The Containment Problem for Unambiguous Register Automata

Antoine Mottet[1]
Department of Algebra, Charles University, Czech Republic
https://orcid.org/0000-0002-3517-1745

Karin Quaas[2]
Universität Leipzig, Germany

Abstract

We investigate the complexity of the containment problem "Does $L(\mathcal{A}) \subseteq L(\mathcal{B})$ hold?", where $\mathcal{B}$ is an unambiguous register automaton and $\mathcal{A}$ is an arbitrary register automaton. We prove that the problem is decidable and give upper bounds on the computational complexity in the general case, and when $\mathcal{B}$ is restricted to have a fixed number of registers.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases Data words, Register automata, Unambiguous Automata, Containment Problem, Language Inclusion Problem

1 Introduction

Register automata [11] are a widely studied model of computation that extend finite automata with finitely many *registers* that are able to hold values from an infinite domain and perform equality comparisons with data from the input word. This allows register automata to accept *data languages*, i.e., sets of *data words* over $\Sigma \times \mathbb{D}$, where Σ is a finite alphabet and $\mathbb{D}$ is an infinite set called the data domain. The study of register automata is motivated by problems in formal verification and database theory, where the objects under study are accompanied by annotations (identification numbers, labels, parameters, ...), see the survey by Ségoufin [19]. One of the central problems in these areas is to check whether a given input document or program complies with a given input specification. In our context, this problem can be formalized as a *containment problem*: given two register automata $\mathcal{A}$ and $\mathcal{B}$, does $L(\mathcal{A}) \subseteq L(\mathcal{B})$ hold, i.e., is the data language accepted by $\mathcal{A}$ included in the data language accepted by $\mathcal{B}$? Here, $\mathcal{B}$ is understood as a specification, and one wants to check whether $\mathcal{A}$ satisfies the specification. For arbitrary register automata, the containment problem is undecidable [15, 5]. It is known that one can recover decidability in two different ways. First, the containment problem is known to be PSPACE-complete when $\mathcal{B}$ is a deterministic register automaton [5]. This is a severe restriction on the expressive power of $\mathcal{B}$, and it is of practical interest to find natural classes of register automata that can be tackled algorithmically and that can express more properties than deterministic register automata. Secondly, one can recover decidability of the containment problem when $\mathcal{B}$ is a non-deterministic register automaton with a single register [11, 5]. However, in this setting, the problem is Ackermann-complete [7]; it can therefore hardly be considered tractable.

[1] This author received funding from DFG Graduiertenkolleg 1763 (QuantLA) and from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 771005, "CoCoSym").

[2] Supported by DFG, QU 316/1-2.

This motivates the study of *unambiguous* register automata, which form the class **URA** of register automata sitting between the class **DRA** of deterministic register automata and the class **NRA** of non-deterministic register automata. Unambiguous register automata are non-deterministic automata for which every data word has at most one accepting run.

In the present paper, we investigate the complexity of the containment problem when $\mathcal{B}$ is restricted to be an unambiguous register automaton. We prove that the problem is decidable with a 2-EXPSPACE complexity, and is even decidable in EXPSPACE if $\mathcal{B}$ is further restricted to have a single register (compare with the undecidability, respectively, Ackermann-completeness, in the non-deterministic case). Classically, one way to approach the containment problem (for general models of computation) is to reduce it to a reachability problem on an infinite transition system, called the *synchronized state space of* $\mathcal{A}$ *and* $\mathcal{B}$, cf. [16]. Proving decidability or complexity upper bounds for the containment problem then amounts to finding criteria of termination or bounds on the complexity of a reachability algorithm on this space. In this paper, our techniques also rely on the analysis of the synchronized state space of $\mathcal{A}$ and $\mathcal{B}$, where our main contribution is to provide a bound on the size of synchronized states that one needs to explore before being able to certify that $L(\mathcal{A}) \subseteq L(\mathcal{B})$ holds. This bound is found by identifying elements of the synchronized state space whose behaviour is similar, and by showing that every element of the synchronized state space is equivalent to a small one. In the general case, where $\mathcal{B}$ is unambiguous and $\mathcal{A}$ is an arbitrary non-deterministic register automaton, we bound the size of the graph that one needs to inspect by a triple exponential in the size of $\mathcal{A}$ and $\mathcal{B}$. In the restricted case that $\mathcal{B}$ has a fixed number of registers, we proceed to give a better bound that is only doubly exponential in the size of $\mathcal{A}$ and $\mathcal{B}$.

Related Literature A thorough study of the current literature on register automata reveals that there exists a variety of different definitions of register automata, partially with significantly different semantics. In this paper, we study register automata as originally introduced by Kaminski and Francez [11]. Such register automata process data words over an infinite data domain. The registers can take data values that appear in the input data word processed so far. The current input datum can be compared for (in)equality with the data that is stored in the registers. Kaminski and Francez study register automata mainly from a language-theoretic point of view; more results on the connection to logic, as well as the decidability status and computational complexity of classical decision problems like emptiness and containment are presented, e.g., in [18, 15, 5]. In [8], register automata over *ordered* data domains are studied.

Kaminski and Zeitlin [12] define a generalisation of the model in [11], in the following called *register automata with guessing.* The registers in such automata can non-deterministically reassign, or "guess", the datum of a register. In particular, such register automata can store data values that have not appeared in the input data word before, in contrast to the register automata in [11]. Register automata with guessing are strictly more expressive than register automata; for instance, there exists a register automaton with guessing that accepts the complement of the data language accepted by the register automaton in Figure 1 (Example 4 in [12]). Figueira [6] studies an alternating version of this model, also over ordered data domains. Colcombet [3, 2] considers *unambiguous* register automata with guessing. In Theorem 12 in [3], it is claimed that this automata class is effectively closed under complement, so that universality, containment and equivalence are decidable; however, to the best of our knowledge, this claim remains unproved.

Finally, unambiguity has become an important topic in automata theory, as witnessed by the growing body of literature in the recent years [9, 14, 4, 17]. In addition to the motivations mentioned above, unambiguous automata form an important model of computation due to their *succinctness* compared to their deterministic counterparts. For example, it is known that unambiguous finite automata can be exponentially smaller than deterministic automata [13] while the fundamental problems (such as emptiness, universality, containment, equivalence) remain tractable.

2 Main Definitions

We study register automata as introduced in the seminal paper by Kaminski and Francez [11]. Throughout the paper, Σ denotes a finite alphabet, and $\mathbb{D}$ denotes an infinite set of data values equipped with some equivalence relation $=$. In our examples, we assume $\mathbb{D} = \mathbb{N}$, the set of non-negative integers. A *data word* is a finite sequence $(\sigma_1, d_1) \dots (\sigma_k, d_k) \in (\Sigma \times \mathbb{D})^*$. A *data language* is a set of data words. We use ε to denote the *empty data word*. The *length* k of a data word w is denoted by $|w|$. Given a data word w as above and $0 \leq i \leq k$, we define the infix $w(i, j] := (\sigma_{i+1}, d_{i+1}) \dots (\sigma_j, d_j)$. Note that $w(i, i] = \varepsilon$. We use $\mathrm{data}(w)$ to denote the set $\{d_1, \dots, d_k\}$ of all data occurring in w. We use $\mathrm{proj}(w)$ to denote the projection of w onto Σ^*, i.e., the word $\sigma_1 \dots \sigma_k$.

Let $\mathbb{D}_\perp$ denote the set $\mathbb{D} \cup \{\perp\}$, where $\perp \notin \mathbb{D}$ is a fresh symbol not occurring in $\mathbb{D}$. We extend the equivalence relation $=$ over $\mathbb{D}$ to $\mathbb{D}_\perp$ by setting $\perp = \perp$ and $d \neq \perp$ for all $d \in \mathbb{D}$. We use boldface lower-case letters like $\boldsymbol{a}, \boldsymbol{b}, \dots$ to denote vectors in $\mathbb{D}_\perp^n$, where $n \in \mathbb{N}$. For a vector $\boldsymbol{a}$, we write a_i for its i-th component. If $\boldsymbol{a} \in \mathbb{D}_\perp^n$, then $\mathrm{data}(\boldsymbol{a})$ denotes the set $\{a_1, \dots, a_n\}$ of all data occurring in $\boldsymbol{a}$.

Let $R = \{r_1, \dots, r_n\}$ be a finite set of *registers*. A *register valuation* is a mapping $\boldsymbol{a} : R \to \mathbb{D}_\perp$; we may write a_i as shorthand for $\boldsymbol{a}(r_i)$. Let $\mathbb{D}_\perp^R$ denote the set of all register valuations. Given $\lambda \subseteq R$ and $d \in \mathbb{D}$, define the register valuation $\boldsymbol{a}[\lambda \leftarrow d]$ by $(\boldsymbol{a}[\lambda \leftarrow d])(r_i) := d$ if $r_i \in \lambda$, and $(\boldsymbol{a}[\lambda \leftarrow d])(r_i) := a_i$ otherwise.

A *register constraint* over R is defined by the grammar

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

where $r \in R$. We use $\Phi(R)$ to denote the set of all register constraints over R. We may use $\neq r$ or $\phi_1 \vee \phi_2$ as shorthand for $\neg(= r)$ and $\neg(\neg\phi_1 \wedge \neg\phi_2)$, respectively. The satisfaction relation $\models$ for $\Phi(R)$ on $\mathbb{D}_\perp^R \times \mathbb{D}$ is defined by structural induction in the obvious way; e.g., $\boldsymbol{a}, d \models (= r_1 \wedge \neq r_2)$ if $a_1 = d$ and $a_2 \neq d$.

