

# Synchronizing Data Words for Register Automata

Karin Quaas  
Universität Leipzig

Mahsa Shirmohammadi  
CNRS & IRIF

## Abstract

Register automata (RAs) are finite automata extended with a finite set of registers to store and compare data from an infinite domain. 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 synchronization problem for DRAs is in general PSPACE-complete, and it is NLOGSPACE-complete for 1-DRAs. For nondeterministic RAs (NRAs), we show that Ackermann( $n$ ) distinct data (where  $n$  is the size of the RA) might be necessary to synchronize. The synchronization problem for NRAs is in general undecidable, however, we establish Ackermann-completeness of the problem for 1-NRAs. Another main result is the NEXPTIME-completeness of the length-bounded synchronization problem for NRAs, where a bound on the length of the synchronizing data word, written in binary, is given. A variant of this last construction allows to prove that the length-bounded universality problem for NRAs is co-NEXPTIME-complete.

## 1 Introduction

Given a deterministic finite automaton (DFA), a *synchronizing word* is a word that sends all states of the automaton to a unique state. Synchronizing words for finite automata have been studied since the 1970s [8, 25, 30, 23] and are the subject of one of the most well known open problems in automata theory—the Černý conjecture. This conjecture states that the length of a shortest synchronizing word for a DFA with  $n$  states is at most  $(n - 1)^2$ . Synchronizing words moreover have applications in planning, control of discrete event systems, biocomputing, and robotics [3, 30, 15]. More recently the notion has been generalized from automata to games [21, 28, 20] and infinite-state systems [14, 9], with applications to modelling complex systems such as distributed data networks or real-time embedded systems.

In this paper we are interested in *synchronizing data words* for *register automata*. *Data words* are sequences of pairs, where the first element of each pair is taken from a finite alphabet and the second element is taken from *an infinite data domain*, such as the natural numbers or ASCII strings. Data words have applications in querying and reasoning about data models with complex structural properties, e.g., XML and graph databases [1, 16, 5, 2]. For reasoning about data words, various formalisms have been considered, including first-order logic for data words [4, 6], extensions of linear temporal logic [22, 12, 11, 13], data automata [7, 4], register automata [19, 26, 24, 11] and extensions thereof, e.g. [29, 17, 10].

*Register automata* (RAs) are a generalization of finite automata for processing data words. RAs are equipped with a finite set of registers that can store data values. While processing a data word such an automaton can store the datum at the current position in one of its registers; it can also test the current datum for equality with data already stored in its registers. In applications, RAs allow for handling parameters such as user names, passwords, identifiers of connections, sessions, etc. RAs come in many variants, including one-way, two-way, deterministic, nondeterministic, and alternating. For alternating one-way RAs, classical language-theoretic decision problems, such as emptiness, universality and inclusion are undecidable. In this paper, we focus on the class of one-way nondeterministic RAs, which have a decidable emptiness problem [19], and the subclass of nondeterministic RAs with a single register, which has a decidable universality problem [11].

Semantically, an RA defines an infinite-state transition system due to the unbounded domain for the data stored in the registers. Synchronizing words were introduced for infinite-state systems with infi-

nite branching in [14, 28]; 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, finding the right definition of synchronizing word is however more challenging [9]. We define the *synchronization problem* for RAs within the framework suggested in [14, 28]: given an RA  $\mathcal{R}$  over a finite alphabet  $\Sigma$  and an infinite data domain  $D$ , does there exist a data word  $w \in (\Sigma \times D)^+$  and some state  $q_w$  such that the word  $w$  sends each of the infinitely many states of  $\mathcal{R}$  to  $q_w$ ? Note that the state  $q_w$  depends on the word  $w$ ; we call such a data word a *synchronizing data word*.

*Contribution.* The problem of finding synchronizing data words for RAs poses 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 as the *data efficiency* of synchronizing data words. We show that the data efficiency is polynomial in the number of registers for deterministic RAs (DRAs). For nondeterministic RAs (NRAs), we provide an example that shows that the data efficiency may be Ackermann( $n$ ), where  $n$  is the number of states of the NRA. Remarkably, the data efficiency is tightly related to the complexity of deciding the synchronization problem. 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  $(\mathcal{R}_k)_{k \in \mathbb{N}}$  of DRAs with  $k$  registers, for which indeed a polynomial data efficiency (in  $k$ ) 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 [14]. We show that the synchronization problems for DRAs with a single register (1-DRAs) and for 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 synchronization problem and the non-universality problem for 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.

Another main contribution is to prove NEXPTIME-completeness of the *length-bounded synchronization problem* for NRAs: given a bound on the length (written in binary), does there exist a synchronizing data word with length at most the given bound? For the lower bound, we present a reduction from the membership problem of  $\mathcal{O}(2^n)$ -time bounded nondeterministic Turing machines. 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 *length-bounded universality problem* for NRAs; the length-bounded universality problem asks whether all data words of length at most a given bound (written in binary) are in the language of the automaton. We further make a connection to the emptiness problem of single-register alternating RAs.

An extended abstract of this article has appeared in the Proceedings of the 41st International Symposium on Mathematical Foundations of Computer Science, (MFCS) 2016 [?]. In comparison with the extended abstract, here we simplify two of the main constructions and add detailed proofs of all results. The main improvement is giving a simpler NEXPTIME-hardness reduction for the length-bounded synchronization problem for NRAs.

## 2 Preliminaries

A deterministic finite-state automaton (DFA) is a tuple  $\mathcal{A} = \langle Q, \Sigma, \Delta \rangle$ , where  $Q$  is a finite set of states,  $\Sigma$  is a finite alphabet, and  $\Delta : Q \times \Sigma \rightarrow Q$  is a transition function that is totally defined. The function  $\Delta$  extends to finite words in a natural way:  $\Delta(q, wa) = \Delta(\Delta(q, w), a)$  for all words  $w \in \Sigma^*$  and letters  $a \in \Sigma$ ; it extends to all sets  $S \subseteq Q$  by  $\Delta(S, w) = \bigcup_{q \in S} \Delta(q, w)$ .

**Data Words and Register automata.** For the rest of this paper, fix an infinite data domain  $D$ . Given a finite alphabet  $\Sigma$ , a *data word over  $\Sigma$*  is a finite words over  $\Sigma \times D$ . For a data word  $w = (a_1, d_1)(a_2, d_2) \cdots (a_n, d_n)$ , the length  $|w|$  of  $w$  is  $n$ . We use  $\mathbf{data}(w) = \{d_1, \dots, d_n\} \subseteq D$  to refer to the set of data values occurring in  $w$ , and we define the *data efficiency of  $w$*  to be  $|\mathbf{data}(w)|$ .

Let  $R$  be a finite set of *register variables*. We define *register constraints  $\phi$  over  $R$*  by the grammar

$$\phi ::= \mathbf{true} \mid =r \mid \phi \wedge \phi \mid \neg\phi,$$

where  $r \in R$ . We denote by  $\Phi(R)$  the set of all register constraints over  $R$ . We may use  $\neq r$  for the inequality constraint  $\neg(=r)$ . A *register valuation* is a mapping  $\nu : R \rightarrow D$  that assigns a data value to each register; we sometimes write  $\nu = (\nu(r_1), \dots, \nu(r_k)) \in D^k$ , where  $R = \{r_1, \dots, r_k\}$ . 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,  $((d_1, d_2, d_1), d_2)$  satisfies  $((= r_1) \wedge (= r_2)) \vee (\neq r_3)$  if  $d_1 \neq d_2$ . For the set  $\text{up} \subseteq R$  and  $d \in D$ , we define the *update*  $\nu[\text{up} := d]$  of valuation  $\nu$  by  $(\nu[\text{up} := d])(r) = d$  if  $r \in \text{up}$ , and  $(\nu[\text{up} := d])(r) = \nu(r)$  otherwise.

A *register automaton* (RA) is a tuple  $\mathcal{R} = \langle L, R, \Sigma, T \rangle$ , where  $L$  is a finite set of locations,  $R$  is a finite set of registers,  $\Sigma$  is a finite alphabet and  $T \subseteq L \times \Sigma \times \Phi(R) \times 2^R \times L$  is a transition relation. We may use  $\ell \xrightarrow{\phi \ a \ \text{up}\downarrow} \ell'$  to show transitions  $(\ell, a, \phi, \text{up}, \ell') \in T$ . We call  $\ell \xrightarrow{\phi \ a \ \text{up}\downarrow} \ell'$  an  $a$ -transition and  $\phi$  the *guard* of this transition. A guard **true** is vacuously true and may be omitted. Likewise we may omit **up** if  $\text{up} = \emptyset$ . We may write  $r\downarrow$  when  $\text{up} = \{r\}$  is a singleton set. For NRAs with only one register, we may shortly write  $=$  and  $\neq$  for the guards  $=r$  and  $\neq r$ , respectively, and  $\downarrow$  for the update  $\downarrow r$ .

A *configuration* of  $\mathcal{R}$  is a pair  $(\ell, \nu) \in L \times D^{|R|}$  of a location  $\ell$  and a register valuation  $\nu$ . We describe the behaviour of  $\mathcal{R}$  as follows: Given a configuration  $q = (\ell, \nu)$  and some input  $(a, d) \in \Sigma \times D$  an  $a$ -transition  $\ell \xrightarrow{\phi \ a \ \text{up}\downarrow} \ell'$  may be fired from  $q$  if  $(\nu, d)$  satisfies the constraint  $\phi$ ; then  $\mathcal{R}$  moves to the *successor configuration*  $q' = (\ell', \nu')$ , where  $\nu' = \nu[\text{up} := d]$  is the update of  $\nu$ . By  $\text{post}(q, (a, d))$ , we denote the set of all successor configurations  $q'$  of  $q$  on input  $(a, d)$ . We extend **post** to sets  $S \subseteq L \times D^{|R|}$  of configurations by  $\text{post}(S, (a, d)) = \bigcup_{q \in S} \text{post}(q, (a, d))$ ; and we extend **post** to words by  $\text{post}(S, w \cdot (a, d)) = \text{post}(\text{post}(S, w), (a, d))$  for all words  $w \in (\Sigma \times D)^*$ , and all inputs  $(a, d) \in \Sigma \times D$ .

A run of  $\mathcal{R}$  over the data word  $w = (a_1, d_1)(a_2, d_2) \dots (a_n, d_n)$  is a sequence of configurations  $q_0 q_1 \dots q_n$ , where  $q_i \in \text{post}(q_{i-1}, (a_i, d_i))$  for all  $1 \leq i \leq n$ . If  $\mathcal{R}$  reaches a configuration  $q = (\ell, x)$  during processing a word  $w$ , we may say that *an  $x$ -token is in  $\ell$*  (or simply *a token is in  $\ell$* ).

In the rest of the paper, we consider *complete* RAs, meaning that for all configurations  $q \in L \times D^{|R|}$  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* RAs (DRAs) and *nondeterministic* (NRAs), where an RA is deterministic if  $|\text{post}(q, (a, d))| \leq 1$  for all configurations  $q$  and all inputs  $(a, d)$ . A  $k$ -NRA ( $k$ -DRA, respectively) is an NRA (DRA, respectively) with  $|R| = k$ .

**Synchronizing words and synchronizing data words.** *Synchronizing words* are a well-studied concept for DFAs, see, e.g., [30]. Informally, a synchronizing word leads the automaton from every state to the same state. Formally, the word  $w \in \Sigma^+$  is synchronizing for a DFA  $\mathcal{A} = \langle Q, \Sigma, \Delta \rangle$  if there exists some state  $q \in Q$  such that  $\Delta(Q, w) = \{q\}$ . The *synchronization problem* for DFAs asks, given a DFA  $\mathcal{A}$ , whether there exists some synchronizing word for  $\mathcal{A}$ .

The synchronization problem for DFAs is in NLOGSPACE by using the *pairwise synchronization* technique: given a DFA  $\mathcal{A} = \langle Q, \Sigma, \Delta \rangle$ , it is known that  $\mathcal{A}$  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 [30] for more details). The pairwise synchronization algorithm initially sets  $S_{|Q|} = Q$ . For  $i = |Q| - 1, \dots, 1$ , the algorithm repeats the following two steps: (a) For two distinct states  $q, q' \in S_{i+1}$ , find  $v_i$  such that  $\Delta(q, v_i) = \Delta(q', v_i)$ . (b) Set  $S_i = \Delta(S_{i+1}, v_i)$  (and repeat the loop). The word  $w = v_{|Q|-1} \dots v_1$  is synchronizing for  $\mathcal{A}$ .

We introduce synchronizing data words for RAs. Given an RA  $\mathcal{R} = \langle L, R, \Sigma, T \rangle$ , a data word  $w \in (\Sigma \times D)^+$  is *synchronizing* for  $\mathcal{R}$  if there exists some configuration  $q_w = (\ell, \nu)$  such that  $\text{post}(L \times D^{|R|}, w) = \{q_w\}$ . Intuitively, no matter what is the starting location and register valuation, by inputting the data word  $w$ ,  $\mathcal{R}$  will be in the unique successor configuration  $q_w$ . This configuration  $q_w$  depends on  $w$ . The *synchronization problem* for RAs asks, given an RA  $\mathcal{R}$  over a data domain  $D$ , whether there exists some synchronizing data word for  $\mathcal{R}$ . The *length-bounded synchronization problem* for RAs decides, given an RA  $\mathcal{R}$  and a bound  $N \in \mathbb{N}$  written in binary, whether there exists some synchronizing data word  $w$  for  $\mathcal{R}$  satisfying  $|w| \leq N$ .

### 3 Synchronizing data words for DRAs

In this section, we first show that the synchronization problems for 1-DRAs and DFAs are NLOGSPACE-interreducible, implying that the problem is NLOGSPACE-complete for 1-DRAs. Next, we prove that

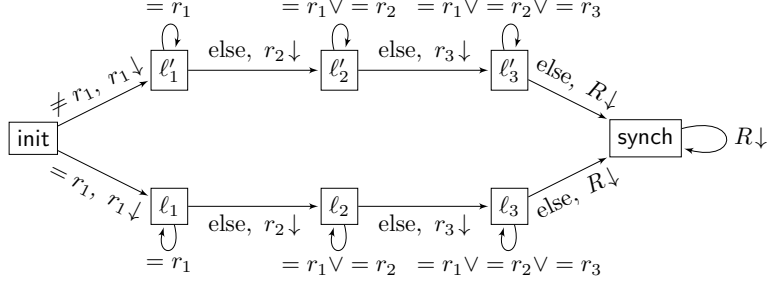


Figure 1: A DRA with registers  $r_1, r_2, r_3$  and the single letter  $a$  (omitted from transitions) that can be synchronized in the configuration  $(\text{synch}, x_4)$  by the data word  $w_{\text{synch}} = (a, x_1)(a, x_2)(a, x_3)(a, x_4)$  if  $\{x_1, x_2, x_3, x_4\} \subseteq D$  is a set of 4 distinct data.

the problem for  $k$ -DRAs, in general, can be decided in PSPACE; a reduction similar to a timed setting, as in [14], provides the matching lower bound. To obtain the complexity upper bounds, we prove that inputting words with data efficiency  $2|R| + 1$  is sufficient to synchronize a DRA.

