Synchronizing Data Words for Register Automata*

Parvaneh Babari[1], Karin Quaas[2], and Mahsa Shirmohammadi[3]

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 PSPACE-complete, and is NLOGSPACE-complete for 1-DRAs. For nondeterministic RAs (NRAs), we show that 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 Ackermann-completeness of the problem for 1-NRAs. Our most substantial achievement is proving 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 co-NEXPTIME-complete.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Data Words, Register Automata, Synchronizing Problem, 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.

* This work was partially supported by Deutsche Forschungsgemeinschaft (DFG), GRK 1763 (QuantLA) and project QU 316/1-2.

41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016).
Editors: Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier; Article No. 15; pp. 15:1–15:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

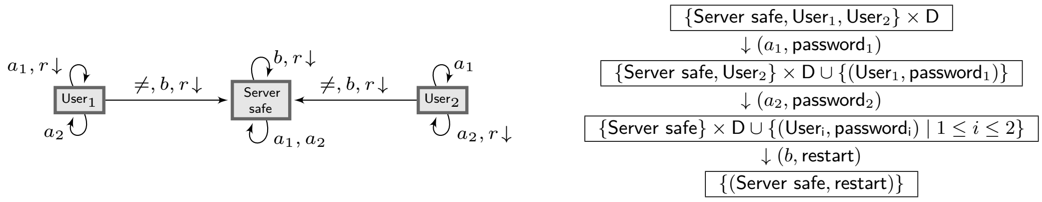


Figure 1 An RA $\mathcal{R}$ with single register r that models the interactive interfaces between a server and two users on the web. An *update*, denoted by $r\downarrow$, stores the input datum into r. Transitions labelled with $\neq$ are only taken if the datum of the current position of the input word and datum in register r are different. The data word $w = (a_1, \mathsf{password}_1)(a_2, \mathsf{password}_2)(b, \mathsf{restart})$ with the distinct datum restart is synchronizing; the set of successors after reading each input of w is shown on the right, where D is the infinite data domain. Observe that $\mathcal{R}$ is synchronized in (Server safe, restart).

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' interfaces. When the server detects that an attacker is eavesdropping on the communication, it guides the system to a *safe* state. The data word $w = (a_1, \mathsf{password}_1)(a_2, \mathsf{password}_2)(b, \mathsf{restart})$ with the distinct datum $\mathsf{restart}$, 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 $(a_1, \mathsf{password}_1)$ updates r in interface of the user 1 which synchronizes the infinite set of states of that user in the state $(\mathsf{User}_1, \mathsf{password}_1)$. 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 $(a_2, \mathsf{password}_2)$. Using the last input $(b, \mathsf{restart})$, the server accomplishes synchronizing $\mathcal{R}$ into $(\mathsf{Server\ safe}, \mathsf{restart})$. 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 $\mathsf{Ackermann}(n)$ 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 $\mathcal{R}$ with k registers, if $\mathcal{R}$ has a synchronizing data word, then it also has one with data efficiency *at most* $2k+1$. We provide a family $(\mathcal{R}_k)_{k\in\mathbb{N}}$ 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 (N)PSPACE-algorithm for DRAs; we prove a matching 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 NLOGSPACE-interreducible, implying that the problem is 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 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 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 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-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 $\mathcal{A} = \langle Q, \Sigma, \Delta \rangle$ where Q is a finite set of states, Σ is a finite alphabet and the transition function $\Delta : Q \times \Sigma \to Q$ is totally defined. The function Δ extends to finite words in a natural way: $\Delta(q, wa) = \Delta(\Delta(q, w), a)$ 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 $\Sigma \times \mathsf{D}$. 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 $\mathsf{data}(w) = \{d_1, \ldots, d_n\} \subseteq \mathsf{D}$ to refer to the set of data values occurring in w, and we say that the *data efficiency of* w is $|\mathsf{data}(w)|$.

Let reg be a finite set of *register variables*. We define *register constraints* ϕ over reg by the grammar $\phi ::= \mathsf{true} \mid {=}r \mid \phi \wedge \phi \mid \neg\phi$, where $r \in \mathsf{reg}$. We simply use $\neq r$ for the inequality constraint $\neg({=}r)$; we denote by $\Phi(\mathsf{reg})$ the set of all register constraints over reg. A *register valuation* is a mapping $\nu : \mathsf{reg} \to \mathsf{D}$ that assigns a data value to each register; by a slight abuse of notation, we sometimes consider $\nu = \begin{pmatrix} \nu(r_1) \\ \cdots \\ \nu(r_k) \end{pmatrix} \in \mathsf{D}^k$ where $\mathsf{reg} = \{r_1, \cdots, r_k\}$. The satisfaction relation of register constraints is defined on $\mathsf{D}^k \times \mathsf{D}$ as follows: (ν, d) satisfies the constraint $= r$ if $\nu(r) = d$; the other cases follow. For example, $(\begin{pmatrix} d_1 \\ d_2 \\ d_1 \end{pmatrix}, d_2)$ satisfies $((= r_1) \wedge (= r_2)) \vee (\neq r_3))$ where $d_1 \neq d_2$. For the set $\mathsf{up} \subseteq \mathsf{reg}$, we define the *update* $\nu[\mathsf{up} := d]$ of valuation ν by $\nu[\mathsf{up} := d](r) = d$ if $r \in \mathsf{up}$, and $\nu[\mathsf{up} := d](r) = \nu(r)$ otherwise.

