Synchronizing Data Words for Register Automata

Parvaneh Babari¹, Karin Quaas², and Mahsa Shirmohammadi³

1 Universität Leipzig, Germany
2 Universität Leipzig, Germany
3 University of Oxford, United Kingdom

Abstract

Register automata (RAs) are finite automata extended with a finite set of registers to store and compare data. We study the concept of synchronizing data words in RAs: Does there exist a data word that sends all states of the RA to a single state?

For deterministic RAs with k registers (k-DRAs), we prove that inputting data words with $2k+1$ distinct data, from the infinite data domain, is sufficient to synchronize. We show that the synchronizing problem for DRAs is in general \textit{PSPACE}-complete, and is \textit{NLOGSPACE}-complete for 1-DRAs. For nondeterministic RAs (NRAs), we show that $\text{Ackermann}(n)$ distinct data (where $n$ is the size of RA) might be necessary to synchronize. The synchronizing problem for NRAs is in general undecidable, however, we establish \text{Ackermann}-completeness of the problem for 1-NRAs. Our most substantial achievement is proving \text{NEXPTIME}-completeness of the length-bounded synchronizing problem in NRAs (length encoded in binary). A variant of this last construction allows to prove that the bounded universality problem in NRAs is \text{co-NEXPTIME}-complete.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Data Words, Register Automata, Synchronizing Problem, \text{Ackermann}-completeness, Bounded Universality, Regular-like expressions with squaring

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.15

1 Introduction

Synchronizing words for finite automata have been studied since the 70’s, see [8, 26, 32, 24]; such a word $w$ drives the automaton from an unknown or unobservable state to a specific state $q_w$ that only depends on $w$. The famous Černý conjecture on synchronizing words is a long-standing open problem in automata theory. The conjecture claims that the length of a shortest synchronizing data word for a deterministic finite automaton (DFA) with $n$ states is at most $(n-1)^2$. There exists a family of DFAs, where the length of the shortest synchronizing word is exactly $(n-1)^2$, which attains the exact claimed bound in the conjecture. Despite all received attention in last decades, this conjecture has not been proved or disproved.

Synchronizing words have applications in planning, control of discrete event systems, biocomputing, and robotics [3, 32, 16]. Over the past few years, this classical notion has sparked renewed interest thanks to its generalization to games on transition systems [22, 29, 21], and to infinite-state systems [15, 10], which are more relevant for modelling complex systems such as distributed data networks or real-time embedded systems. These studies have inspired an elegant extension of temporal logics to capture synchronizing properties [9]; the proposed logic is more expressive than classical computation tree logic.
Synchronizing Data Words for Register Automata

In this paper, we are interested in synchronizing data words for register automata. Data words are sequences of pairs where the first element is taken from a finite alphabet and the second element is taken from an infinite data domain such as natural numbers or ASCII strings. In recent years, this structure has become an active subject of research thanks to applications in querying and reasoning about data models with complex structural properties, in XML, and lately also in graph databases [17, 2, 1, 5]. For reasoning about data words, various formalisms have been considered, ranging over first-order logic for data words [4, 6], extensions of linear temporal logic [23, 13, 12, 14], data automata [4, 7], register automata [20, 27, 25, 12] and extensions thereof, e.g. [31, 18, 11].

Register automata (RAs) are a natural generalization of finite automata over data words, and are equipped with a finite set of registers. When processing a data word, the automaton may store the data value of the current position in one or more registers. It may also test the data value of the current position for equality with the values stored in the registers, where the result of this test determines how the RA evolves. This allows for handling parameters like user names, passwords, identifiers of connections, sessions, etc., in a fashion similar to, and more expressive than, the class of data-independent systems. RAs come in different variants, e.g., one-way vs. two-way, deterministic vs. non-deterministic, alternating vs. non-alternating. For alternating RAs, classical decision problems like non-emptiness, universality and language inclusion are undecidable. We focus on the class of one-way RAs without alternation: They have a decidable non-emptiness problem [20], and the subclass of nondeterministic RAs with a single register has a decidable non-universality problem [12].

Semantically, an RA defines an infinite-state system, due to the unbounded domain for data stored in registers. Synchronizing words were introduced for infinite-state systems with infinite branching in [15, 29]; in particular, the notion of synchronizing words is motivated and studied for weighted automata and timed automata. In some infinite-state settings such as nested word automata (or equivalently visibly pushdown automata), finding the right definition of synchronizing words is however more challenging [10]. We define the synchronizing problem for RAs along the suggested framework in [15, 29]; Given an RA $\mathcal{R}$, does there exist a data word $w$ that brings each of the infinitely many states of $\mathcal{R}$ to some specific state (depending only on $w$)? Such a data word is called a synchronizing data word.