A *register automaton over* Σ is a tuple $\mathcal{A} = (R, \mathcal{L}, \ell_{\mathrm{in}}, \mathcal{L}_{\mathrm{acc}}, E)$, where

- R is a finite set of registers,
- $\mathcal{L}$ is a finite set of *locations*,
- $\ell_{\mathrm{in}} \in \mathcal{L}$ is the *initial location*,
- $\mathcal{L}_{\mathrm{acc}} \subseteq \mathcal{L}$ is the set of *accepting locations*, and
- $E \subseteq \mathcal{L} \times \Sigma \times \Phi(R) \times 2^R \times \mathcal{L}$ is a finite set of *edges*. We may write $\ell \xrightarrow{\sigma,\phi,\lambda} \ell'$ to denote an edge $(\ell, \sigma, \phi, \lambda, \ell') \in E$. Here, σ is the label of the edge, ϕ is the register constraint of the edge, and λ is the set of updated registers of the edge. A clock constraint $\texttt{true}$ is vacuously true and may be omitted; likewise we may omit λ if $\lambda = \emptyset$.

A *state* of $\mathcal{A}$ is a pair $(\ell, \boldsymbol{a}) \in \mathcal{L} \times \mathbb{D}_\perp^R$, where ℓ is the current location and $\boldsymbol{a}$ is the current register valuation. Given two states $(\ell, \boldsymbol{a})$ and $(\ell', \boldsymbol{a'})$ and some input letter $(\sigma, d) \in (\Sigma \times \mathbb{D})$, we postulate a transition $(\ell, \boldsymbol{a}) \xrightarrow{\sigma,d}_{\mathcal{A}} (\ell', \boldsymbol{a'})$ if there exists some edge $\ell \xrightarrow{\sigma,\phi,\lambda} \ell'$ such that $\boldsymbol{a}, d \models \phi$ and $\boldsymbol{a'} = \boldsymbol{a}[\lambda \leftarrow d]$. If the context is clear, we may omit the index $\mathcal{A}$ and write $(\ell, \boldsymbol{a}) \xrightarrow{\sigma,d} (\ell', \boldsymbol{a'})$ instead of $(\ell, \boldsymbol{a}) \xrightarrow{\sigma,d}_{\mathcal{A}} (\ell', \boldsymbol{a'})$. We use $\longrightarrow^*$ to denote the reflexive transitive closure of $\longrightarrow$. A *run* of $\mathcal{A}$ on the data word $(\sigma_1, d_1) \dots (\sigma_k, d_k)$ is a sequence $(\ell_0, \boldsymbol{a^0}) \xrightarrow{\sigma_1,d_1} (\ell_1, \boldsymbol{a^1}) \xrightarrow{\sigma_2,d_2} \dots \xrightarrow{\sigma_k,d_k} (\ell_n, \boldsymbol{a^k})$ of transitions. We say that a run *starts in* $(\ell, \boldsymbol{a})$ if $(\ell_0, \boldsymbol{a^0}) = (\ell, \boldsymbol{a})$. A run is *initialized* if it starts in $(\ell_{\text{in}}, \{\bot\}^R)$, and a run is *accepting* if $\ell_k \in \mathcal{L}_{\text{acc}}$. The data language *accepted* by $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set of data words $w \in (\Sigma \times \mathbb{D})^*$ such that there exists an initialized accepting run of $\mathcal{A}$ on w.

We classify register automata into *deterministic register automata* (DRA), *unambiguous register automata* (URA), and *non-deterministic register automata* (NRA). A register automaton is a DRA if for every data word w there is at most one initialized run. A register automaton is a URA if for every data word w there is at most one initialized accepting run. A register automaton without any restriction is an NRA. We say that a data language $L \subseteq (\Sigma \times \mathbb{D})^*$ is DRA-recognizable (URA-recognizable and NRA-recognizable, respectively), if there exists a DRA (URA and NRA, respectively) $\mathcal{A}$ over Σ such that $L(\mathcal{A}) = L$. We write **DRA**, **URA**, and **NRA** for the class of DRA-recognizable, URA-recognizable, and NRA-recognizable, respectively, data languages. Note that **DRA** $\subseteq$ **URA** $\subseteq$ **NRA**. Also note that, albeit a semantical property, the unambiguity of a register automaton can be decided using a simple extension of a product construction, cf. [3].

The *containment problem* is the following decision problem: given two register automata $\mathcal{A}$ and $\mathcal{B}$, does $L(\mathcal{A}) \subseteq L(\mathcal{B})$ hold? We consider two more decision problems that stand in a close relation to the containment problem (namely, they both reduce to the containment problem): the *universality problem* is the question whether $L(\mathcal{B}) = (\Sigma \times \mathbb{D})^*$ for a given register automaton $\mathcal{B}$. The *equivalence problem* is to decide, given two register automata $\mathcal{A}$ and $\mathcal{B}$, whether $L(\mathcal{A}) = L(\mathcal{B})$.

3 Some Facts about Register Automata

For many computational models, a straightforward approach to solve the containment problem is by reduction to the emptiness problem using the equivalence: $L(\mathcal{A}) \subseteq L(\mathcal{B})$ if, and only if, $L(\mathcal{A}) \cap \overline{L(\mathcal{B})} = \emptyset$. This approach proves useful for **DRA**, which is closed under complementation. Using the decidability of the emptiness problem for NRA, as well as the closure of **NRA** under intersection [11], we obtain the decidability of the containment problem for the case that $\mathcal{A}$ is an NRA and $\mathcal{B}$ is a DRA. More precisely, and using results in [5], the containment problem for this particular case is PSPACE-complete.

In contrast to **DRA**, the class **NRA** is not closed under complementation [11] so that the above approach must fail. Indeed, it is well known that the containment problem for the case that $\mathcal{B}$ is an NRA is undecidable [5]. The proof is a reduction from the halting problem for Minsky machines: an NRA is capable to accept the complement of a set of data words encoding halting computations of a Minsky machine.

In this paper, we are interested in the containment problem for the case that $\mathcal{A}$ is an NRA and $\mathcal{B}$ is a URA. When attempting to solve this problem, an obvious idea is to ask whether the class **URA** is closed under complementation. Kaminski and Francez [11] proved that **URA** is *not* closed under complementation, and this even holds for the class of data languages that are accepted by URA that only use a single register. In Figure 1, we show a standard example of a URA for which the complement of the accepted data language cannot even be accepted by an NRA [12]. Intuitively, this automaton is unambiguous because it is not possible for two different runs of the automaton on some data word to reach the location ℓ_1 with the same register valuation at the same time. Therefore, at any time only one run can proceed to the accepting location ℓ_2. Note that this also implies **DRA** $\subsetneq$ **URA**.

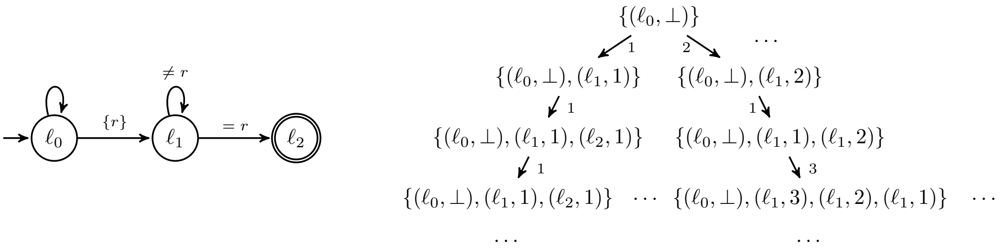
$\neq r$
$\{r\}$
$= r$
ℓ_0
ℓ_1
ℓ_2
$\{(\ell_0,\bot)\}$
1
2
$\{(\ell_0,\bot),(\ell_1,1)\}$
$\{(\ell_0,\bot),(\ell_1,2)\}$
1
1
$\{(\ell_0,\bot),(\ell_1,1),(\ell_2,1)\}$
$\{(\ell_0,\bot),(\ell_1,1),(\ell_1,2)\}$
1
3
$\{(\ell_0,\bot),(\ell_1,1),(\ell_2,1)\}$
$\{(\ell_0,\bot),(\ell_1,3),(\ell_1,2),(\ell_1,1)\}$
$\cdots$

Figure 1 On the left we depict a URA with a single register r and over a singleton alphabet (we omit the labels at the edges). The complement of the data language accepted by this URA cannot be accepted by any NRA. On the right we show a finite part of the infinite state space of the URA.

An alternative approach for solving the containment problem is to explore the (possibly infinite) *synchronized state space of* $\mathcal{A}$ *and* $\mathcal{B}$, cf. [16]. Intuitively, the synchronized state space of $\mathcal{A}$ and $\mathcal{B}$ stores for every state $(\ell, \boldsymbol{a})$ that $\mathcal{A}$ is in after processing a data word w the *set of states* that $\mathcal{B}$ is in after processing the same data word w. For an example, see the computation tree on the right side of Figure 1, where the leftmost branch shows the set of states that the URA on the left side of Figure 1 reaches after processing the data word $(\sigma,1)(\sigma,1)(\sigma,1)$, and the rightmost branch shows the set of states that the URA reaches after processing the data word $(\sigma,2)(\sigma,1)(\sigma,3)$. The key property of the synchronized state space of $\mathcal{A}$ and $\mathcal{B}$ is that it contains sufficient information to decide whether for every data word for which there is an initialized accepting run in $\mathcal{A}$ there is also an initialized accepting run in $\mathcal{B}$. We formalize this intuition in the following paragraphs.