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

The *states* of $\mathcal{R}$ are pairs $(\ell, \nu) \in \mathcal{L} \times \mathsf{D}^{|\mathsf{reg}|}$ of locations ℓ and register valuations ν; since the data domains for registers are infinite, RAs are infinite-state transitions systems. We describe the behaviour of $\mathcal{R}$ as follows: Given that $\mathcal{R}$ is in state $q = (\ell, \nu)$, on inputting the letter a and datum d, an a-transition $\ell \xrightarrow{\phi \ a \ \mathsf{up}\downarrow} \ell'$ may be fired if (ν, d) satisfies the constraint ϕ; then $\mathcal{R}$ starts in *successor* state $q' = (\ell', \nu')$ where $\nu' = \nu[\mathsf{up} := d]$ is the update on registers. By $\mathsf{post}(q, (a, d))$, we denote all successor states q' of q, on inputting letter a and datum d. A run of $\mathcal{R}$ 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 $q_i \in \mathsf{post}(q_{i-1}, (a_i, d_i))$ for all $1 \leq i \leq n$.

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

In the rest of paper, we consider *complete* RAs, meaning that for all states $q \in \mathcal{L} \times \mathsf{D}^{|\mathsf{reg}|}$ and all inputs $(a, d) \in \Sigma \times \mathsf{D}$, there is at least one successor: $|\mathsf{post}(q, (a, d))| \geq 1$. We also classify the RAs into *deterministic* (DRAs) and *nondeterministic* (NRAs), where an RA is deterministic if $|\mathsf{post}(q, (a, d))| \leq 1$ 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 $\mathcal{A} = \langle Q, \Sigma, \Delta \rangle$ 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 $\mathcal{A}$, whether there exists some synchronizing word for $\mathcal{A}$.

We introduce synchronizing data words for RAs: for an RA $\mathcal{R} = \langle \mathcal{L}, \mathsf{reg}, \Sigma, T\rangle$ over a data domain D, a data word $w \in (\Sigma \times \mathsf{D})^+$ is *synchronizing* if there exists some state $(\bar{\ell}, \bar{\nu})$ such that $\mathsf{post}(\mathcal{L} \times \mathsf{D}^{|\mathsf{reg}|}, w) = \{(\bar{\ell}, \bar{\nu})\}$. 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 $\mathsf{length} \in \mathbb{N}$ encoded in binary, whether $\mathcal{R}$ has such synchronizing data word w with $|w| \leq \mathsf{length}$.

3 Synchronizing data words for DRAs

In this section, we first show that the synchronizing problems in 1-DRAs and DFAs are NLOGSPACE-interreducible, implying that the problem is NLOGSPACE-complete for 1-DRAs. Next, we prove that the problem for k-DRAs, in general, can be decided in 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 $2|\mathsf{reg}| + 1$ 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 $(\ell_{\mathsf{synch}}, \nu_{\mathsf{synch}})$ that only depends on the data word w_{synch}: $\mathsf{post}(\mathcal{L} \times \mathsf{D}, w_{\mathsf{synch}}) = \{(\ell_{\mathsf{synch}}, \nu_{\mathsf{synch}})\}$. While processing a synchronizing data word, the infinite set of states in RAs must necessarily shrink to a finite set of states. The RA $\mathcal{R}$ with 3 registers depicted in Figure 2 illustrates this phenomenon. Considering the set $\{x_1, x_2, x_3\} \subseteq \mathsf{D}$ of distinct data values; starting from states in $\{\mathsf{init}\} \times \mathsf{D}^3$, the infinite set of runs of $\mathcal{R}$ over the data word $(a, x_1)(a, x_2)(a, x_3)$ is merged into the finite set $\{(\ell_3, \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}), (\ell_3', \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix})\}$. We use this observation to provide a linear bound on the sufficient number of required distinct data while synchronizing RAs.

In Lemma 1, we prove that data words over only $|\mathsf{reg}|$ distinct data values are sufficient to shrink states of RAs to a finite set. We establish this result based on the following two key facts: (1) to shrink the set $\mathcal{L} \times \mathsf{D}^{|\mathsf{reg}|}$, one can find a word w_ℓ that brings the RA from $\{\ell\} \times \mathsf{D}^{|\mathsf{reg}|}$ to some finite set for every $\ell \in \mathcal{L}$. Thanks to being deterministic, appending some prefix or suffix to w_ℓ would achieve the same objective; so the successor set of $\mathcal{L} \times \mathsf{D}^{|\mathsf{reg}|}$ and $(w_\ell)_{\ell \in \mathcal{L}}$ is a finite set. Moreover (2), when processing a synchronizing data word w_{synch} from a state (ℓ, ν) with $\nu(r) \notin \mathsf{data}(w_{\mathsf{synch}})$ for some $r \in \mathsf{reg}$, 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 $d_1, d_2 \notin \mathsf{data}(w_{\mathsf{synch}})$. The two runs of $\mathcal{R}$ starting from $(\mathsf{init}, \begin{pmatrix} d_1 \\ d_1 \\ d_1 \end{pmatrix})$ and $(\mathsf{init}, \begin{pmatrix} d_2 \\ d_2 \\ d_2 \end{pmatrix})$ first take the transition $\mathsf{init} \xrightarrow{\neq r_1 \;\; a \;\; r_1\downarrow} \ell_1'$ updating register r_1. Next, the two runs must take $\ell_1' \xrightarrow{\mathsf{else} \;\; a \;\; r_2\downarrow} \ell_2'$ to update r_2 and $\ell_2' \xrightarrow{\mathsf{else} \;\; a \;\; r_3\downarrow} \ell_3'$ 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 $|\mathsf{reg}|$ such that $\mathsf{post}(\mathcal{L} \times \mathsf{D}^{|\mathsf{reg}|}, w) \subseteq \mathcal{L} \times (\mathsf{data}(w))^{|\mathsf{reg}|}$.*