Figure 1 depicts a web interface modelled by an RA $\mathcal{R}$ with register $r$. The RA models communications between a server and two users over an interactive interface. The server execute commands $a_1, a_2$ or $b$, and users locally attach private information as data to the input. The register $r$ in each user’s interface can be used to store local information such as the password, which implies the server has only partial information about the current state of the users’ in-
When the server detects that an attacker is eavesdropping on the communication, it guides the system to a safe state. The data word \( w = (a_1, \text{password}_1)(a_2, \text{password}_2)(b, \text{restart}) \) with the distinct datum \( \text{restart} \), is synchronizing for the RA. We display the successive states after reading each input of \( w \) in Figure 1. The computation starts in the infinite set of all states in which the server and users might be; registers may have stored any datum from the data domain \( D \), ranging over infinitely many possible data values (e.g. ASCII strings or numbers). The input \( (a_1, \text{password}_1) \) updates \( r \) in interface of the user 1 which synchronizes the infinite set of states of that user in the state \((\text{User}_1, \text{password}_1)\). However, no update has taken place in interface of the user 2. In fact, the register of that interface may still store any datum from \( D \); this changes after inputting \( (a_2, \text{password}_2) \). Using the last input \( (b, \text{restart}) \), the server accomplishes synchronizing \( R \) into \((\text{Server safe}, \text{restart})\). Now, the users can renew their passwords to prevent the attacker from future eavesdropping.

**Contribution.** The problem of finding synchronizing data words for RAs imposes new challenges in the area of synchronization. It is natural to ask how many distinct data are necessary and sufficient to synchronize an RA, which we refer to by the notion of data efficiency of synchronizing data words. We show this data efficiency to be polynomial in the number of registers for deterministic RAs (DRAs), and Ackermann \((n)\) for nondeterministic RAs (NRAs), where \( n \) is the number of states. Remarkably, data efficiency is tightly related to the complexity of deciding the existence of a synchronizing data word.

For DRAs, we prove that for all automata \( R \) with \( k \) registers, if \( R \) has a synchronizing data word, then it also has one with data efficiency at most \( 2^k + 1 \). We provide a family \( (R_k)_{k \in \mathbb{N}} \) with \( k \) registers, for which indeed a polynomial data efficiency (in the size of \( k \)) is necessary to synchronize. This bound is the base of an \((\text{N})\text{PSPACE}\)-algorithm for DRAs; we prove a matching \( \text{PSPACE} \) lower bound by ideas carried over from timed settings [15]. We argue that, the synchronizing problems in DRAs with a single register (1-DRAs) and DFAs are \( \text{NLOGSPACE} \)-interreducible, implying that the problem is \( \text{NLOGSPACE} \)-complete for 1-DRAs.

For NRAs, a reduction from the non-universality problem yields the undecidability of the synchronization problem. For single-register NRAs (1-NRAs), we prove \text{Ackermann-}completeness of the problem by a novel construction proving that the synchronizing problem and the non-universality problem in 1-NRAs are polynomial-time interreducible. We believe that this technique is useful in studying synchronization in all nondeterministic settings, requiring careful analysis of the size of the construction.

Our most substantial achievement is proving \( \text{NEXPTIME} \)-completeness of the length-bounded synchronizing problem in NRAs: Does there exist a synchronizing data word with at most a given length (encoded in binary)?

For the lower bound, we present a non-trivial reduction from the bounded non-universality problem for regular-like expressions with squaring, which is known to be \( \text{NEXPTIME} \)-complete [30]. The crucial ingredient in this reduction is a family of RAs implementing binary counters. A variant of our construction yields a proof for co-\( \text{NEXPTIME} \)-completeness of the bounded universality problem in NRAs; the bounded universality problem asks whether all data words with at most a given length (encoded in binary) are in the language of the automaton.
2 Preliminaries

Deterministic finite-state automata (DFAs) are tuples \( A = \langle Q, \Sigma, \Delta \rangle \) where \( Q \) is a finite set of states, \( \Sigma \) is a finite alphabet and the transition function \( \Delta : Q \times \Sigma \to Q \) is totally defined. The function \( \Delta \) extends to finite words in a natural way: \( \Delta(q, w) = \Delta(\Delta(q, w), a) \) for all words \( w \in \Sigma^* \) and letters \( a \in \Sigma \); and it extends to all sets \( S \) by \( \Delta(S, w) = \bigcup_{q \in S} \Delta(q, w) \).

Data Words and Register Automata. Given an infinite data domain \( D \), data words are finite words over \( \Sigma \times D \). For a data word \( w = (a_1, d_1)(a_2, d_2)\ldots(a_n, d_n) \), the length of \( w \) is \( |w| = n \). We use \( \text{data}(w) = \{d_1, \ldots, d_n\} \subseteq D \) to refer to the set of data values occurring in \( w \), and we say that the data efficiency of \( w \) is \( |\text{data}(w)| \).

Let \( \text{reg} \) be a finite set of register variables. We define register constraints \( \phi \) over \( \text{reg} \) by the grammar \( \phi ::= \text{true} \mid = r \mid \phi \land \phi \mid \neg \phi \), where \( r \in \text{reg} \). We simply use \( \neq r \) for the inequality constraint \( \neg(= r) \); we denote by \( \Phi(\text{reg}) \) the set of all register constraints over \( \text{reg} \). A register valuation is a mapping \( \nu : \text{reg} \to D \) that assigns a data value to each register; by slight abuse of notation, we sometimes consider \( \nu = \{ \nu(r_1), \nu(r_2) \} \) in \( D^k \) where \( \text{reg} = \{ r_1, \cdots, r_k \} \).