The concept of synchronization requires that all runs of an RA, whatever the initial configuration (initial location and register valuations), end in the same configuration  $(\ell_{\text{synch}}, \nu_{\text{synch}})$ , only depending on the synchronizing data word  $w_{\text{synch}}$ , formally  $\text{post}(L \times D^{|R|}, w_{\text{synch}}) = \{(\ell_{\text{synch}}, \nu_{\text{synch}})\}$ . While processing a synchronizing data word, the infinite set of configurations of RAs must necessarily shrink to a finite set of configurations. The DRA  $\mathcal{R}$  with 3 registers depicted in Figure 1 illustrates this phenomenon. Consider the set  $\{x_1, x_2, x_3\} \subseteq D$  of distinct data values: starting from any of the infinite configurations in  $\{\text{init}\} \times D^3$ , when processing the data word  $(a, x_1)(a, x_2)(a, x_3)$ ,  $\mathcal{R}$  will be in a configuration in the finite set  $\{(\ell_3, (x_1, x_2, x_3)), (\ell'_3, (x_1, x_2, x_3))\}$ . We use this observation to provide a linear bound on the number of distinct data values that is sufficient for synchronizing DRAs.

In Lemma 1 below, we prove that data words over only  $|R|$  distinct data values are sufficient to shrink the infinite set of all configurations of DRAs to a finite set. We establish this result based on the following two key facts:

(1) When processing a synchronizing data word  $w_{\text{synch}}$  from a configuration  $(\ell, \nu)$  with some register  $r \in R$  such that  $\nu(r) \notin \text{data}(w_{\text{synch}})$ , 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). As an example, consider the DRA  $\mathcal{R}$  in Figure 1, and assume  $d_1, d_2 \notin \text{data}(w_{\text{synch}})$ . The two runs of  $\mathcal{R}$  starting from  $(\text{init}, d_1, d_1, d_1)$  and  $(\text{init}, d_2, d_2, d_2)$  first take the transition  $\text{init} \xrightarrow{\neq r_1 \ a \ r_1 \downarrow} \ell'_1$  updating register  $r_1$ . Next, the two runs must take  $\ell'_1 \xrightarrow{\text{else } a \ r_2 \downarrow} \ell'_2$  to update  $r_2$  and  $\ell'_2 \xrightarrow{\text{else } a \ r_3 \downarrow} \ell'_3$  to update  $r_3$ ; otherwise these two runs would never be synchronized in a single configuration.

(2) Moreover, to shrink the set  $L \times D^{|R|}$ , for every  $\ell \in L$ , one can find a word  $w_\ell$  that leads the DRA from  $\{\ell\} \times D^{|R|}$  to some finite set. Since  $\mathcal{R}$  is deterministic, appending some prefix or suffix to  $w_\ell$  achieves the same objective. This allows us to use a variant of the *pairwise synchronization* technique to shrink the infinite set  $L \times D^{|R|}$  to a finite set, by successively inputting  $w_\ell$  for a location  $\ell$  that appears with infinitely many data in the current successor set of  $L \times D^{|R|}$ .

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

*Proof.* Let  $\mathcal{R} = \langle L, R, \Sigma, T \rangle$  be a DRA on the data domain  $D$  with  $k \geq 1$  registers. Let  $v$  be a synchronizing data word for  $\mathcal{R}$  with  $N = |\text{data}(v)|$  distinct data. Suppose that  $k < N$ ; otherwise the statement of the lemma trivially holds.

For all  $1 \leq i \leq k$ , we say that  $x_i$  is the  $i$ -th datum in the synchronizing data word  $v = (a_1, d_1)(a_2, d_2) \cdots (a_n, d_n)$  if there exists  $j \leq k$  such that  $x_i = d_j$ ,  $x_i \notin \{d_1, \dots, d_{j-1}\}$  and  $|\{d_1, \dots, d_j\}| = i$ . For every  $i \leq k$ ,

denote by  $\langle L, i \rangle$  the set

$$\langle L, i \rangle = L \times \{\nu \in D^k \mid \exists R' \subseteq R \cdot |R'| \geq i \cdot \forall r \in R' \cdot \nu(r) \in \{x_1, \dots, x_i\}\}.$$

We **Claim** that for all locations  $\ell \in L$  and all  $1 \leq i \leq k$ , there exists some data word  $u_i$  such that

- $\text{data}(u_i) \subseteq \{x_1, x_2, \dots, x_i\}$ , and
- $\text{post}(\{\ell\} \times D^k, u_i) \subseteq \langle L, i \rangle$ , meaning that after reading  $u_i$  all reached configurations have at least  $i$  registers with values from  $\{x_1, x_2, \dots, x_i\}$ .

For  $\ell \in L$ , let  $w_\ell = u_k$  satisfy the above condition. Set  $S_0 = L \times D^k$  and  $w_0 = \varepsilon$ . Then, for all  $i = 1, \dots, |L|$ , repeat the following: if there exists some  $\ell \in L$  such that  $\{\ell\} \times (D \setminus \{x_1, \dots, x_k\})^k \cap S_{i-1} \neq \emptyset$ , then set  $w_i = w_\ell$  and  $S_i = \text{post}(S_{i-1}, w_i)$ . Otherwise set  $w_i = w_{i-1}$  and  $S_i = S_{i-1}$ . Observe that  $w = (w_i)_{1 \leq i \leq |L|}$  proves the statement of Lemma. It remains to prove the **Claim**.

**Proof of Claim.** Let  $\hat{\ell}$  be some location in the DRA  $\mathcal{R}$ . The proof is by an induction on  $i$ .

**Base of induction.** Let  $\text{wait} = \{\hat{\ell}\} \times (D \setminus \text{data}(v))^k$  be the set of configurations with location  $\hat{\ell}$  such that the data stored in all  $k$  registers is not in  $\text{data}(v)$ . Note that for all configurations  $(\hat{\ell}, \nu) \in \text{wait}$ , the unique run of  $\mathcal{R}$  starting in  $(\hat{\ell}, \nu)$  on (a prefix of)  $v$  consists of the same sequence of the following transitions:

- a prefix of transitions  $\xrightarrow{\bigwedge_{r \in R} \neq r \quad \emptyset \downarrow}$ , with inequality guards on all registers and with no register update,
- followed by a transition  $\xrightarrow{\bigwedge_{r \in R} \neq r \quad \text{up} \downarrow}$ , with inequality guard on all registers and with an update for some non-empty set  $\text{up} \subseteq R$ .

Otherwise, the two runs starting from any pair of configurations  $(\hat{\ell}, \nu_1), (\hat{\ell}, \nu_2) \in \text{wait}$  with unequal valuations  $\nu_1 \neq \nu_2$  would end up in distinct configurations, say  $(\ell, \nu'_1), (\ell, \nu'_2)$  with  $\nu'_1 \neq \nu'_2$ . This is a contradiction to the fact that the data word  $v$  is synchronizing.

Now let the inequality-guarded transition  $\xrightarrow{\bigwedge_{r \in R} \neq r \quad \text{up} \downarrow}$ , updating the registers in  $\text{up}$ , be fired at the  $j$ -th input  $(a_j, d_j)$  while reading  $v$ ; see Figure 2. We prove that the data word  $u_1 = (a_1, x_1)(a_2, x_1) \cdots (a_j, x_1)$  with  $\text{data}(u_1) = \{x_1\}$  guides  $\{\hat{\ell}\} \times D^k$  to a subset in which each configuration has some register with value  $x_1$ :  $\text{post}(\{\hat{\ell}\} \times D^k, u_1) \subseteq \langle L, 1 \rangle$ . This phenomenon is depicted in Figure 3 and can be argued as follows. Observe that  $x_1 = d_1$  is the first input datum; thus after inputting  $(a_1, x_1)$  the set of successors is a disjoint union of two branches:

- either at least one register  $r$  has datum  $x_1$  after the transition  $\xrightarrow{\bigvee_{r \in R} = r \quad a_1}$ . All the following successors in this branch, on input  $(a_2, x_1)(a_3, x_1) \cdots (a_j, x_1)$ , preserve the datum  $x_1$  in the register  $r$ ;
- or none of the registers is assigned  $x_1$  after the transition  $\xrightarrow{\bigwedge_{r \in R} \neq r \quad a_1}$ . By inputting  $(a_2, x_1)(a_3, x_1) \cdots (a_j, x_1)$ , all the following successors in this branch, thus, take inequality-guarded transitions, and would not update any registers, except for the last transition  $\xrightarrow{\bigwedge_{r \in R} \neq r \quad \text{up} \downarrow}$  fired by  $(a_j, x_1)$ .

The above argument proves that  $u_1$  with  $\text{data}(u_1) \subseteq \{x_1\}$  is such that  $\text{post}(\{\hat{\ell}\} \times D^k, u_1) \subseteq \langle L, 1 \rangle$ . The base of induction holds.

**Step of induction.** Assume that the induction hypothesis holds for  $i - 1$ , namely, there exists some word  $u_{i-1}$  with  $\text{data}(u_{i-1}) \subseteq \{x_1, \dots, x_{i-1}\}$  such that  $\text{post}(\{\hat{\ell}\} \times D^k, u_{i-1}) \subseteq \langle L, i - 1 \rangle$ . To construct  $u_i$ , we define the concept of a symbolic state: we say  $(\ell, \text{up}, \nu, j)$  is a symbolic state if  $\ell \in L$ , the set  $\text{up} \subseteq R$  of registers is such that  $|\text{up}| \geq \min(j, k)$  and  $\nu \in \{x_1, \dots, x_j\}^k$  and  $j \leq N$ . The semantics of  $(\ell, \text{up}, \nu, j)$  is the following set:

$$\llbracket (\ell, \text{up}, \nu, j) \rrbracket = \{\ell\} \times \{\nu' \in D^k \mid \nu'(r) = \nu(r) \text{ if } r \in \text{up}\}.$$

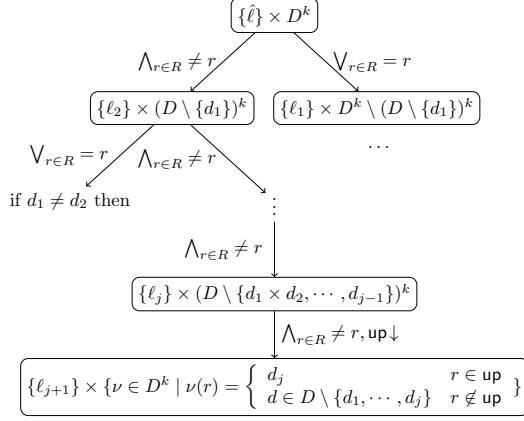


Figure 2: Runs of  $\mathcal{R}$  over the data word  $(a_1, d_1)(a_2, d_2) \cdots (a_j, d_j)$ .

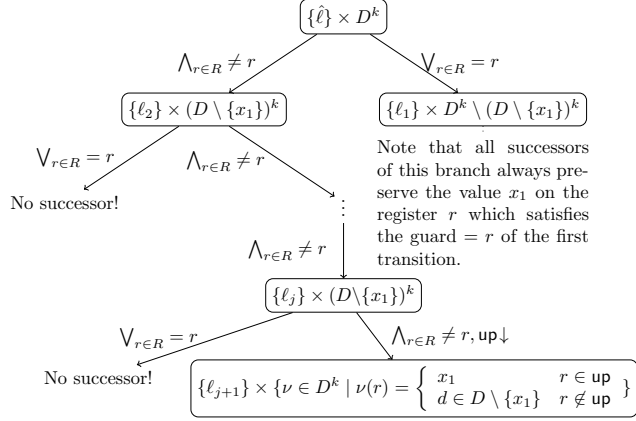


Figure 3: Runs of  $\mathcal{R}$  over the data word  $(a_1, x_1)(a_2, x_1) \cdots (a_j, x_1)$ .

Denote by  $\Gamma$  the set of all such symbolic states  $(\ell, \text{up}, \nu, i - 1)$ . By definition, the set  $\Gamma$  is finite. Now we can construct  $u_i$  as follows. Let  $S_0 = \text{post}(\{\hat{\ell}\} \times D^k, u_{i-1})$  and  $w_0 = u_{i-1}$ . Recall that  $S_0 \subseteq \langle L, i - 1 \rangle$  and observe that  $S_0 \subseteq \bigcup_{q \in \Gamma} \llbracket q \rrbracket$ . Start with  $j = 0$  and, while  $S_j \neq \emptyset$ , pick a symbolic state  $q = (\ell, \text{up}, \nu, i - 1)$  such that  $\llbracket q \rrbracket \cap S_j \neq \emptyset$  and construct a word  $u_q$  (as explained in the details below) such that

- $\text{data}(u_q) = \{x_1, x_2, \dots, x_i\}$ , and
- $\text{post}(\llbracket q \rrbracket, u_q) \subseteq \langle L, i \rangle$ .

Let  $S_{j+1} = \text{post}(S_j \setminus \llbracket q \rrbracket, u_q)$  and  $w_{j+1} = w_j \cdot u_q$ . Repeat the loop for  $j + 1$ . Observe that  $u_i = w_{j^*}$ , where  $j^* \leq |S_0|$  is such that  $S_{j^*} = \emptyset$ , satisfies the induction statement.

Below, given a symbolic state  $q = (\ell, \text{up}, \nu, i - 1)$ , the aim is to construct the data word  $u_q$ . Without loss of generality, we assume that  $|\text{up}| = i - 1$ ; otherwise  $u_q = u_{i-1}$ . Let

$$\text{wait} = \llbracket (\ell, \text{up}, \nu, i - 1) \rrbracket \cap \{ \ell \} \times \{ \nu' \mid \nu'(r) \in D \setminus \text{data}(v) \text{ if } r \notin \text{up} \}$$

be the set of all configurations in the symbolic state  $q$ , where all data stored in the registers  $r \notin \text{up}$  are not in  $\text{data}(v)$ . Similarly to the induction base, no matter what the register valuation in a configuration in  $\text{wait}$  looks like, the unique run of  $\mathcal{R}$  on the synchronizing word  $v = (a_1, d_1)(a_2, d_2) \cdots (a_n, d_n)$  starting in that configuration takes the same sequence of transitions. Since  $\nu \in \{x_0, \dots, x_{i-1}\}^k$ , after inputting successive data from  $\text{data}(v)$ , all successors of configurations in  $\text{wait}$  are elements of a symbolic state. For all  $0 \leq j \leq n$ , let the symbolic state  $q^j = (\ell^j, \text{up}^j, \nu^j, N)$  be such that  $\llbracket q^0 \rrbracket = \llbracket q \rrbracket \cap \text{wait}$ , and  $\text{post}(\llbracket q^{j-1} \rrbracket, (a_j, d_j)) \subseteq \llbracket q^j \rrbracket$  if  $j \geq 1$ .

