Theory of Computing Systems
https://doi.org/10.1007/s00224-020-09997-2

# The Containment Problem for Unambiguous Register Automata and Unambiguous Timed Automata

Antoine Mottet[1] · Karin Quaas[2]

Published online: 15 September 2020

**Abstract**
We investigate the complexity of the containment problem "Does $L(\mathcal{A}) \subseteq L(\mathcal{B})$ hold?" for register automata and timed automata, where $\mathcal{B}$ is assumed to be unambiguous and $\mathcal{A}$ is arbitrary. We prove that the problem is decidable in the case of register automata over $(\mathbb{N}, =)$, in the case of register automata over $(\mathbb{Q}, <)$ when $\mathcal{B}$ has a single register, and in the case of timed automata when $\mathcal{B}$ has a single clock. We give a 2-EXPSPACE algorithm in the first case, whose complexity is a single exponential in the case that $\mathcal{B}$ has a bounded number of registers. In the other cases, we give an EXPSPACE algorithm.

**Keywords** Containment · Unambiguous automata · Register automata · Timed automata

## 1 Introduction

Register automata [10] are a widely studied model of computation that extend finite automata with finitely many *registers* that are able to hold values from an infinite

This article belongs to the Topical Collection: *Special Issue on Theoretical Aspects of Computer Science (2019)*
Guest Editors: Rolf Niedermeier and Christophe Paul

The first author received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant Agreement No 771005, CoCoSym).

The second author was supported by Deutsche Forschungsgemeinschaft (DFG), Project 406907430

✉ Antoine Mottet
mottet@karlin.mff.cuni.cz

[1] Department of Algebra, Faculty of Mathematics and Physics, Charles University in Prague, Prague, Czech Republic

[2] Universität Leipzig, Leipzig, Germany

domain and perform 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 $\Sigma$ is a finite alphabet and $\mathbb{D}$ is an infinite set called the data domain. Typical examples of such data domains are the non-negative integers with equality $(\mathbb{N};=)$, or the rational numbers with the order $(\mathbb{Q};<)$. 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 [21]. 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 non-deterministic register automata, the containment problem is undecidable [5, 16], even if it is defined over a simple data structure like the non-negative integers with equality. 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 [5, 10]. However, in this setting, the problem is Ackermann-complete [7]; it can therefore hardly be considered tractable.

This motivates the study of *unambiguous* register automata, which are non-deterministic register automata for which every data word has at most one accepting run. Such automata are strictly more expressive than deterministic register automata [10, 11].

In the present paper, we investigate the complexity of the containment problem when $\mathcal{B}$ is restricted to be an unambiguous register automaton. We start with register automata defined over $(\mathbb{N};=)$. We prove that for such register automata the problem is decidable with a 2-EXPSPACE complexity, and is even decidable in EXPSPACE if the number of registers of $\mathcal{B}$ is a fixed constant. This is a striking difference to the non-deterministic case, where even for a fixed number of registers greater than one the problem is undecidable. Let us give a brief explanation how we achieve this result. Classically, one way to approach the containment problem (for general models of computation) is to reduce it to a reachability problem on an infinite state transition system, called the *synchronized state space of $\mathcal{A}$ and $\mathcal{B}$*, cf. [17]. 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 (infinite) 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 $\mathcal{B}$.

In the second part of the paper, we aim to solve the containment problem for register automata over $(\mathbb{Q}; <)$. Unfortunately, it turns out that the methods we developed for register automata over $(\mathbb{N}; =)$ are in general not applicable to this case. We present, however, an algorithm for deciding $L(\mathcal{A}) \subseteq L(\mathcal{B})$ for the simpler case where $\mathcal{B}$ is an unambiguous register automaton with a single register, which runs in EXPSPACE, as in the single-register case for register automata over $(\mathbb{N}, =)$. Finally, and very much related to that, we aim to apply our method to solving the containment problem for *timed automata*. A timed automaton is a finite automaton extended with a finite set of real-valued clocks that are used to measure the time that the automaton spends in a location. The edges of such automata are labeled with time constraints that determine when the automaton may take the edge. While both the syntax and the semantics of timed automata and register automata appear in different shapes, it was recently established [8] that there is a tight relationship between the two automata models. Using a very similar method as for register automata over $(\mathbb{Q}; <)$, we show that the containment problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$ is in EXPSPACE if $\mathcal{B}$ is an unambiguous timed automaton with a single clock.

**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 that are close to the model originally introduced by Kaminski and Francez [10]. 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 [5, 16, 19]. In [8], register automata over *ordered* data domains are studied.

Kaminski and Zeitlin [11] define a generalisation of the model in [10], 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 [10]. 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 Fig. 1 (Example 4 in [11]). Figueira [6] studies an alternating version of this model, also over ordered data domains. Colcombet [2, 3] 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

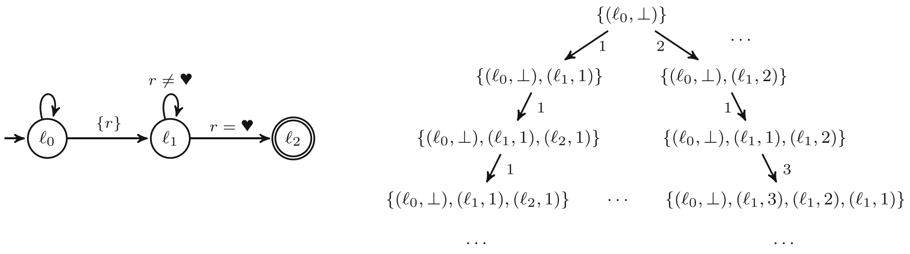
$r \neq ♥$
$\{r\}$
$r = ♥$
$\ell_0$
$\ell_1$
$\ell_2$
$\{(\ell_0, \bot)\}$
$\{(\ell_0, \bot), (\ell_1, 1)\}$
$\{(\ell_0, \bot), (\ell_1, 2)\}$
$\{(\ell_0, \bot), (\ell_1, 1), (\ell_2, 1)\}$
$\{(\ell_0, \bot), (\ell_1, 1), (\ell_1, 2)\}$
$\{(\ell_0, \bot), (\ell_1, 1), (\ell_2, 1)\}$
$\{(\ell_0, \bot), (\ell_1, 3), (\ell_1, 2), (\ell_1, 1)\}$

**Fig. 1** On the left we depict a URA over $(\mathbb{N}; =)$ and a singleton alphabet (we omit the labels at the edges) with a single register $r$. 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

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 [4, 9, 13, 18]. 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 [12] while the fundamental problems (such as emptiness, universality, containment, equivalence) remain tractable.

An extended abstract of this article has appeared in the Proceedings of the 36th International Symposium on Theoretical Aspects of Computer Science, (STACS) 2019 [14]. In comparison with the extended abstract, we improved the presentation of the main technical achievements for register automata defined over the non-negative integers with equality, and we added new results concerning register automata defined over the rationals with order and timed automata.

## 2 Main Definitions

Throughout the paper, $\Sigma$ denotes a finite alphabet, and $\mathcal{D}$ denotes a relational structure $(\mathbb{D}; R_1^{k_1}, \ldots, R_p^{k_p})$ with infinite domain $\mathbb{D}$ and $k_i$-ary relations $R_i^{k_i}$ over $\mathbb{D}$. In this paper, we are mainly interested in the relational structures $(\mathbb{N}; =)$ of the non-negative integers with equality, and $(\mathbb{Q}; <)$ of the rational numbers with the usual order relation.

A *partial isomorphism* of a relational structure $(\mathbb{D}; R_1^{k_1}, \ldots, R_p^{k_p})$ is an injective map $f : D \to \mathbb{D}$ with finite we have $(a_1, \ldots, a_{k_i}) \in R_i^{k_i}$ if and only if $(f(a_1), \ldots, f(a_{k_i})) \in R_i^{k_i}$ for all $a_1, \ldots, a_{k_i} \in D$. Thus, partial isomorphisms of $(\mathbb{N}; =)$ are injective maps $D \to \mathbb{N}$ with finite domain $D$, and partial isomorphisms of $(\mathbb{Q}; <)$ are increasing maps $D \to \mathbb{Q}$ with finite domain $D$.

A *data word* is a finite sequence $(\sigma_1, d_1) \ldots (\sigma_k, d_k) \in (\Sigma \times \mathbb{D})^*$. A *data language* is a set of data words. We use $\varepsilon$ 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 $w(i, j] := (\sigma_{i+1}, d_{i+1}) \ldots (\sigma_j, d_j)$. Note that $w(i, i] = \varepsilon$. We use $\text{data}(w)$

to denote the set $\{d_1, \ldots, d_k\}$ of all data occurring in $w$. We use $\mathrm{proj}(w)$ to denote the projection of $w$ onto $\Sigma^*$, i.e., the word $\sigma_1 \ldots \sigma_k$.

In register automata (defined below), registers take values in $\mathbb{D}$. For technical reasons, the value of a register may also be undefined. We use some fresh element $\bot \notin \mathbb{D}$ not occurring in $\mathbb{D}$ to denote that the value of a register is undefined, and we define $\mathbb{D}_\bot := \mathbb{D} \cup \{\bot\}$. Whenever needed, we implicitly assume that the relational structure $\mathcal{D}$ uses the domain $\mathbb{D}_\bot$ and an additional unary relation $\{\bot\}$. Moreover, we assume that $\bot$ does not occur in any tuple of the original relations of $\mathcal{D}$.

We use boldface lower-case letters like $\boldsymbol{a}, \boldsymbol{b}, \ldots$ to denote tuples in $\mathbb{D}_\bot^n$, where $n \in \mathbb{N}$. Given a tuple $\boldsymbol{a} \in \mathbb{D}_\bot^n$, we write $a_i$ for its $i$-th component, and $\mathrm{data}(\boldsymbol{a})$ denotes the set $\{a_1, \ldots, a_n\} \subseteq \mathbb{D}_\bot$ of all data occurring in $\boldsymbol{a}$.

Let $\mathcal{R} = \{r_1, \ldots, r_n\}$ be a finite set of *registers*. A *register valuation* is a mapping $\boldsymbol{a} : \mathcal{R} \to \mathbb{D}_\bot$; we may write $a_i$ as shorthand for $\boldsymbol{a}(r_i)$. Let $\mathbb{D}_\bot^{\mathcal{R}}$ denote the set of all register valuations. Given $\lambda \subseteq \mathcal{R}$ 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 $\mathcal{R}$ is defined by the grammar

$$\phi ::= \mathtt{true} \mid R(t_1, \ldots, t_k) \mid \neg\phi \mid \phi \wedge \phi\,,$$

where each $t_i$ is either a placeholder $\heartsuit$ (representing the current input datum) or some $r \in \mathcal{R}$, and $R$ is a $k$-ary relation symbol from $\mathcal{D}$. We use $\Phi(\mathcal{R})$ to denote the set of all register constraints over $\mathcal{R}$. We may use $\phi_1 \vee \phi_2$ as shorthand for $\neg(\neg\phi_1 \wedge \neg\phi_2)$. The satisfaction relation $\models$ for $\Phi(\mathcal{R})$ on $\mathbb{D}_\bot^{\mathcal{R}} \times \mathbb{D}$ is defined by structural induction, where the base case is $\boldsymbol{a}, d \models R(t_1, \ldots, t_k)$ if and only if $(t'_1, \ldots, t'_k) \in R$, where

$$t'_i = \begin{cases} a_j & \text{if } t_i = r_j \\ d & \text{if } t_i = \heartsuit. \end{cases}$$

For example, over the relational structure $(\mathbb{N}; =)$, we have $(2, 3, 4), 3 \models r_2 = \heartsuit$ and $(3, 3), 2 \models r_1 = r_2$.

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

- $\mathcal{R}$ 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
- $E \subseteq \mathcal{L} \times \Sigma \times \Phi(\mathcal{R}) \times 2^{\mathcal{R}} \times \mathcal{L}$ is a finite set of *edges*. Given an edge $(\ell, \sigma, \phi, \lambda, \ell') \in E$, $\sigma$ is the label of the edge, $\phi$ is the register constraint of the edge, and $\lambda$ is the set of registers to be updated. A register constraint $\mathtt{true}$ is vacuously true and may be omitted; likewise we may omit $\lambda$ if $\lambda = \emptyset$.

A *state* of $\mathcal{A}$ is a pair $(\ell, \boldsymbol{a}) \in \mathcal{L} \times \mathbb{D}_\bot^{\mathcal{R}}$, where $\ell$ 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 $(\ell, \sigma, \phi, \lambda, \ell') \in E$ 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 a data word $(\sigma_1, d_1) \dots (\sigma_k, d_k)$ is a sequence $(\ell_0, \boldsymbol{a^0}) \xrightarrow{\sigma_1,d_1} (\ell_1, \boldsymbol{a^1}) \xrightarrow{\sigma_2,d_2} \dots \xrightarrow{\sigma_k,d_k} (\ell_k, \boldsymbol{a^k})$ 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 $(\ell_{\text{in}}, \bot)$, where $\bot$ is the function assigning $\bot$ to every register, 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-acceptable (URA-acceptable and NRA-acceptable, respectively), if there exists a DRA (URA and NRA, respectively) $\mathcal{A}$ over $\mathcal{D}$ and $\Sigma$ such that $L(\mathcal{A}) = L$. We write **DRA**, **URA**, and **NRA** for the class of DRA-acceptable, URA-acceptable, and NRA-acceptable, 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 a 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 [10], we obtain the decidability of the containment problem for the case where $\mathcal{A}$ 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 [10] so that the above approach must fail if $\mathcal{B}$ is an NRA. Indeed, it is well known that the containment problem for the case where $\mathcal{B}$ is an NRA is undecidable [5]. The proof is a reduction from the halting problem for Minsky machines: an NRA is able 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 where $\mathcal{A}$ 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 [10] 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 Fig. 1, we show a standard example of a URA over $(\mathbb{N}; =)$ and a singleton alphabet for which the complement of the accepted data language cannot even be accepted by an NRA [11]. 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 $\ell_1$ with the same register valuation at the same time. Therefore, at any time only one run can proceed to the accepting location $\ell_2$. Note that this also implies $\mathbf{DRA} \subsetneq \mathbf{URA}$.