We start by defining the *state space* of a given NRA. Fix an NRA $\mathcal{A} = (R, \mathcal{L}, \ell_{\text{in}}, \mathcal{L}_{\text{acc}}, E)$ over Σ. A *configuration* of $\mathcal{A}$ is a set $C \subseteq (\mathcal{L} \times \mathbb{D}_\bot^R)$ of states of $\mathcal{A}$; if $C = \{(\ell, \boldsymbol{a})\}$ is a singleton set, in slight abuse of notation and if the context is clear, we may omit the parentheses and write $(\ell, \boldsymbol{a})$. Given a configuration C and an input letter $(\sigma, d) \in (\Sigma \times \mathbb{D})$, we use $\text{Succ}_{\mathcal{A}}(C, (\sigma, d))$ to denote the *successor configuration of* C *on the input* (σ, d), formally defined by

$$\text{Succ}_{\mathcal{A}}(C, (\sigma, d)) := \{(\ell, \boldsymbol{a}) \in (\mathcal{L} \times \mathbb{D}_\bot^R) \mid \exists (\ell', \boldsymbol{a}') \in C.(\ell', \boldsymbol{a}') \xrightarrow{\sigma,d}_{\mathcal{A}} (\ell, \boldsymbol{a})\}.$$

For extending this definition to data words, we define inductively $\text{Succ}_{\mathcal{A}}(C, \varepsilon) := C$ and $\text{Succ}_{\mathcal{A}}(C, w \cdot (\sigma, d)) := \text{Succ}_{\mathcal{A}}(\text{Succ}_{\mathcal{A}}(C, w), (\sigma, d))$. We say that a configuration C is *reachable in* $\mathcal{A}$ if there exists some data word w such that $C = \text{Succ}_{\mathcal{A}}((\ell_{\text{in}}, \{\bot\}^R), w)$. We say that a configuration C is *coverable in* $\mathcal{A}$ if there exists some configuration $C' \supseteq C$ such that C' is reachable in $\mathcal{A}$. We say that a configuration C is *accepting* if there exists $(\ell, \boldsymbol{a}) \in C$ such that $\ell \in \mathcal{L}_{\text{acc}}$; otherwise we say that C is *non-accepting*. We define $\text{data}(C) := \bigcup_{(\ell,\boldsymbol{a})} \text{data}(\boldsymbol{a})$ as the set of data occurring in configuration C.

The following proposition follows immediately from the definition of URA.

▶ **Proposition 1.** *If* $\mathcal{A}$ *is a* URA *and* C, C' *are two configurations of* $\mathcal{A}$ *such that* $C \cap C' = \emptyset$ *and* $C \cup C'$ *is coverable, then for every data word* w *the following holds: if* $\text{Succ}_{\mathcal{A}}(C, w)$ *is accepting, then* $\text{Succ}_{\mathcal{A}}(C', w)$ *is non-accepting.*

Let C, C' be two configurations of $\mathcal{A}$. Consider two data words $w = (\sigma_1, d_1) \dots (\sigma_k, d_k)$ and $w' = (\sigma_1, d_1') \dots (\sigma_k, d_k')$ such that $\text{proj}(w) = \text{proj}(w')$. Let $f : \text{data}(C) \cup \text{data}(w) \to \text{data}(C') \cup \text{data}(w')$ be a bijective mapping. We say that C, w and C', w' are *equivalent with respect to* f, written $C, w \sim_f C', w'$, if the following two conditions are satisfied:

(1) $f(C) = C'$, where f maps every datum $d \in \text{data}(C)$ to $f(d)$, and

(2) $f(w) = w'$, where f maps every letter (σ_i, d_i) in w to $(\sigma_i, f(d_i))$.

If $w = w' = \varepsilon$, then we may simply write $C \sim_f C'$. We write $C \sim C'$ if $C \sim_f C'$ for some bijective mapping f.

▶ **Proposition 2.** *If* $C, w \sim C', w'$, *then* $\mathrm{Succ}_{\mathcal{A}}(C, w(0,i]), w(i,k] \sim \mathrm{Succ}_{\mathcal{A}}(C', w'(0,i]), w'(i,k]$ *for all* $0 \leq i \leq k$, *where* $k = |w|$.

Proof. The proof is by induction on i. For the induction base, let $i = 0$. But then $\mathrm{Succ}_{\mathcal{A}}(C, w(0,0])) = \mathrm{Succ}_{\mathcal{A}}(C, \varepsilon) = C$ and $w(0,k] = w$, and similarly for C' and w', so that the statement holds by assumption. For the induction step, let $i > 0$. Define $C_{i-1} := \mathrm{Succ}_{\mathcal{A}}(C, w(0,i-1])$ and similarly C'_{i-1}. By induction hypothesis, there exists some bijective mapping

$$f_{i-1} : \mathrm{data}(C_{i-1}) \cup \mathrm{data}(w(i-1,k]) \to \mathrm{data}(C'_{i-1}) \cup \mathrm{data}(w'(i-1,k])$$

satisfying (1) $f_{i-1}(C_{i-1}) = C'_{i-1}$ and (2) $f_{i-1}(w(i-1,k]) = w'(i-1,k]$. Define $C_i := \mathrm{Succ}_{\mathcal{A}}(C_{i-1}, (\sigma_i, d_i))$ and $C'_i := \mathrm{Succ}_{\mathcal{A}}(C'_{i-1}, (\sigma_i, d'_i))$. Note that $\mathrm{data}(C_i) \subseteq \mathrm{data}(C_{i-1}) \cup \{d_i\}$, and similarly for $\mathrm{data}(C'_i)$. Let f_i be the restriction of f_{i-1} to $\mathrm{data}(C_i) \cup \mathrm{data}(w(i,k])$. We are going to prove that $C_i, w(i,k] \sim_{f_i} C'_i, w'(i,k]$. Note that $f_i(w(i,k]) = w'(i,k]$ holds by definition of f_i and (2). We prove $f_i(C_i) \subseteq C'_i$. Suppose $(\ell, \boldsymbol{a}) \in C_i$. Hence there exists $(\ell_{i-1}, \boldsymbol{b}) \in C_{i-1}$ such that $(\ell_{i-1}, \boldsymbol{b}) \xrightarrow{\sigma_i, d_i} (\ell, \boldsymbol{a})$. Thus there exists an edge $\ell_{i-1} \xrightarrow{\sigma_i, \phi, \lambda} \ell$ such that $\boldsymbol{b}, d_i \models \phi$ and $\boldsymbol{a} = \boldsymbol{b}[\lambda \leftarrow d_i]$. By induction hypothesis, there exists $(\ell_{i-1}, \boldsymbol{b'}) \in C'_{i-1}$ such that $f_{i-1}(\boldsymbol{b}) = \boldsymbol{b'}$. By induction on the structure of ϕ, one can easily prove that $\boldsymbol{b}, d_i \models \phi$ if, and only if, $\boldsymbol{b'}, d'_i \models \phi$. Define $\boldsymbol{a'} := \boldsymbol{b'}[\lambda \leftarrow d'_i]$. We prove $f_i(\boldsymbol{a}) = \boldsymbol{a'}$: there are two cases: (i) If $r \in \lambda$, then $f_i(\boldsymbol{a}(r)) = f_i(d_i) = d'_i = \boldsymbol{a'}(r)$. (ii) If $r \notin \lambda$, then $f_i(\boldsymbol{a}(r)) = f_i(\boldsymbol{b}(r)) = f_{i-1}(\boldsymbol{b}(r)) = \boldsymbol{a'}(r)$. Hence, $f_i(\boldsymbol{a}) = \boldsymbol{a'}$. Altogether $(\ell, f_i(\boldsymbol{a})) \in C'_i$, and thus $f_i(C_i) \subseteq C'_i$. The proof for $C'_i \subseteq f_i(C_i)$ is analogous. Altogether, $C_i, w(i,k] \sim_{f_i} C'_i, w'(i,k]$. ◀

As an immediate consequence of Proposition 2, we obtain that $\sim$ preserves the configuration properties of being *accepting* respectively *non-accepting*.

▶ **Corollary 1.** *Let* C *and* C' *be two configurations of* $\mathcal{A}$. *If* $C, w \sim C', w'$ *and* $\mathrm{Succ}_{\mathcal{A}}(C, w)$ *is non-accepting (accepting, respectively), then* $\mathrm{Succ}_{\mathcal{A}}(C', w')$ *is non-accepting (accepting, respectively).*

Combining the last corollary with the definition of URA, we obtain

▶ **Corollary 2.** *If* $\mathcal{A}$ *is a* URA *and* C, C' *are two configurations such that* $C \cap C' = \emptyset$ *and* $C \cup C'$ *is coverable in* $\mathcal{A}$, *then for every data word* w *such that* $C, w \sim C', w$, *the configurations* $\mathrm{Succ}_{\mathcal{A}}(C, w)$ *and* $\mathrm{Succ}_{\mathcal{A}}(C', w)$ *are non-accepting.*

For the rest of this paper, let $\mathcal{A} = (R^{\mathcal{A}}, \mathcal{L}^{\mathcal{A}}, \ell^{\mathcal{A}}_{\mathrm{in}}, \mathcal{L}^{\mathcal{A}}_{\mathrm{acc}}, E^{\mathcal{A}})$ be an NRA over Σ, and let $\mathcal{B} = (R^{\mathcal{B}}, \mathcal{L}^{\mathcal{B}}, \ell^{\mathcal{B}}_{\mathrm{in}}, \mathcal{L}^{\mathcal{B}}_{\mathrm{acc}}, E^{\mathcal{B}})$ be a URA over Σ. Without loss of generality, we assume $R^{\mathcal{A}} \cap R^{\mathcal{B}} = \emptyset$ and $\mathcal{L}^{\mathcal{A}} \cap \mathcal{L}^{\mathcal{B}} = \emptyset$. We let m be the number of registers of $\mathcal{A}$, and we let n be the number of registers of $\mathcal{B}$.