In the sequel, we argue that there exists some  $1 \leq m \leq n$  such that, in the sequence of transitions from one symbolic state to another symbolic state over the prefix  $(a_1, d_1)(a_2, d_2) \cdots (a_m, d_m)$  of  $v$  (the first  $m$  inputs), the following holds:

- on inputting  $(a_j, d_j)$  for all  $1 \leq j < m$ , the transition  $\xrightarrow{(\bigwedge_{r \in \Lambda_j} = r) \wedge (\bigwedge_{r \notin \Lambda_j} \neq r) \ a_j \ \Gamma_j \downarrow}$  with  $\Lambda_j, \Gamma_j \subseteq \text{up}$  is taken from  $q^{j-1}$  to  $q^j$ . It implies that  $\nu^{j-1}(r) = d_j$  for all  $r \in \Lambda_j$ , and  $\nu^j(r) = d_j$  for all  $r \in \Gamma_j$ .
- and on inputting  $(a_m, d_m)$ , the transition  $\xrightarrow{(\bigwedge_{r \in \Lambda_m} = r) \wedge (\bigwedge_{r \notin \Lambda_m} \neq r) \ a_m \ \Gamma_m \downarrow}$ , that is taken from  $q^{m-1}$  to  $q^m$ , is such that  $\Lambda_m \subseteq \text{up}^m$  whereas  $\Gamma_m \not\subseteq \text{up}^m$ .

Now from the prefix  $(a_1, d_1)(a_2, d_2) \cdots (a_m, d_m)$  of  $v$ , *i.e.*, the first  $m$  inputs, and from the set of data  $\{x_1, x_2, \dots, x_i\}$ , we construct the word  $u_q = (a_1, y_1)(a_2, y_2) \cdots (a_m, y_m)$  for  $q = (\ell, \text{up}, \nu, i - 1)$  as follows: for all  $1 \leq j \leq m$ ,

- if  $\Lambda_j \neq \emptyset$ , *i.e.*, some register  $r \in \text{up}$  already stores the datum  $d_j$ , then  $y_j = d_j$ .
- if  $\Lambda_j = \emptyset$ , *i.e.*, none of the registers  $r \in \text{up}$  stores the datum  $d_j$ , then  $y_j = d$  where  $d \in \{x_1, x_2, \dots, x_i\} \setminus \{\nu^{j-1}(r) \mid r \in \text{up}\}$ . The existence of such  $d$  is guaranteed since  $|\text{up}| = i - 1$  and  $|\{x_1, x_2, \dots, x_i\}| = i$ . Moreover, since the transitions  $\xrightarrow{(\bigwedge_{r \in \text{up} \neq r} a_j \Gamma_j \downarrow)}$  have inequality guards for all registers, then changing the datum from  $d_j$  to  $y_j$  would result only in taking the same transition.

Observe that  $\text{data}(u_q) \subseteq \{x_1, \dots, x_i\}$ . As a result, all registers that are updated along the runs of  $\mathcal{R}$  over  $u_q$  store some datum from  $\{x_1, \dots, x_i\}$ . This argument shows that  $\text{post}(\llbracket q \rrbracket, u_q) \subseteq \langle L, i \rangle$ . This concludes the step of induction, and completes the proof.  $\square$

After reading some word that shrinks the infinite set of configurations of DRAs to a finite set  $S$  of configurations, we generalize the *pairwise synchronization* technique [30] to finally synchronize configurations in  $S$ . By this generalization, we achieve the following Lemma 2, for which the detailed proof can be found in Appendix 6.

**Lemma 2.** *For all DRAs for which there exist synchronizing data words, there exists a synchronizing data word  $w$  such that  $|w| \leq 2|R| + 1$ .*

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

**Theorem 3.** *The synchronization problem for 1-DRAs is 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  $n = |R|$  registers and  $\mathcal{O}(n)$  locations, such that all synchronizing data words have data efficiency  $\Omega(n)$ .*

*Proof.* The family of DRAs  $\mathcal{R}_n (n \in \mathbb{N})$  is defined over an infinite data domain  $D$ . The DRA  $\mathcal{R}_n$  has  $n$  registers and a single letter  $a$ . The structure of  $\mathcal{R}_n$  is composed of two distinguished locations **init** and **synch** and two chains, where each chain has  $n$  locations:  $\ell_1, \ell_2, \dots, \ell_n$  and  $\ell'_1, \ell'_2, \dots, \ell'_n$ . The DRA  $\mathcal{R}_3$  is shown in Figure 1. The only transition in **synch** is a self-loop with update on all  $n$  registers, thus  $\mathcal{R}_n$  can only be synchronized in **synch**. There are two transitions in **init**, each going to one of the chains:

$$\text{init} \xrightarrow{=r_1 \ a \ r_1 \downarrow} \ell_1 \quad \text{and} \quad \text{init} \xrightarrow{\neq r_1 \ a \ r_1 \downarrow} \ell'_1.$$

Then,  $\text{post}(\{\text{init}\} \times D^n, (a, x)) = \{\ell_1, \ell'_1\} \times (\{x\} \times D^{n-1})$  for all  $x \in D$ .

From  $\{\ell_1, \ell'_1\} \times (\{x\} \times D^{n-1})$ , informally speaking, in both chains the respective  $i$ -th locations are simultaneously reached after inputting  $i$  distinct data: for all  $1 \leq i < n$ , in each  $\ell_i$  and  $\ell'_i$  there are two transitions. One transition is a self-loop, with a satisfied equality guard on at least one of the updated registers  $r_1, \dots, r_i$  so far. The other transition goes to the next location  $\ell_{i+1}$  in the chain, with an inequality guard on all updated registers  $r_1, r_2, \dots, r_i$  so far, and an update on the next register  $r_{i+1}$ .

$$\ell_i \xrightarrow{\bigvee_{r \in \{r_1, \dots, r_i\}} (=r_i) \ a} \ell_i \quad \text{and} \quad \ell_i \xrightarrow{\bigwedge_{r \in \{r_1, \dots, r_i\}} (\neq r_i) \ a \ r_{i+1} \downarrow} \ell_{i+1},$$

$$\ell'_i \xrightarrow{\bigvee_{r \in \{r_1, \dots, r_i\}} (=r_i) \ a} \ell'_i \quad \text{and} \quad \ell'_i \xrightarrow{\bigwedge_{r \in \{r_1, \dots, r_i\}} (\neq r_i) \ a \ r_{i+1} \downarrow} \ell'_{i+1}.$$

At the last locations  $\ell_n$  and  $\ell'_n$  of the two chains, there is one transition with inequality guards on all registers leaving the chain to **synch**, and there is one transition which is, again, a self-loop with an equality constraint for at least one of the registers.

$$\ell_n \xrightarrow{\bigwedge_{r \in R} (\neq r_i) \ a \ R \downarrow} \mathbf{synch} \quad \text{and} \quad \ell_n \xrightarrow{\text{else } a} \ell_n \quad \ell'_n \xrightarrow{\bigwedge_{r \in R} (\neq r_i) \ a \ R \downarrow} \mathbf{synch} \quad \text{and} \quad \ell'_n \xrightarrow{\text{else } a} \ell'_n.$$

By construction, we see that  $n + 1$  distinct data values must be read for reaching **synch** from the infinite set  $\{\text{init}\} \times D^n$ . Since  $\mathcal{R}_n$  can only be synchronized in **synch**, all synchronizing data words must have data efficiency at least  $n + 1 \in \Omega(n)$ .

It remains to prove that  $\mathcal{R}_n$  has indeed some synchronizing word. Let  $\{x_1, x_2, \dots, x_{n+1}\}$  be a set of  $n + 1$  distinct data values and  $w_{\text{synch}} = (a, x_1)(a, x_2) \cdots (a, x_n)(a, x_{n+1})$ . For the configuration space  $L = \{\text{init}, \text{synch}, \ell_1, \dots, \ell_n, \ell'_1, \dots, \ell'_n\}$ , observe that  $\text{post}(L \times D^n, w_{\text{synch}}) = \{(\text{synch}, x_{n+1})\}$  and  $|\text{data}(w_{\text{synch}})| = n + 1$ . The proof is complete.  $\square$

**Theorem 5.** *The synchronization problem for  $k$ -DRAs is PSPACE-complete.*

*Proof.* (Sketch) The synchronization problem for  $k$ -DRA is in PSPACE using the following co-(N)PSPACE algorithm: (1) pick a set  $X = \{x_1, x_2, \dots, x_{2k+1}\}$  of distinct data values. (2) guess some location  $\ell \in L$  and check if there is no word  $w \in (\Sigma \times \{x_1, x_2, \dots, x_k\})^*$  with length  $|w| \leq 2^{k|L||\Sigma|}$  such that along firing transitions that are inequality-guarded on all  $k$  registers, some registers are not updated. If (2) is satisfied, then return “no” (meaning that there is no synchronizing data word for the input  $k$ -DRA). Otherwise, (3) guess two configurations  $q_1, q_2 \in L \times X^k$  such that there is no word  $w \in (\Sigma \times X)^*$  with length  $|w| \leq 2^{(2k+1)|L||\Sigma|}$  such that  $|\text{post}(\{q_1, q_2\}, w)| = 1$ . If (3) is satisfied, then the algorithm returns “no”; otherwise return “yes”.

For PSPACE-hardness, we adapt an established reduction (see, e.g., [14]) from the non-emptiness problem for  $k$ -DRA, see Appendix 6. The result then follows by PSPACE-completeness of the non-emptiness problem for  $k$ -DRA [11].  $\square$

## 4 Synchronizing data words for NRAs

In this section, we study the synchronization problems for NRAs. We slightly update a result in [14] to present a general reduction from the *non-universality* problem to the synchronization problem for NRAs. This reduction proves the *undecidability* result for the synchronization problem for  $k$ -NRAs, and Ackermann-hardness in 1-NRAs. We then prove that for 1-NRAs, the synchronization and non-universality problems are indeed interreducible, which completes the picture by Ackermann-completeness of the synchronization problem for 1-NRAs.

In the nondeterministic synchronization setting, we present two kinds of *counting features*, which are useful for later constructions. For the first one, we define a family  $(\mathcal{R}_{\text{counter}(n)})_{n \in \mathbb{N}}$  of 1-NRAs with size only linear in  $n$ , where an input datum  $x \in D$  must be read  $2^n$  times to achieve synchronization.

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

*Proof.* (Sketch) The 1-NRA  $\mathcal{R}_{\text{counter}(n)}$  shown in Figure 4 encodes a binary counter that ensures that in every synchronizing data word  $w$  some datum  $x \in \text{data}(w)$  appears at least  $2^n$  times. The location **synch** has self-loops on all letters, thus,  $\mathcal{R}_{\text{counter}(n)}$  can only be synchronized in location **synch**. Generally speaking, the counting involves an *initializing process* and several *incrementing processes*. The *initializing process* is started by firing a  $\star$ -transition, which places a token, let us say: an  $x$ -token, into location **zero**. This sets the counter to 0. Note that firing  $\star$ -transitions is the only way to guide tokens out of **reset**; hence, whenever there is some token in **reset**, a new initializing process must be started. We use this to enforce a new initializing process whenever some transition is fired that is incorrect with respect to the incrementing process.



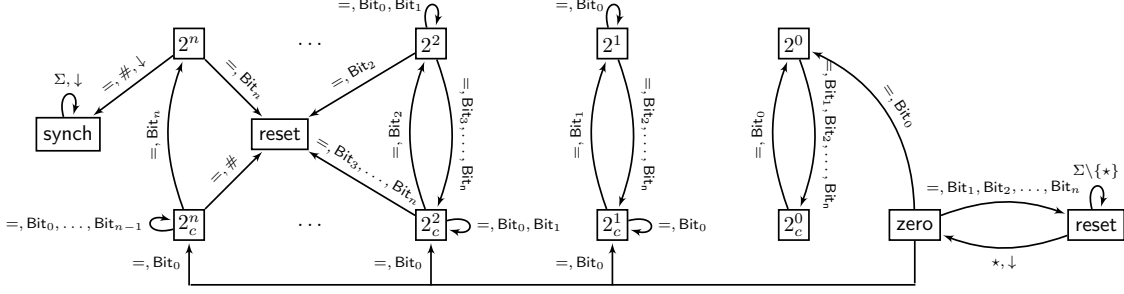


Figure 4: A partial picture of the 1-NRA  $\mathcal{R}_{\text{counter}(n)}$  (with  $n \geq 3$ ) implementing a binary counter. In order to avoid crossing edges in the figure, we use two copies of the same location `reset`. All locations have inequality-guarded self-loops for all letters in  $\Sigma \setminus \{\star\}$ . All missing equality-guarded  $\star$ -transitions are directed to `zero`. For all  $0 \leq i < n$ , missing equality-guarded  $\#$ -transitions from  $2^i$  are guided to `synch` with an update on the register. All other non-depicted equality-guarded transitions are directed to `reset`, and inequality-guarded transitions are self-loops.

An *incrementing process* can be set off by inputting the datum  $x$  via equality guards. The numbers  $1 \leq m \leq 2^n$  are represented by placing a copy of the  $x$ -token in the locations corresponding to the binary representation of  $m$ . An  $x$ -token in location  $2^i$  (in  $2_c^i$ , respectively) means that the  $i$ -th least significant in the binary representation is set to 1 (to 0, respectively). First, a  $\text{Bit}_0$ -transition places a copy of the  $x$ -token in each of  $\{2_c^n, \dots, 2_c^2, 2_c^1, 2_c^0\}$  to represent  $0 \dots 001$ . In each incrementation step the  $x$ -tokens are re-placed by firing specific  $\text{Bit}_i$ -transitions ( $0 \leq i \leq n$ ), following the standard procedure of binary incrementation. At the end, when a copy of the  $x$ -token locates in each of  $\{2^n, 2^{n-1}, \dots, 2^0\}$  (representing  $10 \dots 0$ ), the  $\#$ -transitions guide all of these tokens to location `synch` and finally synchronize  $\mathcal{R}_{\text{counter}(n)}$ . We give a detailed explanation of the structure of  $\mathcal{R}_{\text{counter}(n)}$  in Appendix 7.  $\square$

We present a second kind of counting features in RAs that explains the hardness of synchronizing NRAs, even with a single register. In Lemma 7, we define a family of 1-NRAs (with only  $\mathcal{O}(n)$  locations), where  $\text{tower}(n)$  distinct data must be read to gain synchronization. Recall from [27] that the function  $\text{tower}$  is at level three of the infinite *Ackermann hierarchy*  $(A_k)_{k \in \mathbb{N}}$  of fast-growing functions  $A_i : \mathbb{N} \rightarrow \mathbb{N}$ , inductively defined by  $A_1(n) = 2n$  and  $A_{k+1}(n) = A_k^n(1) = \underbrace{A_k(\dots(A_k(n))\dots)}_{n \text{ times}}$ . Hence, applying

$\text{doub} \stackrel{\text{def}}{=} A_1$ ,  $\text{exp} \stackrel{\text{def}}{=} A_2$ , and  $\text{tower} \stackrel{\text{def}}{=} A_3$ , respectively, on some natural number  $n$  results in some number that is *double*, *exponential*, and *tower*, respectively, in  $n$ . The function  $A_\omega(n) = A_n(n)$  is a non-primitive recursive Ackermann-like function, defined by diagonalization.