An alternative approach for solving the containment problem is to explore the (possibly infinite) *synchronized state space of* $\mathcal{A}$ *and* $\mathcal{B}$, cf. [17]. 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 on the right side of Fig. 1 the computation tree for the URA on the left side of Fig. 1: the leftmost branch shows the set of states that the URA 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.

### 3.1 The State Space of an NRA

We start by defining the *state space* of a given NRA. Consider an NRA $\mathcal{A} = (\mathcal{R}, \mathcal{L}, \ell_{\text{in}}, \mathcal{L}_{\text{acc}}, E)$ over $\mathcal{D}$ and $\Sigma$. A *configuration* of $\mathcal{A}$ is a finite set $C \subseteq (\mathcal{L} \times \mathbb{D}_\perp^{\mathcal{R}})$ 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 curly brackets 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* $(\sigma, d)$, formally defined by

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

In order to extend 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 $C = \text{Succ}_{\mathcal{A}}((\ell_{\text{in}}, \{\perp\}^{\mathcal{R}}), w)$. 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 $\text{data}(C) := \bigcup_{(\ell, \boldsymbol{a}) \in C} \text{data}(\boldsymbol{a})$ as the set of data occurring in the configuration $C$.[1]

[1]We note that this coincides with the notion of *support* for a finite configuration, in the terminology of set theory with atoms. Similarly, some notions that we define hereafter are classical in the "atom" terminology. We choose to ignore this terminology for the sake of clarity.

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* $\mathrm{Succ}_{\mathcal{A}}(C, w)$ *is accepting, then* $\mathrm{Succ}_{\mathcal{A}}(C', w)$ *is non-accepting.*

Let $C, C'$ be two configurations of $\mathcal{A}$, and consider two data words $w = (\sigma_1, d_1) \ldots (\sigma_k, d_k)$ and $w' = (\sigma_1, d'_1) \ldots (\sigma_k, d'_k)$ such that $\mathrm{proj}(w) = \mathrm{proj}(w')$. Let $f$ be a partial isomorphism of $\mathcal{D}$ (c.f. Section 2), and let $C$ be a configuration with $\mathrm{data}(C) \subseteq \mathrm{dom}(f)$. We define $f(C) := \{(\ell, f(d_1), \ldots, f(d_{|\mathcal{R}|})) \mid (\ell, d_1, \ldots, d_{|\mathcal{R}|}) \in C\}$; likewise, if $\{d_1, \ldots, d_k\} \subseteq \mathrm{dom}(f)$, we define

$$f((\sigma_1, d_1) \ldots (\sigma_k, d_k)) := (\sigma_1, f(d_1)) \ldots (\sigma_k, f(d_k)).$$

Finally, we write $f(C, w) = (C', w')$ if

$$f(C) = C' \text{ and } f(w) = w' \qquad (\star)$$

and $C, w \sim C', w'$ if there exists a partial isomorphism $f$ such that $f(C, w) = (C', w')$. Note that $\sim$ is an equivalence relation.

**Proposition 2** *If $C, w \sim C', w'$, then* $\mathrm{Succ}_{\mathcal{A}}(C, w) \sim \mathrm{Succ}_{\mathcal{A}}(C', w')$.

*Proof* Let $k = |w| = |w'|$. We prove by induction that for all $i \in \{0, \ldots, k\}$, the statement

$$\mathrm{Succ}_{\mathcal{A}}(C, w(0, i]), w(i, k] \sim \mathrm{Succ}_{\mathcal{A}}(C', w'(0, i]), w'(i, k]$$

holds. Note that we obtain the desired result with $i = k$.

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 partial isomorphism

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

satisfying ($\star$) $f_{i-1}(C_{i-1}) = C'_{i-1}$ and $f_{i-1}(w(i-1, k]) = w'(i-1, k]$. 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 $f_i(C_i, w(i, k]) = (C'_i, w'(i, k])$. Note that $f_i(w(i, k]) = w'(i, k]$ holds by the definition of $f_i$ and ($\star$). For the proof of $f_i(C_i) \subseteq C'_i$, suppose $(\ell, \boldsymbol{a}) \in C_i$. 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 $(\ell_{i-1}, \sigma_i, \phi, \lambda, \ell) \in E$ 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 $\phi$, 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, $f_i(C_i, w(i, k]) = (C'_i, w'(i, k])$. $\square$

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

**Corollary 1** *Let $C$ and $C'$ be two configurations of $\mathcal{A}$ and $w$, $w'$ be data words such that $C, w \sim C', w'$. Then* $\mathrm{Succ}_{\mathcal{A}}(C, w)$ *is accepting, if, and only if,* $\mathrm{Succ}_{\mathcal{A}}(C', w')$ *is accepting.*

Combining the last corollary with Proposition 1, 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.*

## 3.2 The Synchronized State Space

For the next two sections, let $\mathcal{A} = (\mathcal{R}^{\mathcal{A}}, \mathcal{L}^{\mathcal{A}}, \ell_{\text{in}}^{\mathcal{A}}, \mathcal{L}_{\text{acc}}^{\mathcal{A}}, E^{\mathcal{A}})$ be an NRA over $\mathcal{D}$ and $\Sigma$, and let $\mathcal{B} = (\mathcal{R}^{\mathcal{B}}, \mathcal{L}^{\mathcal{B}}, \ell_{\text{in}}^{\mathcal{B}}, \mathcal{L}_{\text{acc}}^{\mathcal{B}}, E^{\mathcal{B}})$ be a URA over $\mathcal{D}$ and $\Sigma$. Without loss of generality, we assume $\mathcal{R}^{\mathcal{A}} \cap \mathcal{R}^{\mathcal{B}} = \emptyset$ 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 $\langle(\ell, \boldsymbol{d}), C\rangle$, where $(\ell, \boldsymbol{d}) \in (\mathcal{L}^{\mathcal{A}} \times \mathbb{D}_{\perp}^{\mathcal{R}^{\mathcal{A}}})$ is a single state of $\mathcal{A}$, and $C \subseteq (\mathcal{L}^{\mathcal{B}} \times \mathbb{D}_{\perp}^{\mathcal{R}^{\mathcal{B}}})$ 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 $S_{\text{in}} := \langle(\ell_{\text{in}}^{\mathcal{A}}, \perp), \{(\ell_{\text{in}}^{\mathcal{B}}, \perp)\}\rangle$ 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 the transition relation $\Rightarrow$ is induced from the transition relations of $\mathcal{A}$ and $\mathcal{B}$ as follows. If $S = \langle(\ell, \boldsymbol{d}), C\rangle$ and $S' = \langle(\ell', \boldsymbol{d'}), C'\rangle$, then $S'$ 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 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 $S_{\text{in}}$ reaches $S$. We say that a synchronized configuration $S = \langle(\ell, \boldsymbol{d}), C\rangle$ is *coverable in* $(\mathbb{S}, \Rightarrow)$ if there exists some synchronized configuration $S' = \langle(\ell, \boldsymbol{d}), C'\rangle$ 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 $\langle(\ell, \boldsymbol{d}), C\rangle$ *bad* if $\ell \in \mathcal{L}_{\text{acc}}^{\mathcal{A}}$ is an accepting location and $C$ is non-accepting, i.e., $\ell' \notin \mathcal{L}_{\text{acc}}^{\mathcal{B}}$ for all $(\ell', \boldsymbol{a}) \in C$. The following proposition is easy to prove, cf. [17].

**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 a natural manner, i.e., we define $\langle(\ell, \boldsymbol{d}), C\rangle \sim \langle(\ell, \boldsymbol{d'}), C'\rangle$ if there exists a partial isomorphism $f$ of $\mathcal{D}$ such that $f(C) = C'$ and $f(\boldsymbol{d}) = \boldsymbol{d'}$. 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 $S = \langle(\ell, \boldsymbol{d}), C\rangle$ 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, since $\mathcal{D}$ has only finitely many relations, there exist synchronized configurations $S_1, \ldots, S_k$ for some $k \in \mathbb{N}$ such that $S_i$ for all $i \in \{1, \ldots, k\}$, and such that for all $S' \in \mathbb{S}$ with $S'$ there exists $i \in \{1, \ldots, k\}$ such that $S_i \sim S'$. For example, if $\mathcal{D}$ is $(\mathbb{N}, =)$, for every two different data values $d, d' \in \mathbb{D}\backslash\mathrm{data}(S)$, if inputting $(\sigma, d)$ induces a transition $S \Rightarrow S'$, then inputting $(\sigma, d')$ induces a transition $S \Rightarrow S''$ such that $S' \sim S''$. This is why we define in Section 4.2 the notion of *abstract synchronized 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 Fig. 1. For every $k \geq 1$, the configuration $\{(\ell_0, \perp), (\ell_1, d_1), (\ell_1, d_2) \ldots, (\ell_1, d_k)\}$ 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 focusing on a subset of configurations of $\mathcal{B}$ that are bounded in size, thus reducing to a reachability problem on a finite graph. Before, we define in the next section one of the central notions for yielding a finite graph: indistinguishability.

## 3.3 Indistinguishability

From the previous section, recall the equivalence relation $\sim$ on $k$-tuples, where for $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{D}_\perp^k$ we have $\boldsymbol{a} \sim \boldsymbol{b}$ if there exists a partial isomorphism $f$ of $\mathcal{D}$ such that $f(\boldsymbol{a}) = \boldsymbol{b}$. When $\mathcal{D}$ is $(\mathbb{N}; =)$ or $(\mathbb{Q}; <)$, then the index of this equivalence relation is finite for all $k$. As an example, for $k = 3$, the equivalence classes of triples over $(\mathbb{N}; =)$ are the classes of $(0, 0, 0)$, $(0, 0, 1)$, $(0, 1, 0)$, $(1, 0, 0)$, $(0, 1, 2)$, and the equivalence classes of triples over $(\mathbb{Q}; <)$ correspond to weak linear orders on three elements.

Recall that $m$ and $n$ denote the number of registers of $\mathcal{A}$ and $\mathcal{B}$. For every $\boldsymbol{e} \in \mathbb{D}_\perp^{m+n}$, for every configuration $C$, and for every equivalence class $B$ induced by the equivalence relation $\sim$ on $(m + 2n)$-tuples, we define the set

$$\mathcal{L}_{\boldsymbol{e}}^{C}(B) := \{\ell' \in \mathcal{L}^{\mathcal{B}} \mid \exists \boldsymbol{c} \in \mathbb{D}_\perp^n \text{ such that } (\ell', \boldsymbol{c}) \in C \text{ and } (\boldsymbol{e}, \boldsymbol{c}) \in B\}.$$

While this set depends on $C$, we will from now on omit the superscript as it will always be clear from the context. Let $S = \langle(\ell, \boldsymbol{d}), C\rangle$ be a synchronized configuration and let $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{D}_\perp^n$ be two register valuations. We say that $\boldsymbol{a}$ and $\boldsymbol{b}$ *are indistinguishable in $S$*, written $\boldsymbol{a} \equiv_S \boldsymbol{b}$, if $\mathcal{L}_{(\boldsymbol{d},\boldsymbol{a})}(B) = \mathcal{L}_{(\boldsymbol{d},\boldsymbol{b})}(B)$ for every equivalence class $B$.

*Example 1* Let us consider two register automata $\mathcal{A}$ and $\mathcal{B}$ over $(\mathbb{N}; =)$, where $\mathcal{A}$ has 1 register and $\mathcal{B}$ has 2 registers. Let $(\ell^{\mathcal{A}}, 3)$ be a state in $\mathcal{A}$ and let $C' = \{(\ell, 1, 3), (\ell, 2, 3), (\ell', 1, 2)\}$ be a configuration of $\mathcal{B}$. Let $S' = \langle(\ell^{\mathcal{A}}, 3), C'\rangle$ be the corresponding synchronized configuration of $\mathcal{A}$ and $\mathcal{B}$. Consider $\boldsymbol{a} = (1, 3)$ and $\boldsymbol{b} = (2, 3)$. For the equivalence class $B$ of $(3, 1, 3, 2, 3)$ we have $\mathcal{L}_{(3,\boldsymbol{a})}(B) = \{\ell\}$ as $(3, \boldsymbol{a}, \boldsymbol{b}) = (3, 1, 3, 2, 3) \in B$ and $(\ell, \boldsymbol{b}) \in C$. Similarly $\mathcal{L}_{(3,\boldsymbol{b})}(B) = \{\ell\}$ as $(\ell, \boldsymbol{a}) \in C$ and $(3, \boldsymbol{b}, \boldsymbol{a}) = (3, 2, 3, 1, 3) \sim (3, 1, 3, 2, 3) \in B$. However, taking now $B$ to be the equivalence class of $(3, 1, 3, 1, 2)$ we have $\mathcal{L}_{(3,\boldsymbol{a})}(B) = \{\ell'\}$ but $\mathcal{L}_{(3,\boldsymbol{b})}(B) = \emptyset$. Thus, $\boldsymbol{a}$ and $\boldsymbol{b}$ are not indistinguishable. However, $\boldsymbol{a} \equiv_S \boldsymbol{b}$ for $S = \langle(\ell^{\mathcal{A}}, 3), C\rangle$ with $C := C' \cup \{(\ell', 2, 1)\}$.