The satisfaction relation of register constraints is defined on \( D^k \times D \) as follows: \( (\nu, d) \) satisfies the constraint \( = r \) if \( \nu(r) = d \); the other cases follow. For example, \( \{ (d_1, d_2), (d_2, d_2) \} \) satisfies \( = r_1 \land \neg(= r_2) \) where \( d_1 \neq d_2 \). For the set \( \up \subseteq \text{reg} \), we define the update \( \nu[\up := d] \) of valuation \( \nu \) by \( \nu[\up := d](r) = d \) if \( r \in \up \), and \( \nu[\up := d](r) = \nu(r) \) otherwise.

Register automata (RAs) over infinite data domains \( D \) are tuples \( R = \langle \mathcal{L}, \text{reg}, \Sigma, T \rangle \) where \( \mathcal{L} \) is a finite set of locations, \( \text{reg} \) is a finite set of registers, \( \Sigma \) is a finite alphabet and \( T \subseteq \mathcal{L} \times \Sigma \times \Phi(\text{reg}) \times 2^{\text{reg}} \times \mathcal{L} \) is a transition relation. We use \( \ell \xrightarrow{\phi \ a \ \uparrow} \ell' \) to show transitions \( (\ell, a, \phi, \up, \ell') \in T \). We call \( \phi \ a \ \uparrow \rightarrow \ell \) an \( a \)-transition and \( \phi \) the guard. We may omit \( \phi \) when \( \phi = \text{true} \), and omit \( \up \) when \( \up = \emptyset \). We write \( \ell \xrightarrow{\down} \) when \( \up = \{ r \} \) is singleton.

The states of \( R \) are pairs \( (\ell, \nu) \in \mathcal{L} \times D^{|\text{reg}|} \) of locations \( \ell \) and register valuations \( \nu \); since the data domains for registers are finite, RAs are infinite-state transition systems. We describe the behaviour of \( R \) as follows: Given that \( R \) is in state \( q = (\ell, \nu) \), on inputting the letter \( a \) and datum \( d \), an \( a \)-transition \( \ell \xrightarrow{\phi \ a \ \uparrow} \ell' \) may be fired if \( (\nu, d) \) satisfies the constraint \( \phi \); then \( R \) starts in successor state \( q' = (\ell', \nu') \) where \( \nu' = \nu[\up := d] \) is the update on registers. By \( \text{post}(q, (a, d)) \), we denote all successor states \( q' \) of \( q \), on inputting letter \( a \) and datum \( d \). A run of \( R \) over the data word \( w = (a_1, d_1)(a_2, d_2)\cdots(a_n, d_n) \) is a sequence of states \( q_0 q_1 \ldots q_n \), where \( q_i \in \text{post}(q_{i-1}, (a_i, d_i)) \) for all \( 1 \leq i \leq n \).

We extend \text{post} to sets of states by \( \text{post}(S, (a, d)) = \bigcup_{q \in S} \text{post}(q, (a, d)) \); and we extend \text{post} to words by \( \text{post}(S, w \cdot (a, d)) = \text{post}(\text{post}(S, w), (a, d)) \) for all words \( w \in (\Sigma \times D)^* \), letters \( a \in \Sigma \) and datum \( d \in D \).

In the rest of paper, we consider complete RAs, meaning that for all states \( q \in \mathcal{L} \times D^{\text{reg}} \) and all inputs \( (a, d) \in \Sigma \times D \), there is at least one successor: \( |\text{post}(q, (a, d))| \geq 1 \). We also classify the RAs into deterministic (DRAs) and nondeterministic (NRAs), where an RA is deterministic if \( |\text{post}(q, (a, d))| \leq 1 \) for all states \( q \) and all inputs \( (a, d) \).

Synchronizing words and synchronizing data words. The synchronizing words are a well-studied concept for DFAs; see [32]. Informally, a synchronizing word leads the automaton from every state to the same state: the word \( w \in \Sigma^* \) is synchronizing for \( A = \langle Q, \Sigma, \Delta \rangle \) if there exists some state \( \bar{q} \in Q \) such that \( \Delta(Q, w) = \{ \bar{q} \} \). The synchronizing problem in DFAs asks, given a DFA \( A \), whether there exists some synchronizing word for \( A \).
We introduce synchronizing data words for RAs: for an RA $\mathcal{R} = \langle \mathcal{L}, \text{reg}, \Sigma, T \rangle$ over a data domain $D$, a data word $w \in (\Sigma \times D)^+$ is synchronizing if there exists some state $(i, \nu)$ such that $\text{post}(\mathcal{L} \times D^{\text{reg}}, w) = \{(i, \nu)\}$. The synchronizing problem asks, given an RA $\mathcal{R}$ over a data domain $D$, whether $\mathcal{R}$ has some synchronizing data word. The bounded synchronizing problem decides, given an RA $\mathcal{R}$ and length $\in \mathbb{N}$ encoded in binary, whether $\mathcal{R}$ has such synchronizing data word $w$ with $|w| \leq \text{length}$.