**Lemma 7.** *There is a family of 1-NRAs  $(\mathcal{R}_{\text{tower}(n)})_{n \in \mathbb{N}}$  with  $\mathcal{O}(n)$  locations, such that  $|\text{data}(w)| \geq \text{tower}(n)$  for all synchronizing data words  $w$ .*

*Proof.* The domain of the family of 1-NRAs  $(\mathcal{R}_{\text{tower}(n)})_{n \in \mathbb{N}}$  is the natural numbers  $\mathbb{N}$ . The alphabet of  $\mathcal{R}_{\text{tower}(n)}$  is  $\Sigma = \{\#, \star, \text{rep}, \text{doub}, \text{exp}, \text{tow}\}$ . The structure of  $\mathcal{R}_{\text{tower}(n)}$  is composed of  $n$  locations  $\text{data}_1, \text{data}_{1,2}, \dots, \text{data}_{1,2,\dots,n}$  and 6 more locations `reset`, `synch`, `store`, `rep`, `waitDoub`, `waitExp`. The general structure of  $\mathcal{R}_{\text{tower}(n)}$  is partially depicted in Figure 5. The NRA  $\mathcal{R}_{\text{tower}(n)}$  is such that  $|\text{data}(w)| \geq \text{tower}(n)$  for all synchronizing data words  $w$ .

All transitions in `synch` are self-loops with an update on the register `synch`  $\xrightarrow{\Sigma \ r \downarrow}$  `synch`; thus,  $\mathcal{R}_{\text{tower}(n)}$  can only be synchronized in `synch`. Moreover, `synch` is only accessible from `store` by a  $\#$ -transition. Assuming  $w$  is one of the shortest synchronizing words, we see that  $\text{post}(L \times D, w) = \{(\text{synch}, x)\}$ , where  $w$  ends with  $(\#, x)$ .

From all locations  $\ell \in L \setminus \{\text{synch}\}$ , we have  $\ell \xrightarrow{\star \ r \downarrow}$  `data`<sub>1</sub>; we say that  $\star$ -transitions *reset*  $\mathcal{R}_{\text{tower}(n)}$ . Moreover, the only outgoing transition in location `reset` is the  $\star$ -transition. Thus, a *reset* must occur in order to synchronize  $\mathcal{R}_{\text{tower}(n)}$ . After this forced reset, say on reading  $(\star, 1)$ , the set of reached

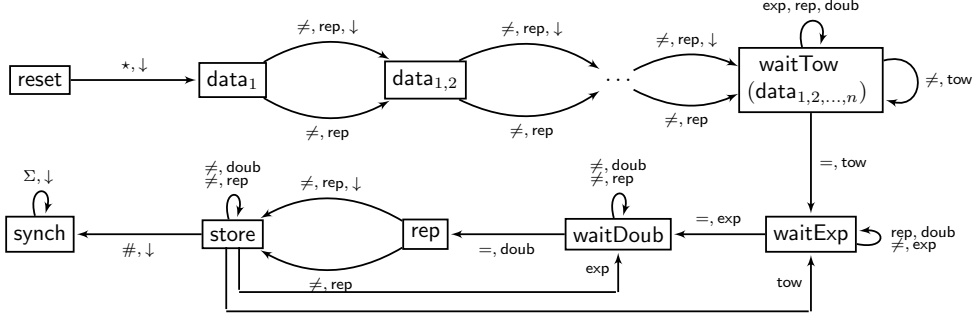


Figure 5: A partial illustration of the 1-NRA  $\mathcal{R}_{\text{tower}(n)}$  for  $n \geq 3$ . All  $\star$ -transitions are guided to  $\text{data}_1$  with an update on the register. All other missing non-depicted transitions are directed to  $\text{reset}$ .

configurations is  $\{(\text{data}_1, 1), (\text{synch}, 1)\}$ . Since resetting is inefficient, we try to avoid it; we call all transitions leading to  $\text{reset}$  *inefficient*.

For all locations  $\text{data}_{1,\dots,i}$  with  $1 \leq i < n$ , we define the two transitions

$$\text{data}_{1,\dots,i} \xrightarrow{\neq r \text{ rep}} \text{data}_{1,\dots,i+1} \quad \text{and} \quad \text{data}_{1,\dots,i} \xrightarrow{\neq r \text{ rep } r\downarrow} \text{data}_{1,\dots,i+1}.$$

All other transitions in  $\text{data}_{1,\dots,i}$  are inefficient and directed to  $\text{reset}$ . Below, we rename  $\text{data}_{1,2,\dots,n}$  to  $\text{waitTow}$ . We partially depict the transitions from  $\text{waitTow}$ ,  $\text{waitExp}$ ,  $\text{waitDoub}$ ,  $\text{rep}$  and  $\text{store}$  in Figure 5. All transitions are inefficient, except

- $\text{waitTow} \xrightarrow{=r \text{ tow}} \text{waitExp}$ ,  $\text{waitTow} \xrightarrow{\neq r \text{ tow}} \text{waitTow}$ , and  $\text{waitTow} \xrightarrow{\sigma} \text{waitTow}$  for all  $\sigma \in \{\text{doub}, \text{exp}, \text{rep}\}$ .
- $\text{waitExp} \xrightarrow{=r \text{ exp}} \text{waitDoub}$ ,  $\text{waitExp} \xrightarrow{\text{doub}} \text{waitExp}$  and  $\text{waitExp} \xrightarrow{\text{rep}} \text{waitExp}$ .
- $\text{waitDoub} \xrightarrow{=r \text{ doub}} \text{rep}$ ,  $\text{waitDoub} \xrightarrow{\neq r \text{ doub}} \text{waitDoub}$  and  $\text{waitDoub} \xrightarrow{\neq r \text{ rep}} \text{waitDoub}$ ,
- $\text{rep} \xrightarrow{\neq r \text{ rep}} \text{store}$  and  $\text{rep} \xrightarrow{\neq r \text{ rep } r\downarrow} \text{store}$ ,
- $\text{store} \xrightarrow{\text{tow}} \text{waitExp}$ ,  $\text{store} \xrightarrow{\text{exp}} \text{waitDoub}$ ,  $\text{store} \xrightarrow{\neq r \text{ doub}} \text{store}$  and  $\text{store} \xrightarrow{\neq r \text{ rep}} \text{store}$ , and
- $\text{store} \xrightarrow{\# \text{ } r\downarrow} \text{synch}$ .

We remark that  $\text{store} \xrightarrow{\# \text{ } r\downarrow} \text{synch}$  is the only  $\#$ -transition that is not inefficient. This implies that for efficiently synchronizing  $\mathcal{R}_{\text{tower}(n)}$ , one needs to re-move all produced tokens to  $\text{store}$  before firing a  $\#$ -transition. The main issue in re-moving produced tokens, however, is that some inequality-guarded transitions are unavoidable, and these transitions may *replicate* the tokens. For example, if one token is in  $\text{data}_1$ , firing two transitions  $\text{data}_1 \xrightarrow{\neq r \text{ rep}} \text{data}_{1,2}$  and  $\text{data}_1 \xrightarrow{\neq r \text{ rep } r\downarrow} \text{data}_{1,2}$  replicates one token to two tokens in  $\text{data}_{1,2}$ . Using this, one can implement *doubling*, *exponentialization*, and *towering* of distinct tokens, as explained in the following.

*Doubling:* Assume that there are  $n$  distinct tokens  $\{1, 2, \dots, n\}$  in  $\text{waitDoub}$ . Then the only efficient transition is  $\text{waitDoub} \xrightarrow{=r \text{ doub}} \text{waitRep}$ . In particular, all  $\{\#, \text{exp}, \text{tow}\}$ -transitions activate a reset. As a result, as long as some token is in  $\text{waitDoub}$ ,  $\{\#, \text{exp}, \text{tow}\}$ -transitions should be avoided for the sake of efficiency. This implies that for all  $1 \leq i \leq n$ , the  $i$ -token in  $\text{waitDoub}$  can leave the location only individually on the input  $(\text{doub}, i)$ . Now, inputting  $(\text{doub}, i)$  moves the  $i$ -token to  $\text{waitRep}$ . Here the  $i$ -token must immediately move on to  $\text{store}$  via the inequality-guarded  $\text{rep}$ -transitions, which will replicate the  $i$ -token into two tokens. Note that we must fire  $\text{rep}$ -transitions with some “fresh” datum  $j$  such that  $j \notin \{1, \dots, n\}$ , otherwise a reset is evoked. (For simplicity, we use  $j = i + n$  by convention.) It can now

be easily seen that the only efficient way to guide all  $n$  tokens out of `waitDoub` is by inputting the data word

$$w_{\text{doub}(n)} = (\text{doub}, 1)(\text{rep}, n + 1)(\text{doub}, 2)(\text{rep}, n + 2) \dots (\text{doub}, n)(\text{rep}, 2n),$$

which puts  $2n$  distinct tokens into `store`.

*Exponentialization:* Assume there are  $n$  distinct tokens  $\{1, 2, \dots, n\}$  in `waitExp`. The only efficient transition is `waitExp`  $\xrightarrow{=r \text{ exp}}$  `waitDoub`. In particular, all  $\{\#, \text{tow}\}$ -transitions activate a reset, and should be avoided as long as some token is in `waitExp`. This implies that for all  $1 \leq i \leq n$ , the  $i$ -token in `waitExp` can leave the location only individually on the input  $(\text{exp}, i)$ . Now, inputting  $(\text{exp}, 1)$  moves the 1-token to `waitDoub`. From above we know that the only efficient way for guiding a single token in `waitDoub` towards synchronization is by inputting the data word  $w_{\text{doub}(1)}$ , resulting in two distinct tokens in `store`: 1 and 2. We can now proceed to remove the 2-token from `waitExp` by inputting  $(\text{exp}, 2)$ . Note that this also guides the  $\{1, 2\}$ -tokens residing in `store` to `waitDoub`. Again, for efficient synchronization, we must input the data word  $w_{\text{doub}(2)}$ , which results in four distinct tokens  $\{1, 2, 3, 4\}$  in `store`. It is now easy to see that the only efficient way to guide all  $n$  tokens out of `waitExp` is by inputting the data word

$$w_{\text{exp}(n)} = (\text{exp}, 1) \cdot w_{\text{doub}(1)} \cdot (\text{exp}, 2) \cdot w_{\text{doub}(2)} \cdot (\text{exp}, 3) \cdot w_{\text{doub}(4)} \cdot \dots \cdot (\text{exp}, n) \cdot w_{\text{doub}(2^{n-1})},$$

which puts  $2^n$  distinct tokens into `store`.

*Towering:* Assume there are  $n$  distinct tokens  $\{1, 2, \dots, n\}$  in `waitTow`. The only efficient transition is `waitExp`  $\xrightarrow{=r \text{ tow}}$  `waitExp`. In particular, firing  $\#$ -transitions activates a reset, and should be avoided as long as some token is in `waitTow`. This implies that for all  $1 \leq i \leq n$ , the  $i$ -token in `waitTow` can leave the location only individually on the input  $(\text{tow}, i)$ . Now, inputting  $(\text{exp}, 1)$  moves the 1-token to `waitExp`. From above we know that the only efficient way for guiding a single token in `waitTow` towards synchronization is by inputting the data word  $w_{\text{exp}(1)}$ , resulting in two distinct tokens in `store`: 1 and 2. We can now proceed to remove the 2-token from `waitTow` by inputting  $(\text{tow}, 2)$ . Note that this also guides the  $\{1, 2\}$ -tokens residing in `store` to `waitExp`. Again, for efficient synchronization, we must input the data word  $w_{\text{exp}(2)}$ , which results in four distinct tokens  $\{1, 2, 3, 4\}$  in `store`. It is now easy to see that the only efficient way to guide all  $n$  tokens out of `waitTow` is by inputting the data word

$$w_{\text{tow}(n)} = (\text{tow}, 1) \cdot w_{\text{exp}(1)} \cdot (\text{tow}, 2) \cdot w_{\text{exp}(2)} \cdot (\text{tow}, 3) \cdot w_{\text{exp}(4)} \cdot \dots \cdot (\text{tow}, n) \cdot w_{\text{exp}(\text{tower}(n-1))},$$

which puts  $\text{tower}(n)$  distinct tokens into `store`.

Now, after the (forced) initial reset by firing  $\star$ -transitions, it is easy to see that the only data word that advances in synchronizing is  $(\text{rep}, 2)(\text{rep}, 3) \dots (\text{rep}, n)$ . It replicates the 1-token to  $n$  distinct tokens  $1, 2, \dots, n$ , which are placed into `waitTow`. From above we know that the only efficient way to guide all  $n$  tokens out of `waitTow` is by inputting  $w_{\text{tow}(n)}$ , which places  $\text{tower}(n)$  distinct tokens into `store`. We can now fire  $\#$ -transitions to synchronize  $\mathcal{R}_{\text{tower}(n)}$  without evoking a reset, but note that due to the equality guard at the  $\#$ -transition from `store` to `synch`, each of the  $\text{tower}(n)$  distinct tokens in `store` can move to `synch` only individually. This implies  $|\text{data}(w)| \geq \text{tower}(n)$  for all synchronizing words  $w$ .  $\square$

We can now use similar ideas as in Lemma 7 for defining a family of 1-NRAs  $\mathcal{R}_{A_n(m)}$  ( $n, m \in \mathbb{N}$ ) such that all synchronizing data words of  $\mathcal{R}_{A_n(m)}$  have data efficiency at least  $A_n(m)$ , where  $A_n$  is at level  $n$  of the Ackermann hierarchy. This provides a good intuition that the synchronization problem for NRAs must be Ackermann-hard, even if the NRA has a single register. In the following, we prove that the synchronization problem and the non-universality problem for NRAs are interreducible.

Let us first define the non-universality problem for RAs. To define the language of a given NRA  $\mathcal{R}$ , we equip it with an initial location  $\ell_{\text{in}}$  and a set  $L_f$  of accepting locations, where, without loss of generality, we assume that all outgoing transitions from  $\ell_{\text{in}}$  update all registers. The language  $L(\mathcal{R})$  is the set of all data words  $w \in (\Sigma \times D)^*$ , for which there is a run from  $(\ell_{\text{in}}, \nu_{\text{in}})$  to  $(\ell_f, \nu_f)$  such that  $\ell_f \in L_f$  and  $\nu_{\text{in}}, \nu_f \in D^{|R|}$ . The non-universality problem asks, given an RA, whether there exists some data word  $w$  over  $\Sigma$  such that  $w \notin L(\mathcal{R})$ . We adopt an established reduction in [14] to provide the following Lemma.

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

The detailed proof can be found in Appendix 7. As an immediate result of Lemma 8 and the undecidability of the non-universality problem for NRAs (Theorems 2.7 and 5.4 in [11]), we obtain the following theorem.

**Theorem 9.** *The synchronization problem for NRAs is undecidable.*

Next, we present a reduction showing that, for 1-NRAs, the synchronization problem is reducible to the non-universality problem, providing the tight complexity bounds for the synchronizing problem.

