. In [13], the synchronization problem for so-called *nested word automata* (NWA) has been studied, where the concept of synchronization has been generalized to bringing all states to one single state such that for all runs the stack is empty (or in its start configuration) after reading the synchronizing word. In this setting, the synchronization problem is solvable in polynomial time, whereas the short synchronizing word problem is PSPACE-complete (here, the length bound is given in binary) and the question of synchronizing from or into a subset is EXPTIME-complete.

The DFA synchronization problem has been generalized in the literature to other automata models including infinite-state systems with infinite branching such as weighted and timed automata [15, 28, 44] or register automata [4]. For instance, register automata are infinite

state systems where a state consists of a control state and register contents. A synchronizing word for a register automaton brings all (infinitely many) states to the same state (and same register content). The synchronization problem for deterministic register automata (DRA) is PSPACE-complete and NLOGSPACE-complete for DRAs with only one register.

Finally, we want to mention that the term *synchronization of push-down automata* has already some occurrences in the literature, i.e., in [10, 3], but here the term *synchronization* refers to some relation of the input symbols to the stack behavior [10] or to reading different words in parallel [3] and is not to be confused with our notion of synchronizing states.

We are presenting an overview on our results at the end of the next section, where we introduce our notions more formally.

2 Definitions

We refer to the empty word as ϵ. For a finite alphabet Σ we denote 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 $[i] = \{1, 2, \ldots, i\}$. 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 $w[i]w[i+1]\ldots w[j]$ of w. We call $w[1..i]$ a prefix and $w[i..|w|]$ a suffix of w. The reversal of w is denoted by w^R, i.e., for $|w| = n$, $w^R = w[n]w[n-1]\ldots w[1]$.

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

We call $M = (Q, \Sigma, \Gamma, \delta, q_0, \bot, F)$ a *deterministic push-down automaton* (DPDA for short) if Q is a finite set of states; the finite sets Σ and Γ are the input and stack alphabet, respectively; δ is a transition function $Q \times \Sigma \times \Gamma \to Q \times \Gamma^*$; q_0 is the initial state; $\bot \in \Gamma$ is the stack bottom symbol which is only allowed as the first (lowest) symbol in the stack, i.e., if $\delta(q, a, \gamma) = (q', \gamma')$ and γ' contains $\bot$, then $\bot$ only occurs in γ' as its prefix and moreover, $\gamma = \bot$; and F is the set of final states. We will only consider *real-time* push-down automata and forbid ϵ-transitions, as can be seen in the definition.[1] Notice that the bottom symbol can be removed, but then the computation gets stuck.

Following [13], 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 $\longrightarrow$ the reflexive transitive closure of the union of $\xrightarrow{\sigma}$ over all letters in Σ. The input words on top of $\longrightarrow$ are concatenated accordingly, so that $\longrightarrow = \bigcup_{w \in \Sigma^*} \xrightarrow{w}$. The language $\mathcal{L}(M)$ accepted by a DPDA M is $\mathcal{L}(M) = \{w \in \Sigma^* \mid (q_0, \bot) \xrightarrow{w} (q_f, \gamma), q_f \in F\}$. We call the sequence of configurations $(q, \bot) \xrightarrow{w} (q', \gamma)$ the *run* induced by w, starting in q, and ending in q'. We might also call q' the *final state* of the run.

[1] Allowing ϵ-transitions, the considered automaton model is closer related to NFAs than to DFAs, and gives rise to adapt the concept of D1, D2, and D3 directing words (introduced in [27]).

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

As mentioned above, we will show that synchronizability of DPDAs is undecidable, which is in stark contrast to the situation with DFAs, where this problem is solvable in polynomial time. Hence, it is interesting to discuss deterministic variants of classical sub-classes of push-down automata. In this paper, we focus on one-counter languages and on linear languages and related classes. A *deterministic (one) counter automaton* (DCA) is a DPDA where $|\Gamma \backslash \{\bot\}| = 1$. Note that our DCAs can perform zero-tests by checking if the bottom-of-stack symbol is on top of the stack. As we will see that also with this restriction, synchronizability is still undecidable, we further restrict them to the *partially blind* setting [22]. This means in our formalization that a transition $\delta(q, \sigma, x) = (q', \gamma)$ either satisfies $x \neq \bot$, or x is a prefix of γ, i.e., $\gamma = x\gamma'$, and then both $\delta(q, \sigma, 1) = (q', 1\gamma')$ (for $\Gamma = \{1, \bot\}$) and $\delta(q, \sigma, \bot) = (q', \bot\gamma')$. The situation is even more delicate with one-turn or, more general, finite-turn DPDAs, whose further discussion and formal definition we defer to the specific section below.

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

▶ **Definition 1** (Sync-DPDA-Empty).
Given: DPDA $M = (Q, \Sigma, \Gamma, \delta, \bot)$.
Question: Does there exist a word $w \in \Sigma^*$ *that synchronizes* M *in the empty stack model?*

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

Outlook and summary of the paper

We summarize our results in Table 1. In short, while already seemingly innocuous extensions of finite automata (with counters or with 1-turn push-downs) result in an undecidable synchronizability problem, some extensions do offer some algorithmic synchronizability checks, although nothing efficient. At the end, we show how to apply some of our techniques to synchronizability questions concerning sequential transducers.

As an auxiliary result for proving undecidability of finding 1-turn synchronizing words for real-time deterministic push-down automata, we also prove undecidability of the inclusion and intersection non-emptiness problems for these automata, which could be an interesting result on its own. We also showcase how a variant of DFA synchronization, called DFA-Sync-From-Into-Subset, is useful to prove membership in PSPACE. As with this type of problems, these membership proofs are sometimes technical, this could be helpful in similar settings. To understand this problem, first look at the following one, called Global Inclusion Problem for Non-Initial Automata in [41]:

Table 1 Complexity status of the synchronization problem for different classes of deterministic real-time push-down automata in different stack synchronization modes as well as finite-turn variants of the respective synchronization problem.