### 3 Synchronizing data words for DRAs

In this section, we first show that the synchronizing problems in 1-DRAs and DFAs are $\text{NLOGSPACE}$-interreducible, implying that the problem is $\text{NLOGSPACE}$-complete for 1-DRAs. Next, we prove that the problem for $k$-DRAs, in general, can be decided in $\text{PSPACE}$; a reduction similar to the timed settings, as in [15], provides the matching lower bound. To obtain the complexity upper bounds, we prove that inputting words with data efficiency $2|\text{reg}| + 1$ is sufficient to synchronize a DRA.

The concept of synchronization requires that all runs of RAs, whatever the initial state (initial location and register valuations), end in the same state (or $\nu(r) \notin \text{data}(w_{\text{synch}})$ for some $r \in \text{reg}$, the register $r$ must be updated. Observe that such updates must happen at inequality-guarded transitions, which themselves must be accessible by inequality-guarded transitions, (possibly with no update).

For the RA $\mathcal{R}$ in Figure 2, assume that $d_1, d_2 \notin \text{data}(w_{\text{synch}})$. The two runs of $\mathcal{R}$ starting from (init, $\begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$) and (init, $\begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$) first take the transition $\begin{pmatrix} \#_1 \\ a \end{pmatrix} \xrightarrow{\#_1} \ell_1' \text{ updating register } r_1$. Next, the two runs must take $\ell_1' \xrightarrow{\text{else } a} \ell_1' \text{ to update } r_2$ and $\ell_2' \xrightarrow{\text{else } a} \ell_3'$ to update $r_3$; otherwise these two runs would never synchronize in a single state.

**Lemma 1.** For all DRAs for which there exist synchronizing data words, there exists some data word $w$ with data efficiency $|\text{reg}|$ such that $\text{post}(\mathcal{L} \times D^{\text{reg}}, w) \subseteq \mathcal{L} \times (\text{data}(w))^{\text{reg}}$.

After reading some word that shrinks the infinite set of states in RAs to a finite set $S$, one can apply the pairwise synchronization technique to synchronize states in $S$. This technique is the core to decide the synchronizing problem in DFAs in $\text{NLOGSPACE}$: Given a DFA $A = (Q, \Sigma, \Delta)$, it is known that it has a synchronizing word if and only if for all pairs of states $q, q' \in Q$, there exists a word $v$ such that $\Delta(q, v) = \Delta(q', v)$ (see [32] for more details). The pairwise synchronization sets $S_{i+1} = Q$, and for all $i = |Q| - 1, \cdots, 1$ repeats the following: find a word $v_i$ such that $\Delta(q, v_i) = \Delta(q', v_i)$ for some pair $q, q' \in S_{i+1}$ and let

\[ MFCS 2016 \]
Theorem 5. The synchronizing problem for k-DRAs is PSPACE-complete.

4 Synchronizing data words for NRAs

In this section, we study the synchronizing problems for NRAs. We slightly update a result in [15] to present a general reduction from the non-universality problem to the synchronizing problem in NRAs. This reduction proves the undecidability result for the synchronizing problem in k-NRAs, and Ackermann-hardness in 1-NRAs. We then prove that in 1-NRA, the
synchronizing and non-universality problems are indeed interreducible, which completes the picture by Ackermann-completeness of the synchronizing problem in 1-NRAs.

In nondeterministic settings, we present two kinds of counting features while synchronizing. A family of 1-NRAs with $O(n)$ locations where Ackermann($n$) distinct data must be read and another family where an input datum $x \in D$ must be read $2^n$ times to achieve synchronization.

The second family can be captured by NFAs if the shortest length to synchronize is of interest.

Lemma 6. There is a family of 1-NRAs $(R_{\text{counter}}(n))_{n \in \mathbb{N}}$ with $O(n)$ locations, such that for all synchronizing data words $w$, some datum $d \in \text{data}(w)$ appears in $w$ at least $2^n$ times.

We next remark that the data efficiency while synchronizing 1-NRAs can be a function in the fast growing hierarchy [28]. Recall that $\text{tower} : \mathbb{N} \rightarrow \mathbb{N}$ is defined inductively by $\text{tower}(0) = 1$ and $\text{tower}(n + 1) = 2^{\text{tower}(n)}$.

Figure 4 shows the 1-NRA $R_{\text{tower}}$ over the data domain $\mathbb{N}$. We indicate that $|\text{data}(w)| \in O(\text{tower}(3))$ for all synchronizing data words $w$. As in $R_{\text{counter}}$, $\text{synch}$ is the location where the RA must be synchronized in, and an initial $\text{reset}$ is enforced to reach the location $\text{Data}_1$.