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

*Proof.* We establish a reduction from the synchronization problem to the non-universality problem for 1-NRAs as follows. Given a 1-NRA  $\mathcal{R} = \langle L, R, \Sigma, T \rangle$ , we construct a 1-NRA  $\mathcal{R}_{\text{comp}}$  equipped with an initial location and a set of accepting locations such that  $\mathcal{R}$  has some synchronizing word if, and only if, there exists some data word that is not in  $L(\mathcal{R}_{\text{comp}})$ .

First, we see that an analogue of Lemma 1 holds for 1-NRAs: for all 1-NRAs with some synchronizing data word, there exists some word  $w$  with data efficiency 1 such that  $\text{post}(L \times D, w) \subseteq L \times \text{data}(w)$ . For all locations  $\ell \in L$ , such a data word must update the register by firing an inequality-guarded transition that is reached only via inequality-guarded transitions; this can be checked in NLOGSPACE. Given  $\mathcal{R}$ , we assume that such a data word  $w$  always exists; otherwise, we define  $\mathcal{R}_{\text{comp}}$  to be a 1-NRA with a single (initial and accepting) location equipped with self-loops for all letters, so that  $L(\mathcal{R}_{\text{comp}}) = (\Sigma \times D)^*$ . Given  $\text{data}(w) = \{x\}$ , we say that  $\mathcal{R}$  has some synchronizing word  $v$  if  $\text{post}(L \times \{x\}, v)$  is a singleton.

Second, we define a data language  $\text{lang}$  such that data words in this language are encodings of the synchronizing process. Let  $L = \{\ell_1, \ell_2, \dots, \ell_n\}$  be the set of locations and  $x, y$  two distinct data. Informally, each data word in  $\text{lang}$  starts with the

- *initial block*: a delimiter  $(\star, y)$ , the sequence  $(\ell_1, x), (\ell_2, x), \dots, (\ell_n, x)$  and an input  $(a, d) \in \Sigma \times D$  as the beginning of a synchronizing word. The initial block is followed by several
- *normal blocks*: the delimiter  $(\star, y)$ , the set of successor configurations reached from the configurations and the input of the previous block, and the next input  $(a', d')$  of the synchronizing data word. The data word finally ends with the
- *final block*: the delimiter  $(\star, y)$ , a single successor configuration reached from the configurations and the input of the previous block, and the delimiter  $(\star, y)$ .

Formally, the language  $\text{lang}$  is defined over the alphabet  $\Sigma_{\text{lang}} = \Sigma \cup L \cup \{\star\}$  where  $\star \notin \Sigma \cup L$ . It contains all data words  $u$  that satisfy the following *membership conditions*:

1. The data words  $u$  starts with  $(\star, y)(\ell_1, x), (\ell_2, x), \dots, (\ell_n, x)$  for some  $x, y \in D$  with  $y \neq x$ ; this condition guarantees the correctness of the encoding for the initial block.
2. Let  $\text{proj}(u)$  be the projection of  $u$  into  $\Sigma_{\text{lang}}$  (*i.e.*, omitting the data values). Then there exists some  $\ell_{\text{synch}} \in L$  where  $\text{proj}(u) \in (\star L^+ \Sigma)^+ \star \ell_{\text{synch}} \star$ . This condition guarantees the right form of data words to be encodings of synchronizing processes.

The next two conditions guarantee the uniqueness of the delimiter:

3. The letter  $\star$  in  $u$  occurs only with datum  $y$ .
4. No other letter in  $u$  occurs with datum  $y$ .

The next three conditions guarantee that all the successors that can be reached from configurations and inputs in each block are correctly inserted in the next block. For all  $(\ell, x) \in L \times D$  and  $(a, d) \in \Sigma \times D$  in the same block,

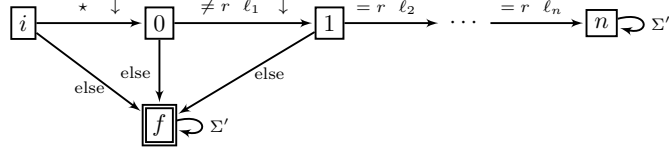
5. if  $x = d$  and there exists a transition  $\ell \xrightarrow{=r, a} \ell'$  (with or without update), then  $(\ell', x)$  must be in the next block.

6. if  $x \neq d$  and there exists a transition  $\ell \xrightarrow{\neq r a} \ell'$ , then  $(\ell', x)$  must be in the next block.
7. if  $x \neq d$  and there exists a transition  $\ell \xrightarrow{\neq r a r \downarrow} \ell'$  then  $(\ell', d)$  must be in the next block.

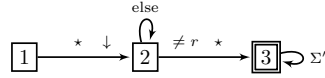
By construction, the NRA  $\mathcal{R}$  has some synchronizing data word if, and only if,  $\text{lang} \neq \emptyset$ . Below, we construct a 1-NRA  $\mathcal{R}_{\text{comp}}$  that accepts the complement of  $\text{lang}$ . Then, the NRA  $\mathcal{R}$  has some synchronizing data word if, and only if, there exists some data word that is not in  $L(\mathcal{R}_{\text{comp}})$ .

The 1-NRA  $\mathcal{R}_{\text{comp}}$  is the union of several 1-NRAs that are in the family of 1-NRAs  $\mathbf{R}_1, \mathbf{R}_2, \dots, \mathbf{R}_7$ , where an 1-NRA is in the family  $\mathbf{R}_i$  if it violates the  $i$ -th condition among the membership conditions in  $\text{lang}$ .

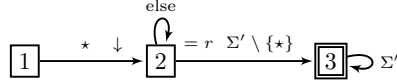
1. Family  $\mathbf{R}_1$ : we add a 1-NRA that accepts data words not starting with  $(\star, y)(\ell_1, x), \dots, (\ell_n, x)$ .



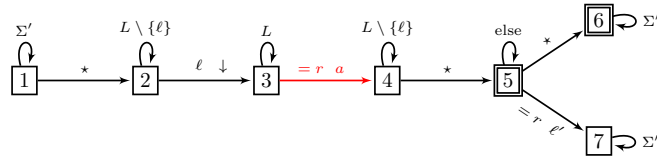
2. Family  $\mathbf{R}_2$ : we add a DFA that accepts data words  $u$  such that  $\text{proj}(u)$  is not in the regular language  $(\star L^+ \Sigma)^+ \star \ell_{\text{synch}} \star$ .
3. Family  $\mathbf{R}_3$ : we add a 1-NRA that accepts data words in which two delimiters  $\star$  have different data.



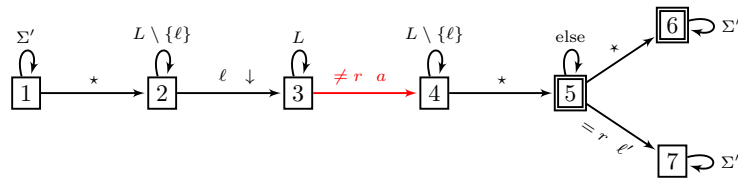
4. Family  $\mathbf{R}_4$ : we add a 1-NRA that accepts data words in which the datum of first  $\star$  is not used only by occurrences of  $\star$ .



5. Family  $\mathbf{R}_5$ : for all transitions  $\ell \xrightarrow{=r a} \ell'$ , we add a 1-NRA that only accepts data words such that one block contains some  $(\ell, x)$  and  $(a, d)$  with  $x = d$  where the next block does not have  $(\ell', x)$ .



6. Family  $\mathbf{R}_6$ : for all transitions  $\ell \xrightarrow{\neq r a} \ell'$ , we add a 1-NRA that only accepts data words such that one block contains some  $(\ell, x)$  and  $(a, d)$  with  $x \neq d$  where the next block does not have  $(\ell', x)$ .



7. Family  $\mathbf{R}_7$ : for all transitions  $\ell \xrightarrow{\neq r a r \downarrow} \ell'$ , we add a 1-NRA that only accepts data words such that one block contains some  $(\ell, x)$  and  $(a, d)$  with  $x \neq d$  where the next block does not have  $(\ell', d)$ .

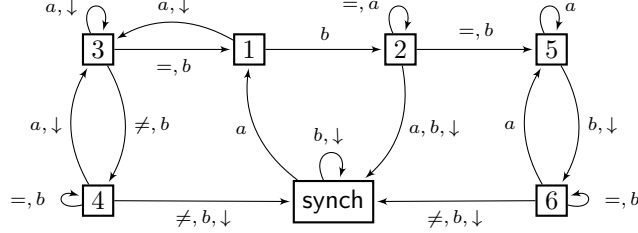
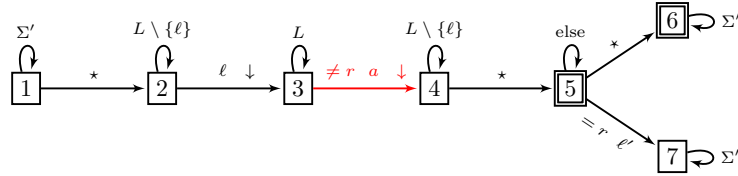


Figure 6: An RA with synchronizing data word  $(a, x)(b, y)(b, z)$  with three distinct data values  $x, y, z$ . The approach of using a unique data value to shrink the infinite set of configurations to a finite subset only yields synchronizing data words of length greater than 3.



The proof is complete.  $\square$

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 [11], and the result for counter automata with incrementing errors in [18], we obtain the following theorem.

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

## 5 Length-Bounded synchronizing data words for NRAs

As proved in the previous section, the synchronization problem for NRAs is in general undecidable. In this section, we study the length-bounded synchronization problem for NRAs, in which the synchronizing data words are required to be shorter than a given length (written in binary).

To decide the synchronization problem in 1-RAs, both in the deterministic and nondeterministic setting, we rely on Lemma 1. With this lemma at hand, it was sufficient to search for synchronizing data words that first input a single datum  $x$  (chosen arbitrary) as many times as necessary to have the set of successor configurations included in  $L \times \{x\}$ . In the next step, this obtained set of successor configurations was synchronized in a singleton. However, the shortest synchronizing data words do not always follow this pattern, for an example see Figure 6. Observe that the data word  $(a, x)(b, y)(b, z)$  is synchronizing with length 3 (not exceeding the bound 3). However, all synchronizing data words that repeat a datum such as  $x$ , to first bring the RA to a finite set, have length at least 4. The example shows that one cannot rely on the techniques developed in Section 4 to decide the length-bounded synchronization problem for NRA.

In this section, we prove

**Theorem 12.** *The length-bounded synchronization problem for NRAs is NEXPTIME-complete.*

The NEXPTIME-membership of the length-bounded synchronization problem is straightforward: guess a data word  $w$  shorter than the given length (that is written in binary and thus may be exponential in the length) and check in EXPTIME whether  $w$  is synchronizing. Our main contribution is to prove the NEXPTIME-hardness of this problem, for which in turn, by Lemma 8, it is sufficient to show that the *length-bounded universality* problem is co-NEXPTIME-complete. The length-bounded universality problem asks, given an RA and  $N \in \mathbb{N}$  encoded in binary, whether all data words  $w$  with  $|w| \leq N$  are in the language of the automaton.

**Theorem 13.** *The length-bounded universality problem for NRAs is co-NEXPTIME-complete.*

*Proof.* The length-bounded universality problem for NRAs can be solved in co-NEXPTIME, by guessing a (possibly exponentially long) data word, and check whether the guessed word is a witness for non-universality of the RA.

We prove that the complement of the length-bounded universality problem is NEXPTIME-hard. The proof is a reduction from the *membership problem of  $\mathcal{O}(2^n)$ -time bounded nondeterministic Turing machines*: given a nondeterministic Turing machine  $\mathcal{M}$  and an input word  $x$ , decide whether  $\mathcal{M}$  accepts  $x$  within time bound  $2^{|x|}$ . This problem is NEXPTIME-complete.

Given a nondeterministic Turing machine  $\mathcal{M}$  and an input  $x$  of length  $n$ , we construct an NRA  $\mathcal{R}$  equipped with an initial location and a set of accepting locations, and a bound  $N$  (encoded in binary) such that there exists a witness of non-universality  $w$  (i.e.,  $w \notin L(\mathcal{R})$ ) with  $|w| \leq N$  if, and only if,  $\mathcal{M}$  has some accepting computation on  $x$  within time bound  $2^n$ .

Let  $\mathcal{M}$  have the set  $Q$  of control states and the tape alphabet  $\Gamma$ . Let us recall that a configuration of  $\mathcal{M}$  is a word in the language  $\Gamma^*(Q \times \Gamma)\Gamma^*$ , where each letter in  $(Q \times \Gamma) \cup \Gamma$  encodes a single cell and the position of the reading/writing head. A computation  $\rho$  of  $\mathcal{M}$  is a sequence  $c_0 c_1 c_2 \dots$  of configurations that respects the transition function of the Turing machine. Without loss of generality, we assume that  $\mathcal{M}$  has a self-loop on all accepting states. Hence for the input  $x \in \Gamma^*$  of length  $n$ , all accepting computations  $\rho$  of  $\mathcal{M}$  are sequences of length exactly  $2^n$ , and all configurations  $c_i$  along such a computation are words  $c_i \in \Gamma^*(Q \times \Gamma)\Gamma^*$  of length at most  $2^n$ . In the following, we pad the configurations shorter than  $2^n$  with  $\square$  at the tail such that the length of all such configurations become equal to  $2^n$ .

Let  $\Sigma_{\mathcal{M}} := \Sigma \cup \Sigma'$ , where

$$\Sigma = (Q \times \Gamma) \cup \Gamma \cup (Q \overset{\cdot}{\times} \Gamma) \cup \overset{\cdot}{\Gamma} \cup \{\square, \overset{\cdot}{\square}, \#, \star\}$$

be such that  $\square, \overset{\cdot}{\square}, \#, \star \notin \Gamma$ . Here,  $(Q \overset{\cdot}{\times} \Gamma)$  and  $\overset{\cdot}{\Gamma}$  denote a *dotted* version of letters in  $Q \times \Gamma$  and  $\Gamma$ ; formally

$$\{(q, \overset{\cdot}{a}) \mid (q, a) \in (Q \times \Gamma)\} \quad \text{and} \quad \{\overset{\cdot}{a} \mid a \in \Gamma\},$$

and  $\Sigma'$  will be defined later. Let  $K = 2^{3n} + 2^{2n} + 1$ . Given a computation  $\rho = c_1 \dots c_{2^n}$ , we define  $u(\rho) \in \Sigma^K$ , roughly speaking, such that

1. It consists of  $2^n$  copies of  $\rho$  (with some extra delimiters).
2. Between all consecutive copies of  $\rho$  there is a  $\star$  delimiter, and  $u(\rho)$  starts and ends with  $\star$ , too. Hence, there are  $2^n + 1$  occurrences of  $\star$  in  $u(\rho)$ .
3. In each copy of  $\rho$ , there is a  $\#$  delimiter between consecutive configurations. Since there are  $2^n$  configurations in (each copy of)  $\rho$ , the number of  $\#$  in  $u(\rho)$  is  $2^n(2^n - 1)$ .
4. In the  $i$ -th copy of  $\rho$ , the letter for the  $i$ -th cell of every participating configuration  $c_i$  is dotted, all other letters are non-dotted. Hence, in each copy of  $\rho$  there are exactly  $2^n$  dotted letters (one in each configuration of  $\rho$ ), with distance  $2^n + 1$ .
5. The distance between two  $\star$  delimiters is  $2^{2n} + 2^n - 1$ , due to the fact that  $\rho$  consists of  $2^n$  configurations, each of which has  $2^n$  tape cells in turn and is separated from the next configuration by a  $\#$  delimiter.