A consequence of the definition is that indistinguishable register valuations appear at the same locations of a configuration:

**Proposition 5** *Let $S = \langle(\ell^{\mathcal{A}}, \boldsymbol{d}), C\rangle$ be a coverable synchronized configuration of $\mathcal{A}$ and $\mathcal{B}$. Let $\boldsymbol{a}, \boldsymbol{b}$ be register valuations appearing in $C$ such that $\boldsymbol{a} \equiv_S \boldsymbol{b}$. Then the map $f\colon \mathrm{data}(\boldsymbol{a}) \to \mathrm{data}(\boldsymbol{b})$ defined by $f(a_i) := b_i$ is a partial isomorphism of $\mathcal{D}$. Moreover, if we 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 $f(C_{\boldsymbol{a}}) = C_{\boldsymbol{b}}$.*

*Proof* Let $B$ be the equivalence class of $(\boldsymbol{d}, \boldsymbol{a}, \boldsymbol{a})$. Note that for two vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{D}^n_{\perp}$, $(\boldsymbol{d}, \boldsymbol{u}, \boldsymbol{v}) \in B$ iff $\boldsymbol{u} = \boldsymbol{v}$ and $(\boldsymbol{d}, \boldsymbol{a}) \sim (\boldsymbol{d}, \boldsymbol{u}) \sim (\boldsymbol{d}, \boldsymbol{v})$.

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

$$(\boldsymbol{d}, \boldsymbol{b}, \boldsymbol{c}) \in B \tag{1}$$

for some $(\ell, \boldsymbol{c}) \in C$. Now, (1) implies $\boldsymbol{b} = \boldsymbol{c}$ and $(\boldsymbol{d}, \boldsymbol{a}) \sim (\boldsymbol{d}, \boldsymbol{b})$. The former implies that $(\ell, \boldsymbol{b}) \in C_{\boldsymbol{b}}$, while the latter implies that the map $f$ as defined in the statement is a partial isomorphism of $\mathcal{D}$. Conversely, we obtain that $(\ell, \boldsymbol{b}) \in C_{\boldsymbol{b}}$ implies $(\ell, \boldsymbol{a}) \in C_{\boldsymbol{a}}$. Hence $f(C_{\boldsymbol{a}}) = C_{\boldsymbol{b}}$. □

## 4 The Containment Problem for Register Automata over $(\mathbb{N}; =)$

In this section, we prove the decidability of the containment problem for register automata over the relational structure $\mathcal{D} = (\mathbb{N}; =)$.

### 4.1 Reducing the Size of Configurations

The crucial ingredient of our algorithm for deciding whether $L(\mathcal{A}) \subseteq L(\mathcal{B})$ for register automata over $(\mathbb{N}; =)$ holds is to prevent configurations $C$ in a synchronized configuration $\langle(\ell, \boldsymbol{d}), C\rangle$ from growing unboundedly. We reduce the size of a configuration by *removing* subconfigurations that behave equivalently with respect to reaching a bad synchronized configuration in $(\mathbb{S}, \Rightarrow)$. The key notion for deciding when subconfigurations can be removed is indistinguishability from Section 3.3.

Intuitively, if the configuration $C$ in a synchronized configuration contains two indistinguishable register valuations $\boldsymbol{a}$ and $\boldsymbol{b}$, then we can safely remove all states $(\ell, \boldsymbol{c})$ from $C$ such that $c_i = b_j$ or $c_i = a_j$ for some $1 \le i, j \le n$ with $a_j \ne b_j$.

**Proposition 6** *Let $S = \langle(\ell^{\mathcal{A}}, \boldsymbol{d}), C\rangle$ 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 $C^- := \{(\ell, \boldsymbol{c}) \in C \mid \ell \in \mathcal{L}^{\mathcal{B}}, \exists 1 \le i, j \le n. (c_i = b_j \text{ or } c_i = a_j), \text{ and } a_j \ne b_j\}$. Then for all $k \ge 0$, $\langle(\ell^{\mathcal{A}}, \boldsymbol{d}), C \setminus C^-\rangle$ reaches a bad synchronized configuration in k steps if, and only if, S reaches a bad synchronized configuration in k steps.*

*Proof* The "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 "only if" direction, suppose that there exists a data word $w$ and an accepting run of $\mathcal{A}$ on $w$ that starts in $(\ell^{\mathcal{A}}, \boldsymbol{d})$, and such that $\mathrm{Succ}_{\mathcal{B}}(C \setminus C^-, w)$ is non-accepting. We assume in the following that $\mathrm{Succ}_{\mathcal{B}}(C^-, w)$ is accepting; otherwise we are done. So let $(\ell^{\mathcal{B}}, \boldsymbol{c}) \in C^-$ be the (unique) state such that $\mathrm{Succ}_{\mathcal{B}}((\ell^{\mathcal{B}}, \boldsymbol{c}), w)$ is accepting.

First, we show that without loss of generality we can assume that $w$ does not use any data from $\mathrm{data}(S)$, except for data also occurring in $\boldsymbol{c}$ or $\boldsymbol{d}$. Suppose that this is not the case, i.e., that $\mathrm{data}(w) \cap \mathrm{data}(S) \nsubseteq \mathrm{data}(\boldsymbol{c}) \cup \mathrm{data}(\boldsymbol{d})$. Then, for every $e \in (\mathrm{data}(w) \cap \mathrm{data}(S)) \setminus (\mathrm{data}(\boldsymbol{c}) \cup \mathrm{data}(\boldsymbol{d}))$, pick a fresh datum $e' \in \mathbb{N}$ not occurring in $\mathrm{data}(w) \cup \mathrm{data}(S)$, and simultaneously replace every occurrence of $e$ in $w$ by $e'$. Let $w'$ be the resulting data word. Then $(\ell^{\mathcal{A}}, \boldsymbol{d}), w \sim (\ell^{\mathcal{A}}, \boldsymbol{d}), w'$ and $(\ell^{\mathcal{B}}, \boldsymbol{c}), w \sim (\ell^{\mathcal{B}}, \boldsymbol{c}), w'$. By Corollary 1, $\mathrm{Succ}_{\mathcal{A}}((\ell^{\mathcal{A}}, \boldsymbol{d}), w')$ is accepting, and $\mathrm{Succ}_{\mathcal{B}}((\ell^{\mathcal{B}}, \boldsymbol{c}), w')$ is accepting, too. Then there must exist some accepting run of $\mathcal{A}$ on $w'$ starting in $(\ell^{\mathcal{A}}, \boldsymbol{d})$, and, by Proposition 1, $\mathrm{Succ}_{\mathcal{B}}(C \setminus C^-, w')$ must be non-accepting. Hence, we could continue the proof with $w'$ instead of $w$, so that we can assume without loss of generality that $\mathrm{data}(w) \cap \mathrm{data}(S) \subseteq \mathrm{data}(\boldsymbol{c}) \cup \mathrm{data}(\boldsymbol{d})$ holds.

Let now $w''$ be the data word obtained from $w$ as follows: for every $1 \le i \le n$, if $c_i \in \mathrm{data}(w)$ and $c_i = b_j$ or $c_i = a_j$ for some $1 \le i \le n$ such that $b_j \ne a_j$, pick some fresh datum $c_i' \in \mathbb{N}$ not occurring in $\mathrm{data}(w) \cup \mathrm{data}(S')$. Then replace every occurrence of such datum $c_i$ in $w$ by $c_i'$.

Note that $(\ell^{\mathcal{A}}, \boldsymbol{d}), w \sim (\ell^{\mathcal{A}}, \boldsymbol{d}), w''$: the key argument for this is that by $\boldsymbol{a} \equiv_S \boldsymbol{b}$ we have $b_j \notin \mathrm{data}(\boldsymbol{d})$ and $a_j \notin \mathrm{data}(\boldsymbol{d})$ whenever $b_j \ne a_j$. Hence, replacing all $c_i$'s satisfying $c_i = b_j$ or $c_i = a_j$ in $w$ by the fresh datum $c_i'$ has no effect on $(\ell^{\mathcal{A}}, \boldsymbol{d})$ at all. By Corollary 1, $\mathrm{Succ}_{\mathcal{A}}((\ell^{\mathcal{A}}, \boldsymbol{d}), w'')$ is accepting. Hence there must exist some accepting run of $\mathcal{A}$ on $w''$ starting in $(\ell^{\mathcal{A}}, \boldsymbol{d})$.

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

– Suppose $\boldsymbol{e}$ is such that for all $1 \le i \le n$, if $c_i = b_j$ or $c_i = a_j$ for some $1 \le j \le n$ such that $a_j \ne b_j$, then $c_i \notin \mathrm{data}(\boldsymbol{e})$. Then any changes in $w$ resulting in $w''$

do not have any effect on $(\ell, \boldsymbol{e})$ at all, or more formally: $(\ell, \boldsymbol{e}), w \sim (\ell, \boldsymbol{e}), w''$. This is witnessed by the map $f\colon \mathrm{data}(\boldsymbol{e}) \cup \mathrm{data}(w) \to \mathrm{data}(\boldsymbol{e}) \cup \mathrm{data}(w'')$ defined by

- $f(c_i) = c'_i$ for all $c_i \in \mathrm{data}(w)$ such that $c_i = b_j$ or $c_i = a_j$ for some $1 \le j \le n$ such that $b_j \neq a_j$, and
- $f(x) = x$ for all other $x \in \mathrm{data}(\boldsymbol{e}) \cup \mathrm{data}(w)$.

Note that $f$ is a well-defined partial isomorphism of $(\mathbb{N}; =)$ and $f((\ell, \boldsymbol{e}), w) = ((\ell, \boldsymbol{e}), w'')$. Recall that by assumption $\mathrm{Succ}_{\mathcal{B}}((\ell, \boldsymbol{e}), w)$ is non-accepting. By Corollary 1, $\mathrm{Succ}_{\mathcal{B}}((\ell, \boldsymbol{e}), w'')$ is non-accepting.

- Suppose $\boldsymbol{e}$ is such that there exists $1 \le i \le n$ with $c_i = b_j$ or $c_i = a_j$ for some $1 \le j \le n$ with $a_j \neq b_j$, and $c_i \in \mathrm{data}(\boldsymbol{e})$. We consider the case $c_i = b_j$; the proof for $c_i = a_j$ is analogous.

  Let $B$ be the equivalence class of $(\boldsymbol{d}, \boldsymbol{b}, \boldsymbol{e})$. Note that $\ell \in \mathcal{L}_{(\boldsymbol{d},\boldsymbol{b})}(B)$. By $\boldsymbol{a} \equiv_S \boldsymbol{b}$, there exists $(\ell, \boldsymbol{e}') \in C$ such that $(\boldsymbol{d}, \boldsymbol{a}, \boldsymbol{e}') \in B$. Note that for all $1 \le i \le n$ such that $e_i = b_j$ we have $e'_i = a_j$. By assumption, there exists $1 \le i \le n$ such that $e_i = b_j$. Since $a_j \neq b_j$, we can infer $e_i \neq e'_i$, and hence $(\ell, \boldsymbol{e}) \neq (\ell, \boldsymbol{e}')$ are two distinct states in $C$.

  Next we prove $(\ell, \boldsymbol{e}), w'' \sim (\ell, \boldsymbol{e}'), w''$. We define the map $f\colon \mathrm{data}(\boldsymbol{e}) \cup \mathrm{data}(w'') \to \mathrm{data}(\boldsymbol{e}') \cup \mathrm{data}(w'')$ as follows:

$$f\colon \begin{cases} e_p \mapsto e'_p & 1 \le p \le n \\ x \mapsto x & x \in \mathrm{data}(w'') \end{cases}$$

  We prove below that

  (i) for all $1 \le p, q \le n$, $e_p = e_q$ iff $e'_p = e'_q$;
  (ii) for all $1 \le p \le n$, for all $x \in \mathrm{data}(w'')$, $x = e_p$ iff $x = e'_p$;

  note that this implies that $f$ is well-defined and $f$ is a partial isomorphism of $(\mathbb{N}; =)$, and hence $f((\ell, \boldsymbol{e}), w'') = ((\ell, \boldsymbol{e}'), w'')$. We have $\mathrm{Succ}_{\mathcal{B}}((\ell, \boldsymbol{e}), w'') \sim \mathrm{Succ}_{\mathcal{B}}((\ell, \boldsymbol{e}'), w'')$ by Proposition 2. But then, by Corollary 2, $\mathrm{Succ}_{\mathcal{B}}((\ell, \boldsymbol{e}), w'')$ and $\mathrm{Succ}_{\mathcal{B}}((\ell, \boldsymbol{e}'), w'')$ are non-accepting. We now prove the two items from above: (i) Follows directly from the fact that $(\boldsymbol{d}, \boldsymbol{a}, \boldsymbol{e}') \in B$ and $(\boldsymbol{d}, \boldsymbol{b}, \boldsymbol{e}) \in B$. For (ii), recall that $\mathrm{data}(w) \cap \mathrm{data}(S) \subseteq \mathrm{data}(\boldsymbol{c}) \cup \mathrm{data}(\boldsymbol{d})$. This, the definition of $w''$, and $\boldsymbol{a} \equiv_S \boldsymbol{b}$ yield the claim.

Altogether, we proved that $\mathrm{Succ}_{\mathcal{B}}(C', w'')$ 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 "only if" direction. □

When $S^-$ is obtained from $S$ by applying Proposition 6 to some family of register valuations, we say that *$S$ reduces to $S^-$*. We say that $S$ is *maximally reduced* if for all pairs $\boldsymbol{a}$ and $\boldsymbol{b}$ of distinct register valuations appearing in $C$ we have that $\boldsymbol{a} \equiv_S \boldsymbol{b}$ does *not* hold. Note that in Proposition 6, the reduced configuration is again coverable. By iterating Proposition 6, one obtains that a coverable synchronized configuration reaches a bad synchronized configuration if, and only if, it reduces in finitely many steps to a maximally reduced synchronized configuration that also reaches a bad synchronized configuration.