A *synchronized configuration of* $\mathcal{A}$ *and* $\mathcal{B}$ is a pair $((\ell, \boldsymbol{d}), C)$, where $(\ell, \boldsymbol{d}) \in (\mathcal{L}^{\mathcal{A}} \times \mathbb{D}_{\perp}^{R^{\mathcal{A}}})$ is a single state of $\mathcal{A}$, and $C \subseteq (\mathcal{L} \times \mathbb{D}_{\perp}^{R^{\mathcal{B}}})$ is a configuration of $\mathcal{B}$. Given a synchronized configuration S, we use $\mathrm{data}(S)$ to denote the set $\mathrm{data}(\boldsymbol{d}) \cup \mathrm{data}(C)$ of all data occurring in S. We define $S_{\mathrm{in}} := ((\ell^{\mathcal{A}}_{\mathrm{in}}, \{\perp\}^m), \{(\ell^{\mathcal{B}}_{\mathrm{in}}, \{\perp\}^n)\})$ to be the *initial synchronized configuration of* $\mathcal{A}$ *and* $\mathcal{B}$. We define the *synchronized state space of* $\mathcal{A}$ *and* $\mathcal{B}$ to be the (infinite) state transition system $(\mathbb{S}, \Rightarrow)$, where $\mathbb{S}$ is the set of all synchronized configurations of $\mathcal{A}$ and $\mathcal{B}$, and $\Rightarrow$ is defined as follows. If $S = ((\ell, \boldsymbol{d}), C)$ and $S' = ((\ell', \boldsymbol{d'}), C')$, then $S \Rightarrow S'$ if there exists a letter $(\sigma, d) \in (\Sigma \times \mathbb{D})$ such that $(\ell, \boldsymbol{d}) \xrightarrow{\sigma,d}_{\mathcal{A}} (\ell', \boldsymbol{d'})$, and $\mathrm{Succ}_{\mathcal{B}}(C, (\sigma, d)) = C'$. We say that a synchronized configuration S *reaches a synchronized configuration* S' *in* $(\mathbb{S}, \Rightarrow)$ if there there exists a path in $(\mathbb{S}, \Rightarrow)$ from S to S'. We say that a synchronized configuration S is *reachable in* $(\mathbb{S}, \Rightarrow)$ if S_{in} reaches S. We say that a synchronized configuration $S = ((\ell, \boldsymbol{d}), C)$ is *coverable in* $(\mathbb{S}, \Rightarrow)$ if there exists some synchronized configuration $S' = ((\ell, \boldsymbol{d}), C')$ such that $C' \supseteq C$ and S' is reachable in $(\mathbb{S}, \Rightarrow)$.

We aim to reduce the containment problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$ to a reachability problem in $(\mathbb{S}, \Rightarrow)$. For this, call a synchronized configuration $((\ell, \boldsymbol{d}), C)$ *bad* if $\ell \in \mathcal{L}^{\mathcal{A}}_{\text{acc}}$ is an accepting location and C is non-accepting, i.e., $\ell' \notin \mathcal{L}^{\mathcal{B}}_{\text{acc}}$ for all $(\ell', \boldsymbol{a}) \in C$. The following proposition is easy to prove, cf. [16].

▶ **Proposition 3.** *$L(\mathcal{A}) \subseteq L(\mathcal{B})$ does not hold if, and only if, some bad synchronized configuration is reachable in $(\mathbb{S}, \Rightarrow)$.*

We extend the equivalence relation $\sim$ defined above to synchronized configurations in the natural manner, i.e, we define $S \sim S'$ if there exists a bijective mapping $f : \mathrm{data}(S) \to \mathrm{data}(S')$ such that $f(S) = S'$. Clearly, an analogon of Proposition 2 holds for this extended relation. In particular, we have the following:

▶ **Proposition 4.** *Let S, S' be two synchronized configurations of $(\mathbb{S}, \Rightarrow)$ such that $S \sim S'$. If S reaches a bad synchronized configuration, so does S'.*

Note that the state transition system $(\mathbb{S}, \Rightarrow)$ is infinite. First of all, $(\mathbb{S}, \Rightarrow)$ is not finitely branching: for every synchronized configuration $S = ((\ell, \boldsymbol{d}), C)$ in $\mathbb{S}$, every datum $d \in \mathbb{D}$ may give rise to its own individual synchronized configuration S_d such that $S \Rightarrow S_d$. However, it can be easily seen that for every two different data values $d, d' \in \mathbb{D} \backslash \mathrm{data}(S)$, if inputting (σ, d) gives rise to a transition $S \Rightarrow S_d$ and inputting (σ, d') gives rise to a transition $S \Rightarrow S_{d'}$ (for some $\sigma \in \Sigma$), then $S_d \sim S_{d'}$. Hence there exist synchronized configurations $S_1, \ldots, S_k$ for some $k \in \mathbb{N}$ such that $S \Rightarrow S_i$ for all $i \in \{1, \ldots, k\}$, and such that for all $S' \in \mathbb{S}$ with $S \Rightarrow S'$ there exists $i \in \{1, \ldots, k\}$ such that $S_i \sim S'$. This in why we define in Section 4.3 the notion of abstract configuration, representing synchronized configurations up to the relation $\sim$. Second, and potentially more harmful for the termination of an algorithm to decide the reachability problem from Proposition 3, the configuration C of $\mathcal{B}$ in a synchronized configuration may grow unboundedly. As an example, consider the URA on the left side of Figure 1. For every $k \geq 1$, the configuration $\{(\ell_0, \bot), (\ell_1, d_1), (\ell_1, d_2) \ldots, (\ell_1, d_k)\}$ with pairwise distinct data values $d_1, \ldots, d_k$ is reachable in this URA by inputting the data word $(\sigma, d_1)(\sigma, d_2) \ldots (\sigma, d_k)$. In the next section, we prove that we can solve the reachability problem from Proposition 3 by focussing on a subset of configurations of $\mathcal{B}$ that are bounded in size, thus reducing to a reachability problem on a finite graph.

4 The Containment Problem for Register Automata

4.1 Types

Given $k \in \mathbb{N}$, a *k-type* of $\mathbb{D}_\bot$ is a set $p(\boldsymbol{y}) = p(y_1, \ldots, y_k)$ of first-order formulas over the structure $(\mathbb{D}_\bot, =)$ with free variables among the given k free variables $y_1, \ldots, y_k$ such that for every finite subset $p'(\boldsymbol{y})$ of $p(\boldsymbol{y})$ there exists some $\boldsymbol{a} \in \mathbb{D}_\bot^k$ such that $p'(\boldsymbol{a})$ holds in $(\mathbb{D}_\bot, =)$. A k-type is *complete* if for all formulas $\psi(\boldsymbol{y})$, either $\psi(\boldsymbol{y}) \in p$ or $\neg\psi(\boldsymbol{y}) \in p$.

The structure $(\mathbb{D}_\perp, =)$ is *homogeneous* (i.e., every partial bijection extends to a permutation of the whole domain) so that in particular every complete type is equivalent to a quantifier-free formula (see for example [10, Corollary 6.4.2]). By abuse of notation, we will also call such formulas types. Given $\boldsymbol{a}$, the *type of* $\boldsymbol{a}$, denoted by $\mathrm{tp}(\boldsymbol{a})$, is the set of formulas $\varphi(\boldsymbol{y})$ such that $\varphi(\boldsymbol{a})$ holds in $(\mathbb{D}_\perp, =)$.

Recall that m and n denote the number of registers of $\mathcal{A}$ and $\mathcal{B}$. Let $S = ((\ell, \boldsymbol{d}), C)$ be a synchronized configuration and let $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{D}_\perp^n$ be two register valuations occurring in C, i.e., there exist $\ell, \ell' \in \mathcal{L}^\mathcal{B}$ such that $(\ell, \boldsymbol{a}), (\ell', \boldsymbol{b}) \in C$. We define, for every complete $(2n+m)$-type $\varphi(\boldsymbol{y})$, where $\boldsymbol{y} = (y_1, \ldots, y_{2n+m})$, the set

$$\mathcal{L}_\varphi(\boldsymbol{a}) = \{\ell \in \mathcal{L}^\mathcal{B} \mid \exists (\ell, \boldsymbol{b}) \in C \text{ such that } \varphi(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{d}) \text{ holds in } (\mathbb{D}_\perp, =)\}.$$

We say that $\boldsymbol{a}$ *and* $\boldsymbol{b}$ *are indistinguishable in* S, written $\boldsymbol{a} \equiv_S \boldsymbol{b}$, if $\mathcal{L}_\varphi(\boldsymbol{a}) = \mathcal{L}_\varphi(\boldsymbol{b})$ for every complete $(2n+m)$-type $\varphi(\boldsymbol{y})$. Note that $\boldsymbol{a} \equiv_S \boldsymbol{b}$ implies $\{\ell \in \mathcal{L}^\mathcal{B} \mid (\ell, \boldsymbol{a}) \in C\} = \{\ell \in \mathcal{L}^\mathcal{B} \mid (\ell, \boldsymbol{b}) \in C\}$.