The main issue is that while synchronizing $R_{\text{tower}}$, some inequality-guarded transitions are unavoidable, which are the ones that may replicate the tokens. For example, if one token in $\text{Data}_1$, firing two transitions $\text{Data}_1 \xrightarrow{\text{rep}} \text{Data}_{1,2}$ and $\text{Data}_1 \xrightarrow{\text{rep}} \text{Data}_{1,2}$ replicates it to two tokens in $\text{Data}_{1,2}$.

Since the question is the required data efficiency of synchronizing words, we always start from datum 1 and feed $R_{\text{tower}}$ with the smallest number $i$ which contributes to synchronization.
Moreover, when resetting we read datum 1. To synchronize \( \mathcal{R}_{\text{tower}} \) with the least data efficiency, we go through the following steps:

- **resetting to \( \text{Data}_1 \):** the \( \ast \)-transitions reset and place one token in \( \text{Data}_1 \) by \( \ell \xrightarrow{\ast \ast} \text{Data}_1 \) for all \( \ell \in \mathcal{L} \setminus \{ \text{synch} \} \). Reading \( \ast \) is necessary for synchronizing since tokens in reset only move out by a \( \ast \)-transition. Since another \( \ast \) eliminates all tokens and places one token in \( \text{Data}_1 \) again, resetting is inefficient; we call all transitions directed to reset inefficient.

- **replicating towering tokens:** after a reset with \( \ast(1) \) and having a 1-token in \( \text{Data}_1 \), the only efficient transitions are on \( \text{rep}(2)\text{rep}(3) \), which results in replicating the 1-token in 3 tokens (shown as \( \{1,2,3\} \)-tokens) and placing them in \( \text{waitTow} \).

- **towering the waiting \( i \)-token:** intuitively, the \( i \)-token in \( \text{waitTow} \) is waiting to trigger the tower\((i)\)-process, right after the process of tower\((i-1)\) is accomplished. After the tower\((i)\)-process, we see that \( \{1,2,\cdots,\text{tower}(i)\} \)-tokens are in store. Next, if no more token is waiting in \( \text{waitTow} \), the \#-transition synchronizes the RA into synch; otherwise, the inefficient \#-transition in \( \text{waitTow} \) resets. Below, we argue how, given a 3-token waiting in \( \text{waitTow} \) and \( \{1,2,\cdots,\text{tower}(2)\} \)-tokens in store, the tower\((3)\)-process proceeds. The first efficient transition is on \( \text{tow}(3) \), which moves all those tokens to \( \text{waitDou} \). Recall that tower\((3) = \text{tower}(2)^2 \), simply doubling 1 for tower\((2) = 4 \) times. Each \( i \)-token waiting in \( \text{waitDou} \) (each in \( \{1,2,3,4\} \)-tokens) is aimed to trigger a doubling.

- **1-token:** the only efficient transitions are on \( \text{doub}(1)(a,1)\text{rep}(2) \), which result in replicating \( \{1,2\} \)-tokens in store.

- **2-token:** inputting \( \text{doub}(2) \), which fires the only efficient transition, moves all the tokens obtained in the previous doubling process into \( \text{waitRep} \). Then, both \( \{1,2\} \)-tokens in \( \text{waitRep} \) will be replicated individually: note that while replicating, if a locally fresh datum from all data in \( \text{waitRep}, \text{Rep} \) and store is not read, an inefficient transition will be fired. After the second doubling by \( \text{doub}(2)(a,1)\text{rep}(3)(a,2)(\text{rep},4) \), the \( \{1,2,3,4\} \)-tokens are produced in store.

- **3-token:** inputting \( \text{doub}(3) \) moves \( \{1,2,3,4\} \)-tokens into \( \text{waitRep} \), which are indeed the tokens obtained in previous doubling process. For all \( 1 \leq i \leq 4 \), the \( i \)-token is replicated into \( \{i,4+i\} \)-tokens by \( \text{doub}(3)(a,i)(\text{rep},4+i) \). This results in storing \( \{1,\cdots,8\} \)-tokens in store.

- **4-token:** it doubles the number of tokens in store for the 4-th time: \( \{1,\cdots,16\} \)-tokens. So, tower\((3) = 2^{\text{tower}(2)^2} = 16 \) distinct data are needed to synchronize \( \mathcal{R}_{\text{tower}} \).
Lemma 7. There is a family of 1-NRAs \( R_{\text{tower}}(n) \) for all synchronizing data words \( w \).

We recall, from [28], that tower is at level 3 of the Ackermann-hierarchy. Using similar ideas as in Lemma 7, we can define a family of 1-NRAs \( R^m_n \) such that all synchronizing data words have data efficiency at least \( \text{ack}_n(m) \), where \( \text{ack}_n \) is at level \( n \) of the Ackermann-hierarchy.

To define the language of a given RA \( R \), we equip it with an initial location \( \ell_i \) and a set \( \mathcal{L}_f \) of accepting locations, where, without loss of generality, we assume that all outgoing transitions from \( \ell_i \) update all registers. The language \( L(R) \) is the set of all data words \( w \in (\Sigma \times D)^+ \), for which there is a run from \( (\ell_i, \nu_i) \) to \( (\ell_f, \nu_f) \) such that \( \ell_f \in \mathcal{L}_f \) and \( \nu_i, \nu_f \in \text{D}^{\log \log} \). The universality problem asks, given an RA, whether \( L(R) = (\Sigma \times D)^+ \). We adopt an established reduction in [15] to provide the following Lemma.