Figure 7 illustrates an example of  $u(\rho)$ . Observe that for all  $\rho = c_0 c_1 \dots c_{2^n}$ , we have

$$|u(\rho)| = \underbrace{2^n}_{\text{number of copies of } \rho} \underbrace{2^n}_{\text{number of configurations } c_i \text{ in } \rho} \underbrace{2^n}_{\text{length of } c_i} + \underbrace{2^n(2^n - 1)}_{\text{number of } \#} + \underbrace{(2^n + 1)}_{\text{number of } \star} = K.$$

We define a data language **lang** over the alphabet  $\Sigma$  such that data words in this language are faithful encodings of computations  $\rho$  of  $\mathcal{M}$  over the input word  $x$ . In particular, the language contains all data words  $v$  that satisfy the following conditions:

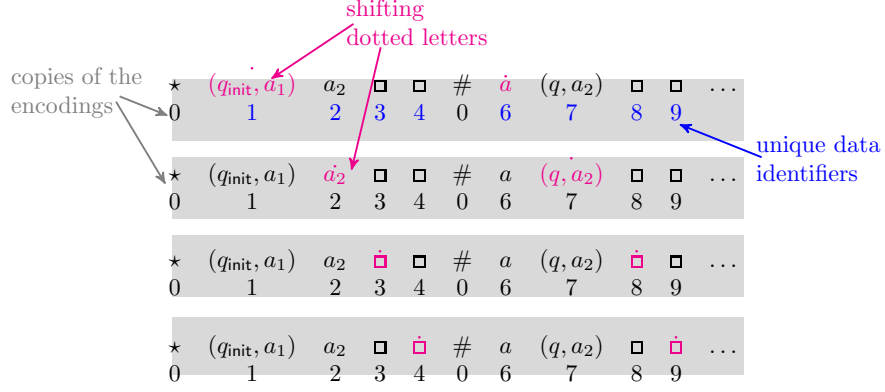


Figure 7: Partial encoding of  $u(\rho)$  for an accepting  $2^2$ -time bounded computation  $\rho$  of a Turing machine on  $a_1a_2$ .

6. Let  $\text{proj}(v)$  be the projection of  $v$  into  $\Sigma$  (i.e., omitting the data values). There exists some accepting computation  $\rho$  of  $\mathcal{M}$  on the input  $x$  such that  $\text{proj}(v) = u(\rho)$ .
7. The letters  $\star$  and  $\#$  occur only with a unique datum, say datum 0 (and no other letter occurs with that datum).
8. For all occurrences of  $\star$ , for all  $1 \leq i \leq 2^{2n} + 2^n - 1$ , all letters at the  $i$ -th positions after each  $\star$  must carry the same datum, say datum  $i$ . Except for occurrences of  $\#$ , the datum  $i$  is exclusive for the  $i$ -th positions after occurrences of  $\star$ .

Given a data word  $v \in \text{lang}$  such that  $\text{proj}(v) = u(\rho)$  for some computation  $\rho$ , condition (8) and previous conditions on  $u(\rho)$  entail that for all  $1 \leq j, k \leq 2^n$  the  $j$ -th tape cell in the  $k$ -th configuration  $c_k$  of all copies of  $\rho$  in  $v$  carries the same datum (revisit Figure 7). Observe that all data words  $v \in \text{lang}$  use exactly  $2^{2n} + 1$  distinct data values.

By definition of  $\text{lang}$ , we see that  $\text{lang}$  is non-empty if, and only if, there is an accepting computation  $\rho$  of  $\mathcal{M}$  over  $x$ . Recall that  $\Sigma_{\mathcal{M}} = \Sigma \cup \Sigma'$  (where  $\Sigma'$  is defined later). Below, we construct a 1-NRA  $\mathcal{R}$  over alphabet  $\Sigma_{\mathcal{M}}$  such that the language accepted by  $\mathcal{R}$  (projected into  $\Sigma$ , ignoring  $\Sigma'$  letters) is the complement of  $\text{lang}$ . At the end, we examine the existence of  $N \in \mathcal{O}(K)$  such that  $\mathcal{M}$  has an accepting computation over  $x$  if, and only if,  $\mathcal{R}$  is (length-bounded) non-universal with respect to the bound  $N$ .

The 1-NRA  $\mathcal{R}$  is the union of several 1-NRAs and DFAs that we describe in the following. Each of these automata violates one of the necessary conditions for data words  $v$  to be in  $\text{lang}$ .

- We add a DFA that accepts data words  $v$  such that  $\text{proj}(v)$  is not in the regular language  $(\star L)^*\star$  where  $L$  is defined by

$$\left( (\Gamma + \dot{\Gamma})^* \left( (Q \times \Gamma) + (Q \dot{\times} \Gamma) \right) (\Gamma + \dot{\Gamma})^* (\square + \dot{\square})^* \# \right)^*$$

- We add a DFA that accepts data words  $v$  such that  $\text{proj}(v)$  does not start with

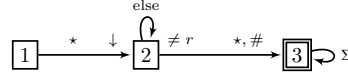
$$\star \left( (q_{\text{init}}, a_1) a_2 a_3 \dots a_n \square^* \# \right),$$

where  $q_{\text{init}}$  is the initial control state of  $\mathcal{M}$  and  $x = a_1a_2 \dots a_n$  is the input. This regular expression also guarantees that in the first copy of  $\rho$ , the first cell is dotted.

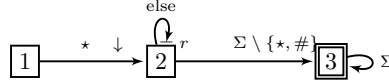
- We add a DFA that accepts data words  $w$  containing at least two dotted letters between two consecutive  $\#$ .



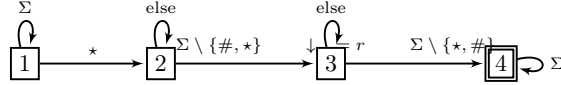
- We add a 1-NRA that accepts data words in which some delimiter occurs with some datum different from the datum for the first  $\star$ .



- We add a 1-NRA that accepts data words in which some other letter appears with the datum dedicated to delimiters  $\star$  and  $\#$ .



- We add 1-NRA that accepts data words in which there are two letters (other than  $\#$ ) between two consecutive  $\star$  that carry the same datum.



- We add a 1-NRA that accepts data words  $v$  such that there are two consecutive  $\#$  whose distance is not exactly  $2^n$  (ignoring the occurrences of  $\star$ ). For this we use a variant of  $\mathcal{R}_{\text{counter}(n)}$  implementing a binary counter introduced in Section 4. For accepting data words  $v$  such that the distance between two consecutive  $\#$  is *less than*  $2^n$ , we add a transition

$$2_c^n \xrightarrow{\#} \ell_f,$$

and for accepting those words that the distance is *more than*  $2^n$ , we add a transition

$$2^n \xrightarrow{\Sigma} \ell_f.$$

Here,  $\ell_f$  is an accepting location with a self-loop for every letter in  $\Sigma$ .

For the next four 1-NRAs we can use simple variants of  $\mathcal{R}_{\text{counter}(n)}$ :

- We add a 1-NRA that accepts data words  $v$  such that between two consecutive  $\star$ , the letter  $\#$  does not occur exactly  $2^n - 1$  times.
- We add a 1-NRA that accepts data words  $v$  such that  $\star$  does not occur exactly  $2^n + 1$  times.
- We add a 1-NRA that accepts data words  $v$  such that the distance between two consecutive dotted letters is not exactly  $2^n + 1$ , if no delimiter  $\star$  is seen between these two letters. We add another 1-NRA that accepts data words  $v$  such that the distance between two consecutive dotted letters is not exactly  $2^n + 2$  if  $\star$  is seen.
- We add a 1-NRA that accepts data words  $v$  such that the letters with  $2^{2n} + 2^n - 1$  distance carry different data.

To implement the above binary counters with 1-NRAs, we finally define

$$\Sigma' = \{\text{Bit}_i^d, \text{Bit}_i^\#, \text{Bit}_i^\star, \text{Bit}_i, \text{Bit}_i^x, \text{Bit}_{i+n}^x \mid 0 \leq i \leq n\},$$

where

- letters  $\text{Bit}_0^d, \dots, \text{Bit}_n^d$  for counting the distance between two consecutive  $\#$ . The counter takes into account only letters in  $\Sigma \setminus \{\star\}$ , ignoring the occurrences of  $\star$  and other  $\text{Bit}_i$ -letters from  $\Sigma'$ . The 1-NRA detects whether the distance is less or greater than  $2^n$ .

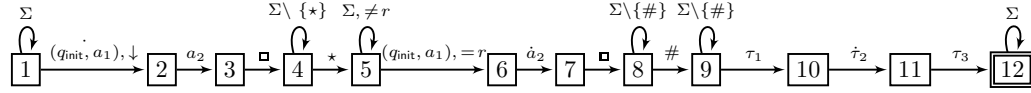
- letters  $\text{Bit}_0^\#, \dots, \text{Bit}_n^\#$  for counting the occurrences of  $\#$ . The 1-NRA detects whether the number of  $\#$  between two consecutive  $\star$  is less or greater than  $2^n - 1$ .
- letters  $\text{Bit}_0^\star, \dots, \text{Bit}_n^\star$  for counting the occurrences of  $\star$  (to check against  $2^n + 1$ ).
- letters  $\text{Bit}_0^\cdot, \dots, \text{Bit}_n^\cdot$  for counting the distance between two consecutive dotted letters (to check against  $2^n + 1$  or  $2^n + 2$ ).
- letters  $\text{Bit}_0^x, \dots, \text{Bit}_{2^n}^x$  for counting the distance between two letters that carry the same datum (to check against  $2^{2^n} + 2^n + 1$ ).

We construct all these gadgets such that the Bit-letters always carry the same datum as the delimiters.

The union of all above 1-NRAs and DFAs accepts all data words except those  $v$  such that  $\text{proj}(v) = (\star\rho)^{2^n} \star$  (that in addition respect the uniqueness conditions on data appearing in  $v$ ). Finally, we add NRAs that check whether  $\rho = c_1 \cdots c_{2^n}$  in such  $v$  is not a faithful computation of  $\mathcal{M}$ , or it is not an *accepting* computation. To this aim, for all words  $\sigma_1\sigma_2\sigma_3 \in ((Q \times \Gamma) \cup \Gamma)^3$  of length three such that  $\sigma_1\sigma_2\sigma_3$  can appear at some position  $i$  in a valid configuration  $c$  of  $\mathcal{M}$ , we define  $\text{Post}(\sigma_1\sigma_2\sigma_3)$  to be the set of words  $u \in ((Q \times \Gamma) \cup \Gamma)^3$  that can appear in a successor configuration of  $c$  in the same position  $i$  (according to the rules of  $\mathcal{M}$ ).

- For all words  $\dot{\sigma}_1\sigma_2\sigma_3 \in (Q \dot{\times} \Gamma) \cup \dot{\Gamma} \cup ((Q \times \Gamma) \cup \Gamma)^2$  that starts with a dotted letter, we add a 1-NRA that accepts data words that for some occurrence of the subword  $(\dot{\sigma}_1, d_1)(\sigma_2, d_2)(\sigma_3, d_3)$  with some data  $d_1, d_2, d_3$ , the subword  $\tau_1\dot{\tau}_2\tau_3$  (ignoring the data values) with exactly  $2^{2^n} + 2^{n+1} + 1$  distance is not in  $\text{Post}(\sigma_1\sigma_2\sigma_3)$ . Observe that the subword  $\dot{\sigma}_1\sigma_2\sigma_3$  is intuitively indicating some part of some configuration  $c$  in some copy of  $\rho$ , and  $\tau_1\dot{\tau}_2\tau_3$  with distance  $2^{2^n} + 2^{n+1} + 1$  is a subword of the successor configuration of  $c$  in the next copy of  $\rho$ .

The following NRA is for the case  $(q_{\text{init}}, a_1)a_2\Box$ . To implement this 1-NRA, we rely on the previous conditions that two letters (apart from the delimiters) with the same datum have the exact distance  $2^{2^n} + 2^{n+1} + 1$  (checked with a parallel 1-NRA).



- We add a DFA that accepts data words  $v$  such that the last configuration in  $\rho$  does not contain a letter in  $(Q_f \times \Gamma) \cup (Q_f \times \Gamma)$ , where  $Q_f$  is the set of accepting control states of  $\mathcal{M}$ .

To complete the proof, we examine the existence of  $N \in \mathcal{O}(K)$  such that  $\mathcal{M}$  has an accepting computation over  $x$  if, and only if,  $\mathcal{R}$  is (length-bounded) non-universal with respect to the bound  $N$ . Given the shortest witness  $w \in \Sigma_{\mathcal{M}}^+$  of non-universality of  $\mathcal{R}$ , the projection  $v$  of  $w$  into  $\Sigma$  encodes an accepting computation of  $\mathcal{M}$  over  $x$ , and subsequently has length exactly  $K$ . The extra letters of  $w$  compared to  $v$  are to implement the five needed counters faithfully. However, these letters do not increase the length of  $w$  much more than  $K$ : for instance, the condition for counting the occurrences of  $\#$  requires that we accompany every  $\#$  with a single  $\text{Bit}_i^\#$ -letter. Hence,

$$N \leq \left( \underbrace{\text{cell letters and Bit}_i^d}_{2^{3n+1}} + \underbrace{2^{n+1}(2^n - 1)}_{\# \text{ and Bit}_i^\#} + \underbrace{2^{n+1} + 2}_{\star \text{ and Bit}_i^\star} + \underbrace{2^{3n+1}}_{\text{cell letters and Bit}_i} + \underbrace{2^{3n+1}}_{\text{cell letters and Bit}_i^x} \right).$$

Note that  $N$  is still exponential in  $n$ .

The construction of  $\mathcal{R}$  is complete and the NEXPTIME-hardness follows from the sketched reduction. Note that the result already holds for 1-NRAs.  $\square$

There is a natural reduction from the non-universality problem for 1-NRAs to the emptiness problem for single-register alternating RAs (1-ARAs). The trivial NEXPTIME membership (guess and check) and Theorem 12 lead to the NEXPTIME-completeness of the length-bounded emptiness problem for 1-ARAs.

**Acknowledgements** We thank Sylvain Schmitz for helpful discussions on well-structured systems and non-elementary complexity classes. We thank James Worrell for inspiring discussions, especially drawing our attention to a trick that simplified the NEXPTIME-hardness construction. We appreciate the anonymous reviewers for their insightful comments and suggestions.