Before we present our algorithm for deciding the containment problem, we would like to point out that the equivalence relation $\sim$ alone is not sufficient for deciding whether synchronized configurations can be reduced. More precisely, given a coverable synchronized configuration $S = \langle(\ell^{\mathcal{A}}, \boldsymbol{d}), C\rangle$ and two register valuations $\boldsymbol{a}$ and $\boldsymbol{b}$ of $\mathcal{B}$ that occur in $C$ and for which $(\boldsymbol{d}, \boldsymbol{a}) \sim (\boldsymbol{d}, \boldsymbol{b})$, it is in general *not* the case that $S$ reaches a bad synchronized configuration if $\langle(\ell^{\mathcal{A}}, \boldsymbol{d}), C \setminus C^{-}\rangle$, where $C^{-}$ is defined as in Proposition 6, reaches a bad synchronized configuration. To see that, consider Fig. 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 $S = \langle(\ell^{\mathcal{A}}, 3), C\rangle$, 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 $(\boldsymbol{a}, \boldsymbol{d}) \sim (\boldsymbol{b}, \boldsymbol{d})$, but $\boldsymbol{a} \equiv_S \boldsymbol{b}$ does not hold (cf. Example 1). Note that $C^{-} = C$. Now, $\mathrm{Succ}_{\mathcal{B}}(\emptyset, w)$ is non-accepting for every data word $w$, but $C$ cannot reach any non-accepting configuration.

## 4.2 Abstract Synchronized Configurations

The main result of this section is that the number of different register valuations in a maximally reduced synchronized configuration is bounded. For this, we study synchronized configurations up to the equivalence relation $\sim$, cf. Section 3.2, and define the notion of *abstract synchronized configurations*.

Recall that $m$ is the number of registers of $\mathcal{A}$ and $n$ is the number of registers of $\mathcal{B}$. An *abstract synchronized configuration of* $\mathcal{A}$ *and* $\mathcal{B}$ is a tuple $\langle k, \ell, \Gamma, B\rangle$, where $k \in \mathbb{N}$, $\ell \in \mathcal{L}^{\mathcal{A}}$, $\Gamma$ is a $k$-tuple of subsets of $\mathcal{L}^{\mathcal{B}}$, and $B$ is an equivalence class induced by the equivalence relation $\sim$ on $(m + kn)$-tuples.

Every synchronized configuration $S = \langle(\ell^{\mathcal{A}}, \boldsymbol{d}), C\rangle$ gives rise to an abstract synchronized configuration in the following way: let $\boldsymbol{a}^1, \ldots, \boldsymbol{a}^k$ be the distinct register valuations in $C$, enumerated in some arbitrary order. Let $B$ be the equivalence class of $(\boldsymbol{d}, \boldsymbol{a}^1, \ldots, \boldsymbol{a}^k) \in \mathbb{N}^{m+kn}$. Let $L_{\boldsymbol{a}^i} := \{\ell \in \mathcal{L}^{\mathcal{B}} \mid (\ell, \boldsymbol{a}^i) \in C\}$. We obtain an abstract synchronized configuration $\langle k, \ell^{\mathcal{A}}, (L_{\boldsymbol{a}^1}, \ldots, L_{\boldsymbol{a}^k}), B\rangle$. Different enumerations of the register valuations of $C$ can yield different abstract synchronized configurations. We let abs(S) be the set of all abstract synchronized configurations that can be

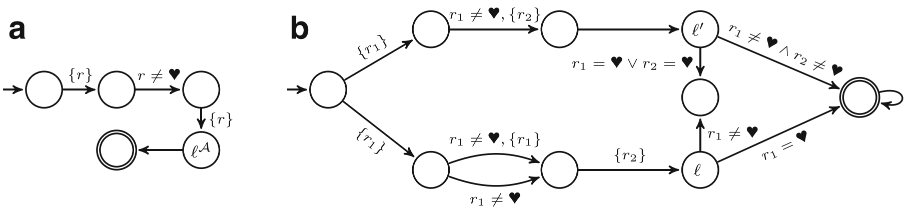

**Fig. 2** An NRA $\mathcal{A}$ and a URA $\mathcal{B}$ over a singleton alphabet for which $L(\mathcal{A}) \subseteq L(\mathcal{B})$

obtained from $S$. Every two abstract synchronized configurations in abs(S) can be obtained from one another by permuting the entries from the tuple and changing the equivalence class accordingly. It is easy to prove that $S \sim S'$ if, and only if, $\text{abs}(S) = \text{abs}(S')$.

*Example 2* Let $\mathcal{A}$ be an NRA with 1 register, and let $\mathcal{B}$ be a URA with 2 registers, and let $S = \langle(\ell^{\mathcal{A}}, 3), \{(\ell_1, 2, 1), (\ell_1, 3, 3), (\ell_2, 2, 0), (\ell_3, 2, 1)\}\rangle$ be a synchronized configuration of $\mathcal{A}$ and $\mathcal{B}$. Then abs($S$) contains the abstract synchronized configuration $\langle 3, \ell^{\mathcal{A}}, (\{\ell_1, \ell_3\}, \{\ell_1\}, \{\ell_2\}), [(3, 2, 1, 3, 3, 2, 0)]_{\sim}\rangle$, where $[\boldsymbol{a}]_{\sim}$ denotes the equivalence class containing $\boldsymbol{a} \in \mathbb{N}^7$.

The *size* of an abstract synchronized configuration $\langle k, \ell, \Gamma, B\rangle$ is defined to be $(m + kn)\log(m + kn) + k|\mathcal{L}^{\mathcal{B}}| + \log(|\mathcal{L}^{\mathcal{A}}|)$, which corresponds to the size needed on the tape of a Turing machine to encode an abstract synchronized configuration.

An abstract synchronized configuration $\langle k, \ell^{\mathcal{A}}, \Gamma, B\rangle$ is said to be *maximally reduced* if there exists a synchronized configuration $S$ such that $\langle k, \ell^{\mathcal{A}}, \Gamma, B\rangle \in \text{abs}(S)$ and such that $S$ is maximally reduced . Equivalently, one could ask that *every* synchronized configuration $S$ such that $\langle k, \ell^{\mathcal{A}}, \Gamma, B\rangle \in \text{abs}(S)$ is maximally reduced: this is because every $S'$ such that abs($S$) contains $\langle k, \ell^{\mathcal{A}}, \Gamma, B\rangle$ must satisfy $S \sim S'$, and the notion of being maximally reduced for a synchronized configuration is invariant under the relation $\sim$.

Let $N_k$ be the number of equivalence classes induced by the equivalence relation $\sim$ on $k$-tuples over $\mathbb{N}$, which is also called the *Bell number of order k*. Note that $N_k$ is bounded from above by $k^k$.

**Proposition 7** *Let $S = \langle(\ell^{\mathcal{A}}, \boldsymbol{d}), C\rangle$ be a maximally reduced synchronized configuration of $\mathcal{A}$ and $\mathcal{B}$. The number of different register valuations appearing in C is bounded by $(N_{m+2n} \cdot 2^{|\mathcal{L}^{\mathcal{B}}|})^{(m+n+1)^n}$.*

*Proof* We first prove a slightly worse upper bound, to give an idea of the proof. Let $K := N_{m+2n}$. We prove that the number of different register valuations is bounded by $2^{|\mathcal{L}^{\mathcal{B}}|K}$. Associate with every register valuation $\boldsymbol{a}$ appearing in $C$ the $K$-tuple $(\mathcal{L}_{(\boldsymbol{d},\boldsymbol{a})}(B_1), \ldots, \mathcal{L}_{(\boldsymbol{d},\boldsymbol{a})}(B_K))$ of subsets of $\mathcal{L}^{\mathcal{B}}$, where $B_1, \ldots, B_K$ is an enumeration of all equivalence classes induced by the equivalence relation $\sim$ on $(m + 2n)$-tuples over $\mathbb{N}$. 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 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 reduced 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 reduced synchronized configuration.

We now proceed to prove the actual bound. The important fact is that when $\boldsymbol{d}$ and $\boldsymbol{a}$ are fixed in $S$, then few (i.e., $\leq (m + n + 1)^n$) entries in the $K$-tuple $(\mathcal{L}_{(\boldsymbol{d},\boldsymbol{a})}(B_1), \ldots, \mathcal{L}_{(\boldsymbol{d},\boldsymbol{a})}(B_K))$ are non-empty. Indeed, if $B$ is the equivalence class containing $(\boldsymbol{d}, \boldsymbol{a}, b_1, \dots b_n)$, then each of the values $b_1, \dots, b_n$ can be constrained to be equal to one of $d_1, \dots, d_m, a_1, \dots, a_n$, 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 $(m+n+1)^n$ non-empty entries. Each such tuple is characterized 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 $(m+n+1)^n$, we obtain that there are at most $K^{(m+n+1)^n} \cdot 2^{|\mathcal{L}^{\mathcal{B}}|(m+n+1)^n}$ possible tuples, and thus at most $(N_{m+2n} \cdot 2^{|\mathcal{L}^{\mathcal{B}}|})^{(m+n+1)^n}$ 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 reduced abstract synchronized configurations.

**Proposition 8** *The number of maximally reduced abstract synchronized 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 reduced synchronized configuration $S = \langle(\ell^{\mathcal{A}}, \boldsymbol{d}), C\rangle$ is such that $C$ contains at most $K := (N_{m+2n} \cdot 2^{|\mathcal{L}^{\mathcal{B}}|})^{(m+n+1)^n}$ different register valuations. Therefore, any abstract synchronized configuration in abs$(S)$ is of the form $\langle k, \ell^{\mathcal{A}}, \Gamma, B\rangle$, for some $k \leq K$. For a given $k$, there are at most $N_{m+kn} \cdot |\mathcal{L}^{\mathcal{B}}|^k \cdot |\mathcal{L}^{\mathcal{A}}|$ different abstract synchronized configurations. Summing up from $k = 0$ to $K$, we obtain that there are at most

$$\begin{aligned}\sum_{k=0}^{K} N_{m+kn} \cdot |\mathcal{L}^{\mathcal{B}}|^k \cdot |\mathcal{L}^{\mathcal{A}}| &\leq |\mathcal{L}^{\mathcal{A}}| \cdot \Big(N_m + N_{m+n}|\mathcal{L}^{\mathcal{B}}| + \dots + N_{m+nK} \cdot |\mathcal{L}^{\mathcal{B}}|^K\Big) \\ &\leq |\mathcal{L}^{\mathcal{A}}| \cdot (1+K) \cdot N_{m+nK} \cdot |\mathcal{L}^{\mathcal{B}}|^K \\ &\leq |\mathcal{L}^{\mathcal{A}}| \cdot (1+K) \cdot (m+nK)^{(m+nK)} \cdot |\mathcal{L}^{\mathcal{B}}|^K\end{aligned}$$

maximally reduced 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 $\langle k, \ell^{\mathcal{A}}, \Gamma, B\rangle$ and $\langle k', \ell'^{\mathcal{A}}, \Gamma', B'\rangle$, define $\langle k, \ell^{\mathcal{A}}, \Gamma, B\rangle \rightsquigarrow \langle k', \ell'^{\mathcal{A}}, \Gamma', B'\rangle$ if there exist synchronized configurations $S$ and $S'$ such that:

- $\mathrm{S} \Rightarrow \mathrm{S}'$
- $\langle k, \ell^{\mathcal{A}}, \Gamma, B\rangle$ is in abs(S),
- $S'$ can be maximally reduced to some $S''$ such that $\langle k', \ell'^{\mathcal{A}}, \Gamma', B'\rangle$ is in abs(S”).

**Lemma 1** *Given abstract synchronized configurations $\langle k, \ell^{\mathcal{A}}, \Gamma, B\rangle$ and $\langle k', \ell'^{\mathcal{A}}, \Gamma', B'\rangle$, deciding whether $\langle k, \ell^{\mathcal{A}}, \Gamma, B\rangle \rightsquigarrow \langle k', \ell'^{\mathcal{A}}, \Gamma', B'\rangle$ holds can be done in polynomial space.*

*Proof* Recall that we work here with $\mathcal{D} = (\mathbb{N}; =)$. Note that there is a synchronized configuration $S$ of the form $\langle(\ell^{\mathcal{A}}, \boldsymbol{d}), D\rangle$ such that $\text{data}(D) \cup \text{data}(\boldsymbol{d}) \subseteq \{1, \ldots, m+kn\}$ and such that $\langle k, \ell^{\mathcal{A}}, \Gamma, B\rangle \in \text{abs(S)}$. This $S$ is moreover computable in polynomial space.

To decide whether $\langle k, \ell^{\mathcal{A}}, \Gamma, B\rangle \rightsquigarrow \langle k', \ell'^{\mathcal{A}}, \Gamma', B'\rangle$ holds, one simply:

1. guesses a letter $\sigma \in \Sigma$ and a datum $d$ in $\{1, \ldots, m + kn + 1\}$,
2. computes a synchronized configuration $S'$ obtained by firing a transition corresponding to $(\sigma, d)$ from $S$,
3. 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 reduced configuration $S''$,
4. checks that $\langle k', \ell'^{\mathcal{A}}, \Gamma', B'\rangle$ is in 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 reducing 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 configurations, an abstract synchronized configuration $\langle k, \ell^{\mathcal{A}}, \Gamma, B\rangle$ is called *bad* if $\ell^{\mathcal{A}}$ is an accepting location and none of the states in $\Gamma$ contains an accepting location. Given an abstract synchronized configurations $A$ and a synchronized configuration $S$, we say that an abstract synchronized configuration *$A$ is reachable from* abs($S$) if there exists an abstract synchronized configuration $A'$ in abs($S$) such that $A' \rightsquigarrow^* A$, where $\rightsquigarrow^*$ denotes the reflexive-transitive closure of $\rightsquigarrow$.