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

▶ **Definition 2** (DFA-Sync-Into-Subset (PSPACE-complete, see Theorem 2.1 in [41])).
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 3** (DFA-Sync-From-Into-Subset).
Given: DFA $A = (Q, \Sigma, \delta)$*, subsets* $S_0, S_1 \subseteq Q$*.*
Question: Is there a word $w \in \Sigma^*$ *such that* $\delta(S_0, w) \subseteq S_1$*?*

Using the previously mentioned PSPACE-hardness and adapting a subset construction as in [42, Theorem 1.22], which then boils down to a reachability problem, one can show:

▶ **Proposition 4.** *DFA-Sync-From-Into-Subset is PSPACE-complete.*

The reader can find proofs for this and several other assertions in the long version of this paper [18].

3 General DCAs and DPDAs: When Synchronizability is Really Hard

The inclusion problem for deterministic real-time one counter automata that can perform zero-tests is undecidable [9, 37]. This result is used to prove undecidability of synchronization in any general setting as the main result of this section. However, there are special cases of DPDAs and DCAs that have a decidable inclusion problem (see [25] as an example) so that this argument does not apply to these sub-classes. We will have a closer look at some of these sub-classes in the following sections.

▶ **Theorem 5.** *The problems Sync-DCA-Empty, Sync-DCA-Same, and Sync-DCA-Arb are undecidable.*

Proof. We give a reduction from the undecidable intersection non-emptiness problem for real-time DCAs [9]. Let $M_1 = (Q_1, \Sigma, \{1, \bot\}, \delta_1, q_0^1, \bot, F_1)$ and $M_2 = (Q_2, \Sigma, \{1, \bot\}, \delta_2, q_0^2, \bot, F_2)$ be two DCAs over the same input alphabet with disjoint state sets. We construct a DCA $M_S = (Q_1 \cup Q_2 \cup \{q_f^1, q_f^2, q_s\}, \Sigma \cup \{a, b\}, \{1, \bot\}, \delta, \bot)$, where we neglect start and final states, which is synchronizable in the empty stack model if and only if the DCAs M_1 and M_2 accept a common word. The same construction also works for the same stack and arbitrary stack models. We assume $\{q_f^1, q_f^2, q_s\} \cap (Q_1 \cup Q_2) = \varnothing$ and $\{a, b\} \cap \Sigma = \varnothing$. For the states in Q_1 and Q_2, the transition function δ agrees with δ_1 and δ_2 for all letters in Σ. In the following, let $i \in \{1, 2\}$. For $q \in Q_i$, we set $\delta(q, a, \bot) = (q_0^i, \bot)$ and $\delta(q, a, 1) = (q_f^i, 1)$. Further, for $q \in Q_i \backslash F_i$ we set $\delta(q, b, 1) = (q_f^i, 1)$ and $\delta(q, b, \bot) = (q_f^i, \bot 1)$. For $q \in F_i$, we set $\delta(q, b, 1) = (q_s, 1)$ and

$\delta(q, b, \bot) = (q_s, \bot)$. For q_f^i, we set $\delta(q_f^i, a, \bot) = (q_0^i, \bot)$ with all other transitions we stay in q_f^i and increase the counter. Hence, the state q_f^i can only be left with an empty counter and this is only the case if no letter other than a has been read before. For the state q_s, we set $\delta(q_s, \Sigma \cup \{a, b\}, \bot) = (q_s, \bot)$, and $\delta(q_s, \Sigma \cup \{a, b\}, 1) = (q_s, \epsilon)$.

First, assume there is a word $w \in \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$. Then, the word awb synchronizes all states of the DCA M_S into the state q_s. Let l_1, l_2 be stack contents such that $(q_0^1, \bot) \xrightarrow{awb} (q_s, l_1)$ and $(q_0^2, \bot) \xrightarrow{awb} (q_s, l_2)$. Let $l = \max(|l_1|, |l_2|)$. Then $awbb^l$ synchronizes M_S in the empty stack model.

For the other direction assume there exists a word $w \in (\Sigma \cup \{a, b\})^*$ that synchronizes M_S in the empty stack model. The states q_f^1 and q_f^2 forces $w[1] = a$ since otherwise these states cannot be left. Since the state q_s has no outgoing transition, it must be our synchronizing state. In order to reach it, w must contain at least one letter b. Let $m \in [|w|]$ be an index such that $w[m] = b$ and for $j < b$, $w[j] \neq b$. With a letter b we move from all final states to the state q_s and from all non-final states of M_1 and M_2 we go to a state q_f^i and increase the counter. As we cannot leave the states q_f^i if we reach them once with a non-empty counter, reading a b from a non-final state causes the automaton to reach a configuration from which we no longer can synchronize the automaton. Hence, we know that after reading $w[1..m]$ all active states are in the set $F_1 \cup F_2 \cup \{q_s\}$. Let $\ell \in [|w|]$ be an index with $\ell < m$ with $w[\ell] = a$ such that for $\ell < i < m$, $w[i] \neq a$. Then $w[\ell + 1 \,..\, m - 1]$ is a word which is accepted by both DCAs M_1 and M_2.

It is easy to see that clearing the stacks in state q_s is not crucial and hence the reduction also works for the same stack and arbitrary stack models. ◀

▶ **Corollary 6.** *The problems* Sync-DPDA-Empty, Sync-DPDA-Same, *and* Sync-DPDA-Arb *are undecidable.* ◀

How can we overcome the problem that, even for deterministic one-counter languages, the synchronizability problem is undecidable? One of the famous further restrictions are (partially) blind counters, to which we turn our attention next.