Lemma 8. The non-universality problem is reducible to the synchronizing problem for NRAs.

As an immediate result of Lemma 8 and the undecidability of the non-universality problem for \( k \)-NRAs (Theorems 2.7 and 5.4 in [12]), we obtain the following theorem.

Theorem 9. The synchronizing problem for \( k \)-NRAs is undecidable.

We present a reduction showing that, for 1-NRAs, the synchronizing problem is reducible to the non-universality problem, providing the tight complexity bounds for the synchronizing problem. We observe that Lemma 1 holds for 1-NRAs, meaning that for all 1-NRAs with some synchronizing data word, there exists some data word \( w \) with data efficiency 1 (for example, \( \text{data}(w) = \{x\} \) such that \( \text{post}(\mathcal{L} \times D, w) \subseteq \mathcal{L} \times \text{data}(w) \). Considering this fact as the skeleton, we define a language \( \text{lang} \) such that data words in this language are encodings of the synchronizing process. Let \( \mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_n\} \) be the set of locations and \( x, y \) two distinct data. Each data word in \( \text{lang} \), if there exists any, consists of

- an initial block: a delimiter \( (*, y) \) with distinct datum, the sequence \( (\ell_1, x), (\ell_2, x), \ldots, (\ell_n, x) \) and an input \( (a, d) \in \Sigma \times D \) as the first input of a synchronizing word. The initial block is followed by

- a sequence of normal blocks: the delimiter \( (*, y) \), successors reached from states and input in the previous block, and the next input of the synchronizing word. Finally, the data word ends with

- a final block: the delimiter \( (*, y) \), a single successor reached from states and input in the previous block and the delimiter \( (*, y) \).

We consider some further membership conditions for \( \text{lang} \), which guarantee the correct semantics of the encoding of runs of \( R \). For instance, we impose the condition that for all \( (\ell, d) \) and \( (a, d') \) with \( d \neq d' \) in one block, if there exists a transition \( \ell \xrightarrow{d, a, d} \ell' \), then \( (\ell', d') \) must be in the next block.

We then construct a 1-NRA \( R_{\text{comp}} \) that accepts the complement of \( \text{lang} \); thus \( R \) has some synchronizing data word if, and only if, the language of \( R_{\text{comp}} \) is not universal. The 1-NRA \( R_{\text{comp}} \) is a finite union of smaller 1-NRAs, each of them violating one of the membership conditions for \( \text{lang} \). For instance, the membership condition stated above is violated by the following 1-NRA.
15:10  Synchronizing Data Words for Register Automata

Figure 5  An RA where all synchronizing data words with length at most 3 require data efficiency 3 to shrink the infinite set of states to a finite subset.

Lemma 10. The synchronizing problem is reducible to the non-universality problem for 1-NRAs.

By Lemmas 8 and 10 and Ackermann-completeness of the non-universality problem for 1-NRA, which follows from Theorem 2.7 and the proof of Theorem 5.2 in [12], and the result for counter automata with incrementing errors in [19], we obtain the following theorem.

Theorem 11. The synchronizing problem for 1-NRAs is Ackermann-complete.

5 Bounded synchronizing data words for NRAs

The synchronizing problem for NRAs is undecidable in general, due to the unbounded length of synchronizing data words; In the following, we study, for NRAs, the bounded synchronizing problem, that requires the synchronizing data words to have at most a given length.

To decide the synchronizing problem in 1-RAs, in both deterministic and nondeterministic settings, we hugely rely on Lemma 1. We thus assume that the RA inputs the same datum x (chosen arbitrary) as many times as necessary to have the successor set included in L × {x}; next, we synchronize this successor set in a singleton. The RA R shown in Figure 5 shows that this approach is not useful when the length of synchronizing words are asked to not exceed a given bound. Observe that the data word (a, x)(b, y)(b, z) is synchronizing with length 3 (not exceeding the bound 3). All synchronizing data words that repeat a datum such as x, to first bring the RA to a finite set, have length at least 5.

We first present a NEXPTIME-hardness result based on the binary counting feature in NRAs. The proof is by a reduction from the bounded non-universality problem for regular-like expressions. A regular-like expression over an alphabet Σ is a well-parenthesized expression built by constants a ∈ Σ, two binary operations · (concatenation) and + (union), and a unary operation 2 (squaring). The language $L(expr)$ of such expressions expr is defined inductively as in regular expressions, where $L(expr^2) = L(expr) \cdot L(expr)$. The bounded universality problem asks, given a regular-like expression expr and length $N \in \mathbb{N}$ written in binary, whether $L(expr)$ includes all strings with length at most $N$; in other words, if $\Sigma^* \subseteq L(expr)$.