▶ **Example 3.** Let $(\ell^\mathcal{A}, 3)$ be a state in some NRA with a single register, and let $C' = \{(\ell, 1, 3), (\ell, 2, 3), (\ell', 1, 2)\}$ be a configuration of a URA with two registers. Let $S' = ((\ell^\mathcal{A}, 3), C')$ be the corresponding synchronized configuration of $\mathcal{A}$ and $\mathcal{B}$. Consider $\boldsymbol{a} = (1, 3)$ and $\boldsymbol{b} = (2, 3)$. For the 5-type

$$\varphi_1 = (y_1 \neq y_2) \wedge (y_1 \neq y_3) \wedge (y_2 = y_4) \wedge (y_4 = y_5) \wedge (y_3 \neq y_2)$$

we have $\mathcal{L}_{\varphi_1}(\boldsymbol{a}) = \{\ell\}$ as $\varphi_1(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{d})$ holds in $(\mathbb{N}, =)$, and similarly, $\mathcal{L}_{\varphi_1}(\boldsymbol{b}) = \{\ell\}$ as $\varphi_1(\boldsymbol{b}, \boldsymbol{a}, \boldsymbol{d})$ holds in $(\mathbb{N}, =)$. However, we have $\mathcal{L}_{\varphi_2}(\boldsymbol{a}) = \{\ell'\}$ and $\mathcal{L}_{\varphi_2}(\boldsymbol{b}) = \emptyset$ for the 5-type

$$\varphi_2 = (y_1 \neq y_2) \wedge (y_1 = y_3) \wedge (y_2 \neq y_4) \wedge (y_2 = y_5) \wedge (y_4 \neq y_1).$$

Hence $\boldsymbol{a} \equiv_{S'} \boldsymbol{b}$ does *not* hold. However, $\boldsymbol{a} \equiv_S \boldsymbol{b}$ for $S = ((\ell^\mathcal{A}, d), C)$ with $C := C' \cup \{(\ell', 2, 1)\}$.

▶ **Proposition 5.** *Let $S = ((\ell^\mathcal{A}, \boldsymbol{d}), C)$ be a coverable synchronized configuration of $\mathcal{A}$ and $\mathcal{B}$. Let $\boldsymbol{a}, \boldsymbol{b}$ be such that $\boldsymbol{a} \equiv_S \boldsymbol{b}$. Let $C_{\boldsymbol{a}} := \{(\ell, \boldsymbol{a}) \in C \mid \ell \in \mathcal{L}^\mathcal{B}\}$ and $C_{\boldsymbol{b}} := \{(\ell, \boldsymbol{b}) \in C \mid \ell \in \mathcal{L}^\mathcal{B}\}$. Then $C_{\boldsymbol{a}} \sim_f C_{\boldsymbol{b}}$ for the bijective mapping $f : \mathrm{data}(C_{\boldsymbol{a}}) \to \mathrm{data}(C_{\boldsymbol{b}})$ defined by $f(a_i) = b_i$ for all $1 \leq i \leq n$.*

Proof. Let φ be the complete $(2n+m)$-type of $(\boldsymbol{a}, \boldsymbol{a}, \boldsymbol{d})$. Note that for two vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{D}_\perp^n$, $\varphi(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{d})$ holds in $(\mathbb{D}_\perp, =)$ iff $\boldsymbol{u} = \boldsymbol{v}$ and $\mathrm{tp}(\boldsymbol{a}, \boldsymbol{d}) = \mathrm{tp}(\boldsymbol{u}, \boldsymbol{d}) = \mathrm{tp}(\boldsymbol{v}, \boldsymbol{d})$.

Let now $(\ell, \boldsymbol{a})$ be in $C_{\boldsymbol{a}}$. By definition, this means that $\ell \in \mathcal{L}_\varphi(\boldsymbol{a})$. By indiscernibility, $\ell \in \mathcal{L}_\varphi(\boldsymbol{b})$ so that

$$\varphi(\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}) \text{ holds in } (\mathbb{D}_\perp, =) \tag{1}$$

for some $(\ell, \boldsymbol{c}) \in C$. Now, (1) implies $\boldsymbol{b} = \boldsymbol{c}$ and $\mathrm{tp}(\boldsymbol{b}) = \mathrm{tp}(\boldsymbol{a})$. The former implies that $(\ell, \boldsymbol{b}) \in C_{\boldsymbol{b}}$, while the latter implies that f is a bijection. Conversely, we obtain that $(\ell, \boldsymbol{b}) \in C_{\boldsymbol{b}}$ implies $(\ell, \boldsymbol{a}) \in C_{\boldsymbol{a}}$. Hence $f(C_{\boldsymbol{a}}) = C_{\boldsymbol{b}}$, and thus $C_{\boldsymbol{a}} \sim_f C_{\boldsymbol{b}}$. ◀

4.2 Collapsing Configurations

As we pointed out in the introduction, the crucial ingredient of our algorithm for deciding whether $L(\mathcal{A}) \subseteq L(\mathcal{B})$ holds is to prevent configurations C in a synchronized configuration $((\ell, \boldsymbol{d}), C)$ to grow unboundedly. We do this by *collapsing two* subconfigurations $C_{\boldsymbol{a}}, C_{\boldsymbol{b}} \subseteq C$ that behave equivalently with respect to reaching a bad synchronized configuration in $(\mathbb{S}, \Rightarrow)$ into a *single* subconfiguration. The key notions for deciding when two subconfigurations can be collapsed into a single one are *k-types* and *indistinguishability* from the previous subsection.

▶ **Proposition 6.** *Let $S' = ((\ell, \boldsymbol{d}), C')$ be a coverable synchronized configuration of $\mathcal{A}$ and $\mathcal{B}$. Let $\boldsymbol{a}$ and $\boldsymbol{b}$ be two distinct register valuations in C' such that $\boldsymbol{a} \equiv_{S'} \boldsymbol{b}$. Let $C_{\boldsymbol{b}} := \{(\ell, \boldsymbol{b}) \in C' \mid \ell \in \mathcal{L}^{\mathcal{B}}\}$. Then $S := ((\ell, \boldsymbol{d}), C' \setminus C_{\boldsymbol{b}})$ reaches a bad synchronized configuration if, and only if, S' reaches a bad synchronized configuration.*

Proof. The "only if" direction follows from the simple observation that for every data word w, if $\mathrm{Succ}_{\mathcal{B}}(C', w)$ is non-accepting, then so is $\mathrm{Succ}_{\mathcal{B}}(D, w)$ for every subset $D \subseteq C'$. For the "if" direction, let $C_{\boldsymbol{a}} := \{(\ell, \boldsymbol{a}) \in C' \mid \ell \in \mathcal{L}^{\mathcal{B}}\}$ and $C := C' \setminus (C_{\boldsymbol{a}} \cup C_{\boldsymbol{b}})$. Suppose that there exists a data word w such that there exists an accepting run of $\mathcal{A}$ on w that starts in $(\ell, \boldsymbol{d})$, and $\mathrm{Succ}_{\mathcal{B}}(C_{\boldsymbol{a}} \cup C, w)$ is non-accepting. We assume in the following that $\mathrm{Succ}_{\mathcal{B}}(C_{\boldsymbol{b}}, w)$ is accepting; otherwise we are done. Without loss of generality, we assume that $\mathrm{data}(w) \cap \mathrm{data}(S') \subseteq \mathrm{data}(C_{\boldsymbol{b}}) \cup \mathrm{data}(\boldsymbol{d})$. Otherwise, pick for every $d \in \mathrm{data}(w) \cap \mathrm{data}(C_{\boldsymbol{a}} \cup C)$ such that $d \notin \mathrm{data}(C_{\boldsymbol{b}}) \cup \mathrm{data}(\boldsymbol{d})$, a fresh datum $d' \in \mathbb{D}$ not occurring in $\mathrm{data}(w) \cup \mathrm{data}(S')$, and simultaneously replace every occurrence of d in w by d'. Let w' be the resulting data word. Then $(\ell, \boldsymbol{d}), w \sim (\ell, \boldsymbol{d}), w'$ and $C_{\boldsymbol{b}}, w \sim C_{\boldsymbol{b}}, w'$. By Corollary 1, $\mathrm{Succ}_{\mathcal{A}}((\ell, \boldsymbol{d}), w')$ is accepting, and $\mathrm{Succ}_{\mathcal{B}}(C_{\boldsymbol{b}}, w')$ is accepting, too. Then there must exist some accepting run of $\mathcal{A}$ on w' starting in $(\ell, \boldsymbol{d})$, and, by Proposition 1, $\mathrm{Succ}_{\mathcal{B}}(C_{\boldsymbol{a}} \cup C, w')$ must be non-accepting. Hence, we could continue the proof with w' instead of w. Let us assume henceforth that $\mathrm{data}(w) \cap \mathrm{data}(S') \subseteq \mathrm{data}(C_{\boldsymbol{b}}) \cup \mathrm{data}(\boldsymbol{d})$ holds.

Let now w'' be the data word obtained from w as follows: for every $b_i \in \mathrm{data}(w)$ with $b_i \neq a_i$, pick some fresh datum $e_i \in \mathbb{D}$ not occurring in $\mathrm{data}(w) \cup \mathrm{data}(S')$. Then replace every occurrence of the letter b_i in w by e_i.

Note that $(\ell, \boldsymbol{d}), w \sim (\ell, \boldsymbol{d}), w''$: the key argument for this is that by $\boldsymbol{a} \equiv_{S'} \boldsymbol{b}$ we have $b_i \notin \mathrm{data}(\boldsymbol{d})$ whenever $b_i \neq a_i$. By Corollary 1, $\mathrm{Succ}_{\mathcal{A}}((\ell, \boldsymbol{d}), w'')$ is accepting. Hence there must exist some accepting run of $\mathcal{A}$ on w'' starting in $(\ell, \boldsymbol{d})$.