4 Partially Blind Deterministic Counter Automata

The blind and partially blind variations of counter automata have been introduced by Greibach in [22]. She already noticed that the emptiness problem for such automata (even with multiple counters) is decidable should the reachability problem for vector addition systems, also known as Petri nets, be decidable, which has been proven some years later [34, 31]; its non-elementary complexity has only been recently fully understood [14]. Although we will stick to the models introduced so far in the following statements and proofs, we want to make explicit that our decidability results also hold for deterministic multi-counter automata. But as we focus on discussing families of automata describing languages between regular and context-free, we refrain from giving further details here.

Because partially blind counters can simulate blind counters, our results hold for blind counters as well, but we make them explicit only for the partially blind case. One formal reason is that we want to preserve our stack model, while it becomes awkward to formalize blind counters in this stack model.

Recall that a partially blind counter automaton will get blocked when its counter gets below zero. The blindness refers to the fact that such a machine can never explicitly test its counter for zero. This translates into our formalization by requiring that a transition $\delta(q, \sigma, x) = (q', \gamma)$ either means that $x \neq \bot$ or x is a prefix of γ, i.e., $\gamma = x\gamma'$, and then both

$\delta(q,\sigma,1) = (q', 1\gamma')$ (for $\Gamma = \{1, \perp\}$) and $\delta(q,\sigma,\perp) = (q', \perp\gamma')$, i.e., the processing will somehow perform the same action, irrespectively of the stack contents, whenever possible; however, the machine will stop if it is trying to pop the bottom-of-stack symbol. As a specialty, such automata accept when having arrived in a final state together with having zero in its counter. A deterministic partially blind (one-)counter automaton is called a DPBCA.

▶ **Theorem 7.** *The problems* Sync-DPBCA-Empty, Sync-DPBCA-Same, *and* Sync-DPBCA-Arb *are decidable.*

We are not specifying the complexity here, but only mention that we are using, in the end, the reachability problem for Petri nets, which is known to be decidable, but only with a non-elementary complexity; see [34, 31, 14]. However, we leave it as an open question if the synchronization complexity of DPBCAs is non-elementary. When looking into this question more in details, the number of states of the counter automaton could be a useful parameter to be discussed, as it influences the number of counters of the partially blind multi-counter automaton that we construct in our proof in order to show the claimed decidability result.

Proof. Let $M = (Q, \Sigma, \{1, \perp\}, \delta, q_0, \perp, F)$ be some DPBCA. Let us first describe the case Sync-DPBCA-Empty. Here, we can first produce the multi-counter $|Q|$-fold product automaton $M^{|Q|}$ from M that starts, assuming $Q = \{q_0, \ldots, q_{|Q|-1}\}$, in the state $(q_0, \ldots, q_{|Q|-1})$. Notice that $M^{|Q|}$ has $|Q|^{|Q|}$ many states and operates $|Q|$ many counters. We could take as the set of final states $F^{|Q|} = \{(q, \ldots, q) \mid q \in Q\}$. This mimicks state synchronization of M: any word that synchronizes all states of M will drive $M^{|Q|}$ into F. As mentioned above, partially blind multi-counter automata accept with final states and empty stacks, so that M is synchronizable in the empty stack model if and only if $M^{|Q|}$ accepts any word.

For the arbitrary stack model, we have to count down (removing 1 from any of the the stacks sequentially) until the bottom-of-stack symbol appears on all stacks on top (at the same time), leading to the variant $M^{|Q|}_{\text{Arb}}$. These are moves without reading the input (or reading arbitrary symbols at the end, this way only prolonging a possibly synchronizing word), but this does not matter, as the emptiness problem is decidable for partially blind nondeterministic multi-counter automata. It should be clear that $M^{|Q|}_{\text{Arb}}$ accepts any word if and only if M is synchronizable. For the case Sync-DPBCA-Same, the counting down at the end should be performed in parallel for all counters instead. ◀

5 Finite-Turn DPDAs

Finite-turn PDAs are introduced in [21]. From the formal language side, it is known that one-turn PDAs characterize the rather familiar family of linear context-free languages, usually defined via grammars. In our setting, the automata view is more interesting. We adopt the definition in [49]. For a DPDA M, an *upstroke* of M is a sequence of configurations induced by an input word w such that no transition decreases the stack-height. Accordingly, a *downstroke* of M is a sequence of configurations in which no transition increases the stack-height. A stroke is either an upstroke or a downstroke. Note that exchanging the top symbol of the stack is allowed in both an up- and a downstroke. A DPDA M is an n-turn DPDA if for all $w \in \mathcal{L}(M)$ the sequence of configurations induced by w can be split into at most $n+1$ strokes. Especially for 1-turn DPDAs, each sequence of configurations induced by an accepting word consists of one upstroke followed by a most one downstroke. There are two subtleties when translating this concept to synchronization: (a) there is no initial state so that there is no

way to associate a stroke counter to a state, and (b) there is no language of accepted words that restricts the set of words on which the number of strokes should be limited. We therefore generalize the concept of finite-turn DPDAs to finite-turn synchronization for DPDAs in the following way. This opens up quite an interesting complexity landscape.

▶ **Definition 8.** *n-Turn-Sync-DPDA-Empty*
Given: DPDA $M = (Q, \Sigma, \Gamma, \delta, q_0, \bot, F)$.
Question: Is there a synchronizing word $w \in \Sigma^*$ in the empty stack model, such that for all states $q \in Q$, the sequence of configurations $(q, \bot) \xrightarrow{w} (\overline{q}, \bot)$ consists of at most $n+1$ strokes?