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 NLOGSPACE: Given a DFA $\mathcal{A} = \langle Q, \Sigma, \Delta\rangle$, 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 $S_{|Q|} = Q$, 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

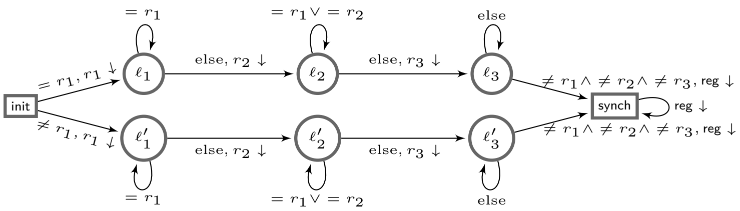


Figure 2 A DRA with three registers r_1, r_2, r_3 and single letter a (omitted from transitions) that can be synchronized in the state (synch, x_4) by the data word $w_{\mathsf{synch}} = (a, x_1)(a, x_2)(a, x_3)(a, x_4)$ if $\{x_1, x_2, x_3, x_4\} \subseteq \mathsf{D}$ is a set of 4 distinct data.

$S_i = \Delta(S_{i+1}, v_i)$. The word $w = v_{n-1} \cdots v_2 \cdot v_1$ is synchronizing for the DFA. We generalize the pairwise synchronization technique for DRAs to establish the following Lemma.

▶ **Lemma 2.** *For all DRAs for which there exist synchronizing data words, there exists a synchronizing data word with data efficiency* $2|\mathsf{reg}| + 1$.

Given a 1-DRA $\mathcal{R}$, the synchronizing problem can be solved by (1) ensuring that from each location ℓ an update on the single register is achieved by going through inequality-guarded transitions, which can be done in NLOGSPACE. Lemma 1 suggests that feeding $\mathcal{R}$ consecutively with a single datum $x \in \mathsf{D}$ is sufficient for this phase and the set of successors of $\mathcal{L} \times D$ would be a subset of $\mathcal{L} \times \{x\}$. Next (2) picking an arbitrary set $\{x, y, z\}$ of data including x, by Lemma 2 and the pairwise synchronization technique, the problem reduces to the synchronizing problem in DFAs where data in registers and input data extend locations and the alphabet: $Q = \mathcal{L} \times \{x, y, z\}$ and $\Sigma \times \{x, y, z\}$. Since a 1-DRA, where all transitions update the register and are guarded with true, models a DFA, we obtain the following result.

▶ **Theorem 3.** *The synchronization problem for* 1*-DRAs is in* NLOGSPACE*-complete.*

We provide a family of DRAs, for which a linear bound on the data efficiency of synchronizing data words, depending on the number of registers, is necessary. This necessary and sufficient bound is crucial to establish membership of synchronizing DRAs in PSPACE.

▶ **Lemma 4.** *There is a family of single-letter DRAs* $(\mathcal{R}_n)_{n \in \mathbb{N}}$*, with* $n = |\mathsf{reg}|$ *registers and* $\mathcal{O}(n)$ *locations, such that all synchronizing data words have data efficiency* $\mathcal{O}(n)$.

The synchronization problem for k-DRA is in PSPACE using the following co-(N)PSPACE algorithm: (1) picking a set $X = \{x_1, x_2, \cdots, x_{2k+1}\}$ of distinct data values, and (2) guessing some location $\ell \in \mathcal{L}$ and checking if there is no word $w \in (\Sigma \times \{x_1, x_2, \cdots, x_k\})^*$ with length $|w| \leq 2^{k|\mathcal{L}||\Sigma|}$ such that along firing inequality-guarded (on all k registers) transitions, some registers are not updated. Next (3) guessing two states $q_1, q_2 \in \mathcal{L} \times X^k$ such that there is no word $w \in (\Sigma \times X)^*$ with length $|w| \leq 2^{(2k+1)|\mathcal{L}||\Sigma|}$ such that $|\mathsf{post}(\{q_1, q_2\}, w)| = 1$.

▶ **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

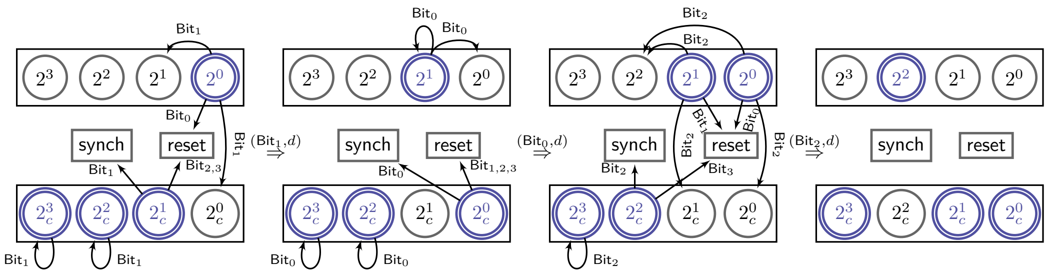


Figure 3 A partial illustration of the incrementing process of the 1-NRA $\mathcal{R}_{\mathsf{counter}}$ of Fig. 4. All Bit_i-transitions are equipped with equality guards. There is an x-token in all doubled transitions.