Further note that $C_{\boldsymbol{a}}, w'' \sim C_{\boldsymbol{b}}, w''$: by Proposition 5, $C_{\boldsymbol{a}} \sim_f C_{\boldsymbol{b}}$, where $f : \mathrm{data}(C_{\boldsymbol{a}}) \to \mathrm{data}(C_{\boldsymbol{b}})$ is the bijective mapping defined by $f(a_i) = b_i$ for all $1 \leq i \leq n$. Now let $g : \mathrm{data}(C_{\boldsymbol{a}}) \cup \mathrm{data}(w'') \to \mathrm{data}(C_{\boldsymbol{b}}) \cup \mathrm{data}(w'')$ be the bijective mapping that agrees with f on all data in $\mathrm{data}(C_{\boldsymbol{a}})$, and that maps each datum $d \in \mathrm{data}(w'') \backslash \mathrm{data}(C_{\boldsymbol{a}})$ to d. One can easily see that g is a bijection such that $g(C_{\boldsymbol{a}}) = C_{\boldsymbol{b}}$ and $g(w'') = w''$ so that indeed $C_{\boldsymbol{a}}, w'' \sim_g C_{\boldsymbol{b}}, w''$. By Corollary 2, $\mathrm{Succ}_{\mathcal{B}}(C_{\boldsymbol{a}}, w'')$ and $\mathrm{Succ}_{\mathcal{B}}(C_{\boldsymbol{b}}, w'')$ are non-accepting.

Finally, we prove that $\mathrm{Succ}_{\mathcal{B}}(C, w'')$ is non-accepting, too. For this, let $(\ell', \boldsymbol{c}) \in C$; we prove that $\mathrm{Succ}_{\mathcal{B}}((\ell', \boldsymbol{c}), w'')$ is non-accepting. We distinguish the following two cases:

- For all $1 \leq i \leq n$ with $a_i \neq b_i$ we have $b_i \notin \mathrm{data}(\boldsymbol{c})$. Then $(\ell', \boldsymbol{c}), w \sim (\ell', \boldsymbol{c}), w''$, as witnessed by the bijection

$$f\colon \begin{cases} b_i \mapsto e_i & b_i \in \mathrm{data}(w), b_i \neq a_i \\ e \mapsto e & \text{otherwise.} \end{cases}$$

  for which we have $f(\boldsymbol{c}) = \boldsymbol{c}$ and $f(w) = w''$. Recall that by assumption $\mathrm{Succ}_{\mathcal{B}}((\ell', \boldsymbol{c}), w)$ is non-accepting. By Corollary 1, $\mathrm{Succ}_{\mathcal{B}}((\ell', \boldsymbol{c}), w'')$ is non-accepting.
- There exists $1 \leq i \leq n$ such that $a_i \neq b_i$ and $b_i \in \mathrm{data}(\boldsymbol{c})$.
  Let $\varphi(\boldsymbol{y})$ be the $(2n+m)$-type of $(\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d})$, and note that $\ell' \in \mathcal{L}_{\varphi}(\boldsymbol{b})$. By assumption $\ell' \in \mathcal{L}_{\varphi}(\boldsymbol{a})$ and there exists a state $(\ell', \boldsymbol{c'}) \in C$ such that $\varphi(\boldsymbol{a}, \boldsymbol{c'}, \boldsymbol{d})$ holds. Note that for all $1 \leq j \leq n$ such that $b_i = c_j$ we have $a_i = c'_j$. By assumption, $b_i = c_j$ for some $1 \leq j \leq n$. Since $a_i \neq b_i$, we can infer $c_j \neq c'_j$, and hence $(\ell', \boldsymbol{c}) \neq (\ell', \boldsymbol{c'})$. Next we prove $(\ell', \boldsymbol{c}), w'' \sim (\ell', \boldsymbol{c'}), w''$. We define $f : \mathrm{data}(\boldsymbol{c}) \cup \mathrm{data}(w'') \to \mathrm{data}(\boldsymbol{c'}) \cup \mathrm{data}(w'')$

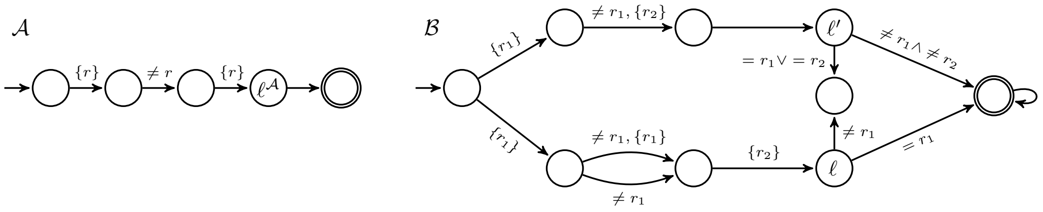

Figure 2 An NRA $\mathcal{A}$ and a URA $\mathcal{B}$ over a singleton alphabet for which $L(\mathcal{A}) \subseteq L(\mathcal{B})$.

as follows:

$$f\colon \begin{cases} c_p \mapsto c'_p & 1 \leq p \leq n \\ e \mapsto e & e \in \text{data}(w'') \end{cases}$$

We prove below that

(i) for all $1 \leq p, q \leq n$, $c_p = c_q$ iff $c'_p = c'_q$;
(ii) for all $1 \leq p \leq n$, for all $e \in \text{data}(w'')$, $e = c_p$ iff $e = c'_p$;

note that this implies that f is well-defined and f is a bijective mapping, and hence $(\ell', \boldsymbol{c}), w'' \sim_f (\ell', \boldsymbol{c'}), w''$. By Proposition 2, $\text{Succ}_{\mathcal{B}}((\ell', \boldsymbol{c}), w'') \sim \text{Succ}_{\mathcal{B}}((\ell', \boldsymbol{c'}), w'')$. By Corollary 2, $\text{Succ}_{\mathcal{B}}((\ell', \boldsymbol{c}), w'')$ and $\text{Succ}_{\mathcal{B}}((\ell', \boldsymbol{c'}), w'')$ are non-accepting. We now prove the two items from above: (i) Follows directly from the fact that $\varphi(\boldsymbol{a}, \boldsymbol{c'}, \boldsymbol{d})$ and $\varphi(\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d})$ hold, which implies that $\boldsymbol{c'}$ and $\boldsymbol{c}$ have the same type. For (ii), recall that $\text{data}(w) \cap \text{data}(S') \subseteq \text{data}(C_{\boldsymbol{b}}) \cup \text{data}(\boldsymbol{d})$. This, the definition of w'', and $\boldsymbol{a} \equiv_{S'} \boldsymbol{b}$ yield the claim.

Altogether, we proved that $\text{Succ}_{\mathcal{B}}(C', w'')$ is non-accepting, while there exists some accepting run $(\ell, \boldsymbol{d}) \longrightarrow^* (\ell'', \boldsymbol{d''})$ of $\mathcal{A}$ on w''. This finishes the proof for the "if" direction. ◀

Before we present our algorithm for deciding the containment problem, we would like to point out that the intuitive notion of *types* alone is not sufficient for deciding whether synchronized configurations can be collapsed. More precisely, given a coverable synchronized configuration $S' = ((\ell^{\mathcal{A}}, \boldsymbol{d}), C')$ and two register valuations $\boldsymbol{a}$ and $\boldsymbol{b}$ that occur in C' and for which $\text{tp}(\boldsymbol{a}, \boldsymbol{d}) = \text{tp}(\boldsymbol{b}, \boldsymbol{d})$, it is in general *not* the case that S' reaches a bad synchronized configuration if $S := ((\ell, \boldsymbol{d}), C' \backslash C_{\boldsymbol{b}})$, where $C_{\boldsymbol{b}} := \{(\ell, \boldsymbol{b}) \in C' \mid \ell \in \mathcal{L}^{\mathcal{B}}\}$, reaches a bad synchronized configuration. To see that, consider Figure 2, where two register automata over a singleton alphabet (we omit the labels at the edges) are depicted: an NRA $\mathcal{A}$ with a single register r on the left side, and a URA $\mathcal{B}$ with two registers r_1 and r_2 on the right side. Note that $L(\mathcal{A}) \subseteq L(\mathcal{B})$. After processing the input data word $w = (\sigma, 1)(\sigma, 2)(\sigma, 3)$, the synchronized configuration $S' = ((\ell^{\mathcal{A}}, 3), C')$, where $C' := \{(\ell, 1, 3), (\ell, 2, 3), (\ell', 1, 2)\})$, is reached in the synchronized state space of $\mathcal{A}$ and $\mathcal{B}$. For $\boldsymbol{a} = (1, 3)$ and $\boldsymbol{b} = (2, 3)$, we have $\text{tp}(\boldsymbol{a}, \boldsymbol{d}) = \text{tp}(\boldsymbol{b}, \boldsymbol{d})$, but $\boldsymbol{a} \equiv_{S'} \boldsymbol{b}$ does not hold (cf. Example 3). Indeed, $\text{Succ}_{\mathcal{B}}(C' \backslash C_{\boldsymbol{b}}, (\sigma, 2))$ is non-accepting, while C' cannot reach any non-accepting configuration.

4.3 Abstract Configurations

In this section, we study synchronized configurations up to the equivalence relation $\sim$. An *abstract synchronized configuration of* $\mathcal{A}$ *and* $\mathcal{B}$ is a tuple (ℓ, C, φ) where φ is a complete $(sn + m)$-type for some $s \in \mathbb{N}$, C is an s-tuple of subsets of $\mathcal{L}^{\mathcal{B}}$, and $\ell \in \mathcal{L}^{\mathcal{A}}$.

The *size* of an abstract synchronized configuration is defined to be $(sn+m)\log(sn+m) + s|\mathcal{L}^{\mathcal{B}}| + \log(|\mathcal{L}^{\mathcal{A}}|)$, which corresponds to the size needed on the tape of a Turing machine to encode an abstract synchronized configuration (where one encodes, for example, an $(sn+m)$-type by giving for each of the $sn+m$ variables, a number in $\{1,\dots,sn+m\}$ in a way that $y_i = y_j$ is a conjunct in φ iff y_i and y_j are assigned the same number).