We call such a synchronizing word w an *n-turn synchronizing word* for M. We define n-Turn-Sync-DPDA-Same and n-Turn-Sync-DPDA-Arb accordingly for the same stack and arbitrary stack models. Further, we extend the problem definition to real-time DCAs.

Motivated by the proof of Theorem 5, we are first reviewing the status of the inclusion problem for 1-turn DPDAs in the literature.

▶ Remark 9. The inclusion problem for 1-turn DPDAs with ϵ-transitions is undecidable [19, 49]. The intersection non-emptiness problem for real-time 1-turn non-deterministic push-down automata is also undecidable [29]. The decidability of the inclusion and intersection non-emptiness problems for *real-time* 1-turn *deterministic* push-down automata have not been settled in the literature; we will do so below by proving undecidability for both problems.

We will present a reduction from the undecidable Post Correspondence Problem (PCP for short) [39] to the intersection non-emptiness for real-time 1-turn DPDAs which also implies undecidability of the inclusion problem for this class since it is closed under complement.

▶ **Definition 10** (PCP)**.**
Given: Two lists of input words over $\{0,1\}$: $A = (a_1, a_2, \ldots, a_n)$, $B = (b_1, b_2, \ldots, b_n)$.
Question: Is there a sequence of indices $i_1, i_2, \ldots, i_k$ with $i_j \in [n]$ for $1 \le j \le k$ such that $a_{i_1} a_{i_2} \ldots a_{i_k} = b_{i_1} b_{i_2} \ldots b_{i_k}$?

Observe that already Post stated this problem over binary alphabets. Much later, Matiyasevich and Sénizergues [33] showed that indeed lists of length seven are sufficient for undecidability. This was recently lowered to lists of length five by Neary [38].

▶ **Theorem 11.** *Let M_1 and M_2 be two real-time 1-turn DPDAs. Then the following problems are undecidable: Is $\mathcal{L}(M_1) \cap \mathcal{L}(M_2) = \varnothing$? Is $\mathcal{L}(M_1) \subseteq \mathcal{L}(M_2)$?*

Proof. Let $A = (a_1, a_2, \ldots, a_n)$, $B = (b_1, b_2, \ldots, b_n)$ be an instance of PCP. We construct from A a real-time 1-turn DPDA $M_A = (Q, \{0,1,\#,\$\} \cup [\overline{n}], \{0,1,\bot\}, \delta, q_0, \bot, \{q_f\})$, where $[\overline{n}] = \{\overline{1}, \overline{2}, \ldots, \overline{n}\}$ are marked numbers from 1 to n. The set Q contains the start state q_0, the states $\overline{q_0}$, q_{check} and q_{fail}, and the single final state q_f. The rest of Q is a partition into state sets $Q_1, Q_2, \ldots, Q_n$ such that the (deterministic partial) sub-automaton induced by Q_i, with start state q_0^i, reads the string $a_i \# b_i$ and thereby pushes each symbol of a_i on the stack, whereas symbols of the string b_i leave the stack content unchanged. The sub-automaton induced by Q_i is embedded into M_A, and thereby completed, by taking the state $\delta(q_0^i, a_i \# b_i)$ to $\overline{q_0}$ with the symbol $\#$. We go from q_0 and $\overline{q_0}$ to the initial state of the sub-automaton induced by Q_i by the letter $\overline{i}$. With the letter \$, the state $\overline{q_0}$ maps to q_{check}. Here, if the input symbol equals the symbol on top of the stack, we pop the stack and stay in q_{check} until we reach the bottom symbol $\bot$, in which case an additional letter \$ brings us to the final state q_f. If the input symbol does not equal the symbol on top of the state we go to q_{fail}, which is a trap state for all letters. Every other not yet defined transition on Q maps to the state q_{fail}. If not stated otherwise, every transition leaves the stack content unchanged.

For the list B, we construct a real-time 1-turn DPDA M_B in a similar way except that here we push the strings b_i on the stack and symbols of strings a_i leave the stack unchanged.

The languages accepted by M_A and M_B are the following ones:

$$\mathcal{L}(M_A) = \{\, \overline{i_1} a_{i_1} \# b_{i_1} \# \overline{i_2} a_{i_2} \# b_{i_2} \# \cdots \overline{i_m} a_{i_m} \# b_{i_m} \# \$ a_{i_m}^R \cdots a_{i_1}^R \$ \mid m \geq 1, i_j \in [n], j \leq m \}$$

$$\mathcal{L}(M_B) = \{\, \overline{i_1} a_{i_1} \# b_{i_1} \# \overline{i_2} a_{i_2} \# b_{i_2} \# \cdots \overline{i_m} a_{i_m} \# b_{i_m} \# \$ b_{i_m}^R \cdots b_{i_1}^R \$ \mid m \geq 1, i_j \in [n], j \leq m \}$$

Obviously, the given PCP has a solution if and only if $\mathcal{L}(M_A) \cap \mathcal{L}(M_B) \neq \varnothing$.

By complementing the set of final states, from M_A one would arrive at a real-time 1-turn DPDA M'_A such that $\mathcal{L}(M'_A)$ is the complement of $\mathcal{L}(M_A)$, so that the given PCP has no solution if and only if $\mathcal{L}(M'_A) \supseteq \mathcal{L}(M_B)$. ◀

We will now adapt the presented construction to show that the synchronization problem for real-time one-turn DPDAs is undecidable in all three synchronization models.