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 $\mathcal{O}(n)$ locations) where $\mathsf{Ackermann}(n)$ distinct data must be read and another family where an input datum $x \in \mathsf{D}$ 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. To give the intuition behind the constructions, we say an x-token is in location ℓ of an RA after reading a data word v if $(\ell, x) \in \mathsf{post}(\mathcal{L} \times \mathsf{D}, v)$.

The 1-NRA $\mathcal{R}_{\mathsf{counter}}$ shown in Figure 4 encodes a binary counter: In every synchronizing data word w, some datum $x \in \mathsf{data}(w)$ must appear at least 2^3 times. The location synch has self-loops on all letters, thus, $\mathcal{R}_{\mathsf{counter}}$ is only synchronized in synch. Generally speaking, the counting involves *resetting* and *incrementing*. The resetting places an x-token in the location zero by an unavoidable $\star$-transition (tokens in reset can only move out by $\star$-transitions). The numbers $m \leq 2^3$ are represented by placing x-tokens in the locations corresponding to binary representation of m. An x-token in location 2^i (and in 2^i_c) means that the i-th least significant bit in binary representation is set to 1 (to 0). First, by resetting, a Bit_0-transition places x-tokens in $\{2^3_c, 2^2_c, 2^1_c, 2^0\}$ to represent 0001. Next, an incrementing process can be set off by inputting the datum x via equality guards. In each increment step the tokens are replaced by firing specific Bit_i-transitions $(0 \leq i \leq 3)$, following the standard procedure of binary addition. Figure 3 shows the three increment steps. At the end, #-transitions takes the token in 2^3 to location synch and finally synchronize $\mathcal{R}_{\mathsf{counter}}$.

▶ **Lemma 6.** *There is a family of 1-NRAs $(\mathcal{R}_{\mathsf{counter}(n)})_{n \in \mathbb{N}}$ with $\mathcal{O}(n)$ locations, such that for all synchronizing data words w, some datum $d \in \mathsf{data}(w)$ 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 $\mathsf{tower} : \mathbb{N} \to \mathbb{N}$ is defined inductively by $\mathsf{tower}(0) = 1$ and $\mathsf{tower}(n+1) = 2^{\mathsf{tower}(n)}$.

Figure 4 shows the 1-NRA $\mathcal{R}_{\mathsf{tower}}$ over the data domain $\mathbb{N}$. We indicate that $|\mathsf{data}(w)| \in \mathcal{O}(\mathsf{tower}(3))$ for all synchronizing data words w. As in $\mathcal{R}_{\mathsf{counter}}$, synch is the location where the RA must be synchronized in, and an initial *reset* is enforced to reach the location Data_1. The main issue is that while synchronizing $\mathcal{R}_{\mathsf{tower}}$, some inequality-guarded transitions are unavoidable, which are the ones that may *replicate* the tokens. For example, if one token in Data_1, firing two transitions $\mathsf{Data}_1 \xrightarrow{\neq r \ \mathsf{rep}} \mathsf{Data}_{1,2}$ and $\mathsf{Data}_1 \xrightarrow{\neq r \ \mathsf{rep} \ r\downarrow} \mathsf{Data}_{1,2}$ replicates it to two tokens in $\mathsf{Data}_{1,2}$.

Since the question is the required data efficiency of synchronizing words, we always start from datum 1 and feed $\mathcal{R}_{\mathsf{tower}}$ with the smallest number i which contributes to synchronization.

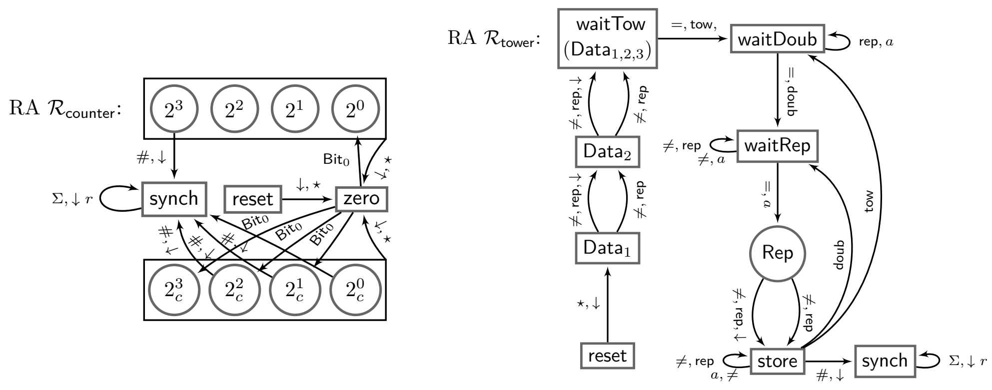


Figure 4 Bit_i-transitions in $\mathcal{R}_{\mathsf{counter}}$ have equality guards. Most of the Bit_i-transitions are omitted; see Figure 3 for partial illustration of such transitions. Not-drawn $\star$-transitions activate a reset to zero in $\mathcal{R}_{\mathsf{counter}}$, resp. to Data_1 in $\mathcal{R}_{\mathsf{tower}}$. All inconsistent and inefficient transitions are omitted.