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

*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 $\text{abs}(S)$. The statement then follows by taking $S := S_{\text{in}}$. 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 $S \Rightarrow S'$ and such that $S'$ reaches a bad synchronized configuration in $n - 1$ steps. Let $S''$ be such that $S'$ can be maximally reduced to $S''$. Since $S$ is coverable, also $S'$ is coverable. By Proposition 6, we have that $S''$ reaches a bad synchronized configuration in $n - 1$ steps. It follows from the induction hypothesis that some $\langle k', \ell', \Gamma', B'\rangle \in \text{abs}(S'')$ reaches a bad abstract synchronized configuration in $n - 1$ steps. Let $\langle k, \ell, \Gamma, B\rangle$ be in abs($S$). We have by definition $\langle k, \ell, \Gamma, B\rangle \rightsquigarrow \langle k', \ell', \Gamma', B'\rangle$, so that $\langle k, \ell, \Gamma, B\rangle$ 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 1** *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* The algorithm checks whether a bad abstract synchronized configuration is reachable from $\mathrm{abs}(S_{\mathrm{in}})$, using the classical non-deterministic logspace algorithm for reachability. Every node of the graph can be stored using double-exponential space (see the second paragraph at the beginning of Section 4.2), 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 1. 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 1, we obtain that the universality problem is in 2-EXPSPACE and in PSPACE for fixed number of registers. Similarly, the equivalence problem for unambiguous register automata is in 2-EXPSPACE.

## 5 The Containment Problem for Register Automata over $(\mathbb{Q}; <)$

In this section, we study the containment problem for the case that $\mathcal{A}$ and $\mathcal{B}$ are register automata over $(\mathbb{Q}, <)$. The following example shows that one *cannot* adapt the proof of Proposition 6 from the previous section to show that for deciding the containment problem one can restrict attention to a finite subset of synchronized configurations of $\mathcal{A}$ and $\mathcal{B}$.

*Example 3* Let $C_4 = \{(\ell', 0, 6), (\ell, 0, 1), (\ell, 1, 2), (\ell, 2, 3), (\ell, 3, 4), (\ell, 4, 5), (\ell, 5, 6)\}$ be a configuration of a URA with 2 registers. Recall the definition of indistinguishability in Section 3.3 for $(\mathbb{Q}; <)$, and note that $(2, 3)$ and $(3, 4)$ are indistinguishable: $\mathcal{L}_{(2,3)}(B) = \mathcal{L}_{(3,4)}(B)$ for every equivalence class $B$ induced by the equivalence relation $\sim$ on 4-tuples. Now, let $\mathcal{B}_4$ be the URA over $(\mathbb{Q}, <)$ and a singleton alphabet partially depicted in Fig. 3. We prove that $\mathrm{Succ}_{\mathcal{B}}(C_4, w)$ is accepting for all data words $w$: for data words $w$ with $|w| \neq 1$ this is clear. So let us consider single-letter data words of the form $w = (\sigma, d)$. Note that in order to gain non-acceptance of $\mathrm{Succ}_{\mathcal{B}}((\ell', 0, 6), w)$, we must have $d \in [0, 6)$. However, in order to gain non-acceptance of $\mathrm{Succ}_{\mathcal{B}}((\ell, i, i + 1), w)$, we must have $d \notin [i, i + 1)$, for all

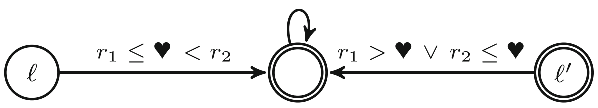
$\ell$
$r_1 \leq \heartsuit < r_2$
$r_1 > \heartsuit \vee r_2 \leq \heartsuit$
$\ell'$

**Fig. 3** A part of a URA over $(\mathbb{Q}; <)$ and a singleton alphabet (we omit the labels at the edges) with two registers $r_1$ and $r_2$

$0 \leq i < 6$. Hence, $\mathrm{Succ}(C_4, w)$ is accepting for all data words $w$. In contrast to this, $\mathrm{Succ}(C \backslash \{(\ell, i, i+1)\}, (\sigma, i))$ is non-accepting, for all $0 \leq i < 6$.

This example shows that removing states from a configuration to yield smaller configurations may be harmful. Note that one can define an infinite family $(\mathcal{B}_k, C_k)_{k \geq 4}$ of URA $\mathcal{B}_k$ with two registers and corresponding configurations $C_k$ yielding counterexamples to (variants of) Proposition 6 as above. In fact, we must leave open whether the containment problem for the case that $\mathcal{B}$ uses at least two registers is decidable or not. We can, however, prove the following variant of Proposition 6 for the case that $\mathcal{B}$ uses only a single register. The restriction to a single register enables us to use a condition on $C$ that is weaker than indistinguishability. (Recall the definition of $\sim$ in Section 3.1.)

**Proposition 10** *Let $S = \langle(\ell^{\mathcal{A}}, \boldsymbol{d}), C\rangle$ be a coverable synchronized configuration of $\mathcal{A}$ and $\mathcal{B}$, where $\mathcal{B}$ uses only one register. Let $a < b$ be two distinct register valuations in $C$ such that $(\boldsymbol{d}, a) \sim (\boldsymbol{d}, b)$ and $\{\ell \in \mathcal{L}^{\mathcal{B}} \mid (\ell, a) \in C\} = \{\ell \in \mathcal{L}^{\mathcal{B}} \mid (\ell, b) \in C\}$. Let $C^- := C \cap (\mathcal{L}^{\mathcal{B}} \times \{a, b\})$. Then $\langle(\ell^{\mathcal{A}}, \boldsymbol{d}), C \setminus C^-\rangle$ reaches a bad synchronized configuration if, and only if, $S$ reaches a bad synchronized configuration.*

*Proof* The "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 $\mathrm{Succ}_{B}(D, w)$ for every subset $D \subseteq C$.

For the "only if" direction, suppose that there exists a data word $w$ such that there exists an accepting run of $\mathcal{A}$ on $w$ that starts in $(\ell^{\mathcal{A}}, \boldsymbol{d})$, and $\mathrm{Succ}_{\mathcal{B}}(C \backslash C^-, w)$ is non-accepting. We assume in the following that $\mathrm{Succ}_{\mathcal{B}}(C^-, w)$ is accepting; otherwise we are done. So let $(\ell^{\mathcal{B}}, c) \in C^-$ be the (unique) state such that $\mathrm{Succ}_{\mathcal{B}}(C^-, w)$ is accepting. We present the proof for the case that $c = b$; the proof for $c = a$ is analogous.

Without loss of generality, we assume that for all $e \in \mathrm{data}(w) \backslash (\mathrm{data}(\boldsymbol{d}) \cup \{b\})$ we have $(e, a) \sim (e, b)$. Otherwise, let $a \leq e_1 < e_2 < \cdots < e_p < b$ be the ordered enumeration of all data $e \in \mathrm{data}(w) \backslash (\mathrm{data}(\boldsymbol{d}) \cup \{b\})$ such that $(e, a) \not\sim (e, b)$. Pick data values $e'_1, e'_2, \ldots, e'_p \in \mathbb{Q}$ such that $e^* < e'_1 < e'_2 < \cdots < e'_p < a$, where $e^* := \max\{e \in \mathrm{data}(\boldsymbol{d}) \cup \mathrm{data}(w) \mid e < a\}$; note that such values exist by density of $(\mathbb{Q}; <)$. Let $w'$ be the data word obtained from $w$ by replacing every occurrence of $e_i$ by $e'_i$, for all $1 \leq i \leq p$. Define the map $f \colon \mathrm{data}(\boldsymbol{d}) \cup \mathrm{data}(w) \to \mathrm{data}(\boldsymbol{d}) \cup \mathrm{data}(w')$ by

$$f(d) := \begin{cases} e'_i & d = e_i \text{ for some } 1 \leq i \leq p \\ d & \text{otherwise.} \end{cases}$$

Note that $f$ is a partial isomorphism of $(\mathbb{Q}; <)$, and $f((\ell^{\mathcal{A}}, \boldsymbol{d}), w) = ((\ell^{\mathcal{A}}, \boldsymbol{d}), w')$ and $f((\ell^{\mathcal{B}}, b), w) = ((\ell^{\mathcal{B}}, b), w')$. By Corollary 1, $\mathrm{Succ}_{\mathcal{A}}((\ell^{\mathcal{A}}, \boldsymbol{d}), w')$ is accepting, and $\mathrm{Succ}_{\mathcal{B}}((\ell^{\mathcal{B}}, b), w')$ is accepting, too. Then there must exist some accepting run of $\mathcal{A}$ on $w'$ starting in $(\ell^{\mathcal{A}}, \boldsymbol{d})$, and, by Proposition 1, $\mathrm{Succ}_{\mathcal{B}}(C \backslash C^-, w')$ must be non-accepting. Hence, we could continue the proof with $w'$ instead of $w$. Let us thus assume henceforth that for all $e \in \mathrm{data}(w) \backslash (\mathrm{data}(\boldsymbol{d}) \cup \{b\})$ we have $(e, a) \sim (e, b)$.

Let now $w''$ be the data word obtained from $w$ by replacing every occurrence of $b$ by $b + \varepsilon$, where $0 < \varepsilon < 1$ is such that $b + \varepsilon < \min\{e \in \mathrm{data}(S) \mid b < e\}$. Define

$f\colon \mathrm{data}(\boldsymbol{d}) \cup \mathrm{data}(w) \to \mathrm{data}(\boldsymbol{d}) \cup \mathrm{data}(w'')$ by

$$f(d) := \begin{cases} b+\varepsilon & d = b \\ d & \text{otherwise.} \end{cases}$$

Clearly, $f$ is a partial isomorphism of $(\mathbb{Q}; <)$. Note that $d_i \neq b$ for all $1 \leq i \leq m$: if, otherwise, $d_i = b$ for some $1 \leq i \leq m$, then by $(\boldsymbol{d}, a) \sim (\boldsymbol{d}, b)$ we have $a = b$, contradicting $a < b$. This and the choice of $\varepsilon$ yield $(\ell^{\mathcal{A}}, \boldsymbol{d}), w \sim (\ell^{\mathcal{A}}, \boldsymbol{d}), w''$. By Corollary 1, $\mathrm{Succ}_{\mathcal{A}}((\ell^{\mathcal{A}}, \boldsymbol{d}), w'')$ is accepting. Hence there must exist some accepting run of $\mathcal{A}$ on $w''$ starting in $(\ell^{\mathcal{A}}, \boldsymbol{d})$.

Next we prove that $\mathrm{Succ}_{\mathcal{B}}(C, w'')$ is non-accepting. For this let $(\ell, e) \in C$. We distinguish the following three cases.

- $e = a$. By assumption, $(\ell, b) \in C$. Define the map $f\colon \mathrm{data}(w'') \cup \{a\} \to \mathrm{data}(w'') \cup \{b\}$ by $f(d) = d$ if $d \in \mathrm{data}(w'')$ and $f(a) = b$. Note that $f$ is a partial isomorphism of $(\mathbb{Q}; <)$, and moreover, by the assumptions on $w$ and $\varepsilon$, one can prove that $f((\ell, a), w'') = ((\ell, b), w'')$. By Corollary 2, $\mathrm{Succ}_{\mathcal{B}}((\ell, e), w'')$ must be non-accepting.
- $e = b$. The proof for this case is symmetric to the previous case.
- $e \notin \{a, b\}$. Define the map $f\colon \mathrm{data}(w) \cup \{e\} \to \mathrm{data}(w'') \cup \{e\}$ by $f(b) = b + \varepsilon$ and $f(d) = d$ otherwise. Clearly, $f$ is a partial isomorphism of $(\mathbb{Q}; <)$, and moreover, by the assumptions on $w$ and $\varepsilon$, one can prove that $f((\ell, e), w) = f((\ell, e), w'')$. By assumption, $\mathrm{Succ}_{\mathcal{B}}((\ell, e), w)$ is non-accepting. This and Corollary 1 imply that also $\mathrm{Succ}_{\mathcal{B}}((\ell, e), w'')$ is non-accepting.

This finishes the proof for the "only if"-direction. □

Using the same strategy as in the previous section, we obtain the decidability for the containment problem in the case of register automata over $(\mathbb{Q}; <)$.

**Theorem 2** *The containment problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$, where $\mathcal{A}$ is a non-deterministic register automaton over $(\mathbb{Q}; <)$ and $\mathcal{B}$ is an unambiguous 1-register automaton over $(\mathbb{Q}; <)$, is in EXPSPACE.*

## 6 The Containment Problem for Timed Automata

In this section, we study *timed automata*, i.e., finite automata augmented with real-valued clocks [1]. We start by defining the model of timed automata, and present a decision procedure for deciding the containment problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$ for the case that $\mathcal{A}$ is a timed automaton and $\mathcal{B}$ is an unambiguous timed automaton with a single clock.

### 6.1 Main Definitions

A *timed word over $\Sigma$* is a finite sequence $(\sigma_1, t_1)(\sigma_2, t_2)\ldots(\sigma_k, t_k)$ over $\Sigma \times \mathbb{R}_{\geq 0}$ such that $t_1 \leq t_2 \leq \cdots \leq t_k$. A timed word is thus a data word over the relational structure $(\mathbb{R}_{\geq 0}; <, =)$, with the additional condition that the sequence $t_1, \ldots, t_k$, where each $t_i$ is called a *timestamp* in $w$, is monotonically increasing.