▶ **Theorem 12.** *The problems 1-Turn-Sync-DPDA-Empty, 1-Turn-Sync-DPDA-Same, and 1-Turn-Sync-DPDA-Arb are undecidable.*

Proof. Let M_A and M_B be the real-time 1-turn DPDAs from the proof in Theorem 11. We take these machines as inputs for the construction in the proof of Theorem 5 to obtain the DPDA M. Therefore, observe that the construction also works if the input machines are general DPDAs and not only DCAs. Further, observe that for M_A and M_B, if for both machines the only active state is the final state, then the stack of all runs is empty and hence the stack need not be altered in the synchronizing state and all transitions here can act as the identity and leave the stack unchanged.

If w is a word in $\mathcal{L}(M_A) \cap \mathcal{L}(M_B)$, then awb synchronizes M in the empty, same, and arbitrary stack models; further awb is a 1-turn synchronizing word for M. Conversely, if w is a 1-turn synchronizing word for M, then w must be of the form avb or $auavb$ where $v \in \mathcal{L}(M_A) \cap \mathcal{L}(M_B)$ and u is a word that does not change the stack, as otherwise M could either not be synchronized or the 1-turn condition is violated. To be more precise, a must be the first letter of w as otherwise we get stuck in q_f^1 and q_f^2. The letter a resets the machines M_A and M_B to their initial state and can only be read when the stack is empty as otherwise the machine gets stuck. In order to reach final states, both machines M_A and M_B must increase and decrease the stack by reading some word v, but as soon as we increased the stack once, we are not allowed to reset the machine anymore due to the 1-turn condition. Hence, letters a can only be read while the stack has not been changed yet. Note that for all three stack models, the construction enforces that any 1-turn synchronizing word brings M into a configuration where the stack is empty. ◀

When considering automata as language accepting devices, there is no good use of 0-turn PDAs, as they cannot exploit their stack. This becomes different if synchronization requires to end in the same configuration, which means that in particular the stack contents are identical.

▶ **Theorem 13.** *The problem 0-Turn-Sync-DPDA-Same is undecidable.*

Proof sketch. The proof is by a straight-forward adaption of the previously presented constructions by getting rid of the check phase; instead, check with the same stack condition that the two words of the PCP coincide. As we never pop the stack, a 0-turn DPDA will be sufficient in the construction. ◀

The picture changes again for other 0-turn stack models, but remains intractable.

▶ **Theorem 14.** *The problems 0-Turn-Sync-DCA-Empty, 0-Turn-Sync-DCA-Same and 0-Turn-Sync-DCA-Arb are PSPACE-hard.*

Proof. We give a reduction from DFA-Sync-Into-Subset. Let $A = (Q, \Sigma, \delta)$ be a DFA with $S \subseteq Q$. We construct from A a DCA $M = (Q \cup \{q_{\text{stall}}, q_{\text{sync}}\}, \Sigma \cup \{a\}, \{N, \bot\}, \delta', \bot)$ where all unions are disjoint. For $q \in Q$, $\sigma \in \Sigma$ and $\gamma \in \{N, \bot\}$, set $\delta'(q, \sigma, \gamma) = (\delta(q, \sigma), \gamma)$. For the letter a, we set for states $q \in S$, $\delta'(q, a, \bot) = (q_{\text{stall}}, \bot)$ and for states $q \in Q \backslash S$, we set $\delta'(q, a, \bot) = (q_{\text{stall}}, N)$. For q_{stall} we set $\delta'(q_{\text{stall}}, a, \bot) = (q_{\text{sync}}, \bot)$ and $\delta'(q_{\text{stall}}, a, N) = (q_{\text{stall}}, N)$. All transitions not yet defined act as the identity and leave the stack unchanged.

First, assume there exists a word $w \in \Sigma^*$ that synchronizes Q into S in the DFA A. Then clearly waa synchronizes M in the arbitrary stack model. Now, assume there exists a word $w \in (\Sigma \cup \{a\})^*$ that synchronizes M in the arbitrary stack model. Then, w must contain at least two occurrences of a to bring all states into the sink state q_{sync}. In order to reach q_{sync} the states in Q need to pass through the state q_{stall} but by doing so, it is noted on the stack if an active state transitions from $Q \backslash S$ into q_{stall} and only the active states coming from S are allowed to pass on to q_{sync}. In q_{stall} the stack content cannot be changed and hence the prefix of w up to the first occurrence of the letter a must have already synchronized Q into S in the DFA A. Note that M can only be synchronized by a word that leaves all stacks empty, hence the result follows for all three stack models. ◀

▶ **Corollary 15.** *The problems 0-Turn-Sync-DPDA-Empty and 0-Turn-Sync-DPDA-Arb are PSPACE-hard.*

Proof. The claim follows from Theorem 14 by inclusion of automata classes. ◀

▶ **Theorem 16.** *0-Turn-Sync-DPDA-Empty, 0-Turn-Sync-DPDA-Arb ∈ PSPACE.*

Proof sketch. In the empty stack model, the 0-turn condition forbids us to write anything on the stack at all. Hence, consider a 0-turn DPDA with state set Q as a DFA A, with one additional state q_{trap} which is entered whenever the DPDA wants to push or pop anything. For each $q \in Q$, run the PSPACE-algorithm for the Sync-From-Into-Subset instance $(A, Q, \{q\})$. For the arbitrary stack model, due to the 0-turn condition, for each run only the symbol on top of the stack must be ever stored. Consider a 0-turn DPDA with state set Q and stack alphabet Γ as a DFA A with state set $Q \times \Gamma$ and one additional state q_{trap} which is entered whenever the DPDA wants to pop anything. For each $q \in Q$, run the PSPACE-algorithm for the Sync-From-Into-Subset instance $(A, Q \times \{\bot\}, \{q\} \times \Gamma)$. ◀

▶ **Theorem 17.** *1-Turn-Sync-DCA-Empty, 1-Turn-Sync-DCA-Same, and 1-Turn-Sync-DCA-Arb in PSPACE.*

Notice that PSPACE-hardness is inherited from corresponding results for visibly counter automata, as obtained in [17].