## References

- [1] R. Angles and C. Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1):1:1–1:39, Feb. 2008.
- [2] P. Barceló, L. Libkin, A. W. Lin, and P. T. Wood. Expressive languages for path queries over graph-structured data. *ACM Trans. Database Syst.*, 37(4):31:1–31:46, Dec. 2012.
- [3] Y. Benenson, R. Adar, T. Paz-Elizur, Z. Livneh, and E. Shapiro. DNA molecule provides a computing machine with both data and fuel. *Proc. National Acad. Sci. USA*, 100:2191–2196, 2003.
- [4] M. Bojańczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. 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.
- [5] M. Bojanczyk and P. Parys. Xpath evaluation in linear time. *J. ACM*, 58(4):17:1–17:33, July 2011.
- [6] A. Bouajjani, P. Habermehl, Y. Jurski, and M. Sighireanu. Rewriting systems with data. In E. Csuhaj-Varjú and Z. 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.
- [7] P. Bouyer, A. Petit, and D. Thérien. An algebraic approach to data languages and timed languages. *Inf. Comput.*, 182(2):137–162, 2003.
- [8] J. Černý. Poznámka k homogénnym experimentom s konečnými automatmi. *Matematicko-fyzikálny časopis*, 14(3):208–216, 1964.
- [9] D. Chistikov, P. Martyugin, and M. 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.
- [10] L. Clemente and S. 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.
- [11] S. Demri and R. Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3), 2009.
- [12] S. Demri, R. Lazic, and D. Nowak. On the freeze quantifier in constraint LTL: decidability and complexity. *Inf. Comput.*, 205(1):2–24, 2007.
- [13] S. Demri, R. Lazic, and A. Sangnier. Model checking memoryful linear-time logics over one-counter automata. *Theor. Comput. Sci.*, 411(22-24):2298–2316, 2010.
- [14] L. Doyen, L. Juhl, K. G. Larsen, N. Markey, and M. Shirmohammadi. Synchronizing words for weighted and timed automata. In V. 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.

- [15] L. Doyen, T. Massart, and M. 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.
- [16] D. 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.
- [17] D. Figueira. Alternating register automata on finite words and trees. *Logical Methods in Computer Science*, 8(1), 2012.
- [18] D. Figueira, S. Figueira, S. Schmitz, and P. 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.
- [19] M. Kaminski and N. Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.
- [20] J. Kretínský, K. G. Larsen, S. Laursen, and J. 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.
- [21] K. G. Larsen, S. Laursen, and J. 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.
- [22] A. Lisitsa and I. 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.
- [23] P. 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.
- [24] F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.
- [25] J. Pin. Sur les mots synthonisants dans un automate fini. *Elektronische Informationsverarbeitung und Kybernetik*, 14(6):297–303, 1978.
- [26] H. Sakamoto and D. Ikeda. Intractability of decision problems for finite-memory automata. *Theor. Comput. Sci.*, 231(2):297–308, 2000.
- [27] S. Schmitz. Complexity hierarchies beyond elementary. *ACM Trans. Comput. Theory*, 8(1):3:1–3:36, 2016.
- [28] M. Shirmohammadi. Phd thesis: Qualitative analysis of probabilistic synchronizing systems. 2014.
- [29] N. Tzevelekos. Fresh-register automata. In T. Ball and M. 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.
- [30] M. V. Volkov. Synchronizing automata and the cerny conjecture. In C. Martín-Vide, F. Otto, and H. 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.

# Appendix

## 6 Proofs for Deterministic Register Automata

**Lemma 2.** *For all DRAs for which there exist synchronizing data words, there exists a synchronizing data word  $w$  such that  $|w| \leq 2|R| + 1$ .*

*Proof.* Let  $\mathcal{R} = \langle L, R, \Sigma, T \rangle$  be a DRA on the data domain  $D$  and with  $k \geq 1$  registers. Recall that we denote by  $\mathbf{data}(w)$  the data occurring in data words  $w$ ; for configurations  $q = (\ell, \nu)$  we use the same notation  $\mathbf{data}(q) = \{\nu(r) \mid r \in R\}$  to denote the data appearing in the valuation of  $q$ . Let  $\pi : Y_1 \rightarrow Y_2$  be a bijection on data where  $Y_1, Y_2 \subseteq D$ . For every configuration  $q = (\ell, \nu)$ , define  $\pi(q) = (\ell, \nu')$ , where  $\nu'$  satisfies  $\nu'(r) = \pi(\nu(r))$  for all  $r \in R$ . For every data word  $w = (a_1, d_1) \dots (a_n, d_n)$ , define  $\pi(w) = (a_1, \pi(d_1)) \dots (a_n, \pi(d_n))$ . Note that the application of  $\pi$  on  $q$  and  $w$  preserves the reachability property, *i.e.*,  $\mathbf{post}(\pi(q), \pi(w)) = \{\pi(q') \mid q' \in \mathbf{post}(q, w)\}$ .

Assuming that  $\mathcal{R}$  has some synchronizing data word, we first prove the following claim by an induction.

**Claim.** For all pairs of configurations  $q_1, q_2$ , if there exists  $w$  such that  $|\mathbf{post}(\{q_1, q_2\}, w)| = 1$ , then

- for all sets  $X = \{x_1, x_2, \dots, x_{2k+1}\} \subseteq D$  with  $\mathbf{data}(q_1), \mathbf{data}(q_2) \subseteq X$ ,
- there exists some data word  $w_{q_1, q_2} \in (\Sigma \times X)^*$  such that  $|\mathbf{post}(\{q_1, q_2\}, w_{q_1, q_2})| = 1$ .

Note that by  $|X| = 2k + 1$ , the data efficiency of  $w_{q_1, q_2}$  is at most  $2k + 1$ .

**Proof of Claim.** Let  $q_1$  and  $q_2$  be two configurations of  $\mathcal{R}$  and define  $\mathbf{data}(q_1, q_2) = \mathbf{data}(q_1) \cup \mathbf{data}(q_2)$ . Since  $\mathcal{R}$  has some synchronizing data words, there exists  $w$  such that  $|\mathbf{post}(\{q_1, q_2\}, w)| = 1$ . The proof is by an induction on the length of  $w$ .

**Base of induction.** Assume  $w = (a, d)$  have length  $|w| = 1$ . Let  $X$  be any arbitrary set of data such that  $|X| = 2k + 1$  and  $\mathbf{data}(q_1, q_2) \subseteq X$ . There are two cases:

- $d \in X$ : This entails that  $\mathbf{data}(w) \subseteq X$ . Observe that  $w_{q_1, q_2} = w$  satisfies the induction statement.
- $d \notin X$ : Since  $|\mathbf{data}(q_1, q_2)| \leq 2k$ , there exists data  $x \neq d$  such that  $x = X \setminus \mathbf{data}(q_1, q_2)$ . Since  $x \neq d$ , we can define the bijection  $\pi : \{d\} \cup \mathbf{data}(q_1, q_2) \rightarrow \{x\} \cup \mathbf{data}(q_1, q_2)$  such that  $\pi(d) = x$  and  $\pi(d') = d'$  for all  $d' \in \mathbf{data}(q_1, q_2)$ . Observe that  $\pi(q_i) = q_i$  for all  $i \in \{1, 2\}$ . Then

$$|\mathbf{post}(\{q_1, q_2\}, (a, d))| = |\mathbf{post}(\{\pi(q_1), \pi(q_2)\}, (a, \pi(d)))| = |\mathbf{post}(\{q_1, q_2\}, (a, x))|.$$

This and the assumption  $|\mathbf{post}(\{q_1, q_2\}, (a, d))| = 1$  yield  $|\mathbf{post}(\{q_1, q_2\}, (a, x))| = 1$ . The word  $w_{q_1, q_2} = (a, x)$  satisfies the induction statement.

The base of induction hence holds.

**Step of induction.** Assume that the induction hypothesis holds for  $i - 1$ . Consider some word  $(a, d) \cdot w$  such that  $|w| = i - 1$  and  $|\mathbf{post}(\{q_1, q_2\}, (a, d) \cdot w)| = 1$ .

Consider some set  $X$  which has cardinality  $2k + 1$  and  $\mathbf{data}(q_1, q_2) \subseteq X$ , we construct the data word  $w_{q_1, q_2}$  as follows. Let  $p_1 = \mathbf{post}(q_1, (a, d))$  and  $p_2 = \mathbf{post}(q_2, (a, d))$ , and let  $\mathbf{data}(p_1, p_2) = \mathbf{data}(p_1) \cup \mathbf{data}(p_2)$ . Due to the fact that  $p_1, p_2$  are successors of  $q_1, q_2$  after inputting  $(a, d)$ , we know that if  $d \in \mathbf{data}(q_1, q_2)$  then  $d \in \mathbf{data}(p_1, p_2)$ . There are two cases:

- $d \in \mathbf{data}(q_1, q_2)$  or  $d \notin \mathbf{data}(p_1, p_2)$ . These guarantee that  $\mathbf{data}(p_1, p_2) \subseteq \mathbf{data}(q_1, q_2)$  if  $d \in \mathbf{data}(q_1, q_2)$ , and that  $\mathbf{data}(p_1, p_2) = \mathbf{data}(q_1, q_2)$  if  $d \notin \mathbf{data}(p_1, p_2)$ . As a result,  $\mathbf{data}(p_1, p_2) \subseteq X$ . By induction hypothesis, there exists some data word  $w_{p_1, p_2}$  over data domain  $X$  such that  $|\mathbf{post}(\{p_1, p_2\}, w_{p_1, p_2})| = 1$ . For  $w_{q_1, q_2} = (a, d) \cdot w_{p_1, p_2}$  the statement of induction holds, as  $|\mathbf{post}(\{q_1, q_2\}, w_{q_1, q_2})| = 1$ .

- $d \notin \text{data}(q_1, q_2)$  and  $d \in \text{data}(p_1, p_2)$ . Without loss of generality, we assume that  $d \notin X$ . Otherwise  $d \in X$  would imply  $\text{data}(p_1, p_2) \subseteq X$ , and we simply let  $w_{q_1, q_2} = w_{p_1, p_2}$ . Since  $|\text{data}(q_1, q_2)| \leq 2k$ , there exists some datum  $x \neq d$  such that  $x \in X \setminus \text{data}(q_1, q_2)$ . Since  $x \neq d$ , we can define the bijection  $\pi : \{d\} \cup \text{data}(q_1, q_2) \rightarrow \{x\} \cup \text{data}(q_1, q_2)$  such that  $\pi(d) = x$  and  $\pi(d') = d'$  for all  $d' \in \text{data}(q_1, q_2)$ . Since  $\text{data}(p_1, p_2) \setminus \{d\} \subseteq \text{data}(q_1, q_2)$ , having  $d$  in the domain of  $\pi$ , the bijection  $\pi$  ranges over  $\text{data}(p_1, p_2)$ . By induction hypothesis, there exists some data word  $w_{p_1, p_2}$  over data domain  $(X \setminus \{x\}) \cup \{d\}$  such that  $|\text{post}(\{p_1, p_2\}, w_{p_1, p_2})| = 1$ . Then,  $|\text{post}(\{\pi(p_1), \pi(p_2)\}, \pi(w_{p_1, p_2}))| = 1$ . For all  $1 \leq i \leq 2$ , we have  $\pi(p_i) \in \text{post}(q_i, (a, x))$  since  $p_i \in \text{post}(q_i, (a, d))$  and  $x = \pi(d)$ . By above arguments, we conclude that  $|\text{post}(\{q_1, q_2\}, (a, x)\pi(w_{p_1, p_2}))| = 1$ . As  $\{x\} \cup \text{data}(\{q_1, q_2\}) \subseteq X$ , thus the data word  $w_{q_1, q_2} = (a, x)\pi(w_{p_1, p_2})$  satisfies the statement of induction.

The above arguments prove that in all cases, there exists  $w_{q_1, q_2} \in (\Sigma \times X)^*$  that merges two configurations  $q_1$  and  $q_2$  into a singleton, which completes the proof of **Claim**.

Since  $\mathcal{R}$  has some synchronizing data word, using Lemma 1, we know that there exists some word  $w$  with data efficiency  $k$  such that  $\text{post}(L \times D^k, w) \subseteq L \times \text{data}(w)^k$ . Consider some set  $X = \{x_1, x_2, \dots, x_{2k+1}\} \subset D$  such that  $\text{data}(w) \subseteq X$ . We use the pairwise synchronization technique as follows. Define  $S_n = L \times X^k$  and  $n = |L|(2k+1)^k$ , i.e.,  $|S_n| = n$ . For all  $i = n-1, \dots, 1$  repeat the following:

1. Take a pair of configurations  $q_1, q_2 \in S_{i+1}$ . By the **Claim** above, one can find some word  $w_{q_1, q_2} \in (\Sigma \times X)^*$  such that  $|\text{post}(\{q_1, q_2\}, w_{q_1, q_2})| = 1$ ,
2. Define  $v_i = w_{q_1, q_2}$  and  $S_i = \text{post}(S_{i+1}, v_i)$ .

Note that by determinism of  $\mathcal{R}$ , for every  $i \in \{1, \dots, n-1\}$ , we have  $|S_i| \leq |S_{i+1}| - 1$ . Thus the word  $w_{\text{synch}} = w \cdot v_{n-1} \cdots v_2 \cdot v_1$  is a synchronizing data word for  $\mathcal{R}$ . Since  $\text{data}(w) \subseteq X$  and  $\text{data}(v_i) \subseteq X$  for all  $i \in \{1, \dots, n-1\}$ , the data efficiency of  $w_{\text{synch}}$  is at most  $2k+1$ . The proof is complete.  $\square$

**Lemma 5.** *The synchronization problem for  $k$ -DRAs is PSPACE-complete.*

*Proof.* We prove PSPACE-hardness by a reduction from the non-emptiness problem for  $k$ -DRA. Let  $\mathcal{R} = (L, R, \Sigma, T)$  be a  $k$ -DRA equipped with an initial location  $\ell_i$  and an accepting location  $\ell_f$ , where, without loss of generality, we assume that all outgoing transitions from  $\ell_i$  update all registers, and that  $\ell_f$  has no outgoing edges. We also assume that  $\mathcal{R}$  is complete, otherwise, we add some non-accepting location and direct all undefined transitions to it.

The reduction is such that from  $\mathcal{R}$  we construct another  $k$ -DRA  $\mathcal{R}_{\text{syn}}$  such that the language of  $\mathcal{R}$  is not empty if, and only if,  $\mathcal{R}_{\text{syn}}$  has some synchronizing data word. We define  $\mathcal{R}_{\text{syn}} = (L_{\text{syn}}, R, \Sigma_{\text{syn}}, T_{\text{syn}})$  as follows. The set of locations is  $L_{\text{syn}} = L \cup \{\text{reset}\}$ , where  $\text{reset} \notin L$  is a new location; the alphabet is  $\Sigma_{\text{syn}} = \Sigma \cup \{\star\}$ , where  $\star \notin \Sigma$ . To define  $T_{\text{syn}}$ , we add the following transitions to  $T$ .

- $\ell_f \xrightarrow{a \ R \downarrow} \ell_f$  for all letters  $a \in \Sigma_{\text{syn}}$ ,
- $\ell_i \xrightarrow{\star \ R \downarrow} \ell_i$
- $\text{reset} \xrightarrow{a \ R \downarrow} \ell_i$  for all letters  $a \in \Sigma_{\text{syn}}$ ,
- $\ell \xrightarrow{\star \ R \downarrow} \text{reset}$  for all  $\ell \in L_{\text{syn}}$  except for  $\text{reset}, \ell_i, \ell_f$ .