Remark. The bounded universality problem of regular-like expressions is co-NEXPTIME-complete, where the membership in co-NEXPTIME comes by guessing a witness string $u$ with length at most $N$, and checking in EXPTIME that $u \not\in L(expr)$. We observe that the reduction presented in [30], for the inequivalence between two regular-like expressions, establishes the co-NEXPTIME-hardness for the bounded universality problem, even if $|\Sigma| = 2$. 
Given a regular-like expression $expr$ and length $N \in \mathbb{N}$, we construct a 1-NRA $R$ and length $\in \mathbb{N}$, such that the language of $expr$ is bounded universal if and only if $R$ has no synchronizing data word with length at most length. The RA $R$ consists of two distinguished locations reset, synch and three main gadgets: Gambling, Freshness and Checking gadget.

The RA $R$ relies on the instincts of a gambler to synchronize. When feeding $R$ with a data word $w$, we say that there is an $x$-token in location $\ell$ if $(\ell, x) \in \text{post}(L \times D, w)$. Intuitively, whenever a token is in location reset, the gambler must restart; and $R$ can only synchronize in synch. The reduction, roughly speaking, is such that the gambler guesses a string $u \in (a + b)^*$, letter-by-letter, and at some point places a bet that $u$ is the witness for bounded non-universality. Gambling gadget discretely checks whether the bet makes sense: $|u| \leq N$. If yes, all tokens in Gambling gadget move to synch; otherwise, all tokens move to reset to give another chance to the gambler. On the other hand, meanwhile the gambler is hesitating to place the bet, Checking gadget tries to counter-attack the gambler by proving that $expr$ generates $u$. To this aim, Checking gadget always follows all possible sub-expressions of $expr$ which may produce $u$. This happens by replicating tokens and letting run computations for each sub-expression in parallel. As soon as one sub-expression fails in producing $u$, its token moves to lost$_{expr}$ (of Checking gadget); and conversely, if a sub-expression definitely generates $u$, then its token moves to win$_{expr}$ (of Checking gadget). The sub-expressions that have a string with prefix $u$ keep their tokens in Checking gadget to follow the next computations (hoping that the gambler will not bet on $u$ and continue guessing more letters). When a bet happens, all tokens in Checking gadget, except tokens in win$_{expr}$, move to synch. In this way, $R$ synchronizes in synch if $|u| \leq N$ and $u \not\in L(expr)$.

Figure 6 depicts the constructed $R$ for $expr = (a + ab)^2a + a$ and $N = 3$. Below, we give more intuitive explanations:
**Gambler resets the guess:** an initial reset is enforced while synchronizing since tokens in reset only move out by a 𝑥-transition. When a reset happens, the gambler has the chance to change the guessed string 𝑢 and to restart. Resetting eliminates all tokens in ℛ and places tokens only to synch and the initial locations of all gadgets: zero, allTokens and 1expr.

**Gambler must only bet on |𝑢| ≤ 𝑁:** after a reset, the sequence of read 𝑎, 𝑏 is the guessed 𝑢 by the gambler. Gambling gadget counts all 𝑎, 𝑏 inputs to check whether |𝑢| ≤ 𝑁. This gadget is a chain of (modified) counting RAs ℛcounter(𝑖) described in Lemma 6, where ℛcounter(𝑖) counts until 2𝑖. We modify ℛcounter(𝑖) such that the increment process, triggered by Bit-transitions is executed after each occurrence of 𝑎 or 𝑏. Gambling gadget in Figure 6 must count up to 𝑁 + 1 = 22 that is achieved by calling ℛcounter(2).

**Freshness gadget:** after a reset, Checking gadget starts with a single token in 1expr, say an 𝑥-token. This token moves along the gadget by reading 𝑢 letter-by-letter and checking if the input prefix of 𝑢 is in expr. For all unions, such as 𝑎 + 𝑏, the token replicates: 𝑥-token checks if 𝑎, and fresh 𝑦-token checks if 𝑎𝑏 contribute in generating 𝑢. Such tokens must move around individually, and thus must be distinctive. Freshness gadget guarantees the global freshness of such tokens: When replicating tokens by fresh-transitions, if the read datum is not fresh, the inconsistent transition allTokens ↝ fresh reset happens.

**Checking gadget:** The checking is the gadget for expr that is built inductively from gadgets 𝑎, 𝑏, 𝑎 + 𝑏, (𝑎 + 𝑏)2 and (𝑎 + 𝑏)2 𝑎. After a reset, it starts with a single token in 1expr, if 𝑢 ∈ ℛ(expr), then some token moves to winexpr spoiling the gambler’s plan in synchronizing. We explain the core of the sub-gadgets by following the scenario for ℛ of expr = (𝑎 + 𝑏)2 𝑎 + 𝑎:

- **When gambler bets on a wrong witness** 𝑢 ∈ ℛ(expr), such as aaa. After a reset, assuming that an 𝑥-token is in 1expr, it replicates by (copy, 𝑥) (fresh, 𝑦) with 𝑥 ≠ 𝑦 to {𝑥, 𝑦}-tokens. The 𝑥-token moves to 13 entering the 𝑎-gadget, and 𝑦-token to 3 entering the (𝑎 + 𝑏)2 𝑎-gadget. The only consistent transition is enter, the initial transition in the squaring. It makes a copy of the entering token in FirstRound to enforce the token to go through the gadget under squaring, two times. After (enter, 𝑦), there are 𝑦-tokens in FirstRound and in 5 as the initial location of the (𝑎 + 𝑏)-gadget. For the union 𝑎 + 𝑏, inputting (copy, 𝑦) (fresh, 𝑧) replicates the 𝑦-token in 5 to {𝑦, 𝑧}-tokens where Freshness gadget guarantees that 𝑧 is globally fresh. The 𝑦-token in 8 starts the 𝑎-gadget and 𝑦-token in 9 the ab-gadget. It is crucial that when union replicates tokens under squaring, their copy in FirstRound (and in SecondRound) must be replicated too: so (copy, 𝑦) (fresh, 𝑧) replicates the 𝑦-token in FirstRound to {𝑦, 𝑧}-tokens. Next 𝑎-transitions are consistent; observe that three tokens {𝑥, 𝑦, 𝑧} check if 𝑎 is generated: as in (𝑎 + 𝑏)2 𝑎 + 𝑎, the first produced 𝑎 may be the result of three expressions: lonely 𝑎 or 𝑎, 𝑏 under squaring.

The 𝑥-token from 13 moves to winexpr meaning that 𝑎 ∈ ℛ(expr); however, the gambler is betting on aaa, and the second a wastes this (fake) win by moving the token to lostexpr.

The 𝑦-token now must start the second round of squaring: inputting (run, 𝑦) brings back the 𝑦-token to 5, the initial of squaring, and also free the 𝑦-token in FirstRound to SecondRound (as a flag that 𝑦-token is ready to leave the squaring gadget). Due to the union again, the 𝑦-token, individually from 𝑧-token, must be replicated. By (copy, 𝑦) (fresh, 𝑑) (𝑎, 𝑦) (leave, 𝑦) (𝑎, 𝑦), the 𝑦-token arrives in winexpr. The gambler places the bet with no more 𝑎, 𝑏, meaning that the 𝑦-token in winexpr has no way to get synchronized, as it moves to reset by the bet-transition.

- **When gambler bets on a right witness** 𝑢 ∉ ℛ(expr), such as bb. Observe that (∗, 𝑥) (copy, 𝑥) (fresh, 𝑦) (enter, 𝑥) (copy, 𝑦) (fresh, 𝑧) (𝑏, 𝑥) (𝑏, 𝑥) (bet, 𝑥) synchronizes ℛ into synch.
When gambler cheats by betting on strings longer than $N$, such as $abbb$. The issue is when $abbb \notin L(\text{expr})$, in these cases data words such as $(*, x)(\text{copy}, x)(\text{fresh}, y)(\text{enter}, x)(\text{copy}, y)(\text{fresh}, z)(a, x)(\text{run}, y)(\text{copy}, y)(\text{fresh}, d)(b, x)(\text{run}, z)(\text{copy}, z)(\text{fresh}, m)(b, x)(b, x)$ would place all tokens of Checking gadget in $\text{lost}_{\text{expr}}$. Now, bet-transitions would move all tokens from Checking gadget to synch. However, Gambling gadget has counted 4, and thus location $2^2$ has a token which goes to reset by placing the bet. This spoils synchronizing $\mathcal{R}$ when the gambler cheats by exceeding the bound $N = 3$. Note that tokens in zero, by bet-transitions, move to reset to forbid that the gambler cheats by the empty word too.

Note that length = 14 of the synchronizing data word is computed inductively: here, $+1$ for resetting $\mathcal{R}$, $+2$ for the first union, $+(2 \cdot (2) + 3)$ for the squaring and union under it, $+1$ for the bet and $+N$ for Gambling gadget.

\begin{lemma}
The bounded synchronization problem for NRAs is NEXPTIME-hard.
\end{lemma}

Guessing a data word $w$ with $|w| \leq \text{length}$ and checking in EXPTIME whether $w$ is synchronizing yields NEXPTIME-membership. Altogether we obtain the following result:

\begin{theorem}
The bounded universality problem for NRAs is NEXPTIME-complete.
\end{theorem}

The bounded universality problem asks, given an RA and $\text{length} \in \mathbb{N}$ encoded in binary, whether all data words $w$ with $|w| \leq \text{length}$ are in the language of the automaton. We state that the bounded universality problem in NRAs is co-NEXPTIME-complete. The membership in co-NEXPTIME follows by guessing a witness $w$ letter-by-letter; and checking if the successor states after reading $w$ are all non-accepting. A variant of the presented reduction allows to prove that the bounded universality problem in NRAs is co-NEXPTIME-hard: equip $\mathcal{R}$ with the initial location reset and set $L_f$ of accepting locations including all locations but synch.

\begin{theorem}
The bounded universality problem for NRAs is co-NEXPTIME-complete.
\end{theorem}

Acknowledgements. We thank Sylvain Schmitz for helpful discussions on well-structured systems and nonelementary complexity classes.

References

Synchronizing Data Words for Register Automata