Proof. Let $M = (Q, \Sigma, \Gamma, \delta, \bot)$ be a DCA. As we are looking into 1-turn behavior, any computation that we are interested would split into two phases: in the first upstroke phase, the counter is incremented or stays constant, while in the second downstroke phase, the counter is decremented or stays constant. In particular, because the counter is 1-turn, after the first counter increment, any zero-test will always return false, while in the downstroke phase, when zero-tests return true, then all future computations cannot decrement the counter any further, so that at the end, the counter will also contain zero. We are formalizing this intuition to create a machine that has an awareness about its phase stored in its states and that is behaving very similar. In the rest of the proof, a *spread-out variant* of a word $a_1 a_2 \cdots a_n$ of length n, with symbols a_i from Σ, is any word in $a_1 \Sigma a_2 \Sigma \cdots a_n \Sigma$ of length $2n$.

We will now construct from M and $q \in Q$ a deterministic 1-turn counter automaton M_q that accepts precisely all spread-out variants of words that M would accept when starting in state q and finishing its 1-turn computation (in any state) with the empty stack, but that keeps track of a basic property of managing the counter in so-called stages. M_q has the state set $Q \times \{1,2,3,4\} \times \{0,1\}$, $(q,1,0)$ as its initial state, and as its set of final states, take $Q \times \{1,4\} \times \{0\}$. The transitions of δ_q can be defined with the following semantics in mind (details are given in [18]): (a) the last bit always alternates, (b) the spread-out is used to enable a *deterministic* work and to make sure that the simulated machine has counter content zero if the simulating automaton M_q is in one of the states from $Q \times \{1,4\} \times \{0\}$, (c) M_q changes from $Q \times \{1\} \times \{0,1\}$ to $Q \times \{2\} \times \{0,1\}$ if the counter is no longer zero, so that the simulated machine has "properly" entered the upstroke phase, (d) M_q changes from $Q \times \{2\} \times \{0,1\}$ to $Q \times \{3\} \times \{0,1\}$ if the simulated machine made its first pop, i.e., it "properly" entered the downstroke phase, (e) M_q changes from $Q \times \{3\} \times \{0,1\}$ to $Q \times \{4\} \times \{0,1\}$ if the counter has become zero again.

Now, we build the $|Q|$-fold product automaton $M_{\text{Empty}}^{|Q|}$ from all automata M_q with the start state $((q_1,1,0),(q_2,1,0),\ldots,(q_{|Q|},1,0))$, assuming $Q = \{q_1,\ldots,q_{|Q|}\}$. This means that $M_{\text{Empty}}^{|Q|}$ has $(8|Q|)^{|Q|}$ many states and $|Q|$ many counters, each of which makes at most one turn. Now observe that a word w synchronizes M with empty stack, say, in state p, if and only if any spread-out variant of w drives $M_{\text{Empty}}^{|Q|}$ into a state $((p,i_1,0),(p,i_2,0),\ldots,(p,i_{|Q|},0))$ for some $i_j \in \{1,4\}$ for all $1 \leq j \leq |Q|$. Now, define $\{((p,i_1,0),(p,i_2,0),\ldots,(p,i_{|Q|},0)) \mid p \in Q, i_j \in \{1,4\} \text{ for } 1 \leq j \leq |Q|\}$ as final states of $M_{\text{Empty}}^{|Q|}$. We see that M is synchronizable with empty stack if and only if $M_{\text{Empty}}^{|Q|}$ accepts any word. As Gurari and Ibarra have shown in [24, Lemma 2], $M_{\text{Empty}}^{|Q|}$ accepts any word if and only if it accepts any word up to length $(|Q|(8|Q|)^{|Q|}|\Sigma|)^{O(|Q|)} \leq (|Q|(8|Q||\Sigma|)^{O(|Q|)^2})$,[2] within the same time bounds. Now, testing all these words for membership basically needs two counters that are able to capture numbers of size $(|Q|(8|Q||\Sigma|)^{O(|Q|)^2})$, which means we need polynomial space in $|Q|$ and $|\Sigma|$ to check if M is synchronizable with empty stack.

By considering $M_{\text{Empty}}^{|Q|}$ with final states $((p,i_1,0),(p,i_2,0),\ldots,(p,i_{|Q|},0))$ for arbitrary $p \in Q$ and $i_j \in \{1,2,3,4\}$, this way defining an automaton $M_{\text{Arb}}^{|Q|}$, we can check if M is synchronizable in the arbitrary stack model also in polynomial space with the same argument. From $M_{\text{Empty}}^{|Q|}$, we can also construct a nondeterministic 1-turn $|Q|$-counter machine $M_{\text{Same}}^{|Q|}$ by adding a nondeterministic move from $((p,i_1,0),(p,i_2,0),\ldots,(p,i_{|Q|},0))$ for arbitrary $p \in Q$ and $i_j \in \{1,2,3,4\}$ upon reading some arbitrary but fixed $\sigma_{\text{sync}} \in \Sigma$ to a special state q_{dec} in which state we loop upon reading $\sigma_{\text{sync}} \in \Sigma$, decrementing all counters at the same time; finally, there is the possibility to move to q_f (that is the only finaly state now) upon reading $\sigma_{\text{sync}} \in \Sigma$ if all counters are empty. Notice that also this automaton $M_{\text{Same}}^{|Q|}$ has $(\mathcal{O}(|Q|))^{|Q|}|\Sigma|$ many transitions, so that with using [24, Lemma 2], we can again conclude that synchronizability with same stack can be checked in polynomial space for M. ◀

[2] In the cited lemma, the number of steps of the checking machine is upper-bounded by $(ms)^{O(m)}$, where m is the number of 1-turn counters and s is the number of transitions of the machine.