Note that  $\mathcal{R}_{\text{syn}}$  is indeed deterministic and complete. To establish the correctness of the reduction, we prove that the language of  $\mathcal{R}$  is not empty if, and only if,  $\mathcal{R}_{\text{syn}}$  has a synchronizing data word.

First, assume that the language of  $\mathcal{R}$  is not empty. Then there exists a data word  $w = (a_1, d_1) \dots (a_n, d_n)$  such that  $w \in L(\mathcal{R})$ . Hence there exists a run starting from  $(\ell_i, \nu_i)$  and ending in  $(\ell_f, \nu_f)$  for some  $\nu_i, \nu_f \in D^{|R|}$ . The data word  $(\star, d)(\star, d)w(\star, d)$  for some  $d \in D$  synchronizes  $\mathcal{R}_{\text{syn}}$  in location  $\ell_f$ .

Second, assume that  $\mathcal{R}_{\text{syn}}$  has some synchronizing data word. Let  $w \in (\Sigma_{\text{syn}} \times D)^*$  be one of the shortest data synchronizing data words. All transitions in  $\ell_f$  are self-loops with update on all registers;

Hence,  $\mathcal{R}_{\text{syn}}$  can only be synchronized in  $\ell_f$ . Hence, we also have  $\text{post}((\ell_i, \nu_i), w) = \{(\ell_f, \nu_f)\}$  (for some  $\nu_i, \nu_f \in D^{|R|}$ ). By the fact that  $w$  is a shortest synchronizing data word, we can infer that the corresponding run does not contain any  $\star$ -transitions except for two self-loops in  $\ell_i$  in the very beginning. Hence there exists a run from  $(\ell_i, \nu_i)$  to  $\ell_f$  and thus  $L(\mathcal{R}) \neq \emptyset$ .  $\square$

## 7 Proofs for Non-deterministic Register Automata

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

*Proof.* The family of 1-NRAs  $(\mathcal{R}_{\text{counter}(n)})_{n \in \mathbb{N}}$  is defined as follows. We define the alphabet of RA  $\mathcal{R}_{\text{counter}(n)}$  by  $\Sigma = \{\#, \star, \text{Bit}_0, \text{Bit}_1, \dots, \text{Bit}_n\}$ . The structure of  $\mathcal{R}_{\text{counter}(n)}$  is composed of three distinguished locations *synch*, *reset*, *zero* and locations  $2^n, 2^{n-1}, \dots, 2^1, 2^0$  and  $2_c^n, 2_c^{n-1}, \dots, 2_c^1, 2_c^0$ . The general structure of  $\mathcal{R}_{\text{counter}(n)}$  is partially depicted in Figure 4. The RA  $\mathcal{R}_{\text{counter}(n)}$  is constructed such that for all synchronizing data words  $w$ , some datum  $x \in \text{data}(w)$  appears in  $w$  at least  $2^n$  times. A counting feature is thus embedded in  $\mathcal{R}_{\text{counter}(n)}$ : intuitively, the set of all reached configurations represents the counter value. Starting from  $\{(\text{zero}, x)\}$ , the first increment results in  $\{2_c^n, \dots, 2_c^2, 2_c^1, 2_c^0\} \times \{x\}$ , where location  $2^i$  means that the  $i$ -th least significant bit in the binary representation of the counter value is set to 1, and location  $2_c^i$  means that the  $i$ -th bit is set to 0. Informally, we say that there is an  $x$ -token in every reached location. Here,  $2_c^n, \dots, 2_c^2, 2_c^1, 2_c^0$  have  $x$ -tokens. A sequence of counter increments is encoded by re-placing the  $x$ -tokens, as shown in the following sequence of sets of locations:  $\{2_c^n, \dots, 2_c^2, 2_c^1, 2_c^0\}$ ,  $\{2_c^n, \dots, 2_c^2, 2_c^1, 2_c^0\}$ ,  $\{2_c^n, \dots, 2_c^3, 2_c^2, 2_c^1, 2_c^0\}$ , etc. The transitions of  $\mathcal{R}_{\text{counter}(n)}$  are defined in such a way that, starting from  $\{(\text{zero}, x)\}$ , either  $2^i$  or  $2_c^i$  have tokens, but never both of them at the same time. We now present a detailed explanation of the structure of  $\mathcal{R}_{\text{counter}(n)}$ .

All transitions in *synch* are self-loops with an update on the register *synch*  $\xrightarrow{\Sigma \ r \downarrow}$  *synch*. Thus,  $\mathcal{R}_{\text{counter}(n)}$  can only be synchronized in *synch*. Moreover, *synch* is only accessible by  $\#$ -transitions. Similarly, all transitions except for those with label  $\star$ , are self-loops in location *reset*; thus,  $\mathcal{R}_{\text{counter}(n)}$  can only be synchronized by leaving *reset* by reading  $\star$ . We use this also to avoid transitions which are *incorrect* with respect to the binary incrementing process: all incorrect actions are guided to *reset* to enforce another  $\star$ . Assuming  $w$  to be one of the shortest synchronizing words, we see that  $\text{post}(L \times D, w) = \{(\text{synch}, x)\}$ , where  $w$  starts with  $(\star, x)$  and ends with  $(\#, x)$ .

The counting involves an *initializing process* and several *incrementing processes*.

- *initializing the counter to zero*: the  $\star$ -transitions are devised to place a token in *zero*: from all locations  $\ell \in L \setminus \{\text{synch}\}$  we have  $\ell \xrightarrow{\star \ r \downarrow} \text{zero}$ . This sets the counter to 0.
- *incrementing the counter*: we use  $\text{Bit}_0, \dots, \text{Bit}_n$ -transitions with equality guards to control the increment. Intuitively, an equality-guarded  $\text{Bit}_i$ -transition is taken to set the  $i$ -th bit in the binary representation of the counter value according to the standard rules of binary incrementation. Initially, the token in *zero* splits in  $2^0$  and  $2_c^n, \dots, 2_c^1$  to represent  $0 \dots 01$ , by taking the transitions  $\text{zero} \xrightarrow{=r \ \text{Bit}_0} 2^0$  and  $\text{zero} \xrightarrow{=r \ \text{Bit}_0} 2_c^j$  for all  $1 \leq j \leq n$ . Equality-guarded  $\text{Bit}_i$ -transitions for  $i \in \{1, \dots, n\}$  are incorrect for *zero* and thus guided to *reset*. Whenever data different from  $x$  is processed,  $\mathcal{R}_{\text{counter}(n)}$  takes self-loops (omitted in Figure 4) and keeps the  $x$ -tokens unmoved. The equality-guarded  $\text{Bit}_i$ -transitions should only be taken if the  $i$ -th bit is not set, or, equivalently, if the location  $2^i$  contains no token. This is guaranteed by a  $\text{Bit}_i$ -transition  $2^i \xrightarrow{=r \ \text{Bit}_i} \text{reset}$ , for every  $0 \leq i \leq n$ , which results in an incorrect transition and should be avoided. (Otherwise the counting process has to restart from 0.) In Figure 4, we depict the corresponding transitions for  $i = 2$  and  $i = n$ .

Further, we need to guarantee that for all  $i \geq 1$  a  $\text{Bit}_i$ -transition is taken only if all less significant bits are set, or, equivalently, if all locations  $2^{i-1}, \dots, 2^0$  contain a token. This is ensured by a  $\text{Bit}_i$ -transition  $2_c^j \xrightarrow{=r \ \text{Bit}_i} \text{reset}$ , for every  $0 \leq j < i$ , which again results in an incorrect transition. See, e.g., the transition  $2_c^2 \xrightarrow{=r \ \text{Bit}_3} \text{reset}$  in Figure 4 for every  $3 \leq i \leq n$ .

Finally,  $\text{Bit}_i$ -transitions must produce tokens in  $2^i$  and  $2_c^0, \dots, 2_c^{i-1}$ , thus  $2_c^i \xrightarrow{r \text{ Bit}_i} 2^i$  and  $2^j \xrightarrow{r \text{ Bit}_i} 2_c^j$  for all  $0 \leq j < i$ . All tokens in locations  $2^j$  and  $2_c^j$ , respectively, for  $j > i$  remain where they are, which is implemented by equality-guarded  $\text{Bit}_i$ -self-loops in  $2^j$  and  $2_c^j$ , respectively.

By construction, it is easy to see that  $\text{Bit}_i$ -transitions are the only way to produce a token in  $2^i$ , which can be fired if  $2_c^i$  has a token. The  $\text{Bit}_i$ -transitions then consume the token in  $2_c^i$ . This guarantees that after the first  $\star$ -transition, which puts a token into zero, the two locations  $2^i$  and  $2_c^i$  will never have a token at the same time.

Finally, all equality-guarded  $\#$ -transitions in  $2_c^n$  and  $2^i$  for all  $0 \leq i < n$  are sent to **reset**. In contrast, all  $\#$ -transitions in  $2^n$  and  $2_c^i$  for all  $0 \leq i < n$  are sent to **synch**, with an update on the register. This guarantees that the counter must correctly count from 0 to  $10 \dots 0$ , meaning that at least one datum  $x$  appears at least  $2^n$  times while synchronizing  $\mathcal{R}_{\text{counter}(n)}$ .  $\square$

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

*Proof.* The reduction is based on the construction presented in Theorem 17 in [14].

Let  $\mathcal{R} = \langle L, R, \Sigma, T \rangle$  be an NRA equipped with an initial location  $\ell_{\text{in}}$  and a set  $L_f$  of accepting locations, where, without loss of generality, we assume that all outgoing transitions from  $\ell_{\text{in}}$  update all registers. We also assume that  $\mathcal{R}$  is complete, otherwise, we add some non-accepting location and direct all undefined transitions to it.

We construct an NRA  $\mathcal{R}_{\text{syn}}$  such that there exists some data word that is not in  $L(\mathcal{R})$  if, and only if,  $\mathcal{R}_{\text{syn}}$  has some synchronizing data word. We define  $\mathcal{R}_{\text{syn}} = \langle L_{\text{syn}}, R, \Sigma_{\text{syn}}, T_{\text{syn}} \rangle$  as follows. The set of locations is  $L_{\text{syn}} = L \cup \{\text{reset}, \text{synch}\}$  where **synch**, **reset**  $\notin L$  are two new locations. The alphabet is  $\Sigma_{\text{synch}} = \Sigma \cup \{\#, \star\}$  where  $\#, \star \notin \Sigma$ . The transition relation  $T_{\text{syn}}$  is the union of  $T$  and set containing the following transitions:

- **synch**  $\xrightarrow{a R \downarrow}$  **synch** for all letters  $a \in \Sigma_{\text{syn}}$ ,
- **reset**  $\xrightarrow{\star R \downarrow}$   $\ell_{\text{in}}$  and **reset**  $\xrightarrow{a R \downarrow}$  **reset** for all letters  $a \in \Sigma_{\text{syn}} \setminus \{\star\}$ ,
- $\ell \xrightarrow{\star R \downarrow} \ell_{\text{in}}$  for all locations  $\ell \in L$ ,
- $\ell \xrightarrow{\# R \downarrow}$  **synch** for all non-accepting locations  $\ell \in L \setminus L_f$ ,
- $\ell \xrightarrow{\# R \downarrow}$  **reset** for all accepting locations  $\ell \in L_f$ .

Next, we prove the correctness of the reduction.

First, assume there exists a data word  $w = (a_1, d_1) \dots (a_n, d_n)$  such that  $w \notin L(\mathcal{R})$ . Hence, all runs starting in  $(\ell_{\text{in}}, \nu_i)$  with  $\nu_i \in D^{|R|}$  end in some configuration  $(\ell, \nu)$  with  $\ell \notin L_f$ . The data word  $(\star, d) \cdot w \cdot (\#, d)$  with  $d \in D$  synchronizes  $\mathcal{R}_{\text{syn}}$  in location **synch**, proving that  $\mathcal{R}_{\text{syn}}$  has some synchronizing data word.

Second, assume that  $\mathcal{R}_{\text{syn}}$  has some synchronizing data word. All transitions in **synch** are self-loops with update on all registers; thus,  $\mathcal{R}_{\text{syn}}$  can only synchronize in **synch**. Moreover, **synch** is only accessible with  $\#$ -transitions; assuming  $w$  is one of the shortest synchronizing data words, we see that  $\text{post}(L \times D, w) = \{(\text{synch}, \nu)\}$  for some  $\nu \in D^{|R|}$ . From all locations  $\ell \in L$  we have  $\ell \xrightarrow{\star R \downarrow} \ell_{\text{in}}$ ; we say that  $\star$ -transitions *reset*  $\mathcal{R}_{\text{syn}}$ . Moreover, the only outgoing transition in location **reset** is the  $\star$ -transition. Thus, a *reset* followed by some  $\#$  must occur while synchronizing. Let  $w = w_0(\star, d_\star)w_1(\#, d_\#)w_2$ , where  $w_1 \in (\Sigma \times D)^+$  is the data word between the last occurrence of  $\star$  and the first following occurrence of  $\#$ , and  $w_2 \in (\Sigma' \setminus \{\star\})^*$ . We prove that  $w_1 \notin L(\mathcal{R})$ . By contradiction, assume that  $w_1$  is in the language; thus, there exist valuations  $\nu_i, \nu_f \in D^{|R|}$  such that  $\mathcal{R}_{\text{syn}}$  has a run over  $w_1$ , *i.e.*, starting in  $(\ell_{\text{in}}, \nu_i)$  and ending in  $(\ell_f, \nu_f)$  where  $\ell_f \in L_f$ . In fact, since all outgoing transitions in  $\ell_{\text{in}}$  update all registers, then for all valuations  $\nu_i$ ,  $\mathcal{R}_{\text{syn}}$  has an accepting run over  $w_1$ .

Note that  $w_0$  cannot be a synchronizing word for  $\mathcal{R}_{\text{syn}}$ , because this would contradict the assumption that  $w$  is one of the shortest synchronizing data word. It implies that there must be some configuration  $q$



such that  $\text{post}_{\mathcal{R}_{\text{syn}}}(q, w_0)$  contains some configuration  $(\ell, \nu)$  with  $\ell \neq \text{synch}$ . From  $(\ell, \nu)$ , inputting the next  $(\star, d_\star)$  (that is after  $w_0$  in synchronizing word  $w$ ), we reach  $(\ell_{\text{in}}, \{d_\star\}^{|R|})$ . Since for all valuations  $\nu_i$ , starting in  $(\ell_{\text{in}}, \nu_i)$ ,  $\mathcal{R}_{\text{synch}}$  has an accepting run over  $w_1$ , it must have an accepting run from  $(\ell_{\text{in}}, \{d_\star\}^{|R|})$  to some accepting configuration  $(\ell_f, \nu_f)$  too. Reading the last  $\#$  (that is after  $w_1$  in synchronizing word  $w$ ), **reset** is reached. Since  $w_2$  does not contain any  $\star$ , **reset** is never left, meaning that  $\mathcal{R}_{\text{syn}}$  cannot synchronize in **synch**, a contradiction. The proof is complete.

Note that the reduction preserves the number of registers in the NRAs. □