Moreover, when *resetting* we read datum 1. To synchronize $\mathcal{R}_{\mathsf{tower}}$ with the least data efficiency, we go through the following steps:

▷ **resetting to** Data_1: the $\star$-transitions reset and place one token in Data_1 by $\ell \xrightarrow{\star\ r\downarrow} \mathsf{Data}_1$ for all $\ell \in \mathcal{L} \setminus \{\mathsf{synch}\}$. Reading $\star$ is necessary for synchronizing since tokens in reset only move out by a $\star$-transition. Since another $\star$ eliminates all tokens and places one token in Data_1 again, resetting is inefficient; we call all transitions directed to reset *inefficient*.

▷ **replicating towering tokens**: after a reset with $(\star, 1)$ and having a 1-token in Data_1, the only efficient transitions are on $(\mathsf{rep}, 2)(\mathsf{rep}, 3)$, which results in replicating the 1-token in 3 tokens (shown as $\{1, 2, 3\}$-tokens) and placing them in waitTow.

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

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

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

▷ **3-token**: inputting $(\mathsf{doub}, 3)$ moves $\{1, 2, 3, 4\}$-tokens into 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 $(a, i)(\mathsf{rep}, 4+i)$. 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, $\mathsf{tower}(3) = 2^{\mathsf{tower}(2)} = 16$ distinct data are needed to synchronize $\mathcal{R}_{\mathsf{tower}}$.

▶ **Lemma 7.** *There is a family of* 1*-NRAs* $(\mathcal{R}_{\mathsf{tower}(n)})_{n\in\mathbb{N}}$ *with* $O(n)$ *locations, such that* $|\mathsf{data}(w)| \in \mathcal{O}(\mathsf{tower}(n))$ *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 $\mathcal{R}_n^m$ $(n, m \in \mathbb{N})$ such that all synchronizing data words have data efficiency at least $\mathsf{ack}_n(m)$, where ack_n is at level n of the Ackermann-hierarchy.

To define the language of a given RA $\mathcal{R}$, we equip it with an initial location ℓ_i and a set $\mathcal{L}_f$ of accepting locations, where, without loss of generality, we assume that all outgoing transitions from ℓ_i update all registers. The language $L(\mathcal{R})$ is the set of all data words $w \in (\Sigma \times \mathsf{D})^+$, for which there is a run from (ℓ_i, ν_i) to (ℓ_f, ν_f) such that $\ell_f \in \mathcal{L}_f$ and $\nu_i, \nu_f \in \mathsf{D}^{|\mathsf{reg}|}$. The universality problem asks, given an RA, whether $L(\mathcal{R}) = (\Sigma \times \mathsf{D})^+$. 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, $\mathsf{data}(w) = \{x\}$) such that $\mathsf{post}(\mathcal{L} \times \mathsf{D}, w) \subseteq \mathcal{L} \times \mathsf{data}(w)$. Considering this fact as the skeleton, we define a language lang such that data words in this language are encodings of the synchronizing process. Let $\mathcal{L} = \{\ell_1, \ell_2, \cdots, \ell_n\}$ be the set of locations and x, y two distinct data. Each data word in lang, if there exists any, consists of

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

▷ **a sequence of normal blocks**: the delimiter $(\star, y)$, 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 $(\star, y)$, a single successor reached from states and input in the previous block and the delimiter $(\star, y)$.

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

We then construct a 1-NRA $\mathcal{R}_{\mathsf{comp}}$ that accepts the complement of lang; thus $\mathcal{R}$ has some synchronizing data word if, and only if, the language of $\mathcal{R}_{\mathsf{comp}}$ is not universal. The 1-NRA $\mathcal{R}_{\mathsf{comp}}$ is a finite union of smaller 1-NRAs, each of them violating one of the membership conditions for lang. For instance, the membership condition stated above is violated by the following 1-NRA.

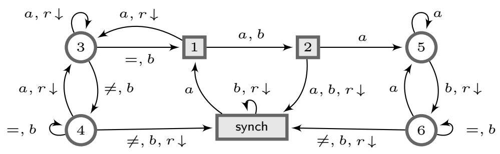
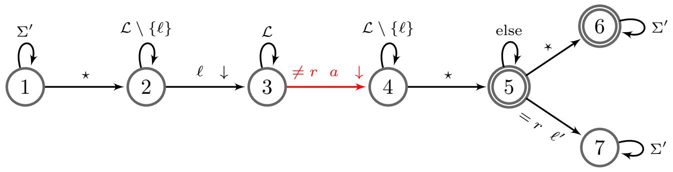


Figure 5 An RA where all synchronizing data words with length at most 3 require data efficiency 3 to shrink the infinite set of states to a finite subset.



▶ **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 $\mathcal{L} \times \{x\}$; next, we synchronize this successor set in a singleton. The RA $\mathcal{R}$ 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 $a \in \Sigma$, two binary operations $\cdot$ (concatenation) and $+$ (union), and a unary operation ^{2} (squaring). The language $L(\mathsf{expr})$ of such expressions expr is defined inductively as in regular expressions, where $L(\mathsf{expr}^2) = L(\mathsf{expr}) \cdot L(\mathsf{expr})$. The *bounded universality problem* asks, given a regular-like expression expr and length $N \in \mathbb{N}$ written in binary, whether $L(\mathsf{expr})$ includes all strings with length at most N; in other words, if $\Sigma^{\leq N} \subseteq L(\mathsf{expr})$.

▶ 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 $u \notin L(\mathsf{expr})$. 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 $|\Sigma| = 2$.