Let $X = \{x_1, \ldots, x_n\}$ be a finite set of *clocks*. A *clock valuation* is a mapping $\boldsymbol{a} : X \to \mathbb{R}_{\geq 0}$; we may write $a_i$ as shorthand for $\boldsymbol{a}(x_i)$. We denote by $\mathbb{R}^X_{\geq 0}$ the set of all clock valuations over $X$. Given $\lambda \subseteq X$, we define the clock valuation $\boldsymbol{a}[\lambda \leftarrow 0]$ to be the clock valuation such that $(\boldsymbol{a}[\lambda \leftarrow 0])(x) = 0$ if $x \in \lambda$, and $(\boldsymbol{a}[\lambda \leftarrow 0])(x) = \boldsymbol{a}(x)$ otherwise. Given $t \in \mathbb{R}_{\geq 0}$, we define the clock valuation $\boldsymbol{a} + t$ to be the clock valuation such that $(\boldsymbol{a} + t)(x) = \boldsymbol{a}(x) + t$ for all $x \in X$.

A *clock constraint* over $X$ is defined by the grammar

$$\phi ::= \texttt{true} \mid x < k \mid x = k \mid x > k \mid \phi \wedge \phi\,,$$

where $k \in \mathbb{N}$ and $x \in X$. We use $\Phi(X)$ to denote the set of all clock constraints over $X$. The satisfaction relation $\models$ for $\Phi(X)$ on $\mathbb{R}^X_{\geq 0}$ is defined by structural induction, where the base case is $\boldsymbol{a} \models x \bowtie k$ if and only if $\boldsymbol{a}(x) \bowtie k$, for all $\bowtie \ \in \{<, =, >\}$ and $k \in \mathbb{N}$.

A *timed automaton over $\Sigma$* is a tuple $\mathcal{A} = (X, \mathcal{L}, \ell_{\text{in}}, \mathcal{L}_{\text{acc}}, E)$, where

- $X$ is a finite set of clocks,
- $\mathcal{L}$ is a finite set of locations,
- $\ell_{\text{in}} \in \mathcal{L}$ is the initial location, and $\mathcal{L}_{\text{acc}} \subseteq \mathcal{L}$ is the set of accepting locations,
- $E \subseteq \mathcal{L} \times \Sigma \times \Phi(X) \times 2^X \times \mathcal{L}$ is a finite set of edges. Given an edge $(\ell, \sigma, \phi, \lambda, \ell') \in E$, $\sigma$ is the label, $\phi$ is the clock constraint, and $\lambda$ is the set of clocks that are reset to zero. The clock constraint $\texttt{true}$ is vacuously true and may be omitted; likewise we may omit $\lambda$ if $\lambda = \emptyset$.

A *state* of $\mathcal{A}$ is a pair $(\ell, \boldsymbol{a}) \in \mathcal{L} \times \mathbb{R}^X_{\geq 0}$. Given two states $(\ell, \boldsymbol{a})$ and $(\ell', \boldsymbol{a'})$, a time delay $\delta$ and a letter $\sigma \in \Sigma$, we postulate a transition $(\ell, \boldsymbol{a}) \xrightarrow{\sigma,\delta}_{\mathcal{A}} (\ell', \boldsymbol{a'})$ if there exists some edge $(\ell, \sigma, \phi, \lambda, \ell') \in E$ such that $(\boldsymbol{a} + \delta) \models \phi$ and $\boldsymbol{a'} = (\boldsymbol{a} + \delta)[\lambda \leftarrow 0]$. A *run* of $\mathcal{A}$ on the timed word $(\sigma_1, t_1) \ldots (\sigma_k, t_k)$ is a sequence $(\ell_0, \boldsymbol{a^0}) \xrightarrow{\sigma_1,\delta_1}_{\mathcal{A}} (\ell_1, \boldsymbol{a^1}) \xrightarrow{\sigma_2,\delta_2}_{\mathcal{A}} \ldots \xrightarrow{\sigma_k,\delta_k}_{\mathcal{A}} (\ell_k, \boldsymbol{a^k})$ of transitions, where $\delta_1 = t_1$ and $\delta_i = t_i - t_{i-1}$ for all $2 \leq i \leq n$. A *run starts in* $(\ell, \boldsymbol{a})$ if $(\ell_0, \boldsymbol{a^0}) = (\ell, \boldsymbol{a})$, it is *initialized* if it starts in $(\ell_{\text{in}}, \mathbf{0})$, where $\mathbf{0}$ denotes the mapping that maps all clocks to 0, and it is accepting if $\ell_k \in \mathcal{L}_{\text{acc}}$. The timed language *accepted* by $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of timed words $w$ such that there exists an initialized accepting run of $\mathcal{A}$ on $w$. A timed automaton $\mathcal{A}$ is *unambiguous*, if there exists at most one accepting run of $\mathcal{A}$ on $w$, for every timed word $w$.

### 6.2 Towards Decidability of the Containment Problem

We study the containment problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$ for the case that $\mathcal{A}$ and $\mathcal{B}$ are timed automata. Similarly to register automata, the problem is undecidable in general [1], but it is decidable [17] for the case that $\mathcal{B}$ uses a single clock only, with a non-primitive recursive computational complexity (more specifically, the problem is "hyper-Ackermann"-complete [20]). It is thus interesting to study the containment problem for the case that $\mathcal{B}$ is unambiguous.

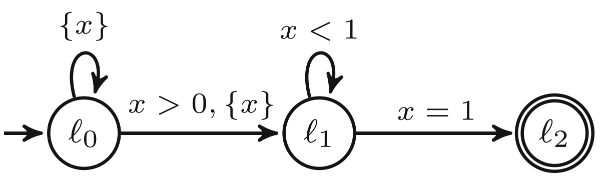

**Fig. 4** An unambiguous timed automaton with a single clock $x$ and over a singleton alphabet (we do not depict the labels of the edges)

Note that, to the best of our knowledge, it is not known whether the class of timed languages accepted by unambiguous timed automata is closed under complementation; we conjecture that this is not the case. For instance, consider the unambiguous timed automaton in Fig. 4. This automaton accepts the timed language $\{(\sigma, t_1)(\sigma, t_2)\dots(\sigma, t_k) \mid \exists 1 \leq i < k.t_k = t_i+1\}$. We conjecture that the complement of this timed language cannot be accepted by any timed automaton.[2] Hence, and similarly to register automata, we cannot solve the containment problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$ by a simple reduction to the non-emptiness problem for a timed automaton accepting the set $L(\mathcal{A}) \cap \overline{L(B)}$. Note that this even applies if $\mathcal{B}$ only uses a single clock (c.f.. Figure 4).

Instead, for solving the containment problem, we rely on the techniques that we developed in the previous sections for register automata: we reduce the containment problem to a reachability problem on a *finite* part of the infinite synchronized state space of $\mathcal{A}$ and $\mathcal{B}$. The crucial ingredient for that is again an analogon to Proposition 10, where we prove that if during the reachability analysis of the synchronized state space one meets a synchronized configuration $\langle(\ell^{\mathcal{A}}, \boldsymbol{d}), C\rangle$ such that $C$ contains two states $(\ell, \boldsymbol{a})$ and $(\ell, \boldsymbol{b})$ that are in a certain sense *equivalent*, then one can safely remove both $(\ell, \boldsymbol{a})$ and $(\ell, \boldsymbol{b})$ from $C$ and continue the reachabilitiy analysis with the synchronized configuration $\langle(\ell^{\mathcal{A}}, \boldsymbol{d}), C\backslash\{(\ell, \boldsymbol{a}), (\ell, \boldsymbol{b})\}\rangle$. However, similarly to register automata over $(\mathbb{Q}; <)$ (c.f. Section 5), we cannot prove such a proposition as soon as we have two clocks. We therefore focus on the containment problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$, where $\mathcal{B}$ is an unambiguous timed automaton with a single clock.

For the rest of this section, fix a timed automaton $\mathcal{A} = (X^{\mathcal{A}}, \mathcal{L}^{\mathcal{A}}, \ell^{\mathcal{A}}_{\text{in}}, \mathcal{L}^{\mathcal{A}}_{\text{acc}}, E^{\mathcal{A}})$ with $m$ clocks, and an unambiguous timed automaton $\mathcal{B} = (X^{\mathcal{B}}, \mathcal{L}^{\mathcal{B}}, \ell^{\mathcal{B}}_{\text{in}}, \mathcal{L}^{\mathcal{B}}_{\text{acc}}, E^{\mathcal{B}})$ with a single clock, both over the alphabet $\Sigma$. Without loss of generality, we assume $X^{\mathcal{A}} \cap X^{\mathcal{B}} = \emptyset$ and $\mathcal{L}^{\mathcal{A}} \cap \mathcal{L}^{\mathcal{B}} = \emptyset$.

The notions of *configurations of* $\mathcal{B}$, *synchronized configurations of* $\mathcal{A}$ *and* $\mathcal{B}$, and all related notions are adapted from Sections 3.1 and 3.2. We also have the following proposition.

**Proposition 11** *Let* $C, C'$ *be two configurations of* $\mathcal{B}$ *such that* $C \cap C' = \emptyset$ *and* $C \cup C'$ *is coverable, then for every timed word* $w$ *the following holds: if* $\mathrm{Succ}_{\mathcal{B}}(C, w)$ *is accepting, then* $\mathrm{Succ}_{\mathcal{B}}(C', w)$ *is non-accepting.*

[2]Note the structural similarity to the unambiguous register automaton in Fig. 1.

Next, we define an equivalence relation over tuples of $\mathbb{R}$, that plays the same role as the equivalence relation on tuples introduced in the previous sections. We start with some auxiliary definitions. Given $a \in \mathbb{R}$, we use $\lfloor a \rfloor$ to denote the integer part of $a$, and we use $\mathrm{frac}(a)$ to denote the fractional part of $a$, formally defined by $\mathrm{frac}(a) := a - \lfloor a \rfloor$. Given a $p$-tuple $\boldsymbol{a} = (a_1, \ldots, a_p) \in \mathbb{R}^p$, we use $-\boldsymbol{a}$ to denote the $p$-tuple $(-a_1, \ldots, -a_p) \in \mathbb{R}^p$.

Let $\boldsymbol{\alpha} \in \mathbb{N}^k$ be a $k$-tuple of non-negative integers. We define the equivalence relation $\sim_{\boldsymbol{\alpha}}$ induced by $\boldsymbol{\alpha}$ over $\mathbb{R}^k$ as follows: for every $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^k$, we have $\boldsymbol{a} \sim_{\boldsymbol{\alpha}} \boldsymbol{b}$ if, and only if, these two conditions hold:

(R1) for all $i \in \{1, \ldots, k\}$ and integers $\alpha \in \{-\alpha_i, \alpha_i + 1, \ldots, \alpha_i - 1, \alpha_i\}$ we have $a_i \leq \alpha$ if, and only if $b_i \leq \alpha$;

(R2) for all $i, j \in \{1, \ldots, k\}$ such that $|a_i| \leq \alpha_i$ and $|a_j| \leq \alpha_j$ we have $\mathrm{frac}(a_i) \leq \mathrm{frac}(a_j)$ if, and only if, $\mathrm{frac}(b_i) \leq \mathrm{frac}(b_j)$.

The equivalence classes of $\sim_{\boldsymbol{\alpha}}$ are called *clock regions*. Given $\boldsymbol{a}, \boldsymbol{a'} \in \mathbb{R}^k$, $\boldsymbol{\alpha} \in \mathbb{N}^k$, and two timed words $w = (\sigma_1, t_1) \ldots (\sigma_n, t_n)$ and $w' = (\sigma_1, t'_1) \ldots (\sigma_n, t'_n)$ such that $\mathrm{proj}(w) = \mathrm{proj}(w')$, we write $\boldsymbol{a}, w \sim_{\boldsymbol{\alpha}} \boldsymbol{a'}, w'$ if $\boldsymbol{a}, t_1, \ldots, t_n \sim_{\boldsymbol{\alpha}} \boldsymbol{a'}, t'_1, \ldots, t'_n$. For instance, we have $-0.3, w \sim_{(1,1,2)} -0.4, w'$ for $w = (\sigma, 0.7)(\sigma, 1.3)$ and $w' = (\sigma, 0.6)(\sigma, 1.3)$.

Recall the following classical result for clock regions.

**Proposition 12** ([1]) *If $\boldsymbol{a} \sim_{\boldsymbol{\alpha}} \boldsymbol{b}$, then for every $1 \leq i \leq k$ with $|a_i| \leq \alpha_i$ we have $a_i \models \phi$ if, and only if, $b_i \models \phi$, for every clock constraint $\phi$ that does not contain any constants greater than $\alpha_i$.*

The following proposition is easy to prove based on observations in [1].

**Proposition 13** *If $\boldsymbol{a} \sim_{\boldsymbol{\alpha}} \boldsymbol{b}$, then for every $t \in \mathbb{R}_{\geq 0}$ there exists $t' \in \mathbb{R}_{\geq 0}$ such that*

(L1) $\boldsymbol{a}, t \sim_{\boldsymbol{\alpha}, \lceil t \rceil} \boldsymbol{a}, t'$, *and*

(L2) $\boldsymbol{a}, t' \sim_{\boldsymbol{\alpha}, \lceil t \rceil} \boldsymbol{b}, t'$.

In the rest of this section, we follow the following convention when writing $\boldsymbol{a} \sim \boldsymbol{a'}$ (without subscript $\boldsymbol{\alpha}$): if $a_i$ and $a'_i$ are numbers coming from clock valuations, we take the corresponding $\alpha_i$ to be the largest constant appearing in a clock constraint of the automata under consideration. If $a_i$ and $a'_i$ are timestamps from a timed word, we take $\alpha_i$ to be larger than $a_i$ and $a'_i$.

The proofs of the following two propositions can be done analogously to the corresponding corollaries for register automata in Section 3.1.