Note that every synchronized configuration $S = ((\ell^{\mathcal{A}}, \boldsymbol{d}), C)$ gives rise to an abstract synchronized configuration in the following way: let $\boldsymbol{a}^1,\dots,\boldsymbol{a}^s$ be the distinct register valuations in C, listed in some arbitrary order. Let φ be the complete $(sn+m)$-type of $(\boldsymbol{a}^1,\dots,\boldsymbol{a}^s,\boldsymbol{d})$. Let $C_{\boldsymbol{a}^i} := \{\ell \in \mathcal{L}^{\mathcal{B}} \mid (\ell, \boldsymbol{a}^i) \in C\}$. We obtain an abstract synchronized configuration $(\ell^{\mathcal{A}}, (C_{\boldsymbol{a}^1},\dots,C_{\boldsymbol{a}^s}), \varphi)$. Different enumerations of the register valuations of C can yield different abstract configurations. We let $\mathrm{abs}(S)$ be the set of all abstract synchronized configurations that can be obtained from S. Every two abstract synchronized configurations in $\mathrm{abs}(S)$ can be obtained from one another by permuting the variables from the type and the entries from the tuple accordingly. It is easy to prove that $S \sim S'$ if, and only if, $\mathrm{abs}(S) = \mathrm{abs}(S')$.

We say that S is *maximally collapsed* if for all pairs $\boldsymbol{a}$ and $\boldsymbol{b}$ of register valuations appearing in C we have that $\boldsymbol{a} \equiv_S \boldsymbol{b}$ does *not* hold. An abstract configuration (ℓ, C, φ) is said to be maximally collapsed if there exists a synchronized configuration S such that $(\ell, C, \varphi) \in \mathrm{abs}(S)$ and such that S is maximally collapsed (equivalently, one could ask that *every* S such that $(\ell, C, \varphi) \in \mathrm{abs}(S)$ is maximally collapsed). The main result of this section is that the number of different register valuations in a maximally collapsed synchronized configuration is bounded. Let B_r be the number of r-types, which is also called the Bell number of order r. Note that B_r is bounded above by r^r.

▶ **Proposition 7.** *Let $S = ((\ell^{\mathcal{A}}, \boldsymbol{d}), C)$ be a maximally collapsed synchronized configuration of $\mathcal{A}$ and $\mathcal{B}$. The number of different register valuations appearing in C is bounded by $(B_{2n+m} \cdot 2^{|\mathcal{L}^{\mathcal{B}}|})^{(2n+m)^n}$.*

Proof. We first prove a slightly worse upper bound, to give an idea of the proof. Let $K := B_{2n+m}$. We prove that the number of different register valuations is bounded by $2^{|\mathcal{L}^{\mathcal{B}}|K}$. To prove the upper bound, associate with every register valuation $\boldsymbol{a}$ appearing in C the K-tuple $(L_{\varphi_1}(\boldsymbol{a}),\dots,L_{\varphi_K}(\boldsymbol{a}))$ of subsets of $\mathcal{L}^{\mathcal{B}}$, where $\varphi_1,\dots,\varphi_K$ is an enumeration of all the complete $(2n+m)$-types. Note that there are at most $2^{|\mathcal{L}^{\mathcal{B}}|K}$ such tuples. Suppose by contradiction that S contains more than $2^{|\mathcal{L}^{\mathcal{B}}|K}$ different different register valuations. By the pigeonhole principle there are two different register valuations $\boldsymbol{a}$ and $\boldsymbol{b}$ that have the same associated K-tuple. Note that if $\boldsymbol{a}$ and $\boldsymbol{b}$ share the same K-tuple, then $\boldsymbol{a} \equiv_S \boldsymbol{b}$. By Proposition 6, S could be collapsed further, contradiction. Hence, we proved an upper bound of $2^{|\mathcal{L}^{\mathcal{B}}|K}$ on the number of different register valuations appearing in a given maximally collapsed synchronized configuration.

We now proceed to prove the actual bound. The important fact is that when $\boldsymbol{a}$ and $\boldsymbol{d}$ are fixed in S, then few (i.e., $\leq (2n+m)^n$) entries in the tuple $(L_{\varphi_1}(\boldsymbol{a}),\dots,L_{\varphi_K}(\boldsymbol{a}))$ are non-empty. Indeed, in a given $(2n+m)$-type, each of the variables $y_{n+1},\dots,y_{2n}$ can be constrained to be equal to one of $y_1,\dots,y_n,y_{2n+1},\dots,y_{2n+m}$, or constrained to be different than all of them.

Therefore, it remains to bound the number of K-tuples with entries in $2^{\mathcal{L}^{\mathcal{B}}}$ and with at most $(2n+m)^n$ non-empty entries. Each such tuple is characterised by the subset $T \subseteq \{1,\dots,K\}$ of entries that are non-empty, together with a $|T|$-tuple of non-empty subsets of $\mathcal{L}^{\mathcal{B}}$. Since $|T|$ can be bounded by $(2n+m)^n$, we obtain that there are at most $K^{(2n+m)^n} \cdot 2^{|\mathcal{L}^{\mathcal{B}}|(2n+m)^n}$ possible tuples, and thus at most $(B_{2n+m} \cdot 2^{|\mathcal{L}^{\mathcal{B}}|})^{(2n+m)^n}$ different register valuations. ◀

Note that the bound in Proposition 7 is doubly exponential in n and exponential in $|\mathcal{L}^{\mathcal{B}}|$

and m. As a direct corollary, we obtain a bound on the number of maximally collapsed abstract synchronized configurations.

▶ **Proposition 8.** *The number of maximally collapsed abstract configurations is bounded by a triple exponential in $|\mathcal{A}|$ and $|\mathcal{B}|$. If n is fixed, then this number is bounded by a double exponential in $|\mathcal{A}|$ and $|\mathcal{B}|$.*

Proof. By Proposition 7, a maximally collapsed synchronized configuration $S = ((\ell^{\mathcal{A}}, \boldsymbol{d}), C)$ is such that C contains at most $K := (B_{2n+m} \cdot 2^{|\mathcal{L}^{\mathcal{B}}|})^{(2n+m)^n}$ different register valuations. Therefore, any abstract synchronized configuration in $\mathrm{abs}(S)$ is described by an $(sn+m)$-type with $s \leq K$. For a given s, there are at most $B_{sn+m} \cdot |\mathcal{L}^{\mathcal{B}}|^s \cdot |\mathcal{L}^{\mathcal{A}}|$ different abstract synchronized configurations. Summing up from $s = 0$ to K, we obtain that there are at most

$$\begin{aligned}\sum_{s=0}^{K} B_{sn+m} \cdot |\mathcal{L}^{\mathcal{B}}|^s \cdot |\mathcal{L}^{\mathcal{A}}| &\leq |\mathcal{L}^{\mathcal{A}}| \cdot \left(B_m + B_{n+m}|\mathcal{L}^{\mathcal{B}}| + \cdots + B_{nK+m} \cdot |\mathcal{L}^{\mathcal{B}}|^K\right) \\ &\leq |\mathcal{L}^{\mathcal{A}}| \cdot (1+K) \cdot B_{nK+m} \cdot |\mathcal{L}^{\mathcal{B}}|^K \\ &\leq |\mathcal{L}^{\mathcal{A}}| \cdot (1+K) \cdot (nK+m)^{(nK+m)} \cdot |\mathcal{L}^{\mathcal{B}}|^K\end{aligned}$$

maximally collapsed abstract synchronized configurations. Since K is doubly exponential in $|\mathcal{A}|$ and $|\mathcal{B}|$, this gives the first result. The second result follows from the fact that for fixed n, K only depends exponentially on m and $|\mathcal{L}^{\mathcal{B}}|$. ◀

Given abstract synchronized configurations $(\ell^{\mathcal{A}}, C, \varphi)$ and $(\ell'^{\mathcal{A}}, C', \varphi')$, define $(\ell^{\mathcal{A}}, C, \varphi) \rightsquigarrow (\ell'^{\mathcal{A}}, C', \varphi')$ if there exist synchronized configurations S and S' such that:

- $S \Rightarrow S'$,
- $(\ell^{\mathcal{A}}, C, \varphi)$ is in $\mathrm{abs}(S)$,
- S' can be maximally collapsed to some S'' such that $(\ell'^{\mathcal{A}}, C', \varphi')$ is in $\mathrm{abs}(S'')$.

▶ **Lemma 4.** *Given two abstract synchronized configurations $(\ell^{\mathcal{A}}, C, \varphi)$ and $(\ell'^{\mathcal{A}}, C', \varphi')$, deciding whether $(\ell^{\mathcal{A}}, C, \varphi) \rightsquigarrow (\ell'^{\mathcal{A}}, C', \varphi')$ holds can be done in polynomial space.*

Proof. In this proof, we assume without loss of generality that $\mathbb{D} = \mathbb{N}$. Let s be such that φ is an $(sn+m)$-type. Note that there is a synchronized configuration S of the form $((\ell^{\mathcal{A}}, \boldsymbol{d}), D)$ such that $\mathrm{data}(D) \cup \mathrm{data}(\boldsymbol{d}) \subseteq \{1, \ldots, sn+m\}$ and such that $(\ell^{\mathcal{A}}, C, \varphi) \in \mathrm{abs}(S)$ This S is moreover computable in polynomial space.

To decide whether $(\ell^{\mathcal{A}}, C, \varphi) \rightsquigarrow (\ell'^{\mathcal{A}}, C', \varphi')$ holds, one simply:

- guesses a letter $\sigma \in \Sigma$ and a datum d in $\{1, \ldots, sn+m+1\}$,
- computes a synchronized configuration S' obtained by firing the transition corresponding to (σ, d) from S,
- guesses a sequence $(\boldsymbol{a}^1, \boldsymbol{b}^1), \ldots, (\boldsymbol{a}^r, \boldsymbol{b}^r)$ of register valuations such that Proposition 6 can be applied r times to obtain a maximally collapsed configuration S'',
- checks that $(\ell'^{\mathcal{A}}, C', \varphi')$ is in $\mathrm{abs}(S'')$.