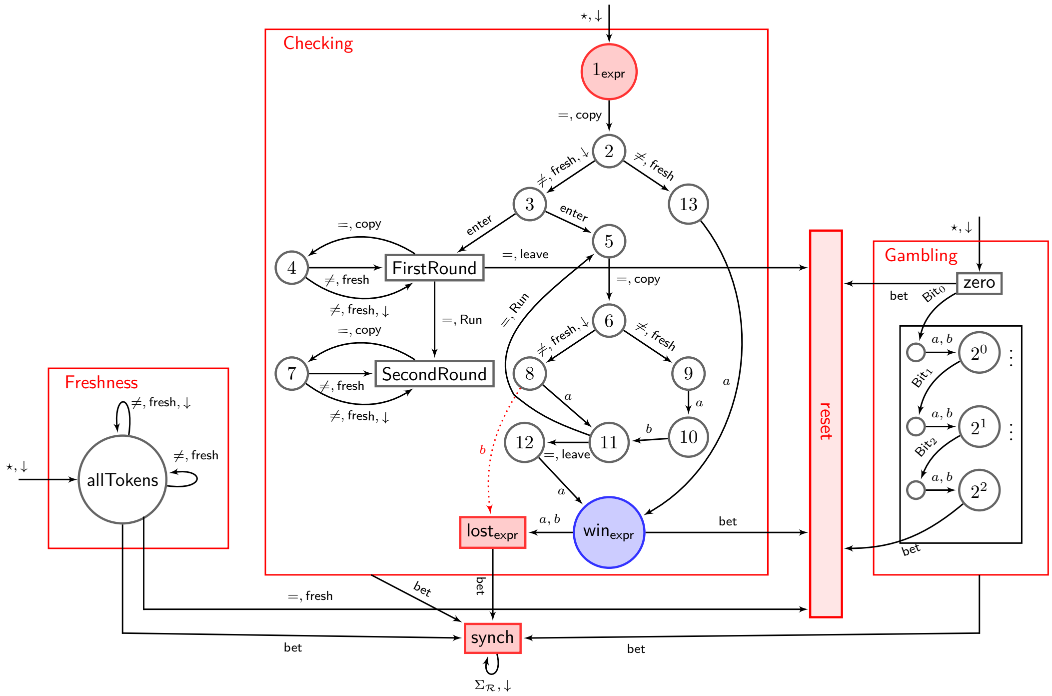


Figure 6 The $\star$-transitions reset $\mathcal{R}$, and all not-drawn a, b-transitions are inconsistent (except in $\mathsf{allTokens}$). Other not-drawn transitions are self-loops.

Given a regular-like expression expr and length $N \in \mathbb{N}$, we construct a 1-NRA $\mathcal{R}$ and $\mathsf{length} \in \mathbb{N}$, such that the language of expr is bounded universal if and only if $\mathcal{R}$ has no synchronizing data word with length at most length. The RA $\mathcal{R}$ consists of two distinguished locations reset, synch and three main gadgets: *Gambling*, *Freshness* and *Checking* gadget.

The RA $\mathcal{R}$ relies on the instincts of a *gambler* to synchronize. When feeding $\mathcal{R}$ with a data word w, we say that there is an x-token in location ℓ if $(\ell, x) \in \mathsf{post}(\mathcal{L} \times \mathsf{D}, w)$. Intuitively, whenever a token is in location reset, the gambler must restart; and $\mathcal{R}$ 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 $\mathsf{lost}_{\mathsf{expr}}$ (of Checking gadget); and conversely, if a sub-expression definitely generates u, then its token moves to $\mathsf{win}_{\mathsf{expr}}$ (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 $\mathsf{win}_{\mathsf{expr}}$, move to synch. In this way, $\mathcal{R}$ synchronizes in synch if $|u| \leq N$ and $u \notin L(\mathsf{expr})$.

Figure 6 depicts the constructed $\mathcal{R}$ for $\mathsf{expr} = (a+ab)^2a + a$ 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 $\star$-transition. When a reset happens, the gambler has the chance to change the guessed string u and to restart. Resetting eliminates all tokens in $\mathcal{R}$ and places tokens only to synch and the initial locations of all gadgets: zero, $\mathsf{allTokens}$ and 1_{expr}.

Gambler must only bet on $|u| \leq N$: 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 $|u| \leq N$. This gadget is a chain of (modified) counting RAs $\mathcal{R}_{\mathsf{counter}(i)}$ described in Lemma 6, where $\mathcal{R}_{\mathsf{counter}(i)}$ counts until 2^i. We modify $\mathcal{R}_{\mathsf{counter}(i)}$ such that the *increment process*, triggered by Bit_i-transitions is executed after each occurrence of a or b. Gambling gadget in Figure 6 must count up to $N+1 = 2^2$ that is achieved by calling $\mathcal{R}_{\mathsf{counter}(2)}$.

Freshness gadget: after a reset, Checking gadget starts with a single token in 1_{expr}, 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 $a + ab$, 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 $\mathsf{allTokens} \xrightarrow{=r\ \mathsf{fresh}} \mathsf{reset}$ happens.