**Proposition 14** *Let $\mathcal{A}$ be a timed automaton with $k$ registers. Let $\boldsymbol{a}$ and $\boldsymbol{a'}$ be two $k$-tuples over $\mathbb{R}_{\geq 0}$ for some $k \in \mathbb{N}$, and let $w$ and $w'$ be two timed words over $\Sigma$. If $-\boldsymbol{a}, w \sim -\boldsymbol{a'}, w'$, then* $\mathrm{Succ}_{\mathcal{A}}((\ell, \boldsymbol{a}), w)$ *is non-accepting if and only if* $\mathrm{Succ}_{\mathcal{A}}((\ell, \boldsymbol{a'}), w')$ *is non-accepting for every $\ell \in \mathcal{L}^{\mathcal{A}}$.*

**Proposition 15** *If $(\ell, \boldsymbol{a}), (\ell, \boldsymbol{a}')$ are two states of $\mathcal{B}$ such that $\boldsymbol{a} \neq \boldsymbol{a}'$ and $\{(\ell, \boldsymbol{a}), (\ell, \boldsymbol{a}')\}$ is coverable in $\mathcal{B}$, then for every timed word $w$ such that $\boldsymbol{a}, w \sim \boldsymbol{a}', w$, the configurations $\mathrm{Succ}_{\mathcal{B}}((\ell, \boldsymbol{a}), w)$ and $\mathrm{Succ}_{\mathcal{B}}((\ell, \boldsymbol{a}'), w)$ are non-accepting.*

## 6.3 Reducing the Size of the Configurations

We present the main technical result of this section, being the key ingredient for deciding the containment problem.

**Proposition 16** *Let $S = \langle(\ell^{\mathcal{A}}, \boldsymbol{d}), C\rangle$ be a coverable synchronized configuration of $\mathcal{A}$ and $\mathcal{B}$, where $\mathcal{B}$ uses only a single clock. Let $c_{\max}$ be the largest integer used in a clock constraint in $\mathcal{A}$ or $\mathcal{B}$. Let $0 < a < b < 1$ be two distinct reals such that:*

- $\boldsymbol{d}, a \sim \boldsymbol{d}, b$,
- $\{\ell \in \mathcal{L}^{\mathcal{B}} \mid (\ell, q + a) \in C\} = \{\ell \in \mathcal{L}^{\mathcal{B}} \mid (\ell, q + b) \in C\}$ *for every* $q \in \{0, \ldots, c_{\max} - 1\}$,
- $\{\ell \in \mathcal{L}^{\mathcal{B}} \mid (\ell, q + a) \in C, q \geq c_{\max}\} = \{\ell \in \mathcal{L}^{\mathcal{B}} \mid (\ell, q + b) \in C, q \geq c_{\max}\}$.

*Let $C^- = (\mathcal{L}^{\mathcal{B}} \times \{q + a, q + b \mid q \geq 0\}) \cap C$. Then $\langle(\ell^{\mathcal{A}}, \boldsymbol{d}), C \setminus C^-\rangle$ reaches a bad synchronized configuration if, and only if, $S$ reaches a bad synchronized configuration.*

*Proof* The "if" direction follows from the simple observation that for every timed 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 "only if" direction, suppose that there exists a timed word $w$ such that there exists an accepting run of $\mathcal{A}$ on $w$ that starts in $(\ell^{\mathcal{A}}, \boldsymbol{d})$, and $\mathrm{Succ}_{\mathcal{B}}(C \setminus C^-, w)$ is non-accepting. We assume in the following that $\mathrm{Succ}_{\mathcal{B}}(C^-, w)$ is accepting; otherwise we are done. So let $(\ell^{\mathcal{B}}, c) \in C^-$ be the (unique) state such that $\mathrm{Succ}_{\mathcal{B}}(C^-, w)$ is accepting. We present the proof for the case that $\mathrm{frac}(c) = b$; the proof for $\mathrm{frac}(c) = a$ is analogous.

Assume $w = (\sigma_1, t_1) \ldots (\sigma_k, t_k)$. Similarly to the proof of Proposition 10, we will first prove that we can assume without loss of generality that for all $1 \leq i \leq k$, if $b + t_i \notin \mathbb{N}$, then $-a, t_i \sim -b, t_i$. Due to time passing and potential resets of the clocks to zero, the proof of this is, however, technically a bit more involved than in the proof of Proposition 10. From Lemma 2 below it follows that one can construct from $w$ a timed word $w' = (\sigma_1, t_1') \ldots (\sigma_k, t_k')$ such that $-\boldsymbol{d}, -b, w \sim -\boldsymbol{d}, -b, w'$ and $-a, t_i' \sim -b, t_i'$ for all $1 \leq i \leq k$ such that $b + t_i \notin \mathbb{N}$.

By (C1) of Lemma 2, we have $-\boldsymbol{d}, -b, w \sim -\boldsymbol{d}, -b, w'$. By assumption and Proposition 14, there exists an accepting run of $\mathcal{A}$ on $w'$ that starts in $(\ell^{\mathcal{A}}, \boldsymbol{d})$, and similarly, $\mathrm{Succ}((\ell^{\mathcal{B}}, b), w')$ is still accepting. By Proposition 11, $\mathrm{Succ}(C \setminus \{(\ell^{\mathcal{B}}, b)\}, w')$ is still non-accepting.

In the next step, the goal is to construct from $w'$ a timed word $w''$ such that $-a, w'' \sim -b, w''$ and $-e, w' \sim -e, w''$ for all $e \in \mathrm{data}(\boldsymbol{d}) \cup \mathrm{data}(C \setminus C^-)$. Let

$\epsilon > 0$ be a small number to be determined later. Define $w'' = (\sigma_1, t''_1) \dots (\sigma_k, t''_k)$ by

$$t''_i := \begin{cases} t'_i - \varepsilon & \text{if } t_i + b \in \mathbb{N} \\ t'_i & \text{otherwise.} \end{cases}$$

We prove $-a, w'' \sim -b, w''$: Note that by (C2) of Lemma 2 and the definition of $w''$ we already have $-a, \boldsymbol{s} \sim -b, \boldsymbol{s}$, where $\boldsymbol{s}$ is the tuple containing all timestamps $t''_i \in \{t''_1, \dots, t''_k\}$ such that $t_i + b \notin \mathbb{N}$. But that $-a, t''_i \sim -b, t''_i$ for all $1 \le i \le k$ such that $b + t_i \in \mathbb{N}$ follows by definition of $t''_i$. Hence $-a, w'' \sim -b, w''$.

Next, we prove $-e, w' \sim -e, w''$ for every $e \in \text{data}(\boldsymbol{d}) \cup \text{data}(C \backslash C^-)$. Let us start by proving $w' \sim w''$. Let $1 \le i \le k$ be such that $t''_i \ne t'_i$. Recall that $t''_i = \lfloor t'_i \rfloor + \text{frac}(t'_i) - \varepsilon$. Hence $\lfloor t'_i \rfloor = \lfloor t''_i \rfloor$ if $\epsilon < \text{frac}(t'_i)$, which proves (R1). For proving (R2), we consider two cases. First, let $1 \le j \le k$ be such that $t''_j \ne t'_j$, i.e., $t'_j + b \in \mathbb{N}$ and thus $\text{frac}(t'_j) = \text{frac}(t'_i)$. Clearly, $\text{frac}(t'_i) \le \text{frac}(t'_j)$ iff $\text{frac}(t''_i) \le \text{frac}(t''_j)$. Second, let $1 \le j \le k$ be such that $t''_j = t'_j$. If $\text{frac}(t'_i) \le \text{frac}(t'_j)$, then clearly $\text{frac}(t''_i) \le \text{frac}(t''_j)$, as by definition of $t''_i$ we have $\text{frac}(t''_i) < \text{frac}(t'_i)$. So assume $\text{frac}(t'_j) < \text{frac}(t'_i)$. Then if $\epsilon < \text{frac}(t'_i) - \text{frac}(t'_j)$, we have $\text{frac}(t''_j) = \text{frac}(t'_j) < \text{frac}(t'_i) - \epsilon = \text{frac}(t''_i)$. This finishes the proof of (R2), and we have $w' \sim w''$. Now, let $e \in \text{data}(\boldsymbol{d}) \cup \text{data}(C \backslash C^-)$. So let $i \in \{1, \dots, k\}$ be such that $t''_i \ne t'_i$. Assume $\text{frac}(t'_i) \le \text{frac}(-e)$. As above, we immediately obtain $\text{frac}(t''_i) \le \text{frac}(-e)$ as, by definition, $\text{frac}(t''_i) < \text{frac}(t'_i)$. So assume $\text{frac}(t'_i) > \text{frac}(-e)$. Then if $\epsilon < \text{frac}(t'_i) - \text{frac}(-e)$, we have that $\text{frac}(t''_i) = \text{frac}(t'_i) - \epsilon > \text{frac}(-e)$. This finishes the proof of (R2) for $-e$ and $t''_i$, and altogether we have $-e, w' \sim -e, w''$ for every $e \in \text{data}(\boldsymbol{d}) \cup \text{data}(C \backslash C^-)$. Overall, we see that it suffices to choose $\epsilon$ smaller than every $\text{frac}(t'_i)$, every $\text{frac}(t'_i) - \text{frac}(t'_j)$ with $\text{frac}(t'_i) > \text{frac}(t'_j)$, and every $\text{frac}(t'_i) - \text{frac}(-e)$ with $e \in \text{data}(\boldsymbol{d}) \cup \text{data}(C \backslash C^-)$ and $\text{frac}(t'_i) > \text{frac}(-e)$.

Now we can follow again the lines of argumentation in the proof of Proposition 10. We prove that $\text{Succ}_{\mathcal{B}}(C, w'')$ is non-accepting. For this let $(\ell, e) \in C$. We distinguish the following cases.

- $e = a$. By assumption, $(\ell, b) \in C$. We proved above that $-a, w'' \sim -b, w''$. By Proposition 15, $\text{Succ}_{\mathcal{B}}((\ell, e), w'')$ must thus be non-accepting.
- $e = b$. The proof is symmetric to the previous case.
- $e \notin \{a, b\}$. We proved above that $-e, w' \sim -e, w''$. By assumption, $\text{Succ}_{\mathcal{B}}((\ell, e), w')$ is non-accepting. This and Proposition 14 imply that $\text{Succ}_{\mathcal{B}}((\ell, e), w'')$ is non-accepting, too.

This finishes the proof for the "only if"-direction. □

We finally prove here Lemma 2, the technical core of the proof of Proposition 16.

**Lemma 2** *With the same assumptions as in Proposition 16, let* $\boldsymbol{r}$ *be the tuple containing all timestamps* $t_i \in \{t_1, \dots, t_k\}$ *satisfying* $-a, t_i \sim -b, t_i$. *Then for all* $0 \le i \le k$, *there exist* $t'_1, \dots, t'_i \in \mathbb{R}_{\ge 0}$ *such that*

(C1) $-\boldsymbol{d}, -b, \boldsymbol{r}, t_1, \dots, t_i \sim -\boldsymbol{d}, -b, \boldsymbol{r}, t'_1, \dots, t'_i,$

(C2) $-\boldsymbol{d}, -b, \boldsymbol{r}, \boldsymbol{s^i} \sim -\boldsymbol{d}, -a, \boldsymbol{r}, \boldsymbol{s^i}$, *where* $\boldsymbol{s^i}$ *is the tuple containing all timestamps* $t'_j \in \{t'_1, \ldots, t'_i\}$ *satisfying* $b + t_j \notin \mathbb{N}$.

*Proof* The proof goes by induction on $i$. For the induction base $i = 0$, note that the assumption $\boldsymbol{d}, a \sim \boldsymbol{d}, b$ implies $-\boldsymbol{d}, -a \sim -\boldsymbol{d}, -b$. This and the definition of $\boldsymbol{r}$ implies the claim. For the induction step, suppose that the induction hypothesis holds for $0 \leq i < k$. We prove the claim for $i + 1$. We distinguish the following four cases:

– There exists $1 \leq p \leq m$ such that $d_p + t_{i+1} \in \mathbb{N}$. Note that this implies

$$\text{frac}(-d_p) = \text{frac}(t_{i+1}) \tag{2}$$

Our goal is to set $t'_{i+1}$ to be equal to $t_{i+1}$; for this we first prove $t'_i \leq t_{i+1}$. Towards contradiction, suppose $t'_i > t_{i+1}$. We distinguish two cases:

1. $\lfloor t'_i \rfloor > \lfloor t_{i+1} \rfloor$. By induction hypothesis, $\lfloor t'_i \rfloor = \lfloor t_i \rfloor$, but this contradicts $t_i \leq t_{i+1}$.
2. $\lfloor t'_i \rfloor = \lfloor t_{i+1} \rfloor$ and $\text{frac}(t'_i) > \text{frac}(t_{i+1})$. By (2), $\text{frac}(t'_i) > \text{frac}(-d_p)$. By induction hypothesis, $\text{frac}(t_i) > \text{frac}(-d_p)$ and $\lfloor t'_i \rfloor = \lfloor t_i \rfloor$. By (2), $\text{frac}(t_i) > \text{frac}(t_{i+1})$, contradicting again $t_i \leq t_{i+1}$.

Hence $t'_i \leq t_{i+1}$ and we set $t'_{i+1} = t_{i+1}$. Next we prove that (C1) holds. That (R1) holds for $t_{i+1}$ and $t'_{i+1}$ is clear. For the proof of (R2), note that for all $1 \leq q \leq i$, we have $\text{frac}(-d_p) \leq \text{frac}(t_q)$ iff $\text{frac}(-d_p) \leq \text{frac}(t'_q)$ by induction hypothesis. But then (2) gives $\text{frac}(t_{i+1}) \leq \text{frac}(t_q)$ iff $\text{frac}(t_{i+1}) \leq \text{frac}(t'_q)$, so that indeed (R2) holds. Hence $-\boldsymbol{d}, -b, \boldsymbol{r}, t_1, \ldots, t_i, t_{i+1} \sim -\boldsymbol{d}, -b, \boldsymbol{r}, t'_1, \ldots, t'_i, t'_{i+1}$, which finishes the proof of (C1). For the proof of (C2), note that by assumption we have $\text{frac}(a) \leq \text{frac}(-d_p)$ iff $\text{frac}(b) \leq \text{frac}(-d_p)$. This and (2) yields $\text{frac}(a) \leq \text{frac}(t'_{i+1})$ iff $\text{frac}(b) \leq \text{frac}(t'_{i+1})$.