At the second step, the size of S' is polynomially bounded by the size of $\mathcal{A}$, $\mathcal{B}$, and of S. Moreover, the maximal length of a collapsing sequence in the third step is also polynomially bounded, as the number of distinct register valuations decreases after each application of Proposition 6. Therefore, this algorithm uses a polynomial amount of space. ◀

As for synchronized configuration, an abstract synchronized configuration $(\ell^{\mathcal{A}}, C, \varphi)$ is called *bad* if $\ell^{\mathcal{A}}$ is an accepting location and none of the states in C contains an accepting location.

▶ **Proposition 9.** *A bad synchronized configuration is reachable in* $(\mathbb{S}, \Rrightarrow)$ *if, and only if, a bad abstract synchronized configuration is reachable from* $\mathrm{abs}(S_{\mathrm{in}})$.

Proof. We prove that for every coverable synchronized configuration S and every $n \geq 0$, a bad synchronized configuration is reachable in n steps from S if, and only if, a bad abstract synchronized configuration is reachable in n steps from $\mathrm{abs}(S)$. The statement then follows by taking $S := S_{\mathrm{in}}$. The proof goes by induction on n, where the case $n = 0$ is trivial in both directions.

Suppose now that S reaches a bad synchronized configuration in n steps. Let S' be such that $S \Rrightarrow S'$ and such that S' reaches a bad synchronized configuration in $n-1$ steps. Let S'' be such that S' can be maximally collapsed to S''. By iterating Proposition 6, we have that S'' reaches a bad synchronized configuration in $n-1$ steps (the fact that the length of the path is unchanged can be seen from the proof of Proposition 6). It follows from the induction hypothesis that some $(\ell', C', \varphi') \in \mathrm{abs}(S'')$ reaches a bad abstract synchronized configuration in $n-1$ steps. Let (ℓ, C, φ) be an arbitrary abstraction in $\mathrm{abs}(S)$. We have by definition $(\ell, C, \varphi) \rightsquigarrow (\ell', C', \varphi')$, so that (ℓ, C, φ) reaches a bad abstract synchronized configuration in n steps.

The converse direction is proved similarly. ◀

Finally, we are able to present the main theorem.

▶ **Theorem 5.** *The containment problem* $L(\mathcal{A}) \subseteq L(\mathcal{B})$, *where* $\mathcal{A}$ *is a non-deterministic register automaton and* $\mathcal{B}$ *is an unambiguous register automaton, is in* 2-EXPSPACE. *If the number of registers of* $\mathcal{B}$ *is fixed, the problem is in* EXPSPACE.

Proof. We use the classical non-deterministic logspace algorithm for reachability on the graph of maximally collapsed abstract synchronized configurations. Every node of the graph can be stored using double-exponential space (see the second paragraph at the beginning of Section 4.3), and the size of the graph is triply exponential in the size of $\mathcal{A}$ and $\mathcal{B}$ by Proposition 8. Moreover, the relation $\rightsquigarrow$ is decidable in polynomial space by Lemma 4. Therefore, we obtain that the algorithm uses at most a double-exponential amount of space. In case the number of registers of $\mathcal{B}$ is fixed, Proposition 8 implies that the size of the graph is doubly exponential in the size of $\mathcal{A}$ and $\mathcal{B}$. We obtain that the algorithm uses at most an exponential amount of space. ◀

As an immediate corollary of Theorem 5, we obtain that the universality problem and the equivalence problem for unambiguous register automata are in 2-EXPSPACE.

5 Open Problems

The most obvious problem is to figure out the *exact* computational complexity of the containment problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$, when $\mathcal{B}$ is an URA. Finding lower bounds for unambiguous automata is a hard problem. Techniques for proving lower complexity bounds of the containment problem (respectively the universality problem) for the case that $\mathcal{B}$ is a non-deterministic automaton rely heavily on non-determinism (cf. Theorem 5.2 in [5]); as was already pointed out in [3], we are lacking techniques for finding lower computational complexity bounds for the case that $\mathcal{B}$ is unambiguous, even for the class of finite automata. Concerning the upper bound, computer experiments revealed that maximally collapsed synchronized configurations seem to remain small. Based on these experiments, we believe that the bound in Proposition 7 is not optimal and can be improved to $O(2^{poly(n,m,|\mathcal{L}^{\mathcal{B}}|)})$.

If this is correct, we would obtain an EXPSPACE upper-bound for the general containment problem.

We also would like to study to what extent our techniques can be used to solve the containment problem for other computation models. In particular, we are interested in the following:

- One can extend the definition of register automata to work over an ordered domain, where the register constraints are of the form $< r$ and $> r$. Proposition 6 turns out to be false in this setting, but it seems plausible that there exists a collapsibility notion that would work for this model.
- An automaton $\mathcal{B}$ is said to be k-ambiguous if it has at most k accepting runs for every input data word, and polynomially ambiguous if the number of accepting runs for some input data word w is bounded by $p(|w|)$ for some polynomial p. Again, it is likely that simple modifications of Proposition 6 would give an algorithm for the containment problem for k-ambiguous register automata.
- Last but not least, we would like to point out that our techniques cannot directly be applied to the class of unambiguous register automata with guessing which we mentioned in the introduction. Thus, the respective containment problem remains open for future research.

References

1 Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors. *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. URL: http://www.dagstuhl.de/dagpub/978-3-95977-076-7.

2 Thomas Colcombet. Forms of determinism for automata (invited talk). In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*, volume 14 of *LIPIcs*, pages 1–23. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012. URL: https://doi.org/10.4230/LIPIcs.STACS.2012.1, doi:10.4230/LIPIcs.STACS.2012.1.

3 Thomas Colcombet. Unambiguity in automata theory. In Jeffrey Shallit and Alexander Okhotin, editors, *Descriptional Complexity of Formal Systems - 17th International Workshop, DCFS 2015, Waterloo, ON, Canada, June 25-27, 2015. Proceedings*, volume 9118 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2015. URL: https://doi.org/10.1007/978-3-319-19225-3_1, doi:10.1007/978-3-319-19225-3_1.

4 Laure Daviaud, Marcin Jurdzinski, Ranko Lazic, Filip Mazowiecki, Guillermo A. Pérez, and James Worrell. When is containment decidable for probabilistic automata? In Chatzigiannakis et al. [1], pages 121:1–121:14. URL: https://doi.org/10.4230/LIPIcs.ICALP.2018.121, doi:10.4230/LIPIcs.ICALP.2018.121.

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

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

7 Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen. Ackermannian and primitive-recursive bounds with dickson's lemma. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24,*

2011, Toronto, Ontario, Canada, pages 269–278. IEEE Computer Society, 2011. URL: http://dx.doi.org/10.1109/LICS.2011.39, doi:10.1109/LICS.2011.39.

8 Diego Figueira, Piotr Hofman, and Slawomir Lasota. Relating timed and register automata. In Sibylle B. Fröschle and Frank D. Valencia, editors, *Proceedings 17th International Workshop on Expressiveness in Concurrency, EXPRESS'10, Paris, France, August 30th, 2010.*, volume 41 of *EPTCS*, pages 61–75, 2010. URL: http://dx.doi.org/10.4204/EPTCS.41.5, doi:10.4204/EPTCS.41.5.

9 Nathanaël Fijalkow, Cristian Riveros, and James Worrell. Probabilistic automata of bounded ambiguity. In Roland Meyer and Uwe Nestmann, editors, *28th International Conference on Concurrency Theory, CONCUR 2017, September 5-8, 2017, Berlin, Germany*, volume 85 of *LIPIcs*, pages 19:1–19:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. URL: https://doi.org/10.4230/LIPIcs.CONCUR.2017.19, doi:10.4230/LIPIcs.CONCUR.2017.19.

10 Wilfrid Hodges. *A shorter model theory.* Cambridge University Press, Cambridge, 1997.

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

12 Michael Kaminski and Daniel Zeitlin. Finite-memory automata with non-deterministic reassignment. *International Journal of Foundations of Computer Science*, Volume 21, Issue 05, 2010.

13 Hing Leung. Descriptional complexity of nfa of different ambiguity. *Int. J. Found. Comput. Sci.*, 16(5):975–984, 2005. URL: https://doi.org/10.1142/S0129054105003418, doi:10.1142/S0129054105003418.

14 Michał Skrzypczak. Unambiguous languages exhaust the index hierarchy. In Chatzigiannakis et al. [1], pages 140:1–140:14. URL: https://doi.org/10.4230/LIPIcs.ICALP.2018.140, doi:10.4230/LIPIcs.ICALP.2018.140.

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

16 Joël Ouaknine and James Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings*, pages 54–63. IEEE Computer Society, 2004. URL: https://doi.org/10.1109/LICS.2004.1319600, doi:10.1109/LICS.2004.1319600.

17 Mikhail Raskin. A superpolynomial lower bound for the size of non-deterministic complement of an unambiguous automaton. In Chatzigiannakis et al. [1], pages 138:1–138:11. URL: https://doi.org/10.4230/LIPIcs.ICALP.2018.138, doi:10.4230/LIPIcs.ICALP.2018.138.

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

19 Luc Segoufin. Automata and logics for words and trees over an infinite alphabet. In Zoltán Ésik, editor, *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, volume 4207 of *Lecture Notes in Computer Science*, pages 41–57. Springer, 2006. URL: https://doi.org/10.1007/11874683_3, doi:10.1007/11874683_3.