6 Sequential Transducers

We will now introduce a new concept of synchronization of sequential transducers.[3] We call $T = (Q, \Sigma, \Gamma, q_0, \delta, F)$ a *sequential transducer* (ST for short) if Q is a finite set of states, Σ is a finite input alphabet, Γ is a finite output alphabet, q_0 is the start state, $\delta\colon Q \times \Sigma \to Q \times \Gamma^*$ is a total transition function, and F is a set of final states. We generalize δ from input letters to words by concatenating the produced outputs, i.e., for $q, q', q'' \in Q$, $\sigma_1, \sigma_2 \in \Sigma$, $\gamma_1, \gamma_2 \in \Gamma^*$ and transitions $\delta(q, \sigma_1) = (q', \gamma_1), \delta(q', \sigma_2) = (q'', \gamma_2)$ we define $\delta(q, \sigma_1\sigma_2) = (q'', \gamma_1\gamma_2)$. We say that a word w *trace-synchronizes* a sequential transducer T if for all states $p, q \in Q$ it holds that $\delta(p, w) = \delta(q, w)$. Intuitively, w brings all states of T to the same state and produces the same output on all states. It can be viewed as a reset word, which is not a separating sequence for any pair of states (see [42] for separating sequences concerning homing and synchronizing sequences of transducers). Again, we might neglect start and final states.

▶ **Definition 18** (Trace-Sync-Transducer)**.**
Given: Sequential transducer $T = (Q, \Sigma, \Gamma, \delta)$.
Question: Does there exists a word $w \in \Sigma^*$ *that trace-synchronizes* T*?*

▶ **Theorem 19.** *The problem Trace-Sync-Transducer is undecidable.*

Proof. (Sketch) We adapt the construction of M in Theorem 13 to obtain a sequential transducer T in the following way: Each time we push a letter to the stack in Q_i^A or Q_i^B, instead we now output this letter. Whenever we leave the stack unchanged, we now simply do not produce any output. The trace-synchronizing condition now steps in and ensures that the the stack contents (in the notion of Theorem 13) are equal. ◀

Observe that relations between undecidability questions of transducers and context-free grammars have been previously noticed in [7, 23, 26].

7 Prospects

It would be interesting to look into the synchronization problem for further automata models. In view of the undecidability results that we obtained in this paper, a special focus should be to look into deterministic automata classes with a known decidable inclusion problem, as otherwise it should be possible to adapt our undecidability proofs for synchronizability to these automata models. To make this research direction more clear: (a) There are quite efficient algorithms for the inclusion problem for so-called *very simple deterministic pushdown automata*, see [51]; (b) a proper super-class of these languages are so-called *NTS languages* that also have a deterministic automaton characterization[4] but their inclusion problem is undecidable, see [43, 8]. The overall aim of this research would be to find the borderline between decidable and undecidable synchronizability and, moreover, within the decidable part, to determine the complexity of this problem. A step in this research direction has been made in direction of (sub-classes of) visibly deterministic pushdown automata in [17]. Interestingly enough, that research line also revealed some cases where synchronizability can be decided in polynomial time, quite in contrast to the situation found in the present study.

[3] The definitions in the literature are not very clear for finite automata with outputs. We follow here the name used by Berstel in [6]; Ginsburg [20] called Berstel's sequential transducers *generalized machines*, but used the term *sequential transducer* for the nondeterministic counterpart.

[4] This is rather implicit in the literature, which is one of the reasons why we do not present more details here; one would have to first define the automaton model properly.

Another approach is to look into variants of synchronization problems for DPDAs, such as restricting the length of a potential synchronizing word. It follows from the NP-hardness of this problem for DFAs [40, 16] and the polynomial-time solvability of the membership problem for DPDAs that for unary encoded length bounds this problem is NP-complete for DPDAs as well, and contained in EXPTIME for binary encoded length bounds. The precise complexity status of this problem for binary encoded length bounds could be a topic of future research.

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 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.

4 Parvaneh Babari, Karin Quaas, and Mahsa Shirmohammadi. Synchronizing Data Words for Register Automata. In Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier, editors, *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, volume 58 of *LIPIcs*, pages 15:1–15:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

5 Marie-Pierre Béal and Dominique Perrin. *Synchronised Automata*, page 213–240. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2016.

6 Jean Berstel. *Transductions and Context-Free Languages*, volume 38 of *Teubner Studienbücher: Informatik*. Teubner, 1979.

7 Meera Blattner. The unsolvability of the equality problem for sentential forms of context-free grammars. *J. Comput. Syst. Sci.*, 7(5):463–468, 1973. doi:10.1016/S0022-0000(73)80002-9.

8 Luc Boasson and Géraud Sénizergues. NTS Languages Are Deterministic and Congruential. *Journal of Computer and System Sciences*, 31(3):332–342, 1985.

9 Stanislav Böhm and Stefan Göller. Language Equivalence of Deterministic Real-Time One-Counter Automata Is NL-Complete. In Filip Murlak and Piotr Sankowski, editors, *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*, volume 6907 of *Lecture Notes in Computer Science*, pages 194–205. Springer, 2011.

10 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.

11 Ján Černý. Poznámka k homogénnym eksperimentom s konečnými automatami. *Matematicko-fyzikalny Časopis Slovensk*, 14(3):208–215, 1964.

12 Ján Cerný. A Note on Homogeneous Experiments with Finite Automata. *Journal of Automata, Languages and Combinatorics*, 24(2-4):123–132, 2019.