– $d_p + t_{i+1} \notin \mathbb{N}$ for all $1 \leq p \leq m$ and $b + t_{i+1} \in \mathbb{N}$. Set $t'_{i+1} = t_{i+1}$. The proof for $t'_i \leq t'_{i+1}$ and (C1) is similar to the proof of the previous case, using the fact that $b + t_{i+1} \in \mathbb{N}$ implies $\text{frac}(-b) = \text{frac}(t_{i+1})$ (Condition (C2) follows easily by induction hypothesis and $t'_{i+1} \notin \text{data}\,(\boldsymbol{n}^i)$).

– $d_p + t_{i+1} \notin \mathbb{N}$ for all $1 \leq p \leq m$ and $b + t_{i+1} \notin \mathbb{N}$ and $-a, t_{i+1} \sim -b, t_{i+1}$. Note that $t_{i+1} \in \text{data}(\boldsymbol{r})$. The claim holds by induction hypothesis for $t'_{i+1} = t_{i+1}$.

– $d_p + t_{i+1} \notin \mathbb{N}$ for all $1 \leq p \leq m$ and $b + t_{i+1} \notin \mathbb{N}$ and $-a, t_{i+1} \not\sim -b, t_{i+1}$. By induction hypothesis, we have

$$-\boldsymbol{d}, -b, \boldsymbol{r}, t_1, \ldots, t_i \sim -\boldsymbol{d}, -b, \boldsymbol{r}, t'_1, \ldots, t'_i. \tag{3}$$

We apply Proposition 13 to (3) and $t_{i+1}$ to obtain some $t''_{i+1} \in \mathbb{R}_{\geq 0}$ satisfying

(L1″) $-\boldsymbol{d}, -b, \boldsymbol{r}, t_1, \ldots, t_i, t_{i+1} \sim -\boldsymbol{d}, -b, \boldsymbol{r}, t_1, \ldots, t_i, t''_{i+1}$,
(L2″) $-\boldsymbol{d}, -b, \boldsymbol{r}, t_1, \ldots, t_i, t''_{i+1} \sim -\boldsymbol{d}, -b, \boldsymbol{r}, t'_1, \ldots, t'_i, t''_{i+1}$.

By induction hypothesis, we further have

$$-\boldsymbol{d}, -b, \boldsymbol{r}, \boldsymbol{s^i} \sim -\boldsymbol{d}, -a, \boldsymbol{r}, \boldsymbol{s^i}. \tag{4}$$

We apply Proposition 13 to (4) and $t''_{i+1}$ to obtain some $t'_{i+1} \in \mathbb{R}_{\geq 0}$ satisfying

(L1 ′) $-\boldsymbol{d}, -b, \boldsymbol{r}, \boldsymbol{s^i}, t''_{i+1} \sim -\boldsymbol{d}, -b, \boldsymbol{r}, \boldsymbol{s^i}, t'_{i+1}$,
(L2 ′) $-\boldsymbol{d}, -b, \boldsymbol{r}, \boldsymbol{s^i}, t'_{i+1} \sim -\boldsymbol{d}, -a, \boldsymbol{r}, \boldsymbol{s^i}, t'_{i+1}$.

Note that $t'_{i+1} \geq t'_i$, and (C2) is already equal to (L2 ′). For the proof of (C1), recall that for all $1 \leq p \leq i$, we have $t'_p \notin \text{data}(\boldsymbol{s^i})$ if $b+t_p \in \mathbb{N}$, or, equivalently, if $\text{frac}(-b) = \text{frac}(t_p)$. Recall from above that in that case we have $t'_p = t_p$. This and (L1 ′) imply

$$-\boldsymbol{d}, -b, \boldsymbol{r}, t'_1, \ldots, t'_i, t''_{i+1} \sim -\boldsymbol{d}, -b, \boldsymbol{r}, t'_1, \ldots, t'_i, t'_{i+1}.$$

This and (L2″) imply

$$-\boldsymbol{d}, -b, \boldsymbol{r}, t_1, \ldots, t_i, t_{i+1} \sim -\boldsymbol{d}, -b, \boldsymbol{r}, t'_1, \ldots, t'_i, t'_{i+1},$$

which finishes the proof of (C1), and therewith also the proof of the lemma. □

**Theorem 3** *The containment problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$, where $\mathcal{A}$ is a non-deterministic timed automaton and $\mathcal{B}$ is an unambiguous timed automaton with a single clock, is in EXPSPACE.*

*Proof* As in the previous sections, the previous proposition allows us to bound the number of different fractional values appearing in a maximally reduced synchronized configuration $S$. Let $S = \langle(\ell^{\mathcal{A}}, \boldsymbol{d}), C\rangle$ be a maximally reduced configuration. Given a number $0 < a < 1$ and an integer $p \in \{0, \ldots, c_{\max}-1\}$, define $\mathcal{L}_a(p) := \{\ell \in \mathcal{L}^{\mathcal{B}} \mid (\ell, p+a) \in C\}$. Let $\mathcal{L}_a(\infty) = \{\ell \in \mathcal{L}^{\mathcal{B}} \mid (\ell, q+a) \in C, q \geq c_{\max}\}$. If $\boldsymbol{d}, a \sim \boldsymbol{d}, b$, $\mathcal{L}_a(\infty) = \mathcal{L}_b(\infty)$, and $\mathcal{L}_a(p) = \mathcal{L}_b(p)$ for all $p \in \{0, \ldots, c_{\max}-1\}$, then one can reduce $S$ by the previous proposition. Thus, one gets that $S$ contains at most $(k+1)2^{\mathcal{L}^{\mathcal{B}}(c_{\max}+1)}$ positive fractional values, where $k$ is the number of clocks of $\mathcal{A}$. Again, as in the previous section we obtain that the number of states in $S$ is bounded by a double exponential in the size of $\mathcal{B}$. Therefore, by a similar reachability algorithm as above, this time on abstract timed configurations (for example, as defined in [17]), we obtain an exponential-space decision algorithm. □

## 7 Open Problems

An obvious open problem is the decidability status of the containment problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$, where $\mathcal{B}$ is an unambiguous register automaton over $(\mathbb{Q}; <)$ with more than one register. In Section 5, we presented an example that shows that our approach of reducing the configurations fails in this case.

Likewise, we leave open the *exact* computational complexity of the containment problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$, when $\mathcal{B}$ is an unambiguous register automaton (or an unambiguous timed automaton). Finding lower bounds for unambiguous automata is a hard problem. Techniques for proving lower complexity bounds of the

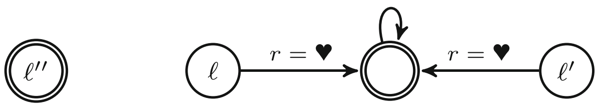

**Fig. 5** A part of a register automaton with guessing over $(\mathbb{N}; =)$ and a singleton alphabet (we omit the labels at the edges) and a single register $r$

containment problem (respectively the universality problem) for the case where $\mathcal{B}$ 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 where $\mathcal{B}$ is unambiguous, even for the class of finite automata. Concerning the upper bound, computer experiments revealed that maximally reduced 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. For instance, 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 each input data word $w$ is bounded by $p(|w|)$ for some polynomial $p$. It seems reasonable that some 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. The crucial difference between register automata as introduced here and register automata with guessing is that a configuration of the latter may contain infinitely many states, albeit with *finite support*. It is easy to construct a register automaton with guessing over $(\mathbb{N}; =)$ and a single register that generates a configuration of the form $C = \{(\ell, 0), (\ell, 1)\} \cup \{(\ell', d) \mid d \in \mathbb{N}\backslash\{0, 1\}\} \cup \{(\ell'', 2)\}$ with finite support $\{0, 1, 2\}$. Note that the two register valuations 0 and 1 are *indistinguishable* in the sense of Section 3.3. Consider the automaton depicted in Fig. 5. It is easy to see that $\text{Succ}(C, w)$ is accepting for every data word $w$: for data words $w$ with $|w| \neq 1$ this is clear. If $w = (\sigma, d)$, we distinguish the following two cases. If $d \in \{0, 1\}$, then $\text{Succ}((\ell, d), w)$ is accepting. If $d \notin \{0, 1\}$, then $\text{Succ}((\ell', d), w)$ is accepting. However, $\text{Succ}(C\backslash\{(\ell, 0), (\ell, 1)\}, w)$ is non-accepting for $w = (\sigma, 0)$ and $w = (\sigma, 1)$. This shows that removing states from a configuration may be harmful. In [15], we recently proved decidability of the containment problem for the case that $\mathcal{B}$ is an unambiguous register automaton with guessing and at most one register, by establishing a variant of Proposition 6 in which we do reduce the size of the *support* of a configuration by not only *removing* but also *adding* states to a configuration. For the case that $\mathcal{B}$ uses more than one register, however, the respective containment problem remains open for future research.

## References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994). https://doi.org/10.1016/0304-3975(94)90010-8
2. Colcombet, T.: Forms of determinism for automata (invited talk). In: Dürr, C., Wilke, T. (eds.) 29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France, Vol 14 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 1–23 (2012). https://doi.org/10.4230/LIPIcs.STACS.2012.1
3. Colcombet, T.: Unambiguity in automata theory. In: Shallit, J., Okhotin, A. (eds.) Descriptional Complexity of Formal Systems - 17th International Workshop, DCFS 2015, Waterloo, ON, Canada, June 25-27, 2015. Proceedings, vol 9118 of Lecture Notes in Computer Science, pp. 3–18. Springer, New York (2015). https://doi.org/10.1007/978-3-319-19225-3_1
4. Daviaud, L., Jurdzinski, M., Lazic, R., Mazowiecki, F., Pérez, G.A., Worrell, J.: When is containment decidable for probabilistic automata? In: 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic, pp. 121:1–121:14 (2018). https://doi.org/10.4230/LIPIcs.ICALP.2018.121
5. Demri, S., Lazic, R.: LTL with the freeze quantifier and register automata. ACM Trans. Comput. Log. 10(3) https://doi.org/10.1145/1507244.1507246 (2009)
6. Figueira, D.: Alternating register automata on finite words and trees. Log. Meth. Comput. Sci. 8 (1) https://doi.org/10.2168/LMCS-8(1:22)2012 (2012)
7. Figueira, D., Figueira, S., Schmitz, S., Schnoebelen, P.: 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, IEEE Computer Society, pp. 269–278 (2011). https://doi.org/10.1109/LICS.2011.39
8. Figueira, D., Hofman, P., Lasota, S.: Relating timed and register automata. Math. Struct. Comput. Sci. **26**(6), 993–1021 (2016). https://doi.org/10.1017/S0960129514000322
9. Fijalkow, N., Riveros, C., Worrell, J.: Probabilistic automata of bounded ambiguity. In: Meyer, R., Nestmann, U. (eds.) 28th International Conference on Concurrency Theory, CONCUR 2017, September 5-8, 2017, Berlin, Germany, vol 85 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 19:1–19:14 (2017). https://doi.org/10.4230/LIPIcs.CONCUR.2017.19
10. Kaminski, M., Francez, N.: Finite-memory automata. Theor. Comput. Sci. **134**(2), 329–363 (1994). https://doi.org/10.1016/0304-3975(94)90242-9
11. Kaminski, M., Zeitlin, D.: Finite-memory automata with non-deterministic reassignment. Int. J. Found. Comput. Sci. 21(05) (2010)
12. Leung, H.: Descriptional complexity of NFA of different ambiguity. Int. J. Found. Comput. Sci. **16**(5), 975–984 (2005). https://doi.org/10.1142/S0129054105003418
13. Skrzypczak, M.: Unambiguous languages exhaust the index hierarchy. In: 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic, pp. 140:1–140:14 (2018). https://doi.org/10.4230/LIPIcs.ICALP.2018.140
14. Mottet, A., Quaas, K.: The containment problem for unambiguous register automata. In: Niedermeier, R., Paul, C. (eds.) 36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany, vol 126 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 53:1–53:15 (2019). https://doi.org/10.4230/LIPIcs.STACS.2019.53
15. Mottet, A., Quaas, K.: On the containment problem for unambiguous single-register automata with guessing. CoRR, abs/1905.12445, arXiv:1905.12445 (2019)
16. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. ACM Trans. Comput. Log. **5**(3), 403–435 (2004). https://doi.org/10.1145/1013560.1013562
17. Ouaknine, J., Worrell, J.: 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, IEEE Computer Society, pp. 54–63 (2004). https://doi.org/10.1109/LICS.2004.1319600
18. Raskin, M.: A superpolynomial lower bound for the size of non-deterministic complement of an unambiguous automaton. In: 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic, pp. 138:1–138:11 (2018). https://doi.org/10.4230/LIPIcs.ICALP.2018.138
19. Sakamoto, H., Ikeda, D.: Intractability of decision problems for finite-memory automata. Theor. Comput. Sci. **231**(2), 297–308 (2000). https://doi.org/10.1016/S0304-3975(99)00105-X

20. Schmitz, S., Schnoebelen, P.: Multiply-recursive upper bounds with higman's lemma. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) Automata, languages and programming - 38th international colloquium, ICALP 2011, zurich, switzerland, july 4-8, 2011, proceedings, part II, vol 6756 of Lecture Notes in Computer Science, pp. 441–452. Springer, New York (2011). https://doi.org/10.1007/978-3-642-22012-8_35
21. Segoufin, L.: Automata and logics for words and trees over an infinite alphabet. In: Ésik, Z. (ed.) Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings, vol 4207 of Lecture Notes in Computer Science, pp. 41–57. Springer, New York (2006). https://doi.org/10.1007/11874683_3