Checking gadget: The checking is the gadget for expr that is built inductively from gadgets a, b, ab, $a+ab$, $(a+ab)^2$ and $(a+ab)^2a$. After a reset, it starts with a single token in 1_{expr}, if $u \in L(\mathsf{expr})$, then some token moves to $\mathsf{win}_{\mathsf{expr}}$ spoiling the gambler's plan in synchronizing. We explain the core of the sub-gadgets by following the scenario for $\mathcal{R}$ of $\mathsf{expr} = (a+ab)^2a+a$:

▷ **When gambler bets on a wrong witness** $u \in L(\mathsf{expr})$, such as aaa. After a reset, assuming that an x-token is in 1_{expr}, it replicates by $(\mathsf{copy}, x)(\mathsf{fresh}, y)$ with $x \neq y$ to $\{x, y\}$-tokens. The x-token moves to 13 entering the a-gadget, and y-token to 3 entering the $(a+ab)^2a$-gadget. The only consistent transition is enter, the initial transition in the squaring. It makes a copy of the entering token in $\mathsf{FirstRound}$ to enforce the token to go through the gadget under squaring, two times. After (enter, y), there are y-tokens in $\mathsf{FirstRound}$ and in 5 as the initial location of the $(a+ab)$-gadget. For the union $a+ab$, inputting $(\mathsf{copy}, y)(\mathsf{fresh}, z)$ replicates the y-token in 5 to $\{y, z\}$-tokens where Freshness gadget guarantees that z is globally fresh. The z-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 $\mathsf{FirstRound}$ (and in $\mathsf{SecondRound}$) must be replicated too: so $(\mathsf{copy}, y)(\mathsf{fresh}, z)$ replicates the y-token in $\mathsf{FirstRound}$ to $\{y, z\}$-tokens. Next a-transitions are consistent; observe that three tokens $\{x, y, z\}$ check if a is generated: as in $(a+ab)^2a+a$, the first produced a may be the result of three expressions: lonely a or a, ab under squaring.
The x-token from 13 moves to $\mathsf{win}_{\mathsf{expr}}$ meaning that $a \in L(\mathsf{expr})$; however, the gambler is betting on aaa, and the second a wastes this (fake) win by moving the token to $\mathsf{lost}_{\mathsf{expr}}$.
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 $\mathsf{FirstRound}$ to $\mathsf{SecondRound}$ (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 $(\mathsf{copy}, y)(\mathsf{fresh}, d)$ $(a, y)(\mathsf{leave}, y)(a, y)$, the y-token arrives in $\mathsf{win}_{\mathsf{expr}}$. The gambler places the bet with no more a, b, meaning that the y-token in $\mathsf{win}_{\mathsf{expr}}$ has no way to get synchronized, as it moves to reset by the bet-transition.
▷ **When gambler bets on a right witness** $u \notin L(\mathsf{expr})$, such as bb. Observe that $(\star, x)(\mathsf{copy}, x)(\mathsf{fresh}, y)(\mathsf{enter}, x)(\mathsf{copy}, y)(\mathsf{fresh}, z)(b, x)(b, x)(\mathsf{bet}, x)$ synchronizes $\mathcal{R}$ into synch.

▷ **When gambler cheats** by betting on strings longer than N, such as *abbb*. The issue is when $abbb \notin L(\mathsf{expr})$, in these cases data words such as $(\star, x)(\mathsf{copy}, x)(\mathsf{fresh}, y)(\mathsf{enter}, x)(\mathsf{copy}, y)(\mathsf{fresh}, z)(a, x)(\mathsf{run}, y)(\mathsf{copy}, y)(\mathsf{fresh}, d)(b, x)(\mathsf{run}, z)(\mathsf{copy}, z)(\mathsf{fresh}, m)(b, x)(b, x)$ would place all tokens of Checking gadget in $\mathsf{lost}_{\mathsf{expr}}$. 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 $\mathsf{length} = 14$ 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.

▶ **Lemma 12.** *The bounded synchronization problem for NRAs is* NEXPTIME*-hard.*

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

▶ **Theorem 13.** *The bounded synchronization problem for NRAs is* NEXPTIME*-complete.*

The *bounded universality problem* asks, given an RA and $\mathsf{length} \in \mathbb{N}$ encoded in binary, whether all data words w with $|w| \leq \mathsf{length}$ 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 $\mathcal{L}_f$ of accepting locations including all locations but synch.

▶ **Theorem 14.** *The bounded universality problem for NRAs is* co-NEXPTIME*-complete.*

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

References

1. Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1):1:1–1:39, February 2008. doi:10.1145/1322432.1322433.
2. Pablo Barceló, Leonid Libkin, Anthony W. Lin, and Peter T. Wood. Expressive languages for path queries over graph-structured data. *ACM Trans. Database Syst.*, 37(4):31:1–31:46, December 2012. doi:10.1145/2389241.2389250.
3. Yaakov Benenson, Rivka Adar, Tamar Paz-Elizur, Zvi Livneh, and Ehud Shapiro. DNA molecule provides a computing machine with both data and fuel. *Proc. National Acad. Sci. USA*, 100:2191–2196, 2003. doi:10.1073/pnas.0535624100.
4. Mikołaj Bojańczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, and Claire David. Two-variable logic on words with data. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 7–16. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.51.
5. Mikolaj Bojanczyk and Pawel Parys. Xpath evaluation in linear time. *J. ACM*, 58(4):17:1–17:33, July 2011. doi:10.1145/1989727.1989731.
6. Ahmed Bouajjani, Peter Habermehl, Yan Jurski, and Mihaela Sighireanu. Rewriting systems with data. In Erzsébet Csuhaj-Varjú and Zoltán Ésik, editors, *Fundamentals of Computation Theory, 16th International Symposium, FCT 2007, Budapest, Hungary, August 27-30, 2007, Proceedings*, volume 4639 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2007. doi:10.1007/978-3-540-74240-1_1.