13 Dmitry Chistikov, Pavel Martyugin, and Mahsa Shirmohammadi. Synchronizing Automata over Nested Words. *Journal of Automata, Languages and Combinatorics*, 24(2-4):219–251, 2019.

14 Wojciech Czerwiński, Slawomir Lasota, Ranko Lazić, Jérôme Leroux, and Filip Mazowiecki. The Reachability Problem for Petri Nets is Not Elementary. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 24–33. ACM, 2019.

15 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.

16 David Eppstein. Reset Sequences for Monotonic Automata. *SIAM Journal on Computing*, 19(3):500–510, 1990.

17 Henning Fernau and Petra Wolf. Synchronization of Deterministic Visibly Push-Down Automata. *CoRR*, abs/2005.01374, 2020. `arXiv:2005.01374`.

18 Henning Fernau, Petra Wolf, and Tomoyuki Yamakami. Synchronizing Deterministic Push-Down Automata Can Be Really Hard. *CoRR*, abs/2005.01381, 2020. `arXiv:2005.01381`.

19 Emily P. Friedman. The Inclusion Problem for Simple Languages. *Theoretical Computer Science*, 1(4):297–316, 1976.

20 Seymour Ginsburg. *The Mathematical Theory of Context-Free Languages*. McGraw-Hill, 1966.

21 Seymour Ginsburg and Edwin H Spanier. Finite-Turn Pushdown Automata. *SIAM Journal on Control*, 4(3):429–453, 1966.

22 Sheila A. Greibach. Remarks on Blind and Partially Blind One-Way Multicounter Machines. *Theoretical Computer Science*, 7:311–324, 1978.

23 Timothy V. Griffiths. The unsolvability of the equivalence problem for lambda-free nondeterministic generalized machines. *J. ACM*, 15(3):409–413, 1968. `doi:10.1145/321466.321473`.

24 Eitan M. Gurari and Oscar H. Ibarra. The Complexity of Decision Problems for Finite-Turn Multicounter Machines. *Journal of Computer and System Sciences*, 22(2):220–229, 1981.

25 Ken Higuchi, Mitsuo Wakatsuki, and Etsuji Tomita. A Polynomial-Time Algorithm for Checking the Inclusion for Real-Time Deterministic Restricted One-Counter Automata Which Accept by Final State. *IEICE Transactions on Information and Systems*, 78-D(8):939–950, 1995.

26 Oscar H. Ibarra. The unsolvability of the equivalence problem for epsilon-free ngsm's with unary input (output) alphabet and applications. *SIAM J. Comput.*, 7(4):524–532, 1978. `doi:10.1137/0207042`.

27 Balázs Imreh and Magnus Steinby. Directable Nondeterministic Automata. *Acta Cybernetica*, 14(1):105–115, 1999.

28 Szabolcs Iván. Synchronizing weighted automata. In Zoltán Ésik and Zoltán Fülöp, editors, *Proceedings 14th International Conference on Automata and Formal Languages, AFL 2014, Szeged, Hungary, May 27-29, 2014*, volume 151 of *EPTCS*, pages 301–313, 2014. `doi:10.4204/EPTCS.151.21`.

29 Changwook Kim. Quasi-Rocking Real-Time Pushdown Automata. *Theoretical Computer Science*, 412(48):6720–6735, 2011.

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

31 S. Rao Kosaraju. Decidability of Reachability in Vector Addition Systems (Preliminary Version). In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 267–281. ACM, 1982.

32 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.

33 Yuri V. Matiyasevich and Géraud Sénizergues. Decision Problems for Semi-Thue Systems with a few Rules. *Theoretical Computer Science*, 330(1):145–169, 2005.

34 Ernst W. Mayr. An Algorithm for the General Petri Net Reachability Problem. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing, May 11-13, 1981, Milwaukee, Wisconsin, USA*, pages 238–246. ACM, 1981.

35 Kurt Mehlhorn. Pebbling Moutain Ranges and its Application of DCFL-Recognition. In J. W. de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming, 7th*

Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings, volume 85 of *Lecture Notes in Computer Science*, pages 422–435. Springer, 1980.

36 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. An English translation is in preparation.

37 Marvin L Minsky. Recursive Unsolvability of Post's Problem of "Tag" and Other Topics in Theory of Turing Machines. *Annals of Mathematics*, pages 437–455, 1961.

38 Turlough Neary. Undecidability in Binary Tag Systems and the Post Correspondence Problem for Five Pairs of Words. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 649–661. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

39 Emil L. Post. A Variant of a Recursively Unsolvable Problem. *Bulletin of the American Mathematical Society*, 52(4):264–268, 1946.

40 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).

41 I. K. Rystsov. Polynomial Complete Problems in Automata Theory. *Information Processing Letters*, 16(3):147–151, 1983.

42 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.

43 Géraud Sénizergues. The Equivalence and Inclusion Problems for NTS Languages. *Journal of Computer and System Sciences*, 31(3):303–331, 1985.

44 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`.

45 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.

46 Peter H. Starke. Eine Bemerkung über homogene Experimente. *Elektronische Informationsverarbeitung und Kybernetik*, 2(4):257–259, 1966.

47 Peter H. Starke. A Remark About Homogeneous Experiments. *Journal of Automata, Languages and Combinatorics*, 24(2-4):133–137, 2019.

48 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.

49 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/`.

50 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.

51 Mitsuo Wakatsuki and Etsuji Tomita. A Fast Algorithm for Checking the Inclusion for Very Simple Deterministic Pushdown Automata. *IEICE Transactions on Information and Systems*, 76-D(10):1224–1233, 1993.