7 Patricia Bouyer, Antoine Petit, and Denis Thérien. An algebraic approach to data languages and timed languages. *Inf. Comput.*, 182(2):137–162, 2003. doi:10.1016/S0890-5401(03)00038-5.

8 Ján Černý. Poznámka k homogénnym experimentom s konečnými automatmi. *Matematicko-fyzikálny časopis*, 14(3):208–216, 1964.

9 Krishnendu Chatterjee and Laurent Doyen. Computation tree logic for synchronization properties. In *To be appear in 43rd Internation Colloquim on Automata, Languages, and programming, ICALP 2016*, 2016.

10 Dmitry Chistikov, Pavel Martyugin, and Mahsa Shirmohammadi. Synchronizing automata over nested words. In *Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9634 of *Lecture Notes in Computer Science*, pages 252–268. Springer, 2016.

11 Lorenzo Clemente and Slawomir Lasota. Timed pushdown automata revisited. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 738–749. IEEE, 2015. doi:10.1109/LICS.2015.73.

12 Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3), 2009. doi:10.1145/1507244.1507246.

13 Stéphane Demri, Ranko Lazic, and David Nowak. On the freeze quantifier in constraint LTL: decidability and complexity. *Inf. Comput.*, 205(1):2–24, 2007. doi:10.1016/j.ic.2006.08.003.

14 Stéphane Demri, Ranko Lazic, and Arnaud Sangnier. Model checking memoryful linear-time logics over one-counter automata. *Theor. Comput. Sci.*, 411(22-24):2298–2316, 2010. doi:10.1016/j.tcs.2010.02.021.

15 Laurent Doyen, Line Juhl, Kim Guldstrand Larsen, Nicolas Markey, and Mahsa Shirmohammadi. Synchronizing words for weighted and timed automata. In Venkatesh Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, volume 29 of *LIPIcs*, pages 121–132. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.121.

16 Laurent Doyen, Thierry Massart, and Mahsa Shirmohammadi. Infinite synchronizing words for probabilistic automata. In *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 278–289. Springer, 2011.

17 Diego Figueira. Satisfiability of downward xpath with data equality tests. In *Proceedings of the Twenty-eighth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '09, pages 197–206, New York, NY, USA, 2009. ACM. doi:10.1145/1559795.1559827.

18 Diego Figueira. Alternating register automata on finite words and trees. *Logical Methods in Computer Science*, 8(1), 2012. doi:10.2168/LMCS-8(1:22)2012.

19 Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen. Ackermannian and primitive-recursive bounds with Dickson's lemma. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*, pages 269–278. IEEE Computer Society, 2011. doi:10.1109/LICS.2011.39.

20 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9.

21 Jan Kretínský, Kim Guldstrand Larsen, Simon Laursen, and Jirí Srba. Polynomial time decidability of weighted synchronization under partial observability. In *26th International*

Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015, volume 42 of *LIPIcs*, pages 142–154. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

22 Kim Guldstrand Larsen, Simon Laursen, and Jirí Srba. Synchronizing strategies under partial observability. In *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*, volume 8704 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2014.

23 Alexei Lisitsa and Igor Potapov. Temporal logic with predicate lambda-abstraction. In *12th International Symposium on Temporal Representation and Reasoning (TIME 2005), 23-25 June 2005, Burlington, Vermont, USA*, pages 147–155. IEEE Computer Society, 2005. doi:10.1109/TIME.2005.34.

24 Pavel V. Martyugin. Complexity of problems concerning carefully synchronizing words for PFA and directing words for NFA. In *Computer Science - Theory and Applications, 5th International Computer Science Symposium in Russia, CSR 2010, Kazan, Russia, June 16-20, 2010. Proceedings*, volume 6072 of *Lecture Notes in Computer Science*, pages 288–302. Springer, 2010.

25 Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004. doi:10.1145/1013560.1013562.

26 Jean-Eric Pin. Sur les mots synthronisants dans un automate fini. *Elektronische Informationsverarbeitung und Kybernetik*, 14(6):297–303, 1978.

27 Hiroshi Sakamoto and Daisuke Ikeda. Intractability of decision problems for finite-memory automata. *Theor. Comput. Sci.*, 231(2):297–308, 2000. doi:10.1016/S0304-3975(99)00105-X.

28 Sylvain Schmitz. Complexity hierarchies beyond elementary. *ACM Trans. Comput. Theory*, 8(1):3:1–3:36, 2016.

29 Mahsa Shirmohammadi. Phd thesis: Qualitative analysis of probabilistic synchronizing systems. 2014.

30 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison, Richard M. Karp, and H. Raymond Strong, editors, *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA*, pages 1–9. ACM, 1973. doi:10.1145/800125.804029.

31 Nikos Tzevelekos. Fresh-register automata. In Thomas Ball and Mooly Sagiv, editors, *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 295–306. ACM, 2011. doi:10.1145/1926385.1926420.

32 Mikhail V. Volkov. Synchronizing automata and the cerny conjecture. In Carlos Martín-Vide, Friedrich Otto, and Henning Fernau, editors, *Language and Automata Theory and Applications, Second International Conference, LATA 2008, Tarragona, Spain, March 13-19, 2008. Revised Papers*, volume 5196 of *Lecture Notes in Computer Science*, pages 11–27. Springer, 2008. doi:10.1007/978-3-540-88282-4_4.